Aldo Benini

Merging two datasets on approximate values
Matching on groups as well as on the nearest value of a numeric variable, in MS Excel and in STATA

Summary

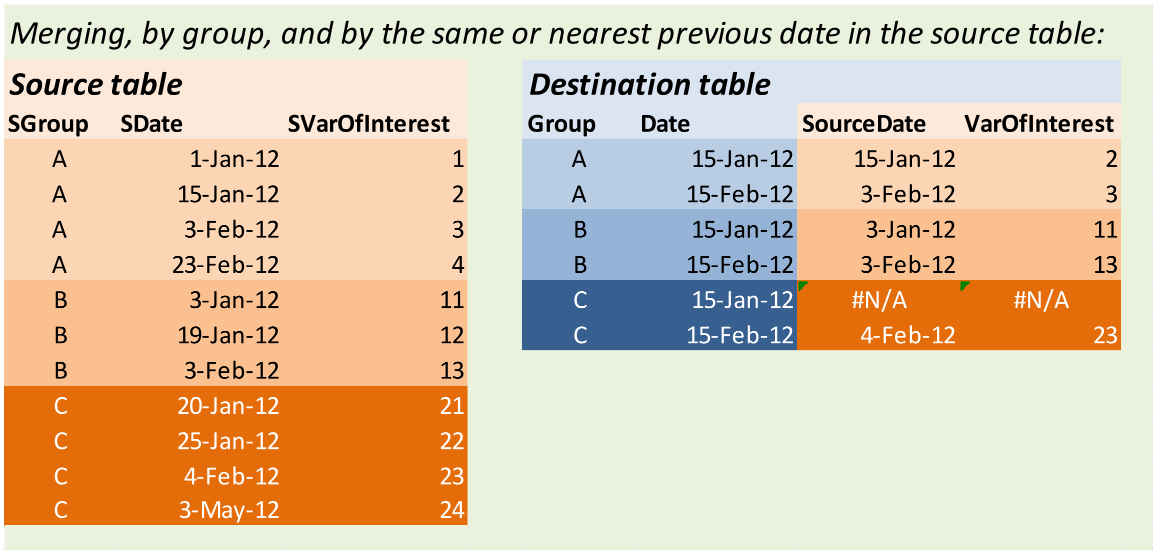
In data management, sets of information may have to be linked for which the common link variables agree only partially. Benini (2008) presented solutions, in Excel as well as STATA, for table merging when the link variables are text-based with spelling differences.

Approximate matches

Some or all of the link variables may be numeric. The agreement in one of the numeric variables may be approximate only. If there is only one lookup variable and a meaningful match can be made using the value in the source table that is identical with, or the next lower to, the lookup value in the destination table, the problem is easily solved. The Excel function VLOOKUP, with the fourth argument [range_value] = TRUE, produces this particular approximate match.

More often, the situation is more complex. The desired linking is by group as well as by approximate numeric variable. The values of the numeric link variable in the source table may or may not be unique.

Table 1: Typical approximate table merge constellation



Merging, by group, and by the same or nearest previous date in the source table:

Source table

SGroup	SDate	SVarOfInterest
A	1-Jan-12	1
A	15-Jan-12	2
A	3-Feb-12	3
A	23-Feb-12	4
B	3-Jan-12	11
B	19-Jan-12	12
B	3-Feb-12	13
C	20-Jan-12	21
C	25-Jan-12	22
C	4-Feb-12	23
C	3-May-12	24

Destination table

Group	Date	SourceDate	VarOfInterest
A	15-Jan-12	15-Jan-12	2
A	15-Feb-12	3-Feb-12	3
B	15-Jan-12	3-Jan-12	11
B	15-Feb-12	3-Feb-12	13
C	15-Jan-12	#N/A	#N/A
C	15-Feb-12	4-Feb-12	23

Demonstration by example

To illustrate, we use an example in which the approximate link variable is a date variable. The researcher wishes to import into the destination table the values of a "variable of interest" from the source table, such that groups match exactly, and the date in the source table selected is the one that approximates the destination date, within the given group, most closely, from the past (i.e., is identical or the closest previous date). The situation arises when the measurement dates for source and destination variables are not identical, yet matching records by closest prior source record date promises valid results. Table 1 provides an example. Because date variables are a subclass of numeric variables, this example does not cause loss of generality.

We demonstrate solutions in Excel as well as in STATA. The Excel solution makes use of the functions IF, INDEX and MATCH. A more flexible solution, although one that needs extra precautions, is offered in STATA. It was implemented as the procedure nearmrg (for "nearest match merging") by Blasnik and Smith (Undated). This note discusses both solutions. For Excel, we provide special formulas.

The appendix gives the Help function texts for the Excel functions as well as for the STATA procedure. A zip file containing an Excel demo workbook and the STATA demo source and destination datasets can be downloaded.

