PSPP Users Guide

GNU PSPP Statistical Analysis Software
Release 0.6.2

This manual is for GNU PSPP version 0.6.2, software for statistical analysis.

Copyright © 1997, 1998, 2004, 2005 Free Software Foundation, Inc.

Table of Contents

1 Introduction

PSPP is a tool for statistical analysis of sampled data. It reads a syntax file and a data file, analyzes the data, and writes the results to a listing file or to standard output.

The language accepted by PSPP is similar to those accepted by SPSS statistical products. The details of PSPP's language are given later in this manual.

PSPP produces output in two forms: tables and charts. Both of these can be written in several formats; currently, ASCII, PostScript, and HTML are supported. In the future, more drivers, such as PCL and X Window System drivers, may be developed. For now, Ghostscript, available from the Free Software Foundation, may be used to convert PostScript chart output to other formats.

The current version of PSPP, 0.6.2, is woefully incomplete in terms of its statistical procedure support. PSPP is a work in progress. The author hopes to fully support all features in the products that PSPP replaces, eventually. The author welcomes questions, comments, donations, and code submissions. See Chapter 16 [Submitting Bug Reports], page 207, for instructions on contacting the author.

2 Your rights and obligations

PSPP is not in the public domain. It is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of this program that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of PSPP, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of PSPP, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for PSPP. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise conditions of the license for PSPP are found in the GNU General Public License. You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA. This manual specifically is covered by the GNU Free Documentation License (see Appendix B [GNU Free Documentation License], page 230).

3 Invoking PSPP

```
pspp [ -B dir | --config-dir=dir ] [ -o device | --device=device ]
       [ -d var[=value] | --define=var[=value] ] [-u var | --undef=var ]
       [ -f file | --out-file=file ] [ -p | --pipe ] [ -I- | --no-include ]
       [ -I dir | --include=dir ] [ -i | --interactive ]
       [ -n | --edit | --dry-run | --just-print | --recon ]
       [ -r | --no-statrc ] [ -h | --help ] [ -l | --list ]
       [ -c command | --command command ] [ -s | --safer ]
       [ --testing-mode ] [ -V | --version ] [ -v | --verbose ]
       [ key=value ] file...
```

3.1 Non-option Arguments

Syntax files and output device substitutions can be specified on PSPP's command line:

file

>A file by itself on the command line will be executed as a syntax file. If multiple
>files are specified, they are executed in order, as if their contents had been given
>in a single file. PSPP terminates after the syntax files run, unless the `-i` or
>`--interactive` option is given (see Section 3.4 [Language control options],
>page 4).

key=value

>Defines an output device macro *key* to expand to *value*, overriding any macro
>having the same *key* defined in the device configuration file. See Section A.5.2
>[Macro definitions], page 221.

There is one other way to specify a syntax file, if your operating system supports it. If
you have a syntax file '`foobar.stat`', put the notation

```
    #! /usr/local/bin/pspp
```

at the top, and mark the file as executable with `chmod +x foobar.stat`. (If PSPP is not
installed in '`/usr/local/bin`', then insert its actual installation directory into the syntax
file instead.) Now you should be able to invoke the syntax file just by typing its name. You
can include any options on the command line as usual. PSPP entirely ignores any lines
beginning with '`#!`'.

3.2 Configuration Options

Configuration options are used to change PSPP's configuration for the current run. The
configuration options are:

`-a {compatible|enhanced}`
`--algorithm={compatible|enhanced}`

>If you chose `compatible`, then PSPP will use the same algorithms as used
>by some proprietary statistical analysis packages. This is not recommended, as
>these algorithms are inferior and in some cases compeletely broken. The default
>setting is `enhanced`. Certain commands have subcommands which allow you
>to override this setting on a per command basis.

`-B dir`

`--config-dir=dir`

>   Sets the configuration directory to *dir*. See Section A.1 [File locations],
>   page 219.

`-o device`

`--device=device`

>   Selects the output device with name *device*. If this option is given more than
>   once, then all devices mentioned are selected. This option disables all devices
>   besides those mentioned on the command line.

3.3 Input and output options

Input and output options affect how PSPP reads input and writes output. These are the
input and output options:

`-f file`

`--out-file=file`

>   This overrides the output file name for devices designated as listing devices. If
>   a file named *file* already exists, it is overwritten.

`-p`

`--pipe`

>   Allows PSPP to be used as a filter by causing the syntax file to be read from
>   stdin and output to be written to stdout. Conflicts with the `-f file` and `--`
>   `file=file` options.

`-I-`

`--no-include`

>   Clears all directories from the include path. This includes all directories put
>   in the include path by default. See Section A.9 [Miscellaneous configuring],
>   page 228.

`-I dir`

`--include=dir`

>   Appends directory *dir* to the path that is searched for include files in PSPP
>   syntax files.

`-c command`

`--command=command`

>   Execute literal command *command*. The command is executed before startup
>   syntax files, if any.

`--testing-mode`

>   Invoke heuristics to assist with testing PSPP. For use by `make check` and similar
>   scripts.

3.4 Language control options

Language control options control how PSPP syntax files are parsed and interpreted. The
available language control options are:

`-i`
`--interactive`

> When a syntax file is specified on the command line, PSPP normally terminates after processing it. Giving this option will cause PSPP to bring up a command prompt after processing the syntax file.
>
> In addition, this forces syntax files to be interpreted in interactive mode, rather than the default batch mode. See Section A.5.6 [Tokenizing lines], page 224, for information on the differences between batch mode and interactive mode command interpretation.

`-n`
`--edit`
`--dry-run`
`--just-print`
`--recon`

> Only the syntax of any syntax file specified or of commands entered at the command line is checked. Transformations are not performed and procedures are not executed. Not yet implemented.

`-r`
`--no-statrc`

> Prevents the execution of the PSPP startup syntax file.

`-s`
`--safer`

> Disables certain unsafe operations. This includes the ERASE and HOST commands, as well as use of pipes as input and output files.

3.5 Informational options

Informational options cause information about PSPP to be written to the terminal. Here are the available options:

`-h`

`--help`

> Prints a message describing PSPP command-line syntax and the available device driver classes, then terminates.

`-l`

`--list`

> Lists the available device driver classes, then terminates.

`-x {compatible|enhanced}`
`--syntax={compatible|enhanced}`

> If you chose `compatible`, then PSPP will only accept command syntax that is compatible with the proprietary program SPSS. If you choose `enhanced` then additional syntax will be available. The default is `enhanced`.

`-V`

`--version`

Prints a brief message listing PSPP's version, warranties you don't have, copying conditions and copyright, and e-mail address for bug reports, then terminates.

`-v`

`--verbose`

Increments PSPP's verbosity level. Higher verbosity levels cause PSPP to display greater amounts of information about what it is doing. Often useful for debugging PSPP's configuration.

This option can be given multiple times to set the verbosity level to that value. The default verbosity level is 0, in which no informational messages will be displayed.

Higher verbosity levels cause messages to be displayed when the corresponding events take place.

1

Driver and subsystem initializations.

2

Completion of driver initializations. Beginning of driver closings.

3

Completion of driver closings.

4

Files searched for; success of searches.

5

Individual directories included in file searches.

Each verbosity level also includes messages from lower verbosity levels.

4 Invoking Psppire

This chapter describes the graphical user interface.

4.1 Primary Windows

The PSPP graphical user interface provides a set of windows through which you store, edit and analyze data. You can choose to perform tasks and operations through interactive drop down menus in the data editor window or through command line statements in the syntax editor. The three primary windows include the Data Editor, Syntax Editor, and the Output Viewer.

Data Editor Central to PSPP, launches when you start a session. Similar in organization to a spreadsheet with rows and columns of data cells, this is where the data is stored, operated on, and edited.
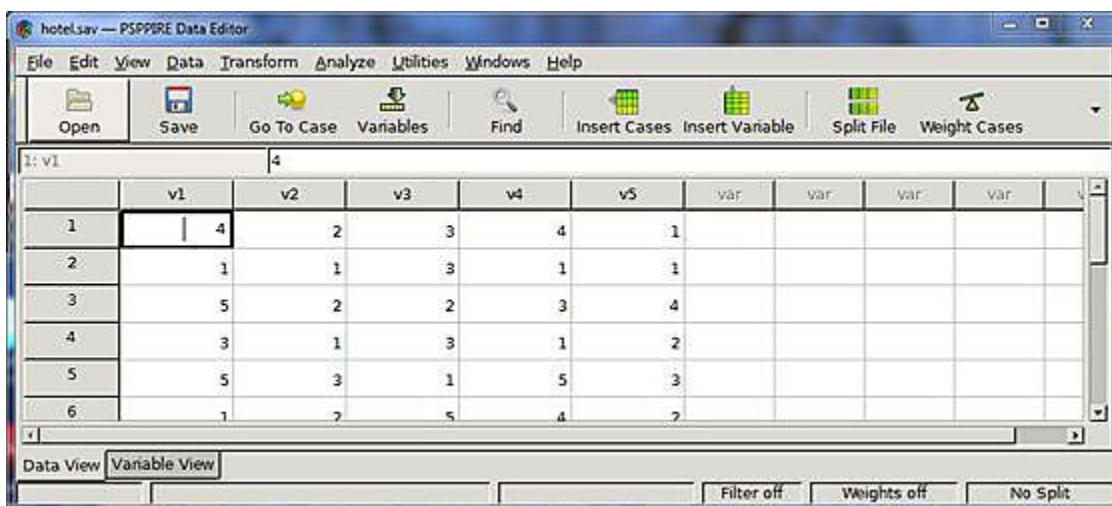
Figure 4.1: Data editor window

Output Viewer Displays executed instructions as well as results of statistical analyses including, text, tables, and charts.

Note: this window is not visible upon start up and only pops open after the first instruction set is executed.

Syntax Editor Enter command line instructions to edit, import and analyze data. This has the advantage of enabling batch processing and code reuse.

Launch the syntax editor from within the data editor by selecting File → Open → Syntax.

Figure 4.2: Syntax editor window

4.2 Data Editor

There are two alternate views: *data view* and *variable view*. Data view shows the numerical value held by each datum while the other displays the metadictionary data of each variable. Toggle between views by clicking on the *Data View* or *Variable View* tab at the bottom left hand corner of the window.
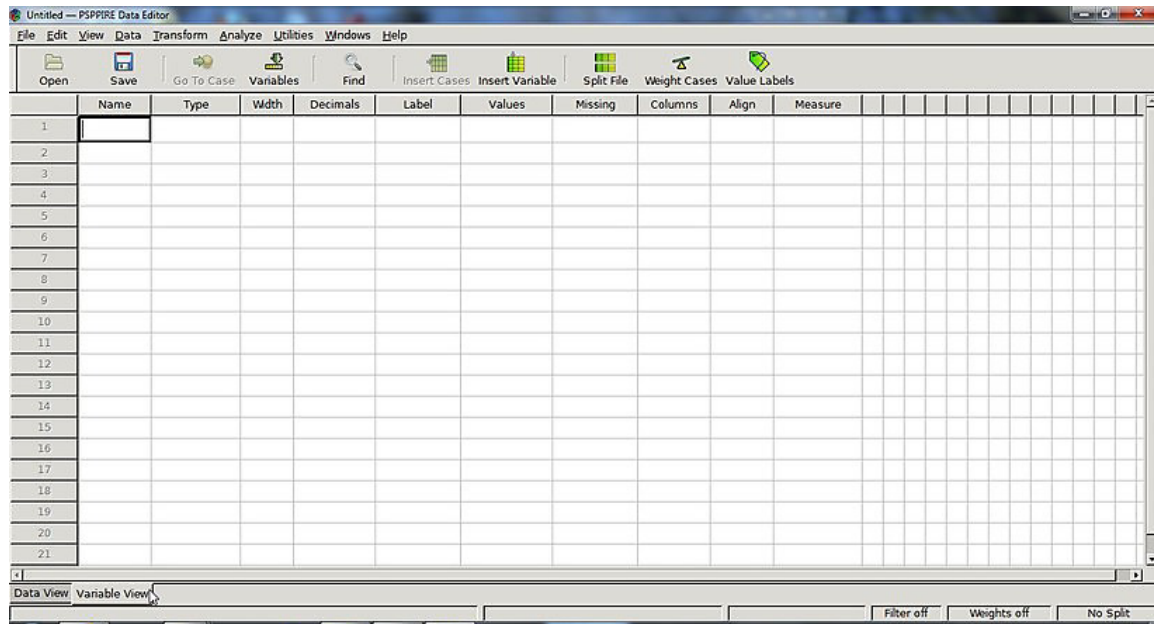
Figure 4.3: Data editor window

4.2.1 Data View
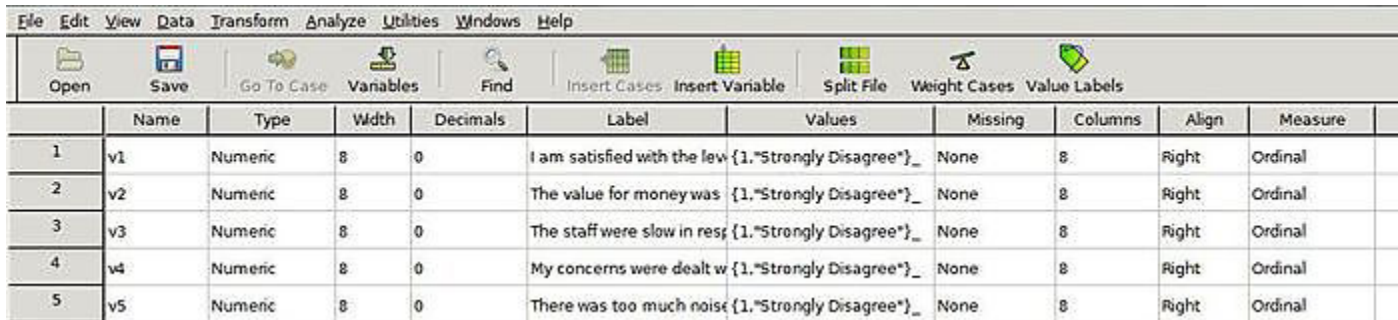
PSPP lists variables in columns and cases in rows, e.g., in the hotel customer satisfaction survey (See Figure 4.4) each row represents the set of answers given by each respondent and each column a question.

Figure 4.4: Data view

4.2.2 Variable View

Display variable attributes associated with each case, where variable attributes (See Chapter 5 [Attributes], page 92) include name, type, width, decimal, variable label, value label, missing values, columns, align, and measurement.

Figure 4.5: Variable view

4.2.3 Data Entry and Editing

There are several functions available in the data editor for entering, editing and sorting data.

4.2.3.1 Edit Variable Attributes
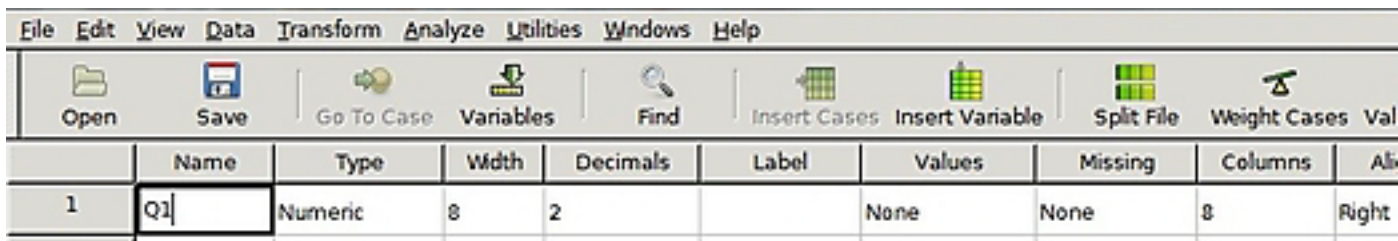
Edit variable attributes such as Name, Type, Width, Decimals, Label, and Values, Missing, Columns, Align, and Measure in variable view.

Name

Click a cell in the *Name* column.
Type a new name in the cell.
Press the enter key.

Figure 4.6: Edit *name* in variable view.

Type

Click a cell in the *Type* column.

Click on the grey box that appears to bring up a *Variable Type* dialogue box.

Choose a variable type (See Chapter 5 [Basic Numeric Formats], page 92) from the list:
Numeric, Comma, Dot, Scientific Notation, Date, Dollar, Custom Currency, and String.

Enter the width and/or number of decimal places, if relevant.

Press OK.
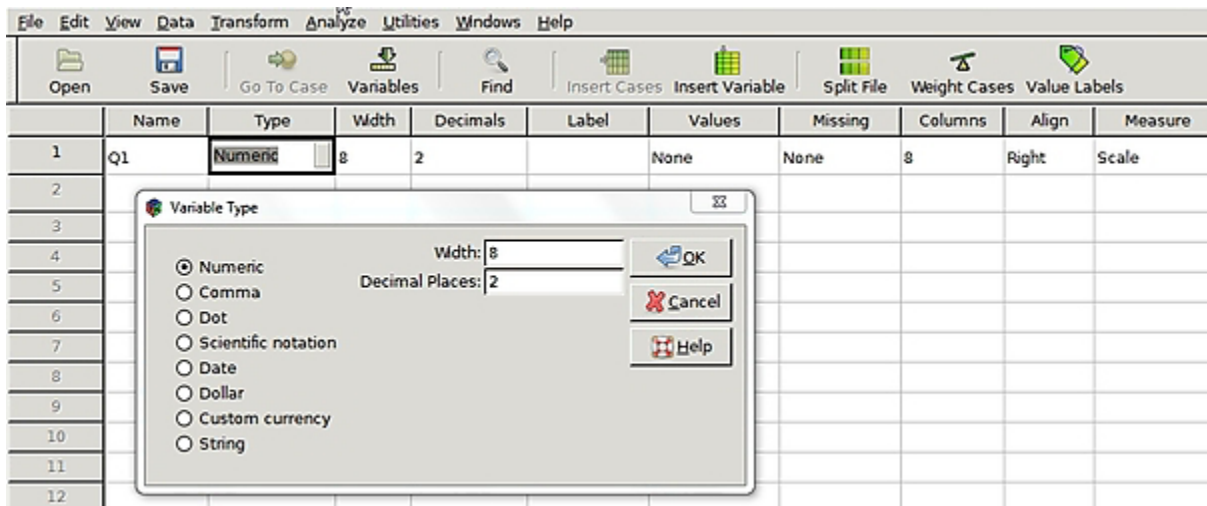
Figure 4.7: Edit *type* in variable view.

Width

Click a cell in the *Width*(See Chapter 5 [Attributes], page 92) column.

Select the up-down arrow that appears in the cell to alter the value.
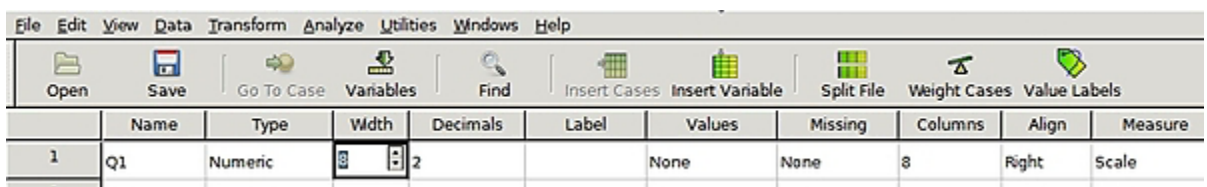
Figure 4.8: Edit *width* in variable view.

Decimals

Click a cell in the *Decimals*(See Chapter 5 [Attributes], page 92) column.

Select the up-down arrow that appears inside the cell to alter the value.
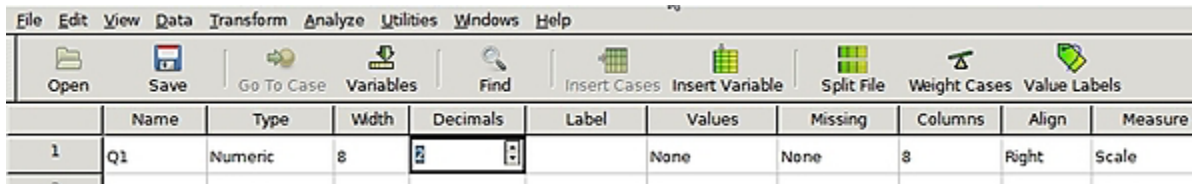
Figure 4.9: Edit *decimals* in variable view.

Label

Click a cell in the *Label* column.
Type a variable label in the cell.
Press enter.
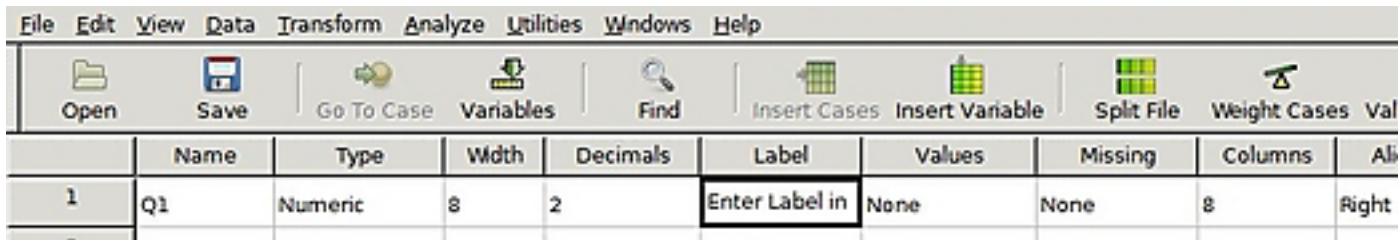
Figure 4.10: Edit *label* in variable view.

Values

Click a cell in the *Values*(See Chapter 5 [Attributes], page 92) column.

Click the gray box that appears inside the cell to bring up the *Value Labels dialogue* box.

Enter the *Value label.*

Click the *Add* button.

Repeat to add other labels, if necessary.

Press OK.
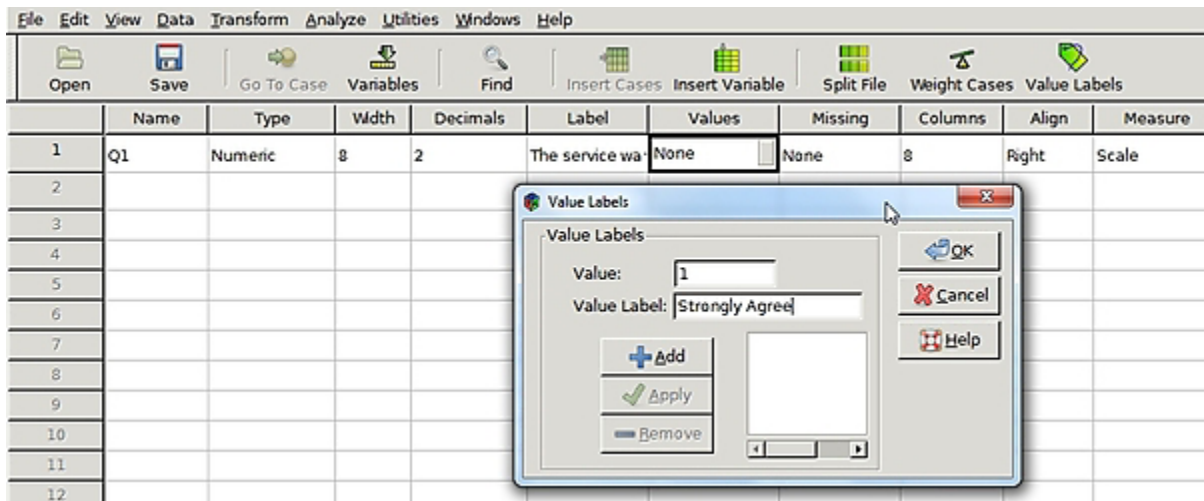
Figure 4.11: Edit *values* in variable view.

Missing

Select a cell in the *Missing* column.

Click the gray box that appears inside the cell to bring up the *Missing Values* dialogue box.

Enter either up to three discrete missing values or specify a numerical range plus one discrete value. See Chapter 5 [Attributes], page 92 for further information.
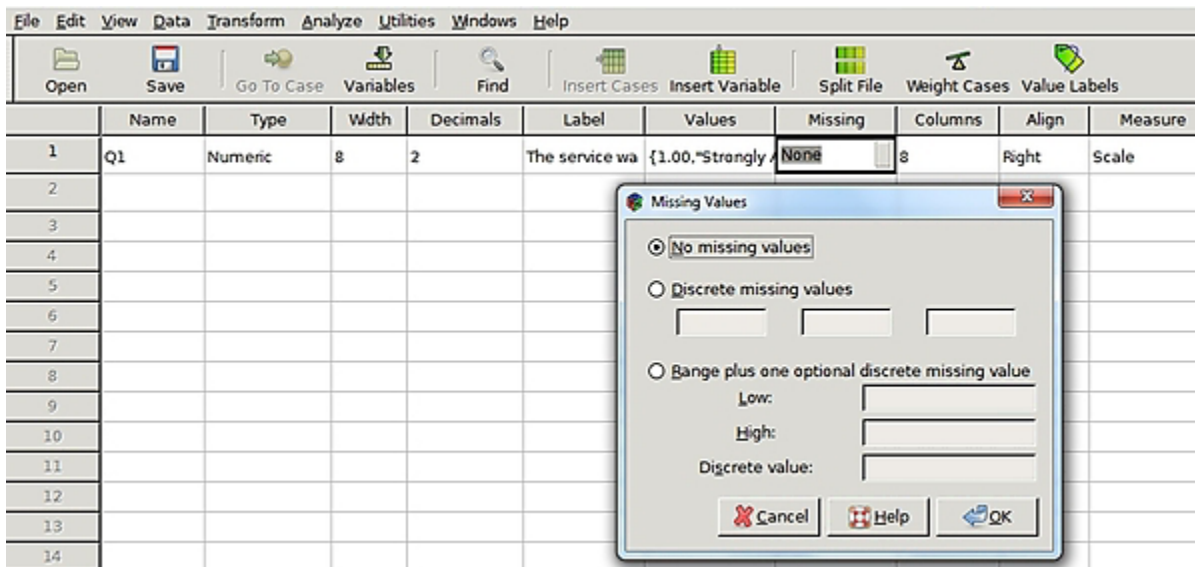
Figure 4.12

Columns

Select a cell in the *Columns* (See Chapter 5 [Attributes], page 92) column.

Click the up-down arrows that appears inside the cell to alter the value.

Figure 4.13: Edit *columns* in variable view.

Align

Align values left, right or center in a cell
Click a cell in the *Align* column.
Select *left*, *right*, or *center* from the drop down menu that appears.

Figure 4.14: Edit *align* in variable view.

Measure

Select a cell in the column *Measure* (See Chapter 5 [Attributes], page 92).
Select either *nominal*, *ordinal*, or *scale* from the drop down menu.

Figure 4.15: Edit *measure* in variable view.

4.2.3.2 New Variable

Enter a new variable by selecting *insert variable* from the *edit* drop down menu or tool bar. This function inserts a new variable with a generic name. In addition, if you already have at least one variable defined, then you can add a new variable to the left of the current column by right clicking on the variables title and selecting insert variable from the pop-up menu.
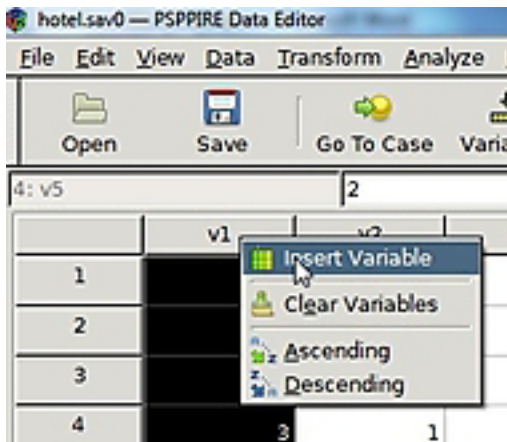
Figure 4.16: Insert new variable

4.2.3.3 New Case

There are a few ways to add a new case to an existing variable:

1. Select Edit → Insert Case from the drop down menu.

2. Select *Insert Cases* from the toolbar.

3. Right click on the row number above the insertion point to bring up a pop menu, and select *Insert Cases*.
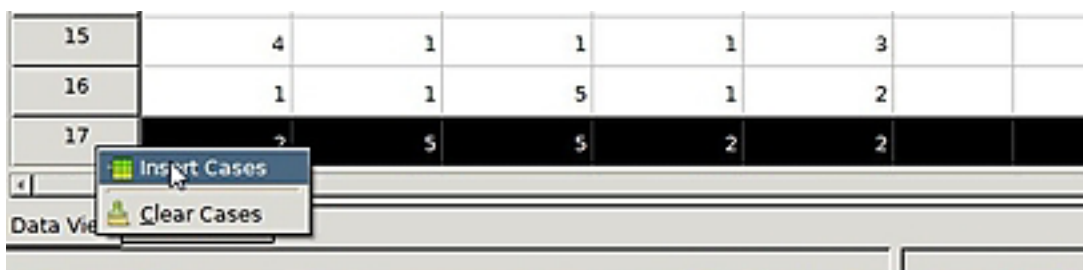
Figure 4.17: Insert new case

4.2.3.4 Read Variable in Data View

Hover the mouse over the variables name at the top of the column to view its label.
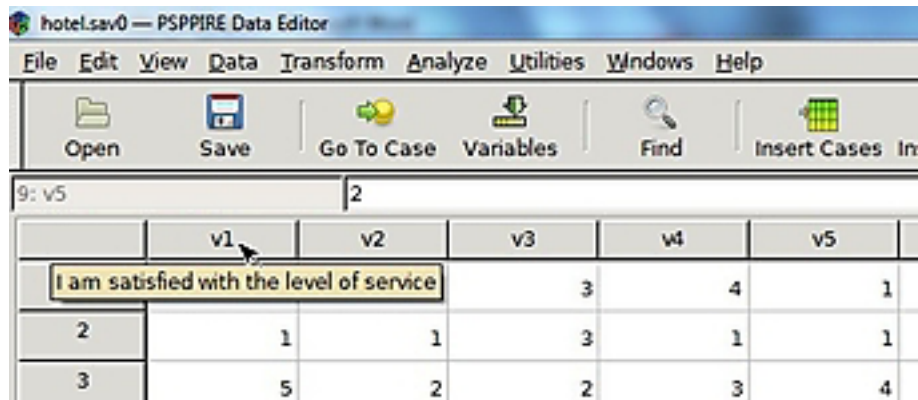
Figure 4.18: Hover mouse over variable title to view its label.

4.2.3.5 Change Data Value

Switch to *data view*, if you are not there already.
Select the cell you want to change.
Type the new value.
Press enter.

4.2.3.6 Select Entire Row or Column

Right click on the column title or row number to select the entire column or row, respectively.

4.2.3.7 Clear Variables

There are a couple of ways to clear columns:

In data view:

1. Right click on the variable name at the top of the column.
   Select *Clear Variables* from the pop-up menu.

2. Right click on the column to select it.
   Select Edit → Clear Variables from the drop down menu.

In variable view:

   Right click on the number title to the left of the row.
   Select *Clear Variables* from the pop-up menu.
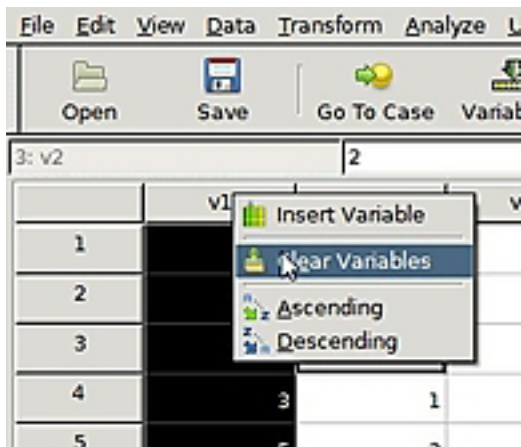
Figure 4.19: Delete variables

4.2.3.8 Clear Cases

There are a couple of ways to clear cases:

1. Right click on the row number to the left of the row.
   Select *Clear Cases* from the pop-up menu.

2. Right click on the row number to the left of the row.
   Select Edit → Clear Cases from the drop down menu.
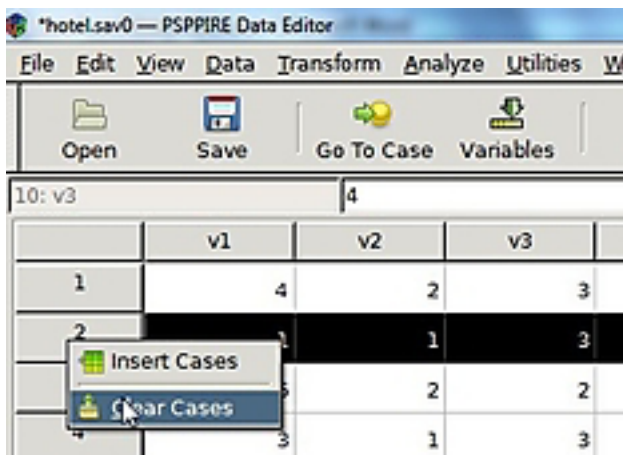
Figure 4.20: Delete cases

4.2.3.9 Sort Cases

There are two alternative ways to sort cases without using the menu:

1. Right click on the variable title and select *Ascending* or *Descending* from the pop-up menu.
2. Right click on the variable title and select Data → Sort Cases (See Section 4.5.5.1 [Sort Cases], page 30).
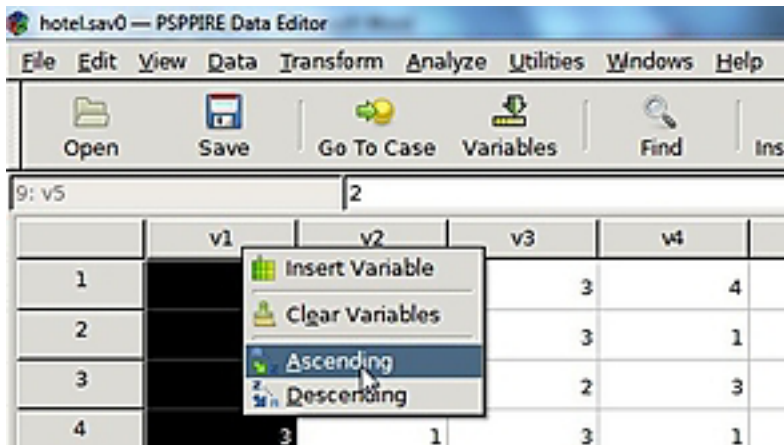
Figure 4.21: Sort cases

4.2.4 Status Bar

Located at the bottom of the data editor window, provides information about the data under process.
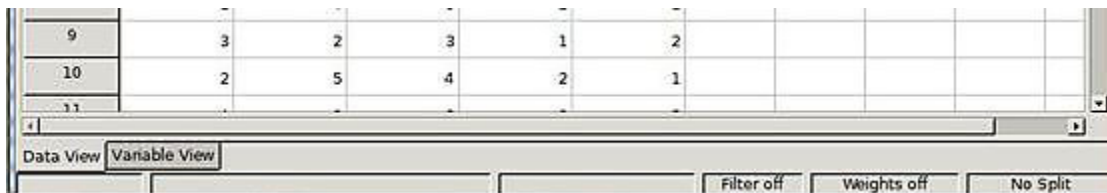
Figure 4.22: Status bar

Filter Status turns on when you filter a subset of cases in the current file for analysis.

Wieght Status states the variable you are weighting the data against in the analysis.

Split File Status indicates there are separate data into groups.

4.3 Output Viewer

The output viewer displays results, charts, and the instruction set executed, whether invoked from a menu command or the syntax window.

Note: The output viewer is not visible when PSPP loads, and appears upon execution of the first set of instructions.

Examples

1.
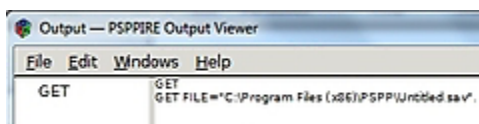
Figure 4.23: Output viewer displays open file instructions.

2.
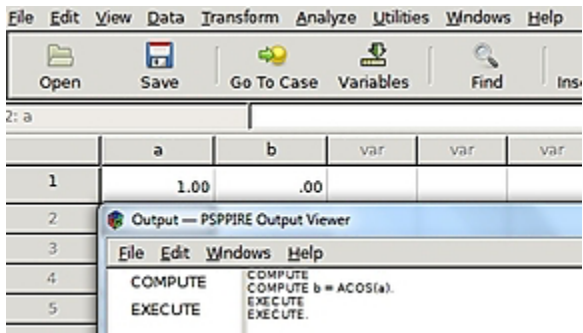
Figure 4.24: Output viewer displays executed statistics functions.

The right pane of the output viewer displays results such as tables, text, and charts. The left pane gives the corresponding outline.
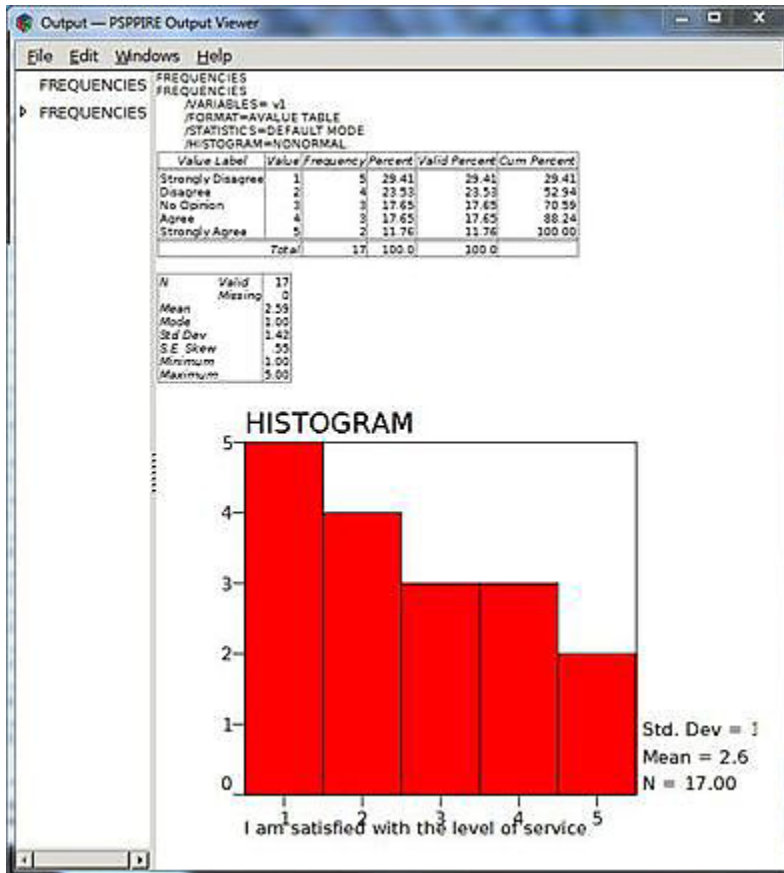
Figure 4.25: Output viewer panes
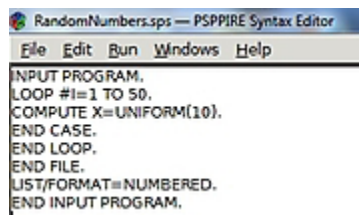
4.3.2 Save Output

To save the results in the output viewer select   File → export. File types include PDF, HTML, Open Document, Text, and Postscript.

4.4 Syntax Editor

Use the Syntax Editor window to enter command line instructions. For syntax and rules
(See Chapter 5 [Language], page 92).

Enter commands into the syntax window and then execute them by selecting run from
the menu. The instruction set can optionally be saved for later reuse. For example, the
following set of commands (See Figure 4.26) populates the data editor with a list of random
numbers.

Figure 4.26: Enter command line instructions in the syntax editor.

4.4.1 Enter Commands

Most data-editor-menu functions enable you to paste the command line equivalent to that
function directly into a syntax window from within the functions dialogue box, by selecting
the paste button.

Example

Place a command line instruction that generates descriptive statistics in the syntax editor
window.

Open file manufacturing.sav.

Select Analyze → Descriptive Statistics → Descriptives
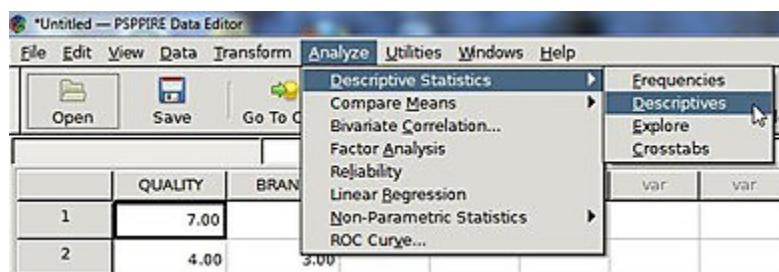
Figure 4.27: *Descriptives* function in the data editor drop down menu.

Select *variables and statistics* in the *Descriptives* dialogue box
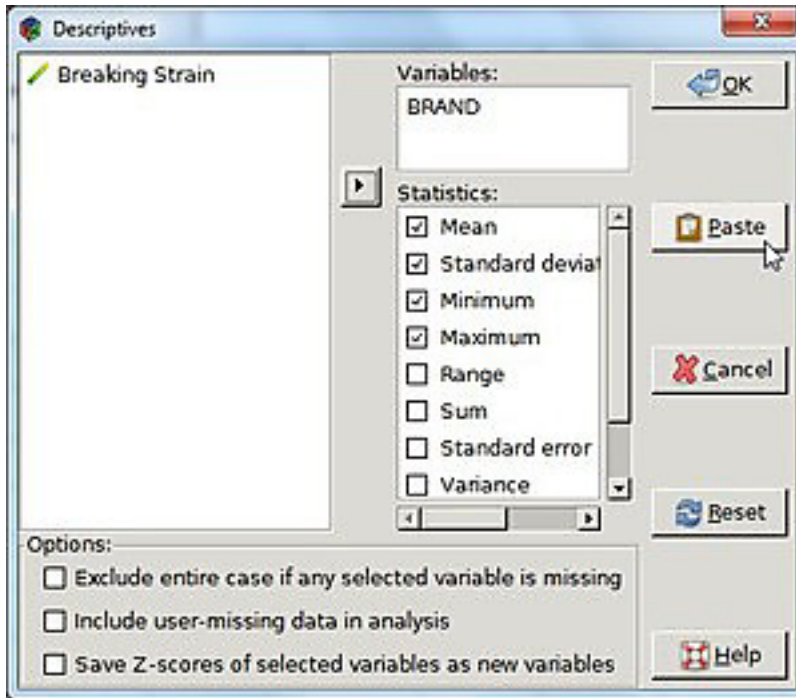
Figure 4.28: *Paste* button in the *Descriptives* dialogue box.

Click the Paste button to insert the command line equivalent of the function into the syntax editor.
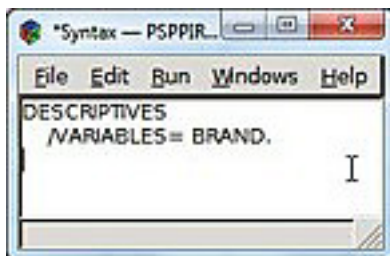
Figure 4.29: Command line equivalent of *Descriptives* function in the drop-down menu.

4.4.2 Run

Select run to execute code in the syntax window. Options include:

Execute all commands	Run → All
Execute highlighted commands	Run → Selection
Execute a single line at current cursor position	Run → Current Line
Execute at cursor position onwards	Run → To End

4.4.3 Save

Save a set of commands by selecting *Save* in the *File* drop down menu.

4.5 GUI Main Menu

4.5.1 Overview

Perform tasks and operations on data with functions available through the drop down menus and toolbar.

File Perform all file handling operations such opening, closing and saving files.

Edit Move data and text around with cut, copy, and paste operations. In addition, can use to insert data and cases and perform searches.

View Use to change fonts, display extra information such as a status bar and value labels, and toggle between *data* and *variable* view.

Data Perform operations on data sets, such as sort, transpose, apply weights, extract subsets of cases, and split files.

Transform Replace select data with the result of applying mathematical or other operations on it.

Analyze Perform statistical analyses on the data.

Utilities Use to insert comments and view summary information of each variable.

Windows Use to toggle between open windows, split and minimize them.

Help Provide help contents.

4.5.2 File

4.5.2.1 New

Open a new data or syntax file.

4.5.2.2 Open

Open data, syntax, system, or portable files.

4.5.2.3 Import Delimited Text Data

Import all or part of a delimited text data file into the data editor. A series of dialogue boxes guides you through this process as follows:

Importing Textual Data Dialogue Box Import all or part of the file.
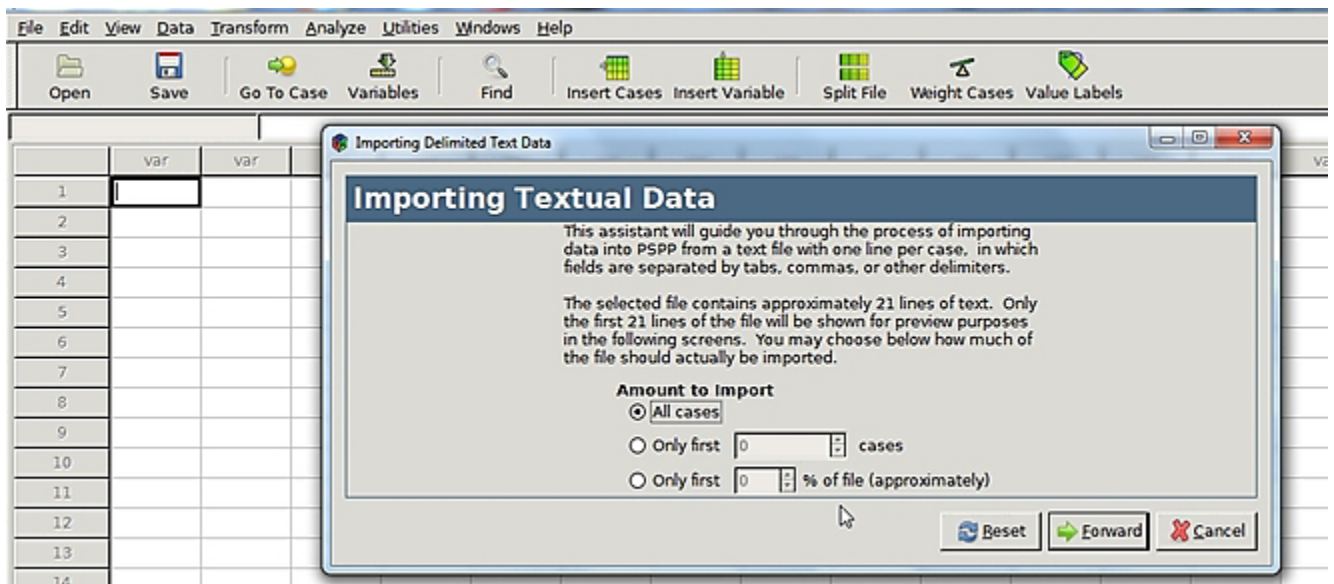
Figure 4.30: Import textual data dialogue box

First Line of Data Dialogue Select the first line of data.
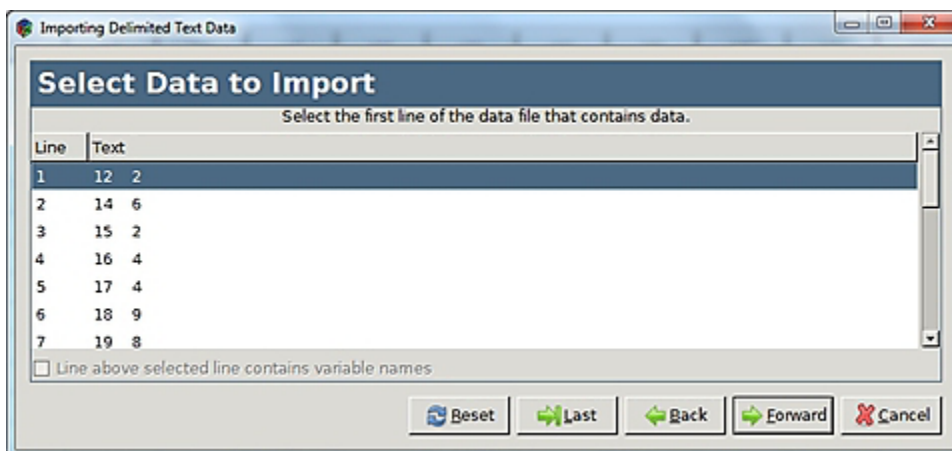
Figure 4.31: Select first line of data.

Choose Separators Dialogue Choose a delimiter. Options include:

Space

Colon (:)

Pipe (|)

Tab

Comma (,)

Semicolon (;)

Bang (!)

