Chapter 119

Transformations

Introduction

Transformations generate new data. They can be used to perform mathematical operations such as addition, multiplication, and exponentiation on columns. For example, you might want to create a new column that is the logarithm of the data in another column, the total of three columns, or the recoding of another column. A rich set of text transformations is also available. Transformation result columns can be used as variables in reports and plots just like any other columns in the dataset.

A transformation formula specifies how the new column is constructed from the other existing columns. The formula is made up of mathematical operators, columns, constants, and common mathematical functions. It is important to understand that these transformations modify columns of data, not individual cells. This is one place where the **NCSS** Data Table is different from traditional spreadsheets.

There are three general types of data transformations available: *Transformations (General)*, *Conditional (If-Then) Transformations*, and *Recode Transformations*. You can type any transformation formula directly into the Column Info Table, or you can enter it using one of three different Transformation Editor windows.

Here are some examples of the three different kinds of transformations:

Result Column	Transformation Formula	Type
C2	C1*100	General
C2	log(C1)	General
C3	C1+C2	General
C3	Sqrt(C1/C2)	General
C4	IfThen(C3=1, C1+C2; Else, 0)	Conditional (If-Then)
C4	Recode(C1; 1="Male", 2="Female")	Recode
C4	Recode(C2; :5="Low", 6:10="Med", Else="High")	Recode

Transformation Components

All transformations are made up of two components: the *result column* and the *transformation formula* or *function*. Each transformation is stored in only one result column. The formula may contain one or more references to other columns from which the result column will be calculated. The formula must not reference the result column. The result column is always locked when a transformation is active and cannot be edited except by changing values in columns included in the transformation formula. When you change the name of a column that is included in a transformation function, the function will be automatically updated with the new column name.

Types of Transformations

As stated earlier, there are three general types of data transformations available: *Transformations (General)*, *Conditional (If-Then) Transformations*, and *Recode Transformations*.

Transformations (General)

General transformations are the most basic and involve no conditional statements; general transformations are always applied to all rows. The syntax of transformation formulas is presented later in this chapter. The general form of a transformation is

Function(*A*,*B*,*C*,*D*)

where *A*, *B*, *C* and *D* are parameters which depend on the individual transformation functions. The actual number of required input parameters also depends on the individual transformation functions.

Non-English Language Setting

If you are using a computer language setting other than English that uses a comma as the decimal symbol, then transformation input parameters should be separated by the computer's specified list separator (this is usually a semicolon if not a comma) and groups should be separated with a vertical bar "|". The general syntax for a computer that uses a comma as the decimal symbol would be

Function(*A*;*B*;*C*;*D*)

Conditional (If-Then) Transformations

Conditional (If-Then) Transformations allow you to selectively apply transformations based on logical statements involving one or more data columns. The form of a conditional transformation is

IfThen([*Condition 1*], [*Result 1*]; [*Condition 2*], [*Result 2*]; ... ; Else, [*Result Else*])

where [*Condition i*] represents condition statements and [*Result i*] represents the transformation formula to apply when the corresponding condition evaluates to True. The word "Else" is used to designate the transformation formula to use when all the other conditions evaluate to False. An "Else" statement is not required; if it is omitted, then the result column will be left unchanged for all rows that do not meet any of the conditions.

The syntax of conditional transformation formula components is presented later in this chapter.

Non-English Language Setting

If you are using a computer language setting other than English that uses a comma as the decimal symbol, then each condition/result pair should be separated by the computer's specified list separator (this is usually a semicolon if not a comma) and groups should be separated with a vertical bar "|". The syntax for a computer that uses a comma as the decimal symbol would be

IfThen([*Condition 1*]; [*Result 1*] | [*Condition 2*]; [*Result 2*] | ... | Else; [*Result Else*])

Recode Transformations

Recode transformations allow you to convert values (e.g., 1= “Male”, 2= “Female”) and store the results in another column. The form of a recode transformation is

Recode([*Column*]; [*Value 1*] = [*Recode Value 1*], [*Value 2*] = [*Recode Value 2*], ...)

where [*Column*] represents the column to recode, [*Value i*] represents a value or range of values in the column to recode, and [*Recode Value i*] represents the result to which the value(s) in the column to recode will be changed.

Non-English Language Setting

If you are using a computer language setting other than English that uses a comma as the decimal symbol, then each value pair should be separated by the computer’s specified list separator (this is usually a semicolon if not a comma) and the column and value pairs should be separated with a vertical bar “|”. The syntax for a computer that uses a comma as the decimal symbol would be

Recode([*Column*] | [*Value 1*] = [*Recode Value 1*]; [*Value 2*] = [*Recode Value 2*], ...)

Transformation Editors

The Transformation Editor windows may be used to facilitate the entry of transformation formulas. You can think of these windows as special viewers for the Transformation column of the Column Info Table. The transformation editors are particularly useful when you don’t know which function to use or when creating conditional and recode transformations. Remember that you do not have to use the transformation editors to create transformations; you can type any transformation formula directly into the Column Info Table.

There are several ways to launch the transformation editors.

1. From the Data Window, select *Data* > *Transformations* and then one of the transformation types. You will be prompted to select a column in which to store the transformation.
2. From the Data Window, select *Data* > *Column X Info* > *Transformation* to create/edit a transformation for the column containing the active cell.
3. Click on the button at the right side of a transformation cell when it is selected in the Column Info Table to create/edit a transformation for the column containing the active cell.
4. Select a cell in the Data Table, right-click, and select *Column X Info* > *Edit Transformation* from the popup menu.

This editor that is launched depends on the type of transformation in the active column. If there is no transformation formula for the selected column, the general transformation editor will appear.