Is approximate matching appropriate?

Approximate matching may not always be the best solution in this kind of situation. If the change that the variable of interest in the source table likely underwent between its measurement and the times when the values of the closest records in the destination table were taken is theoretically non-negligible, a different approach may produce more valid results. It may be necessary to resort to interpolation.

Suggested citation:

Benini, A. (2012). Merging two datasets on approximate values. Matching on groups as well as on the nearest value of a numeric variable, in MS Excel and in STATA. Washington DC.

Contents

Acknowledgement

This paper relies heavily the procedure STATA *nearmrg* written by Blasnik and Smith (Blasnik and Smith Undated). Their coordinates are given within the Help text the very end of the appendix.

Introduction

Motivation

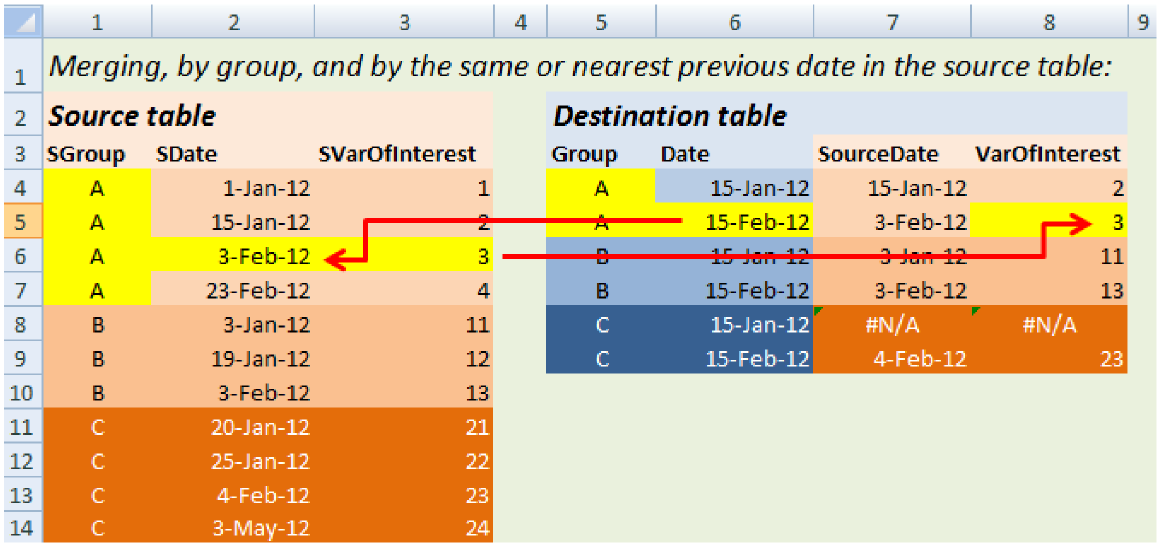
In analyzing time to delinquency in a microfinance program in Bangladesh (Benini, von Bünau et al. 2011), a difficult data linking problem arose. We needed to import into a table of quarter-yearly loan episodes, with 3.4 million records, variables of interest (overdue rates) that had been calculated for collectivities - all the loans under a group of borrowers, a frontline worker, and a bank branch. The group identifiers needed to link the tables (borrower group ID, credit organizer ID, resp. bank branch ID) were well defined and fully matchable. However, the end dates of the loan episodes and of the borrower group, etc. reporting periods matched only approximately.

STATA provided a solution; the procedure used will be presented further below. However, the problem is of a more general interest. It arises whenever the desired linking is by group as well as by approximate numeric variable. We wish to make generic solutions available in Excel as well as in STATA.

Didactic example

Both in the microfinance analysis mentioned above and in this example, the approximate match variable is a date variable. Since both Excel and STATA (and, presumably, other statistical applications) handle this type of variable internally as a numeric variable, we develop the logic without loss of generality. The example as shown in Table 2 demonstrates the lookup in one destination-table record. Arrows point to the source table and back to the destination table.

Table 2: Example of a lookup operation



	1	2	3	4	5	6	7	8	9
1	*Merging, by group, and by the same or nearest previous date in the source table:*								
2	***Source table***				***Destination table***				
3	**SGroup**	**SDate**	**SVarOfInterest**		**Group**	**Date**	**SourceDate**	**VarOfInterest**	
4	A	1-Jan-12	1		A	15-Jan-12	15-Jan-12	2	
5	A	15-Jan-12	2		A	15-Feb-12	3-Feb-12	3	
6	A	3-Feb-12	3		B	15-Jan-12	3-Jan-12	11	
7	A	23-Feb-12	4		B	15-Feb-12	3-Feb-12	13	
8	B	3-Jan-12	11		C	15-Jan-12	#N/A	#N/A	
9	B	19-Jan-12	12		C	15-Feb-12	4-Feb-12	23	
10	B	3-Feb-12	13						
11	C	20-Jan-12	21						
12	C	25-Jan-12	22						
13	C	4-Feb-12	23						
14	C	3-May-12	24						

In the example, the challenge is to find, for Group A and on 15 February 2012 in the destination table, the value of the variable of interest in the source table. Searching within Group-A records of the source table, we establish that the closest previous date, counting backward from 15

February, is 3 February 2012. The value of interest in the source table is 3, which is then transferred to the destination table. For easier understanding (and audit purposes!), the date of the merged source table record is also noted in the destination table.