Hyphen (-)

Slash (/)

Custom Separator Choose how to quote separator characters as demonstrated below (See Figure 4.32).
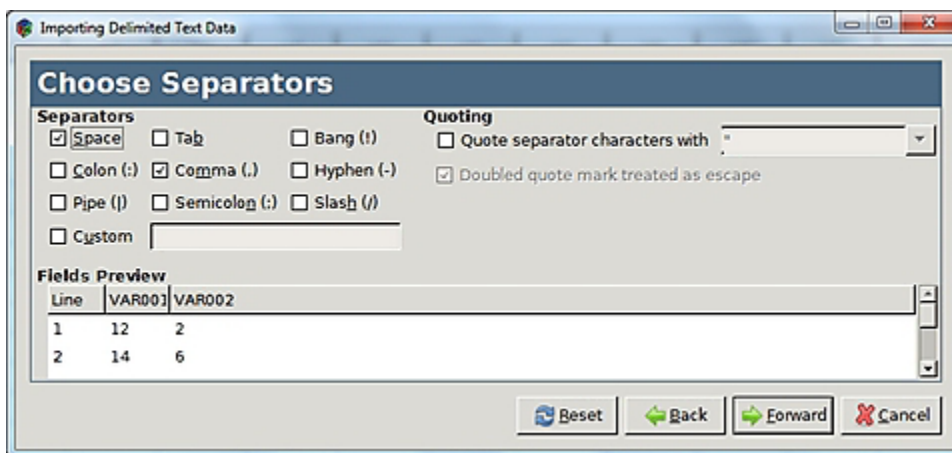
Figure 4.32: Choose data separators dialogue box

Adjust Variable Formats Verify and edit variable attributes (See Chapter 5 [Attributes], page 92).
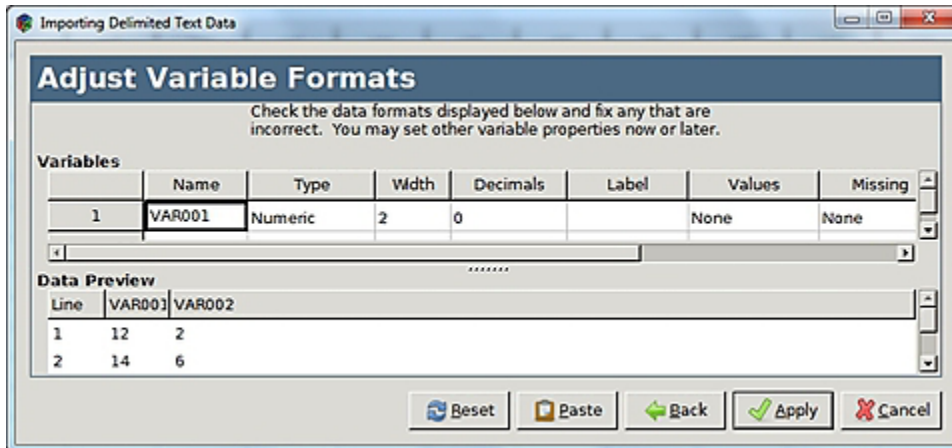
Figure 4.33: Check and fix data formats.

Select the *Apply* button to import the data.

4.5.2.4 Save

Save the current file.

4.5.2.5 Save As

Save the current file with a name and location of your choosing.

4.5.2.6 Display Data File Information

Working File Display the dictionary of the currently opened file.

Figure 4.34: Dictionary of current file.

External File Display the dictionary of a data file that is not yet open.

4.5.2.7 Recently Used Data

Links to recently used data files.

4.5.2.8 Recently Used Files

Links to recently used syntax files.

4.5.2.9 Quit

Close the current data file and shutdown PSPP.

4.5.3 Edit

4.5.3.1 Insert Variable

Insert a new variable in data view to the left of the current column. The editor fills in system-missing values for all cases as a placeholder. By default, PSPP inserts a numerical variable type (See Chapter 5 [Basic Numeric Formats], page 92).

Example

Select Edit → Insert Variable to enter a new variable.
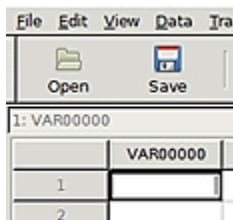
Figure 4.35: Insert variable in data editor.

4.5.3.3 Insert Cases

Insert a new case in the selected row if it is empty, or, above otherwise.

4.5.3.4 Go To Case

Jump to the selected case number.

4.5.3.5 Cut

Remove the selected text and place it elsewhere by selecting paste at the desired insertion point.

4.5.3.6 Copy

Copies text onto the clipboard. Highlight the text and then select Edit → Copy.

4.5.3.7 Paste

Places the contents stored on the clipboard at the insertion point.

4.5.3.8 Find

Look for a case.

4.5.4 View

4.5.4.1 Status Bar

Turn the status bar on or off. See Section 4.5.4.1 [Status Bar], page 29 for further information.

4.5.4.2 Font

View and edit the current font settings.

4.5.4.3 Grid Lines

Make the grid lines in the data editor window visible or invisible.

4.5.4.4 Value Labels

Show the value label associated with each result in data view.

Value Labels

The data editor window on the right displays the value labels of the numerical survey data displayed in the window to the left (See Figure 4.36).
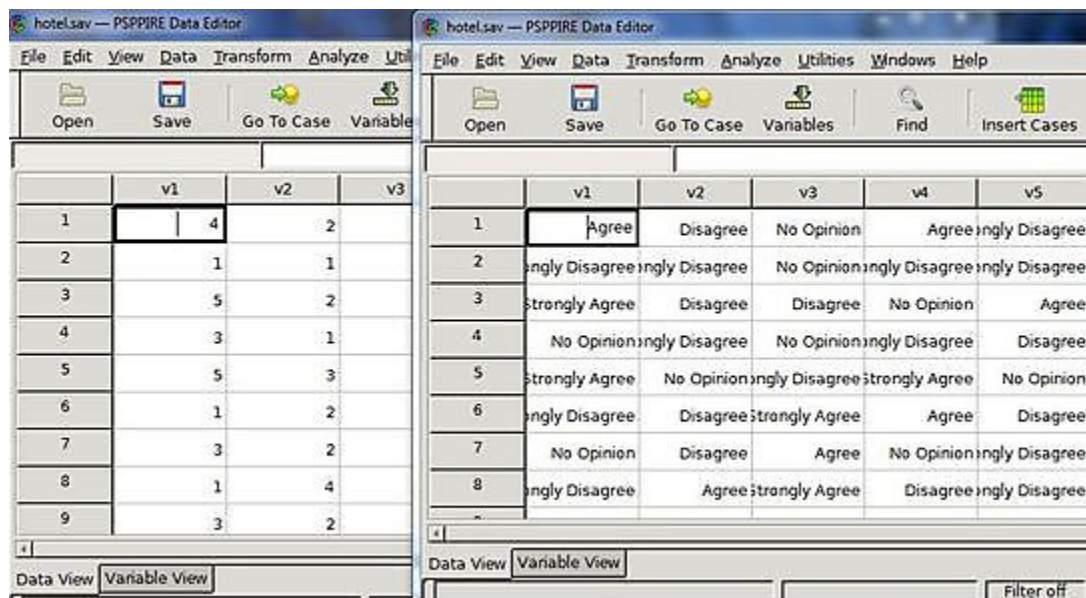
Figure 4.36: Value labels of the data in the left window.

4.5.4.6 Variables

Switch the data editor between data and variable view.

4.5.5 Data

4.5.5.1 Sort Cases

Sort cases in ascending or descending order based on one or more variables.

4.5.5.2 Transpose

Exchange rows and columns on data that is in the reverse order to the PSPP file format. Standard file format lists cases in rows and variables in columns.

4.5.5.3 Split File

Split data into groups using grouping variables. If you select more than one grouping variable then you need to nest each successive category within the previous one. sp 1

Note: Cases should be sorted with the *Sort Cases* command first such that they are in the same order as the grouping variables listed under the *Groups based on*textbox.
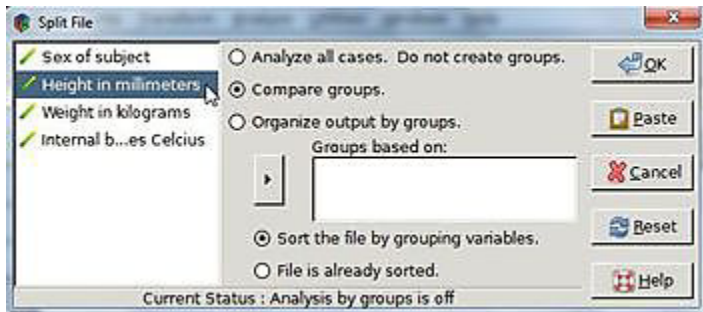
Figure 4.37: *Split File* Dialogue Box

Group a file with physiology data according to height.

Figure 4.38: Physiology data file

Open the file physiology.sav.
Select Data → Split File.
Select the *Compare groups*radio button in the pop up window.
Click *Height in millimeters* to highlight the variable.
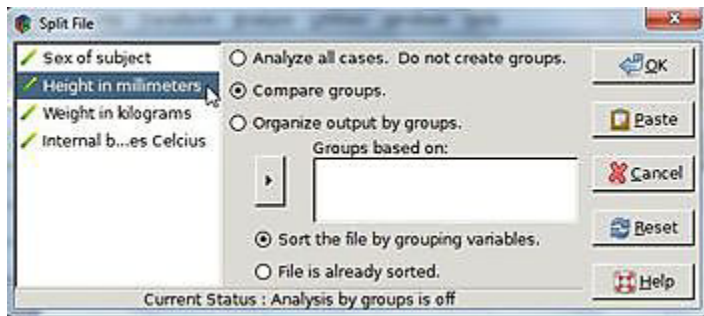
Figure 4.39: *Split-File* dialogue box

Select the arrow to the left of the *Groups based on* textbox to move the *height* variable inside that textbox.

Click OK to see the results in the output viewer.

Figure 4.40: Split-file results in output viewer

Compare Groups

Show split-file group output along side each other. For example, the output viewer displays a chart for each group in a single window.

Organize Output by Groups

Show results for each split-file group separately.

4.5.5.6 Weight Cases

Assign weights to variables. Note: Cases with a zero, negative, or missing value for the weighting variable are not included in the analysis.
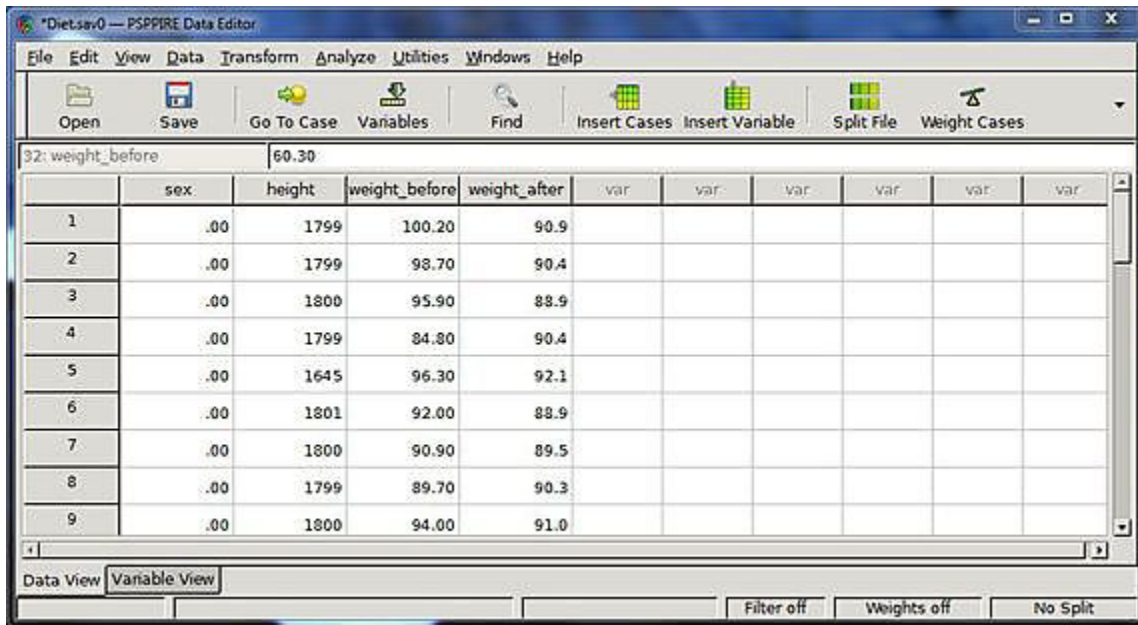
4.5.6 Transform

4.5.6.1 Compute

Perform operations on numeric and string variables. Compute new variables or replace existing ones.

Example

Compute the absolute difference in weight before and after participants follow a new diet plan (See ⟨undefined⟩ [Diet.sav], page ⟨undefined⟩).

Figure 4.41: Volunteers weights before and after dieting.

Open the file Diet.sav.

Select Transform → Compute from the drop-down menu to bring up a Compute Variable dialogue box.
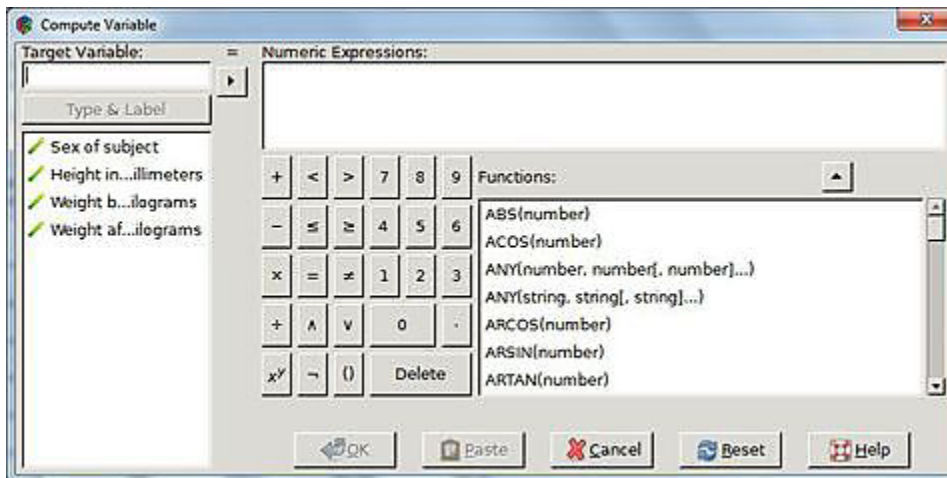
Figure 4.42: *Compute* dialogue box

Enter *weightChange* in the Target Variable textbox and click the emphType and Label button below it; This variable holds the result of the calculation.
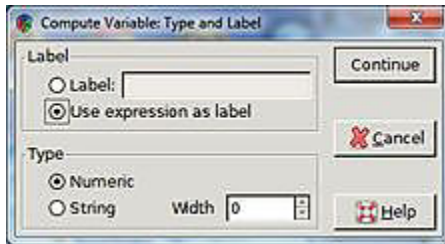
Figure 4.43: Compute solution *Type* and *Label*

Select the *Label* radio button in the *Type and Label* dialogue box, and enter *weightChange*.

Press Continue.

Select the ABS(number) function from the list of functions and press the up arrow to place the function up in the Numeric Expressions textbox.

Select *weightAfter* from the list of variables located on the left of the dialog box.

Note: when you hover the mouse over the list of variables this brings up their names.
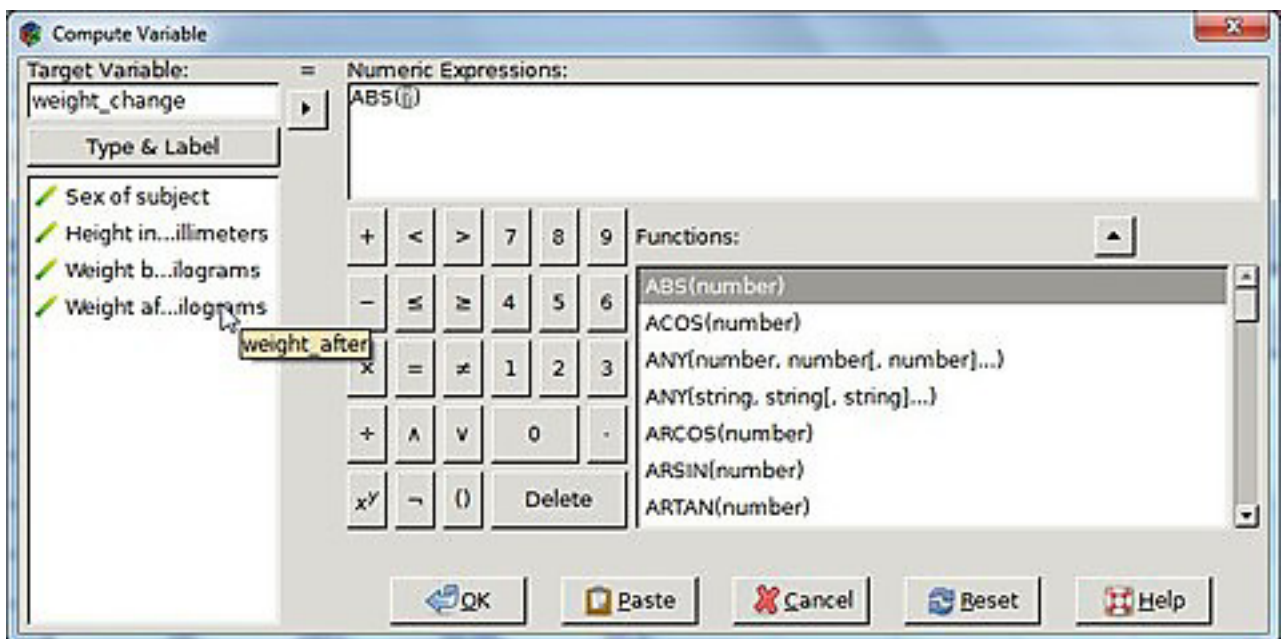
Figure 4.44: Enter numeric expression into *(*compute*)* dialogue box.

Click in between the parenthesis of the function in the *Numeric Expressions* textbox to
move the cursor position.

Click the right arrow to the left of the Numerical Expressions textbox to move
ABS(weightAfter) variable to the cursor position in the box.
Delete the question mark if it is still present in the ABS arguments.

Enter a space followed by a minus sign after *weightAfter*, as follows: ABS(weightAfter -)

Select the *weightBefore* variable from the list on the left side of the Compute Variable
dialog box.

Place the cursor after the minus sign of the mathematical expression in the *Numeric
Expressions* textbox .

Click the right arrow.
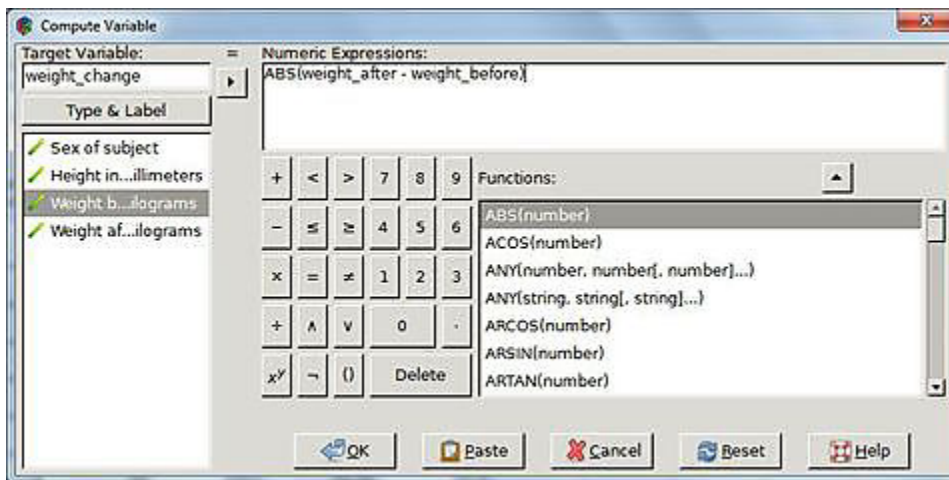
Figure 4.45: Enter formula into *compute* variable dialogue box.

Click OK to add the result to the data file.

Figure 4.46: New variable *weight change* from calculation in *compute* dialogue box.

4.5.6.3 Rank Cases

Create new variables that hold ranks, Savage scores, Normal scores, and percentile values for numeric variables in the current data file. PSPP automatically names and labels these such as to indicate original variable name and selected measures.
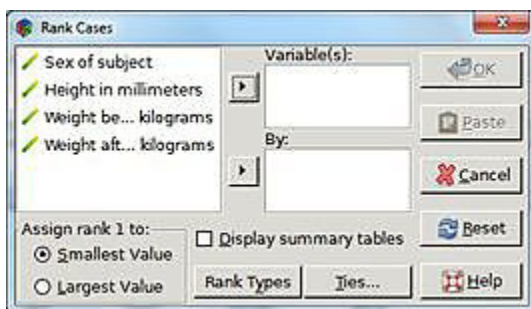
Figure 4.47: Rank cases by one or more variables.

Select the *Smallest Value* option from the *Assign rank 1 to* box for ascending order and *Largest Value* for descending.

Rank numerical variables by moving them into the Variables textbox (highlight the variable then click the right arrow).

Organize rankings into subgroups by moving the variables upon which the others will depend into the *By* textbox, e.g., move *gender* and *height* to calculate ranks for each combination of gender and height.

Rank Types

Bring up the *Rank Cases Types* dialogue box by clicking on the *Rank Types* Button towards the bottom of the *Rank Cases* dialogue. This dialogue box includes the following options: Rank, Savage score, Fractional rank, Fractional rank as a percent, Sum of case weights, Ntiles , Proportion Estimates, and Normal Scores.
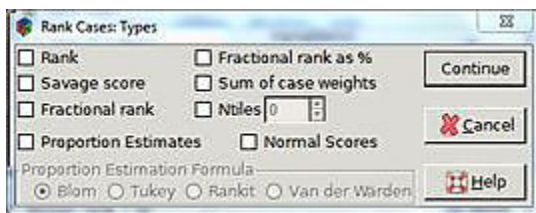
Figure 4.48: *Rank case types* dialogue box.

Rank The value of the new variable is its rank.

Savage Scores calculated using an exponential distribution

Fractional Rank Return rank divided by the sum of the weights.

Fractional Rank as a Percent Return rank divided by the number of cases as a percent.

Sum of Case Weights is returned.

Ntiles Return a rank according to the percentile group. The size of each percentile group is given by $100/Ntile$.

Proportion Estimates Estimate of the cumulative proportion of the distribution corresponding to a specific rank.

Normal Scores Return z scores corresponding to the estimated cumulative proportion.

Choose from a range of proportion estimate formulae when calculating proportion estimates and normal scores; options include Blom, Tukey, Rankit, or Van der Waerden.

> **Blom** Ranking variable based on proportion estimates which employ the formula $(r - 3/8)/(w+)$, where mathw is the sum of the case weights and r is the rank.

> **Tukey** Apply the formula $(r - 1/3)/(w + 1/3)$, where r is the rank and w is the sum of the case weights.

> **Rankit** Estimate expected values using $(r - 1/2)/w$, where w is the number of observations and r is the rank which ranges from unity to w.

> **Van der Waerden** Return Van der Waerdens transformation, given by the formula $r/(w + 1)$, where w is the sum of the case weights and r is the rank, ranging from 1 to w.

Ties Use to select the method for assigning rankings to cases with the same value on the original variable.
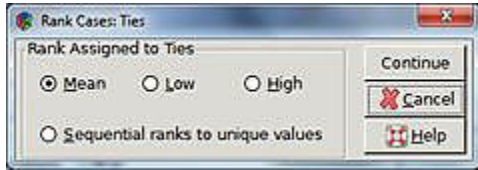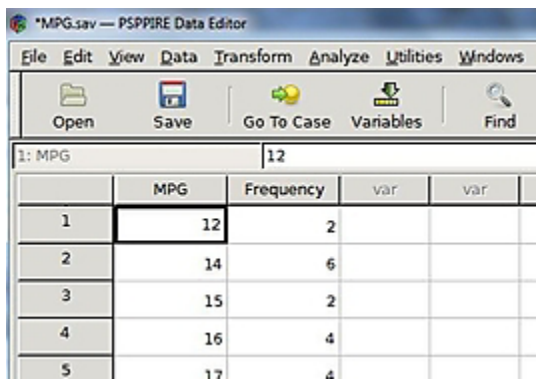
Figure 4.49: *Rank case ties* dialogue box

4.5.6.5 Recode into Same Variables

Recode different values of the same type (numeric or string) into a variable. There are two options available: 1) change a single value to a new one or 2) change a range of values to a category.

Example

Increase the value 12 Miles per Gallon (MPG) to 13 in the data set listing the mileage per gallon of a specific brand of car.

Figure 4.50: Mileage-per-gallon data file

Open file MPG.sav

Select Transform → *Recode into Same Variables* to bring up a Recode dialogue box.
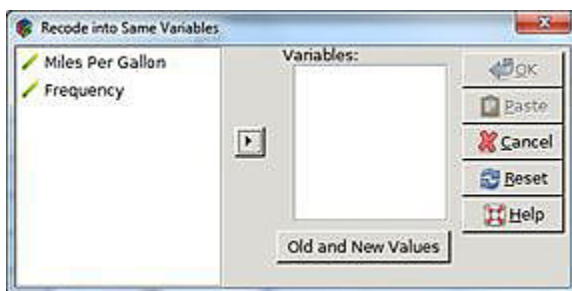
Figure 4.51: *Recode into same variables* pop-up box

Select *Miles Per Gallon* and click the right arrow to move the variable into the *Variables* textbox.

Click the *Old and New Values* button to bring up its dialogue box.
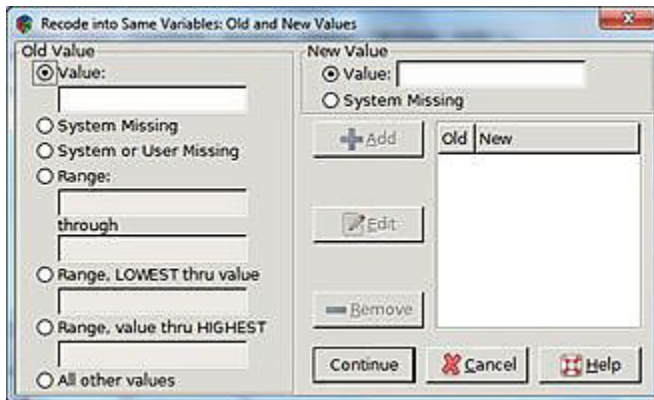
Figure 4.52: *Recode into same variables: Old and new values* pop-up box

Type the number 12 in the *Value* text box under *Old Value* on the left side.

Type the number 13 in the *Value* text box under *New Value* on the right side.
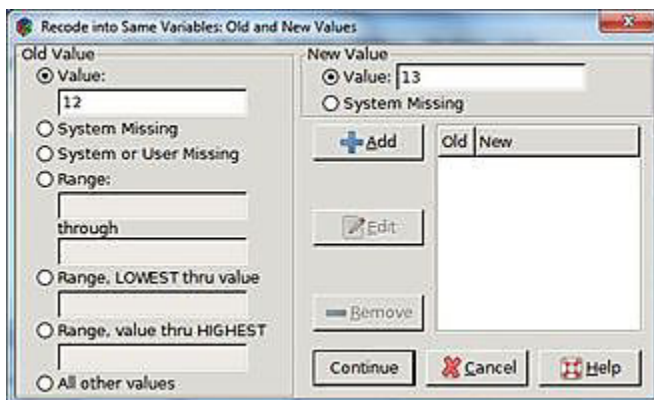
Figure 4.53: Replace cases with a new value.

Click the *Add* button. The number 12 now appears under *old* and 13 under *new*.
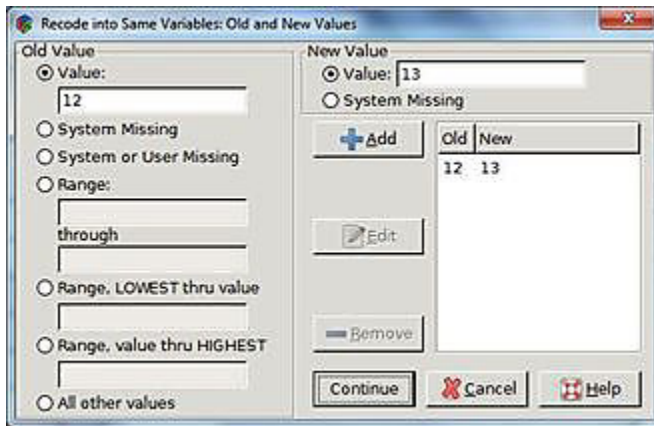
Figure 4.54: Display old and new case values in *Recode into same variables* pop-up box.

Example

Open file MPG.sav

Convert the mileage per gallon values in the MPG.sav automobile data set into three categories, as follows:

> If *Miles Per Gallon* is less than 19 MPG then give it a value of one.

> If *Miles Per Gallon* is greater than or equal to 19 MPG and less than 24 then give it a value of 2.

> If *Miles Per Gallon* is greater or equal to 24 then give it a value of three.

Select Transform → Recode into Same Variables.

Select the *range LOWEST thru value* radio button, and enter 18.

Enter the number 1 in the value field under *New Value* on the right side of the dialogue box.

Click the *Add button*, to get the *LOWEST*-18 listed under *old* and 1 under *new*.

Repeat with the other value ranges:
> Select the emphRange-through radio button and then enter 19 in the first text box and 23 in the second.

> Enter the number 2 in the value text box under *New Value*, and click the Add button.

> Select the *range thru HIGHEST* radio button and enter 24.

> Enter the number 3 in the value field under *New Value*, and click the Add button.
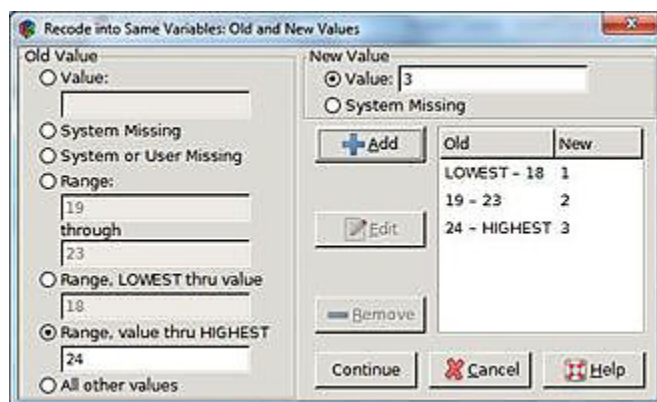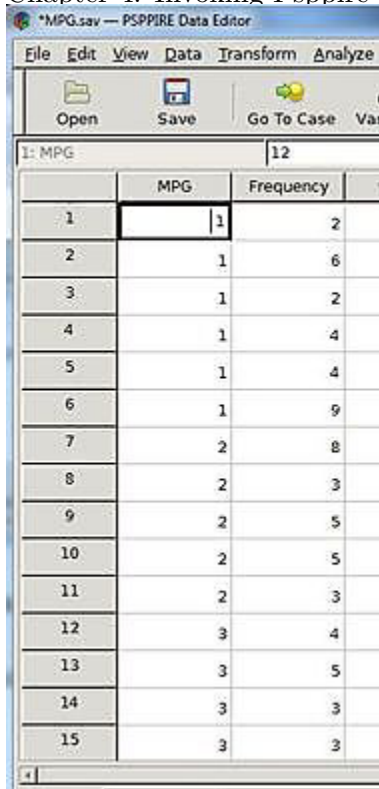
Figure 4.55: Collapse a range of values into categories with *recode into same variables*.

New Values dialogue box disappears.

New Variables dialogue box to put the new values in the

Figure 4.56: Calculated *frequency* data inserted into data editor.

4.5.6.8 Recode into Different Variables

Change variable values and save into a new one. There are two options available; change a single value to a new one or change a range of values to a category. Note, that the new variable type must be the same as the original; numeric or string (See Section 4.5.6.5 [Recode into Same Variables], page 40).
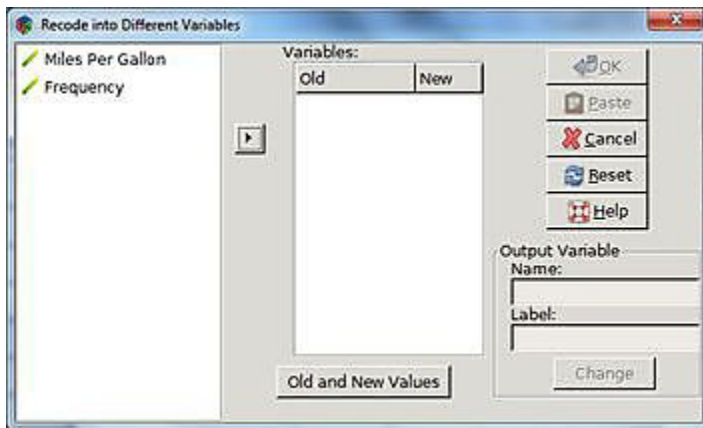
Figure 4.57: Recode case values into a new variable.

4.5.7 Analyze

4.5.7.1 Descriptive Statistics Definitions

Central Tendency Measure of the central position of a set of data.

Mean Average value, or first moment (see definition below), defined as sum of all data within a set divided by the number of cases in the set. Note: the mean is susceptible to

outlying values, e.g., the average temperature over a month in a region that has had a few unusually hot days will be higher than usual.

Note: **Moments** are the sums of integer powers of the values, where a higher integer power corresponds to a higher moment e.g. the variance is the second moment, skewness the third, and kurtosis the fourth.

Median The central value of an ordered set of numbers. In other words, 50% of the values fall above or below it. When the number of data values is even the median is the average of the middle two. The median is not as susceptible to outlying values as the mean.

Mode Most frequently occurring value. Note that there can be more than one mode in a data set.

Sum The addition of all values across the different cases excluding missing ones.

Dispersion A measure of the spread or variation in data values about the mean.

Standard Deviation A measure of the spread in values about the mean equal to the square root of the sum of the squares of the difference between each value and the mean, for all values, divided by one less than the number of cases.

Variance The standard deviation squared, or the second moment.

Range The difference between the largest and smallest numerical values in the set.

Minimum The smallest numerical value in the set.

Maximum The largest numerical value in the set.

S.E. Mean The standard error of the mean is a measure of by how much the value of the mean varies with different samples from the same distribution.

Distribution Statistics that describe the shape and symmetry of a distribution.

Skewness A measure of the asymmetry of a distribution. Estimate this by taking the third moment (proportional to the sum of the cube of the difference of each data value and the mean). Unlike the mean and variance the definition is such as to make it a dimensionless quantity, and this pure number characterizes the shape of the distribution. Positive values indicate the distribution has a more heavily pronounced right tail, while negative ones indicate a more prominent left one. When both tails are equal the skewness is zero.

Kurtosis A measure of the relative peakedness or flatness compared to a normal distribution. Estimate this by taking the fourth moment (proportional to the sum of the fourth power of the difference of each data value and the mean). Like the skewness, this is also a dimensionless quantity. A distribution with negative kurtosis is termed leptokurtic,

while a distribution with a positive value as platykurtic, and an in-between one is termed mesorkurtic.

4.5.7.2 Frequencies

Calculate statistics including central tendency (mean, median, mode, and sum), dispersion (standard deviation, variances, range, minimum, maximum, standard error of the mean) and distribution (skewness and kurtosis). Optionally, view frequency tables and charts.
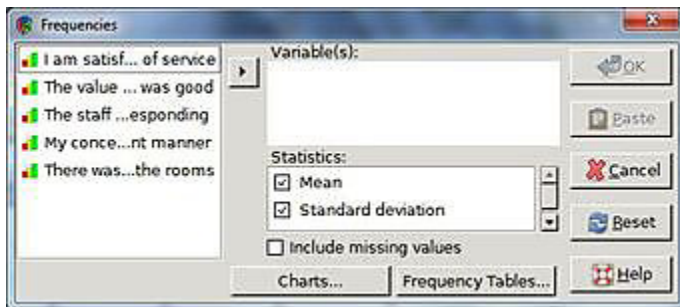
Figure 4.58: *Frequencies* dialogue box

The frequencies dialogue box lists all variables along the left side. In order to perform statistical analyses or plot charts move the variables into the *Variables* dialogue box. To do this, first, click on the variable until it is highlighted and then select the right arrow.
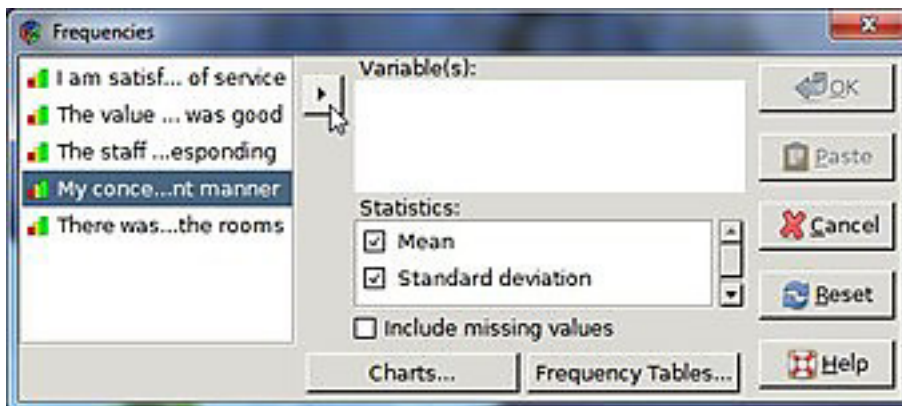
Figure 4.59: Select variables in the *frequencies* dialogue box.

Choose whether to display frequency tables, and in what way to order the tables; ascending or descending value or ascending or descending frequency.
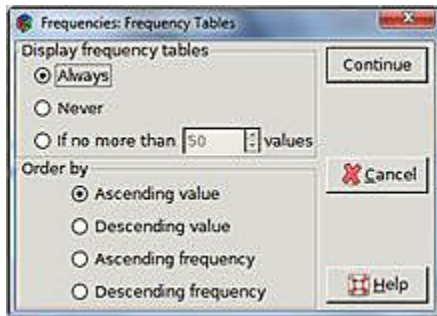
Figure 4.60: *Frequency tables* pop-up box

Plot histograms or pie charts with the Chart Dialogue box.
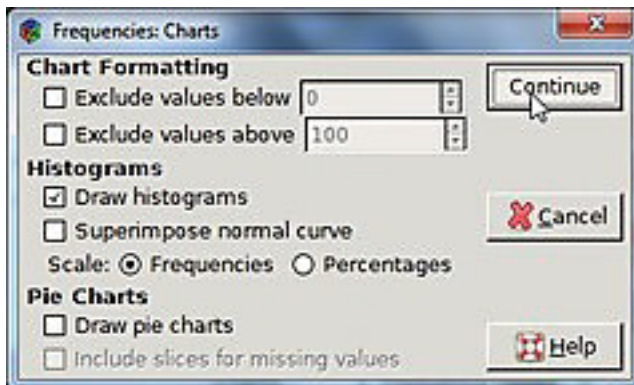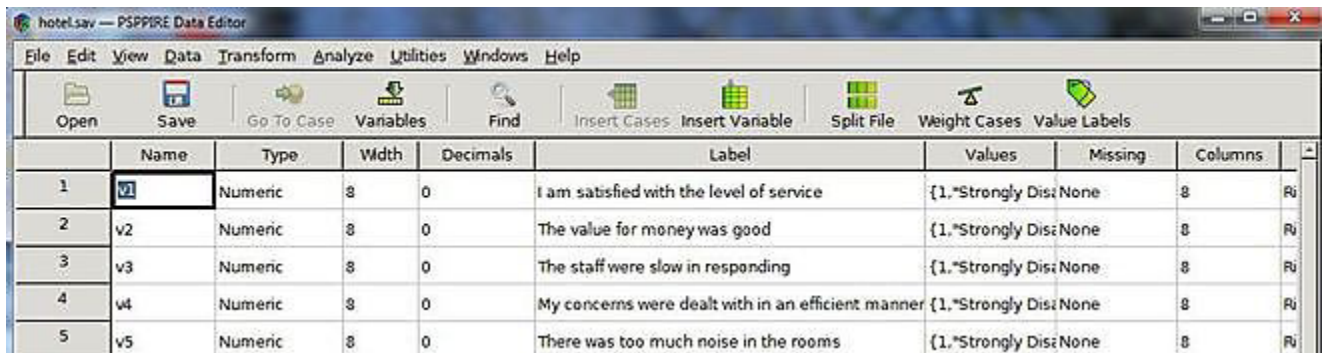
Figure 4.61: *Frequency charts* pop-up box.

Histogram options include scaling with either frequencies or percentages and superimposing a normal curve.

Pie charts come with the option to include slices for missing values.

Example

Analyze the extent to which hotel customers were satisfied with the level of service in the following survey (See Figure 4.62).

Figure 4.62: Customer satisfaction survey

Open file hotel.sav.

Select Analyze → Descriptive Statistics → Frequencies to bring up the *Frequencies* dialogue box.

Select the first variable from the list on the left side then click the right arrow to move it inside the *Variables* dialogue box.
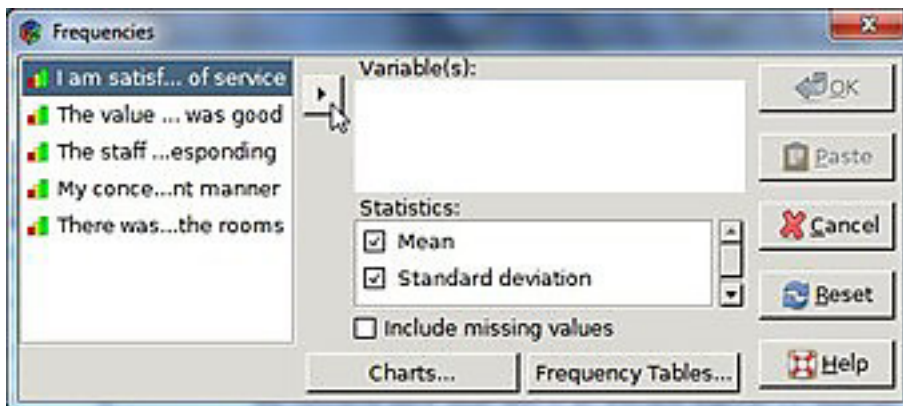
Figure 4.63: Pick variables in the *frequencies* dialogue box.

Select the *charts* button to bring up the *Charts* dialogue box.

Select *Draw Histograms* and press *Continue* to close this dialogue box.
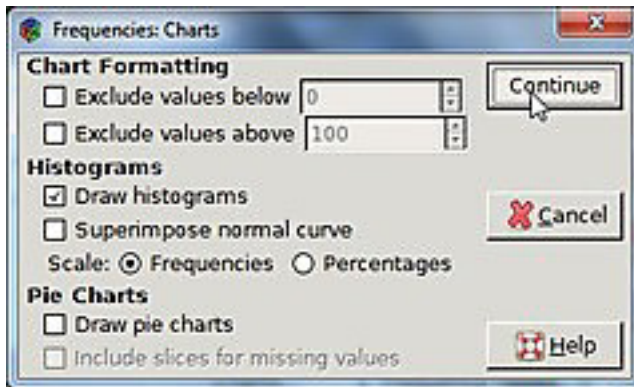
Figure 4.64: Plot histograms of selected data.

Select *Mode* from the *Statistics* options in the *Frequencies* dialogue box.
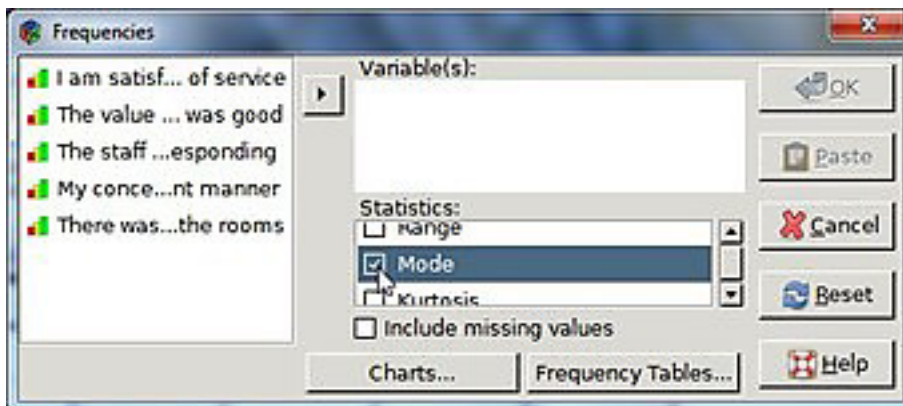
Figure 4.65: Select statistical analyses in the *frequencies* pop-up.

Press OK to display the results in the output viewer.
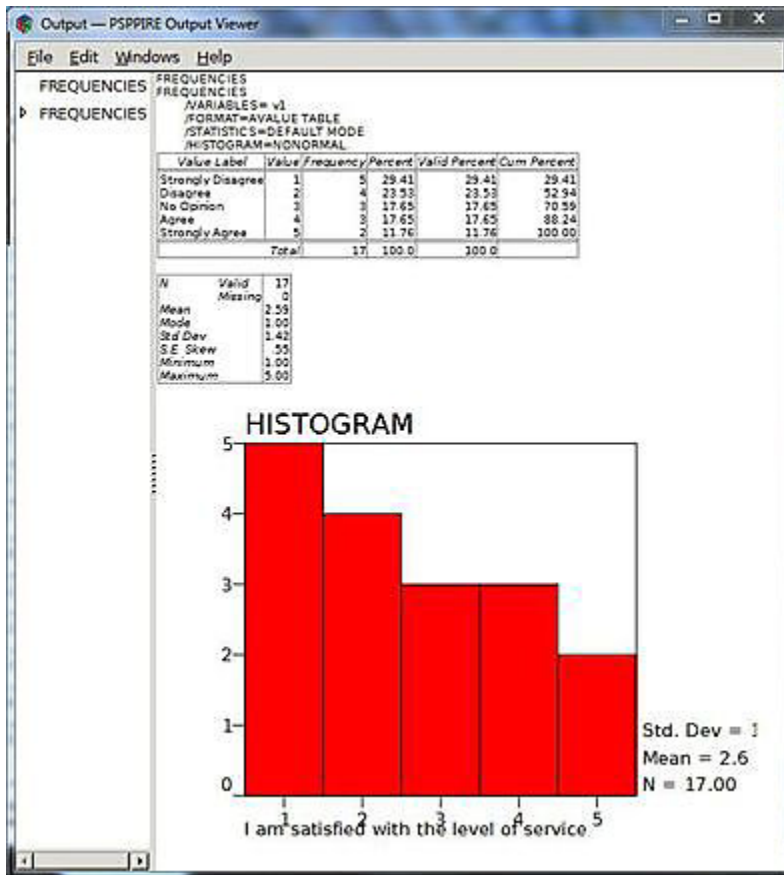
Figure 4.66: Results of *frequencies* function displayed in output viewer.

4.5.7.4 Descriptives

Display univariate summary statistics and z scores. Statistics include central tendency (mean), dispersion (standard deviation, variance, range, minimum, maximum, and standard error of the mean), and distribution (kurtosis and skewness).

Z-score is a dimensionless variable that classifies how many standard deviations a value is from the mean. Also known as standardizing or normalizing. Based on normal theory, these are best for quantitative variables with symmetrical distributions. Note, if the distribution is skewed the z scores will be as well.

Example

Begin by making a data set in the syntax editor. Write a small program to generate normally distributed numbers that represent SAT test scores with a mean of 500 and standard deviation of 110 as follows:

ax to open a syntax window.

nd lines (See Figure 4.67) into the syntax window.
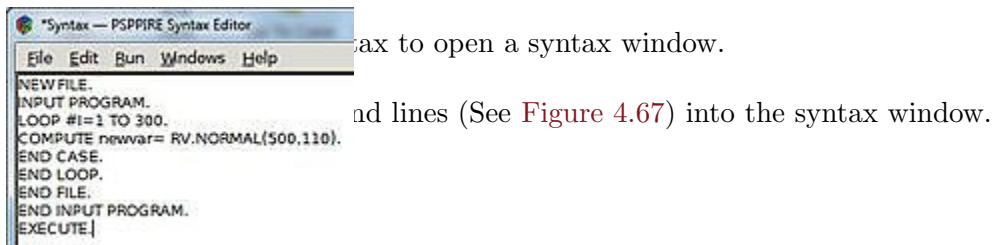
Figure 4.67: Generate data (random numbers) in the syntax editor.

Select Run → All to execute the program and populate the data editor with random numbers drawn from a normal distribution.

Figure 4.68: New data generated with command line instructions in the syntax editor.
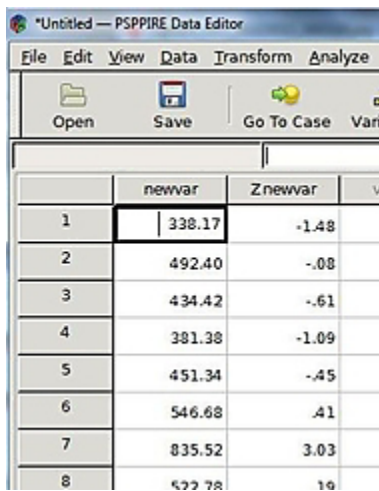
Next, calculate the z scores using the *Descriptives* menu command.

Select Analyze → Descriptive Statistics → Descriptives to bring up the *Descriptives* dialogue box.

Select the *newvar* variable by clicking on it in the variables list to the left of the dialogue box, and then by clicking the right arrow to move it over into the *Variables* textbox.

Select the *save Z scores of selected variables as new variables* check box.

Click OK to generate a list of z scores under the new variable in the data editor window.
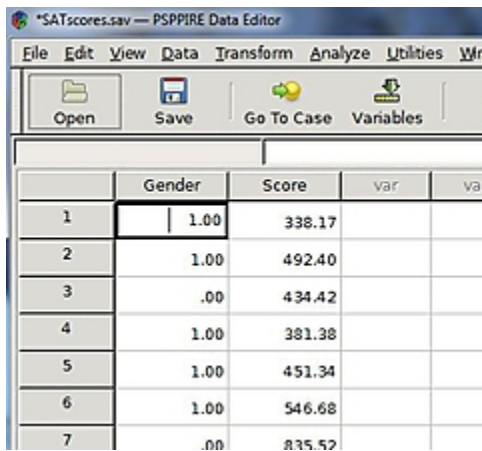
Figure 4.69: *Z-scores* generated with the *descriptives* function.

4.5.7.6 Explore

Provides descriptive statistics including percentiles and extremes, and in addition, enables you to examine subgroups of data. Descriptive statistics include the mean, median, variance, standard deviation, minimum, maximum, range, skewness, and kurtosis.

Example

Generate descriptive statistics of the following data set of SAT test scores versus gender for everyone, independent of gender, or break it down into subgroups and do a separate analysis for each gender type.

Figure 4.70: Data sheet of SAT scores by gender.

Open file SATscores.sav.
Start by looking at all test scores independent of gender.

Select Analyze → Descriptive Statistics → Explore to open the *Explore* dialogue box.
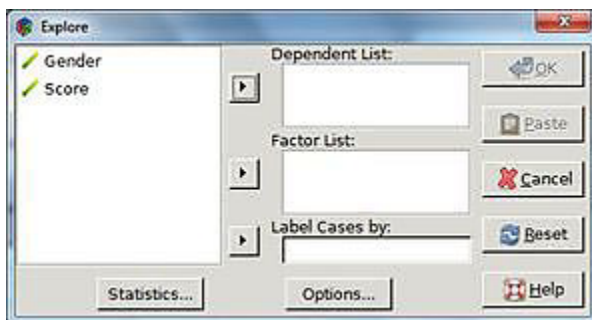
Figure 4.71: *Explore* dialogue box

Select *Score* and press the right arrow button next to the *Dependent List* box.