Calculation

The calculation of transformations is executed depending on the *Auto Recalculate Transformations* option located on the Transformation Toolbar (To view the Transformation Toolbar, select *View* > *Transformation Toolbar* from the Data Window menu). If the option is checked then transformations will be applied automatically (this is the default setting); otherwise, transformations will only be applied when the user clicks the *Recalculate All Transformations* button. When *Auto Recalculate Transformations* is on, all transformations will be recalculated every time one of the columns included in the transformations is changed. When *Auto Recalculate Transformations* is turned off, the user will be prompted to recalculate transformations whenever a column that is involved in a transformation is edited (the column background in the Data Table will become light red and a *Recalc All Transformations* button will appear on the bottom bar).

Order of Calculation

When transformations are calculated, the program will determine the correct order in which to execute transformations so that any result columns that are involved in other formulas will be calculated before downstream formulas. For example, if C3 contains the transformation “C1+C2” and C4 contains the transformation “C3*100”, then the transformation in C3 should be executed before C4 because C4 depends on C3. The transformations do not have to be in any specific order going left to right for the calculations to happen correctly; **NCSS** figures out the correct order for you.

Activating and Deactivating Transformations

When you enter a transformation into a column, that column will become locked to data editing, and you will not be able to change data for that column in the Data Table without first deactivating the transformation. You will know that a column contains a transformation and is locked by the function-lock icon that appears in the column header of the Data Table. The background of the column in the Data Table will also become light blue. You can deactivate a transformation without deleting it by loading the transformation editor window and checking *Deactivate Transformation* or by simply typing a single quote character in front of the transformation in the Column Info Table (e.g., 'C1+C2). Deactivating the transformation will unlock the result column for editing. Inactive transformations appear grayed out. The transformation can be reactivated at any time by loading transformation editor window and unchecking *Deactivate Transformation* or by removing the single quote mark from the beginning of the transformation in the Column Info Table. When a transformation is reactivated, any data in the column containing the transformation will be overwritten, so be careful.

Transformation Formula Syntax

Transformation functions must follow certain syntax in order for the program to interpret them correctly. Any functions that are not understood will be overlooked during calculation, and you will be notified of the error(s). The follow sections will describe appropriate syntax for creating transformation functions.

Arguments

Transformations are used to formulate new columns by manipulating existing data. The arguments of a function are the individual entities that make up the formula. For example, in the formula

3.2 + 54 / 22

the numbers 3.2, 54, and 22 are the arguments. Arguments may be numbers, text, columns, or another formula contained within parentheses. Each argument must produce a valid value or an error or missing value will result. When one formula is contained within another formula, it is said to be nested. You can nest as many functions within each other as you like.

Following are a few examples of transformation formulas. Note that the result column names are not included in the formulas.

C1+C2+C3
C1*4+2
Log(C2)
100*C1/(Sin(C2+4))

Operators

Formulas are made by combining arguments using operators. Common operators are addition, subtraction, multiplication, and division. A complete list of the operators available is given below.

Note that no two operators can appear consecutively. A formula like **C1+/C2** is illegal. However, you can enter **C4*-5** because **-5** is treated as a constant. The operators and parentheses serve as delimiters for the arguments of a function.

The standard order of calculation is used. This means that embedded functions are calculated first, then exponentiation, multiplication, division, addition, and subtraction. Hence, the formula **4+2*6** would yield **16**, not **36**!

Addition and Subtraction Operators

+ This is the addition symbol. In the formula **X + Y**, it signifies adding **X** to **Y**. Multiplications and divisions are performed before additions and subtractions. If either **X** or **Y** is missing, the result is missing.

- This is the subtraction symbol. In the formula **Y - X**, it signifies subtracting **X** from **Y**. If either **X** or **Y** is missing, the result is missing.

Multiplication and Division Operators

***** This is the multiplication symbol. In the formula **X * Y**, it signifies the product of **X** and **Y**. If either **X** or **Y** is missing, the result is missing.

/ This is the division symbol. In the formula **Y / X**, it signifies dividing **Y** by **X**. If either **X** or **Y** is missing, the result is missing. Also note that **X** cannot be equal to zero (in which case a missing value results).

Exponentiation Operator

^ This is the exponentiation symbol. In the formula **X ^ Y**, it signifies raising **X** to the power **Y**. If either **X** or **Y** is missing, the result is missing. Note that the result must be less than 10^308 in absolute value. Also note that **X** must be a nonnegative number.

Examples: 12^2 = 144; 2^3 = 8; 9^(1/2) = 3; 10^(-1) = 0.10.

Logic Operators

< This is the less-than symbol used in a logic expression. If **(X<Y)** is true, a one results. If **(X<Y)** is false, a zero results.

Examples: (4<6) = 1 and (6<4) = 0.

<= This is the less-than or equal-to symbol used in a logic expression. If **(X<=Y)** is true, a one results. If **(X<=Y)** is false, a zero results.

Examples: (4<=6) = 1; (6<=6) = 1; and (6<=4) = 0.

> This is the greater-than symbol used in a logic expression. If **(X>Y)** is true, a one results. If **(X>Y)** is false, a zero results.

Examples: (4>6) = 0 and (6>4) = 1.

>= This is the greater-than or equal-to symbol used in a logic expression. If **(X>=Y)** is true, a one results. If **(X>=Y)** is false, a zero results.

Examples: (4>=6) = 0; (6>=6) = 1; and (6>=4) = 1.

= This is the equal-to symbol used in a logic expression. If **(X=Y)** is true, a one results. If **(X=Y)** is false, a zero results.

Examples: (4=6) = 0 and (6=6) = 1.

<> This is the not-equal-to symbol used in a logic expression. If **(X<>Y)** is true, a one results. If **(X<>Y)** is false, a zero results.

Examples: (4<>6) = 1 and (6<>6) = 0.