The reader may himself try to find matches for other records in the destination table. Note that for Group C and 15 January 2012, no match was found. This is so because within Group C the source table holds no records with any date equal to, or earlier than, 15 January.

Sorting the source table; different approximation modes

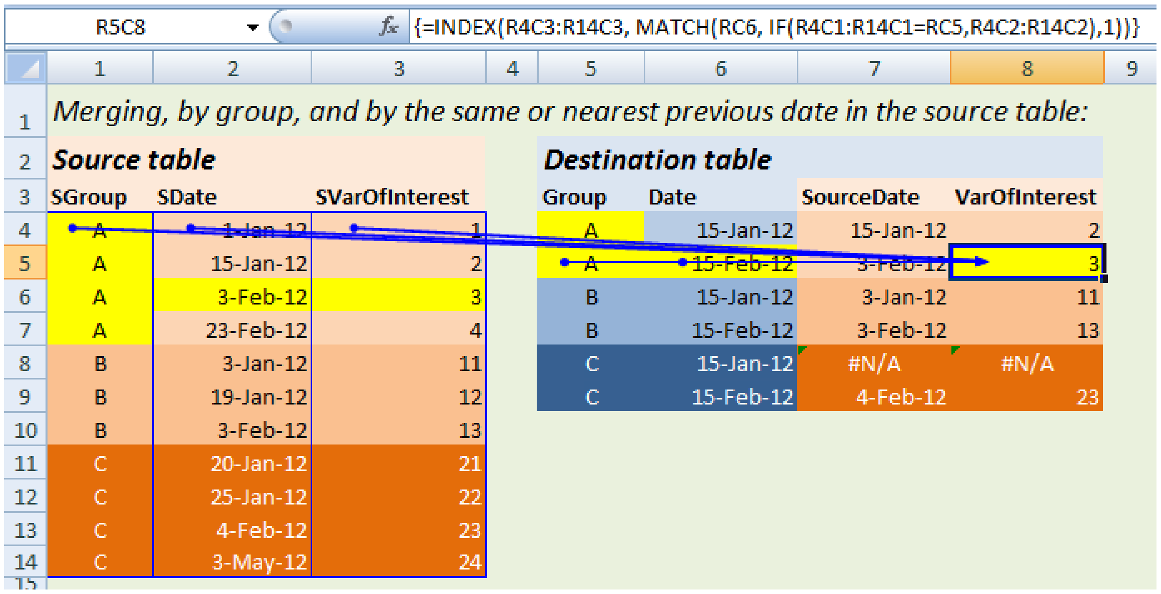
Note also that the source table has been sorted by group, then date. This is required. Note further that in this case the approximate value is defined as the first source table date equal and prior to the destination table date in point. Other options may make sense in other context: first equal or greater, or smallest absolute difference.

Solutions for Excel users

Excel users may initially think of VLOOKUP. This very serviceable workhorse function does do approximate lookups when its fourth argument is set to TRUE (exact matches work with FALSE). But it does not admit multiple link variables such as are needed when the merging is to proceed by group[1].

However, the problem can be tackled with a combination of the functions INDEX, MATCH and IF. This figure shows the dependencies of the lookup formula used in cell R5C8.

Figure 1: Destination cell with precedent ranges, and formula



R5C8 {=INDEX(R4C3:R14C3, MATCH(RC6, IF(R4C1:R14C1=RC5,R4C2:R14C2),1))}

	1	2	3	4	5	6	7	8	9
1	*Merging, by group, and by the same or nearest previous date in the source table:*								
2	***Source table***				***Destination table***				
3	SGroup	SDate	SVarOfInterest		Group	Date	SourceDate	VarOfInterest	
4	A	1-Jan-12	1		A	15-Jan-12	15-Jan-12	2	
5	A	15-Jan-12	2		A	15-Feb-12	3-Feb-12	3	
6	A	3-Feb-12	3		B	15-Jan-12	3-Jan-12	11	
7	A	23-Feb-12	4		B	15-Feb-12	3-Feb-12	13	
8	B	3-Jan-12	11		C	15-Jan-12	#N/A	#N/A	
9	B	19-Jan-12	12		C	15-Feb-12	4-Feb-12	23	
10	B	3-Feb-12	13						
11	C	20-Jan-12	21						
12	C	25-Jan-12	22						
13	C	4-Feb-12	23						
14	C	3-May-12	24						

[1] There are multiple elaborations of "extended VLOOKUP" available on the Internet, but they all seem to work for *exact* matches with multiple lookup variables.