Click the *Statistics* button at the bottom left side of the dialogue box to bring up the *Statistics* pop-up box.

Select the *Descriptives* and *Extremes* check boxes, and then press continue.

Figure 4.72: *Explore* statistic functions

Press OK on the *Explore* dialogue box to display the results in the output editor.

Figure 4.73: *Explore* statistical analysis results displayed in output viewer.

Repeat the analysis, examining each gender group separately, where 0 represents male.

Select Analyze → Descriptive Statistics → Explore to open the Explore dialogue box.

Select *Score* and press the right arrow button beside the *Dependent List* textbox.

Select *Gender* and press the right arrow button beside the *Factor List* textbox.
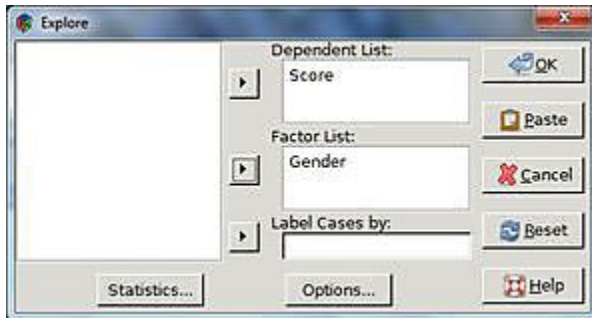
Figure 4.74: Compare SAT scores by gender with the *Explore* function.

Press the Statistics button at the bottom of the Explore dialogue box.

Select the *Descriptives* and *Extremes* check boxes and then press the continue button.

Press OK on the Explore dialogue box to display the results.

The output viewer lists different statistics for each group. For example, the lowest and highest test scores are 182.98 and 835.52, respectively, for the males, and, 179.03 and 805.10 for the female group.

Figure 4.75: Results of *Explore* function analysis in output viewer.

4.5.7.8 Crosstabs

Examine bivariate relationships through cross tabulation. This tool is useful for determining if one variable is dependent upon another, e.g., in a customer satisfaction survey you might discover a dependency of customer satisfaction levels on price.

Statistics and Measures of Association

Chi-Square Use Pearsons Chi-Square to test if the frequency in each cell is at least five; when the frequency is less than five and the contingency table is 2 x 2 in size then Fishers exact test is performed. When both variables are quantitative this function also returns a linear-by-linear association test.

Correlations Return Spearmans correlation coefficient and rho for numeric data when the values in the rows and columns are ordered. When both variables are quantitative returns the Pearsons R.

Nominal Data Measure the degree of association with several coefficients.

> **Phi** Measure of association that returns the square root of the chi-square coefficient divided by the sample size. Typically, apply this to 2 x 2 tables although it can handle larger sizes. For 2 x 2 tables phi is identical to the correlation coefficient, but for larger ones the interpretation can be more complex.

> **Contingency Coefficient** An adjustment to phi in order to make it suitable for tables larger than 2 x 2. The coefficient is equal to the square root of chi-square divided by chi-square plus the sample size. Note: For tables smaller than 5-by-5 this underestimates the level of association.

> **Cramer's V** This measure of association gives accurate results when the number of rows and columns are equal, independent of size. V is equal to the square root of chi-square divided by the product of the sample size and the length of the smallest row or column.

> **Lamda** Returns the percent by which you reduce the error when estimating the value of the dependent variable based upon on the value of the independent one. A lambda of zero means that there is no advantage in using the distribution of the independent variable to estimate the value of the dependent variable.

> **Uncertainty Coefficient** Otherwise known as the entropy coefficient, is the percent reduction in error accounting for the variance in the dependent variable, where variance is defined in terms of entropy. When the uncertainty coefficient is zero, the independent variable is not helpful in predicting the dependent variable.

Ordinal Data

> **Gamma** Returns the strength of association for ordinal data, ignoring tied (cases have equal values on both variables) pairs. Gamma is defined as the number of concordant pairs minus the number discordant ones over the sum of each type of pair. Cases form a concordant pair when the ranking agrees;

meaning one case has a higher value than another case on both variables. When one case has a higher value on one variable, but a lower value on another the ranking disagrees and the cases form a discordant pair. Values of gamma range from -1 to 1. A value of 1 indicates perfect agreement between the rankings, while a value of -1 means there is perfect disagreement. When gamma is equal to zero then there is no association between cases.
Note: Since gamma ignores ties, the strength of association between two variables might be over-exaggerated.

Somers'd Modification to Gamma that accounts for pairs tied on the dependent variable; values range from -1 for negative relationships to +1 for positive ones.

Kendall's Tau B A modification to gamma that takes into account tied pairs tied on both the dependent and independent variable. Kendalls Tau-B is symmetrical in that the same value is produced regardless of which variable is considered independent. Tau-b can only reach its maximum or minimum values of 1 and -1 for square tables.

Kendall's Tau C This is a variant of Kendall's Tau-B that can reach the maximum and minimum values of +1 and -1 for non-square tables.

Nominal by Interval

Eta Gauge of association when the dependent variable is a quantitative measure and the independent variable is categorical. Eta ranges from 0 to 1, with increasing values indicating a higher degree of association.

Cohen's Kappa Kappa is a measure of the degree to which two raters agree over their sorting of items into mutually exclusive categories, and takes into account agreement that occurs by chance. Values range from -1 to +1 and represent the proportion of agreements between raters after adjusting for those that take place by chance. A value of +1 indicates perfect agreement and -1 perfect disagreement. When kappa is zero, then there is no relationship between the raters and any agreements or disagreements are due to chance only.

Estimated Relative Risk The observed risk of having the outcome of interest be greater due to a specific factor. Use the odds ratio as an estimate or measure of risk if the occurrence of the factor is rare. A confidence interval greater than one indicates there is a greater chance of the event occurring in the presence of the factor. Note: this is only suitable for 2 x 2 tables.
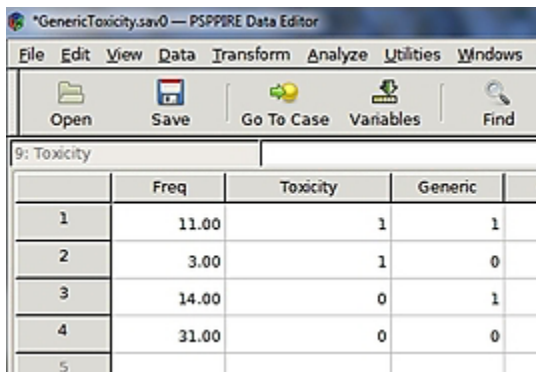
Example

Consider a medical study examining the relative risk of patients having cerebella toxicity who have taken a generic chemotherapeutic agent rather than a non-generic one.

	Toxicity+	Toxicity-
Generic+	11	14
Generic-	3	31

Table 4.1: Toxicity Versus Chemotheraputic Agent

Enter the data from the table (See Table 4.1) into a PSPP data sheet (See Figure 4.76), where +1 represents positive (+) and 0 negative (-). Alternatively, open file GeneralToxicity.sav.

Figure 4.76: Toxicity data in the data editor

Weight cases by selecting Data → Weight Cases, and then choose the *Weight Cases By:* option (See Figure 4.77).
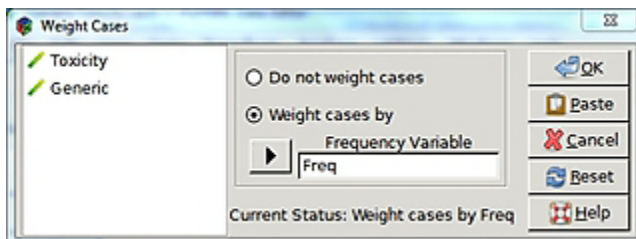
Figure 4.77: *Weight cases* function

Select Analyze → Descriptive Statistics → Crosstabs to open the *Crosstabs* dialogue box.

Move the variable *toxicity* from the list on the left side of the dialogue box into *Rows* and *Generic* into *Columns*.

Click *Statistics* to bring up the *Statistics* dialogue box.
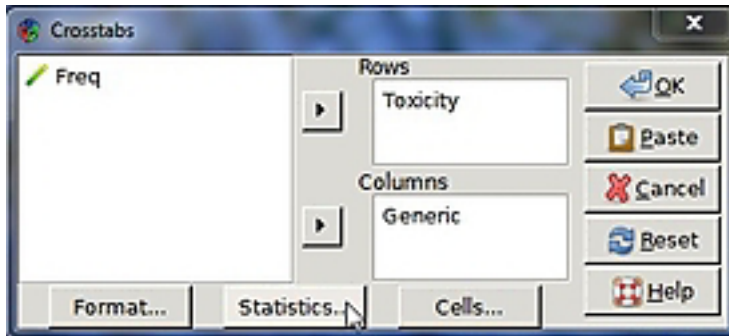
Figure 4.78: *Crosstabs* dialogue box.

Select the *Risk* check box and then click *Continue* to close this dialogue box.

Click OK on the *Crosstabs* dialogue box to display the results in the output viewer.

Figure 4.79: *Crosstab* statistics results in output viewer.

4.5.7.10 Compare Means

4.5.7.11 One-Sample T-Test

Tests if the mean of a single variable differs from a specified constant.

Compare the mean of the measured lifetime of 100 soft white light bulbs to that claimed by the manufacturer (700 hours).

Open the file LightBulbHours.sav.

Select Analyze → Compare Means → One Sample T Test.

Select the *hour* variable from the list on the left side of the dialogue box and then click the right arrow to move it into the *Test Variables* textbox.

Enter the value 700 into the *Test Value* textbox.
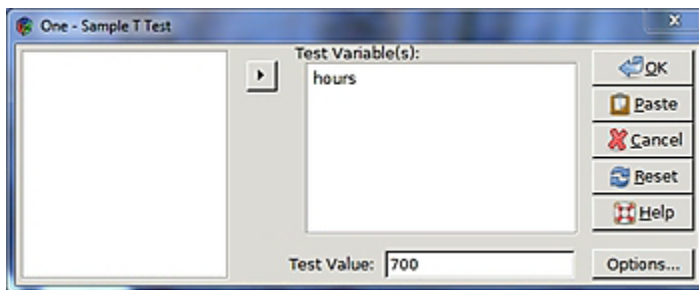
Figure 4.80: *One-Sample T Test* dialogue box

Click OK to view the results in the output window.

Figure 4.81: *One-Sample T Test* output results

4.5.7.12 Independent Samples T Test

Compare means of the same variable taken from two independent samples. Although the test requires normally distributed data, minor departures from this assumption are tolerable, and there is no requirement that the samples are the same size.

Example

Compare mean heights, in centimeters, of ten twelve-year-old boys to ten twelve-year-old girls.

Enter the following *height* and *gender* data (See Table 4.2) into a data file, where 0 represents female and 1 male, or open file Height12YearOlds.sav.

152.71	0
142.57	0
155.48	0
148.79	0
167.74	0
149.51	0
142.00	0
150.21	0
171.19	0
141.99	0
152.67	1
139.65	1
152.19	1
156.94	1
151.86	1
157.79	1
152.49	1
157.52	1
156.95	1
141.61	1

Table 4.2: Height of twelve year old children

Name the first variable *Height* and the second *Gender*.

Select Analyze → Compare Means → Independent Samples T Test.

Select *Height* from the list on the left side of the dialogue box and then click the upper right arrow to move it into the *Test Variables* box.
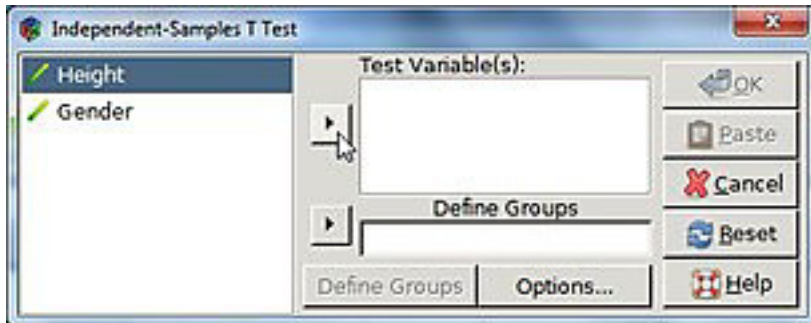
Figure 4.82: *Independent-Samples T Test* dialogue box

Select *Gender* and click the lower right arrow to move the variable into the *Define Groups* area.

Click the *Define Groups* button at the bottom of the dialogue box.
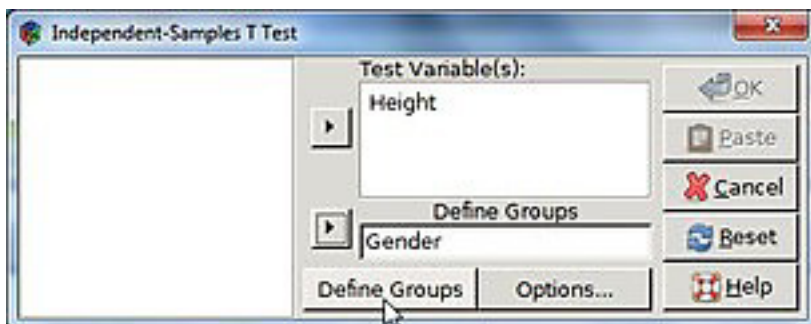
Figure 4.83: Define variables and groups in the *Independent-Samples T Test* dialogue.

Enter 0 into the *Group1 Value* textbox of the *Define Groups* dialogue box.

Enter 1 into the Group2 Value textbox of the *Define Groups* dialogue box.

Select *Continue.*
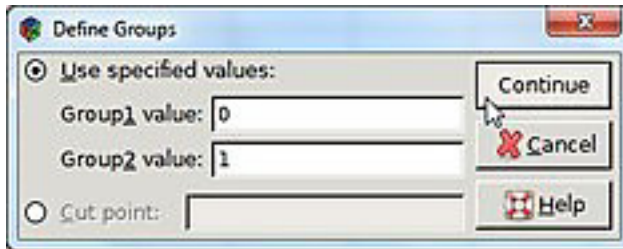
Figure 4.84: Organize data into groups.

Click OK on the *Independent-Samples T Test* dialogue box to examine the results in the output viewer.

Figure 4.85: Results of the *Independent-Samples T Test* in the output viewer.

4.5.7.14 Paired Samples T Test

Compare means of two sets of quantitative data, where data sets are organized into pairs with a definite relationship, e.g., one set of data might represent a measurement taken before some event and the other a measurement after. Each pair is then the measurement before coupled with the measurement after.

Compare the means of the following measurements (See Figure 4.86) from nine asthmatics of their peak expiratory flow rate before and after a walk on a cold winters day.

Figure 4.86: Before and after data

Enter the data in See Figure 4.86 into a new data file.
Select Analyze → Compare Means → Paired Samples T Test.
Highlight the *Before* variable with the mouse and then click the right arrow to place it in the *Test Variables* dialogue box under *Var1*.
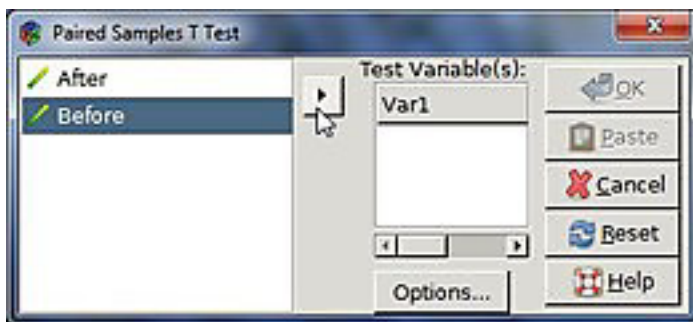
Figure 4.87: *Paired Samples T Test* dialogue box.

Repeat the same steps with the *After* variable to place it under *Var2*.
          sp 1

Click OK to display the results in the Output window.

Figure 4.88: *Paired Samples T Test* output results.

4.5.7.15 One-Way Anova

Indicates if there are significant differences between means of three or more independent groups, where the independent variable consists of two or more categorical independent groups and the dependent one is either interval or ratio. Note: this only tells you if the means of the groups are significantly different and not which ones.

Optional Statistics include:

Descriptives (mean, standard deviation, standard error of the mean, minimum, maximum, and 95% confidence intervals for each dependent variable)
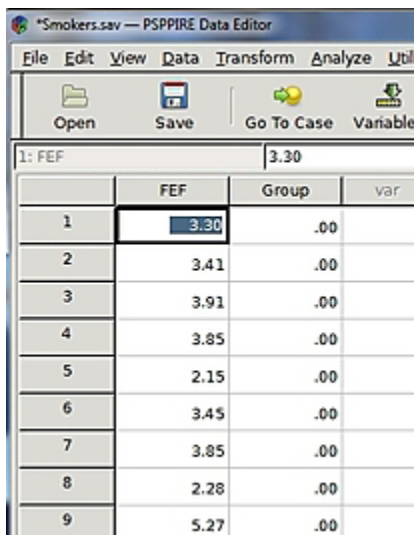
Homogeneity of variance test that calculates the Levene statistic to look for the equality of variances

Specify contrasts the you want tested by the t statistic with the *Contrasts* button in the dialogue box.

Example

Determine if the mean of each smoker group's lung capacity varies, where the different groups include non-smoker, light-smoker, medium-smoker, and heavy-smoker.

Open the data file Smokers.sav. This lists forced expiratory flow measurements under the variable *FEF* and the corresponding group type (non-smoker=0, light-smoker=1, medium-smoker=2, heavy-smoker=3) under *Groups*.

Figure 4.89: Data describing the lung capacity of several groups of smokers.

Select Analyze → Compare Means → One-Way ANOVA
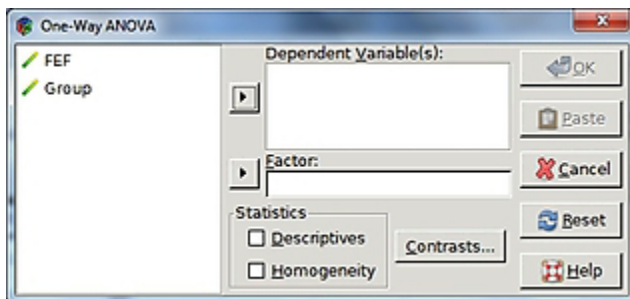
Figure 4.90: *One-Way ANOVA* dialogue box

Click on *FEF* from the list of variables on the left side of the dialogue box then select the upper right arrow to move it into the *Dependent Variables* textbox.
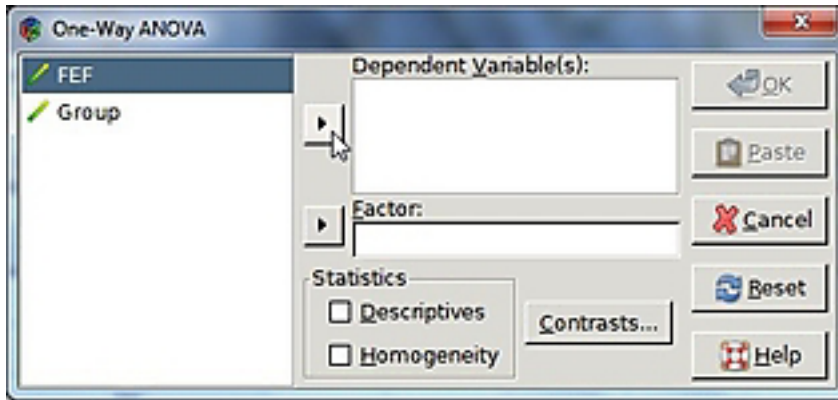
Figure 4.91: Select dependent variable in *One-Way ANOVA* dialogue box

Select the *Group* variable and click the lower right arrow button to place it inside the *Factor* textbox.
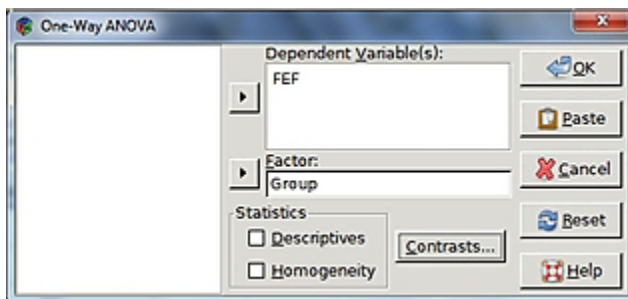
Figure 4.92: Dependent and independent variables in the *One-Way ANOVA* dialogue.

Click OK to displays the results in the output viewer.

Figure 4.93: *One-Way ANOVA* output

Contrast

Contrast is a weighted sum of the means and enables you to test your research hypothesis as opposed to the statistical one.

Example

Open the data file Smokers.sav. This lists forced expiratory flow measurements under the
variable *FEF* and the corresponding group type (non-smoker=0, light-smoker=1,
medium-smoker=2, heavy-smoker=3) under *Groups*.
Check if the mean lung capacity of non-smokers varies from that of smokers, independent
of how many they smoke. There are four different groups, of which three are smokers of
different intensities. This means the non-smoking group should have a mean with a
weighting of unity and the others a weighting of one-third.

Select *Contrasts* on the *One-Way ANOVA* dialogue box.
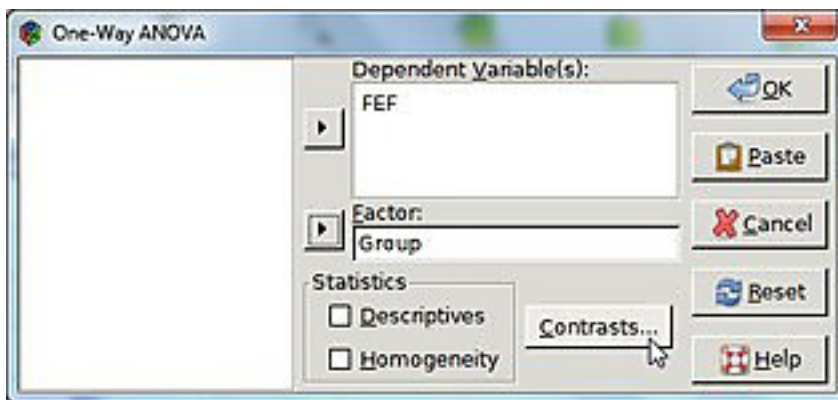
Figure 4.94: Use *contasts* button to calculate weighted sum of the means.

Enter 1.0 in the *Coefficients* textbox and then click the *Add* button.

Add the values -0.333, -0.333, and -0.334 to the *Coefficients* list.
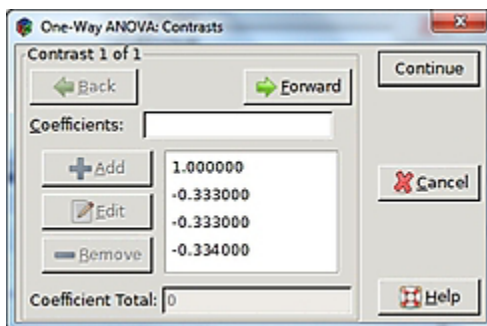
Figure 4.95: *Contrasts* dialogue box

Press *Continue* to close the *Contrasts* dialogue box.

Press OK on the *One-Way ANOVA* dialogue box to display the results in the output viewer.

Figure 4.96: *One-Way ANOVA* output

4.5.7.18 Bivariate Correlation

Computes Pearsons correlation coefficient, defined as a value between -1 and +1 that indicates the degree of association between variables. A positive value means large values of one variable are associated with large values of the other and,likewise, small values of one variable are associated with small values of the other. A negative value indicates that large values of one variable are associated with small values of the other and vise-versa.

Example

Use this feature to investigate the correlation, if any, between ice cream sales and outside temperature.

Open a new data file.

Enter the following pairs of values listed (See Table 4.3) of the number of ice cream cones sold each day one summer at a specified store along with the corresponding daytime high temperature, or open file IceCream.sav .

Number of Cones Sold	Temperature (Fahrenheit)
171.00	76.00
161.00	71.00
156.00	66.00
151.00	61.00
154.00	73.00
143.00	61.00
146.00	64.00
157.00	69.00
171.00	78.00
171.00	78.00
166.00	76.00
168.00	76.00
176.00	81.00
181.00	83.00
181.00	86.00
144.00	60.00
174.00	87.00
177.00	89.00
178.00	86.00
186.00	91.00
164.00	75.00
161.00	74.00
163.00	72.00
149.00	69.00
161.00	71.00
155.00	74.00
170.00	86.00
179.00	91.00
176.00	71.00
165.00	82.00

Table 4.3: Ice Cream Cones Sold Versus Outside Temperature

Select Analyze → Bivariate Correlation, to bring up the *Bivariate Correlation* dialogue box.

Select *Cones* and click the right arrow button to move it into the right-hand-side textbox.
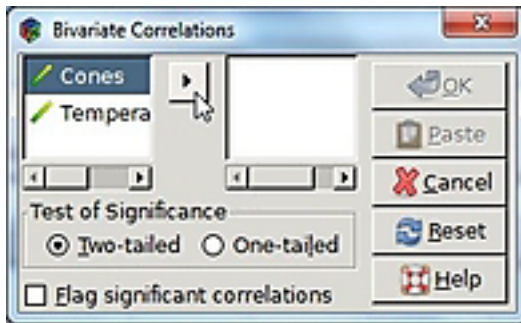
Figure 4.97: *Bivariate Correlations* dialogue box

Repeat the same steps with the *Temperature* variable.

Click OK to display the results in the output viewer.

Figure 4.98: emphBivariate Correlations output

4.5.7.20 Factor Analysis

Helps identify the underlying variables, or factors, to explain correlations. Variables should be either interval or ratio, and not categorical.

Note: PSPP assumes the data has a nearly bivariate normal distribution for each pair of variables and that observations are independent.

Two main statistic functions are available: *Extraction* and *Rotation* (see fig x).
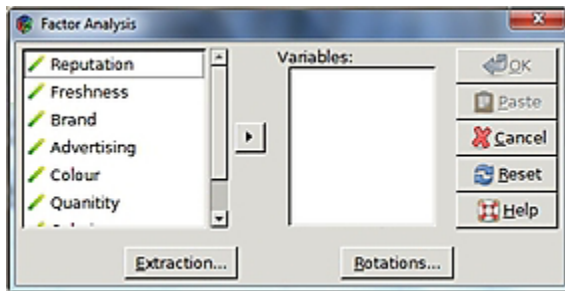
Figure 4.99: *Factor Analysis* dialogue box
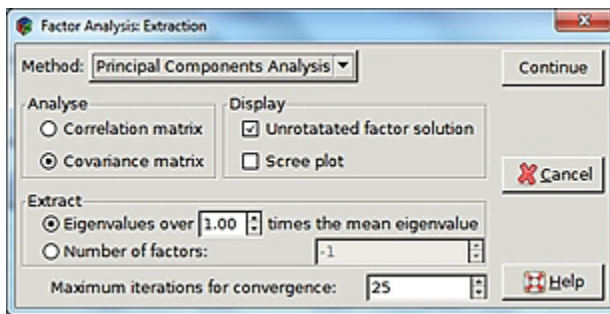
Figure 4.100: *Factor Analysis Extraction* dialogue box

Method The emphMethod button on the emphExtraction dialogue box enables you to choose the method of factor extraction. Options include emphPrincipal Components Analysis and emphPrincipal Axis Factoring.

> **Principal Components Analysis** Use when the correlation matrix is singular. This function assembles uncorrelated linear combinations of the observed

variables such that each successive component accounts for a progressively smaller portion of the variance.

Principal Axis Factoring An iterative procedure that extracts and replaces factors from the current correlation matrix with new estimates repeatedly, until the convergence criterion for extraction is satisfied.

Analyze Choose either a correlation or covariance matrix.

Display Display the unrotated factor solution and Scree plot.

Unrotated Factor Solution Display the unrotated factor loadings, communalities, and the solution eigenvalues.

Scree Plot A plot of the variance for each factor.
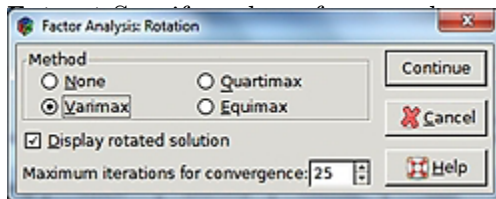
eigenvalues exceed a specified factor above the mean of factors to keep.

Specify the maximum number of steps the ition.

Figure 4.101: *Factor Analysis Rotation* dialogue box

4.5.7.22 Rotation Dialogue Box

Method Choose the type of factor rotation method. Options include none, varimax, quartimax, and equimax.

Varimax An orthogonal rotation of the factor axes which has the effect of differentiating the original variables by the extracted factor. This solution

provides results that make it easier to identify each variable with a single factor.

Quartimax An orthogonal rotation that minimizes the number of factors needed to explain each variable.
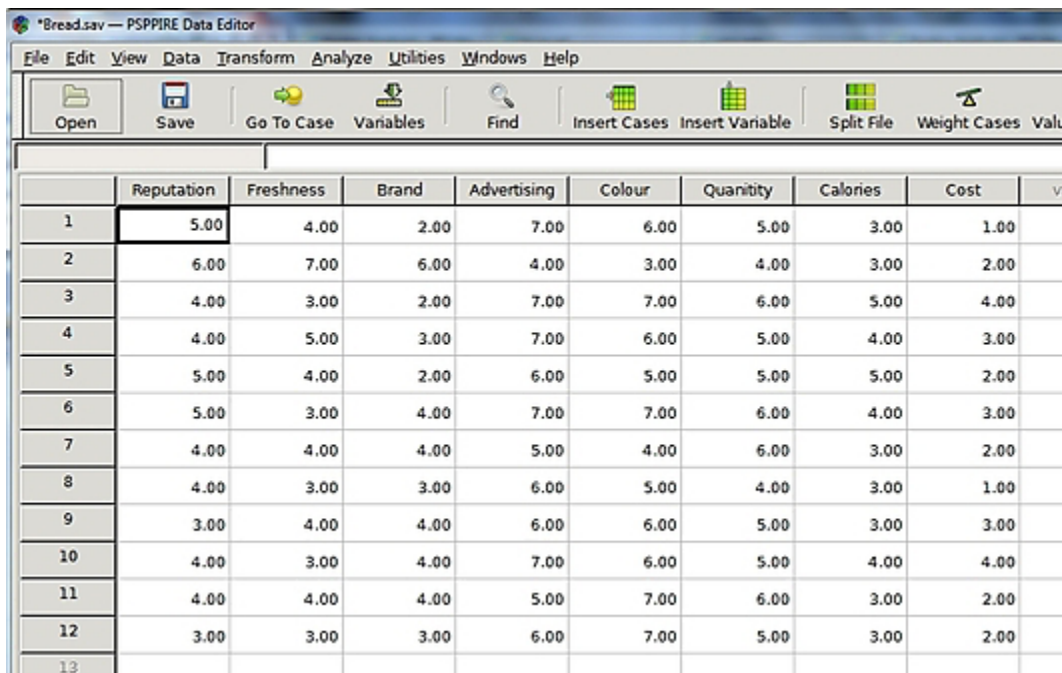
Equimax A combination of the varimax and quartimax rotations.

Display Rotated Solution Display the solution in this way if you have selected a rotation method.

Maximum Iterations for Convergence Choose the maximum number of steps the algorithm can take to reach convergence. The default value is twenty-five.

Example

Perform factor analysis to determine the leading factors that drive people to purchase specific brands of bread with the data (See Figure 4.102).

	Reputation	Freshness	Brand	Advertising	Colour	Quanitity	Calories	Cost
1	5.00	4.00	2.00	7.00	6.00	5.00	3.00	1.00
2	6.00	7.00	6.00	4.00	3.00	4.00	3.00	2.00
3	4.00	3.00	2.00	7.00	7.00	6.00	5.00	4.00
4	4.00	5.00	3.00	7.00	6.00	5.00	4.00	3.00
5	5.00	4.00	2.00	6.00	5.00	5.00	5.00	2.00
6	5.00	3.00	4.00	7.00	7.00	6.00	4.00	3.00
7	4.00	4.00	4.00	5.00	4.00	6.00	3.00	2.00
8	4.00	3.00	3.00	6.00	5.00	4.00	3.00	1.00
9	3.00	4.00	4.00	6.00	6.00	5.00	3.00	3.00
10	4.00	3.00	4.00	7.00	6.00	5.00	4.00	4.00
11	4.00	4.00	4.00	5.00	7.00	6.00	3.00	2.00
12	3.00	3.00	3.00	6.00	7.00	5.00	3.00	2.00
13								

Figure 4.102: Survey data describing influential factors in purchasing.

Enter the following consumer data (See Figure 4.102), or open file Bread.sav.
Select Analyze → Factor Analysis to bring up *Factor Analysis* dialogue box.

Click the first variable, *Reputation*, and then press the right arrow button to move it into
the *Variables* dialogue box.
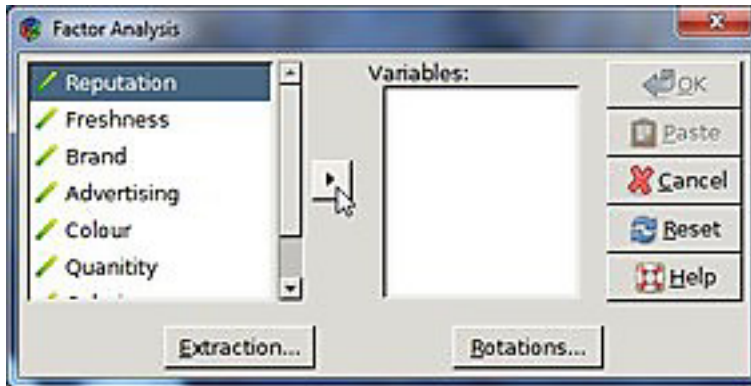
Figure 4.103: Pick variables for *Factor Analysis*.

Repeat with all the other variables until no more remain on the left side.
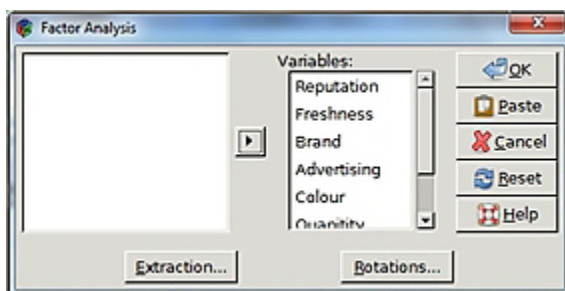
Figure 4.104: All variables are selected for *Factor Analysis*.

Click the *Extraction* button.

Select the *Principal Components Analysis* method.

Select the *Correlation matrix* option under *Analyse*.

Select *Unrotated factor solution* and *Scree plot* under *Display*.

Click *Continue* to close this dialogue box.

Select the *Rotations* button in the *Factor Analysis* dialogue box.

Select the *Varimax* option under *Method*.

Select *Display rotated solution*.

Click *Continue* to close the dialogue box.

Click the OK button to display the results in the output viewer.

	Initial	Extraction
Reputation	1.00	.78
Freshness	1.00	.80
Brand	1.00	.90
Advertising	1.00	.79
Colour	1.00	.79
Quantity	1.00	.56
Calories	1.00	.91
Cost	1.00	.87

	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared Loadings		
Component	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	3.76	46.96	46.96	3.76	46.96	46.96	2.09	26.16	26.16
2	1.44	17.96	64.92	1.44	17.96	64.92	1.90	23.79	49.95
3	1.20	14.97	79.89	1.20	14.97	79.89	2.40	29.95	79.89
4	.71	8.82	88.71						
5	.42	5.23	93.94						
6	.28	3.56	97.49						
7	.18	2.24	99.73						
8	.02	.27	100.00						

Scree Plot
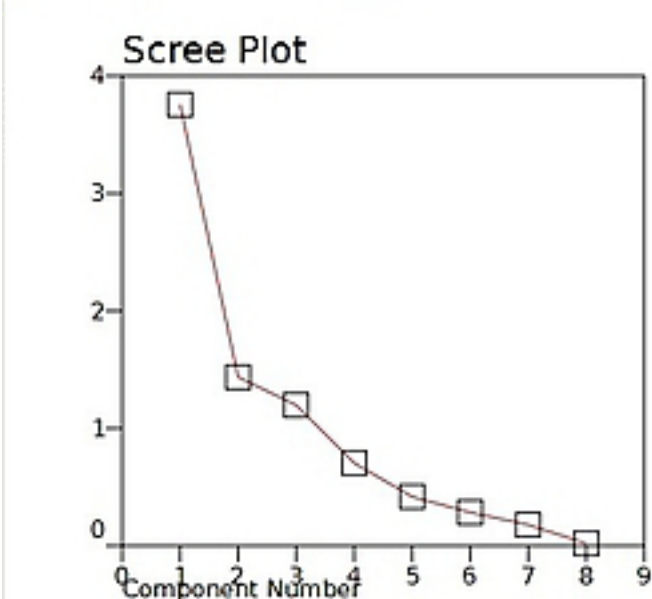
	Component
	1 2 3

****************************two output figs in one pic above

4.5.7.24 Relibility

Estimate the internal consistency reliability. Options include Split-half reliability and Chronbach's Alpha.
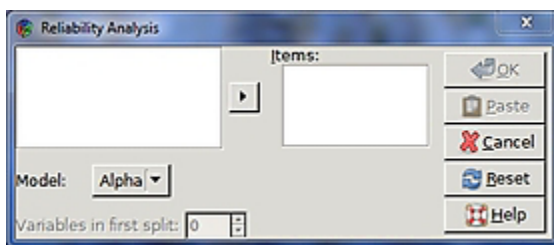
Figure 4.106: *Reliability Analysis* dialogue box

4.5.7.25 Split-half Reliability

An estimation based on the correlation of two equivalent forms of the scale. This is a three-step process: 1) Randomly divide all items that measure the same construct into two sets, 2) administer the whole instrument and divide the total score for each item in half, and 3) estimate the split-half reliability by calculating the correlation of these two sets.

4.5.7.26 Chronbach's Alpha

This estimation is equivalent to the average of all possible split-half estimates, and is an indication of the correlation among the variables.

Example

Examine the internal reliability of the following pass-fail test data (See Figure 4.107), where zero represents fail and one pass.

Figure 4.107: Fail-pass test data

Enter the test data (See Figure 4.107) into a new data file, or open the file PassFail.sav.

Select Analyze → Reliability.

Click the first variable and select the right arrow to move it into the epphItems textbox.

Figure 4.108: Pick variables for *Reliability Analysis*

Repeat with the other variables until they are all listed in the *Items* text box.
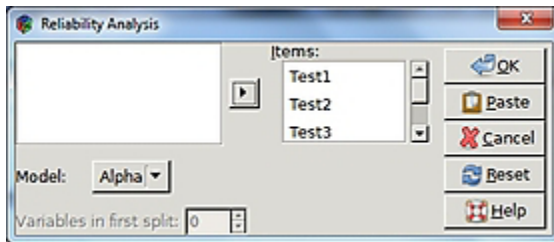
Figure 4.109: All variables are chosen for *Reliability Analysis*.

Click OK to display the results. In this instance, *Cronbachs Alpha* is 0.61 indicating the internal consistency is questionable.
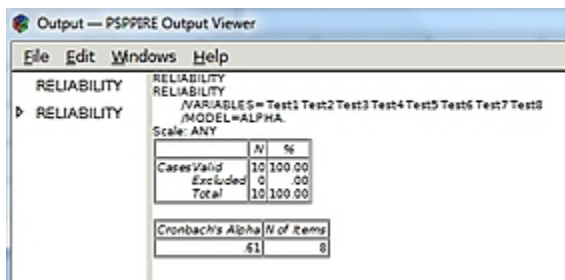
Figure 4.110: *Reliability Analysis* output

4.5.7.28 Non-Parametric Statistics

Provides Binomial statistics and Chi-Square for an arbitrary distribution of data. Tests the differences between the observed and expected distribution of a sample, and can also test the difference between two or more samples.

4.5.7.29 Binomial

Compare the observed frequencies of dichotomous data to the expected values from the binomial distribution. The primary purpose of this test is to observe if the number of outcomes for each variable differs, e.g., you can verify a fair coin by comparing the frequency of heads and tails in several coin tosses.

Example

The following list of *HotCold* and *Range* data (See Table 4.4) indicates the temperatures at which customers ordered their gourmet drinks on a midsummers day; available menu options include ice-cold, luke-warm, warm, or steaming-hot. The first column, emphHotColdis a dichotomous variable with a value of zero for ice-cold drinks and one

otherwise. The next column, emphRange varies from 0 to 3 depending upon temperature range of the drink, where 0=cold, 1=luke-warm, 2=warm, and 3= hot.

HotCold	Range
0	0
0	0
1	2
1	2
0	0
1	1
1	1
1	1
1	2
1	1
1	3
1	3
1	2
0	0
0	0
0	0
1	1
0	0
0	0
1	2
1	2
0	0
1	3
0	0

Table 4.4: Gourmet Drink Temperatures

Start by examining the data in the first column; *HotCold*.

Copy the gourmet drink data (See Table 4.4) into a data file, or open
coffeeTemperature.sav.

Select Analyze → Non-Parametric Statistics → Binomial.

Select *HotCold* and click the right arrow button to place it in the *Test Variable List*
textbox.

Verify *Test proportion* has the value 0.5.

Verify *Get from data* is selected under *Define Dichotomy*.
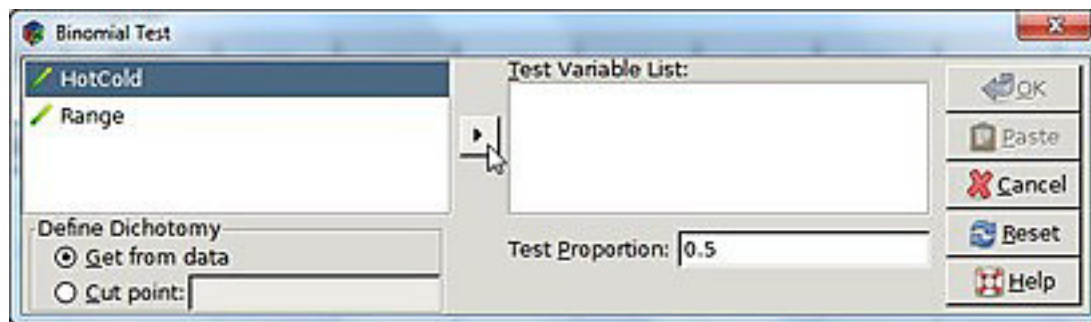
Figure 4.111: *Binomial* dialogue box

Click OK to display the results in the output viewer.

Figure 4.112: *Binomial Test* output

4.5.7.31 Chi-Square

Used to check if distributions of categorical variables differ, where categorical variables are
used to label categorical data, e.g. assign a value of 1 for values in range a to c, value of 2
for values in range d to f, and so on.

Example

Enter the gourmet drink data (See Table 4.4) from the previous section (See [Binomial Example], page 82) into a new data file.

Select Analyze → Non-Parametric Statistics → Chi-Square.

Select the *Range* variable, and then click the right arrow to place it into the *Test Variable List* textbox.
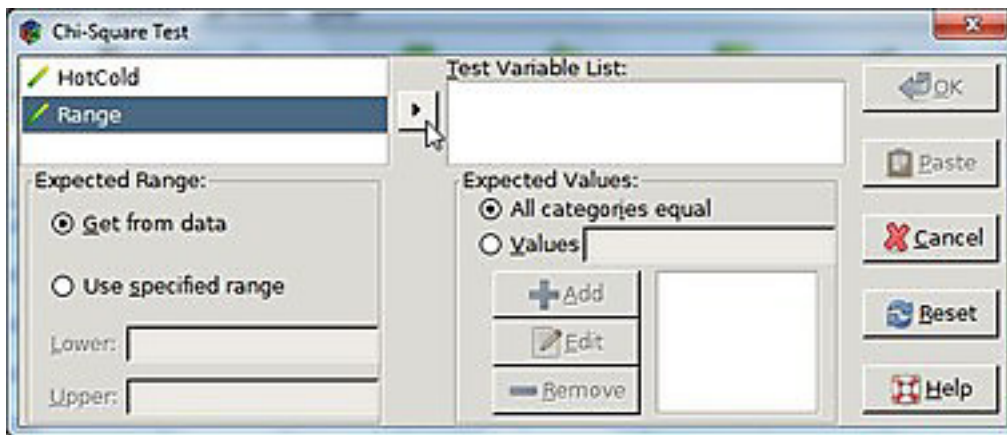
Figure 4.113: *Chi-Square Test* dialogue box

Verify that *All categories equal* is selected under *Expected Values.*

Verify that the *Get from data* option under *Expected Range* is selected.

Click OK to display the results in the output viewer.

Figure 4.114: *Chi-Square Test* output

4.5.7.33 ROC Curve

Use to analyze binary classification systems. Examples of binary classification schemes include passing or failing newly made items during an inspection, and labeling patients to be either with or without a medical problem. The Roc Curve function returns an ROC

curve with the option to display a diagonal reference line, standard error and confidence interval, as well as all coordinate points.
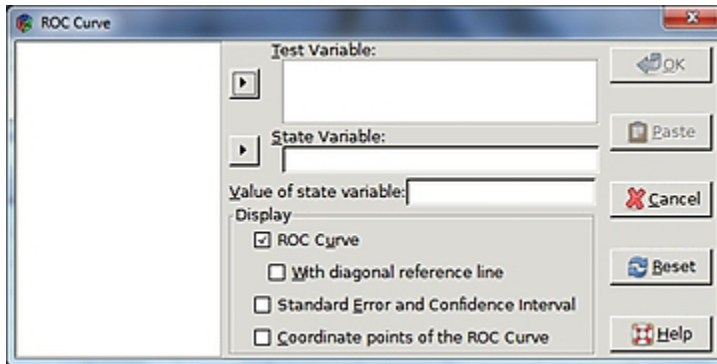
Figure 4.115: *ROC Curve* dialogue box

4.5.7.34 Test Variable

Continuous numerical data holding the test results.

4.5.7.35 State Variable

Indicates which category is positive.

4.5.7.36 Display

Gives options for the output viewer including:

ROC curve with or without a diagonal reference line

Display standard error and confidence interval

Display coordinate points of the ROC curve
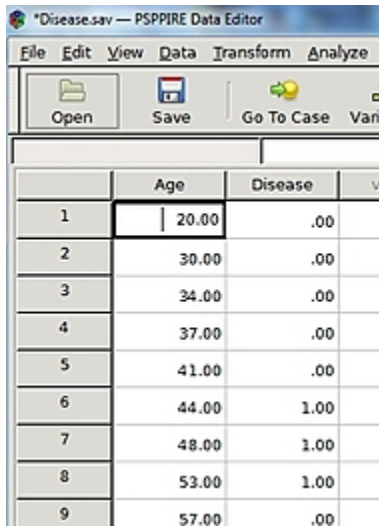
Display ROC Curve Produce a plot of one classification rate from a binary classification scheme versus the other as the criterion changes, e.g., you can plot the true positive rate versus the false positive rate for a disease as the threshold value between the two classifications varies.

Display Diagonal Reference Line Include a diagonal reference line in the plot. An ROC curve that coincides with this line indicates the distribution of each classification group is similar.

Display Standard Error and Confidence Interval Show the standard error, confidence interval, and area in a separate table in the output viewer. The area ranges in value from 0 to 1, where 1 indicates the distribution of each classification group does not overlap and 0.5 means both distributions are the same.

Example

Make an ROC curve of the following medical test data describing the dependence of a disease with age. The *age* variable holds the patients ages and the corresponding *disease* column is either 0 indicating a negative diagnosis or 1 otherwise.

Figure 4.116: Medical test data

Open the file Disease.sav.

Select Analyze → ROC Curve.

Select *Age* from the list of variables on the left side of the dialogue box, and click the upper right arrow button to move it under *Test Variable*.

Select *Disease* and click the lower right arrow button to move it into the *State Variable* box.

Select all the check boxes under *Display*.