Arrangement Operators

( ) The parentheses are used to force the order of precedence in an expression. Expressions inside a set of parentheses are evaluated first. Note that they must be used as a pair; each left parenthesis must have a corresponding right parenthesis or an error will result. For example, 3+4*2 = 11; while (3+4)*2 = 14.

{ } The braces are used interchangeably with the parentheses. You would usually use them to avoid confusion when multiple sets of parentheses are called for.

[ ] The brackets are used interchangeably with the parentheses. You would usually use them to avoid confusion when multiple sets of parentheses are called for.

Numeric Functions

Numeric functions have a name and one or more arguments contained within parentheses. Inside an expression, a function is treated just like a number or a column. The following functions may be used in combination with other expressions. You must be careful to include the correct number of arguments and to make sure that the values of the arguments are within appropriate limits. For example, you cannot have **SQRT(-5)**.

Date and Time Functions

Date (Month,Day,Year)

Returns a date-time value. **Month** is the number of the month (1 to 12). **Day** is the number of the day (1 to 31). **Year** is the number of the year (100 to 9999).

Example: **Date(9,23,2014)** yields **09/23/2014** (September 23, 2014)**.**

DateDT (DateTime)

Returns the date portion of a date-time value. **DateTime** is a date-time value.

Example: **DATEDT(09/23/2014 10:54:43 AM)** yields **09/23/2014**.

DateTime (Date,Time)

Returns a date-time value which combines the input date and time. This transformation can be used in conjunction with the Date and Time transformations. **Date** is a date-time value from which only the date portion will be extracted. **Time** is a date-time value from which only the time portion will be extracted.

Example: If **X is September 23, 2014**, and **Y is 10:26:52 AM**, then **DateTime(X,Y)** yields **09/23/2014 10:26:52 AM**.

DateToJulian (DateTime)

Converts a date-time value to a Julian date. **DateTime** is a date-time value.

Example: If **X is September 23, 2014**, **DateToJulian(X)** yields **2456924**.

Day (Julian)

Returns the number of the day in a Julian date. **Julian** is a Julian date.

Example: If **X is 2456924 (September 23, 2014)**, **Day(X)** yields **23**.

DayDT (DateTime)

Returns the number of the day in a date-time value. **DateTime** is a date-time value.

Example: If **X is September 23, 2014**, **DayDT(X)** yields **23**.

HourDT (DateTime)

Returns the number of the hour (24-hour format) in a date-time value. **DateTime** is a date-time value.

Examples: **HourDT(10:54:43 AM)** yields **10**.

HourDT(10:54:43 PM) yields **22**.

Julian (Month,Day,Year)

Returns a Julian date. **Month** is the number of the month (1 to 12). **Day** is the number of the day (1 to 31). **Year** is the number of the year (0 to 2050).

Example: **Julian(9,23,2014)** yields **2456924** (September 23, 2014)**.**

JulianToDate (Julian)

Converts a Julian date to a date-time value. **Julian** is a Julian date.

Example: If **X is 2456924**, **JulianToDate(X)** yields **09/23/2014** (September 23, 2014).

MinuteDT (DateTime)

Returns the number of the minute in a date-time value. **DateTime** is a date-time value.

Example: **MinuteDT(10:54:43 AM)** yields **54**.

Month (Julian)

Returns the number of the month in a Julian date. **Julian** is a Julian date.

Example: If **X is 2456924** (September 23, 2014), **Month(X)** yields **9**.

MonthDT (DateTime)

Returns the number of the month in a date-time value. **DateTime** is a date-time value.

Example: If **X is September 23, 2014**, **MonthDT(X)** yields **9**.

SecondDT (DateTime)

Returns the number of the second in a date-time value. **DateTime** is a date-time value.

Example: **SecondDT(10:54:43 AM)** yields **43**.

Time (Hour,Minute,Second)

Returns a date-time value. **Hour** is the number of the hour in 24-hour format (0 to 23). **Minute** is the number of the minute (0 to 59). **Second** is the number of the second (0 to 59).

Examples: **Time(10,26,52)** yields **10:26:52 AM**.

Time(17,26,52) yields **5:26:52 PM**.

TimeDT (DateTime)

Returns the time portion of a date-time value. **DateTime** is a date-time value.

Example: **TIMEDT(09/23/2014 10:54:43 AM)** yields **10:54:43 AM**.

Year (Julian)

Returns the number of the year in a Julian date. **Julian** is a Julian date.

Example: If **X is 2456924** (September 23, 2014), **Year(X)** yields **2014**.

YearDT (DateTime)

Returns the number of the year in a date-time value. **DateTime** is a date-time value.

Example: If **X is September 23, 2014**, **YearDT(X)** yields **2014**.

File Function

File (XYZ.txt)

This function returns the contents of the file specified between the parentheses. A path to the file may be included in the file name. If no path is specified, the path to the currently open database is used.

The transformation in the file may be a segment or a complete transformation. The contents of the file simply replace the file statement before the transformation is processed. The file may include several lines. When the file is loaded, all carriage returns and line feeds are removed as the file is read, so a multi-line file becomes a single line of text.

This statement is especially useful for handling long transformations that might not easily fit on a single line.

Mathematical Functions

Abs (X)

Returns the absolute value of **X**.

Example: **ABS(-4)** yields **4**.

Exp (X)

Returns the value of e raised to the power **X**. The opposite of log (base e) of **X**.

Fraction (X)

Returns the fractional (noninteger) portion of **X**. **X** is a number.

Example: **Fraction(5.1234)** yields **0.1234**.

Int (X)

Returns the integer portion (the whole number part) of **X**.

Ln (X)

Returns the logarithm (base e) of **X**. **X** is a number greater than zero.

Log (X)

Returns the log (base 10) of **X**. **X** is a number greater than zero.

LogBase (X,Base)

Returns the logarithm of **X** in base **Base**. **X** is a number > 0. **Base** is the base, a number > 0.