Two situations

This solution works in two situations:

1. The values of the approximately-to-be-matched variable in the source table are, *within each group*, *unique* (This is the case in our example. Note they are not unique across groups - 3 February 2012 occurs in group A and B.
2. Rarely, we may find that t he values within all or some of the groups are *not unique*, but the combinations, in the source table records concerned, of those values and of the values of the variables of substantive interest are identical (these records may differ in other variables that are not of interest for the purpose of the current merge).

Using absolute cell reference mode

[Implemented in the sheet "AbsoluteCellRef" in the demo workbook]

We return to our example table, which is of the first type (unique values within groups). As expected, the formula accesses the three ranges that store the source table information (each with a blue border) and the group and date information in its own destination table record. The exact formula

=INDEX(R4C3:R14C3, MATCH(RC6, IF(R4C1:R14C1=RC5,R4C2:R14C2),1))

is surrounded, in the Excel formula bar, by curly brackets. These result from entering the formula as an array formula, by pressing Ctrl + Shift + Enter simultaneously. See instructions on entering array formulas in a box below; for general interest, consult the Help function.

Note the use of absolute and mixed references, e.g., *R4C3:R14C3* (absolute mode) to address the range holding the variable of interest in the source table, or *RC6* (mixed) to refer to the date in the destination table. This usage of references makes the formulas copiable[2].

Important: Enter array formulas correctly

1. Select **only one** cell, e.g. *R4C7*
2. Write the formula there, e.g. *=INDEX(R4C2:R14C2, MATCH(RC6, IF(R4C1:R14C1=RC5,R4C2:R14C2),1))*
3. Enter as array formula, by pressing Ctrl+Shift+Enter. Curly brackets will appear in the formula bar: *{=INDEX(R4C2:R14C2, MATCH(RC6, IF(R4C1:R14C1=RC5,R4C2:R14C2),1))}*

4. Copy this formula and paste it into the concerned range, e.g. to *R5C7:R9C7*.
[Copy - paste special (formula only) if you wish to preserve cell formats]

DO NOT select the entire range to write the formula and then enter it as array formula.
[as we would do with multi-cell output functions such as FREQUENCY]

[2] We use R1C1 notation. The same formula, in A1-notation, would be *=INDEX($C$4:$C$14, MATCH($F5, IF($A$4:$A$14=$E5,$B$4:$B$14),1))*, with the same usage of absolute and mixed references.

Using named ranges

[Implemented in the sheet "ExampleTable" in the demo workbook]

To explain how the formula works, we will name the ranges in the source table as in

Name	**Refers to**
SGroup	=ExampleTable!R4C1:R14C1
SDate	=ExampleTable!R4C2:R14C2
SVarOfInterest	=ExampleTable!R4C3:R14C3

assuming that our Excel table is called ExampleTable. Naming the ranges makes it much easier to follow the next few steps.

The formula then can be written as

=INDEX(SVarOfInterest, MATCH(RC6, IF(SGroup=RC5,SDate),1))

Working from inside out we color its parts as

=INDEX(SVarOfInterest, MATCH(RC6, IF(SGroup=RC5,SDate),1))

to mean that

1. If the lookup value in RC5 is found in the range SGroup, IF returns a range of the same dimensions as SDate (n x 1), with cell values equal to the source dates if the group tallies, and with "FALSE" else. The dates within this range are unique (because they are for one given group). If the lookup value is not found, IF returns nothing.
2. MATCH establishes the relative position of the lookup value RC6 (the date in the destination table) in the range returned by IF.
3. INDEX returns from the range of interest (SVarOfInterest) the value of the cell that is in the relative position returned by MATCH (in other words, MATCH returns the relative row number of the cell for the range of interest). The "1" at the right side of the formula tells INDEX that it has to look it up in the first column of the range. This is trivial here (because SVarOfInterest is a one-column range), but not always.

Using the same formula to merge multiple variables of interest

[Implemented in the sheet "LookUpMultipleVars" in the demo workbook]

We name the entire source table range as STable.

The same formula *=INDEX(STable,MATCH(RC7, IF(SGroup=RC6,SDate)),COLUMN(RC)-6)* is used for all variables to merge. It has two new elements, both relating to the function INDEX:

=INDEX(STable,MATCH(RC7, IF(SGroup=RC6,SDate)),COLUMN(RC)-6)

INDEX now returns a value from the entire source table, not from a single-column range that holds one variable. As before, the relative row number is from the MATCH result.