Enter the number 1 in the *Value of state variable* box.
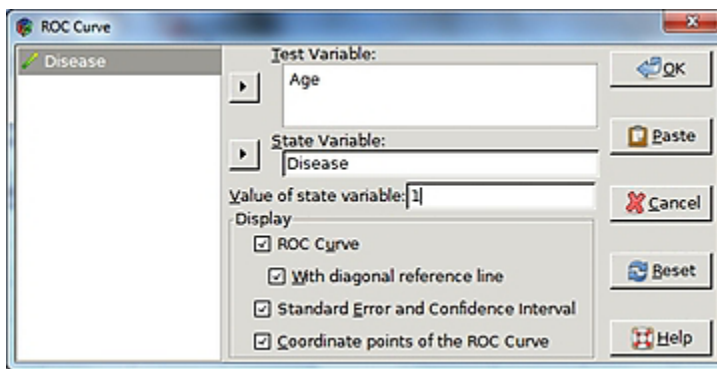
Figure 4.117: *ROC Curve* dialogue box

Click OK to display the ROC curve and accompanying tables in the output viewer.
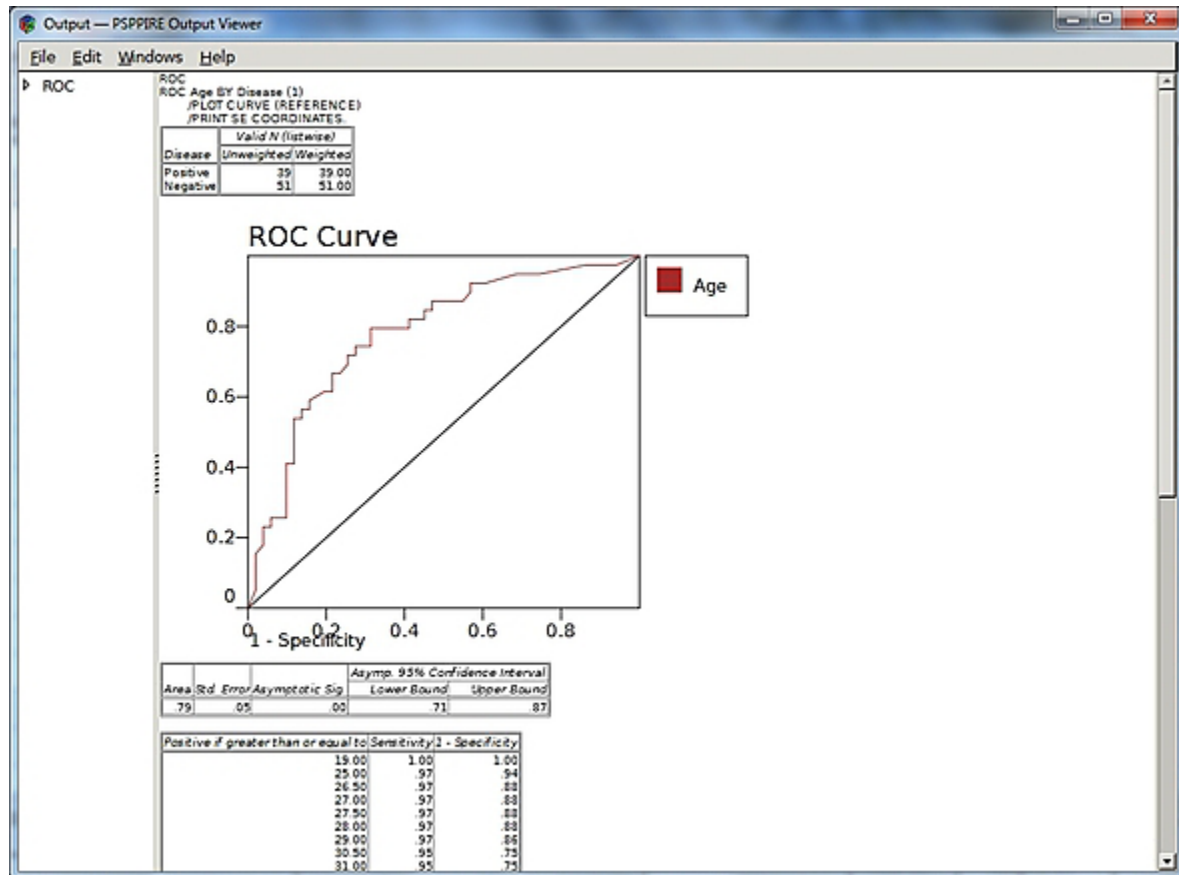
Figure 4.118: *ROC Curve* output

4.5.8 Utilities

4.5.8.1 Variables

View variable properties and/or go-to it in the data editor window. Properties include variable name and label, Data format, user-missing values, measurement level, and value labels. For a definition of each variable property, (See Section 4.2.3.1 [Edit Variable Attributes], page 11).

Example

open hotel.sav.
Select Utilities → Variables to bring up the *Variables* dialogue box.

Select a variable from the list in the left column to view its properties on the right side of the dialogue box.
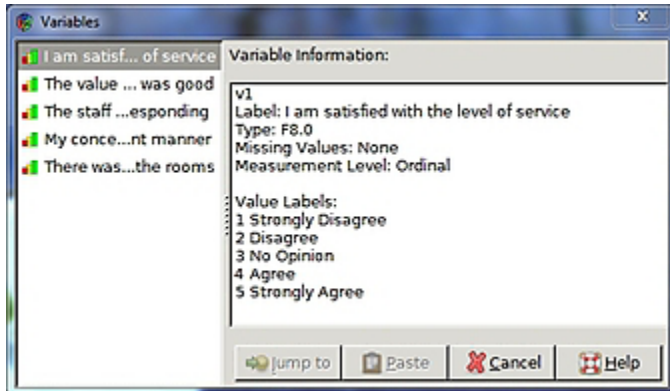
Figure 4.119: View variable properties in *Variables* dialogue box.

4.5.8.3 Data File Comments

This feature enables you to add comments to a data file.
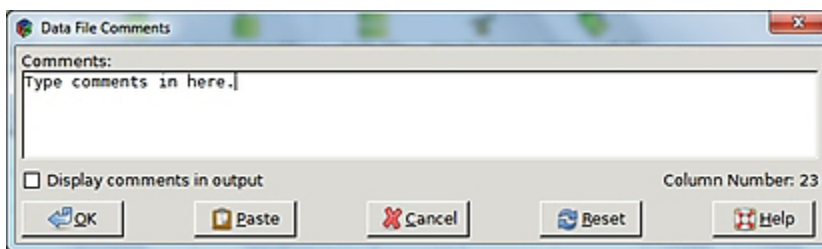
Figure 4.120: *Comments* dialogue box

4.5.8.4 Display Comments in Output

Display comments in the output viewer.

4.5.8.5 Reset

Clears the current comment.

4.5.9 Windows

4.5.9.1 Minimize all Windows

Collapse all windows. Click the PSPP icon on the tool bar to open them again.

4.5.9.2 Split

Split the current data editor window (See Figure 4.121).
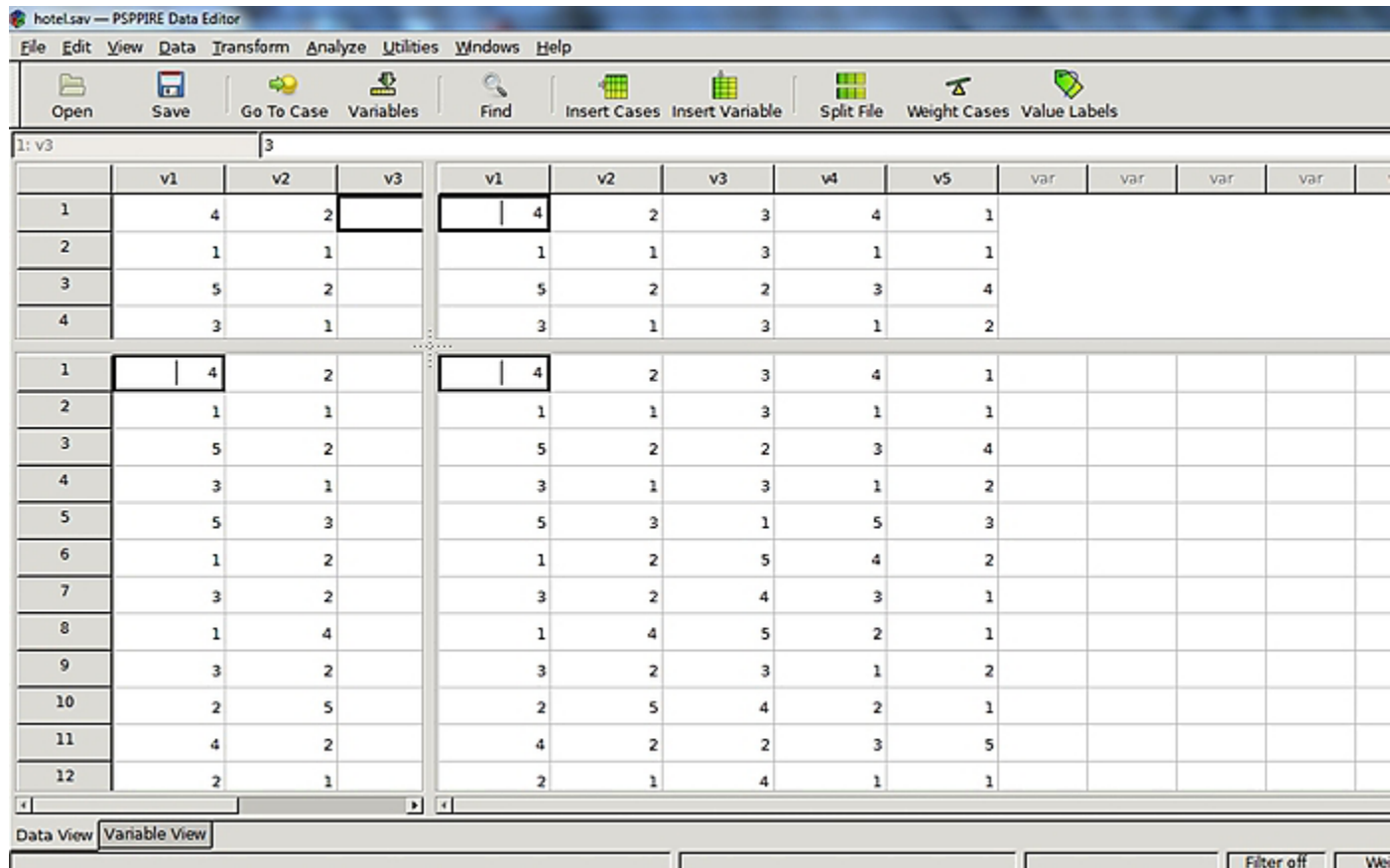
Figure 4.121: Split data-editor window

4.5.10 Toolbar

PSPP provides a tool bar across the top with short cuts to more commonly used menu commands.
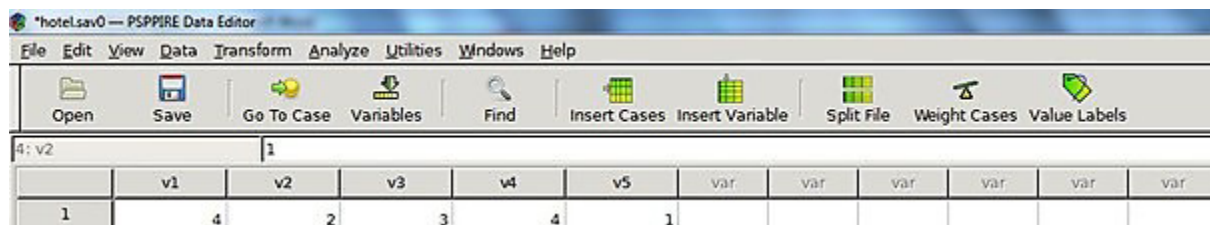
Figure 4.122: Data editor toolbar

5 The PSPP language

> **Please note:** PSPP is not even close to completion. Only a few statistical procedures are implemented. PSPP is a work in progress.

This chapter discusses elements common to many PSPP commands. Later chapters will describe individual commands in detail.

5.1 Tokens

PSPP divides most syntax file lines into series of short chunks called *tokens*. Tokens are then grouped to form commands, each of which tells PSPP to take some action—read in data, write out data, perform a statistical procedure, etc. Each type of token is described below.

Identifiers   Identifiers are names that typically specify variables, commands, or subcommands. The first character in an identifier must be a letter, '`#`', or '`@`'. The remaining characters in the identifier must be letters, digits, or one of the following special characters:

> .   _ $ # @

Identifiers may be any length, but only the first 64 bytes are significant. Identifiers are not case-sensitive: `foobar`, `Foobar`, `FooBar`, `FOOBAR`, and `FoObaR` are different representations of the same identifier.

Some identifiers are reserved. Reserved identifiers may not be used in any context besides those explicitly described in this manual. The reserved identifiers are:

> ALL AND BY EQ GE GT LE LT NE NOT OR TO WITH

Keywords   Keywords are a subclass of identifiers that form a fixed part of command syntax. For example, command and subcommand names are keywords. Keywords may be abbreviated to their first 3 characters if this abbreviation is unambiguous. (Unique abbreviations of 3 or more characters are also accepted: '`FRE`', '`FREQ`', and '`FREQUENCIES`' are equivalent when the last is a keyword.)

Reserved identifiers are always used as keywords. Other identifiers may be used both as keywords and as user-defined identifiers, such as variable names.

Numbers   Numbers are expressed in decimal. A decimal point is optional. Numbers may be expressed in scientific notation by adding '`e`' and a base-10 exponent, so that '`1.234e3`' has the value 1234. Here are some more examples of valid numbers:

> -5 3.14159265359 1e100 -.707 8945.

Negative numbers are expressed with a '`-`' prefix. However, in situations where a literal '`-`' token is expected, what appears to be a negative number is treated as '`-`' followed by a positive number.

No white space is allowed within a number token, except for horizontal white space between '`-`' and the rest of the number.

The last example above, '`8945.`' will be interpreted as two tokens, '`8945`' and '`.`', if it is the last token on a line. See Section 5.2 [Forming commands of tokens], page 93.

Strings   Strings are literal sequences of characters enclosed in pairs of single quotes ('')
or double quotes ('"'). To include the character used for quoting in the string,
double it, e.g. `''it''s an apostrophe''`. White space and case of letters are
significant inside strings.

Strings can be concatenated using '+', so that '`"a" + 'b' + 'c'`' is equivalent
to `''abc''`. Concatenation is useful for splitting a single string across multiple
source lines. The maximum length of a string, after concatenation, is 255
characters.

Strings may also be expressed as hexadecimal, octal, or binary character values
by prefixing the initial quote character by 'X', 'O', or 'B' or their lowercase
equivalents. Each pair, triplet, or octet of characters, according to the radix,
is transformed into a single character with the given value. If there is an
incomplete group of characters, the missing final digits are assumed to be 'O'.
These forms of strings are nonportable because numeric values are associated
with different characters by different operating systems. Therefore, their use
should be confined to syntax files that will not be widely distributed.

The character with value 00 is reserved for internal use by PSPP. Its use in
strings causes an error and replacement by a space character.

Punctuators and Operators

These tokens are the punctuators and operators:

    , / = ( ) + - * / ** < <= <> > >= ~= & | .

Most of these appear within the syntax of commands, but the period ('.')
punctuator is used only at the end of a command. It is a punctuator only as
the last character on a line (except white space). When it is the last non-space
character on a line, a period is not treated as part of another token, even if it
would otherwise be part of, e.g., an identifier or a floating-point number.

Actually, the character that ends a command can be changed with SET's END-
CMD subcommand (see Section 14.17 [SET], page 193), but we do not recom-
mend doing so. Throughout the remainder of this manual we will assume that
the default setting is in effect.

5.2 Forming commands of tokens

Most PSPP commands share a common structure. A command begins with a command
name, such as FREQUENCIES, DATA LIST, or N OF CASES. The command name may
be abbreviated to its first word, and each word in the command name may be abbreviated
to its first three or more characters, where these abbreviations are unambiguous.

The command name may be followed by one or more *subcommands*. Each subcommand
begins with a subcommand name, which may be abbreviated to its first three letters. Some
subcommands accept a series of one or more specifications, which follow the subcommand
name, optionally separated from it by an equals sign ('='). Specifications may be separated
from each other by commas or spaces. Each subcommand must be separated from the next
(if any) by a forward slash ('/').

There are multiple ways to mark the end of a command. The most common way is
to end the last line of the command with a period ('.') as described in the previous sec-
tion (see Section 5.1 [Tokens], page 92). A blank line, or one that consists only of white

space or comments, also ends a command by default, although you can use the NULLINE subcommand of SET to disable this feature (see Section 14.17 [SET], page 193).

5.3 Variants of syntax.

There are two variants of command syntax, *viz*: *batch* mode and *interactive* mode. Batch mode is the default when reading commands from a file. Interactive mode is the default when commands are typed at a prompt by a user. Certain commands, such as INSERT (see Section 14.15 [INSERT], page 192), may explicitly change the syntax mode.

In batch mode, any line that contains a non-space character in the leftmost column begins a new command. Thus, each command consists of a flush-left line followed by any number of lines indented from the left margin. In this mode, a plus or minus sign ('+', '−') as the first character in a line is ignored and causes that line to begin a new command, which allows for visual indentation of a command without that command being considered part of the previous command. The period terminating the end of a command is optional but recommended.

In interactive mode, each command must either be terminated with a period, or an empty line must follow the command. The use of ('+' and '−' as continuation characters is not permitted.

5.4 Types of Commands

Commands in PSPP are divided roughly into six categories:

Utility commands
> Set or display various global options that affect PSPP operations. May appear anywhere in a syntax file. See Chapter 14 [Utility commands], page 190.

File definition commands
> Give instructions for reading data from text files or from special binary "system files". Most of these commands replace any previous data or variables with new data or variables. At least one file definition command must appear before the first command in any of the categories below. See Chapter 7 [Data Input and Output], page 128.

Input program commands
> Though rarely used, these provide tools for reading data files in arbitrary textual or binary formats. See Section 7.7 [INPUT PROGRAM], page 135.

Transformations
> Perform operations on data and write data to output files. Transformations are not carried out until a procedure is executed.

Restricted transformations
> Transformations that cannot appear in certain contexts. See Section 5.5 [Order of Commands], page 95, for details.

Procedures
> Analyze data, writing results of analyses to the listing file. Cause transformations specified earlier in the file to be performed. In a more general sense, a *procedure* is any command that causes the active file (the data) to be read.

5.5 Order of Commands

PSPP does not place many restrictions on ordering of commands. The main restriction is that variables must be defined before they are otherwise referenced. This section describes the details of command ordering, but most users will have no need to refer to them.

PSPP possesses five internal states, called initial, INPUT PROGRAM, FILE TYPE, transformation, and procedure states. (Please note the distinction between the INPUT PROGRAM and FILE TYPE *commands* and the INPUT PROGRAM and FILE TYPE *states*.)

PSPP starts in the initial state. Each successful completion of a command may cause a state transition. Each type of command has its own rules for state transitions:

Utility commands
- Valid in any state.
- Do not cause state transitions. Exception: when N OF CASES is executed in the procedure state, it causes a transition to the transformation state.

DATA LIST
- Valid in any state.
- When executed in the initial or procedure state, causes a transition to the transformation state.
- Clears the active file if executed in the procedure or transformation state.

INPUT PROGRAM
- Invalid in INPUT PROGRAM and FILE TYPE states.
- Causes a transition to the INPUT PROGRAM state.
- Clears the active file.

FILE TYPE
- Invalid in INPUT PROGRAM and FILE TYPE states.
- Causes a transition to the FILE TYPE state.
- Clears the active file.

Other file definition commands
- Invalid in INPUT PROGRAM and FILE TYPE states.
- Cause a transition to the transformation state.
- Clear the active file, except for ADD FILES, MATCH FILES, and UPDATE.

Transformations
- Invalid in initial and FILE TYPE states.
- Cause a transition to the transformation state.

Restricted transformations
- Invalid in initial, INPUT PROGRAM, and FILE TYPE states.
- Cause a transition to the transformation state.

Procedures
- Invalid in initial, INPUT PROGRAM, and FILE TYPE states.
- Cause a transition to the procedure state.

5.6 Handling missing observations

PSPP includes special support for unknown numeric data values. Missing observations are assigned a special value, called the *system-missing value*. This "value" actually indicates the absence of a value; it means that the actual value is unknown. Procedures automatically exclude from analyses those observations or cases that have missing values. Details of missing value exclusion depend on the procedure and can often be controlled by the user; refer to descriptions of individual procedures for details.

The system-missing value exists only for numeric variables. String variables always have a defined value, even if it is only a string of spaces.

Variables, whether numeric or string, can have designated *user-missing values*. Every user-missing value is an actual value for that variable. However, most of the time user-missing values are treated in the same way as the system-missing value. String variables that are wider than a certain width, usually 8 characters (depending on computer architecture), cannot have user-missing values.

For more information on missing values, see the following sections: Section 5.7 [Variables], page 96, Section 9.7 [MISSING VALUES], page 157, Chapter 6 [Expressions], page 110. See also the documentation on individual procedures for information on how they handle missing values.

5.7 Variables

Variables are the basic unit of data storage in PSPP. All the variables in a file taken together, apart from any associated data, are said to form a *dictionary*. Some details of variables are described in the sections below.

5.7.1 Attributes of Variables

Each variable has a number of attributes, including:

Name   An identifier, up to 64 bytes long. Each variable must have a different name. See Section 5.1 [Tokens], page 92.

Some system variable names begin with '$', but user-defined variables' names may not begin with '$'.

The final character in a variable name should not be '.', because such an identifier will be misinterpreted when it is the final token on a line: `FOO.` will be divided into two separate tokens, 'FOO' and '.', indicating end-of-command. See Section 5.1 [Tokens], page 92.

The final character in a variable name should not be '_', because some such identifiers are used for special purposes by PSPP procedures.

As with all PSPP identifiers, variable names are not case-sensitive. PSPP capitalizes variable names on output the same way they were capitalized at their point of definition in the input.

Type   Numeric or string.

Width   (string variables only) String variables with a width of 8 characters or fewer are called *short string variables*. Short string variables can be used in many

procedures where *long string variables* (those with widths greater than 8) are not allowed.

Certain systems may consider strings longer than 8 characters to be short strings. Eight characters represents a minimum figure for the maximum length of a short string.

Position   Variables in the dictionary are arranged in a specific order. DISPLAY can be used to show this order: see Section 9.3 [DISPLAY], page 155.

Initialization

Either reinitialized to 0 or spaces for each case, or left at its existing value. See Section 9.6 [LEAVE], page 156.

Missing values

Optionally, up to three values, or a range of values, or a specific value plus a range, can be specified as *user-missing values*. There is also a *system-missing value* that is assigned to an observation when there is no other obvious value for that observation. Observations with missing values are automatically excluded from analyses. User-missing values are actual data values, while the system-missing value is not a value at all. See Section 5.6 [Missing Observations], page 96.

Variable label

A string that describes the variable. See Section 9.14 [VARIABLE LABELS], page 159.

Value label

Optionally, these associate each possible value of the variable with a string. See Section 9.12 [VALUE LABELS], page 159.

Print format

Display width, format, and (for numeric variables) number of decimal places. This attribute does not affect how data are stored, just how they are displayed. Example: a width of 8, with 2 decimal places. See Section 5.7.4 [Input and Output Formats], page 98.

Write format

Similar to print format, but used by the WRITE command (see Section 7.15 [WRITE], page 142).

5.7.2 Variables Automatically Defined by PSPP

There are seven system variables. These are not like ordinary variables because system variables are not always stored. They can be used only in expressions. These system variables, whose values and output formats cannot be modified, are described below.

$CASENUM   Case number of the case at the moment. This changes as cases are shuffled around.

$DATE   Date the PSPP process was started, in format A9, following the pattern DD MMM YY.

$JDATE   Number of days between 15 Oct 1582 and the time the PSPP process was started.

$LENGTH   Page length, in lines, in format F11.

$SYSMIS   System missing value, in format F1.

$TIME   Number of seconds between midnight 14 Oct 1582 and the time the active file
           was read, in format F20.

$WIDTH   Page width, in characters, in format F3.

5.7.3 Lists of variable names

To refer to a set of variables, list their names one after another. Optionally, their names
may be separated by commas. To include a range of variables from the dictionary in the
list, write the name of the first and last variable in the range, separated by TO. For instance,
if the dictionary contains six variables with the names ID, X1, X2, GOAL, MET, and NEXTGOAL,
in that order, then X2 TO MET would include variables X2, GOAL, and MET.

Commands that define variables, such as DATA LIST, give TO an alternate meaning.
With these commands, TO define sequences of variables whose names end in consecutive
integers. The syntax is two identifiers that begin with the same root and end with numbers,
separated by TO. The syntax X1 TO X5 defines 5 variables, named X1, X2, X3, X4, and X5. The
syntax ITEM0008 TO ITEM0013 defines 6 variables, named ITEM0008, ITEM0009, ITEM0010,
ITEM0011, ITEM0012, and ITEM00013. The syntaxes QUES001 TO QUES9 and QUES6 TO QUES3
are invalid.

After a set of variables has been defined with DATA LIST or another command with
this method, the same set can be referenced on later commands using the same syntax.

5.7.4 Input and Output Formats

An *input format* describes how to interpret the contents of an input field as a number or
a string. It might specify that the field contains an ordinary decimal number, a time or
date, a number in binary or hexadecimal notation, or one of several other notations. Input
formats are used by commands such as DATA LIST that read data or syntax files into the
PSPP active file.

Every input format corresponds to a default *output format* that specifies the formatting
used when the value is output later. It is always possible to explicitly specify an output
format that resembles the input format. Usually, this is the default, but in cases where the
input format is unfriendly to human readability, such as binary or hexadecimal formats, the
default output format is an easier-to-read decimal format.

Every variable has two output formats, called its *print format* and *write format*. Print
formats are used in most output contexts; write formats are used only by WRITE (see
Section 7.15 [WRITE], page 142). Newly created variables have identical print and write
formats, and FORMATS, the most commonly used command for changing formats (see
Section 9.5 [FORMATS], page 156), sets both of them to the same value as well. Thus,
most of the time, the distinction between print and write formats is unimportant.

Input and output formats are specified to PSPP with a *format specification* of the form
TYPEw or TYPEw.d, where TYPE is one of the format types described later, w is a field width
measured in columns, and d is an optional number of decimal places. If d is omitted, a
value of 0 is assumed. Some formats do not allow a nonzero d to be specified.

The following sections describe the input and output formats supported by PSPP.

5.7.4.1 Basic Numeric Formats

The basic numeric formats are used for input and output of real numbers in standard or scientific notation. The following table shows an example of how each format displays positive and negative numbers with the default decimal point setting:

Format	3141.59	−3141.59
F8.2	3141.59	−3141.59
COMMA9.2	3,141.59	−3,141.59
DOT9.2	3.141,59	−3.141,59
DOLLAR10.2	$3,141.59	−$3,141.59
PCT9.2	3141.59%	−3141.59%
E8.1	3.1E+003	−3.1E+003

On output, numbers in F format are expressed in standard decimal notation with the requested number of decimal places. The other formats output some variation on this style:

- Numbers in COMMA format are additionally grouped every three digits by inserting a grouping character. The grouping character is ordinarily a comma, but it can be changed to a period (see [SET DECIMAL], page 195).

- DOT format is like COMMA format, but it interchanges the role of the decimal point and grouping characters. That is, the current grouping character is used as a decimal point and vice versa.

- DOLLAR format is like COMMA format, but it prefixes the number with '$'.

- PCT format is like F format, but adds '%' after the number.

- The E format always produces output in scientific notation.

On input, the basic numeric formats accept positive and numbers in standard decimal notation or scientific notation. Leading and trailing spaces are allowed. An empty or all-spaces field, or one that contains only a single period, is treated as the system missing value.

In scientific notation, the exponent may be introduced by a sign ('+' or '−'), or by one of the letters 'e' or 'd' (in uppercase or lowercase), or by a letter followed by a sign. A single space may follow the letter or the sign or both.

On fixed-format DATA LIST (see Section 7.3.1 [DATA LIST FIXED], page 129) and in a few other contexts, decimals are implied when the field does not contain a decimal point. In F6.5 format, for example, the field 314159 is taken as the value 3.14159 with implied decimals. Decimals are never implied if an explicit decimal point is present or if scientific notation is used.

E and F formats accept the basic syntax already described. The other formats allow some additional variations:

- COMMA, DOLLAR, and DOT formats ignore grouping characters within the integer part of the input field. The identity of the grouping character depends on the format.

- DOLLAR format allows a dollar sign to precede the number. In a negative number, the dollar sign may precede or follow the minus sign.

- PCT format allows a percent sign to follow the number.

All of the basic number formats have a maximum field width of 40 and accept no more than 16 decimal places, on both input and output. Some additional restrictions apply:

- As input formats, the basic numeric formats allow no more decimal places than the field width. As output formats, the field width must be greater than the number of decimal places; that is, large enough to allow for a decimal point and the number of requested decimal places. DOLLAR and PCT formats must allow an additional column for '$' or '%'.

- The default output format for a given input format increases the field width enough to make room for optional input characters. If an input format calls for decimal places, the width is increased by 1 to make room for an implied decimal point. COMMA, DOT, and DOLLAR formats also increase the output width to make room for grouping characters. DOLLAR and PCT further increase the output field width by 1 to make room for '$' or '%'. The increased output width is capped at 40, the maximum field width.

- The E format is exceptional. For output, E format has a minimum width of 7 plus the number of decimal places. The default output format for an E input format is an E format with at least 3 decimal places and thus a minimum width of 10.

More details of basic numeric output formatting are given below:

- Output rounds to nearest, with ties rounded away from zero. Thus, 2.5 is output as 3 in F1.0 format, and -1.125 as -1.13 in F5.1 format.

- The system-missing value is output as a period in a field of spaces, placed in the decimal point's position, or in the rightmost column if no decimal places are requested. A period is used even if the decimal point character is a comma.

- A number that does not fill its field is right-justified within the field.

- A number is too large for its field causes decimal places to be dropped to make room. If dropping decimals does not make enough room, scientific notation is used if the field is wide enough. If a number does not fit in the field, even in scientific notation, the overflow is indicated by filling the field with asterisks ('*').

- COMMA, DOT, and DOLLAR formats insert grouping characters only if space is available for all of them. Grouping characters are never inserted when all decimal places must be dropped. Thus, 1234.56 in COMMA5.2 format is output as ' 1235' without a comma, even though there is room for one, because all decimal places were dropped.

- DOLLAR or PCT format drop the '$' or '%' only if the number would not fit at all without it. Scientific notation with '$' or '%' is preferred to ordinary decimal notation without it.

- Except in scientific notation, a decimal point is included only when it is followed by a digit. If the integer part of the number being output is 0, and a decimal point is included, then the zero before the decimal point is dropped.

  In scientific notation, the number always includes a decimal point, even if it is not followed by a digit.

- A negative number includes a minus sign only in the presence of a nonzero digit: -0.01 is output as '-.01' in F4.2 format but as '   .0' in F4.1 format. Thus, a "negative zero" never includes a minus sign.

- In negative numbers output in DOLLAR format, the dollar sign follows the negative sign. Thus, -9.99 in DOLLAR6.2 format is output as `-$9.99`.

- In scientific notation, the exponent is output as 'E' followed by '+' or '-' and exactly three digits. Numbers with magnitude less than 10**-999 or larger than 10**999 are not supported by most computers, but if they are supported then their output is considered to overflow the field and will be output as asterisks.

- On most computers, no more than 15 decimal digits are significant in output, even if more are printed. In any case, output precision cannot be any higher than input precision; few data sets are accurate to 15 digits of precision. Unavoidable loss of precision in intermediate calculations may also reduce precision of output.

- Special values such as infinities and "not a number" values are usually converted to the system-missing value before printing. In a few circumstances, these values are output directly. In fields of width 3 or greater, special values are output as however many characters will fit from `+Infinity` or `-Infinity` for infinities, from `NaN` for "not a number," or from `Unknown` for other values (if any are supported by the system). In fields under 3 columns wide, special values are output as asterisks.

5.7.4.2 Custom Currency Formats

The custom currency formats are closely related to the basic numeric formats, but they allow users to customize the output format. The SET command configures custom currency formats, using the syntax

> SET CC*x*="*string*".

where *x* is A, B, C, D, or E, and *string* is no more than 16 characters long.

string must contain exactly three commas or exactly three periods (but not both), except that a single quote character may be used to "escape" a following comma, period, or single quote. If three commas are used, commas will be used for grouping in output, and a period will be used as the decimal point. Uses of periods reverses these roles.

The commas or periods divide *string* into four fields, called the *negative prefix*, *prefix*, *suffix*, and *negative suffix*, respectively. The prefix and suffix are added to output whenever space is available. The negative prefix and negative suffix are always added to a negative number when the output includes a nonzero digit.

The following syntax shows how custom currency formats could be used to reproduce basic numeric formats:

```
SET CCA="-,,,".  /* Same as COMMA.
SET CCB="-...".  /* Same as DOT.
SET CCC="-,$,,". /* Same as DOLLAR.
SET CCD="-,,%,". /* Like PCT, but groups with commas.
```

Here are some more examples of custom currency formats. The final example shows how to use a single quote to escape a delimiter:

```
SET CCA=",EUR,,-".   /* Euro.
SET CCB="(,USD ,,)". /* US dollar.
SET CCC="-.R$..".    /* Brazilian real.
SET CCD="-,, NIS,".  /* Israel shekel.
SET CCE="-.Rp'. ..". /* Indonesia Rupiah.
```

These formats would yield the following output:

Format	3145.59	−3145.59
CCA12.2	EUR3,145.59	EUR3,145.59−
CCB14.2	USD 3,145.59	(USD 3,145.59)
CCC11.2	R$3.145,59	−R$3.145,59
CCD13.2	3,145.59 NIS	−3,145.59 NIS
CCE10.0	Rp. 3.146	−Rp. 3.146

The default for all the custom currency formats is '-,,,', equivalent to COMMA format.

5.7.4.3 Legacy Numeric Formats

The N and Z numeric formats provide compatibility with legacy file formats. They have much in common:

- Output is rounded to the nearest representable value, with ties rounded away from zero.
- Numbers too large to display are output as a field filled with asterisks ('*').
- The decimal point is always implicitly the specified number of digits from the right edge of the field, except that Z format input allows an explicit decimal point.
- Scientific notation may not be used.
- The system-missing value is output as a period in a field of spaces. The period is placed just to the right of the implied decimal point in Z format, or at the right end in N format or in Z format if no decimal places are requested. A period is used even if the decimal point character is a comma.
- Field width may range from 1 to 40. Decimal places may range from 0 up to the field width, to a maximum of 16.
- When a legacy numeric format used for input is converted to an output format, it is changed into the equivalent F format. The field width is increased by 1 if any decimal places are specified, to make room for a decimal point. For Z format, the field width is increased by 1 more column, to make room for a negative sign. The output field width is capped at 40 columns.

N Format

The N format supports input and output of fields that contain only digits. On input, leading or trailing spaces, a decimal point, or any other non-digit character causes the field to be read as the system-missing value. As a special exception, an N format used on DATA LIST FREE or DATA LIST LIST is treated as the equivalent F format.

On output, N pads the field on the left with zeros. Negative numbers are output like the system-missing value.

Z Format

The Z format is a "zoned decimal" format used on IBM mainframes. Z format encodes the sign as part of the final digit, which must be one of the following:

```
0123456789
```

```
{ABCDEFGHI
}JKLMNOPQR
```

where the characters in each row represent digits 0 through 9 in order. Characters in the first two rows indicate a positive sign; those in the third indicate a negative sign.

On output, Z fields are padded on the left with spaces. On input, leading and trailing spaces are ignored. Any character in an input field other than spaces, the digit characters above, and '.' causes the field to be read as system-missing.

The decimal point character for input and output is always '.', even if the decimal point character is a comma (see [SET DECIMAL], page 195).

Nonzero, negative values output in Z format are marked as negative even when no nonzero digits are output. For example, -0.2 is output in Z1.0 format as 'J'. The "negative zero" value supported by most machines is output as positive.

5.7.4.4 Binary and Hexadecimal Numeric Formats

The binary and hexadecimal formats are primarily designed for compatibility with existing machine formats, not for human readability. All of them therefore have a F format as default output format. Some of these formats are only portable between machines with compatible byte ordering (endianness) or floating-point format.

Binary formats use byte values that in text files are interpreted as special control functions, such as carriage return and line feed. Thus, data in binary formats should not be included in syntax files or read from data files with variable-length records, such as ordinary text files. They may be read from or written to data files with fixed-length records. See Section 7.6 [FILE HANDLE], page 133, for information on working with fixed-length records.

P and PK Formats

These are binary-coded decimal formats, in which every byte (except the last, in P format) represents two decimal digits. The most-significant 4 bits of the first byte is the most-significant decimal digit, the least-significant 4 bits of the first byte is the next decimal digit, and so on.

In P format, the most-significant 4 bits of the last byte are the least-significant decimal digit. The least-significant 4 bits represent the sign: decimal 15 indicates a negative value, decimal 13 indicates a positive value.

Numbers are rounded downward on output. The system-missing value and numbers outside representable range are output as zero.

The maximum field width is 16. Decimal places may range from 0 up to the number of decimal digits represented by the field.

The default output format is an F format with twice the input field width, plus one column for a decimal point (if decimal places were requested).

IB and PIB Formats

These are integer binary formats. IB reads and writes 2's complement binary integers, and PIB reads and writes unsigned binary integers. The byte ordering is by default the host machine's, but SET RIB may be used to select a specific byte ordering for reading (see [SET RIB], page 195) and SET WIB, similarly, for writing (see [SET WIB], page 197).

The maximum field width is 8. Decimal places may range from 0 up to the number of decimal digits in the largest value representable in the field width.

The default output format is an F format whose width is the number of decimal digits in the largest value representable in the field width, plus 1 if the format has decimal places.

RB Format

This is a binary format for real numbers. By default it reads and writes the host machine's floating-point format, but SET RRB may be used to select an alternate floating-point format for reading (see [SET RRB], page 195) and SET WRB, similarly, for writing (see [SET WRB], page 197).

The recommended field width depends on the floating-point format. NATIVE (the default format), IDL, IDB, VD, VG, and ZL formats should use a field width of 8. ISL, ISB, VF, and ZS formats should use a field width of 4. Other field widths will not produce useful results. The maximum field width is 8. No decimal places may be specified.

The default output format is F8.2.

PIBHEX and RBHEX Formats

These are hexadecimal formats, for reading and writing binary formats where each byte has been recoded as a pair of hexadecimal digits.

A hexadecimal field consists solely of hexadecimal digits '0'...'9' and 'A'...'F'. Uppercase and lowercase are accepted on input; output is in uppercase.

Other than the hexadecimal representation, these formats are equivalent to PIB and RB formats, respectively. However, bytes in PIBHEX format are always ordered with the most-significant byte first (big-endian order), regardless of the host machine's native byte order or PSPP settings.

Field widths must be even and between 2 and 16. RBHEX format allows no decimal places; PIBHEX allows as many decimal places as a PIB format with half the given width.

5.7.4.5 Time and Date Formats

In PSPP, a *time* is an interval. The time formats translate between human-friendly descriptions of time intervals and PSPP's internal representation of time intervals, which is simply the number of seconds in the interval. PSPP has two time formats:

Time Format	Template	Example
TIME	hh:MM:SS.ss	04:31:17.01
DTIME	DD HH:MM:SS.ss	00 04:31:17.01

A *date* is a moment in the past or the future. Internally, PSPP represents a date as the number of seconds since the *epoch*, midnight, Oct. 14, 1582. The date formats translate between human-readable dates and PSPP's numeric representation of dates and times. PSPP has several date formats:

Date Format	Template	Example
DATE	`dd-mmm-yyyy`	`01-OCT-1978`
ADATE	`mm/dd/yyyy`	`10/01/1978`
EDATE	`dd.mm.yyyy`	`01.10.1978`
JDATE	`yyyyjjj`	`1978274`
SDATE	`yyyy/mm/dd`	`1978/10/01`
QYR	`q Q yyyy`	`3 Q 1978`
MOYR	`mmm yyyy`	`OCT 1978`
WKYR	`ww WK yyyy`	`40 WK 1978`
DATETIME	`dd-mmm-yyyy HH:MM:SS.ss`	`01-OCT-1978 04:31:17.01`

The templates in the preceding tables describe how the time and date formats are input and output:

`dd`   Day of month, from 1 to 31. Always output as two digits.

`mm`

`mmm`   Month. In output, `mm` is output as two digits, `mmm` as the first three letters of an English month name (January, February, . . .). In input, both of these formats, plus Roman numerals, are accepted.

`yyyy`   Year. In output, DATETIME always produces a 4-digit year; other formats can produce a 2- or 4-digit year. The century assumed for 2-digit years depends on the EPOCH setting (see [SET EPOCH], page 195). In output, a year outside the epoch causes the whole field to be filled with asterisks ('`*`').

`jjj`   Day of year (Julian day), from 1 to 366. This is exactly three digits giving the count of days from the start of the year. January 1 is considered day 1.

`q`   Quarter of year, from 1 to 4. Quarters start on January 1, April 1, July 1, and October 1.

`ww`   Week of year, from 1 to 53. Output as exactly two digits. January 1 is the first day of week 1.

`DD`   Count of days, which may be positive or negative. Output as at least two digits.

`hh`   Count of hours, which may be positive or negative. Output as at least two digits.

`HH`   Hour of day, from 0 to 23. Output as exactly two digits.

`MM`   Minute of hour, from 0 to 59. Output as exactly two digits.

`SS.ss`   Seconds within minute, from 0 to 59. The integer part is output as exactly two digits. On output, seconds and fractional seconds may or may not be included, depending on field width and decimal places. On input, seconds and fractional seconds are optional. The DECIMAL setting controls the character accepted and displayed as the decimal point (see [SET DECIMAL], page 195).

For output, the date and time formats use the delimiters indicated in the table. For input, date components may be separated by spaces or by one of the characters '`-`', '`/`',

'.', or ',', and time components may be separated by spaces, ':', or '.'. On input, the 'Q' separating quarter from year and the 'WK' separating week from year may be uppercase or lowercase, and the spaces around them are optional.

On input, all time and date formats accept any amount of leading and trailing white space.

The maximum width for time and date formats is 40 columns. Minimum input and output width for each of the time and date formats is shown below:

Format	Min. Input Width	Min. Output Width	Option
DATE	8	9	4-digit year
ADATE	8	8	4-digit year
EDATE	8	8	4-digit year
JDATE	5	5	4-digit year
SDATE	8	8	4-digit year
QYR	4	6	4-digit year
MOYR	6	6	4-digit year
WKYR	6	8	4-digit year
DATETIME	17	17	seconds
TIME	5	5	seconds
DTIME	8	8	seconds

In the table, "Option" describes what increased output width enables:

4-digit year

A field 2 columns wider than minimum will include a 4-digit year. (DATETIME format always includes a 4-digit year.)

seconds A field 3 columns wider than minimum will include seconds as well as minutes. A field 5 columns wider than minimum, or more, can also include a decimal point and fractional seconds (but no more than allowed by the format's decimal places).

For the time and date formats, the default output format is the same as the input format, except that PSPP increases the field width, if necessary, to the minimum allowed for output.

Time or dates narrower than the field width are right-justified within the field.

When a time or date exceeds the field width, characters are trimmed from the end until it fits. This can occur in an unusual situation, e.g. with a year greater than 9999 (which adds an extra digit), or for a negative value on TIME or DTIME (which adds a leading minus sign).

The system-missing value is output as a period at the right end of the field.

5.7.4.6 Date Component Formats

The WKDAY and MONTH formats provide input and output for the names of weekdays and months, respectively.

On output, these formats convert a number between 1 and 7, for WKDAY, or between 1 and 12, for MONTH, into the English name of a day or month, respectively. If the name is longer than the field, it is trimmed to fit. If the name is shorter than the field, it is padded

on the right with spaces. Values outside the valid range, and the system-missing value, are output as all spaces.

On input, English weekday or month names (in uppercase or lowercase) are converted back to their corresponding numbers. Weekday and month names may be abbreviated to their first 2 or 3 letters, respectively.

The field width may range from 2 to 40, for WKDAY, or from 3 to 40, for MONTH. No decimal places are allowed.

The default output format is the same as the input format.

5.7.4.7 String Formats

The A and AHEX formats are the only ones that may be assigned to string variables. Neither format allows any decimal places.

In A format, the entire field is treated as a string value. The field width may range from 1 to 32,767, the maximum string width. The default output format is the same as the input format.

In AHEX format, the field is composed of characters in a string encoded as hex digit pairs. On output, hex digits are output in uppercase; on input, uppercase and lowercase are both accepted. The default output format is A format with half the input width.

5.7.5 Scratch Variables

Most of the time, variables don't retain their values between cases. Instead, either they're being read from a data file or the active file, in which case they assume the value read, or, if created with COMPUTE or another transformation, they're initialized to the system-missing value or to blanks, depending on type.

However, sometimes it's useful to have a variable that keeps its value between cases. You can do this with LEAVE (see Section 9.6 [LEAVE], page 156), or you can use a *scratch variable*. Scratch variables are variables whose names begin with an octothorpe ('**#**').

Scratch variables have the same properties as variables left with LEAVE: they retain their values between cases, and for the first case they are initialized to 0 or blanks. They have the additional property that they are deleted before the execution of any procedure. For this reason, scratch variables can't be used for analysis. To use a scratch variable in an analysis, use COMPUTE (see Section 10.3 [COMPUTE], page 165) to copy its value into an ordinary variable, then use that ordinary variable in the analysis.

5.8 Files Used by PSPP

PSPP makes use of many files each time it runs. Some of these it reads, some it writes, some it creates. Here is a table listing the most important of these files:

command file
syntax file These names (synonyms) refer to the file that contains instructions that tell PSPP what to do. The syntax file's name is specified on the PSPP command line. Syntax files can also be read with INCLUDE (see Section 14.14 [IN-CLUDE], page 192).

data file   Data files contain raw data in text or binary format. Data can also be embedded in a syntax file with BEGIN DATA and END DATA.

listing file   One or more output files are created by PSPP each time it is run. The output files receive the tables and charts produced by statistical procedures. The output files may be in any number of formats, depending on how PSPP is configured.

active file   The active file is the "file" on which all PSPP procedures are performed. The active file consists of a dictionary and a set of cases. The active file is not necessarily a disk file: it is stored in memory if there is room.

system file

System files are binary files that store a dictionary and a set of cases. GET and SAVE read and write system files.

portable file

Portable files are files in a text-based format that store a dictionary and a set of cases. IMPORT and EXPORT read and write portable files.

scratch file

Scratch files consist of a dictionary and cases and may be stored in memory or on disk. Most procedures that act on a system file or portable file can use a scratch file instead. The contents of scratch files persist within a single PSPP session only. GET and SAVE can be used to read and write scratch files. Scratch files are a PSPP extension.

5.9 File Handles

A *file handle* is a reference to a data file, system file, portable file, or scratch file. Most often, a file handle is specified as the name of a file as a string, that is, enclosed within ''' or '"'.

A file name string that begins or ends with '|' is treated as the name of a command to pipe data to or from. You can use this feature to read data over the network using a program such as 'curl' (e.g. GET '|curl -s -S http://example.com/mydata.sav'), to read compressed data from a file using a program such as 'zcat' (e.g. GET '|zcat mydata.sav.gz'), and for many other purposes.

PSPP also supports declaring named file handles with the FILE HANDLE command. This command associates an identifier of your choice (the file handle's name) with a file. Later, the file handle name can be substituted for the name of the file. When PSPP syntax accesses a file multiple times, declaring a named file handle simplifies updating the syntax later to use a different file. Use of FILE HANDLE is also required to read data files in binary formats. See Section 7.6 [FILE HANDLE], page 133, for more information.

PSPP assumes that a file handle name that begins with '#' refers to a scratch file, unless the name has already been declared on FILE HANDLE to refer to another kind of file. A scratch file is similar to a system file, except that it persists only for the duration of a given PSPP session. Most commands that read or write a system or portable file, such as GET and SAVE, also accept scratch file handles. Scratch file handles may also be declared explicitly with FILE HANDLE. Scratch files are a PSPP extension.

In some circumstances, PSPP must distinguish whether a file handle refers to a system file or a portable file. When this is necessary to read a file, e.g. as an input file for GET or MATCH FILES, PSPP uses the file's contents to decide. In the context of writing a file,

e.g. as an output file for SAVE or AGGREGATE, PSPP decides based on the file's name: if it ends in '.por' (with any capitalization), then PSPP writes a portable file; otherwise, PSPP writes a system file.

INLINE is reserved as a file handle name. It refers to the "data file" embedded into the syntax file between BEGIN DATA and END DATA. See Section 7.1 [BEGIN DATA], page 128, for more information.

The file to which a file handle refers may be reassigned on a later FILE HANDLE command if it is first closed using CLOSE FILE HANDLE. The CLOSE FILE HANDLE command is also useful to free the storage associated with a scratch file. See Section 7.2 [CLOSE FILE HANDLE], page 128, for more information.

5.10 Backus-Naur Form

The syntax of some parts of the PSPP language is presented in this manual using the formalism known as *Backus-Naur Form*, or BNF. The following table describes BNF:

- Words in all-uppercase are PSPP keyword tokens. In BNF, these are often called *terminals*. There are some special terminals, which are written in lowercase for clarity:

  `number`   A real number.

  `integer` An integer number.

  `string`   A string.

  `var-name` A single variable name.

  `=, /, +, -,` etc.
  Operators and punctuators.

  `.`   The end of the command. This is not necessarily an actual dot in the syntax file: See Section 5.2 [Commands], page 93, for more details.

- Other words in all lowercase refer to BNF definitions, called *productions*. These productions are also known as *nonterminals*. Some nonterminals are very common, so they are defined here in English for clarity:

  `var-list` A list of one or more variable names or the keyword `ALL`.

  `expression`
  An expression. See Chapter 6 [Expressions], page 110, for details.

- '::=' means "is defined as". The left side of '::=' gives the name of the nonterminal being defined. The right side of '::=' gives the definition of that nonterminal. If the right side is empty, then one possible expansion of that nonterminal is nothing. A BNF definition is called a *production*.

- So, the key difference between a terminal and a nonterminal is that a terminal cannot be broken into smaller parts—in fact, every terminal is a single token (see Section 5.1 [Tokens], page 92). On the other hand, nonterminals are composed of a (possibly empty) sequence of terminals and nonterminals. Thus, terminals indicate the deepest level of syntax description. (In parsing theory, terminals are the leaves of the parse tree; nonterminals form the branches.)

- The first nonterminal defined in a set of productions is called the *start symbol*. The start symbol defines the entire syntax for that command.

6 Mathematical Expressions

Expressions share a common syntax each place they appear in PSPP commands. Expressions are made up of *operands*, which can be numbers, strings, or variable names, separated by *operators*. There are five types of operators: grouping, arithmetic, logical, relational, and functions.

Every operator takes one or more operands as input and yields exactly one result as output. Depending on the operator, operands accept strings or numbers as operands. With few exceptions, operands may be full-fledged expressions in themselves.

6.1 Boolean Values

Some PSPP operators and expressions work with Boolean values, which represent true/false conditions. Booleans have only three possible values: 0 (false), 1 (true), and system-missing (unknown). System-missing is neither true nor false and indicates that the true value is unknown.

Boolean-typed operands or function arguments must take on one of these three values. Other values are considered false, but provoke a warning when the expression is evaluated.

Strings and Booleans are not compatible, and neither may be used in place of the other.

6.2 Missing Values in Expressions

Most numeric operators yield system-missing when given any system-missing operand. A string operator given any system-missing operand typically results in the empty string. Exceptions are listed under particular operator descriptions.

String user-missing values are not treated specially in expressions.

User-missing values for numeric variables are always transformed into the system-missing value, except inside the arguments to the `VALUE` and `SYSMIS` functions.