Logit (X)

Returns the logit of **X** which is **Ln(X/(1-X))**. **X** is a number between 0 and 1.

Mod (X,Y)

Returns the remainder after dividing **X** by **Y**. This is sometimes written as **X Mod Y**. **X** is a number. **Y** is a positive integer less than 255.

Example: **Mod(15,4)** yields **3**.

Round (X,D)

Returns a value rounded off to the nearest roundoff unit. **X** is the number to be rounded. **D** is the roundoff unit.

Example: **Round(10.432,.1)** yields **10.4**.

Short (X)

Returns a single-precision version of **X**.

X is the number to be changed.

Sign (X)

Returns the sign of **X**. If **X** is negative, the result is -1. If **X** is positive, the result is +1. If **X** is zero, the result is zero.

Example: **SIGN(-4.2)** yields **-1**.

Sqrt (X)

Returns the positive square root of **X**. **X** is a non-negative number.

Probability Functions

BetaProb (X,A,B)

Returns the probability of being less than **X** when **X** follows the Beta distribution. This is the left-tail probability. **X** is a real number between 0 and 1. **A** is a nonnegative number. **B** is a nonnegative number.

BetaValue (Prob,A,B)

Returns the inverse of the Beta distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a real number between 0 and 1. **A** is a nonnegative number. **B** is a nonnegative number.

BinomProb (R,N,P)

Returns the probability of being less than or equal to **R** when **R** follows the binomial distribution with sample size, **N**, and prob of success, **P**. This is the left-tail probability. **R** is a non-negative number less than or equal to **N**. **N** is the number of trials (sample size) and must be a positive integer. **P** is probability of a success on a particular trial. It must be between 0 and 1.

BinomValue (Prob,N,P)

Returns the inverse of the binomial distribution. It gives the number, **R**, such that the cumulative binomial probability is less than or equal to **Prob**. **Prob** is a real number between 0 and 1. **N** is the number of trials (sample size) and must be a positive integer. **P** is probability of a success on a particular trial. It must be between 0 and 1.

BinormProb (X,Y,Rho)

Returns the probability of being greater than **X** and greater than **Y** when **X** and **Y** follow the standardized bivariate-normal distribution. **X** is any real number. Usually, **|X|**<3. **Y** is any real number. Usually, **|Y|**<3. **Rho** is the population correlation coefficient between **X** and **Y**. **-1 < Rho < 1**.

CorrProb (R,N,Rho)

Returns the probability of being less than **R** when **R** follows the correlation distribution. This is the left-tail probability. **R** is any real number. **N** is the sample size. It must be a positive number. **Rho** is the population correlation coefficient. **-1 < Rho < 1**.

CorrValue (Prob,N,Rho)

Returns the inverse of the correlation distribution. It gives the value, **R**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **N** is the sample size. It must be a positive number. **Rho** is the population correlation coefficient. **-1 < Rho < 1**.

CsProb (X,Df)

Returns the probability of being less than **X** when **X** follows the Chi-Square distribution. This is the left-tail probability. **X** is a positive real number. **Df** is the degrees of freedom. It must be a positive number.

CsValue (Prob,Df)

Returns the inverse of the Chi-Square distribution. It gives the value, X, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df** is the degrees of freedom. It must be a positive number.

ExpoProb (T,Scale)

Returns the probability of being less than or equal to **T** when **T** follows the Exponential distribution. This is the left-tail probability. **T** is a positive number. **Scale** is the scale parameter.

ExpoValue (Prob,Scale)

Returns the inverse of the Exponential distribution. It gives the number, **T**, such that the Exponential probability is less than or equal to **Prob**. **Prob** is a real number between 0 and 1. **Scale** is the scale parameter.

Fprob (F,Df1,Df2)

Returns the probability of being less than **F** when **F** follows the F distribution. This is the left-tail probability. **F** is a positive real number. **Df1** is the numerator degrees of freedom. It must be a positive number. **Df2** is the denominator degrees of freedom. It must be a positive number.

Fvalue (Prob,Df1,Df2)

Returns the inverse of the F distribution. It gives the value, **F**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df1** is the numerator degrees of freedom. It must be a positive number. **Df2** is the denominator degrees of freedom. It must be a positive number.

GammaProb (X,A)

Returns the probability of being less than **X** when **X** follows the Gamma distribution. This is the left-tail probability. **X** is a non-negative number. **A** is a positive number.

GammaValue (Prob,A)

Returns the inverse of the Gamma distribution. It gives the value, **X,** such that the area to the left is equal to **Prob**. **Prob** is a real number between 0 and 1. **A** is a nonnegative number.

HypergeoProb (X,N,R,M)

Returns the probability of the hypergeometric distribution. This is the left-tail probability, including the current value of **X**. **X** is the number of successes, where **X>=0** and **X<=R**. **N** is the population size, where **N>=1**. **R** is the sample size, where **R>=1** and **R<=N**. **M** is the subgroup size, where **M>=1** and **M<=N.**

LogGamma (X)

Returns the natural logarithm of the gamma function. **X** is a positive number.

NcBetaProb (X,A,B,Ncp)

Returns the probability of being less than **X** when **X** follows the noncentral-Beta distribution. This is the left-tail probability. **X** is a real number between 0 and 1. **A** is a non-negative number. **B** is a non-negative number. **Ncp** is the noncentrality parameter.

NcBetaValue (Prob,A,B,Ncp)

Returns the inverse of the noncentral-Beta distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a real number between 0 and 1. **A** is a non-negative number. **B** is a non-negative number. **Ncp** is the noncentrality parameter.

NcCsProb (X,Df,Ncp)

Returns the probability of being less than **X** when **X** follows the noncentral-Chi-Square distribution. This is the left-tail probability. **X** is a positive real number. **Df** is the degrees of freedom. It must be a positive number. **Ncp** is the noncentrality parameter.