The relative column number is calculated as COLUMN(RC) + x, where COLUMN(RC) is the column number (in the spreadsheet!) of the destination cell, and x is an adjustment that the user calculates such that COLUMN(RC) + x becomes the desired column number in the named source table range.

In our example, x = -6 because the column 8 (Source date in the destination table) needs values from the 8 - 6 = 2nd column of the source table range, column 9 (VarOfInterest1 in the destination table needs values from the 9 - 6 = 3rd column of the source table range, etc.

To prevent confusion: In the example, the first column of the source table happens to be the first column of the worksheet, but this is irrelevant for the syntax of =INDEX(STable, ..). The logic of passing the desired relative column number via COLUMN(RC) + x holds also when the source table is kept in a different worksheet. In fact, with large datasets, it is advisable to keep source and destination tables in two different sheets.

When several variables define groups

In our example, the groups are defined by one variable only. If there are several group-defining variables necessary - e.g. "District" and "Village" (because village names may recur across districts), two approaches seem feasible:

1. Concatenation of the defining variables in a new variable each in the source and destination tables; using the concatenated result for the exact; see Help for the function CONCATENATE, or
2. Combining the defining variables in the IF clause with the help of the function AND, on the lines of

    = INDEX(STable,MATCH(RCx, IF(AND(SDistrict =RCy, SVillage = RCz), SApprox)), COLUMN(RC) + adjust)

where SDistrict is the named range for the district variable, SVillage for the village variable, in the source table, and RCx, RCy, etc. are the relevant lookup values in the destination table. SApprox is the numeric variable in the source table on which an approximate match is made.

Approximate merging in STATA

The user-written procedure *nearmrg* (Blasnik and Smith Undated) accomplishes approximate merges by group.

nearmrg is more flexible, and its setup much faster, than the demonstrated Excel solutions. Notably *nearmrg* admits multiple group-defining variables without additional operations. It also offers four options of how to define the approximation; see explanations in the appendix.

However, some special steps are necessary. Both the source and the destination table have to be sorted on group identifiers and on the numeric variable for the approximate match, and the variables have to have the same names in exactly both tables. Thus,in both tables, sort by, e.g.:

gsort DistrictID VillageID Date

The *nearmrg* Help text says that the numeric variable must have unique values only in the source table, but we have found that it is enough if they are unique *within* groups. The presence of duplicates within any group aborts the command, but we will show a work-around below.

Terminology

In *nearmrg*, the destination table is called the "master dataset", the source table is called the "using dataset", in keeping with the lingo of the STATA procedure *merge*, which is used for exact merges. In STATA, the master and using datasets are held in different files. In both files, the group-defining and approximate-match variables need to have the same names.

Example

We use the same example as for the Excel section, with this **MasterData** set:

Group	**Date**
A	15jan2012
A	15feb2012
B	15jan2012
B	15feb2012
C	15jan2012
C	15feb2012

and this **UsingData** set:

Group	**Date**	**SVarOfInterest1**	**SVarOfInterest2**
A	01jan2012	1	201
A	15jan2012	2	202
A	03feb2012	3	203
A	23feb2012	4	204
B	03jan2012	11	211
B	19jan2012	12	212
B	03feb2012	13	213
C	20jan2012	21	221
C	25jan2012	22	222
C	04feb2012	23	223
C	03may2012	24	224

Syntax

For this example, with MasterData opened, we fill in the generic syntax *nearmrg [varlist] using [using dataset], nearvar(varname) [genmatch(newvarname) lower upper roundup keep mergeoptions] [see appendix for explanations]*

as

nearmrg Group using UsingData, nearvar(Date) genmatch(SourceDate) lower

and obtain the result

Group	**Date**	**SVarOfInterest1**	**SVarOfInterest2**	**_merge**	**SourceDate**
A	15jan2012	2	202	3	15jan2012
A	15feb2012	3	203	3	03feb2012
B	15jan2012	11	211	3	03jan2012
B	15feb2012	13	213	3	03feb2012
C	15feb2012	23	223	3	04feb2012
C	15jan2012			1	

The result is the same as in Excel, except for the sort by _merge, the merge status variable. The unmatched record was placed at the bottom.

When duplicates are present in the using dataset

When duplicates of the approximate matching variable are present within groups, as in this fictitious example from a using dataset:

GroupID	**Date**	**VarOfInterest**	**AuxilTag**
1	30 Apr 03	.000043	1
1	30 Apr 03	.000038	2

nearmrg, without additional precautions, will abort. There are several treatments available, depending on analytical interest.