The missing-value functions can be used to precisely control how missing values are treated in expressions. See Section 6.7.4 [Missing Value Functions], page 113, for more details.

6.3 Grouping Operators

Parentheses ('`()`') are the grouping operators. Surround an expression with parentheses to force early evaluation.

Parentheses also surround the arguments to functions, but in that situation they act as punctuators, not as operators.

6.4 Arithmetic Operators

The arithmetic operators take numeric operands and produce numeric results.

`a + b`   Yields the sum of a and b.

`a - b`   Subtracts b from a and yields the difference.

`a * b`   Yields the product of a and b. If either a or b is 0, then the result is 0, even if the other operand is missing.

`a / b`   Divides *a* by *b* and yields the quotient. If *a* is 0, then the result is 0, even if *b* is missing. If *b* is zero, the result is system-missing.

`a ** b`   Yields the result of raising *a* to the power *b*. If *a* is negative and *b* is not an integer, the result is system-missing. The result of `0**0` is system-missing as well.

`- a`   Reverses the sign of *a*.

6.5 Logical Operators

The logical operators take logical operands and produce logical results, meaning "true or false." Logical operators are not true Boolean operators because they may also result in a system-missing value. See Section 6.1 [Boolean Values], page 110, for more information.

`a AND b`
`a & b`   True if both *a* and *b* are true, false otherwise. If one operand is false, the result is false even if the other is missing. If both operands are missing, the result is missing.

`a OR b`
`a | b`   True if at least one of *a* and *b* is true. If one operand is true, the result is true even if the other operand is missing. If both operands are missing, the result is missing.

`NOT a`
`~ a`   True if *a* is false. If the operand is missing, then the result is missing.

6.6 Relational Operators

The relational operators take numeric or string operands and produce Boolean results.

   Strings cannot be compared to numbers. When strings of different lengths are compared, the shorter string is right-padded with spaces to match the length of the longer string.

   The results of string comparisons, other than tests for equality or inequality, depend on the character set in use. String comparisons are case-sensitive.

`a EQ b`
`a = b`   True if *a* is equal to *b*.

`a LE b`
`a <= b`   True if *a* is less than or equal to *b*.

`a LT b`
`a < b`   True if *a* is less than *b*.

`a GE b`
`a >= b`   True if *a* is greater than or equal to *b*.

`a GT b`
`a > b`   True if *a* is greater than *b*.

`a NE b`
`a ~= b`
`a <> b`   True if *a* is not equal to *b*.

6.7 Functions

PSPP functions provide mathematical abilities above and beyond those possible using simple operators. Functions have a common syntax: each is composed of a function name followed by a left parenthesis, one or more arguments, and a right parenthesis.

Function names are not reserved. Their names are specially treated only when followed by a left parenthesis, so that `EXP(10)` refers to the constant value `e` raised to the 10th power, but `EXP` by itself refers to the value of variable EXP.

The sections below describe each function in detail.

6.7.1 Mathematical Functions

Advanced mathematical functions take numeric arguments and produce numeric results.

`EXP (`*exponent*`)`   [Function]

    Returns *e* (approximately 2.71828) raised to power *exponent*.

`LG10 (`*number*`)`   [Function]

    Takes the base-10 logarithm of *number*. If *number* is not positive, the result is system-missing.

`LN (`*number*`)`   [Function]

    Takes the base-*e* logarithm of *number*. If *number* is not positive, the result is system-missing.

`LNGAMMA (`*number*`)`   [Function]

    Yields the base-*e* logarithm of the complete gamma of *number*. If *number* is a negative integer, the result is system-missing.

`SQRT (`*number*`)`   [Function]

    Takes the square root of *number*. If *number* is negative, the result is system-missing.

6.7.2 Miscellaneous Mathematical Functions

Miscellaneous mathematical functions take numeric arguments and produce numeric results.

`ABS (`*number*`)`   [Function]

    Results in the absolute value of *number*.

`MOD (`*numerator*`, `*denominator*`)`   [Function]

    Returns the remainder (modulus) of *numerator* divided by *denominator*. If *numerator* is 0, then the result is 0, even if *denominator* is missing. If *denominator* is 0, the result is system-missing.

`MOD10 (`*number*`)`   [Function]

    Returns the remainder when *number* is divided by 10. If *number* is negative, MOD10(*number*) is negative or zero.

`RND (`*number*`)`   [Function]

    Takes the absolute value of *number* and rounds it to an integer. Then, if *number* was negative originally, negates the result.

`TRUNC (`*number*`)`   [Function]

    Discards the fractional part of *number*; that is, rounds *number* towards zero.

6.7.3 Trigonometric Functions

Trigonometric functions take numeric arguments and produce numeric results.

ARCOS (*number*)   [Function]
ACOS (*number*)   [Function]
> Takes the arccosine, in radians, of *number*. Results in system-missing if *number* is not between -1 and 1 inclusive. This function is a PSPP extension.

ARSIN (*number*)   [Function]
ASIN (*number*)   [Function]
> Takes the arcsine, in radians, of *number*. Results in system-missing if *number* is not between -1 and 1 inclusive.

ARTAN (*number*)   [Function]
ATAN (*number*)   [Function]
> Takes the arctangent, in radians, of *number*.

COS (*angle*)   [Function]
> Takes the cosine of *angle* which should be in radians.

SIN (*angle*)   [Function]
> Takes the sine of *angle* which should be in radians.

TAN (*angle*)   [Function]
> Takes the tangent of *angle* which should be in radians. Results in system-missing at values of *angle* that are too close to odd multiples of pi/2. Portability: none.

6.7.4 Missing-Value Functions

Missing-value functions take various numeric arguments and yield various types of results. Except where otherwise stated below, the normal rules of evaluation apply within expression arguments to these functions. In particular, user-missing values for numeric variables are converted to system-missing values.

MISSING (*expr*)   [Function]
> Returns 1 if *expr* has the system-missing value, 0 otherwise.

NMISS (*expr* [, *expr*]...)   [Function]
> Each argument must be a numeric expression. Returns the number of system-missing values in the list, which may include variable ranges using the `var1` TO `var2` syntax.

NVALID (*expr* [, *expr*]...)   [Function]
> Each argument must be a numeric expression. Returns the number of values in the list that are not system-missing. The list may include variable ranges using the `var1` TO `var2` syntax.

SYSMIS (*expr*)   [Function]
> When *expr* is simply the name of a numeric variable, returns 1 if the variable has the system-missing value, 0 if it is user-missing or not missing. If given *expr* takes another form, results in 1 if the value is system-missing, 0 otherwise.

VALUE (`variable`) [Function]

> Prevents the user-missing values of *variable* from being transformed into system-missing values, and always results in the actual value of *variable*, whether it is valid, user-missing, or system-missing.

6.7.5 Set-Membership Functions

Set membership functions determine whether a value is a member of a set. They take a set of numeric arguments or a set of string arguments, and produce Boolean results.

String comparisons are performed according to the rules given in Section 6.6 [Relational Operators], page 111.

ANY (`value`, `set` [, `set`]...) [Function]

> Results in true if *value* is equal to any of the *set* values. Otherwise, results in false. If *value* is system-missing, returns system-missing. System-missing values in *set* do not cause ANY to return system-missing.

RANGE (`value`, `low`, `high` [, `low`, `high`]...) [Function]

> Results in true if *value* is in any of the intervals bounded by *low* and *high* inclusive. Otherwise, results in false. Each *low* must be less than or equal to its corresponding *high* value. *low* and *high* must be given in pairs. If *value* is system-missing, returns system-missing. System-missing values in *set* do not cause RANGE to return system-missing.

6.7.6 Statistical Functions

Statistical functions compute descriptive statistics on a list of values. Some statistics can be computed on numeric or string values; other can only be computed on numeric values. Their results have the same type as their arguments. The current case's weighting factor (see Section 11.7 [WEIGHT], page 173) has no effect on statistical functions.

These functions' argument lists may include entire ranges of variables using the `var1 TO var2` syntax.

Unlike most functions, statistical functions can return non-missing values even when some of their arguments are missing. Most statistical functions, by default, require only 1 non-missing value to have a non-missing return, but CFVAR, SD, and VARIANCE require 2. These defaults can be increased (but not decreased) by appending a dot and the minimum number of valid arguments to the function name. For example, `MEAN.3(X, Y, Z)` would only return non-missing if all of 'X', 'Y', and 'Z' were valid.

CFVAR (`number`, `number`[, ...]) [Function]

> Results in the coefficient of variation of the values of *number*. (The coefficient of variation is the standard deviation divided by the mean.)

MAX (`value`, `value`[, ...]) [Function]

> Results in the value of the greatest *value*. The *value*s may be numeric or string.

MEAN (`number`, `number`[, ...]) [Function]

> Results in the mean of the values of *number*.

MIN (`number`, `number`[, ...]) [Function]

> Results in the value of the least *value*. The *value*s may be numeric or string.

SD (*number*, *number*[, . . .])   [Function]
>   Results in the standard deviation of the values of *number*.

SUM (*number*, *number*[, . . .])   [Function]
>   Results in the sum of the values of *number*.

VARIANCE (*number*, *number*[, . . .])   [Function]
>   Results in the variance of the values of *number*.

6.7.7 String Functions

String functions take various arguments and return various results.

CONCAT (*string*, *string*[, . . .])   [Function]
>   Returns a string consisting of each *string* in sequence. CONCAT("abc", "def",
>   "ghi") has a value of "abcdefghi". The resultant string is truncated to a maximum
>   of 255 characters.

INDEX (*haystack*, *needle*)   [Function]
>   Returns a positive integer indicating the position of the first occurrence of *needle* in
>   *haystack*. Returns 0 if *haystack* does not contain *needle*. Returns system-missing if
>   *needle* is an empty string.

INDEX (*haystack*, *needles*, *needle_len*)   [Function]
>   Divides *needles* into one or more needles, each with length *needle_len*. Searches
>   *haystack* for the first occurrence of each needle, and returns the smallest value. Re-
>   turns 0 if *haystack* does not contain any part in *needle*. It is an error if *needle_len*
>   does not evenly divide the length of *needles*. Returns system-missing if *needles* is an
>   empty string.

LENGTH (*string*)   [Function]
>   Returns the number of characters in *string*.

LOWER (*string*)   [Function]
>   Returns a string identical to *string* except that all uppercase letters are changed
>   to lowercase letters. The definitions of "uppercase" and "lowercase" are system-
>   dependent.

LPAD (*string*, *length*)   [Function]
>   If *string* is at least *length* characters in length, returns *string* unchanged. Otherwise,
>   returns *string* padded with spaces on the left side to length *length*. Returns an empty
>   string if *length* is system-missing, negative, or greater than 255.

LPAD (*string*, *length*, *padding*)   [Function]
>   If *string* is at least *length* characters in length, returns *string* unchanged. Otherwise,
>   returns *string* padded with *padding* on the left side to length *length*. Returns an
>   empty string if *length* is system-missing, negative, or greater than 255, or if *padding*
>   does not contain exactly one character.

LTRIM (*string*)   [Function]
>   Returns *string*, after removing leading spaces. Other white space, such as tabs, car-
>   riage returns, line feeds, and vertical tabs, is not removed.

LTRIM (*string*, *padding*)   [Function]
> Returns *string*, after removing leading *padding* characters. If *padding* does not contain exactly one character, returns an empty string.

NUMBER (*string*, *format*)   [Function]
> Returns the number produced when *string* is interpreted according to format specifier *format*. If the format width w is less than the length of *string*, then only the first w characters in *string* are used, e.g. `NUMBER("123", F3.0)` and `NUMBER("1234", F3.0)` both have value 123. If w is greater than *string*'s length, then it is treated as if it were right-padded with spaces. If *string* is not in the correct format for *format*, system-missing is returned.

RINDEX (*string*, *format*)   [Function]
> Returns a positive integer indicating the position of the last occurrence of *needle* in *haystack*. Returns 0 if *haystack* does not contain *needle*. Returns system-missing if *needle* is an empty string.

RINDEX (*haystack*, *needle*, *needle_len*)   [Function]
> Divides *needle* into parts, each with length *needle_len*. Searches *haystack* for the last occurrence of each part, and returns the largest value. Returns 0 if *haystack* does not contain any part in *needle*. It is an error if *needle_len* does not evenly divide the length of *needle*. Returns system-missing if *needle* is an empty string.

RPAD (*string*, *length*)   [Function]
> If *string* is at least *length* characters in length, returns *string* unchanged. Otherwise, returns *string* padded with spaces on the right to length *length*. Returns an empty string if *length* is system-missing, negative, or greater than 255.

RPAD (*string*, *length*, *padding*)   [Function]
> If *string* is at least *length* characters in length, returns *string* unchanged. Otherwise, returns *string* padded with *padding* on the right to length *length*. Returns an empty string if *length* is system-missing, negative, or greater than 255, or if *padding* does not contain exactly one character.

RTRIM (*string*)   [Function]
> Returns *string*, after removing trailing spaces. Other types of white space are not removed.

RTRIM (*string*, *padding*)   [Function]
> Returns *string*, after removing trailing *padding* characters. If *padding* does not contain exactly one character, returns an empty string.

STRING (*number*, *format*)   [Function]
> Returns a string corresponding to *number* in the format given by format specifier *format*. For example, `STRING(123.56, F5.1)` has the value `"123.6"`.

SUBSTR (*string*, *start*)   [Function]
> Returns a string consisting of the value of *string* from position *start* onward. Returns an empty string if *start* is system-missing, less than 1, or greater than the length of *string*.

SUBSTR (*string*, *start*, *count*)   [Function]

Returns a string consisting of the first *count* characters from *string* beginning at position *start*. Returns an empty string if *start* or *count* is system-missing, if *start* is less than 1 or greater than the number of characters in *string*, or if *count* is less than 1. Returns a string shorter than *count* characters if *start* + *count* - 1 is greater than the number of characters in *string*. Examples: `SUBSTR("abcdefg", 3, 2)` has value `"cd"`; `SUBSTR("nonsense", 4, 10)` has the value `"sense"`.

UPCASE (*string*)   [Function]

Returns *string*, changing lowercase letters to uppercase letters.

6.7.8 Time & Date Functions

For compatibility, PSPP considers dates before 15 Oct 1582 invalid. Most time and date functions will not accept earlier dates.

6.7.8.1 How times & dates are defined and represented

Times and dates are handled by PSPP as single numbers. A *time* is an interval. PSPP measures times in seconds. Thus, the following intervals correspond with the numeric values given:

```
10 minutes                      600
1 hour                        3,600
1 day, 3 hours, 10 seconds   97,210
40 days                   3,456,000
```

A *date*, on the other hand, is a particular instant in the past or the future. PSPP represents a date as a number of seconds since midnight preceding 14 Oct 1582. Because midnight preceding the dates given below correspond with the numeric PSPP dates given:

```
15 Oct 1582            86,400
 4 Jul 1776     6,113,318,400
 1 Jan 1900    10,010,390,400
 1 Oct 1978    12,495,427,200
24 Aug 1995    13,028,601,600
```

6.7.8.2 Functions that Produce Times

These functions take numeric arguments and return numeric values that represent times.

TIME.DAYS (*ndays*)   [Function]

Returns a time corresponding to *ndays* days.

TIME.HMS (*nhours*, *nmins*, *nsecs*)   [Function]

Returns a time corresponding to *nhours* hours, *nmins* minutes, and *nsecs* seconds. The arguments may not have mixed signs: if any of them are positive, then none may be negative, and vice versa.

6.7.8.3 Functions that Examine Times

These functions take numeric arguments in PSPP time format and give numeric results.

CTIME.DAYS (*time*)   [Function]

Results in the number of days and fractional days in *time*.

CTIME.HOURS (`time`) [Function]
> Results in the number of hours and fractional hours in *time*.

CTIME.MINUTES (`time`) [Function]
> Results in the number of minutes and fractional minutes in *time*.

CTIME.SECONDS (`time`) [Function]
> Results in the number of seconds and fractional seconds in *time*. (`CTIME.SECONDS` does nothing; `CTIME.SECONDS(x)` is equivalent to `x`.)

6.7.8.4 Functions that Produce Dates

These functions take numeric arguments and give numeric results that represent dates. Arguments taken by these functions are:

day   Refers to a day of the month between 1 and 31. Day 0 is also accepted and refers to the final day of the previous month. Days 29, 30, and 31 are accepted even in months that have fewer days and refer to a day near the beginning of the following month.

month   Refers to a month of the year between 1 and 12. Months 0 and 13 are also accepted and refer to the last month of the preceding year and the first month of the following year, respectively.

quarter   Refers to a quarter of the year between 1 and 4. The quarters of the year begin on the first day of months 1, 4, 7, and 10.

week   Refers to a week of the year between 1 and 53.

yday   Refers to a day of the year between 1 and 366.

year   Refers to a year, 1582 or greater. Years between 0 and 99 are treated according to the epoch set on SET EPOCH, by default beginning 69 years before the current date (see [SET EPOCH], page 195).

If these functions' arguments are out-of-range, they are correctly normalized before conversion to date format. Non-integers are rounded toward zero.

DATE.DMY (`day`, `month`, `year`) [Function]
DATE.MDY (`month`, `day`, `year`) [Function]
> Results in a date value corresponding to the midnight before day *day* of month *month* of year *year*.

DATE.MOYR (`month`, `year`) [Function]
> Results in a date value corresponding to the midnight before the first day of month *month* of year *year*.

DATE.QYR (`quarter`, `year`) [Function]
> Results in a date value corresponding to the midnight before the first day of quarter *quarter* of year *year*.

DATE.WKYR (`week`, `year`) [Function]
> Results in a date value corresponding to the midnight before the first day of week *week* of year *year*.

DATE.YRDAY (`year`, `yday`)   [Function]
    Results in a date value corresponding to the day *yday* of year *year*.

6.7.8.5 Functions that Examine Dates

These functions take numeric arguments in PSPP date or time format and give numeric results. These names are used for arguments:

date   A numeric value in PSPP date format.

time   A numeric value in PSPP time format.

time-or-date
        A numeric value in PSPP time or date format.

XDATE.DATE (`time-or-date`)   [Function]
    For a time, results in the time corresponding to the number of whole days *date-or-time* includes. For a date, results in the date corresponding to the latest midnight at or before *date-or-time*; that is, gives the date that *date-or-time* is in.

XDATE.HOUR (`time-or-date`)   [Function]
    For a time, results in the number of whole hours beyond the number of whole days represented by *date-or-time*. For a date, results in the hour (as an integer between 0 and 23) corresponding to *date-or-time*.

XDATE.JDAY (`date`)   [Function]
    Results in the day of the year (as an integer between 1 and 366) corresponding to *date*.

XDATE.MDAY (`date`)   [Function]
    Results in the day of the month (as an integer between 1 and 31) corresponding to *date*.

XDATE.MINUTE (`time-or-date`)   [Function]
    Results in the number of minutes (as an integer between 0 and 59) after the last hour in *time-or-date*.

XDATE.MONTH (`date`)   [Function]
    Results in the month of the year (as an integer between 1 and 12) corresponding to *date*.

XDATE.QUARTER (`date`)   [Function]
    Results in the quarter of the year (as an integer between 1 and 4) corresponding to *date*.

XDATE.SECOND (`time-or-date`)   [Function]
    Results in the number of whole seconds after the last whole minute (as an integer between 0 and 59) in *time-or-date*.

XDATE.TDAY (`date`)   [Function]
    Results in the number of whole days from 14 Oct 1582 to *date*.

XDATE.TIME (`date`) [Function]

   Results in the time of day at the instant corresponding to *date*, as a time value. This is the number of seconds since midnight on the day corresponding to *date*.

XDATE.WEEK (`date`) [Function]

   Results in the week of the year (as an integer between 1 and 53) corresponding to *date*.

XDATE.WKDAY (`date`) [Function]

   Results in the day of week (as an integer between 1 and 7) corresponding to *date*, where 1 represents Sunday.

XDATE.YEAR (`date`) [Function]

   Returns the year (as an integer 1582 or greater) corresponding to *date*.

6.7.8.6 Time and Date Arithmetic

Ordinary arithmetic operations on dates and times often produce sensible results. Adding a time to, or subtracting one from, a date produces a new date that much earlier or later. The difference of two dates yields the time between those dates. Adding two times produces the combined time. Multiplying a time by a scalar produces a time that many times longer. Since times and dates are just numbers, the ordinary addition and subtraction operators are employed for these purposes.

   Adding two dates does not produce a useful result.

   Dates and times may have very large values. Thus, it is not a good idea to take powers of these values; also, the accuracy of some procedures may be affected. If necessary, convert times or dates in seconds to some other unit, like days or years, before performing analysis.

   PSPP supplies a few functions for date arithmetic:

DATEDIFF (`date2`, `date1`, `unit`) [Function]

   Returns the span of time from *date1* to *date2* in terms of *unit*, which must be a quoted string, one of 'years', 'quarters', 'months', 'weeks', 'days', 'hours', 'minutes', and 'seconds'. The result is an integer, truncated toward zero.

   One year is considered to span from a given date to the same month, day, and time of day the next year. Thus, from Jan. 1 of one year to Jan. 1 the next year is considered to be a full year, but Feb. 29 of a leap year to the following Feb. 28 is not. Similarly, one month spans from a given day of the month to the same day of the following month. Thus, there is never a full month from Jan. 31 of a given year to any day in the following February.

DATESUM (`date`, `quantity`, `unit`[, `method`]) [Function]

   Returns *date* advanced by the given *quantity* of the specified *unit*, which must be one of the strings 'years', 'quarters', 'months', 'weeks', 'days', 'hours', 'minutes', and 'seconds'.

   When *unit* is 'years', 'quarters', or 'months', only the integer part of *quantity* is considered. Adding one of these units can cause the day of the month to exceed the number of days in the month. In this case, the *method* comes into play: if it is omitted or specified as 'closest' (as a quoted string), then the resulting day is the

last day of the month; otherwise, if it is specified as '`rollover`', then the extra days roll over into the following month.

When *unit* is '`weeks`', '`days`', '`hours`', '`minutes`', or '`seconds`', the *quantity* is not rounded to an integer and *method*, if specified, is ignored.

6.7.9 Miscellaneous Functions

LAG (`variable`[, `n`])   [Function]

variable must be a numeric or string variable name. `LAG` yields the value of that variable for the case *n* before the current one. Results in system-missing (for numeric variables) or blanks (for string variables) for the first *n* cases.

`LAG` obtains values from the cases that become the new active file after a procedure executes. Thus, `LAG` will not return values from cases dropped by transformations such as SELECT IF, and transformations like COMPUTE that modify data will change the values returned by `LAG`. These are both the case whether these transformations precede or follow the use of `LAG`.

If `LAG` is used before TEMPORARY, then the values it returns are those in cases just before TEMPORARY. `LAG` may not be used after TEMPORARY.

If omitted, *ncases* defaults to 1. Otherwise, *ncases* must be a small positive constant integer. There is no explicit limit, but use of a large value will increase memory consumption.

YRMODA (`year`, `month`, `day`)   [Function]

year is a year, either between 0 and 99 or at least 1582. Unlike other PSPP date functions, years between 0 and 99 always correspond to 1900 through 1999. *month* is a month between 1 and 13. *day* is a day between 0 and 31. A *day* of 0 refers to the last day of the previous month, and a *month* of 13 refers to the first month of the next year. *year* must be in range. *year*, *month*, and *day* must all be integers.

`YRMODA` results in the number of days between 15 Oct 1582 and the date specified, plus one. The date passed to `YRMODA` must be on or after 15 Oct 1582. 15 Oct 1582 has a value of 1.

VALUELABEL (`variable`)   [Function]

Returns a string matching the label associated with the current value of *variable*. If the current value of *variable* has no associated label, then this function returns the empty string. *variable* may be a numeric or string variable.

6.7.10 Statistical Distribution Functions

PSPP can calculate several functions of standard statistical distributions. These functions are named systematically based on the function and the distribution. The table below describes the statistical distribution functions in general:

PDF.*dist* (x[, *param*...])

Probability density function for *dist*. The domain of x depends on *dist*. For continuous distributions, the result is the density of the probability function at x, and the range is nonnegative real numbers. For discrete distributions, the result is the probability of x.

CDF.*dist* (x[, *param*. . .])

> Cumulative distribution function for *dist*, that is, the probability that a random variate drawn from the distribution is less than x. The domain of x depends *dist*. The result is a probability.

SIG.*dist* (x[, *param*. . .)

> Tail probability function for *dist*, that is, the probability that a random variate drawn from the distribution is greater than x. The domain of x depends *dist*. The result is a probability. Only a few distributions include an SIG function.

IDF.*dist* (p[, *param*. . .])

> Inverse distribution function for *dist*, the value of x for which the CDF would yield p. The value of p is a probability. The range depends on *dist* and is identical to the domain for the corresponding CDF.

RV.*dist* ([*param*. . .])

> Random variate function for *dist*. The range depends on the distribution.

NPDF.*dist* (x[, *param*. . .])

> Noncentral probability density function. The result is the density of the given noncentral distribution at x. The domain of x depends on *dist*. The range is nonnegative real numbers. Only a few distributions include an NPDF function.

NCDF.*dist* (x[, *param*. . .])

> Noncentral cumulative distribution function for *dist*, that is, the probability that a random variate drawn from the given noncentral distribution is less than x. The domain of x depends *dist*. The result is a probability. Only a few distributions include an NCDF function.

The individual distributions are described individually below.

6.7.10.1 Continuous Distributions

The following continuous distributions are available:

PDF.BETA (x)	[Function]
CDF.BETA (x, a, b)	[Function]
IDF.BETA (p, a, b)	[Function]
RV.BETA (a, b)	[Function]
NPDF.BETA (x, a, b, lambda)	[Function]
NCDF.BETA (x, a, b, lambda)	[Function]

> Beta distribution with shape parameters a and b. The noncentral distribution takes an additional parameter *lambda*. Constraints: a > 0, b > 0, lambda >= 0, 0 <= x <= 1, 0 <= p <= 1.

PDF.BVNOR (x0, x1, rho)	[Function]
CDF.VBNOR (x0, x1, rho)	[Function]

> Bivariate normal distribution of two standard normal variables with correlation coefficient *rho*. Two variates x0 and x1 must be provided. Constraints: 0 <= rho <= 1, 0 <= p <= 1.

PDF.CAUCHY (*x*, *a*, *b*)   [Function]
CDF.CAUCHY (*x*, *a*, *b*)   [Function]
IDF.CAUCHY (*p*, *a*, *b*)   [Function]
RV.CAUCHY (*a*, *b*)   [Function]
    Cauchy distribution with location parameter *a* and scale parameter *b*. Constraints:
    $b > 0$, $0 < p < 1$.

PDF.CHISQ (*x*, *df*)   [Function]
CDF.CHISQ (*x*, *df*)   [Function]
SIG.CHISQ (*x*, *df*)   [Function]
IDF.CHISQ (*p*, *df*)   [Function]
RV.CHISQ (*df*)   [Function]
NPDF.CHISQ (*x*, *df*, *lambda*)   [Function]
NCDF.CHISQ (*x*, *df*, *lambda*)   [Function]
    Chi-squared distribution with *df* degrees of freedom. The noncentral distribution
    takes an additional parameter *lambda*. Constraints: $df > 0$, $lambda > 0$, $x >= 0$, 0
    $<= p < 1$.

PDF.EXP (*x*, *a*)   [Function]
CDF.EXP (*x*, *a*)   [Function]
IDF.EXP (*p*, *a*)   [Function]
RV.EXP (*a*)   [Function]
    Exponential distribution with scale parameter *a*. The inverse of *a* represents the rate
    of decay. Constraints: $a > 0$, $x >= 0$, $0 <= p < 1$.

PDF.XPOWER (*x*, *a*, *b*)   [Function]
RV.XPOWER (*a*, *b*)   [Function]
    Exponential power distribution with positive scale parameter *a* and nonnegative power
    parameter *b*. Constraints: $a > 0$, $b >= 0$, $x >= 0$, $0 <= p <= 1$. This distribution is
    a PSPP extension.

PDF.F (*x*, *df1*, *df2*)   [Function]
CDF.F (*x*, *df1*, *df2*)   [Function]
SIG.F (*x*, *df1*, *df2*)   [Function]
IDF.F (*p*, *df1*, *df2*)   [Function]
RV.F (*df1*, *df2*)   [Function]
NPDF.F (*x*, *df1*, *df2*, *lambda*)   [Function]
NCDF.F (*x*, *df1*, *df2*, *lambda*)   [Function]
    F-distribution of two chi-squared deviates with *df1* and *df2* degrees of freedom. The
    noncentral distribution takes an additional parameter *lambda*. Constraints: $df1 > 0$,
    $df2 > 0$, $lambda >= 0$, $x >= 0$, $0 <= p < 1$.

PDF.GAMMA (*x*, *a*, *b*)   [Function]
CDF.GAMMA (*x*, *a*, *b*)   [Function]
IDF.GAMMA (*p*, *a*, *b*)   [Function]
RV.GAMMA (*a*, *b*)   [Function]
    Gamma distribution with shape parameter *a* and scale parameter *b*. Constraints: *a*
    > 0, $b > 0$, $x >= 0$, $0 <= p < 1$.

PDF.HALFNRM (x, a, b)   [Function]
CDF.HALFNRM (x, a, b)   [Function]
IDF.HALFNRM (p, a, b)   [Function]
RV.HALFNRM (a, b)   [Function]
    Half-normal distribution with location parameter a and shape parameter b. Constraints: $b > 0$, $0 < p < 1$.

PDF.IGAUSS (x, a, b)   [Function]
CDF.IGAUSS (x, a, b)   [Function]
IDF.IGAUSS (p, a, b)   [Function]
RV.IGAUSS (a, b)   [Function]
    Inverse Gaussian distribution with parameters a and b. Constraints: $a > 0$, $b > 0$, $x > 0$, $0 <= p < 1$.

PDF.LANDAU (x)   [Function]
RV.LANDAU ()   [Function]
    Landau distribution.

PDF.LAPLACE (x, a, b)   [Function]
CDF.LAPLACE (x, a, b)   [Function]
IDF.LAPLACE (p, a, b)   [Function]
RV.LAPLACE (a, b)   [Function]
    Laplace distribution with location parameter a and scale parameter b. Constraints: $b > 0$, $0 < p < 1$.

RV.LEVY (c, $alpha$)   [Function]
    Levy symmetric alpha-stable distribution with scale c and exponent $alpha$. Constraints: $0 < alpha <= 2$.

RV.LVSKEW (c, $alpha$, $beta$)   [Function]
    Levy skew alpha-stable distribution with scale c, exponent $alpha$, and skewness parameter $beta$. Constraints: $0 < alpha <= 2$, $-1 <= beta <= 1$.

PDF.LOGISTIC (x, a, b)   [Function]
CDF.LOGISTIC (x, a, b)   [Function]
IDF.LOGISTIC (p, a, b)   [Function]
RV.LOGISTIC (a, b)   [Function]
    Logistic distribution with location parameter a and scale parameter b. Constraints: $b > 0$, $0 < p < 1$.

PDF.LNORMAL (x, a, b)   [Function]
CDF.LNORMAL (x, a, b)   [Function]
IDF.LNORMAL (p, a, b)   [Function]
RV.LNORMAL (a, b)   [Function]
    Lognormal distribution with parameters a and b. Constraints: $a > 0$, $b > 0$, $x >= 0$, $0 <= p < 1$.

PDF.NORMAL (x, mu, $sigma$)   [Function]
CDF.NORMAL (x, mu, $sigma$)   [Function]

IDF.NORMAL (*p*, *mu*, *sigma*)   [Function]
RV.NORMAL (*mu*, *sigma*)   [Function]

> Normal distribution with mean *mu* and standard deviation *sigma*. Constraints: $b > 0$, $0 < p < 1$. Three additional functions are available as shorthand:

>> CDFNORM (*x*)   [Function]
>>> Equivalent to CDF.NORMAL(*x*, 0, 1).

>> PROBIT (*p*)   [Function]
>>> Equivalent to IDF.NORMAL(*p*, 0, 1).

>> NORMAL (*sigma*)   [Function]
>>> Equivalent to RV.NORMAL(0, *sigma*).

PDF.NTAIL (*x*, *a*, *sigma*)   [Function]
RV.NTAIL (*a*, *sigma*)   [Function]

> Normal tail distribution with lower limit *a* and standard deviation *sigma*. This distribution is a PSPP extension. Constraints: $a > 0$, $x > a$, $0 < p < 1$.

PDF.PARETO (*x*, *a*, *b*)   [Function]
CDF.PARETO (*x*, *a*, *b*)   [Function]
IDF.PARETO (*p*, *a*, *b*)   [Function]
RV.PARETO (*a*, *b*)   [Function]

> Pareto distribution with threshold parameter *a* and shape parameter *b*. Constraints: $a > 0$, $b > 0$, $x >= a$, $0 <= p < 1$.

PDF.RAYLEIGH (*x*, *sigma*)   [Function]
CDF.RAYLEIGH (*x*, *sigma*)   [Function]
IDF.RAYLEIGH (*p*, *sigma*)   [Function]
RV.RAYLEIGH (*sigma*)   [Function]

> Rayleigh distribution with scale parameter *sigma*. This distribution is a PSPP extension. Constraints: $sigma > 0$, $x > 0$.

PDF.RTAIL (*x*, *a*, *sigma*)   [Function]
RV.RTAIL (*a*, *sigma*)   [Function]

> Rayleigh tail distribution with lower limit *a* and scale parameter *sigma*. This distribution is a PSPP extension. Constraints: $a > 0$, $sigma > 0$, $x > a$.

CDF.SMOD (*x*, *a*, *b*)   [Function]
IDF.SMOD (*p*, *a*, *b*)   [Function]

> Studentized maximum modulus distribution with parameters *a* and *b*. Constraints: $a > 0$, $b > 0$, $x > 0$, $0 <= p < 1$.

CDF.SRANGE (*x*, *a*, *b*)   [Function]
IDF.SRANGE (*p*, *a*, *b*)   [Function]

> Studentized range distribution with parameters *a* and *b*. Constraints: $a >= 1$, $b >= 1$, $x > 0$, $0 <= p < 1$.

PDF.T (*x*, *df*)   [Function]
CDF.T (*x*, *df*)   [Function]

`IDF.T (p, df)`	[Function]
`RV.T (df)`	[Function]
`NPDF.T (x, df, lambda)`	[Function]
`NCDF.T (x, df, lambda)`	[Function]

> T-distribution with *df* degrees of freedom. The noncentral distribution takes an additional parameter *lambda*. Constraints: $df > 0$, $0 < p < 1$.

`PDF.T1G (x, a, b)`	[Function]
`CDF.T1G (x, a, b)`	[Function]
`IDF.T1G (p, a, b)`	[Function]

> Type-1 Gumbel distribution with parameters *a* and *b*. This distribution is a PSPP extension. Constraints: $0 < p < 1$.

`PDF.T2G (x, a, b)`	[Function]
`CDF.T2G (x, a, b)`	[Function]
`IDF.T2G (p, a, b)`	[Function]

> Type-2 Gumbel distribution with parameters *a* and *b*. This distribution is a PSPP extension. Constraints: $x > 0$, $0 < p < 1$.

`PDF.UNIFORM (x, a, b)`	[Function]
`CDF.UNIFORM (x, a, b)`	[Function]
`IDF.UNIFORM (p, a, b)`	[Function]
`RV.UNIFORM (a, b)`	[Function]

> Uniform distribution with parameters *a* and *b*. Constraints: `a <= x <= b`, `0 <= p <= 1`. An additional function is available as shorthand:

> | | |
> |---|---|
> | `UNIFORM (b)` | [Function] |
>
> > Equivalent to RV.UNIFORM(0, *b*).

`PDF.WEIBULL (x, a, b)`	[Function]
`CDF.WEIBULL (x, a, b)`	[Function]
`IDF.WEIBULL (p, a, b)`	[Function]
`RV.WEIBULL (a, b)`	[Function]

> Weibull distribution with parameters *a* and *b*. Constraints: `a > 0`, `b > 0`, `x >= 0`, `0 <= p < 1`.

6.7.10.2 Discrete Distributions

The following discrete distributions are available:

`PDF.BERNOULLI (x)`	[Function]
`CDF.BERNOULLI (x, p)`	[Function]
`RV.BERNOULLI (p)`	[Function]

> Bernoulli distribution with probability of success *p*. Constraints: `x = 0 or 1`, `0 <= p <= 1`.

`PDF.BINOMIAL (x, n, p)`	[Function]
`CDF.BINOMIAL (x, n, p)`	[Function]
`RV.BINOMIAL (n, p)`	[Function]

> Binomial distribution with *n* trials and probability of success *p*. Constraints: integer `n > 0`, `0 <= p <= 1`, integer `x <= n`.

```
PDF.GEOM (x, n, p)                                              [Function]
CDF.GEOM (x, n, p)                                              [Function]
RV.GEOM (n, p)                                                  [Function]
```
   Geometric distribution with probability of success p. Constraints: $0 \text{ <= } p \text{ <= } 1$, integer $x \text{ > } 0$.

```
PDF.HYPER (x, a, b, c)                                          [Function]
CDF.HYPER (x, a, b, c)                                          [Function]
RV.HYPER (a, b, c)                                              [Function]
```
   Hypergeometric distribution when b objects out of a are drawn and c of the available objects are distinctive. Constraints: integer $a \text{ > } 0$, integer $b \text{ <= } a$, integer $c \text{ <= } a$, integer $x \text{ >= } 0$.

```
PDF.LOG (x, p)                                                  [Function]
RV.LOG (p)                                                      [Function]
```
   Logarithmic distribution with probability parameter p. Constraints: $0 \text{ <= } p \text{ < } 1$, $x \text{ >= } 1$.

```
PDF.NEGBIN (x, n, p)                                            [Function]
CDF.NEGBIN (x, n, p)                                            [Function]
RV.NEGBIN (n, p)                                                [Function]
```
   Negative binomial distribution with number of successes paramter n and probability of success parameter p. Constraints: integer $n \text{ >= } 0$, $0 \text{ < } p \text{ <= } 1$, integer $x \text{ >= } 1$.

```
PDF.POISSON (x, mu)                                            [Function]
CDF.POISSON (x, mu)                                            [Function]
RV.POISSON (mu)                                                [Function]
```
   Poisson distribution with mean mu. Constraints: $mu \text{ > } 0$, integer $x \text{ >= } 0$.

6.8 Operator Precedence

The following table describes operator precedence. Smaller-numbered levels in the table have higher precedence. Within a level, operations are always performed from left to right. The first occurrence of '-' represents unary negation, the second binary subtraction.

1. ( )
2. **
3. -
4. * /
5. + -
6. EQ GE GT LE LT NE
7. AND NOT OR

7 Data Input and Output

Data are the focus of the PSPP language. Each datum belongs to a *case* (also called an *observation*). Each case represents an individual or "experimental unit". For example, in the results of a survey, the names of the respondents, their sex, age, etc. and their responses are all data and the data pertaining to single respondent is a case. This chapter examines the PSPP commands for defining variables and reading and writing data. There are alternative commands to read data from predefined sources such as system files or databases (See Section 8.3 [GET], page 144.)

> **Note:** These commands tell PSPP how to read data, but the data will not actually be read until a procedure is executed.

7.1 BEGIN DATA

    BEGIN DATA.
    . . .
    END DATA.

BEGIN DATA and END DATA can be used to embed raw ASCII data in a PSPP syntax file. DATA LIST or another input procedure must be used before BEGIN DATA (see Section 7.3 [DATA LIST], page 128). BEGIN DATA and END DATA must be used together. END DATA must appear by itself on a single line, with no leading white space and exactly one space between the words `END` and `DATA`, like this:

    END DATA.

7.2 CLOSE FILE HANDLE

    CLOSE FILE HANDLE handle_name.

CLOSE FILE HANDLE disassociates the name of a file handle with a given file. The only specification is the name of the handle to close. Afterward FILE HANDLE.

If the file handle name refers to a scratch file, then the storage associated with the scratch file in memory or on disk will be freed. If the scratch file is in use, e.g. it has been specified on a GET command whose execution has not completed, then freeing is delayed until it is no longer in use.

The file named INLINE, which represents data entered between BEGIN DATA and END DATA, cannot be closed. Attempts to close it with CLOSE FILE HANDLE have no effect.

CLOSE FILE HANDLE is a PSPP extension.

7.3 DATA LIST

Used to read text or binary data, DATA LIST is the most fundamental data-reading command. Even the more sophisticated input methods use DATA LIST commands as a building block. Understanding DATA LIST is important to understanding how to use PSPP to read your data files.

There are two major variants of DATA LIST, which are fixed format and free format. In addition, free format has a minor variant, list format, which is discussed in terms of its differences from vanilla free format.

Each form of DATA LIST is described in detail below.

See Section 8.4 [GET DATA], page 145, for a command that offers a few enhancements over DATA LIST and that may be substituted for DATA LIST in many situations.

7.3.1 DATA LIST FIXED

```
DATA LIST [FIXED]
        {TABLE,NOTABLE}
        [FILE='file-name']
        [RECORDS=record_count]
        [END=end_var]
        [SKIP=record_count]
        /[line_no] var_spec...
```

where each var_spec takes one of the forms
```
        var_list start-end [type_spec]
        var_list (fortran_spec)
```

DATA LIST FIXED is used to read data files that have values at fixed positions on each line of single-line or multiline records. The keyword FIXED is optional.

The FILE subcommand must be used if input is to be taken from an external file. It may be used to specify a file name as a string or a file handle (see Section 5.9 [File Handles], page 108). If the FILE subcommand is not used, then input is assumed to be specified within the command file using BEGIN DATA...END DATA (see Section 7.1 [BEGIN DATA], page 128).

The optional RECORDS subcommand, which takes a single integer as an argument, is used to specify the number of lines per record. If RECORDS is not specified, then the number of lines per record is calculated from the list of variable specifications later in DATA LIST.

The END subcommand is only useful in conjunction with INPUT PROGRAM. See Section 7.7 [INPUT PROGRAM], page 135, for details.

The optional SKIP subcommand specifies a number of records to skip at the beginning of an input file. It can be used to skip over a row that contains variable names, for example.

DATA LIST can optionally output a table describing how the data file will be read. The TABLE subcommand enables this output, and NOTABLE disables it. The default is to output the table.

The list of variables to be read from the data list must come last. Each line in the data record is introduced by a slash ('/'). Optionally, a line number may follow the slash. Following, any number of variable specifications may be present.

Each variable specification consists of a list of variable names followed by a description of their location on the input line. Sets of variables may be specified using the DATA LIST TO convention (see Section 5.7.3 [Sets of Variables], page 98). There are two ways to specify the location of the variable on the line: columnar style and FORTRAN style.

In columnar style, the starting column and ending column for the field are specified after the variable name, separated by a dash ('-'). For instance, the third through fifth columns on a line would be specified '3-5'. By default, variables are considered to be in 'F' format

(see Section 5.7.4 [Input and Output Formats], page 98). (This default can be changed; see Section 14.17 [SET], page 193 for more information.)

In columnar style, to use a variable format other than the default, specify the format type in parentheses after the column numbers. For instance, for alphanumeric 'A' format, use '(A)'.

In addition, implied decimal places can be specified in parentheses after the column numbers. As an example, suppose that a data file has a field in which the characters '1234' should be interpreted as having the value 12.34. Then this field has two implied decimal places, and the corresponding specification would be '(2)'. If a field that has implied decimal places contains a decimal point, then the implied decimal places are not applied.

Changing the variable format and adding implied decimal places can be done together; for instance, '(N,5)'.

When using columnar style, the input and output width of each variable is computed from the field width. The field width must be evenly divisible into the number of variables specified.

FORTRAN style is an altogether different approach to specifying field locations. With this approach, a list of variable input format specifications, separated by commas, are placed after the variable names inside parentheses. Each format specifier advances as many characters into the input line as it uses.

Implied decimal places also exist in FORTRAN style. A format specification with d decimal places also has d implied decimal places.

In addition to the standard format specifiers (see Section 5.7.4 [Input and Output Formats], page 98), FORTRAN style defines some extensions:

X   Advance the current column on this line by one character position.

Tx   Set the current column on this line to column x, with column numbers considered to begin with 1 at the left margin.

NEWRECx   Skip forward x lines in the current record, resetting the active column to the left margin.

Repeat count
            Any format specifier may be preceded by a number. This causes the action of that format specifier to be repeated the specified number of times.

(spec1, ..., specN)
            Group the given specifiers together. This is most useful when preceded by a repeat count. Groups may be nested arbitrarily.

FORTRAN and columnar styles may be freely intermixed. Columnar style leaves the active column immediately after the ending column specified. Record motion using NEWREC in FORTRAN style also applies to later FORTRAN and columnar specifiers.

Examples

1.

```
DATA LIST TABLE /NAME 1-10 (A) INFO1 TO INFO3 12-17 (1).
```

```
BEGIN DATA.
John Smith 102311
Bob Arnold 122015
Bill Yates  918 6
END DATA.
```

Defines the following variables:

- `NAME`, a 10-character-wide long string variable, in columns 1 through 10.
- `INFO1`, a numeric variable, in columns 12 through 13.
- `INFO2`, a numeric variable, in columns 14 through 15.
- `INFO3`, a numeric variable, in columns 16 through 17.

The `BEGIN DATA`/`END DATA` commands cause three cases to be defined:

```
Case   NAME          INFO1   INFO2   INFO3
   1   John Smith      10      23      11
   2   Bob Arnold      12      20      15
   3   Bill Yates       9      18       6
```

The `TABLE` keyword causes PSPP to print out a table describing the four variables defined.

2.

```
DAT LIS FIL="survey.dat"
        /ID 1-5 NAME 7-36 (A) SURNAME 38-67 (A) MINITIAL 69 (A)
        /Q01 TO Q50 7-56
        /.
```

Defines the following variables:

- `ID`, a numeric variable, in columns 1-5 of the first record.
- `NAME`, a 30-character long string variable, in columns 7-36 of the first record.
- `SURNAME`, a 30-character long string variable, in columns 38-67 of the first record.
- `MINITIAL`, a 1-character short string variable, in column 69 of the first record.
- Fifty variables `Q01`, `Q02`, `Q03`, ..., `Q49`, `Q50`, all numeric, `Q01` in column 7, `Q02` in column 8, ..., `Q49` in column 55, `Q50` in column 56, all in the second record.

Cases are separated by a blank record.

Data is read from file 'survey.dat' in the current directory.

This example shows keywords abbreviated to their first 3 letters.

7.3.2 DATA LIST FREE

```
DATA LIST FREE
      [({TAB,'c'}, ...)]
      [{NOTABLE,TABLE}]
      [FILE='file-name']
      [SKIP=record_cnt]
      /var_spec...
```

where each var_spec takes one of the forms

>   var_list [(type_spec)]
>   var_list *

In free format, the input data is, by default, structured as a series of fields separated by spaces, tabs, commas, or line breaks. Each field's content may be unquoted, or it may be quoted with a pairs of apostrophes ('') or double quotes ('"'). Unquoted white space separates fields but is not part of any field. Any mix of spaces, tabs, and line breaks is equivalent to a single space for the purpose of separating fields, but consecutive commas will skip a field.

Alternatively, delimiters can be specified explicitly, as a parenthesized, comma-separated list of single-character strings immediately following FREE. The word TAB may also be used to specify a tab character as a delimiter. When delimiters are specified explicitly, only the given characters, plus line breaks, separate fields. Furthermore, leading spaces at the beginnings of fields are not trimmed, consecutive delimiters define empty fields, and no form of quoting is allowed.

The NOTABLE and TABLE subcommands are as in DATA LIST FIXED above. NO-TABLE is the default.

The FILE and SKIP subcommands are as in DATA LIST FIXED above.

The variables to be parsed are given as a single list of variable names. This list must be introduced by a single slash ('/'). The set of variable names may contain format specifications in parentheses (see Section 5.7.4 [Input and Output Formats], page 98). Format specifications apply to all variables back to the previous parenthesized format specification.

In addition, an asterisk may be used to indicate that all variables preceding it are to have input/output format 'F8.0'.

Specified field widths are ignored on input, although all normal limits on field width apply, but they are honored on output.

7.3.3 DATA LIST LIST

>   DATA LIST LIST
>   [({TAB,'c'}, ...)]
>   [{NOTABLE,TABLE}]
>   [FILE='file-name']
>   [SKIP=record_count]
>   /var_spec...
>
>   where each var_spec takes one of the forms
>   var_list [(type_spec)]
>   var_list *

With one exception, DATA LIST LIST is syntactically and semantically equivalent to DATA LIST FREE. The exception is that each input line is expected to correspond to exactly one input record. If more or fewer fields are found on an input line than expected, an appropriate diagnostic is issued.

7.4 END CASE

>   END CASE.

END CASE is used only within INPUT PROGRAM to output the current case. See Section 7.7 [INPUT PROGRAM], page 135, for details.

7.5 END FILE

        END FILE.

END FILE is used only within INPUT PROGRAM to terminate the current input program. See Section 7.7 [INPUT PROGRAM], page 135.

7.6 FILE HANDLE

        For text files:
                FILE HANDLE handle_name
                        /NAME='file-name'
                        [/MODE=CHARACTER]
                        /TABWIDTH=tab_width

        For binary files in native encoding with fixed-length records:
                FILE HANDLE handle_name
                        /NAME='file-name'
                        /MODE=IMAGE
                        [/LRECL=rec_len]

        For binary files in native encoding with variable-length records:
                FILE HANDLE handle_name
                        /NAME='file-name'
                        /MODE=BINARY
                        [/LRECL=rec_len]

        For binary files encoded in EBCDIC:
                FILE HANDLE handle_name
                        /NAME='file-name'
                        /MODE=360
                        /RECFORM={FIXED,VARIABLE,SPANNED}
                        [/LRECL=rec_len]

        To explicitly declare a scratch handle:
                FILE HANDLE handle_name
                        /MODE=SCRATCH

Use FILE HANDLE to associate a file handle name with a file and its attributes, so that later commands can refer to the file by its handle name. Names of text files can be specified directly on commands that access files, so that FILE HANDLE is only needed when a file is not an ordinary file containing lines of text. However, FILE HANDLE may be used even for text files, and it may be easier to specify a file's name once and later refer to it by an abstract handle.