NcCsValue (Prob,Df,Ncp)

Returns the inverse of the noncentral-Chi-Square distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df** is the degrees of freedom. It must be a postive number. **Ncp** is the noncentrality parameter.

NcFprob (F,Df1,Df2,Ncp)

Returns the probability of being less than **F** when **F** follows the noncentral-F distribution. This is the left-tail probability. **F** is a positive real number. **Df1** is the numerator degrees of freedom. It must be a positive number. **Df2** is the denominator degrees of freedom. It must be a positive number. **Ncp** is the noncentrality parameter.

NcFvalue (Prob,Df1,Df2,Ncp)

Returns the inverse of the noncentral-F distribution. It gives the value, F, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df1** is the numerator degrees of freedom. It must be a positive number. **Df2** is the denominator degrees of freedom. It must be a positive number. **Ncp** is the noncentrality parameter.

NcTprob (T,Df,Ncp)

Returns the probability of being less than **T** when **T** follows the noncentral-t distribution. This is the left-tail probability. **T** is any real number. **Df** is the degrees of freedom. It must be a positive number. **Ncp** is the noncentrality parameter.

NcTvalue (Prob,Df,Ncp)

Returns the inverse of the noncentral-t distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df** is the degrees of freedom. It must be a positive number. **Ncp** is the noncentrality parameter.

NegBinomProb (X,R,P)

Returns the probability of the negative binomial distribution with number of successes: **R** and probability of success: **P**. This is the left-tail probability. **X** is a nonnegative number less than or equal to **R**. **R** is the number of trials resulting in a success and must be a positive integer. **P** is the probability of a success on a particular trial. It must be between 0 and 1.

NormalProb (Z)

Returns the probability of being less than **Z** when **Z** follows the standard-normal distribution. This is the left-tail probability. **Z** is any real number.

NormalValue (Prob)

Returns the inverse of the standard normal distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1.

PoissonProb (X,Mean)

Returns the probability of the Poisson distribution with given mean. This is the left-tail probability. **X** is a non-negative number. **Mean** is the mean number of occurrences per unit time.

StdRangeProb (R,Nm,Df)

Returns the probability of being less than or equal to **R** when **R** follows the Studentized-Range distribution. This is the left-tail probability. **R** is a positive number. **Nm** is the number of means included in the range. It must be between 2 and 200. **Df** is the degrees of freedom of the standard-error term used in the denominator.

StdRangeValue (Prob,Nm,Df)

Returns the inverse of the Studentized-Range distribution. It gives the number, **R**, such that the Studentized-Range probability is less than or equal to **Prob**. **Prob** is a real number between 0.90 and 0.99. **Nm** is the number of means included in the range. **Df** is the degrees of freedom of the standard-error term used in the denominator.

Tprob (T,Df)

Returns the probability of being less than **T** when **T** follows the Student's t distribution. This is the left-tail probability. **T** is any real number. **Df** is the degrees of freedom. It must be a positive number.

Tvalue (Prob,Df)

Returns the inverse of the Student's t distribution. It gives the value, **X**, such that the area to the left is equal to **Prob**. **Prob** is a probability value. It must be between 0 and 1. **Df** is the degrees of freedom. It must be a postive number.

WeibullProb (T,Scale,Shape)

Returns the probability of being less than or equal to **T** when **T** follows the Weibull distribution. This is the left-tail probability. **T** is a positive number. **Scale** is the scale parameter. **Shape** is the shape parameter.

WeibullValue (Prob,Scale,Shape)

Returns the inverse of the Weibull distribution. It gives the number, **T**, such that the Weibull probability is less than or equal to **Prob**. **Prob** is a real number between 0 and 1. **Scale** is the scale parameter. **Shape** is the shape parameter.

Random-Number Functions

RandomNormal ()

Returns a random number from the standard normal distribution (mean 0, sigma 1).

RandomUniform () or Uniform ()

Returns a random number from the uniform distribution. The number is between 0 and 1.

Random Sampling Functions

Random sampling transformations allow you to randomly sample values or rows from the dataset. Random numbers are generated using the Mersenne Twister algorithm with a random seed.

RandomSample (X,k)

Returns a random sample of **k** values from the column, **X**. This function cannot be used with any other function. **X** is the column from which to draw the sample. **k** is number of values to select.

Example: **RandomSample(X,50)** returns **50 randomly-selected values from X**.

RandomSampleBin (X,k)

Creates a binary indicator variable by randomly selecting **k** values from the column, **X**. 1's are returned for selected values and 0's are returned for all other values. This function cannot be used with any other function. **X** is the column from which to draw the sample. **k** is number of values to select.

Example: **RandomSampleBin(X,50)** returns a 1 for **50 randomly-selected values from X**.

RandomSampleRows (k)

Creates a binary indicator variable by randomly selecting k non-empty rows. 1's are returned for selected rows and 0's are returned for all other rows. This function cannot be used with any other function. **k** is number of rows to select.

Example: **RandomSampleRows(50)** returns a **1 for 50 non-empty, randomly-selected rows in the dataset**.

Rearrangement Functions

Collate (Type,X1,X2)

This function has been removed. To perform this operation, select Data then Stack Data from the Data Window menu.

Rank(X)

Returns the rank of the values in **X**. Ties are assigned the average rank. This function cannot be used with any other function. **X** is a column to be ranked.

Sort (X)

Returns a sorted version of the data in **X**. Note that only this column is sorted. This function cannot be used with any other function. **X** is the column to be sorted.

Splice (Type,X1,X2, ...)

This function has been removed. To perform this operation, select Data then Stack Data from the Data Window menu.

UnCollate (N,I,X)

This function has been removed. To perform this operation, select Data then Unstack Data from the Data Window menu.