1. The duplicates can simply be deleted, with a command such as

duplicates drop GroupID Date, force

However, the researcher may not want to reduce the using dataset. The reasons may differ. He may await further calculated variables later in the process that should inform the selection within duplicates of the records to be used in the merging. Or, the records that are duplicates regarding the matching variables may differ on the existing variables of interest. A rule for selection may already be known and substantively motivated (take the smaller, larger, first, last value, etc.).

2. In such cases, an auxiliary tag may be created in both master and using dataset, to be used as an additional group variable. Let us assume that the match is desired for the tag value 1. Then, e.g.,

In the master dataset:

```
gen AuxilTag = 1

gsort GroupID AuxilTag Date
```

In the using dataset:

[for example, for first duplicate to be selected:]

```
gsort GroupID Date

by GroupID Date: gen AuxilTag = _n

gsort GroupID AuxilTag Date
```

and back to the master dataset, where *nearmrg* is now run through this command:

```
nearmrg GroupID AuxilTag using UsingData, nearvar(Date) genmatch(NearDate) lower
```

[or options other than "lower", as desired for the approximation rule; see *nearmrg* Help in the appendix]

AuxilTag ensures that Date, as the approximate value, is unique within each combination of GroupID and AuxilTag in the using dataset. AuxilTag = 1 for all records of the master dataset ensures that only using the first among duplicates from the using dataset will be merged in.

If nearvar() is a string variable

The Help text indicates that *nearmrg* admits approximately to be matched-on variables may be in string format. We have not investigated this case nor seen any examples implementing it. Presumably, this facility allows matching on neighbors in alphanumerically sorted sets, in the sense that AA in the master dataset would be more closely matched by AB and by AC from the using dataset[3].

Final thoughts

The situations described in this paper are not very common in data management, but they do occasionally occur and pose challenges. For example, they are to be expected in operations for multi-level models when the measurements at different levels were taken at different times, and a linear or other simple transformation rule for aligning the measurement times does not exist.

[3] Similarly, in Excel, VLOOKUP with range_lookup = TRUE is not restricted to numeric values in the lookup variable as long as the first column of the lookup table is sorted ascendingly.

We presented solutions for Excel and for STATA users. The STATA solution was already given in the shape of the user-written procedure *nearmrg*; our expressed precautions regarding duplicates are but a minor footnote added. The Excel solution was easy to derive from a voluminous advisory literature on the combined used of the functions INDEX, MATCH and IF. It should be kept in mind, however, that the first two, plus the use of absolute and mixed cell references and of array formulas must not be taken for granted among the majority of basic and mid-level Excel users.

We offer the Excel solution, although more difficult to learn and less versatile than STATA's *nearmrg*, because the Excel user community is massively larger than STATA's, and in developing nation monitoring and research units, members of the latter species are few and far between.

Is an approximate match always the best solution?

As a last thought, we encourage researchers to consider whether an approximate match is really the best solution at hand, given the substantive interest. The variable of interest in the using dataset likely underwent some change between its measurement and the times when the nearvar() in the master dataset was measured. If that change is theoretically non-negligible, a different approach may produce more valid results. It may be necessary to resort to interpolation, within the using dataset, to the points in time used in the master dataset. This would be followed by an *exact* match on these, importing the interpolated values into the master dataset.

The necessary procedures cannot be fully elaborated here. In Excel, a macro should not be too difficult to devise. In STATA, one might try to *contract* the master dataset by GroupID and Date, then *append* the resulting table to the using set, sort and interpolate (*ipolate* with by).

However daunting that may seem, here in this paper the intent was strictly to make workable solutions available for table linkage using approximate matches.

References

Benini, A. (2008)."Efficient linking of lists in humanitarian data management." Washington DC. from http://aldo-benini.org/Level2/humanitarian_data_analysis.htm.

Benini, A., P. von Bünau, et al. (2011). RDRS and the Poor: Microfinance as Partnership. Twenty Years of Microfinance in RDRS Bangladesh. Dhaka, RDRS Bangladesh and North Bengal Institute.

Blasnik, M. and K. Smith (Undated). nearmrg - Nearest match merging of datasets [STATA ado and help file], M Blasnik & Associates and Clinical Epidemiology and Biostatistics Unit, Murdoch Childrens Research Institute

26 March 2012

Appendix

Excel Help texts on the functions used

The Help function examples are not included here.

IF

This article describes the formula syntax and usage of the **IF** function in Microsoft Office Excel.

Description

The **IF** function returns one value if a condition you specify evaluates to TRUE, and another value if that condition evaluates to FALSE. For example, the formula **=IF(A1>10,"Over 10","10 or less")** returns "Over 10" if A1 is greater than 10, and "10 or less" if A1 is less than or equal to 10.