Specify the file handle name as the identifier immediately following the FILE HANDLE command name. The identifier INLINE is reserved for representing data embedded in the

syntax file (see Section 7.1 [BEGIN DATA], page 128) The file handle name must not already have been used in a previous invocation of FILE HANDLE, unless it has been closed by an intervening command (see Section 7.2 [CLOSE FILE HANDLE], page 128).

The effect and syntax of FILE HANDLE depends on the selected MODE:

- In CHARACTER mode, the default, the data file is read as a text file, according to the local system's conventions, and each text line is read as one record.

  In CHARACTER mode only, tabs are expanded to spaces by input programs, except by DATA LIST FREE with explicitly specified delimiters. Each tab is 4 characters wide by default, but TABWIDTH (a PSPP extension) may be used to specify an alternate width. Use a TABWIDTH of 0 to suppress tab expansion.

- In IMAGE mode, the data file is treated as a series of fixed-length binary records. LRECL should be used to specify the record length in bytes, with a default of 1024. On input, it is an error if an IMAGE file's length is not a integer multiple of the record length. On output, each record is padded with spaces or truncated, if necessary, to make it exactly the correct length.

- In BINARY mode, the data file is treated as a series of variable-length binary records. LRECL may be specified, but its value is ignored. The data for each record is both preceded and followed by a 32-bit signed integer in little-endian byte order that specifies the length of the record. (This redundancy permits records in these files to be efficiently read in reverse order, although PSPP always reads them in forward order.) The length does not include either integer.

- Mode 360 reads and writes files in formats first used for tapes in the 1960s on IBM mainframe operating systems and still supported today by the modern successors of those operating systems. For more information, see *OS/400 Tape and Diskette Device Programming*, available on IBM's website.

  Alphanumeric data in mode 360 files are encoded in EBCDIC. PSPP translates EBCDIC to or from the host's native format as necessary on input or output, using an ASCII/EBCDIC translation that is one-to-one, so that a "round trip" from ASCII to EBCDIC back to ASCII, or vice versa, always yields exactly the original data.

  The RECFORM subcommand is required in mode 360. The precise file format depends on its setting:

F
FIXED   This record format is equivalent to IMAGE mode, except for EBCDIC translation.

          IBM documentation calls this `*F` (fixed-length, deblocked) format.

V
VARIABLE
          The file comprises a sequence of zero or more variable-length blocks. Each block begins with a 4-byte *block descriptor word* (BDW). The first two bytes of the BDW are an unsigned integer in big-endian byte order that specifies the length of the block, including the BDW itself. The other two bytes of the BDW are ignored on input and written as zeros on output.

          Following the BDW, the remainder of each block is a sequence of one or more variable-length records, each of which in turn begins with a 4-byte

record descriptor word (RDW) that has the same format as the BDW. Following the RDW, the remainder of each record is the record data.

The maximum length of a record in VARIABLE mode is 65,527 bytes: 65,535 bytes (the maximum value of a 16-bit unsigned integer), minus 4 bytes for the BDW, minus 4 bytes for the RDW.

In mode VARIABLE, LRECL specifies a maximum, not a fixed, record length, in bytes. The default is 8,192.

IBM documentation calls this `*VB` (variable-length, blocked, unspanned) format.

VS
SPANNED

The file format is like that of VARIABLE mode, except that logical records may be split among multiple physical records (called *segments*) or blocks. In SPANNED mode, the third byte of each RDW is called the segment control character (SCC). Odd SCC values cause the segment to be appended to a record buffer maintained in memory; even values also append the segment and then flush its contents to the input procedure. Canonically, SCC value 0 designates a record not spanned among multiple segments, and values 1 through 3 designate the first segment, the last segment, or an intermediate segment, respectively, within a multi-segment record. The record buffer is also flushed at end of file regardless of the final record's SCC.

The maximum length of a logical record in VARIABLE mode is limited only by memory available to PSPP. Segments are limited to 65,527 bytes, as in VARIABLE mode.

This format is similar to what IBM documentation call `*VS` (variable-length, deblocked, spanned) format.

In mode 360, fields of type A that extend beyond the end of a record read from disk are padded with spaces in the host's native character set, which are then translated from EBCDIC to the native character set. Thus, when the host's native character set is based on ASCII, these fields are effectively padded with character `X'80'`. This wart is implemented for compatibility.

- SCRATCH mode is a PSPP extension that designates the file handle as a scratch file handle. Its use is usually unnecessary because file handle names that begin with '`#`' are assumed to refer to scratch files. see Section 5.9 [File Handles], page 108, for more information.

The NAME subcommand specifies the name of the file associated with the handle. It is required in all modes but SCRATCH mode, in which its use is forbidden.

7.7 INPUT PROGRAM

```
INPUT PROGRAM.
    . . . input commands . . .
END INPUT PROGRAM.
```

INPUT PROGRAM…END INPUT PROGRAM specifies a complex input program. By placing data input commands within INPUT PROGRAM, PSPP programs can take advantage of more complex file structures than available with only DATA LIST.

The first sort of extended input program is to simply put multiple DATA LIST commands within the INPUT PROGRAM. This will cause all of the data files to be read in parallel. Input will stop when end of file is reached on any of the data files.

Transformations, such as conditional and looping constructs, can also be included within INPUT PROGRAM. These can be used to combine input from several data files in more complex ways. However, input will still stop when end of file is reached on any of the data files.

To prevent INPUT PROGRAM from terminating at the first end of file, use the END subcommand on DATA LIST. This subcommand takes a variable name, which should be a numeric scratch variable (see Section 5.7.5 [Scratch Variables], page 107). (It need not be a scratch variable but otherwise the results can be surprising.) The value of this variable is set to 0 when reading the data file, or 1 when end of file is encountered.

Two additional commands are useful in conjunction with INPUT PROGRAM. END CASE is the first. Normally each loop through the INPUT PROGRAM structure produces one case. END CASE controls exactly when cases are output. When END CASE is used, looping from the end of INPUT PROGRAM to the beginning does not cause a case to be output.

END FILE is the second. When the END subcommand is used on DATA LIST, there is no way for the INPUT PROGRAM construct to stop looping, so an infinite loop results. END FILE, when executed, stops the flow of input data and passes out of the INPUT PROGRAM structure.

All this is very confusing. A few examples should help to clarify.

```
INPUT PROGRAM.
        DATA LIST NOTABLE FILE='a.data'/X 1-10.
        DATA LIST NOTABLE FILE='b.data'/Y 1-10.
END INPUT PROGRAM.
LIST.
```

The example above reads variable X from file 'a.data' and variable Y from file 'b.data'. If one file is shorter than the other then the extra data in the longer file is ignored.

```
INPUT PROGRAM.
        NUMERIC #A #B.

        DO IF NOT #A.
                DATA LIST NOTABLE END=#A FILE='a.data'/X 1-10.
        END IF.
        DO IF NOT #B.
                DATA LIST NOTABLE END=#B FILE='b.data'/Y 1-10.
        END IF.
        DO IF #A AND #B.
                END FILE.
        END IF.
        END CASE.
```

```
    END INPUT PROGRAM.
    LIST.
```

The above example reads variable X from 'a.data' and variable Y from 'b.data'. If
one file is shorter than the other then the missing field is set to the system-missing value
alongside the present value for the remaining length of the longer file.

```
    INPUT PROGRAM.
            NUMERIC #A #B.

            DO IF #A.
                    DATA LIST NOTABLE END=#B FILE='b.data'/X 1-10.
                    DO IF #B.
                            END FILE.
                    ELSE.
                            END CASE.
                    END IF.
            ELSE.
                    DATA LIST NOTABLE END=#A FILE='a.data'/X 1-10.
                    DO IF NOT #A.
                            END CASE.
                    END IF.
            END IF.
    END INPUT PROGRAM.
    LIST.
```

The above example reads data from file 'a.data', then from 'b.data', and concatenates
them into a single active file.

```
    INPUT PROGRAM.
            NUMERIC #EOF.

            LOOP IF NOT #EOF.
                    DATA LIST NOTABLE END=#EOF FILE='a.data'/X 1-10.
                    DO IF NOT #EOF.
                            END CASE.
                    END IF.
            END LOOP.

            COMPUTE #EOF = 0.
            LOOP IF NOT #EOF.
                    DATA LIST NOTABLE END=#EOF FILE='b.data'/X 1-10.
                    DO IF NOT #EOF.
                            END CASE.
                    END IF.
            END LOOP.

            END FILE.
    END INPUT PROGRAM.
    LIST.
```

The above example does the same thing as the previous example, in a different way.

```
INPUT PROGRAM.
        LOOP #I=1 TO 50.
                COMPUTE X=UNIFORM(10).
                END CASE.
        END LOOP.
        END FILE.
END INPUT PROGRAM.
LIST/FORMAT=NUMBERED.
```

The above example causes an active file to be created consisting of 50 random variates between 0 and 10.

7.8 LIST

```
LIST
        /VARIABLES=var_list
        /CASES=FROM start_index TO end_index BY incr_index
        /FORMAT={UNNUMBERED,NUMBERED} {WRAP,SINGLE}
                {NOWEIGHT,WEIGHT}
```

The LIST procedure prints the values of specified variables to the listing file.

The VARIABLES subcommand specifies the variables whose values are to be printed. Keyword VARIABLES is optional. If VARIABLES subcommand is not specified then all variables in the active file are printed.

The CASES subcommand can be used to specify a subset of cases to be printed. Specify FROM and the case number of the first case to print, TO and the case number of the last case to print, and BY and the number of cases to advance between printing cases, or any subset of those settings. If CASES is not specified then all cases are printed.

The FORMAT subcommand can be used to change the output format. NUMBERED will print case numbers along with each case; UNNUMBERED, the default, causes the case numbers to be omitted. The WRAP and SINGLE settings are currently not used. WEIGHT will cause case weights to be printed along with variable values; NOWEIGHT, the default, causes case weights to be omitted from the output.

Case numbers start from 1. They are counted after all transformations have been considered.

LIST attempts to fit all the values on a single line. If needed to make them fit, variable names are displayed vertically. If values cannot fit on a single line, then a multi-line format will be used.

LIST is a procedure. It causes the data to be read.

7.9 NEW FILE

```
NEW FILE.
```

NEW FILE command clears the current active file.

7.10 PRINT

```
PRINT
        OUTFILE='file-name'
        RECORDS=n_lines
        {NOTABLE,TABLE}
        [/[line_no] arg...]
```

arg takes one of the following forms:
```
        'string' [start-end]
        var_list start-end [type_spec]
        var_list (fortran_spec)
        var_list *
```

The PRINT transformation writes variable data to the listing file or an output file. PRINT is executed when a procedure causes the data to be read. Follow PRINT by EXE-CUTE to print variable data without invoking a procedure (see Section 14.10 [EXECUTE], page 191).

All PRINT subcommands are optional. If no strings or variables are specified, PRINT outputs a single blank line.

The OUTFILE subcommand specifies the file to receive the output. The file may be a file name as a string or a file handle (see Section 5.9 [File Handles], page 108). If OUTFILE is not present then output will be sent to PSPP's output listing file. When OUTFILE is present, a space is inserted at beginning of each output line, even lines that otherwise would be blank.

The RECORDS subcommand specifies the number of lines to be output. The number of lines may optionally be surrounded by parentheses.

TABLE will cause the PRINT command to output a table to the listing file that describes what it will print to the output file. NOTABLE, the default, suppresses this output table.

Introduce the strings and variables to be printed with a slash ('/'). Optionally, the slash may be followed by a number indicating which output line will be specified. In the absence of this line number, the next line number will be specified. Multiple lines may be specified using multiple slashes with the intended output for a line following its respective slash.

Literal strings may be printed. Specify the string itself. Optionally the string may be followed by a column number or range of column numbers, specifying the location on the line for the string to be printed. Otherwise, the string will be printed at the current position on the line.

Variables to be printed can be specified in the same ways as available for DATA LIST FIXED (see Section 7.3.1 [DATA LIST FIXED], page 129). In addition, a variable list may be followed by an asterisk ('*'), which indicates that the variables should be printed in their dictionary print formats, separated by spaces. A variable list followed by a slash or the end of command will be interpreted the same way.

If a FORTRAN type specification is used to move backwards on the current line, then text is written at that point on the line, the line will be truncated to that length, although additional text being added will again extend the line to that length.

7.11 PRINT EJECT

```
PRINT EJECT
        OUTFILE='file-name'
        RECORDS=n_lines
        {NOTABLE,TABLE}
        /[line_no] arg...

arg takes one of the following forms:
        'string' [start-end]
        var_list start-end [type_spec]
        var_list (fortran_spec)
        var_list *
```

PRINT EJECT advances to the beginning of a new output page in the listing file or output file. It can also output data in the same way as PRINT.

All PRINT EJECT subcommands are optional.

Without OUTFILE, PRINT EJECT ejects the current page in the listing file, then it produces other output, if any is specified.

With OUTFILE, PRINT EJECT writes its output to the specified file. The first line of output is written with '1' inserted in the first column. Commonly, this is the only line of output. If additional lines of output are specified, these additional lines are written with a space inserted in the first column, as with PRINT.

See Section 7.10 [PRINT], page 139, for more information on syntax and usage.

7.12 PRINT SPACE

```
PRINT SPACE OUTFILE='file-name' n_lines.
```

PRINT SPACE prints one or more blank lines to an output file.

The OUTFILE subcommand is optional. It may be used to direct output to a file specified by file name as a string or file handle (see Section 5.9 [File Handles], page 108). If OUTFILE is not specified then output will be directed to the listing file.

n_lines is also optional. If present, it is an expression (see Chapter 6 [Expressions], page 110) specifying the number of blank lines to be printed. The expression must evaluate to a nonnegative value.

7.13 REREAD

```
REREAD FILE=handle COLUMN=column.
```

The REREAD transformation allows the previous input line in a data file already processed by DATA LIST or another input command to be re-read for further processing.

The FILE subcommand, which is optional, is used to specify the file to have its line re-read. The file must be specified as the name of a file handle (see Section 5.9 [File Handles], page 108). If FILE is not specified then the last file specified on DATA LIST will be assumed (last file specified lexically, not in terms of flow-of-control).

By default, the line re-read is re-read in its entirety. With the COLUMN subcommand, a prefix of the line can be exempted from re-reading. Specify an expression (see Chapter 6

[Expressions], page 110) evaluating to the first column that should be included in the re-read line. Columns are numbered from 1 at the left margin.

Issuing REREAD multiple times will not back up in the data file. Instead, it will re-read the same line multiple times.

7.14 REPEATING DATA

    REPEATING DATA
            /STARTS=start-end
            /OCCURS=n_occurs
            /FILE='file-name'
            /LENGTH=length
            /CONTINUED[=cont_start-cont_end]
            /ID=id_start-id_end=id_var
            /{TABLE,NOTABLE}
            /DATA=var_spec. . .

    where each var_spec takes one of the forms
            var_list start-end [type_spec]
            var_list (fortran_spec)

REPEATING DATA parses groups of data repeating in a uniform format, possibly with several groups on a single line. Each group of data corresponds with one case. REPEATING DATA may only be used within an INPUT PROGRAM structure (see Section 7.7 [INPUT PROGRAM], page 135). When used with DATA LIST, it can be used to parse groups of cases that share a subset of variables but differ in their other data.

The STARTS subcommand is required. Specify a range of columns, using literal numbers or numeric variable names. This range specifies the columns on the first line that are used to contain groups of data. The ending column is optional. If it is not specified, then the record width of the input file is used. For the inline file (see Section 7.1 [BEGIN DATA], page 128) this is 80 columns; for a file with fixed record widths it is the record width; for other files it is 1024 characters by default.

The OCCURS subcommand is required. It must be a number or the name of a numeric variable. Its value is the number of groups present in the current record.

The DATA subcommand is required. It must be the last subcommand specified. It is used to specify the data present within each repeating group. Column numbers are specified relative to the beginning of a group at column 1. Data is specified in the same way as with DATA LIST FIXED (see Section 7.3.1 [DATA LIST FIXED], page 129).

All other subcommands are optional.

FILE specifies the file to read, either a file name as a string or a file handle (see Section 5.9 [File Handles], page 108). If FILE is not present then the default is the last file handle used on DATA LIST (lexically, not in terms of flow of control).

By default REPEATING DATA will output a table describing how it will parse the input data. Specifying NOTABLE will disable this behavior; specifying TABLE will explicitly enable it.

The LENGTH subcommand specifies the length in characters of each group. If it is not present then length is inferred from the DATA subcommand. LENGTH can be a number or a variable name.

Normally all the data groups are expected to be present on a single line. Use the CONTINUED command to indicate that data can be continued onto additional lines. If data on continuation lines starts at the left margin and continues through the entire field width, no column specifications are necessary on CONTINUED. Otherwise, specify the possible range of columns in the same way as on STARTS.

When data groups are continued from line to line, it is easy for cases to get out of sync through careless hand editing. The ID subcommand allows a case identifier to be present on each line of repeating data groups. REPEATING DATA will check for the same identifier on each line and report mismatches. Specify the range of columns that the identifier will occupy, followed by an equals sign ('=') and the identifier variable name. The variable must already have been declared with NUMERIC or another command.

REPEATING DATA should be the last command given within an INPUT PROGRAM. It should not be enclosed within a LOOP structure (see Section 12.4 [LOOP], page 175). Use DATA LIST before, not after, REPEATING DATA.

7.15 WRITE

```
WRITE
        OUTFILE='file-name'
        RECORDS=n_lines
        {NOTABLE,TABLE}
        /[line_no] arg. . .

arg takes one of the following forms:
        'string' [start-end]
        var_list start-end [type_spec]
        var_list (fortran_spec)
        var_list *
```

WRITE writes text or binary data to an output file.

See Section 7.10 [PRINT], page 139, for more information on syntax and usage. PRINT and WRITE differ in only a few ways:

- WRITE uses write formats by default, whereas PRINT uses print formats.
- PRINT inserts a space between variables unless a format is explicitly specified, but WRITE never inserts space between variables in output.
- PRINT inserts a space at the beginning of each line that it writes to an output file (and PRINT EJECT inserts '1' at the beginning of each line that should begin a new page), but WRITE does not.
- PRINT outputs the system-missing value according to its specified output format, whereas WRITE outputs the system-missing value as a field filled with spaces. Binary formats are an exception.

8 System Files and Portable Files

The commands in this chapter read, write, and examine system files and portable files.

8.1 APPLY DICTIONARY

APPLY DICTIONARY FROM={'file-name',file_handle}.

APPLY DICTIONARY applies the variable labels, value labels, and missing values taken from a file to corresponding variables in the active file. In some cases it also updates the weighting variable.

Specify a system file, portable file, or scratch file with a file name string or as a file handle (see Section 5.9 [File Handles], page 108). The dictionary in the file will be read, but it will not replace the active file dictionary. The file's data will not be read.

Only variables with names that exist in both the active file and the system file are considered. Variables with the same name but different types (numeric, string) will cause an error message. Otherwise, the system file variables' attributes will replace those in their matching active file variables, as described below.

If a system file variable has a variable label, then it will replace the active file variable's variable label. If the system file variable does not have a variable label, then the active file variable's variable label, if any, will be retained.

If the active file variable is numeric or short string, then value labels and missing values, if any, will be copied to the active file variable. If the system file variable does not have value labels or missing values, then those in the active file variable, if any, will not be disturbed.

Finally, weighting of the active file is updated (see Section 11.7 [WEIGHT], page 173). If the active file has a weighting variable, and the system file does not, or if the weighting variable in the system file does not exist in the active file, then the active file weighting variable, if any, is retained. Otherwise, the weighting variable in the system file becomes the active file weighting variable.

APPLY DICTIONARY takes effect immediately. It does not read the active file. The system file is not modified.

8.2 EXPORT

```
EXPORT
        /OUTFILE='file-name'
        /UNSELECTED={RETAIN,DELETE}
        /DIGITS=n
        /DROP=var_list
        /KEEP=var_list
        /RENAME=(src_names=target_names)...
        /TYPE={COMM,TAPE}
        /MAP
```

The EXPORT procedure writes the active file dictionary and data to a specified portable file or scratch file.

By default, cases excluded with FILTER are written to the file. These can be excluded by specifying DELETE on the UNSELECTED subcommand. Specifying RETAIN makes the default explicit.

Portable files express real numbers in base 30. Integers are always expressed to the maximum precision needed to make them exact. Non-integers are, by default, expressed to the machine's maximum natural precision (approximately 15 decimal digits on many machines). If many numbers require this many digits, the portable file may significantly increase in size. As an alternative, the DIGITS subcommand may be used to specify the number of decimal digits of precision to write. DIGITS applies only to non-integers.

The OUTFILE subcommand, which is the only required subcommand, specifies the portable file or scratch file to be written as a file name string or a file handle (see Section 5.9 [File Handles], page 108).

DROP, KEEP, and RENAME follow the same format as the SAVE procedure (see Section 8.7 [SAVE], page 152).

The TYPE subcommand specifies the character set for use in the portable file. Its value is currently not used.

The MAP subcommand is currently ignored.

EXPORT is a procedure. It causes the active file to be read.

8.3 GET

```
GET
        /FILE={'file-name',file_handle}
        /DROP=var_list
        /KEEP=var_list
        /RENAME=(src_names=target_names)...
```

GET clears the current dictionary and active file and replaces them with the dictionary and data from a specified file.

The FILE subcommand is the only required subcommand. Specify the system file, portable file, or scratch file to be read as a string file name or a file handle (see Section 5.9 [File Handles], page 108).

By default, all the variables in a file are read. The DROP subcommand can be used to specify a list of variables that are not to be read. By contrast, the KEEP subcommand can be used to specify variable that are to be read, with all other variables not read.

Normally variables in a file retain the names that they were saved under. Use the RENAME subcommand to change these names. Specify, within parentheses, a list of variable names followed by an equals sign ('=') and the names that they should be renamed to. Multiple parenthesized groups of variable names can be included on a single RENAME subcommand. Variables' names may be swapped using a RENAME subcommand of the form '/RENAME=(A B=B A)'.

Alternate syntax for the RENAME subcommand allows the parentheses to be eliminated. When this is done, only a single variable may be renamed at once. For instance, '/RENAME=A=B'. This alternate syntax is deprecated.

DROP, KEEP, and RENAME are executed in left-to-right order. Each may be present any number of times. GET never modifies a file on disk. Only the active file read from the file is affected by these subcommands.

GET does not cause the data to be read, only the dictionary. The data is read later, when a procedure is executed.

Use of GET to read a portable file or scratch file is a PSPP extension.

8.4 GET DATA

        GET DATA
            /TYPE={GNM,PSQL,TXT}
            . . . additional subcommands depending on TYPE. . .

The GET DATA command is used to read files and other data sources created by other applications. When this command is executed, the current dictionary and active file are replaced with variables and data read from the specified source.

The TYPE subcommand is mandatory and must be the first subcommand specified. It determines the type of the file or source to read. PSPP currently supports the following file types:

GNM   Spreadsheet files created by Gnumeric (http://gnumeric.org).

PSQL   Relations from PostgreSQL databases (http://postgresql.org).

TXT   Textual data files in columnar and delimited formats.

Each supported file type has additional subcommands, explained in separate sections below.

8.4.1 Gnumeric Spreadsheet Files

        GET DATA /TYPE=GNM
            /FILE={'file-name'}
            /SHEET={NAME 'sheet-name', INDEX n}
            /CELLRANGE={RANGE 'range', FULL}
            /READNAMES={ON, OFF}
            /ASSUMEDVARWIDTH=n.

To use GET DATA to read a spreadsheet file created by Gnumeric (http://gnumeric. org), specify TYPE=GNM to indicate the file's format and use FILE to indicate the Gnumeric file to be read. All other subcommands are optional.

The format of each variable is determined by the format of the spreadsheet cell containing the first datum for the variable. If this cell is of string (text) format, then the width of the variable is determined from the length of the string it contains, unless the ASSUMEDVARWIDTH subcommand is given.

The FILE subcommand is mandatory. Specify the name of the file to be read.

The SHEET subcommand specifies the sheet within the spreadsheet file to read. There are two forms of the SHEET subcommand. In the first form, '/SHEET=name *sheet-name*', the string *sheet-name* is the name of the sheet to read. In the second form, '/SHEET=index *idx*', *idx* is a integer which is the index of the sheet to read. The first sheet has the index

1. If the SHEET subcommand is omitted, then the command will read the first sheet in the file.

The CELLRANGE subcommand specifies the range of cells within the sheet to read. If the subcommand is given as '/CELLRANGE=FULL', then the entire sheet is read. To read only part of a sheet, use the form '/CELLRANGE=range '*top-left-cell*:*bottom-right-cell*''. For example, the subcommand '/CELLRANGE=range 'C3:P19'' reads columns C–P, and rows 3–19 inclusive. If no CELLRANGE subcommand is given, then the entire sheet is read.

If '/READNAMES=ON' is specified, then the contents of cells of the first row are used as the names of the variables in which to store the data from subsequent rows. If the READNAMES command is omitted, or if '/READNAMES=OFF' is used, then the variables receive automatically assigned names.

The ASSUMEDVARWIDTH subcommand specifies the maximum width of string variables read from the file. If omitted, the default value is determined from the length of the string in the first spreadsheet cell for each variable.

8.4.2 Postgres Database Queries

```
GET DATA /TYPE=PSQL
        /CONNECT={connection info}
        /SQL={query}
        [/ASSUMEDVARWIDTH=n]
        [/UNENCRYPTED]
        [/BSIZE=n].
```

The PSQL type is used to import data from a postgres database server. The server may be located locally or remotely. Variables are automatically created based on the table column names or the names specified in the SQL query. Postgres data types of high precision, will loose precision when imported into PSPP. Not all the postgres data types are able to be represented in PSPP. If a datum cannot be represented a warning will be issued and that datum will be set to SYSMIS.

The CONNECT subcommand is mandatory. It is a string specifying the parameters of the database server from which the data should be fetched. The format of the string is given in the postgres manual http://www.postgresql.org/docs/8.0/static/libpq.html#LIBPQ-CONNECT.

The SQL subcommand is mandatory. It must be a valid SQL string to retrieve data from the database.

The ASSUMEDVARWIDTH subcommand specifies the maximum width of string variables read from the database. If omitted, the default value is determined from the length of the string in the first value read for each variable.

The UNENCRYPTED subcommand allows data to be retrieved over an insecure connection. If the connection is not encrypted, and the UNENCRYPTED subcommand is not given, then an error will occur. Whether or not the connection is encrypted depends upon the underlying psql library and the capabilities of the database server.

The BSIZE subcommand serves only to optimise the speed of data transfer. It specifies an upper limit on number of cases to fetch from the database at once. The default value is 4096. If your SQL statement fetches a large number of cases but only a small number of

variables, then the data transfer may be faster if you increase this value. Conversely, if the number of variables is large, or if the machine on which PSPP is running has only a small amount of memory, then a smaller value will be better.

The following syntax is an example:

```
GET DATA /TYPE=PSQL
    /CONNECT='host=example.com port=5432 dbname=product user=fred passwd=xxxx'
    /SQL='select * from manufacturer'.
```

8.4.3 Textual Data Files

```
GET DATA /TYPE=TXT
    /FILE={'file-name',file_handle}
    [/ARRANGEMENT={DELIMITED,FIXED}]
    [/FIRSTCASE={first_case}]
    [/IMPORTCASE={ALL,FIRST max_cases,PERCENT percent}]
    ...additional subcommands depending on ARRANGEMENT...
```

When TYPE=TXT is specified, GET DATA reads data in a delimited or fixed columnar format, much like DATA LIST (see Section 7.3 [DATA LIST], page 128).

The FILE subcommand is mandatory. Specify the file to be read as a string file name or (for textual data only) a file handle (see Section 5.9 [File Handles], page 108).

The ARRANGEMENT subcommand determines the file's basic format. DELIMITED, the default setting, specifies that fields in the input data are separated by spaces, tabs, or other user-specified delimiters. FIXED specifies that fields in the input data appear at particular fixed column positions within records of a case.

By default, cases are read from the input file starting from the first line. To skip lines at the beginning of an input file, set FIRSTCASE to the number of the first line to read: 2 to skip the first line, 3 to skip the first two lines, and so on.

IMPORTCASE can be used to limit the number of cases read from the input file. With the default setting, ALL, all cases in the file are read. Specify FIRST max_cases to read at most max_cases cases from the file. Use PERCENT $percent$ to read only $percent$ percent, approximately, of the cases contained in the file. (The percentage is approximate, because there is no way to accurately count the number of cases in the file without reading the entire file. The number of cases in some kinds of unusual files cannot be estimated; PSPP will read all cases in such files.)

FIRSTCASE and IMPORTCASE may be used with delimited and fixed-format data. The remaining subcommands, which apply only to one of the two file arrangements, are described below.

8.4.3.1 Reading Delimited Data

```
GET DATA /TYPE=TXT
    /FILE={'file-name',file_handle}
    [/ARRANGEMENT={DELIMITED,FIXED}]
    [/FIRSTCASE={first_case}]
    [/IMPORTCASE={ALL,FIRST max_cases,PERCENT percent}]

    /DELIMITERS="delimiters"
```

[/QUALIFIER="quotes" [/ESCAPE]]
[/DELCASE={LINE,VARIABLES n_variables}]
/VARIABLES=del_var [del_var]. . .
where each del_var takes the form:
variable format

The GET DATA command with TYPE=TXT and ARRANGEMENT=DELIMITED reads input data from text files in delimited format, where fields are separated by a set of user-specified delimiters. Its capabilities are similar to those of DATA LIST FREE (see Section 7.3.2 [DATA LIST FREE], page 131), with a few enhancements.

The required FILE subcommand and optional FIRSTCASE and IMPORTCASE subcommands are described above (see Section 8.4.3 [GET DATA /TYPE=TXT], page 147).

DELIMITERS, which is required, specifies the set of characters that may separate fields. Each character in the string specified on DELIMITERS separates one field from the next. The end of a line also separates fields, regardless of DELIMITERS. Two consecutive delimiters in the input yield an empty field, as does a delimiter at the end of a line. A space character as a delimiter is an exception: consecutive spaces do not yield an empty field and neither does any number of spaces at the end of a line.

To use a tab as a delimiter, specify '\t' at the beginning of the DELIMITERS string. To use a backslash as a delimiter, specify '\\' as the first delimiter or, if a tab should also be a delimiter, immediately following '\t'. To read a data file in which each field appears on a separate line, specify the empty string for DELIMITERS.

The optional QUALIFIER subcommand names one or more characters that can be used to quote values within fields in the input. A field that begins with one of the specified quote characters ends at the next matching quote. Intervening delimiters become part of the field, instead of terminating it. The ability to specify more than one quote character is a PSPP extension.

By default, a character specified on QUALIFIER cannot itself be embedded within a field that it quotes, because the quote character always terminates the quoted field. With ESCAPE, however, a doubled quote character within a quoted field inserts a single instance of the quote into the field. For example, if ''' is specified on QUALIFIER, then without ESCAPE 'a''b' specifies a pair of fields that contain 'a' and 'b', but with ESCAPE it specifies a single field that contains 'a'b'. ESCAPE is a PSPP extension.

The DELCASE subcommand controls how data may be broken across lines in the data file. With LINE, the default setting, each line must contain all the data for exactly one case. For additional flexibility, to allow a single case to be split among lines or multiple cases to be contained on a single line, specify VARIABLES *n_variables*, where *n_variables* is the number of variables per case.

The VARIABLES subcommand is required and must be the last subcommand. Specify the name of each variable and its input format (see Section 5.7.4 [Input and Output Formats], page 98) in the order they should be read from the input file.

Examples

On a Unix-like system, the '/etc/passwd' file has a format similar to this:

```
root:$1$nyeSP5gD$pDq/:0:0:,,,:/root:/bin/bash
blp:$1$BrP/pFg4$g7OG:1000:1000:Ben Pfaff,,,:/home/blp:/bin/bash
```

```
john:$1$JBuq/Fioq$g4A:1001:1001:John Darrington,,,:/home/john:/bin/bash
jhs:$1$D3li4hPL$88X1:1002:1002:Jason Stover,,,:/home/jhs:/bin/csh
```

The following syntax reads a file in the format used by '/etc/passwd':

```
GET DATA /TYPE=TXT /FILE='/etc/passwd' /DELIMITERS=':'
        /VARIABLES=username A20
                   password A40
                   uid F10
                   gid F10
                   gecos A40
                   home A40
                   shell A40.
```

Consider the following data on used cars:

```
model    year    mileage price    type    age
Civic    2002    29883   15900    Si      2
Civic    2003    13415   15900    EX      1
Civic    1992    107000  3800     n/a     12
Accord   2002    26613   17900    EX      1
```

The following syntax can be used to read the used car data:

```
GET DATA /TYPE=TXT /FILE='cars.data' /DELIMITERS=' ' /FIRSTCASE=2
        /VARIABLES=model A8
                   year F4
                   mileage F6
                   price F5
                   type A4
                   age F2.
```

Consider the following information on animals in a pet store:

```
'Pet''s Name', "Age", "Color", "Date Received", "Price", "Height", "Type"
, (Years), , , (Dollars), ,
"Rover", 4.5, Brown, "12 Feb 2004", 80, '1''4"', "Dog"
"Charlie", , Gold, "5 Apr 2007", 12.3, "3"""", "Fish"
"Molly", 2, Black, "12 Dec 2006", 25, '5"', "Cat"
"Gilly", , White, "10 Apr 2007", 10, "3"""", "Guinea Pig"
```

The following syntax can be used to read the pet store data:

```
GET DATA /TYPE=TXT /FILE='pets.data' /DELIMITERS=', ' /QUALIFIER='''"' /ESCAPE
        /FIRSTCASE=3
        /VARIABLES=name A10
                   age F3.1
                   color A5
                   received EDATE10
                   price F5.2
                   height a5
                   type a10.
```

8.4.3.2 Reading Fixed Columnar Data

```
GET DATA /TYPE=TXT
```

        /FILE={'file-name',file_handle}
        [/ARRANGEMENT={DELIMITED,FIXED}]
        [/FIRSTCASE={first_case}]
        [/IMPORTCASE={ALL,FIRST max_cases,PERCENT percent}]

        [/FIXCASE=n]
        /VARIABLES fixed_var [fixed_var]. . .
            [/rec# fixed_var [fixed_var]. . .]. . .
    where each fixed_var takes the form:
        variable start-end format

The GET DATA command with TYPE=TXT and ARRANGEMENT=FIXED reads input data from text files in fixed format, where each field is located in particular fixed column positions within records of a case. Its capabilities are similar to those of DATA LIST FIXED (see Section 7.3.1 [DATA LIST FIXED], page 129), with a few enhancements.

The required FILE subcommand and optional FIRSTCASE and IMPORTCASE subcommands are described above (see Section 8.4.3 [GET DATA /TYPE=TXT], page 147).

The optional FIXCASE subcommand may be used to specify the positive integer number of input lines that make up each case. The default value is 1.

The VARIABLES subcommand, which is required, specifies the positions at which each variable can be found. For each variable, specify its name, followed by its start and end column separated by '-' (e.g. '0-9'), followed by the input format type (e.g. 'F'). For this command, columns are numbered starting from 0 at the left column. Introduce the variables in the second and later lines of a case by a slash followed by the number of the line within the case, e.g. '/2' for the second line.

Examples

Consider the following data on used cars:

```
model   year    mileage price   type    age
Civic   2002    29883   15900   Si      2
Civic   2003    13415   15900   EX      1
Civic   1992    107000  3800    n/a     12
Accord  2002    26613   17900   EX      1
```

The following syntax can be used to read the used car data:

```
GET DATA /TYPE=TXT /FILE='cars.data' /ARRANGEMENT=FIXED /FIRSTCASE=2
        /VARIABLES=model 0-7 A
                    year 8-15 F
                    mileage 16-23 F
                    price 24-31 F
                    type 32-40 A
                    age 40-47 F.
```

8.5 IMPORT

        IMPORT
            /FILE='file-name'
            /TYPE={COMM,TAPE}

> /DROP=var_list
> /KEEP=var_list
> /RENAME=(src_names=target_names). . .

The IMPORT transformation clears the active file dictionary and data and replaces them with a dictionary and data from a system, portable file, or scratch file.

The FILE subcommand, which is the only required subcommand, specifies the portable file to be read as a file name string or a file handle (see Section 5.9 [File Handles], page 108).

The TYPE subcommand is currently not used.

DROP, KEEP, and RENAME follow the syntax used by GET (see Section 8.3 [GET], page 144).

IMPORT does not cause the data to be read, only the dictionary. The data is read later, when a procedure is executed.

Use of IMPORT to read a system file or scratch file is a PSPP extension.

8.6 MATCH FILES

> MATCH FILES
>   /{FILE,TABLE}={*,'file-name'}
>   /RENAME=(src_names=target_names). . .
>   /IN=var_name
>
>   /BY=var_list
>   /DROP=var_list
>   /KEEP=var_list
>   /FIRST=var_name
>   /LAST=var_name
>   /MAP

MATCH FILES merges one or more system, portable, or scratch files, optionally including the active file. Cases with the same values for BY variables are combined into a single case. Cases with different values are output in order. Thus, multiple sorted files are combined into a single sorted file based on the value of the BY variables. The results of the merge become the new active file.

Specify FILE with a system, portable, or scratch file as a file name string or file handle (see Section 5.9 [File Handles], page 108), or with an asterisk ('*') to indicate the current active file. The files specified on FILE are merged together based on the BY variables, or combined case-by-case if BY is not specified.

Specify TABLE with a file to use it as a *table lookup file*. Cases in table lookup files are not used up after they've been used once. This means that data in table lookup files can correspond to any number of cases in FILE files. Table lookup files correspond to lookup tables in traditional relational database systems. If a table lookup file contains more than one case with a given set of BY variables, only the first case is used.

Any number of FILE and TABLE subcommands may be specified. Ordinarily, at least two FILE subcommands, or one FILE and at least one TABLE, should be specified. Each instance of FILE or TABLE can be followed by any sequence of RENAME subcommands.

These have the same form and meaning as the corresponding subcommands of GET (see Section 8.3 [GET], page 144), but apply only to variables in the given file.

Each FILE or TABLE may optionally be followed by an IN subcommand, which creates a numeric variable with the specified name and format F1.0. The IN variable takes value 1 in a case if the given file contributed a row to the merged file, 0 otherwise. The DROP, KEEP, and RENAME subcommands do not affect IN variables.

When more than one FILE or TABLE contains a variable with a given name, those variables must all have the same type (numeric or string) and, for string variables, the same width. This rules applies to variable names after renaming with RENAME; thus, RENAME can be used to resolve conflicts.

FILE and TABLE must be specified at the beginning of the command, with any RE-NAME or IN specifications immediately after the corresponding FILE or TABLE. These subcommands are followed by BY, DROP, KEEP, FIRST, LAST, and MAP.

The BY subcommand specifies a list of variables that are used to match cases from each of the files. When TABLE or IN is used, BY is required; otherwise, it is optional. When BY is specified, all the files named on FILE and TABLE subcommands must be sorted in ascending order of the BY variables. Variables belonging to files that are not present for the current case are set to the system-missing value for numeric variables or spaces for string variables.

The DROP and KEEP subcommands allow variables to be dropped from or reordered within the new active file. These subcommands have the same form and meaning as the corresponding subcommands of GET (see Section 8.3 [GET], page 144). They apply to the new active file as a whole, not to individual input files. The variable names specified on DROP and KEEP are those after any renaming with RENAME.

The optional FIRST and LAST subcommands name variables that MATCH FILES adds to the active file. The new variables are numeric with print and write format F1.0. The value of the FIRST variable is 1 in the first case with a given set of values for the BY variables, and 0 in other cases. Similarly, the LAST variable is 1 in the last case with a given of BY values, and 0 in other cases.

MATCH FILES may not be specified following TEMPORARY (see Section 11.6 [TEM-PORARY], page 172) if the active file is used as an input source.

Use of portable or scratch files on MATCH FILES is a PSPP extension.

8.7 SAVE

```
SAVE
      /OUTFILE={'file-name',file_handle}
      /UNSELECTED={RETAIN,DELETE}
      /{COMPRESSED,UNCOMPRESSED}
      /PERMISSIONS={WRITEABLE,READONLY}
      /DROP=var_list
      /KEEP=var_list
      /VERSION=version
      /RENAME=(src_names=target_names)...
      /NAMES
```

/MAP

The SAVE procedure causes the dictionary and data in the active file to be written to a system file or scratch file.

OUTFILE is the only required subcommand. Specify the system file or scratch file to be written as a string file name or a file handle (see Section 5.9 [File Handles], page 108).

By default, cases excluded with FILTER are written to the system file. These can be excluded by specifying DELETE on the UNSELECTED subcommand. Specifying RETAIN makes the default explicit.

The COMPRESS and UNCOMPRESS subcommand determine whether the saved system file is compressed. By default, system files are compressed. This default can be changed with the SET command (see Section 14.17 [SET], page 193).

The PERMISSIONS subcommand specifies permissions for the new system file. WRITE-ABLE, the default, creates the file with read and write permission. READONLY creates the file for read-only access.

By default, all the variables in the active file dictionary are written to the system file. The DROP subcommand can be used to specify a list of variables not to be written. In contrast, KEEP specifies variables to be written, with all variables not specified not written.

Normally variables are saved to a system file under the same names they have in the active file. Use the RENAME subcommand to change these names. Specify, within parentheses, a list of variable names followed by an equals sign ('=') and the names that they should be renamed to. Multiple parenthesized groups of variable names can be included on a single RENAME subcommand. Variables' names may be swapped using a RENAME subcommand of the form '/RENAME=(A B=B A)'.

Alternate syntax for the RENAME subcommand allows the parentheses to be eliminated. When this is done, only a single variable may be renamed at once. For instance, '/RENAME=A=B'. This alternate syntax is deprecated.

DROP, KEEP, and RENAME are performed in left-to-right order. They each may be present any number of times. SAVE never modifies the active file. DROP, KEEP, and RENAME only affect the system file written to disk.

The VERSION subcommand specifies the version of the file format. Valid versions are 2 and 3. The default version is 3. In version 2 system files, variable names longer than 8 bytes will be truncated. The two versions are otherwise identical.

The NAMES and MAP subcommands are currently ignored.

SAVE causes the data to be read. It is a procedure.

8.8 SYSFILE INFO

SYSFILE INFO FILE='file-name'.

SYSFILE INFO reads the dictionary in a system file and displays the information in its dictionary.

Specify a file name or file handle. SYSFILE INFO reads that file as a system file and displays information on its dictionary.

SYSFILE INFO does not affect the current active file.

8.9 XEXPORT

        XEXPORT
                /OUTFILE='file-name'
                /DIGITS=n
                /DROP=var_list
                /KEEP=var_list
                /RENAME=(src_names=target_names). . .
                /TYPE={COMM,TAPE}
                /MAP

The EXPORT transformation writes the active file dictionary and data to a specified portable file.

This transformation is a PSPP extension.

It is similar to the EXPORT procedure, with two differences:

- XEXPORT is a transformation, not a procedure. It is executed when the data is read by a procedure or procedure-like command.

- XEXPORT does not support the UNSELECTED subcommand.

See , for more information.

8.10 XSAVE

        XSAVE
                /OUTFILE='file-name'
                /{COMPRESSED,UNCOMPRESSED}
                /PERMISSIONS={WRITEABLE,READONLY}
                /DROP=var_list
                /KEEP=var_list
                /VERSION=version
                /RENAME=(src_names=target_names). . .
                /NAMES
                /MAP

The XSAVE transformation writes the active file dictionary and data to a system file or scratch file. It is similar to the SAVE procedure, with two differences:

- XSAVE is a transformation, not a procedure. It is executed when the data is read by a procedure or procedure-like command.

- XSAVE does not support the UNSELECTED subcommand.

See , for more information.

9 Manipulating variables

The variables in the active file dictionary are important. There are several utility functions for examining and adjusting them.

9.1 ADD VALUE LABELS

> ADD VALUE LABELS
>   /var_list value 'label' [value 'label']. . .

ADD VALUE LABELS has the same syntax and purpose as VALUE LABELS (see Section 9.12 [VALUE LABELS], page 159), but it does not clear value labels from the variables before adding the ones specified.

9.2 DELETE VARIABLES

> DELETE VARIABLES var_list.

DELETE VARIABLES deletes the specified variables from the dictionary. It may not be used to delete all variables from the dictionary; use NEW FILE to do that (see Section 7.9 [NEW FILE], page 138).

DELETE VARIABLES should not used after defining transformations and before executing a procedure. If it is used in such a context, it causes the data to be read. If it is used while TEMPORARY is in effect, it causes the temporary transformations to become permanent.

9.3 DISPLAY

> DISPLAY {NAMES,INDEX,LABELS,VARIABLES,DICTIONARY,SCRATCH}
>   [SORTED] [var_list]

DISPLAY displays requested information on variables. Variables can optionally be sorted alphabetically. The entire dictionary or just specified variables can be described.

One of the following keywords can be present:

NAMES   The variables' names are displayed.

INDEX   The variables' names are displayed along with a value describing their position within the active file dictionary.

LABELS   Variable names, positions, and variable labels are displayed.

VARIABLES
    Variable names, positions, print and write formats, and missing values are displayed.

DICTIONARY
    Variable names, positions, print and write formats, missing values, variable labels, and value labels are displayed.

SCRATCH
    Varible names are displayed, for scratch variables only (see Section 5.7.5 [Scratch Variables], page 107).

If SORTED is specified, then the variables are displayed in ascending order based on their names; otherwise, they are displayed in the order that they occur in the active file dictionary.

9.4 DISPLAY VECTORS

DISPLAY VECTORS.

DISPLAY VECTORS lists all the currently declared vectors.

9.5 FORMATS

FORMATS var_list (fmt_spec).

FORMATS set both print and write formats for the specified numeric variables to the specified format specification. See Section 5.7.4 [Input and Output Formats], page 98.

Specify a list of variables followed by a format specification in parentheses. The print and write formats of the specified variables will be changed.

Additional lists of variables and formats may be included if they are delimited by a slash ('/').

FORMATS takes effect immediately. It is not affected by conditional and looping structures such as DO IF or LOOP.

9.6 LEAVE

LEAVE var_list.

LEAVE prevents the specified variables from being reinitialized whenever a new case is processed.

Normally, when a data file is processed, every variable in the active file is initialized to the system-missing value or spaces at the beginning of processing for each case. When a variable has been specified on LEAVE, this is not the case. Instead, that variable is initialized to 0 (not system-missing) or spaces for the first case. After that, it retains its value between cases.

This becomes useful for counters. For instance, in the example below the variable SUM maintains a running total of the values in the ITEM variable.

```
DATA LIST /ITEM 1-3.
COMPUTE SUM=SUM+ITEM.
PRINT /ITEM SUM.
LEAVE SUM
BEGIN DATA.
123
404
555
999
END DATA.
```

Partial output from this example:

```
123    123.00
404    527.00
```

```
555  1082.00
999  2081.00
```

It is best to use LEAVE command immediately before invoking a procedure command, because the left status of variables is reset by certain transformations—for instance, COMPUTE and IF. Left status is also reset by all procedure invocations.

9.7 MISSING VALUES

MISSING VALUES var_list (missing_values).

missing_values takes one of the following forms:

num1
num1, num2
num1, num2, num3
num1 THRU num2
num1 THRU num2, num3
string1
string1, string2
string1, string2, string3

As part of a range, LO or LOWEST may take the place of num1;
HI or HIGHEST may take the place of num2.

MISSING VALUES sets user-missing values for numeric and short string variables. Long string variables may not have missing values.

Specify a list of variables, followed by a list of their user-missing values in parentheses. Up to three discrete values may be given, or, for numeric variables only, a range of values optionally accompanied by a single discrete value. Ranges may be open-ended on one end, indicated through the use of the keyword LO or LOWEST or HI or HIGHEST.

The MISSING VALUES command takes effect immediately. It is not affected by conditional and looping constructs such as DO IF or LOOP.

9.8 MODIFY VARS

MODIFY VARS
/REORDER={FORWARD,BACKWARD} {POSITIONAL,ALPHA} (var_list)...
/RENAME=(old_names=new_names)...
/{DROP,KEEP}=var_list
/MAP

MODIFY VARS reorders, renames, and deletes variables in the active file.

At least one subcommand must be specified, and no subcommand may be specified more than once. DROP and KEEP may not both be specified.

The REORDER subcommand changes the order of variables in the active file. Specify one or more lists of variable names in parentheses. By default, each list of variables is rearranged into the specified order. To put the variables into the reverse of the specified order, put keyword BACKWARD before the parentheses. To put them into alphabetical order in the dictionary, specify keyword ALPHA before the parentheses. BACKWARD and ALPHA may also be combined.

To rename variables in the active file, specify RENAME, an equals sign ('='), and lists of the old variable names and new variable names separated by another equals sign within parentheses. There must be the same number of old and new variable names. Each old variable is renamed to the corresponding new variable name. Multiple parenthesized groups of variables may be specified.

The DROP subcommand deletes a specified list of variables from the active file.

The KEEP subcommand keeps the specified list of variables in the active file. Any unlisted variables are deleted from the active file.

MAP is currently ignored.

If either DROP or KEEP is specified, the data is read; otherwise it is not.

MODIFY VARS may not be specified following TEMPORARY (see Section 11.6 [TEMPORARY], page 172).

9.9 NUMERIC

> NUMERIC /var_list [(fmt_spec)].

NUMERIC explicitly declares new numeric variables, optionally setting their output formats.

Specify a slash ('/'), followed by the names of the new numeric variables. If you wish to set their output formats, follow their names by an output format specification in parentheses (see Section 5.7.4 [Input and Output Formats], page 98); otherwise, the default is F8.2.

Variables created with NUMERIC are initialized to the system-missing value.

9.10 PRINT FORMATS

> PRINT FORMATS var_list (fmt_spec).

PRINT FORMATS sets the print formats for the specified numeric variables to the specified format specification.