Uniques (X)

Returns the ordered unique values of a column. This function cannot be used with any other function. **X** is the column from which the unique values are obtained.

Example: The following transformation was used to generate column **X2**.

Uniques(X1)

X1	X2
1	1
3	2
2	3
3	4
2	
4	
3	
4	
2	

UnSplice (N,I,X)

This function has been removed. To perform this operation, select Data then Unstack Data from the Data Window menu.

Recode Functions

Lookup (X1,X2,X3,Compare)

This function has been removed. Use the RECODE function instead along with the Recode Transformation Editor tool to easily input value lists using a spreadsheet input that is separate from the Data Table.

Recode (X; A=B, ... , Missing="NA", Else=0)

Recodes the values of **X**. It lets you change the values of one column according to rules you supply. It provides a type of If-Then transformation. Note that the recode statements (the phrases inside a pair of parentheses) are processed sequentially, so that the last true condition is the one used. This function cannot be used with any other function. **X** is a column to be recoded. It cannot be an expression. **A** represents the current values of **X** that are to be recoded. These values can be numeric or text. If the value is text, it must be enclosed in double-quotes. A range may be specified using the format **Smallest : Largest**. The colon is the "range" operator. All values between, and including, the two boundaries specified are in the range. If the first boundary is omitted, negative infinity is assumed. If the second boundary is omitted, positive infinity is assumed. The keyword **Else** (for otherwise) may be used to indicate what to enter when none of the conditions are met. The keyword **Missing** may be used to indicate a missing value. **B** is the new value to be assigned if the value of **X** is equal to, or falls in the range, designated as **A**. It may be a numeric value, a text value (between double-quotes), or a column. It cannot be an expression.

Example: **Recode(Q1; :0="Below", 1=5, 2:4=6, 5:="Above", Missing="NA", Else=Q2)**

Q1	R1
0	Below
1	5
5	Above
4	6
2	6
10	Above
	NA
-1	Below
3	6

Sequences and Series Functions

Sequence (Inc)

Returns a series beginning at **Inc** that adds **Inc** at each subsequent row. **Inc** is the increment added at each row to the amount of the last row.

Example: **Sequence(5)** yields **5,10,15,20,25,...**

Series (Length,Restart) or Ser (Length,Restart)

Returns a series beginning at 1 that adds 1 at each new row. **Length** is the number of rows before the series is increased by one. **Restart** is the number of rows before the series is restarted.

Example: **Series(2,6)** yields **1,1,2,2,3,3,1,1,2,2,3,3,...**

Statistical Functions

AbsDev (X)

Returns the absolute deviation from the mean of the data in X. The formula is NEW=|X-MEAN|.

Average (X1, X2, X3, X5:X8)

Returns the average of the list of numbers and columns. Note that a colon may be used to signify a contiguous set of columns. Hence, **X5:X8** means **X5, X6, X7, X8**.

Count (X1, X2, X5:X8)

Returns the number of non-missing items in the list.

Cum (X)

Returns the sum of the values of **X** up to and including the current row.

Dev (X)

Returns the deviation from the mean of the data in X. The formula is NEW=X-MEAN.

ExpNorScore (X)

Returns the expected value of the normal order statistic corresponding to **X**. This function cannot be used with any other function. **X** is a column. Ties are assigned the average of the tied expected normal scores.

LOESS (Y,X,PERCENT,ORDER,ITER)

Returns the smoothed Loess regression values based on **Y** and **X**. This function cannot be used with any other function. **Y** is the dependent variable. **X** is the independent variable. **PERCENT** is the percentage (1 to 100) of the data range that is used in each LOESS calculation (defaults to 40%). **ORDER** is the order of the approximating polynomial (1=Linear, 2=Quadratic, defaults to 1). **ITER** is the number of robust iterations to downplay the influence of outlying points (defaults to 0).

Example: **Loess(Y,X,30%,1,0)**

Max (X1, X2, X5:X8)

Returns the maximum of the items in the list.

Median (X1, X2, X3, X5:X8)

Returns the median of the list of numbers or columns. For an odd number of values, this function returns the middle value. For an even number of values, this function returns the average of the two middle values. Note that a colon may be used to signify a contiguous set of columns. Hence, **X5:X8** means **X5, X6, X7, X8**.

Min (X1, X2, X5:X8)

Returns the minimum of the items in the list.

MovingAverage (X,k) or Mav (X,k)

Returns the **k**-period moving average of a column. This function cannot be used with any other function. **k** is number of rows to be averaged. The current row is the last item in the average. **X** is a column.

Example: **MovingAverage(X,5)**

NormScore (X)

Returns the inverse-normal quantiles of **X**. These are standard-normal values, not probabilities. This function cannot be used with any other function. **X** is a column. The quantiles use rank/(n+1). Ties are assigned the average quantile. See also ExNorScore above.

Smooth (X)

Returns the 3RSSH-smooth of the values in **X**. This function cannot be used with any other function. **X** is a column to be smoothed. The 3RSSH is John Tukey's famous median smoother. It behaves much like a moving average operator that downplays the influence of large jumps in the series.

Standardize (X)

Returns a standardized version of the data in **X**. The formula is **New=(X-Mean)/Sigma**. This function cannot be used with any other function. **X** is a column to be standardized.

StdDev (X1, X2, X5:X8)

Returns the standard deviation of the list of numbers.

Example: **Stddev(X1:X20)** yields the standard deviation of columns **X1** through **X20**.

Sum (X1, X2, X5:X8)

Returns the sum of the list of numbers. Missing values are treated as zeros.

Text Functions

Text functions have a name and one or more arguments contained within parentheses. Inside an expression, a function is treated just like a number or a column. The following functions are designed to work specifically with text data.