Syntax

```
IF(logical_test, value_if_true, [value_if_false])
```

The IF function syntax has the following arguments:

- **logical_test** Required. Any value or expression that can be evaluated to TRUE or FALSE. For example, A10=100 is a logical expression; if the value in cell A10 is equal to 100, the expression evaluates to TRUE. Otherwise, the expression evaluates to FALSE. This argument can use any comparison calculation operator.
- **value_if_true** Required. The value that you want to be returned if the ***logical_test*** argument evaluates to TRUE. For example, if the value of this argument is the text string "Within budget" and the ***logical_test*** argument evaluates to TRUE, the **IF** function returns the text "Within budget." If ***logical_test*** evaluates to TRUE and the ***value_if_true*** argument is omitted (that is, there is only a comma following the ***logical_test*** argument), the **IF** function returns 0 (zero). To display the word TRUE, use the logical value TRUE for the ***value_if_true*** argument.
- **value_if_false** Optional. The value that you want to be returned if the ***logical_test*** argument evaluates to FALSE. For example, if the value of this argument is the text string "Over budget" and the ***logical_test*** argument evaluates to FALSE, the **IF** function returns the text "Over budget." If ***logical_test*** evaluates to FALSE and the ***value_if_false*** argument is omitted, (that is, there is no comma following the ***value_if_true*** argument), the **IF** function returns the logical value FALSE. If ***logical_test*** evaluates to FALSE and the value of the ***value_if_false*** argument is omitted (that is, in the **IF** function, there is no comma following the ***value_if_true*** argument), the **IF** function returns the value 0 (zero).

Remarks

- Up to 64 **IF** functions can be nested as ***value_if_true*** and ***value_if_false*** arguments to construct more elaborate tests. (See Example 3 for a sample of nested **IF** functions.) Alternatively, to test many conditions, consider using the LOOKUP, VLOOKUP, HLOOKUP, or CHOOSE functions. (See Example 4 for a sample of the **LOOKUP** function.)

- If any of the arguments to **IF** are arrays, every element of the array is evaluated when the **IF** statement is carried out.
- Excel provides additional functions that can be used to analyze your data based on a condition. For example, to count the number of occurrences of a string of text or a number within a range of cells, use the COUNTIF or the COUNTIFS worksheet functions. To calculate a sum based on a string of text or a number within a range, use the SUMIF or the SUMIFS worksheet functions.

MATCH

Description

The **MATCH** function searches for a specified item in a range of cells, and then returns the relative position of that item in the range. For example, if the range A1:A3 contains the values 5, 25, and 38, then the formula

=MATCH(25,A1:A3,0)

returns the number 2, because 25 is the second item in the range.

Use **MATCH** instead of one of the **LOOKUP** functions when you need the position of an item in a range instead of the item itself. For example, you might use the **MATCH** function to provide a value for the ***row_num*** argument of the **INDEX** function.

Syntax

```
MATCH(lookup_value, lookup_array, [match_type])
```

The **MATCH** function syntax has the following arguments:

- **lookup_value** Required. The value that you want to match in ***lookup_array***. For example, when you look up someone's number in a telephone book, you are using the person's name as the lookup value, but the telephone number is the value you want.

  The ***lookup_value*** argument can be a value (number, text, or logical value) or a cell reference to a number, text, or logical value.

- **lookup_array** Required. The range of cells being searched.
- **match_type** Optional. The number -1, 0, or 1. The ***match_type*** argument specifies how Excel matches ***lookup_value*** with values in ***lookup_array***. The default value for this argument is 1.

  The following table describes how the function finds values based on the setting of the ***match_type*** argument.

Match_type	Behavior

1 or omitted	**MATCH** finds the largest value that is less than or equal to ***lookup_value***. The values in the ***lookup_array*** argument must be placed in ascending order, for example: ...-2, -1, 0, 1, 2, ..., A-Z, FALSE, TRUE.
0	**MATCH** finds the first value that is exactly equal to ***lookup_value***. The values in the ***lookup_array*** argument can be in any order.
-1	**MATCH** finds the smallest value that is greater than or equal to ***lookup_value***. The values in the ***lookup_array*** argument must be placed in descending order, for example: TRUE, FALSE, Z-A, ...2, 1, 0, -1, -2, ..., and so on.

NOTES

- **MATCH** returns the position of the matched value within ***lookup_array***, not the value itself. For example, **MATCH("b",{"a","b","c"},0)** returns 2, which is the relative position of "b" within the array {"a","b","c"}.
- **MATCH** does not distinguish between uppercase and lowercase letters when matching text values.
- If **MATCH** is unsuccessful in finding a match, it returns the #N/A error value.
- If ***match_type*** is 0 and ***lookup_value*** is a text string, you can use the wildcard characters — the question mark (**?**) and asterisk (*****) — in the ***lookup_value*** argument. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (**~**) before the character.