Its syntax is identical to that of FORMATS (see Section 9.5 [FORMATS], page 156), but PRINT FORMATS sets only print formats, not write formats.

9.11 RENAME VARIABLES

> RENAME VARIABLES (old_names=new_names). . . .

RENAME VARIABLES changes the names of variables in the active file. Specify lists of the old variable names and new variable names, separated by an equals sign ('='), within parentheses. There must be the same number of old and new variable names. Each old variable is renamed to the corresponding new variable name. Multiple parenthesized groups of variables may be specified.

RENAME VARIABLES takes effect immediately. It does not cause the data to be read.

RENAME VARIABLES may not be specified following TEMPORARY (see Section 11.6 [TEMPORARY], page 172).

9.12 VALUE LABELS

> VALUE LABELS
>> /var_list value 'label' [value 'label']. . .

VALUE LABELS allows values of numeric and short string variables to be associated with labels. In this way, a short value can stand for a long value.

To set up value labels for a set of variables, specify the variable names after a slash ('/'), followed by a list of values and their associated labels, separated by spaces. Long string variables may not be specified.

Before VALUE LABELS is executed, any existing value labels are cleared from the variables specified. Use ADD VALUE LABELS (see Section 9.1 [ADD VALUE LABELS], page 155) to add value labels without clearing those already present.

9.13 STRING

> STRING /var_list (fmt_spec).

STRING creates new string variables for use in transformations.

Specify a slash ('/'), followed by the names of the string variables to create and the desired output format specification in parentheses (see Section 5.7.4 [Input and Output Formats], page 98). Variable widths are implicitly derived from the specified output formats.

Created variables are initialized to spaces.

9.14 VARIABLE LABELS

> VARIABLE LABELS
>> var_list 'var_label'
>> [/var_list 'var_label']
>> .
>> .
>> .
>> [/var_list 'var_label']

VARIABLE LABELS associates explanatory names with variables. This name, called a *variable label*, is displayed by statistical procedures.

To assign a variable label to a group of variables, specify a list of variable names and the variable label as a string. To assign different labels to different variables in the same command, precede the subsequent variable list with a slash ('/').

9.15 VARIABLE ALIGNMENT

> VARIABLE ALIGNMENT
>> var_list (LEFT | RIGHT | CENTER)
>> [/var_list (LEFT | RIGHT | CENTER)]
>> .
>> .
>> .
>> [/var_list (LEFT | RIGHT | CENTER)]

VARIABLE ALIGNMENT sets the alignment of variables for display editing purposes. This only has effect for third party software. It does not affect the display of variables in the PSPP output.

9.16 VARIABLE WIDTH

        VARIABLE WIDTH
                var_list (width)
                [/var_list (width)]
                .
                .
                .
                [/var_list (width)]

VARIABLE WIDTH sets the column width of variables for display editing purposes. This only affects third party software. It does not affect the display of variables in the PSPP output.

9.17 VARIABLE LEVEL

        VARIABLE LEVEL
                var_list (SCALE | NOMINAL | ORDINAL)
                [/var_list (SCALE | NOMINAL | ORDINAL)]
                .
                .
                .
                [/var_list (SCALE | NOMINAL | ORDINAL)]

VARIABLE LEVEL sets the measurement level of variables. Currently, this has no effect except for certain third party software.

9.18 VECTOR

        Two possible syntaxes:
                VECTOR vec_name=var_list.
                VECTOR vec_name_list(count [format]).

VECTOR allows a group of variables to be accessed as if they were consecutive members of an array with a vector(index) notation.

To make a vector out of a set of existing variables, specify a name for the vector followed by an equals sign ('=') and the variables to put in the vector. All the variables in the vector must be the same type. String variables in a vector must all have the same width.

To make a vector and create variables at the same time, specify one or more vector names followed by a count in parentheses. This will cause variables named `vec1` through `veccount` to be created as numeric variables. By default, the new variables have print and write format F8.2, but an alternate format may be specified inside the parentheses before or after the count and separated from it by white space or a comma. Variable names including numeric suffixes may not exceed 64 characters in length, and none of the variables may exist prior to VECTOR.

Vectors created with VECTOR disappear after any procedure or procedure-like command is executed. The variables contained in the vectors remain, unless they are scratch variables (see Section 5.7.5 [Scratch Variables], page 107).

Variables within a vector may be referenced in expressions using `vector(index)` syntax.

9.19 WRITE FORMATS

WRITE FORMATS var_list (fmt_spec).

WRITE FORMATS sets the write formats for the specified numeric variables to the specified format specification. Its syntax is identical to that of FORMATS (see Section 9.5 [FORMATS], page 156), but WRITE FORMATS sets only write formats, not print formats.

10 Data transformations

The PSPP procedures examined in this chapter manipulate data and prepare the active file for later analyses. They do not produce output, as a rule.

10.1 AGGREGATE

        AGGREGATE
            OUTFILE={*,'file-name',file_handle}
            /PRESORTED
            /DOCUMENT
            /MISSING=COLUMNWISE
            /BREAK=var_list
            /dest_var['label']. . .=agr_func(src_vars, args. . .). . .

AGGREGATE summarizes groups of cases into single cases. Cases are divided into groups that have the same values for one or more variables called *break variables*. Several functions are available for summarizing case contents.

The OUTFILE subcommand is required and must appear first. Specify a system file, portable file, or scratch file by file name or file handle (see Section 5.9 [File Handles], page 108). The aggregated cases are written to this file. If '*' is specified, then the aggregated cases replace the active file. Use of OUTFILE to write a portable file or scratch file is a PSPP extension.

By default, the active file will be sorted based on the break variables before aggregation takes place. If the active file is already sorted or otherwise grouped in terms of the break variables, specify PRESORTED to save time.

Specify DOCUMENT to copy the documents from the active file into the aggregate file (see Section 14.4 [DOCUMENT], page 190). Otherwise, the aggregate file will not contain any documents, even if the aggregate file replaces the active file.

Normally, only a single case (for SD and SD., two cases) need be non-missing in each group for the aggregate variable to be non-missing. Specifying /MISSING=COLUMNWISE inverts this behavior, so that the aggregate variable becomes missing if any aggregated value is missing.

If PRESORTED, DOCUMENT, or MISSING are specified, they must appear between OUTFILE and BREAK.

At least one break variable must be specified on BREAK, a required subcommand. The values of these variables are used to divide the active file into groups to be summarized. In addition, at least one *dest_var* must be specified.

One or more sets of aggregation variables must be specified. Each set comprises a list of aggregation variables, an equals sign ('='), the name of an aggregation function (see the list below), and a list of source variables in parentheses. Some aggregation functions expect additional arguments following the source variable names.

Aggregation variables typically are created with no variable label, value labels, or missing values. Their default print and write formats depend on the aggregation function used, with details given in the table below. A variable label for an aggregation variable may be specified just after the variable's name in the aggregation variable list.

Each set must have exactly as many source variables as aggregation variables. Each aggregation variable receives the results of applying the specified aggregation function to the corresponding source variable. The MEAN, SD, and SUM aggregation functions may only be applied to numeric variables. All the rest may be applied to numeric and short and long string variables.

The available aggregation functions are as follows:

FGT(var_name, value)

> Fraction of values greater than the specified constant. The default format is F5.3.

FIN(var_name, low, high)

> Fraction of values within the specified inclusive range of constants. The default format is F5.3.

FLT(var_name, value)

> Fraction of values less than the specified constant. The default format is F5.3.

FIRST(var_name)

> First non-missing value in break group. The aggregation variable receives the complete dictionary information from the source variable. The sort performed by AGGREGATE (and by SORT CASES) is stable, so that the first case with particular values for the break variables before sorting will also be the first case in that break group after sorting.

FOUT(var_name, low, high)

> Fraction of values strictly outside the specified range of constants. The default format is F5.3.

LAST(var_name)

> Last non-missing value in break group. The aggregation variable receives the complete dictionary information from the source variable. The sort performed by AGGREGATE (and by SORT CASES) is stable, so that the last case with particular values for the break variables before sorting will also be the last case in that break group after sorting.

MAX(var_name)

> Maximum value. The aggregation variable receives the complete dictionary information from the source variable.

MEAN(var_name)

> Arithmetic mean. Limited to numeric values. The default format is F8.2.

MIN(var_name)

> Minimum value. The aggregation variable receives the complete dictionary information from the source variable.

N(var_name)

> Number of non-missing values. The default format is F7.0 if weighting is not enabled, F8.2 if it is (see Section 11.7 [WEIGHT], page 173).

N

> Number of cases aggregated to form this group. The default format is F7.0 if weighting is not enabled, F8.2 if it is (see Section 11.7 [WEIGHT], page 173).

NMISS(var_name)

> Number of missing values. The default format is F7.0 if weighting is not enabled, F8.2 if it is (see Section 11.7 [WEIGHT], page 173).

NU(var_name)

> Number of non-missing values. Each case is considered to have a weight of 1, regardless of the current weighting variable (see Section 11.7 [WEIGHT], page 173). The default format is F7.0.

NU

> Number of cases aggregated to form this group. Each case is considered to have a weight of 1, regardless of the current weighting variable. The default format is F7.0.

NUMISS(var_name)

> Number of missing values. Each case is considered to have a weight of 1, regardless of the current weighting variable. The default format is F7.0.

PGT(var_name, value)

> Percentage between 0 and 100 of values greater than the specified constant. The default format is F5.1.

PIN(var_name, low, high)

> Percentage of values within the specified inclusive range of constants. The default format is F5.1.

PLT(var_name, value)

> Percentage of values less than the specified constant. The default format is F5.1.

POUT(var_name, low, high)

> Percentage of values strictly outside the specified range of constants. The default format is F5.1.

SD(var_name)

> Standard deviation of the mean. Limited to numeric values. The default format is F8.2.

SUM(var_name)

> Sum. Limited to numeric values. The default format is F8.2.

Aggregation functions compare string values in terms of internal character codes. On most modern computers, this is a form of ASCII.

The aggregation functions listed above exclude all user-missing values from calculations. To include user-missing values, insert a period ('.') at the end of the function name. (e.g. 'SUM.'). (Be aware that specifying such a function as the last token on a line will cause the period to be interpreted as the end of the command.)

AGGREGATE both ignores and cancels the current SPLIT FILE settings (see Section 11.5 [SPLIT FILE], page 171).

10.2 AUTORECODE

> AUTORECODE VARIABLES=src_vars INTO dest_vars
> /DESCENDING
> /PRINT

The AUTORECODE procedure considers the *n* values that a variable takes on and maps them onto values 1...*n* on a new numeric variable.

Subcommand VARIABLES is the only required subcommand and must come first. Specify VARIABLES, an equals sign ('='), a list of source variables, INTO, and a list of target variables. There must the same number of source and target variables. The target variables must not already exist.

By default, increasing values of a source variable (for a string, this is based on character code comparisons) are recoded to increasing values of its target variable. To cause increasing values of a source variable to be recoded to decreasing values of its target variable (*n* down to 1), specify DESCENDING.

PRINT is currently ignored.

AUTORECODE is a procedure. It causes the data to be read.

10.3 COMPUTE

> COMPUTE variable = expression.
> or
> COMPUTE vector(index) = expression.

COMPUTE assigns the value of an expression to a target variable. For each case, the expression is evaluated and its value assigned to the target variable. Numeric and short and long string variables may be assigned. When a string expression's width differs from the target variable's width, the string result of the expression is truncated or padded with spaces on the right as necessary. The expression and variable types must match.

For numeric variables only, the target variable need not already exist. Numeric variables created by COMPUTE are assigned an `F8.2` output format. String variables must be declared before they can be used as targets for COMPUTE.

The target variable may be specified as an element of a vector (see Section 9.18 [VECTOR], page 160). In this case, a vector index expression must be specified in parentheses following the vector name. The index expression must evaluate to a numeric value that, after rounding down to the nearest integer, is a valid index for the named vector.

Using COMPUTE to assign to a variable specified on LEAVE (see Section 9.6 [LEAVE], page 156) resets the variable's left state. Therefore, `LEAVE` should be specified following COMPUTE, not before.

COMPUTE is a transformation. It does not cause the active file to be read.

When COMPUTE is specified following TEMPORARY (see Section 11.6 [TEMPORARY], page 172), the LAG function may not be used (see [LAG], page 121).

10.4 COUNT

> COUNT var_name = var... (value...).

Each value takes one of the following forms:

> number
> string
> num1 THRU num2
> MISSING
> SYSMIS

In addition, num1 and num2 can be LO or LOWEST, or HI or HIGHEST, respectively.

COUNT creates or replaces a numeric *target* variable that counts the occurrence of a *criterion* value or set of values over one or more *test* variables for each case.

The target variable values are always nonnegative integers. They are never missing. The target variable is assigned an F8.2 output format. See Section 5.7.4 [Input and Output Formats], page 98. Any variables, including long and short string variables, may be test variables.

User-missing values of test variables are treated just like any other values. They are **not** treated as system-missing values. User-missing values that are criterion values or inside ranges of criterion values are counted as any other values. However (for numeric variables), keyword MISSING may be used to refer to all system- and user-missing values.

COUNT target variables are assigned values in the order specified. In the command `COUNT A=A B(1) /B=A B(2).`, the following actions occur:

− The number of occurrences of 1 between `A` and `B` is counted.

− `A` is assigned this value.

− The number of occurrences of 1 between `B` and the **new** value of `A` is counted.

− `B` is assigned this value.

Despite this ordering, all COUNT criterion variables must exist before the procedure is executed—they may not be created as target variables earlier in the command! Break such a command into two separate commands.

The examples below may help to clarify.

A. Assuming `Q0`, `Q2`, . . . , `Q9` are numeric variables, the following commands:

   1. Count the number of times the value 1 occurs through these variables for each case and assigns the count to variable `QCOUNT`.

   2. Print out the total number of times the value 1 occurs throughout *all* cases using DESCRIPTIVES. See Section 13.1 [DESCRIPTIVES], page 177, for details.

```
COUNT QCOUNT=Q0 TO Q9(1).
DESCRIPTIVES QCOUNT /STATISTICS=SUM.
```

B. Given these same variables, the following commands:

   1. Count the number of valid values of these variables for each case and assigns the count to variable `QVALID`.

   2. Multiplies each value of `QVALID` by 10 to obtain a percentage of valid values, using COMPUTE. See Section 10.3 [COMPUTE], page 165, for details.

   3. Print out the percentage of valid values across all cases, using DESCRIPTIVES. See Section 13.1 [DESCRIPTIVES], page 177, for details.

```
COUNT QVALID=Q0 TO Q9 (LO THRU HI).
COMPUTE QVALID=QVALID*10.
DESCRIPTIVES QVALID /STATISTICS=MEAN.
```

10.5 FLIP

FLIP /VARIABLES=var_list /NEWNAMES=var_name.

FLIP transposes rows and columns in the active file. It causes cases to be swapped with variables, and vice versa.

All variables in the transposed active file are numeric. String variables take on the system-missing value in the transposed file.

No subcommands are required. If specified, the VARIABLES subcommand selects variables to be transformed into cases, and variables not specified are discarded. If the VARIABLES subcommand is omitted, all variables are selected for transposition.

The variables specified by NEWNAMES, which must be a string variable, is used to give names to the variables created by FLIP. Only the first 8 characters of the variable are used. If NEWNAMES is not specified then the default is a variable named CASE_LBL, if it exists. If it does not then the variables created by FLIP are named VAR000 through VAR999, then VAR1000, VAR1001, and so on.

When a NEWNAMES variable is available, the names must be canonicalized before becoming variable names. Invalid characters are replaced by letter 'V' in the first position, or by '_' in subsequent positions. If the name thus generated is not unique, then numeric extensions are added, starting with 1, until a unique name is found or there are no remaining possibilities. If the latter occurs then the FLIP operation aborts.

The resultant dictionary contains a CASE_LBL variable, a string variable of width 8, which stores the names of the variables in the dictionary before the transposition. Variables names longer than 8 characters are truncated. If the active file is subsequently transposed using FLIP, this variable can be used to recreate the original variable names.

FLIP honors N OF CASES (see Section 11.2 [N OF CASES], page 170). It ignores TEMPORARY (see Section 11.6 [TEMPORARY], page 172), so that "temporary" transformations become permanent.

10.6 IF

IF condition variable=expression.
  or
IF condition vector(index)=expression.

The IF transformation conditionally assigns the value of a target expression to a target variable, based on the truth of a test expression.

Specify a boolean-valued expression (see Chapter 6 [Expressions], page 110) to be tested following the IF keyword. This expression is evaluated for each case. If the value is true, then the value of the expression is computed and assigned to the specified variable. If the value is false or missing, nothing is done. Numeric and short and long string variables may be assigned. When a string expression's width differs from the target variable's width, the string result of the expression is truncated or padded with spaces on the right as necessary. The expression and variable types must match.

The target variable may be specified as an element of a vector (see Section 9.18 [VEC-TOR], page 160). In this case, a vector index expression must be specified in parentheses following the vector name. The index expression must evaluate to a numeric value that, after rounding down to the nearest integer, is a valid index for the named vector.

Using IF to assign to a variable specified on LEAVE (see Section 9.6 [LEAVE], page 156) resets the variable's left state. Therefore, `LEAVE` should be specified following IF, not before.

When IF is specified following TEMPORARY (see Section 11.6 [TEMPORARY], page 172), the LAG function may not be used (see [LAG], page 121).

10.7 RECODE

RECODE var_list (src_value. . .=dest_value). . . [INTO var_list].

src_value may take the following forms:
> number
> string
> num1 THRU num2
> MISSING
> SYSMIS
> ELSE

Open-ended ranges may be specified using LO or LOWEST for num1
or HI or HIGHEST for num2.

dest_value may take the following forms:
> num
> string
> SYSMIS
> COPY

RECODE translates data from one range of values to another, via flexible user-specified mappings. Data may be remapped in-place or copied to new variables. Numeric, short string, and long string data can be recoded.

Specify the list of source variables, followed by one or more mapping specifications each enclosed in parentheses. If the data is to be copied to new variables, specify INTO, then the list of target variables. String target variables must already have been declared using STRING or another transformation, but numeric target variables can be created on the fly. There must be exactly as many target variables as source variables. Each source variable is remapped into its corresponding target variable.

When INTO is not used, the input and output variables must be of the same type. Otherwise, string values can be recoded into numeric values, and vice versa. When this is done and there is no mapping for a particular value, either a value consisting of all spaces or the system-missing value is assigned, depending on variable type.

Mappings are considered from left to right. The first src_value that matches the value of the source variable causes the target variable to receive the value indicated by the dest_value. Literal number, string, and range src_value's should be self-explanatory. MISSING as a src_value matches any user- or system-missing value. SYSMIS matches the system missing

value only. ELSE is a catch-all that matches anything. It should be the last src_value specified.

Numeric and string dest_value's should be self-explanatory. COPY causes the input values to be copied to the output. This is only valid if the source and target variables are of the same type. SYSMIS indicates the system-missing value.

If the source variables are strings and the target variables are numeric, then there is one additional mapping available: (CONVERT), which must be the last specified mapping. CONVERT causes a number specified as a string to be converted to a numeric value. If the string cannot be parsed as a number, then the system-missing value is assigned.

Multiple recodings can be specified on a single RECODE invocation. Introduce additional recodings with a slash ('/') to separate them from the previous recodings.

10.8 SORT CASES

SORT CASES BY var_list[({D|A}] [var_list[({D|A}]] ...

SORT CASES sorts the active file by the values of one or more variables.

Specify BY and a list of variables to sort by. By default, variables are sorted in ascending order. To override sort order, specify (D) or (DOWN) after a list of variables to get descending order, or (A) or (UP) for ascending order. These apply to all the listed variables up until the preceding (A), (D), (UP) or (DOWN).

The sort algorithms used by SORT CASES are stable. That is, records that have equal values of the sort variables will have the same relative order before and after sorting. As a special case, re-sorting an already sorted file will not affect the ordering of cases.

SORT CASES is a procedure. It causes the data to be read.

SORT CASES attempts to sort the entire active file in main memory. If workspace is exhausted, it falls back to a merge sort algorithm that involves creates numerous temporary files.

SORT CASES may not be specified following TEMPORARY.

11 Selecting data for analysis

This chapter documents PSPP commands that temporarily or permanently select data records from the active file for analysis.

11.1 FILTER

> FILTER BY var_name.
> FILTER OFF.

FILTER allows a boolean-valued variable to be used to select cases from the data stream for processing.

To set up filtering, specify BY and a variable name. Keyword BY is optional but recommended. Cases which have a zero or system- or user-missing value are excluded from analysis, but not deleted from the data stream. Cases with other values are analyzed. To filter based on a different condition, use transformations such as COMPUTE or RECODE to compute a filter variable of the required form, then specify that variable on FILTER.

`FILTER OFF` turns off case filtering.

Filtering takes place immediately before cases pass to a procedure for analysis. Only one filter variable may be active at a time. Normally, case filtering continues until it is explicitly turned off with `FILTER OFF`. However, if FILTER is placed after TEMPORARY, it filters only the next procedure or procedure-like command.

11.2 N OF CASES

> N [OF CASES] num_of_cases [ESTIMATED].

N OF CASES limits the number of cases processed by any procedures that follow it in the command stream. `N OF CASES 100`, for example, tells PSPP to disregard all cases after the first 100.

When N OF CASES is specified after TEMPORARY, it affects only the next procedure (see Section 11.6 [TEMPORARY], page 172). Otherwise, cases beyond the limit specified are not processed by any later procedure.

If the limit specified on N OF CASES is greater than the number of cases in the active file, it has no effect.

When N OF CASES is used along with SAMPLE or SELECT IF, the case limit is applied to the cases obtained after sampling or case selection, regardless of how N OF CASES is placed relative to SAMPLE or SELECT IF in the command file. Thus, the commands `N OF CASES 100` and `SAMPLE .5` will both randomly sample approximately half of the active file's cases, then select the first 100 of those sampled, regardless of their order in the command file.

N OF CASES with the `ESTIMATED` keyword gives an estimated number of cases before DATA LIST or another command to read in data. `ESTIMATED` never limits the number of cases processed by procedures. PSPP currently does not make use of case count estimates.

11.3 SAMPLE

SAMPLE num1 [FROM num2].

SAMPLE randomly samples a proportion of the cases in the active file. Unless it follows TEMPORARY, it operates as a transformation, permanently removing cases from the active file.

The proportion to sample can be expressed as a single number between 0 and 1. If k is the number specified, and N is the number of currently-selected cases in the active file, then after `SAMPLE` k`.`, approximately `k*N` cases will be selected.

The proportion to sample can also be specified in the style `SAMPLE` m `FROM` N. With this style, cases are selected as follows:

1. If N is equal to the number of currently-selected cases in the active file, exactly m cases will be selected.

2. If N is greater than the number of currently-selected cases in the active file, an equivalent proportion of cases will be selected.

3. If N is less than the number of currently-selected cases in the active, exactly m cases will be selected *from the first N cases in the active file.*

SAMPLE and SELECT IF are performed in the order specified by the syntax file.

SAMPLE is always performed before `N OF CASES`, regardless of ordering in the syntax file (see Section 11.2 [N OF CASES], page 170).

The same values for SAMPLE may result in different samples. To obtain the same sample, use the `SET` command to set the random number seed to the same value before each SAMPLE. Different samples may still result when the file is processed on systems with differing endianness or floating-point formats. By default, the random number seed is based on the system time.

11.4 SELECT IF

SELECT IF expression.

SELECT IF selects cases for analysis based on the value of a boolean expression. Cases not selected are permanently eliminated from the active file, unless TEMPORARY is in effect (see Section 11.6 [TEMPORARY], page 172).

Specify a boolean expression (see Chapter 6 [Expressions], page 110). If the value of the expression is true for a particular case, the case will be analyzed. If the expression has a false or missing value, then the case will be deleted from the data stream.

Place SELECT IF as early in the command file as possible. Cases that are deleted early can be processed more efficiently in time and space.

When SELECT IF is specified following TEMPORARY (see Section 11.6 [TEMPO-RARY], page 172), the LAG function may not be used (see [LAG], page 121).

11.5 SPLIT FILE

SPLIT FILE [{LAYERED, SEPARATE}] BY var_list.
SPLIT FILE OFF.

SPLIT FILE allows multiple sets of data present in one data file to be analyzed separately using single statistical procedure commands.

Specify a list of variable names to analyze multiple sets of data separately. Groups of adjacent cases having the same values for these variables are analyzed by statistical procedure commands as one group. An independent analysis is carried out for each group of cases, and the variable values for the group are printed along with the analysis.

When a list of variable names is specified, one of the keywords LAYERED or SEPARATE may also be specified. If provided, either keyword are ignored.

Groups are formed only by *adjacent* cases. To create a split using a variable where like values are not adjacent in the working file, you should first sort the data by that variable (see Section 10.8 [SORT CASES], page 169).

Specify OFF to disable SPLIT FILE and resume analysis of the entire active file as a single group of data.

When SPLIT FILE is specified after TEMPORARY, it affects only the next procedure (see Section 11.6 [TEMPORARY], page 172).

11.6 TEMPORARY

TEMPORARY.

TEMPORARY is used to make the effects of transformations following its execution temporary. These transformations will affect only the execution of the next procedure or procedure-like command. Their effects will not be saved to the active file.

The only specification on TEMPORARY is the command name.

TEMPORARY may not appear within a DO IF or LOOP construct. It may appear only once between procedures and procedure-like commands.

Scratch variables cannot be used following TEMPORARY.

An example may help to clarify:

```
DATA LIST /X 1-2.
BEGIN DATA.
 2
 4
10
15
20
24
END DATA.
COMPUTE X=X/2.
TEMPORARY.
COMPUTE X=X+3.
DESCRIPTIVES X.
DESCRIPTIVES X.
```

The data read by the first DESCRIPTIVES are 4, 5, 8, 10.5, 13, 15. The data read by the first DESCRIPTIVES are 1, 2, 5, 7.5, 10, 12.

11.7 WEIGHT

WEIGHT BY var_name.
WEIGHT OFF.

WEIGHT assigns cases varying weights, changing the frequency distribution of the active file. Execution of WEIGHT is delayed until data have been read.

If a variable name is specified, WEIGHT causes the values of that variable to be used as weighting factors for subsequent statistical procedures. Use of keyword BY is optional but recommended. Weighting variables must be numeric. Scratch variables may not be used for weighting (see Section 5.7.5 [Scratch Variables], page 107).

When OFF is specified, subsequent statistical procedures will weight all cases equally.

A positive integer weighting factor w on a case will yield the same statistical output as would replicating the case w times. A weighting factor of 0 is treated for statistical purposes as if the case did not exist in the input. Weighting values need not be integers, but negative and system-missing values for the weighting variable are interpreted as weighting factors of 0. User-missing values are not treated specially.

When WEIGHT is specified after TEMPORARY, it affects only the next procedure (see Section 11.6 [TEMPORARY], page 172).

WEIGHT does not cause cases in the active file to be replicated in memory.

12 Conditional and Looping Constructs

This chapter documents PSPP commands used for conditional execution, looping, and flow of control.

12.1 BREAK

> BREAK.

BREAK terminates execution of the innermost currently executing LOOP construct.

BREAK is allowed only inside LOOP. . .END LOOP. See Section 12.4 [LOOP], page 175, for more details.

12.2 DO IF

> DO IF condition.
>   . . .
> [ELSE IF condition.
>   . . .
>]. . .
> [ELSE.
>   . . .]
> END IF.

DO IF allows one of several sets of transformations to be executed, depending on user-specified conditions.

If the specified boolean expression evaluates as true, then the block of code following DO IF is executed. If it evaluates as missing, then none of the code blocks is executed. If it is false, then the boolean expression on the first ELSE IF, if present, is tested in turn, with the same rules applied. If all expressions evaluate to false, then the ELSE code block is executed, if it is present.

When DO IF or ELSE IF is specified following TEMPORARY (see Section 11.6 [TEM-PORARY], page 172), the LAG function may not be used (see [LAG], page 121).

12.3 DO REPEAT

> DO REPEAT dummy_name=expansion. . . .
>   . . .
> END REPEAT [PRINT].

> expansion takes one of the following forms:
>   var_list
>   num_or_range. . .
>   'string'. . .

> num_or_range takes one of the following forms:
>   number
>   num1 TO num2

DO REPEAT repeats a block of code, textually substituting different variables, numbers, or strings into the block with each repetition.

Specify a dummy variable name followed by an equals sign ('=') and the list of replacements. Replacements can be a list of variables (which may be existing variables or new variables or some combination), numbers, or strings. When new variable names are specified, DO REPEAT creates them as numeric variables. When numbers are specified, runs of increasing integers may be indicated as `num1 TO num2`, so that '1 TO 5' is short for '1 2 3 4 5'.

Multiple dummy variables can be specified. Each variable must have the same number of replacements.

The code within DO REPEAT is repeated as many times as there are replacements for each variable. The first time, the first value for each dummy variable is substituted; the second time, the second value for each dummy variable is substituted; and so on.

Dummy variable substitutions work like macros. They take place anywhere in a line that the dummy variable name occurs as a token, including command and subcommand names. For this reason, words commonly used in command and subcommand names should not be used as dummy variable identifiers.

If PRINT is specified on END REPEAT, the commands after substitutions are made are printed to the listing file, prefixed by a plus sign ('+').

12.4 LOOP

> LOOP [index_var=start TO end [BY incr]] [IF condition].
>
>   . . .
>
> END LOOP [IF condition].

LOOP iterates a group of commands. A number of termination options are offered.

Specify index_var to make that variable count from one value to another by a particular increment. index_var must be a pre-existing numeric variable. start, end, and incr are numeric expressions (see Chapter 6 [Expressions], page 110.)

During the first iteration, index_var is set to the value of start. During each successive iteration, index_var is increased by the value of incr. If end > start, then the loop terminates when index_var > end; otherwise it terminates when index_var < end. If incr is not specified then it defaults to +1 or -1 as appropriate.

If end > start and incr < 0, or if end < start and incr > 0, then the loop is never executed. index_var is nevertheless set to the value of start.

Modifying index_var within the loop is allowed, but it has no effect on the value of index_var in the next iteration.

Specify a boolean expression for the condition on LOOP to cause the loop to be executed only if the condition is true. If the condition is false or missing before the loop contents are executed the first time, the loop contents are not executed at all.

If index and condition clauses are both present on LOOP, the index variable is always set before the condition is evaluated. Thus, a condition that makes use of the index variable will always see the index value to be used in the next execution of the body.

Specify a boolean expression for the condition on END LOOP to cause the loop to terminate if the condition is true after the enclosed code block is executed. The condition

is evaluated at the end of the loop, not at the beginning, so that the body of a loop with only a condition on END LOOP will always execute at least once.

If neither the index clause nor either condition clause is present, then the loop is executed MXLOOPS (see Section 14.17 [SET], page 193) times.

BREAK also terminates LOOP execution (see Section 12.1 [BREAK], page 174).

Loop index variables are by default reset to system-missing from one case to another, not left, unless a scratch variable is used as index. When loops are nested, this is usually undesired behavior, which can be corrected with LEAVE (see Section 9.6 [LEAVE], page 156) or by using a scratch variable as the loop index.

When LOOP or END LOOP is specified following TEMPORARY (see Section 11.6 [TEMPORARY], page 172), the LAG function may not be used (see [LAG], page 121).

13 Statistics

This chapter documents the statistical procedures that PSPP supports so far.

13.1 DESCRIPTIVES

```
DESCRIPTIVES
      /VARIABLES=var_list
      /MISSING={VARIABLE,LISTWISE} {INCLUDE,NOINCLUDE}
      /FORMAT={LABELS,NOLABELS} {NOINDEX,INDEX} {LINE,SERIAL}
      /SAVE
      /STATISTICS={ALL,MEAN,SEMEAN,STDDEV,VARIANCE,KURTOSIS,
                SKEWNESS,RANGE,MINIMUM,MAXIMUM,SUM,DEFAULT,
                SESKEWNESS,SEKURTOSIS}
    /SORT={NONE,MEAN,SEMEAN,STDDEV,VARIANCE,KURTOSIS,SKEWNESS,
        RANGE,MINIMUM,MAXIMUM,SUM,SESKEWNESS,SEKURTOSIS,NAME}
          {A,D}
```

The DESCRIPTIVES procedure reads the active file and outputs descriptive statistics requested by the user. In addition, it can optionally compute Z-scores.

The VARIABLES subcommand, which is required, specifies the list of variables to be analyzed. Keyword VARIABLES is optional.

All other subcommands are optional:

The MISSING subcommand determines the handling of missing variables. If INCLUDE is set, then user-missing values are included in the calculations. If NOINCLUDE is set, which is the default, user-missing values are excluded. If VARIABLE is set, then missing values are excluded on a variable by variable basis; if LISTWISE is set, then the entire case is excluded whenever any value in that case has a system-missing or, if INCLUDE is set, user-missing value.

The FORMAT subcommand affects the output format. Currently the LABELS/NOLABELS and NOINDEX/INDEX settings are not used. When SERIAL is set, both valid and missing number of cases are listed in the output; when NOSERIAL is set, only valid cases are listed.

The SAVE subcommand causes DESCRIPTIVES to calculate Z scores for all the specified variables. The Z scores are saved to new variables. Variable names are generated by trying first the original variable name with Z prepended and truncated to a maximum of 8 characters, then the names ZSC000 through ZSC999, STDZ00 through STDZ09, ZZZZ00 through ZZZZ09, ZQZQ00 through ZQZQ09, in that sequence. In addition, Z score variable names can be specified explicitly on VARIABLES in the variable list by enclosing them in parentheses after each variable.

The STATISTICS subcommand specifies the statistics to be displayed:

ALL   All of the statistics below.

MEAN   Arithmetic mean.

SEMEAN   Standard error of the mean.

STDDEV   Standard deviation.

VARIANCE   Variance.

KURTOSIS   Kurtosis and standard error of the kurtosis.

SKEWNESS   Skewness and standard error of the skewness.

RANGE   Range.

MINIMUM   Minimum value.

MAXIMUM   Maximum value.

SUM   Sum.

DEFAULT   Mean, standard deviation of the mean, minimum, maximum.

SEKURTOSIS
             Standard error of the kurtosis.

SESKEWNESS
             Standard error of the skewness.

The SORT subcommand specifies how the statistics should be sorted. Most of the possible values should be self-explanatory. NAME causes the statistics to be sorted by name. By default, the statistics are listed in the order that they are specified on the VARIABLES subcommand. The A and D settings request an ascending or descending sort order, respectively.

13.2 FREQUENCIES

```
FREQUENCIES
      /VARIABLES=var_list
      /FORMAT={TABLE,NOTABLE,LIMIT(limit)}
            {STANDARD,CONDENSE,ONEPAGE[(onepage_limit)]}
            {LABELS,NOLABELS}
            {AVALUE,DVALUE,AFREQ,DFREQ}
            {SINGLE,DOUBLE}
            {OLDPAGE,NEWPAGE}
      /MISSING={EXCLUDE,INCLUDE}
    /STATISTICS={DEFAULT,MEAN,SEMEAN,MEDIAN,MODE,STDDEV,VARIANCE,
            KURTOSIS,SKEWNESS,RANGE,MINIMUM,MAXIMUM,SUM,
            SESKEWNESS,SEKURTOSIS,ALL,NONE}
      /NTILES=ntiles
      /PERCENTILES=percent...
      /HISTOGRAM=[MINIMUM(x_min)] [MAXIMUM(x_max)]
            [{FREQ,PCNT}] [{NONORMAL,NORMAL}]
    /PIECHART=[MINIMUM(x_min)] [MAXIMUM(x_max)] {NOMISSING,MISSING}

  (These options are not currently implemented.)
      /BARCHART=...
      /HBAR=...
      /GROUPED=...
```

The FREQUENCIES procedure outputs frequency tables for specified variables. FRE-QUENCIES can also calculate and display descriptive statistics (including median and mode) and percentiles.

FREQUENCIES also support graphical output in the form of histograms and pie charts. In the future, it will be able to produce bar charts and output percentiles for grouped data.

The VARIABLES subcommand is the only required subcommand. Specify the variables to be analyzed.

The FORMAT subcommand controls the output format. It has several possible settings:

- TABLE, the default, causes a frequency table to be output for every variable specified. NOTABLE prevents them from being output. LIMIT with a numeric argument causes them to be output except when there are more than the specified number of values in the table.

- STANDARD frequency tables contain more complete information, but also to take up more space on the printed page. CONDENSE frequency tables are less informative but take up less space. ONEPAGE with a numeric argument will output standard frequency tables if there are the specified number of values or less, condensed tables otherwise. ONEPAGE without an argument defaults to a threshold of 50 values.

- LABELS causes value labels to be displayed in STANDARD frequency tables. NO-LABLES prevents this.

- Normally frequency tables are sorted in ascending order by value. This is AVALUE. DVALUE tables are sorted in descending order by value. AFREQ and DFREQ tables are sorted in ascending and descending order, respectively, by frequency count.

- SINGLE spaced frequency tables are closely spaced. DOUBLE spaced frequency tables have wider spacing.

- OLDPAGE and NEWPAGE are not currently used.

The MISSING subcommand controls the handling of user-missing values. When EX-CLUDE, the default, is set, user-missing values are not included in frequency tables or statistics. When INCLUDE is set, user-missing are included. System-missing values are never included in statistics, but are listed in frequency tables.

The available STATISTICS are the same as available in DESCRIPTIVES (see Section 13.1 [DESCRIPTIVES], page 177), with the addition of MEDIAN, the data's median value, and MODE, the mode. (If there are multiple modes, the smallest value is reported.) By default, the mean, standard deviation of the mean, minimum, and maximum are reported for each variable.

PERCENTILES causes the specified percentiles to be reported. The percentiles should be presented at a list of numbers between 0 and 100 inclusive. The NTILES subcommand causes the percentiles to be reported at the boundaries of the data set divided into the specified number of ranges. For instance, `/NTILES=4` would cause quartiles to be reported.

The HISTOGRAM subcommand causes the output to include a histogram for each specified variable. The X axis by default ranges from the minimum to the maximum value observed in the data, but the MINIMUM and MAXIMUM keywords can set an explicit range. The Y axis by default is labeled in frequencies; use the PERCENT keyword to causes it to be labeled in percent of the total observed count. Specify NORMAL to superimpose a normal curve on the histogram.

The PIECHART adds a pie chart for each variable to the data. Each slice represents one value, with the size of the slice proportional to the value's frequency. By default, all non-missing values are given slices. The MINIMUM and MAXIMUM keywords can be used to limit the displayed slices to a given range of values. The MISSING keyword adds slices for missing values.

13.3 EXAMINE

```
EXAMINE
        VARIABLES=var_list [BY factor_list ]
        /STATISTICS={DESCRIPTIVES, EXTREME[(n)], ALL, NONE}
        /PLOT={BOXPLOT, NPPLOT, HISTOGRAM, ALL, NONE}
        /CINTERVAL n
        /COMPARE={GROUPS,VARIABLES}
        /ID={case_number, var_name}
        /{TOTAL,NOTOTAL}
         /PERCENTILE=[value_list]={HAVERAGE, WAVERAGE, ROUND, AEM-
    PIRICAL, EMPIRICAL }
        /MISSING={LISTWISE, PAIRWISE} [{EXCLUDE, INCLUDE}]
    [{NOREPORT,REPORT}]
```

The EXAMINE command is used to test how closely a distribution is to a normal distribution. It also shows you outliers and extreme values.

The VARIABLES subcommand specifies the dependent variables and the independent variable to use as factors for the analysis. Variables listed before the first BY keyword are the dependent variables. The dependent variables may optionally be followed by a list of factors which tell PSPP how to break down the analysis for each dependent variable. The format for each factor is

    var [BY var].

The STATISTICS subcommand specifies the analysis to be done. DESCRIPTIVES will produce a table showing some parametric and non-parametrics statistics. EXTREME produces a table showing extreme values of the dependent variable. A number in parentheses determines how many upper and lower extremes to show. The default number is 5.

The PLOT subcommand specifies which plots are to be produced if any.

The COMPARE subcommand is only relevant if producing boxplots, and it is only useful there is more than one dependent variable and at least one factor. If /COMPARE=GROUPS is specified, then one plot per dependent variable is produced, containing boxplots for all the factors. If /COMPARE=VARIABLES is specified, then one plot per factor is produced, each each containing one boxplot per dependent variable. If the /COMPARE subcommand is ommitted, then PSPP uses the default value of /COMPARE=GROUPS.

The CINTERVAL subcommand specifies the confidence interval to use in calculation of the descriptives command. The default it 95%.

The PERCENTILES subcommand specifies which percentiles are to be calculated, and which algorithm to use for calculating them. The default is to calculate the 5, 10, 25, 50, 75, 90, 95 percentiles using the HAVERAGE algorithm.

The TOTAL and NOTOTAL subcommands are mutually exclusive. If NOTOTAL is given and factors have been specified in the VARIABLES subcommand, then then statistics for the unfactored dependent variables are produced in addition to the factored variables. If there are no factors specified then TOTAL and NOTOTAL have no effect.

Warning! If many dependent variable are given, or factors are given for which there are many distinct values, then EXAMINE will produce a very large quantity of output.

13.4 CROSSTABS

```
CROSSTABS
      /TABLES=var_list BY var_list [BY var_list]...
      /MISSING={TABLE,INCLUDE,REPORT}
      /WRITE={NONE,CELLS,ALL}
      /FORMAT={TABLES,NOTABLES}
            {LABELS,NOLABELS,NOVALLABS}
            {PIVOT,NOPIVOT}
            {AVALUE,DVALUE}
            {NOINDEX,INDEX}
            {BOX,NOBOX}
      /CELLS={COUNT,ROW,COLUMN,TOTAL,EXPECTED,RESIDUAL,SRESIDUAL,
            ASRESIDUAL,ALL,NONE}
      /STATISTICS={CHISQ,PHI,CC,LAMBDA,UC,BTAU,CTAU,RISK,GAMMA,D,
            KAPPA,ETA,CORR,ALL,NONE}


(Integer mode.)
      /VARIABLES=var_list (low,high)...
```

The CROSSTABS procedure displays crosstabulation tables requested by the user. It can calculate several statistics for each cell in the crosstabulation tables. In addition, a number of statistics can be calculated for each table itself.

The TABLES subcommand is used to specify the tables to be reported. Any number of dimensions is permitted, and any number of variables per dimension is allowed. The TABLES subcommand may be repeated as many times as needed. This is the only required subcommand in *general mode*.

Occasionally, one may want to invoke a special mode called *integer mode*. Normally, in general mode, PSPP automatically determines what values occur in the data. In integer mode, the user specifies the range of values that the data assumes. To invoke this mode, specify the VARIABLES subcommand, giving a range of data values in parentheses for each variable to be used on the TABLES subcommand. Data values inside the range are truncated to the nearest integer, then assigned to that value. If values occur outside this range, they are discarded. When it is present, the VARIABLES subcommand must precede the TABLES subcommand.

In general mode, numeric and string variables may be specified on TABLES. Although long string variables are allowed, only their initial short-string parts are used. In integer mode, only numeric variables are allowed.

The MISSING subcommand determines the handling of user-missing values. When set to TABLE, the default, missing values are dropped on a table by table basis. When set to

INCLUDE, user-missing values are included in tables and statistics. When set to REPORT, which is allowed only in integer mode, user-missing values are included in tables but marked with an 'M' (for "missing") and excluded from statistical calculations.

Currently the WRITE subcommand is ignored.

The FORMAT subcommand controls the characteristics of the crosstabulation tables to be displayed. It has a number of possible settings:

- TABLES, the default, causes crosstabulation tables to be output. NOTABLES suppresses them.
- LABELS, the default, allows variable labels and value labels to appear in the output. NOLABELS suppresses them. NOVALLABS displays variable labels but suppresses value labels.
- PIVOT, the default, causes each TABLES subcommand to be displayed in a pivot table format. NOPIVOT causes the old-style crosstabulation format to be used.
- AVALUE, the default, causes values to be sorted in ascending order. DVALUE asserts a descending sort order.
- INDEX/NOINDEX is currently ignored.
- BOX/NOBOX is currently ignored.

The CELLS subcommand controls the contents of each cell in the displayed crosstabulation table. The possible settings are:

COUNT   Frequency count.

ROW   Row percent.

COLUMN   Column percent.

TOTAL   Table percent.

EXPECTED
          Expected value.

RESIDUAL
          Residual.

SRESIDUAL
          Standardized residual.

ASRESIDUAL
          Adjusted standardized residual.

ALL   All of the above.

NONE   Suppress cells entirely.

'/CELLS' without any settings specified requests COUNT, ROW, COLUMN, and TO-TAL. If CELLS is not specified at all then only COUNT will be selected.

The STATISTICS subcommand selects statistics for computation:

CHISQ
          Pearson chi-square, likelihood ratio, Fisher's exact test, continuity correction, linear-by-linear association.

PHI   Phi.

CC   Contingency coefficient.

LAMBDA   Lambda.

UC   Uncertainty coefficient.

BTAU   Tau-b.

CTAU   Tau-c.

RISK   Risk estimate.

GAMMA   Gamma.

D   Somers' D.

KAPPA   Cohen's Kappa.

ETA   Eta.

CORR   Spearman correlation, Pearson's r.

ALL   All of the above.

NONE   No statistics.

Selected statistics are only calculated when appropriate for the statistic. Certain statistics require tables of a particular size, and some statistics are calculated only in integer mode.

'/STATISTICS' without any settings selects CHISQ. If the STATISTICS subcommand is not given, no statistics are calculated.

Please note: Currently the implementation of CROSSTABS has the followings bugs:

- Pearson's R (but not Spearman) is off a little.
- T values for Spearman's R and Pearson's R are wrong.
- Significance of symmetric and directional measures is not calculated.
- Asymmetric ASEs and T values for lambda are wrong.
- ASE of Goodman and Kruskal's tau is not calculated.
- ASE of symmetric somers' d is wrong.
- Approximate T of uncertainty coefficient is wrong.

Fixes for any of these deficiencies would be welcomed.

13.5 NPAR TESTS

NPAR TESTS

nonparametric test subcommands
.
.
.

[/STATISTICS={DESCRIPTIVES}]

[/MISSING={ANALYSIS, LISTWISE} {INCLUDE, EXCLUDE}]

NPAR TESTS performs nonparametric tests. Non parametric tests make very few assumptions about the distribution of the data. One or more tests may be specified by using the corresponding subcommand. If the /STATISTICS subcommand is also specified, then summary statistics are produces for each variable that is the subject of any test.

13.5.1 Binomial test

[/BINOMIAL[(p)]=var_list[(value1[, value2)]]]

The binomial test compares the observed distribution of a dichotomous variable with that of a binomial distribution. The variable p specifies the test proportion of the binomial distribution. The default value of 0.5 is assumed if p is omitted.

If a single value appears after the variable list, then that value is used as the threshold to partition the observed values. Values less than or equal to the threshold value form the first category. Values greater than the threshold form the second category.

If two values appear after the variable list, then they will be used as the values which a variable must take to be in the respective category. Cases for which a variable takes a value equal to neither of the specified values, take no part in the test for that variable.

If no values appear, then the variable must assume dichotomous values. If more than two distinct, non-missing values for a variable under test are encountered then an error occurs.

If the test proportion is equal to 0.5, then a two tailed test is reported. For any other test proportion, a one tailed test is reported. For one tailed tests, if the test proportion is less than or equal to the observed proportion, then the significance of observing the observed proportion or more is reported. If the test proportion is more than the observed proportion, then the significance of observing the observed proportion or less is reported. That is to say, the test is always performed in the observed direction.

PSPP uses a very precise approximation to the gamma function to compute the binomial significance. Thus, exact results are reported even for very large sample sizes.

13.5.2 Chisquare test

[/CHISQUARE=var_list[(lo,hi)] [/EXPECTED={EQUAL|f1, f2 ... fn}]]

The chisquare test produces a chi-square statistic for the differences between the expected and observed frequencies of the categories of a variable. Optionally, a range of values may appear after the variable list. If a range is given, then non integer values are truncated, and values outside the specified range are excluded from the analysis.

The /EXPECTED subcommand specifies the expected values of each category. There must be exactly one non-zero expected value, for each observed category, or the EQUAL keywork must be specified. You may use the notation $n*f$ to specify n consecutive expected categories all taking a frequency of f. The frequencies given are proportions, not absolute frequencies. The sum of the frequencies need not be 1. If no /EXPECTED subcommand is given, then then equal frequencies are expected.

13.6 T-TEST

        T-TEST
                /MISSING={ANALYSIS,LISTWISE} {EXCLUDE,INCLUDE}
                /CRITERIA=CIN(confidence)

        (One Sample mode.)
                TESTVAL=test_value
                /VARIABLES=var_list

        (Independent Samples mode.)
                GROUPS=var(value1 [, value2])
                /VARIABLES=var_list

        (Paired Samples mode.)
                PAIRS=var_list [WITH var_list [(PAIRED)]]

The T-TEST procedure outputs tables used in testing hypotheses about means. It operates in one of three modes:

- One Sample mode.

- Independent Groups mode.

- Paired mode.

Each of these modes are described in more detail below. There are two optional subcommands which are common to all modes.

The /CRITERIA subcommand tells PSPP the confidence interval used in the tests. The default value is 0.95.

The MISSING subcommand determines the handling of missing variables. If INCLUDE is set, then user-missing values are included in the calculations, but system-missing values are not. If EXCLUDE is set, which is the default, user-missing values are excluded as well as system-missing values. This is the default.

If LISTWISE is set, then the entire case is excluded from analysis whenever any variable specified in the /VARIABLES, /PAIRS or /GROUPS subcommands contains a missing value. If ANALYSIS is set, then missing values are excluded only in the analysis for which they would be needed. This is the default.

13.6.1 One Sample Mode

The TESTVAL subcommand invokes the One Sample mode. This mode is used to test a population mean against a hypothesised mean. The value given to the TESTVAL subcommand is the value against which you wish to test. In this mode, you must also use the /VARIABLES subcommand to tell PSPP which variables you wish to test.