Contains (X,Chars,Logic)

Returns a numeric value indicating the position of **Chars** in **X. X** is a value or column. **Chars** is the text to be searched for. **Logic** is 0 if case-sensitive, 1 if the case of the letters does not matter.

Example: **Contains("The box of oranges","BOX",1)** yields **5.**

Extract (X,First,Nchar)

Returns a value made by extracting from **X** the next **Nchar** characters, beginning at position **First**. **X** is text or a numeric value. **First** is the beginning position. Each character in **X** takes up one position. **Nchar** is the number of characters. If the end is reached before **Nchar** characters are obtained, the number of characters is reduced to the number available.

Example: **Extract("HINTZE",2,3)** yields **"INT"**

Join (X,Y)

Returns the text made by connecting the text in X (on the left) to that in Y (on the right). **X** is text. If it is a number, it is used as a text value. **Y** is text. If it is a number, it is used as a text value.

Example: **Join("LOG","CABIN")** yields **"LOGCABIN"**

Lcase (X)

Returns the lower case of the letters contained in **X**. Non-letters are unchanged. **X** is a column.

Example: **Lcase("CaT")** yields **"cat"**.

Left (X,Nchar)

Returns a value made of the first **Nchar** characters of **X**. **X** is text or a numeric value. **Nchar** is the number of characters. If the end is reached before **Nchar** characters are obtained, the number of characters is reduced to the number available.

Example: **Left("Howdy",3)** yields **"How"**.

Length (X)

Returns the number of characters (letters, numbers, spaces, etc.) in **X**. **X** is a column.

Example: **Length("Computer Program")** yields **16**.

Remove (X,Old)

Returns a value made by removing the characters in **Old** from **X**. **X** is a value or column. **Old** is the text (enclosed in double quotes) to be removed.

Example: **Remove("SMILES","MILE")** yields **"SS"**.

Repeat (X,N)

Returns a value made by repeating **X** a total of **N** times. **X** is a numeric value, a text value (enclosed in double quotes), or a column. **N** is the number of times **X** is repeated.

Example: **Repeat("AB",3)** yields **"ABABAB"**.

Replace (X,Old,New)

Returns a value made by replacing the characters in **OLD** with the characters in **NEW** in the column **X**. **X** is a column. **Old** is the text (enclosed in double quotes) to be replaced. **New** is the text (enclosed in double quotes) that will be inserted.

Example: **Replace("XOXOX","O","AA")** yields **"XAAXAAX"**.

Right (X,Nchar)

Returns a value made of the last **Nchar** characters of **X**. **X** is text or a numeric value. **Nchar** is the number of characters. If the beginning is reached before **Nchar** characters are obtained, the number of characters is reduced to the number available.

Example: **Right("Howdy",2)** yields **"dy"**.

Ucase (X)

Returns the upper case of the letters contained in **X**. Non-letters are unchanged. **X** is a column.

Example: **Ucase("cat")** yields **"CAT"**.

Time Series Functions

FirstDiff (X)

Returns the first-order differencing of a column (D1[i] = X[i] - X[i-1]). Cannot be combined with any other function. **X** is a column.

Lag (X,k)

Returns the **k**-period lag of a column. This function cannot be used with any other function. **k** is number of rows to lag. **X** is a column.

Example: **Lag(X,2)**

Lead (X,k) or Led (X,k)

Returns the **k**-period lead of a column. This function cannot be used with any other function. **k** is number of rows to lead. **X** is a column.

Example: **Lead(X,3)**

SecondDiff (X)

Returns the second-order differencing of a column (D2[i] = X[i] - 2X[i-1] + X[i-2]). Cannot be combined with any other function. **X** is a column.

Trigonometric Functions

ArCosh (X)

Returns the inverse hyperbolic cosine of **X**. **X** is a number >= 1.

ArSin (X)

Returns the arc sine (the angle in radians whose sine is **X**) of **X**. Note that **X** is a number between **-1** and **1**.

ArSinh (X)

Returns the inverse hyperbolic sine of **X**. **X** is a number.

ArTan (X)

Returns the arc tangent (the angle in radians whose tangent is **X**) of **X**.

ArTanh (X)

Returns the inverse hyperbolic tangent of **X**. **X** is a number between -1 and 1.

Cos (X)

Returns the cosine of the angle **X** (in radians).

Cosh (X)

Returns the hyperbolic cosine of **X**. **X** is a number whose absolute value is less than 705.

Sin (X)

Returns the sine of the angle **X**. **X** is an angle in radians.

Sinh (X)

Returns the hyperbolic sine of **X**. **X** is a number whose absolute value is less than 705.

Tan (X)

Returns the tangent of the angle **X**. **X** is an angle in radians.

Tanh (X)

Returns the hyperbolic tangent of **X**. **X** is a number whose absolute value is less than 705.

Indicator Variables

The *Recode* function will usually suffice for recoding data. However, there are times when a richer set of transformations is needed. The techniques described here will let you combine logic statements in any way you like to form very complex, logical transformations.

An indicator function is one if a condition is true and zero if the condition is false. Indicator variables are used a great deal in regression analysis. The following two examples would give identical results (note which one is simpler):

(X<4)

Recode(X; (:4=1)(Else=0))

This technique is based on the logic operators which were discussed above. To review, the statement **(X<4)** results in a "0" if the statement is false and a "1" if the statement is true. Note that the less-than symbol may be replaced with any of the other logic symbols. The following table presents a few examples of the types of expressions that may be generated.

Transformation	X=0	X=1	X=2
(X<1)	1	0	0
(X<=1)	1	1	0
(X<1)*4	4	0	0
(X<1)+(X<2)	2	1	0
(X=1)	0	1	0
(X=1)*2+(X=2)*4	0	2	4

The next concept that you need to understand is that the logical "OR" is represented by adding two statements and the logical "AND" is formed by multiplying two statements. Hence, the statement

If X is equal to two or six, recode it to four

would be represented by the expression

4*[(X=2)+(X=6)].