INDEX

INDEX comes in two forms, the array form and the reference form. The formulas in this note use the array form. The reference form Help text is not copied here.

Array form

Returns the value of an element in a table or an array, selected by the row and column number indexes.

Use the array form if the first argument to INDEX is an array constant.

INDEX(**array**,row_num,column_num)

Array is a range of cells or an array constant.

- If array contains only one row or column, the corresponding row_num or column_num argument is optional.
- If array has more than one row and more than one column, and only row_num or column_num is used, INDEX returns an array of the entire row or column in array.

Row_num selects the row in array from which to return a value. If row_num is omitted, column_num is required.

Column_num selects the column in array from which to return a value. If column_num is omitted, row_num is required.

Remarks

- If both the row_num and column_num arguments are used, INDEX returns the value in the cell at the intersection of row_num and column_num.
- If you set row_num or column_num to 0 (zero), INDEX returns the array of values for the entire column or row, respectively. To use values returned as an array, enter the INDEX function as an array formula in a horizontal range of cells for a row, and in a vertical range of cells for a column. To enter an array formula, press CTRL+SHIFT+ENTER.
- Row_num and column_num must point to a cell within array; otherwise, INDEX returns the #REF! error value.

Help for the STATA procedure nearmrg: Nearest match merging of datasets

Notes:

1. To install, type "findit nearmrg" in the STATA command window
2. Author information at the end

nearmrg [varlist] using , nearvar(varname) [genmatch(newvarname) lower upper roundup keep mergeoptions]

Description

nearmrg performs nearest match merging of two datasets. nearmrg matches observations that are closest on the variable specified in nearvar().

The options upper, lower, and roundup allow some adjustments to how closest is defined. Additional variables may be specified in an optional varlist and these variables are treated as standard merge variable which must match exactly.. This option allows nearest matching within subsets defined by the varlist.

nearmrg was designed as a way to use lookup tables that have binned or rounded values on the variable of interest.

Options

nearvar() specifies the variable in the master and using datasets that is to be matched as closely as possible. nearvar() is not optional and must be unique in the using dataset, but not necessarily in the master dataset.

[Note by AB: This is unnecessarily restrictive. In fact, nearmrg only requires that nearvar() be unique *within* groups.]

lower, upper, roundup are mutually exclusive options that alter the default approach to defining the nearest match for nearvar. lower matches to the closest value of nearvar in the using dataset that is less than or equal to nearvar in the master dataset. upper matches to the closest value that is greater than or equal to nearvar. roundup breaks distance ties by always selecting the higher value instead of the default lower value. If none of these options are specified, nearmrg matches to the closest observation defined as minimizing the absolute difference between nearvar in the master and using datasets. If nearvar is a string variable, then either the lower or upper option must be selected or an error will occur when calculating distance.

genmatch() specifies that a new variable should be created in the master datset that identifies the specific value of nearvar in the using dataset that was matched.

mergeoptions allows the user to specify any of the standard options for merge such as update, _merge(varname), etc. One difference from the standard behavior of merge is that nokeep is the default but can be overridden by the keep option. see help merge

keep overrides the default behavior (always using the nokeep option of merge) and instead includes unmatched observations from the using dataset.

Examples

```
. neamrg using ageinfo, nearvar(age) genmatch(agebin)
```

to look up information by age in ageinfo.dta where values are binned and create new var agebin with closest matched age value

```
. neamrg region gender using ageinfo2, nearvar(age) genmatch(agebin) _merge(mrgagebin)
```

to look up information by age within region and gender and name the _merge var mrgagebin

Authors

Michael Blasnik
M Blasnik & Associates
michael.blasnik@verizon.net

Katherine Smith
Clinical Epidemiology and Biostatistics Unit
Murdoch Childrens Research Institute
katherine.smith@mcri.edu.au

About the author

Aldo Benini has a dual career in rural development, with a focus on Bangladesh and another on organizations of the poor, and in humanitarian action. In the latter capacity, he has worked for the International Committee of the Red Cross and for the Global Landmine Survey. He has a Ph.D. in sociology from the University of Bielefeld, Germany, based on field research in community development in West Africa.

Benini is a citizen of Switzerland and an independent researcher based in Washington DC.

He can be contacted at *aldobenini [at] gmail.com*. Several of his publications are available at http://aldo-benini.org.

Other technical notes include:

(2007). The Wealth of the Poor. Simplifying living standards measurements with Rasch scales.

(2008). Efficient linking of lists in humanitarian data management.

(2010). Text Analysis under Time Pressure. Tools for humanitarian and development workers.

(2010). Efficient Survey Data Entry. A template for development NGOs.

(2010) "What are your parents' names?" - Efficient creation of variables based on subgroup-specific attribute values.