13.6.2 Independent Samples Mode

The GROUPS subcommand invokes Independent Samples mode or 'Groups' mode. This mode is used to test whether two groups of values have the same population mean. In this mode, you must also use the /VARIABLES subcommand to tell PSPP the dependent variables you wish to test.

The variable given in the GROUPS subcommand is the independent variable which determines to which group the samples belong. The values in parentheses are the specific values of the independent variable for each group. If the parentheses are omitted and no values are given, the default values of 1.0 and 2.0 are assumed.

If the independent variable is numeric, it is acceptable to specify only one value inside the parentheses. If you do this, cases where the independent variable is greater than or equal to this value belong to the first group, and cases less than this value belong to the second group. When using this form of the GROUPS subcommand, missing values in the independent variable are excluded on a listwise basis, regardless of whether /MISSING=LISTWISE was specified.

13.6.3 Paired Samples Mode

The PAIRS subcommand introduces Paired Samples mode. Use this mode when repeated measures have been taken from the same samples. If the `WITH` keyword is omitted, then tables for all combinations of variables given in the PAIRS subcommand are generated. If the `WITH` keyword is given, and the `(PAIRED)` keyword is also given, then the number of variables preceding `WITH` must be the same as the number following it. In this case, tables for each respective pair of variables are generated. In the event that the `WITH` keyword is given, but the `(PAIRED)` keyword is omitted, then tables for each combination of variable preceding `WITH` against variable following `WITH` are generated.

13.7 ONEWAY

```
ONEWAY
        [/VARIABLES = ] var_list BY var
        /MISSING={ANALYSIS,LISTWISE} {EXCLUDE,INCLUDE}
        /CONTRAST= value1 [, value2] ... [,valueN]
        /STATISTICS={DESCRIPTIVES,HOMOGENEITY}
```

The ONEWAY procedure performs a one-way analysis of variance of variables factored by a single independent variable. It is used to compare the means of a population divided into more than two groups.

The variables to be analysed should be given in the `VARIABLES` subcommand. The list of variables must be followed by the `BY` keyword and the name of the independent (or factor) variable.

You can use the `STATISTICS` subcommand to tell PSPP to display ancilliary information. The options accepted are:

- DESCRIPTIVES Displays descriptive statistics about the groups factored by the independent variable.
- HOMOGENEITY Displays the Levene test of Homogeneity of Variance for the variables and their groups.

The `CONTRAST` subcommand is used when you anticipate certain differences between the groups. The subcommand must be followed by a list of numerals which are the coefficients of the groups to be tested. The number of coefficients must correspond to the number of distinct groups (or values of the independent variable). If the total sum of the coefficients are not zero, then PSPP will display a warning, but will proceed with the analysis. The `CONTRAST` subcommand may be given up to 10 times in order to specify different contrast tests.

13.8 RANK

        RANK
                [VARIABLES=] var_list [{A,D}] [BY var_list]
                /TIES={MEAN,LOW,HIGH,CONDENSE}
                /FRACTION={BLOM,TUKEY,VW,RANKIT}
                /PRINT[={YES,NO}
                /MISSING={EXCLUDE,INCLUDE}

                /RANK [INTO var_list]
                /NTILES(k) [INTO var_list]
                /NORMAL [INTO var_list]
                /PERCENT [INTO var_list]
                /RFRACTION [INTO var_list]
                /PROPORTION [INTO var_list]
                /N [INTO var_list]
                /SAVAGE [INTO var_list]

The RANK command ranks variables and stores the results into new variables.

The VARIABLES subcommand, which is mandatory, specifies one or more variables whose values are to be ranked. After each variable, 'A' or 'D' may appear, indicating that the variable is to be ranked in ascending or descending order. Ascending is the default. If a BY keyword appears, it should be followed by a list of variables which are to serve as group variables. In this case, the cases are gathered into groups, and ranks calculated for each group.

The TIES subcommand specifies how tied values are to be treated. The default is to take the mean value of all the tied cases.

The FRACTION subcommand specifies how proportional ranks are to be calculated. This only has any effect if NORMAL or PROPORTIONAL rank functions are requested.

The PRINT subcommand may be used to specify that a summary of the rank variables created should appear in the output.

The function subcommands are RANK, NTILES, NORMAL, PERCENT, RFRACTION, PROPORTION and SAVAGE. Any number of function subcommands may appear. If none are given, then the default is RANK. The NTILES subcommand must take an integer specifying the number of partitions into which values should be ranked. Each subcommand may be followed by the INTO keyword and a list of variables which are the variables to be created and receive the rank scores. There may be as many variables specified as there are variables named on the VARIABLES subcommand. If fewer are specified, then the variable names are automatically created.

The MISSING subcommand determines how user missing values are to be treated. A setting of EXCLUDE means that variables whose values are user-missing are to be excluded from the rank scores. A setting of INCLUDE means they are to be included. The default is EXCLUDE.

13.9 REGRESSION

The REGRESSION procedure fits linear models to data via least-squares estimation. The procedure is appropriate for data which satisfy those assumptions typical in linear regression:

- The data set contains n observations of a dependent variable, say $Y_1, \ldots, Y_n$, and n observations of one or more explanatory variables. Let $X_{11}, X_{12}, \ldots, X_{1n}$ denote the n observations of the first explanatory variable; $X_{21}, \ldots, X_{2n}$ denote the n observations of the second explanatory variable; $X_{k1}, \ldots, X_{kn}$ denote the n observations of the kth explanatory variable.

- The dependent variable Y has the following relationship to the explanatory variables: $Y_i = b_0 + b_1 X_{1i} + \ldots + b_k X_{ki} + Z_i$ where $b_0, b_1, \ldots, b_k$ are unknown coefficients, and $Z_1, \ldots, Z_n$ are independent, normally distributed "noise" terms with mean zero and common variance. The noise, or "error" terms are unobserved. This relationship is called the "linear model."

The REGRESSION procedure estimates the coefficients $b_0, \ldots, b_k$ and produces output relevant to inferences for the linear model.

13.9.1 Syntax

```
REGRESSION
        /VARIABLES=var_list
        /DEPENDENT=var_list
        /STATISTICS={ALL, DEFAULTS, R, COEFF, ANOVA, BCOV}
        /SAVE={PRED, RESID}
```

The REGRESSION procedure reads the active file and outputs statistics relevant to the linear model specified by the user.

The VARIABLES subcommand, which is required, specifies the list of variables to be analyzed. Keyword VARIABLES is required. The DEPENDENT subcommand specifies the dependent variable of the linear model. The DEPENDENT subcommand is required. All variables listed in the VARIABLES subcommand, but not listed in the DEPENDENT subcommand, are treated as explanatory variables in the linear model.

All other subcommands are optional:

The STATISTICS subcommand specifies the statistics to be displayed:

ALL   All of the statistics below.

R   The ratio of the sums of squares due to the model to the total sums of squares for the dependent variable.

COEFF   A table containing the estimated model coefficients and their standard errors.

ANOVA   Analysis of variance table for the model.

BCOV   The covariance matrix for the estimated model coefficients.

   The SAVE subcommand causes PSPP to save the residuals or predicted values from the fitted model to the active file. PSPP will store the residuals in a variable called RES1 if no such variable exists, RES2 if RES1 already exists, RES3 if RES1 and RES2 already exist, etc. It will choose the name of the variable for the predicted values similarly, but with PRED as a prefix.

13.9.2 Examples

The following PSPP syntax will generate the default output and save the predicted values and residuals to the active file.

```
title 'Demonstrate REGRESSION procedure'.
data list / v0 1-2 (A) v1 v2 3-22 (10).
begin data.
b  7.735648 -23.97588
b  6.142625 -19.63854
a  7.651430 -25.26557
c  6.125125 -16.57090
a  8.245789 -25.80001
c  6.031540 -17.56743
a  9.832291 -28.35977
c  5.343832 -16.79548
a  8.838262 -29.25689
b  6.200189 -18.58219
end data.
list.
regression /variables=v0 v1 v2 /statistics defaults /dependent=v2
           /save pred resid /method=enter.
```

14 Utilities

Commands that don't fit any other category are placed here.

Most of these commands are not affected by commands like IF and LOOP: they take effect only once, unconditionally, at the time that they are encountered in the input.

14.1 ADD DOCUMENT

        ADD DOCUMENT
            'line one' 'line two' . . . 'last line' .

ADD DOCUMENT adds one or more lines of descriptive commentary to the active file. Documents added in this way are saved to system files. They can be viewed using SYSFILE INFO or DISPLAY DOCUMENTS. They can be removed from the active file with DROP DOCUMENTS.

Each line of documentary text must be enclosed in quotation marks, and may not be more than 80 bytes long. See Section 14.4 [DOCUMENT], page 190.

14.2 CD

        CD 'new directory' .

CD changes the current directory. The new directory will become that specified by the command.

14.3 COMMENT

        Two possibles syntaxes:
                COMMENT comment text . . . .
                *comment text . . . .

COMMENT is ignored. It is used to provide information to the author and other readers of the PSPP syntax file.

COMMENT can extend over any number of lines. Don't forget to terminate it with a dot or a blank line.

14.4 DOCUMENT

        DOCUMENT *documentary_text*.

DOCUMENT adds one or more lines of descriptive commentary to the active file. Documents added in this way are saved to system files. They can be viewed using SYSFILE INFO or DISPLAY DOCUMENTS. They can be removed from the active file with DROP DOCUMENTS.

Specify the *documentary text* following the DOCUMENT keyword. It is interpreted literally — any quotes or other punctuation marks will be included in the file. You can extend the documentary text over as many lines as necessary. Lines are truncated at 80 bytes. Don't forget to terminate the command with a dot or a blank line. See Section 14.1 [ADD DOCUMENT], page 190.

14.5 DISPLAY DOCUMENTS

DISPLAY DOCUMENTS.

DISPLAY DOCUMENTS displays the documents in the active file. Each document is preceded by a line giving the time and date that it was added. See Section 14.4 [DOCU-MENT], page 190.

14.6 DISPLAY FILE LABEL

DISPLAY FILE LABEL.

DISPLAY FILE LABEL displays the file label contained in the active file, if any. See Section 14.11 [FILE LABEL], page 191.

This command is a PSPP extension.

14.7 DROP DOCUMENTS

DROP DOCUMENTS.

DROP DOCUMENTS removes all documents from the active file. New documents can be added with DOCUMENT (see Section 14.4 [DOCUMENT], page 190).

DROP DOCUMENTS changes only the active file. It does not modify any system files stored on disk.

14.8 ECHO

ECHO 'arbitrary text' .

Use ECHO to write arbitrary text to the output stream. The text should be enclosed in quotation marks following the normal rules for string tokens (see Section 5.1 [Tokens], page 92).

14.9 ERASE

ERASE FILE file_name.

ERASE FILE deletes a file from the local filesystem. file_name must be quoted. This command cannot be used if the SAFER setting is active.

14.10 EXECUTE

EXECUTE.

EXECUTE causes the active file to be read and all pending transformations to be executed.

14.11 FILE LABEL

FILE LABEL file_label.

FILE LABEL provides a title for the active file. This title will be saved into system files and portable files that are created during this PSPP run.

file_label need not be quoted. If quotes are included, they become part of the file label.

14.12 FINISH

> FINISH.

FINISH terminates the current PSPP session and returns control to the operating system.

14.13 HOST

> HOST.

HOST suspends the current PSPP session and temporarily returns control to the operating system. This command cannot be used if the SAFER setting is active.

14.14 INCLUDE

> INCLUDE [FILE=]'file-name'.

INCLUDE causes the PSPP command processor to read an additional command file as if it were included bodily in the current command file. If errors are encountered in the included file, then command processing will stop and no more commands will be processed. Include files may be nested to any depth, up to the limit of available memory.

The INSERT command (see Section 14.15 [INSERT], page 192) may be used instead of INCLUDE if you require more flexible options. The syntax

```
INCLUDE FILE=file-name.
```

functions identically to

```
INSERT FILE=file-name ERROR=STOP CD=NO SYNTAX=BATCH.
```

14.15 INSERT

> INSERT [FILE=]'file-name'
>   [CD={NO,YES}]
>   [ERROR={CONTINUE,STOP}]
>   [SYNTAX={BATCH,INTERACTIVE}].

INSERT is similar to INCLUDE (see Section 14.14 [INCLUDE], page 192) but somewhat more flexible. It causes the command processor to read a file as if it were embedded in the current command file.

If 'CD=YES' is specified, then before including the file, the current directory will be changed to the directory of the included file. The default setting is 'CD=NO'. Note that this directory will remain current until it is changed explicitly (with the CD command, or a subsequent INSERT command with the 'CD=YES' option). It will not revert to its original setting even after the included file is finished processing.

If 'ERROR=STOP' is specified, errors encountered in the inserted file will cause processing to immediately cease. Otherwise processing will continue at the next command. The default setting is 'ERROR=CONTINUE'.

If 'SYNTAX=INTERACTIVE' is specified then the syntax contained in the included file must conform to interactive syntax conventions. See Section 5.3 [Syntax Variants], page 94. The default setting is 'SYNTAX=BATCH'.

14.16 PERMISSIONS

      PERMISSIONS
            FILE='file-name'
            /PERMISSIONS = {READONLY,WRITEABLE}.

PERMISSIONS changes the permissions of a file. There is one mandatory subcommand which specifies the permissions to which the file should be changed. If you set a file's permission to READONLY, then the file will become unwritable either by you or anyone else on the system. If you set the permission to WRITEABLE, then the file will become writeable by you; the permissions afforded to others will be unchanged. This command cannot be used if the SAFER setting is active.

14.17 SET

      SET

      (data input)
            /BLANKS={SYSMIS,'.',number}
            /DECIMAL={DOT,COMMA}
            /FORMAT=fmt_spec
            /EPOCH={AUTOMATIC,year}
            /RIB={NATIVE,MSBFIRST,LSBFIRST,VAX}
            /RRB={NATIVE,ISL,ISB,IDL,IDB,VF,VD,VG,ZS,ZL}

      (program input)
            /ENDCMD='.'
            /NULLINE={ON,OFF}

      (interaction)
            /CPROMPT='cprompt_string'
            /DPROMPT='dprompt_string'
            /ERRORBREAK={OFF,ON}
            /MXERRS=max_errs
            /MXWARNS=max_warnings
            /PROMPT='prompt'
            /WORKSPACE=workspace_size

      (program execution)
            /MEXPAND={ON,OFF}
            /MITERATE=max_iterations
            /MNEST=max_nest
            /MPRINT={ON,OFF}
            /MXLOOPS=max_loops
            /SEED={RANDOM,seed_value}
            /UNDEFINED={WARN,NOWARN}

      (data output)

```
/CC{A,B,C,D,E}={'npre,pre,suf,nsuf','npre.pre.suf.nsuf'}
/DECIMAL={DOT,COMMA}
/FORMAT=fmt_spec
/WIB={NATIVE,MSBFIRST,LSBFIRST,VAX}
/WRB={NATIVE,ISL,ISB,IDL,IDB,VF,VD,VG,ZS,ZL}
```

(output routing)
```
/ECHO={ON,OFF}
/ERRORS={ON,OFF,TERMINAL,LISTING,BOTH,NONE}
/INCLUDE={ON,OFF}
/MESSAGES={ON,OFF,TERMINAL,LISTING,BOTH,NONE}
/PRINTBACK={ON,OFF}
/RESULTS={ON,OFF,TERMINAL,LISTING,BOTH,NONE}
```

(output driver options)
```
/HEADERS={NO,YES,BLANK}
/LENGTH={NONE,length_in_lines}
/LISTING={ON,OFF,'file-name'}
/MORE={ON,OFF}
/WIDTH={NARROW,WIDTH,n_characters}
```

(logging)
```
/JOURNAL={ON,OFF} ['file-name']
```

(system files)
```
/COMPRESSION={ON,OFF}
/SCOMPRESSION={ON,OFF}
```

(security)
```
/SAFER=ON
```

(obsolete settings accepted for compatibility, but ignored)
```
/BOXSTRING={'xxx','xxxxxxxxxxx'}
/CASE={UPPER,UPLOW}
/CPI=cpi_value
/DISK={ON,OFF}
/HIGHRES={ON,OFF}
/HISTOGRAM='c'
/LOWRES={AUTO,ON,OFF}
/LPI=lpi_value
/MENUS={STANDARD,EXTENDED}
/MXMEMORY=max_memory
/SCRIPTTAB='c'
/TB1={'xxx','xxxxxxxxxxx'}
/TBFONTS='string'
/XSORT={YES,NO}
```

SET allows the user to adjust several parameters relating to PSPP's execution. Since there are many subcommands to this command, its subcommands will be examined in groups.

On subcommands that take boolean values, ON and YES are synonym, and as are OFF and NO, when used as subcommand values.

The data input subcommands affect the way that data is read from data files. The data input subcommands are

BLANKS   This is the value assigned to an item data item that is empty or contains only white space. An argument of SYSMIS or '.' will cause the system-missing value to be assigned to null items. This is the default. Any real value may be assigned.

DECIMAL

The default DOT setting causes the decimal point character to be '.' and the grouping character to be ','. A setting of COMMA causes the decimal point character to be ',' and the grouping character to be '.'.

FORMAT   Allows the default numeric input/output format to be specified. The default is F8.2. See Section 5.7.4 [Input and Output Formats], page 98.

EPOCH   Specifies the range of years used when a 2-digit year is read from a data file or used in a date construction expression (see Section 6.7.8.4 [Date Construction], page 118). If a 4-digit year is specified for the epoch, then 2-digit years are interpreted starting from that year, known as the epoch. If AUTOMATIC (the default) is specified, then the epoch begins 69 years before the current date.

RIB

PSPP extension to set the byte ordering (endianness) used for reading data in IB or PIB format (see Section 5.7.4.4 [Binary and Hexadecimal Numeric Formats], page 103). In MSBFIRST ordering, the most-significant byte appears at the left end of a IB or PIB field. In LSBFIRST ordering, the least-significant byte appears at the left end. VAX ordering is like MSBFIRST, except that each pair of bytes is in reverse order. NATIVE, the default, is equivalent to MSBFIRST or LSBFIRST depending on the native format of the machine running PSPP.

RRB

PSPP extension to set the floating-point format used for reading data in RB format (see Section 5.7.4.4 [Binary and Hexadecimal Numeric Formats], page 103). The possibilities are:

NATIVE   The native format of the machine running PSPP. Equivalent to either IDL or IDB.

ISL   32-bit IEEE 754 single-precision floating point, in little-endian byte order.

ISB   32-bit IEEE 754 single-precision floating point, in big-endian byte order.

IDL   64-bit IEEE 754 double-precision floating point, in little-endian byte order.

IDB   64-bit IEEE 754 double-precision floating point, in big-endian byte order.

VF   32-bit VAX F format, in VAX-endian byte order.

VD   64-bit VAX D format, in VAX-endian byte order.

VG   64-bit VAX G format, in VAX-endian byte order.

ZS   32-bit IBM Z architecture short format hexadecimal floating point, in big-endian byte order.

ZL   64-bit IBM Z architecture long format hexadecimal floating point, in big-endian byte order.

             Z architecture also supports IEEE 754 floating point. The ZS and ZL formats are only for use with very old input files.

The default is NATIVE.

Program input subcommands affect the way that programs are parsed when they are typed interactively or run from a command file. They are

ENDCMD This is a single character indicating the end of a command. The default is '.'. Don't change this.

NULLINE Whether a blank line is interpreted as ending the current command. The default is ON.

Interaction subcommands affect the way that PSPP interacts with an online user. The interaction subcommands are

CPROMPT

             The command continuation prompt. The default is ' > '.

DPROMPT

             Prompt used when expecting data input within BEGIN DATA (see Section 7.1 [BEGIN DATA], page 128). The default is '`data>` '.

ERRORBREAK

             Whether an error causes PSPP to stop processing the current command file after finishing the current command. The default is OFF.

MXERRS   The maximum number of errors before PSPP halts processing of the current command file. The default is 50.

MXWARNS

             The maximum number of warnings + errors before PSPP halts processing the current command file. The default is 100.

PROMPT   The command prompt. The default is '`PSPP>` '.

Program execution subcommands control the way that PSPP commands execute. The program execution subcommands are

MEXPAND
MITERATE
MNEST
MPRINT   Currently not used.

MXLOOPS

> The maximum number of iterations for an uncontrolled loop (see Section 12.4 [LOOP], page 175).

SEED   The initial pseudo-random number seed. Set to a real number or to RANDOM, which will obtain an initial seed from the current time of day.

UNDEFINED

> Currently not used.

WORKSPACE

> The maximum amount of memory that PSPP will use to store data being processed. If memory in excess of the workspace size is required, then PSPP will start to use temporary files to store the data. Setting a higher value will, in general, mean procedures will run faster, but may cause other applications to run slower. On platforms without virtual memory management, setting a very large workspace may cause PSPP to abort.

Data output subcommands affect the format of output data. These subcommands are

CCA
CCB
CCC
CCD
CCE

> Set up custom currency formats. See Section 5.7.4.2 [Custom Currency Formats], page 101, for details.

DECIMAL

> The default DOT setting causes the decimal point character to be '.'. A setting of COMMA causes the decimal point character to be ','.

FORMAT Allows the default numeric input/output format to be specified. The default is F8.2. See Section 5.7.4 [Input and Output Formats], page 98.

WIB

> PSPP extension to set the byte ordering (endianness) used for writing data in IB or PIB format (see Section 5.7.4.4 [Binary and Hexadecimal Numeric Formats], page 103). In MSBFIRST ordering, the most-significant byte appears at the left end of a IB or PIB field. In LSBFIRST ordering, the least-significant byte appears at the left end. VAX ordering is like MSBFIRST, except that each pair of bytes is in reverse order. NATIVE, the default, is equivalent to MSBFIRST or LSBFIRST depending on the native format of the machine running PSPP.

WRB

> PSPP extension to set the floating-point format used for writing data in RB format (see Section 5.7.4.4 [Binary and Hexadecimal Numeric Formats], page 103). The choices are the same as SET RIB. The default is NATIVE.

Output routing subcommands affect where the output of transformations and procedures is sent. These subcommands are

ECHO

> If turned on, commands are written to the listing file as they are read from command files. The default is OFF.

ERRORS
INCLUDE
MESSAGES
PRINTBACK
RESULTS Currently not used.

Output driver option subcommands affect output drivers' settings. These subcommands are

HEADERS
LENGTH
LISTING
MORE
PAGER
WIDTH

Logging subcommands affect logging of commands executed to external files. These subcommands are

JOURNAL
LOG   These subcommands, which are synonyms, control the journal. The default is ON, which causes commands entered interactively to be written to the journal file. Commands included from syntax files that are included interactively and error messages printed by PSPP are also written to the journal file, prefixed by '>'. OFF disables use of the journal.

> The journal is named 'pspp.jnl' by default. A different name may be specified.

System file subcommands affect the default format of system files produced by PSPP. These subcommands are

COMPRESSION

> Not currently used.

SCOMPRESSION

> Whether system files created by SAVE or XSAVE are compressed by default. The default is ON.

Security subcommands affect the operations that commands are allowed to perform. The security subcommands are

SAFER   Setting this option disables the following operations:

- The ERASE command.
- The HOST command.
- The PERMISSIONS command.
- Pipes (file names beginning or ending with '|').

> Be aware that this setting does not guarantee safety (commands can still overwrite files, for instance) but it is an improvement. When set, this setting cannot be reset during the same session, for obvious security reasons.

14.18 SHOW

SHOW
>   [ALL]
>   [BLANKS]
>   [CC]
>   [CCA]
>   [CCB]
>   [CCC]
>   [CCD]
>   [CCE]
>   [COPYING]
>   [DECIMALS]
>   [ENDCMD]
>   [FORMAT]
>   [LENGTH]
>   [MXERRS]
>   [MXLOOPS]
>   [MXWARNS]
>   [SCOMPRESSION]
>   [UNDEFINED]
>   [WARRANTY]
>   [WEIGHT]
>   [WIDTH]

SHOW can be used to display the current state of PSPP's execution parameters. Parameters that can be changed using SET (see Section 14.17 [SET], page 193), can be examined using SHOW using the subcommand with the same name. `SHOW` supports the following additional subcommands:

ALL   Show all settings.

CC   Show all custom currency settings (CCA through CCE).

WARRANTY   Show details of the lack of warranty for PSPP.

COPYING   Display the terms of PSPP's copyright licence (see Chapter 2 [License], page 2).

Specifying SHOW without any subcommands is equivalent to SHOW ALL.

14.19 SUBTITLE

SUBTITLE 'subtitle_string'.
  or
SUBTITLE subtitle_string.

SUBTITLE provides a subtitle to a particular PSPP run. This subtitle appears at the top of each output page below the title, if headers are enabled on the output device.

Specify a subtitle as a string in quotes. The alternate syntax that did not require quotes is now obsolete. If it is used then the subtitle is converted to all uppercase.

14.20 TITLE

TITLE 'title_string'.
 or
TITLE title_string.

TITLE provides a title to a particular PSPP run. This title appears at the top of each output page, if headers are enabled on the output device.

Specify a title as a string in quotes. The alternate syntax that did not require quotes is now obsolete. If it is used then the title is converted to all uppercase.

15 Not Implemented

This chapter lists parts of the PSPP language that are not yet implemented.

2SLS   Two stage least squares regression

ACF   Autocorrelation function

ADD FILES
            Add files to dictionary

ALSCAL   Multidimensional scaling

ANACOR   Correspondence analysis

ANOVA   Factorial analysis of variance

CASEPLOT
            Plot time series

CASESTOVARS
            Restructure complex data

CATPCA   Categorical principle components analysis

CATREG   Categorical regression

CCF   Time series cross correlation

CLEAR TRANSFORMATIONS
            Clears transformations from active file

CLUSTER
            Hierachial clustering

CONJOINT
            Analyse full concept data

CORRELATIONS
            Correlation coefficients

CORRESPONDENCE
            Show correspondence

COXREG   Cox proportional hazards regression

CREATE   Create time series data

CSDESCRIPTIVES
            Complex samples descriptives

CSGLM   Complex samples GLM

CSLOGISTIC
            Complex samples logistic regression

CSPLAN   Complex samples design

CSSELECT
            Select complex samples

CSTABULATE
          Tabulate complex samples

CTABLES
          Display complex samples

CURVEFIT
          Fit curve to line plot

DATAFILE ATTRIBUTE
          User defined datafile attributes

DATASET
          Alternate data set

DATE   Create time series data

DEFINE   Syntax macros

DETECTANOMALY
          Find unusual cases

DISCRIMINANT
          Linear discriminant analysis

EDIT   obsolete

END FILE TYPE
          Ends complex data input

FACTOR   Factor analysis

FILE TYPE
          Complex data input

FIT   Goodness of Fit

GENLOG   Categorical model fitting

GET TRANSLATE
          Read other file formats

GGRAPH   Custom defined graphs

GLM   General Linear Model

GRAPH   Draw graphs

HILOGLINEAR
          Hierarchial loglinear models

HOMALS   Homogeneity analysis

IGRAPH   Interactive graphs

INFO   Local Documentation

INSERT   Insert file

KEYED DATA LIST
          Read nonsequential data

KM   Kaplan-Meier

LOGISTIC REGRESSION
            Regression Analysis

LOGLINEAR
            General model fitting

MANOVA   Multivariate analysis of variance

MAPS   Geographical display

MATRIX   Matrix processing

MATRIX DATA
            Matrix data input

MCONVERT
            Convert covariance/correlation matrices

MIXED   Mixed linear models

MODEL CLOSE
            Close server connection

MODEL HANDLE
            Define server connection

MODEL LIST
            Show existing models

MODEL NAME
            Specify model label

MRSETS   Multiple response sets

MULTIPLE CORRESPONDENCE
            Multiple correspondence analysis

MULT RESPONSE
            Multiple reponse analysis

MVA   Missing value analysis

NAIVEBAYES
            Small sample bayesian prediction

NLR   Non Linear Regression

NOMREG
            Multinomial logistic regression

NONPAR CORR
            Nonparametric correlation

NUMBERED
OLAP CUBES
            On-line analytical processing

OMS   Output management

ORTHOPLAN
             Orthogonal effects design

OVERALS
             Nonlinear canonical correlation

PACF   Partial autocorrelation

PARTIAL CORR
             Partial correlation

PEARSON CORRELATIONS
             Correlation coefficients

PLANCARDS
             Conjoint analysis planning

PLUM   Estimate ordinal regression models

POINT   Marker in keyed file

PPLOT   Plot time series variables

PREDICT
             Specify forecast period

PREFSCAL
             Multidimensional unfolding

PRESERVE
             Push settings

PRINCALS
             PCA by alternating least squares

PROBIT   Probit analysis

PROCEDURE OUTPUT
             Specify output file

PROXIMITIES
             Pairwise similarity

PROXSCAL
             Multidimensional scaling of proximity data

QUICK CLUSTER
             Fast clustering

RATIO STATISTICS
             Descriptives of ratios

READ MODEL
             Read new model

RECORD TYPE
             Defines a type of record within FILE TYPE

REFORMAT
        Read obsolete files

RELIABILITY
        Reliability estimates

REPEATING DATA
        Specify multiple cases per input record

REPORT   Pretty print working file

RESTORE
        Restore settings

RMV   Replace missing values

ROC   Receiver operating characteristic

SAVE TRANSLATE
        Save to foriegn format

SCRIPT   Run script file

SEASON   Estimate seasonal factors

SELECTPRED
        Select predictor variables

SPCHART
        Plot control charts

SPECTRA
        Plot spectral density

SUMMARIZE
        Univariate statistics

SURVIVAL
        Survival analysis

TDISPLAY
        Display active models

TREE   Create classification tree

TSAPPLY
        Apply time series model

TSET   Set time sequence variables

TSHOW   Show time sequence variables

TSMODEL
        Estimate time series model

TSPLOT   Plot time sequence variables

TWOSTEP CLUSTER
        Cluster observations

UNIANOVA
       Univariate analysis

UNNUMBERED
       obsolete

UPDATE   Update working file

VALIDATEDATA
       Identify suspicious cases

VARCOMP
       Estimate variance

VARSTOCASES
       Restructure complex data

VERIFY   Report time series

WLS   Weighted least squares regression

XGRAPH   High resolution charts

16 Bugs

PSPP does have bugs. We do our best to fix them, but our limited resources mean that some may remain for a long time. Our best alternative is to make you aware of PSPP's known bugs. To see a list, visit PSPP's project webpage at `https://savannah.gnu.org/projects/pspp`. You can also submit your own bug report there: click on "Bugs," then on "Submit a Bug," and fill out the form. Alternatively, PSPP bug reports may be sent by email to `<bug-gnu-pspp@gnu.org>`.

For known bugs in individual language features, see the documentation for that feature.

17 Function Index

V

X

Y

18 Command Index

19 Concept Index

Appendix A Configuring PSPP

This chapter describe how to configure PSPP for your system.

A.1 Locating configuration files

PSPP searches each directory in the configuration file path for most configuration files. The default configuration file path searches first '`$HOME/.pspp`', then the package system configuration directory (usually '`/usr/local/etc/pspp`' or '`/etc/pspp`'). The value of environment variable `PSPP_CONFIG_PATH`, if defined, overrides this default path. Finally, '`-B path`' or '`--config-dir=path`' specified on the command line has highest priority.

A.2 Configuration techniques

There are many ways that PSPP can be configured. These are described in the list below. Values given by earlier items take precedence over those given by later items.

1. Syntax commands that modify settings, such as SET. See Section 14.17 [SET], page 193.
2. Command-line options. See Chapter 3 [Invocation], page 3.
3. PSPP-specific environment variable contents. See Section A.4 [Environment variables], page 220.
4. General environment variable contents. See Section A.4 [Environment variables], page 220.
5. Configuration file contents. See Section A.3 [Configuration files], page 219.
6. Fallback defaults.

   Some of the above may not apply to a particular setting.

A.3 Configuration files

Most configuration files have a common form:

- Each line forms a separate command or directive. This means that lines cannot be broken up, unless they are spliced together with a trailing backslash, as described below.
- Before anything else is done, trailing white space is removed.
- When a line ends in a backslash ('`\`'), the backslash is removed, and the next line is read and appended to the current line.
  - White space preceding the backslash is retained.
  - This rule continues to be applied until the line read does not end in a backslash.
  - It is an error if the last line in the file ends in a backslash.
- Comments are introduced by an octothorpe ('`#`'), and continue until the end of the line.
  - An octothorpe inside balanced pairs of double quotation marks ('"') or single quotation marks ('') does not introduce a comment.
  - The backslash character can be used inside balanced quotes of either type to escape the following character as a literal character.

    (This is distinct from the use of a backslash as a line-splicing character.)

> − Line splicing takes place before comment removal.

- Blank lines, and lines that contain only white space, are ignored.

A.4 Environment variables

You may think the concept of environment variables is a fairly simple one. However, the author of PSPP has found a way to complicate even something so simple. Environment variables are further described in the sections below:

A.4.1 Environment substitutions

Much of the power of environment variables lies in the way that they may be substituted into configuration files. Variable substitutions are described below.

The line is scanned from left to right. In this scan, all characters other than dollar signs ('`$`') are retained without change. Dollar signs introduce environment variable references. References take three forms:

`$var`   Replaced by the value of environment variable *var*. *var* must consist of either one or more letters, or exactly one non-alphabetic character other than a left brace ('`{`').

`${var}`   Same as above, but *var* may contain any character (except '`}`').

`$$`   Replaced by a single dollar sign.

Undefined variables expand to a empty value.

A.4.2 Predefined environment variables

There are two environment variables predefined for use in environment substitutions:

'`VER`'   Defined as the version number of PSPP, as a string, in a format something like '`0.9.4`'.

'`ARCH`'   Defined as the host architecture of PSPP, as a string, in standard cpu-manufacturer-OS format. For instance, Debian GNU/Linux 1.1 on an Intel machine defines this as '`i586-unknown-linux`'. This is somewhat dependent on the system used to compile PSPP.

Nothing prevents these values from being overridden, although it's a good idea not to do so.

A.5 Output devices

Configuring output devices is the most complicated aspect of configuring PSPP. The output device configuration file is named '`devices`'. It is searched for using the usual algorithm for finding configuration files (see Section A.1 [File locations], page 219). Each line in the file is read in the usual manner for configuration files (see Section A.3 [Configuration files], page 219).

Lines in '`devices`' are divided into three categories, described briefly in the table below:

driver category definitions
        Define a driver in terms of other drivers.

macro definitions

> Define environment variables local to the output driver configuration file.

device definitions

> Describe the configuration of an output device.

The following sections further elaborate the contents of the '`devices`' file.

A.5.1 Driver categories

Drivers can be divided into categories. Drivers are specified by their names, or by the names of the categories that they are contained in. Only certain drivers are enabled each time PSPP is run; by default, these are the drivers in the category 'default'. To enable a different set of drivers, use the '`-o device`' command-line option (see Chapter 3 [Invocation], page 3).

Categories are specified with a line of the form '`category=driver1 driver2 driver3 ... drivern`'. This line specifies that the category *category* is composed of drivers named *driver1*, *driver2*, and so on. There may be any number of drivers in the category, from zero on up.

Categories may also be specified on the command line (see Chapter 3 [Invocation], page 3).

This is all you need to know about categories. If you're still curious, read on.

First of all, the term 'categories' is a bit of a misnomer. In fact, the internal representation is nothing like the hierarchy that the term seems to imply: a linear list is used to keep track of the enabled drivers.

When PSPP first begins reading '`devices`', this list contains the name of any drivers or categories specified on the command line, or the single item 'default' if none were specified.

Each time a category definition is specified, the list is searched for an item with the value of *category*. If a matching item is found, it is deleted. If there was a match, the list of drivers (*driver1* through *drivern*) is then appended to the list.

Each time a driver definition line is encountered, the list is searched. If the list contains an item with that driver's name, the driver is enabled and the item is deleted from the list. Otherwise, the driver is not enabled.

It is an error if the list is not empty when the end of '`devices`' is reached.

A.5.2 Macro definitions

Macro definitions take the form '`define macroname definition`'. In such a macro definition, the environment variable *macroname* is defined to expand to the value *definition*. Before the definition is made, however, any macros used in *definition* are expanded.

Please note the following nuances of macro usage:

- For the purposes of this section, *macro* and *environment variable* are synonyms.
- Macros may not take arguments.
- Macros may not recurse.
- Macros are just environment variable definitions like other environment variable definitions, with the exception that they are limited in scope to the '`devices`' configuration file.

- Macros override other all environment variables of the same name (within the scope of 'devices').

- Earlier macro definitions for a particular *key* override later ones. In particular, macro definitions on the command line override those in the device definition file. See Section 3.1 [Non-option Arguments], page 3.

- There are two predefined macros, whose values are determined at runtime:

  'viewwidth'
          Defined as the width of the console screen, in columns of text.

  'viewlength'
          Defined as the length of the console screen, in lines of text.

A.5.3 Driver definitions

Driver definitions are the ultimate purpose of the 'devices' configuration file. These are where the real action is. Driver definitions tell PSPP where it should send its output.

Each driver definition line is divided into four fields. These fields are delimited by colons (':'). Each line is subjected to environment variable interpolation before it is processed further (see Section A.4.1 [Environment substitutions], page 220). From left to right, the four fields are, in brief:

driver name
          A unique identifier, used to determine whether to enable the driver.

class name
          One of the predefined driver classes supported by PSPP. The currently supported driver classes include 'postscript' and 'ascii'.

device type(s)
          Zero or more of the following keywords, delimited by spaces:

          screen
                          Indicates that the device is a screen display. This may reduce the amount of buffering done by the driver, to make interactive use more convenient.

          printer
                          Indicates that the device is a printer.

          listing
                          Indicates that the device is a listing file.

          These options are just hints to PSPP and do not cause the output to be directed to the screen, or to the printer, or to a listing file—those must be set elsewhere in the options. They are used primarily to decide which devices should be enabled at any given time. See Section 14.17 [SET], page 193, for more information.

options   An optional set of options to pass to the driver itself. The exact format for the options varies among drivers.

The driver is enabled if:

1. Its driver name is specified on the command line, or

2. It's in a category specified on the command line, or

3. If no categories or driver names are specified on the command line, it is in category `default`.

For more information on driver names, see Section A.5.1 [Driver categories], page 221.

The class name must be one of those supported by PSPP. The classes supported depend on the options with which PSPP was compiled. See later sections in this chapter for descriptions of the available driver classes.

Options are dependent on the driver. See the driver descriptions for details.

A.5.4 Dimensions

Quite often in configuration it is necessary to specify a length or a size. PSPP uses a common syntax for all such, calling them collectively by the name *dimensions*.

- You can specify dimensions in decimal form ('`12.5`') or as fractions, either as mixed numbers ('`12-1/2`') or raw fractions ('`25/2`').

- A number of different units are available. These are suffixed to the numeric part of the dimension. There must be no spaces between the number and the unit. The available units are identical to those offered by the popular typesetting system TeX:

  `in`   inch (1 `in` = 2.54 `cm`)

  `"`   inch (1 `in` = 2.54 `cm`)

  `pt`   printer's point (1 `in` = 72.27 `pt`)

  `pc`   pica (12 `pt` = 1 `pc`)

  `bp`   PostScript point (1 `in` = 72 `bp`)

  `cm`   centimeter

  `mm`   millimeter (10 `mm` = 1 `cm`)

  `dd`   didot point (1157 `dd` = 1238 `pt`)

  `cc`   cicero (1 `cc` = 12 `dd`)

  `sp`   scaled point (65536 `sp` = 1 `pt`)

- If no explicit unit is given, PSPP attempts to guess the best unit:
  - Numbers less than 50 are assumed to be in inches.
  - Numbers 50 or greater are assumed to be in millimeters.

A.5.5 How lines are divided into types

The lines in '`devices`' are distinguished in the following manner:

1. Leading white space is removed.

2. If the resulting line begins with the exact string `define`, followed by one or more white space characters, the line is processed as a macro definition.

3. Otherwise, the line is scanned for the first instance of a colon ('`:`') or an equals sign ('`=`').

4. If a colon is encountered first, the line is processed as a driver definition.

5. Otherwise, if an equals sign is encountered, the line is processed as a macro definition.

6. Otherwise, the line is ill-formed.

A.5.6 How lines are divided into tokens

Each driver definition line is run through a simple tokenizer. This tokenizer recognizes two basic types of tokens.

The first type is an equals sign ('='). Equals signs are both delimiters between tokens and tokens in themselves.

The second type is an identifier or string token. Identifiers and strings are equivalent after tokenization, though they are written differently. An identifier is any string of characters other than white space or equals sign.

A string is introduced by a single- or double-quote character ('' or '"') and, in general, continues until the next occurrence of that same character. The following standard C escapes can also be embedded within strings:

\'   A single-quote ('').

\"   A double-quote ('"').

\?   A question mark ('?'). Included for hysterical raisins.

\\   A backslash ('\').

\a   Audio bell (ASCII 7).

\b   Backspace (ASCII 8).

\f   Formfeed (ASCII 12).

\n   New-line (ASCII 10)

\r   Carriage return (ASCII 13).

\t   Tab (ASCII 9).

\v   Vertical tab (ASCII 11).

\\ooo\   Each 'o' must be an octal digit. The character is the one having the octal value specified. Any number of octal digits is read and interpreted; only the lower 8 bits are used.

\x\hh\   Each 'h' must be a hex digit. The character is the one having the hexadecimal value specified. Any number of hex digits is read and interpreted; only the lower 8 bits are used.

Tokens, outside of quoted strings, are delimited by white space or equals signs.

A.6 The PostScript driver class

The `postscript` driver class is used to produce output that is acceptable to PostScript printers and other interpreters.

The available options are listed below.

`output-file=`*file-name*

> File to which output should be sent. This can be an ordinary file name (i.e., `"pspp.ps"`), a pipe (i.e., `"|lpr"`), or stdout (`"-"`). Default: `"pspp.ps"`.

`headers=`*boolean*

> Controls whether the standard headers showing the time and date and title and subtitle are printed at the top of each page. Default: `on`.

`paper-size=`*paper-size*

> Paper size. You may specify a name (e.g. `a4`, `letter`) or measurements (e.g. `210x297`, `8.5x11in`).
>
> The default paper size is taken from the `PAPERSIZE` environment variable or the file indicated by the `PAPERCONF` environment variable, if either variable is set. If not, and your system supports the `LC_PAPER` locale category, then the default paper size is taken from the locale. Otherwise, if '`/etc/papersize`' exists, the default paper size is read from it. As a last resort, A4 paper is assumed.

`orientation=`*orientation*

> Either `portrait` or `landscape`. Default: `portrait`.

`left-margin=`*dimension*
`right-margin=`*dimension*
`top-margin=`*dimension*
`bottom-margin=`*dimension*

> Sets the margins around the page. The headers, if enabled, are not included in the margins; they are in addition to the margins. For a description of dimensions, see Section A.5.4 [Dimensions], page 223. Default: `0.5in`.

`prop-font=`*afm-file*`[,`*font-file*`[,`*encoding-file*`]]`
`emph-font=`*afm-file*`[,`*font-file*`[,`*encoding-file*`]]`
`fixed-font=`*afm-file*`[,`*font-file*`[,`*encoding-file*`]]`

> Sets the font used for proportional, emphasized, or fixed-pitch text. The only required value is *afm-file*, the AFM file for the font.
>
> If specified, *font-file* will be downloaded to the printer at the beginning of the print job. The font file may be in PFA or PFB format.
>
> The font is reencoded as specified in *encoding-file*, if specified. Each line in *encoding-file* should consist of a PostScript character name and a decimal encoding value (between 0 and 255), separated by white space. Blank lines and comments introduced by '`#`' are also allowed.
>
> The files specified on these options are located as follows. If the file name begins with '`/`', then it is taken as an absolute path. Otherwise, PSPP searches its configuration path for the specified name prefixed by `psfonts/` (see Section A.1 [File locations], page 219).

Default: proportional font `Times-Roman.afm`, emphasis font `Times-Italic.afm`, fixed-pitch font `Courier.afm`.

`font-size=`*font-size*

Sets the size of the default fonts, in thousandths of a point. Default: 10000 (10 point).

`line-gutter=`*dimension*

Sets the width of white space on either side of lines that border text or graphics objects. See . Default: `1pt`.

`line-spacing=`*dimension*

Sets the spacing between the lines in a double line in a table. Default: `1pt`.

`line-width=`*dimension*

Sets the width of the lines used in tables. Default: `0.5pt`.

A.7 The ASCII driver class

The ASCII driver class produces output that can be displayed on a terminal or output to printers. The ASCII driver has class name 'ascii'.

The available options are listed below.

`output-file=`*file-name*

File to which output should be sent. This can be an ordinary file name (e.g., `"pspp.txt"`), a pipe (e.g., `"|more"`), or stdout (`"-"`). Default: `"pspp.list"`.

`chart-files=`*file-name-template*

Template for the file names used for charts. The name should contain a single '#', which is replaced by the chart number. Default: '`"pspp-#.png"`'.

`chart-type=`*type*.

Type of charts to output. Available types typically include 'X', 'png', 'gif', 'svg', 'ps', 'cgm', 'fig', 'pcl', 'hpgl', 'regis', 'tek', and 'meta'. Default: 'png'.

You may specify 'none' to disable chart output. Charts are also disabled if your installation of PSPP was compiled without `libplot`.

`paginate=`*boolean*

If set, a formfeed will be written at the end of every page. Default: `on`.

`tab-width=`*tab-width-value*

The distance between tab stops for this device. If set to 0, tabs will not be used in the output. Default: `8`.

`headers=`*boolean*

If enabled, two lines of header information giving title and subtitle, page number, date and time, and PSPP version are printed at the top of every page. These two lines are in addition to any top margin requested. Default: `on`.

`length=`*line-count*

Physical length of a page. Headers and margins are subtracted from this value. You may specify the number of lines as a number, or for screen output you may specify `auto` to track the height of the terminal as it changes. Default: `66`.

`width=`*`character-count`*

        Physical width of a page. Margins are subtracted from this value. You may specify the width as a number of characters, or for screen output you may specify `auto` to track the width of the terminal as it changes. Default: `79`.

`top-margin=`*`top-margin-lines`*

        Length of the top margin, in lines. PSPP subtracts this value from the page length. Default: `2`.

`bottom-margin=`*`bottom-margin-lines`*

        Length of the bottom margin, in lines. PSPP subtracts this value from the page length. Default: `2`.

`box[`*`line-type`*`]=`*`box-chars`*

        The characters used for lines in tables produced by the ASCII driver can be changed using this option. *line-type* is used to indicate which type of line to change; *box-chars* is the character or string of characters to use for this type of line.

        line-type must be a 4-digit number. The digits are in the order 'right', 'bottom', 'left', 'top'. The possibilities for each digit are:

0   No line.

1   Single line.

2   Double line.

        Examples:

`box[0101]="|"`

        Sets '|' as the character to use for a single-width line with bottom and top components.

`box[2222]="#"`

        Sets '#' as the character to use for the intersection of four double-width lines, one each from the top, bottom, left and right.

`box[1100]="\xda"`

        Sets '`"\xda"`', which under MS-DOS is a box character suitable for the top-left corner of a box, as the character for the intersection of two single-width lines, one each from the right and bottom.

        Defaults:

- `box[0000]=" "`
- `box[1000]="-"`
  `box[0010]="-"`
  `box[1010]="-"`
- `box[0100]="|"`
  `box[0001]="|"`
  `box[0101]="|"`
- `box[2000]="="`
  `box[0020]="="`
  `box[2020]="="`

- `box[3000]="="`
  `box[0030]="="`
  `box[3030]="="`

- For all others, '`+`' is used unless there are double lines or special lines, in which case '`#`' is used.

`init=`*`init-string`*

> If set, this string is written at the beginning of each output file. It can be used to initialize device features, e.g. to enable VT100 line-drawing characters.

`emphasis=`*`emphasis-style`*

> How to emphasize text. Your choices are `bold`, `underline`, or `none`. Bold and underline emphasis are achieved with overstriking, which may not be supported by all the software to which you might pass the output.

A.8 The HTML driver class

The `html` driver class is used to produce output for viewing in tables-capable web browsers such as Emacs' w3-mode. Its configuration is very simple. Currently, the output has a very plain format. In the future, further work may be done on improving the output appearance.

There are only a few options:

`output-file=`*`file-name`*

> File to which output should be sent. This can be an ordinary file name (i.e., `"pspp.ps"`), a pipe (i.e., `"|lpr"`), or stdout (`"-"`). Default: '`"pspp.html"`'.

`chart-files=`*`file-name-template`*

> Template for the file names used for charts, which are output in PNG format. The name should contain a single '`#`', which is replaced by the chart number. Default: '`"pspp-#.png"`'.

A.9 Miscellaneous configuration

The following environment variables can be used to further configure PSPP:

`HOME`

> Used to determine the user's home directory. No default value.

`STAT_INCLUDE_PATH`

> Path used to find include files in PSPP syntax files. Defaults vary across operating systems:
>
> UNIX
>
> > - '`.`'
> > - '`$HOME/.pspp/include`'
> > - '`/usr/local/lib/pspp/include`'
> > - '`/usr/lib/pspp/include`'
> > - '`/usr/local/share/pspp/include`'
> > - '`/usr/share/pspp/include`'

MS-DOS

- '.'
- 'C:\PSPP\INCLUDE'
- '$PATH'

Other OSes

No default path.

TERM

The terminal type `termcap` or `ncurses` will use, if such support was compiled into PSPP.

STAT_OUTPUT_INIT_FILE

The basename used to search for the driver definition file. See Section A.5 [Output devices], page 220. See Section A.1 [File locations], page 219. Default: `devices`.

STAT_OUTPUT_INIT_PATH

The path used to search for the driver definition file. See Section A.1 [File locations], page 219. Default: the standard configuration path.

TMPDIR

The directory in which PSPP stores its temporary files (used when sorting cases or concatenating large numbers of cases). Default: (UNIX) '/tmp', (MS-DOS) '\', (other OSes) empty string.

TEMP

TMP

Under MS-DOS only, these variables are consulted after TMPDIR, in this order.

Appendix B GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.