Likewise, the statement

if X is less than four and Y is greater than 2, set the result equal to 2

might be written

2*[(X<4) * (Y>2)].

Now, building upon these few simple concepts you can build any logic operator you want.

Conditional (If-Then) Transformations

Conditional (If-Then) Transformations allow you to selectively apply transformations based on logical statements involving one or more data columns. The form of a conditional transformation is

IfThen([*Condition 1*], [*Result 1*]; [*Condition 2*], [*Result 2*]; ... ; Else, [*Result Else*])

where [*Condition i*] represents condition statements and [*Result i*] represents the transformation formula to apply when the corresponding condition evaluates to True. The word "Else" is used to designate the transformation formula to use when all the other conditions evaluate to False. An "Else" statement is not required; if it is omitted, then the result column will be left unchanged for all rows that do not meet any of the conditions.

Examples:

IfThen(C3=1, C1+C2; Else, 0)
IfThen(C2 > 35, C1+C2; C2 > 10, C3+C4)
IfThen((C5 > 1) AND (C5 <= 10), C1+C2; Else, Missing)

Condition Statements

Enter logical condition statements using the syntax of a regular filter. If you want apply a transformation to all rows, leave the condition blank.

Syntax for [Condition i]

The condition statements can include lists, ranges, and/or functions. The basic syntax of a condition statement is

[*COLUMN*] [*LOGIC OPERATOR*] [*VALUE or FUNCTION*]

With 6 possible logic operators:

= Equal to
<> Not equal to (≠)
< Less than
<= Less than or equal to (≤)
> Greater than
>= Greater than or equal to (≥)

Rules:

1. Combine several statements using AND or OR and parentheses (e.g., (C1=4) OR (C2<7)).
2. Text values must be enclosed in double quotes (e.g., "Male").
3. Use the word MISSING to represent a missing value (a blank).
4. Use the word ELSE to designate the result when all the other conditions evaluate to false.
5. You cannot use functions that require more than one row of data such as LAG, LED, etc.
6. You cannot use special functions such as RECODE, LOCATE, etc.

Examples:

C1 = 0
C2 > 35
C3 > log(C2)
C4 <= C5 + C6
(C5 > 1) AND (C5 <= 10)
((C7 > 5) AND (C8 <= 30)) OR (C9 = "High")
Else

Result Functions

Enter a value or function to which the result column will be set if the corresponding conditional statement evaluates to true. If you want to apply this transformation to all rows, leave the corresponding IF statement blank. There is no need to type in the name of the result column, just the result value or function.

Syntax for [Result i]

Rules:

1. Enter an expression using the syntax rules of a transformation.
2. Text values must be enclosed in double quotes (e.g., "Male").
3. Use the word MISSING to represent a missing value (a blank).
4. You cannot use functions that require more than one row of data such as LAG, LED, etc.
5. You cannot use special functions such as RECODE, LOCATE, etc.
6. You cannot enter a list of values.

Examples:

5
"Group 1"
log10(C1)
C5 + C6
C7^2
ln(C3+C4)

Non-English Language Setting

If you are using a computer language setting other than English that uses a comma as the decimal symbol, then each condition/result pair should be separated by the computer's specified list separator (this is usually a semicolon if not a comma) and groups should be separated with a vertical bar "|". The syntax for a computer that uses a comma as the decimal symbol would be

IfThen([*Condition 1*]; [*Result 1*] | [*Condition 2*]; [*Result 2*] | ... | Else; [*Result Else*])

Recode Transformations

Recode transformations allow you to convert values (e.g., 1= “Male”, 2= “Female”) and store the results in another column. The form of a recode transformation is

Recode([*Column*]; [*Value 1*] = [*Recode Value 1*], [*Value 2*] = [*Recode Value 2*], ...)

where [*Column*] represents the column to recode, [*Value i*] represents a value or range of values in the column to recode, and [*Recode Value i*] represents the result to which the value(s) in the column to recode will be changed.

Examples:

Recode(C1; 1="Male", 2="Female")
Recode(C2; :5="Low", 6:10="Medium", Else="High")

Values

Enter logical condition statements using the syntax of a regular filter. If you want to apply a transformation to all rows, leave the condition blank.

Syntax for [Value i]

Rules:

1. Enter a single numeric, text, or date value.
2. Text values must be enclosed in double quotes.
3. Enter a one-sided range of values formatted as :UPPER or LOWER:.
4. Enter a two-sided range of values formatted as SMALLEST:LARGEST.
5. Use the word ELSE for otherwise.
6. Use the word MISSING to represent a missing value (a blank).

Examples:

0
5.889
"High"
MISSING
ELSE
:5 (values less than or equal to 5)
3: (values greater than or equal to 3)
6:10 (values between 6 and 10, including 6 and 10)

Recode Values

Enter a value or function to which the result column will be set if the corresponding conditional statement evaluates to true. If you want to apply this transformation to all rows, leave the corresponding IF statement blank. There is no need to type in the name of the result column, just the result value or function.

Syntax for [Recode Value i]

Rules:

1. Enter a single numeric, text, or date value. Alternatively, enter a column name.
2. Text values must be enclosed in double quotes.
3. Use the word MISSING to represent a missing value (a blank).

Examples:

0
"Low"
C1
Missing

Non-English Language Setting

If you are using a computer language setting other than English that uses a comma as the decimal symbol, then each value pair should be separated by the computer's specified list separator (this is usually a semicolon if not a comma) and the column and value pairs should be separated with a vertical bar "|". The syntax for a computer that uses a comma as the decimal symbol would be

Recode([*Column*] | [*Value 1*] = [*Recode Value 1*]; [*Value 2*] = [*Recode Value 2*], ...)