Aldo Benini

"What are your parents' names?" -

Efficient creation of variables based on subgroup-specific attribute values. A technical note for program monitors and researchers working with large data tables

Summary

This technical note addresses a specific data management problem: how to transfer attributes of one member of a group to a new variable taking that value uniformly in all members of the group. For example, a study of school drop-out may use a subset of records from a large demographic table in a household survey - the records of adolescents who may have completed their education or not. Completion rates may be of interest for children of literate vs. illiterate parents. However, the literacy status of the parents will have been entered into their own records, not in the children's directly. The information needs to be transferred to newly added context variables.

The data manipulations needed in this situation are not straightforward, at least not for the large group of program monitors and researchers in developing nations who manage data primarily in spreadsheets. I give solutions for MS Excel, MS Access and for STATA users. Demonstration files can be downloaded in both applications. The example is from a specific survey context, but the solutions are easy to generalize to problems that are formally analogous.

This note may be used also as a supplement to an earlier technical note *"Efficient linking of lists in humanitarian data management"* (Benini 2008). Here, I detailed solutions, in Excel and STATA, of how to link two lists when the identifying information (names, addresses) on both sides may have spelling differences. The present note supplements it with techniques to recover identifying information (e.g. the names of parents) initially held in records other than those of the units to be matched (e.g. children in household member lists to be matched to students in lists prepared by teachers).

Contents

Summary..1

Acknowledgement..3

Introduction..4

The data management challenge..5

The strategy for creating group-wise context variables..6

Solutions for Excel users..8

Solution #1: Formulas directly operating in the new variables..8

Solution #2: Importing values from temporary lookup tables..10

Creating the variables in MS Access..11

Creating the variables in STATA..15

Final thoughts..15

References..16

Appendix..18

STATA do-file..18

About the author..20

Acknowledgement

Paul von Bünau, Technical University Berlin, contributed the SQL code and its explanations for the solution in MS Access.

Introduction

In analyzing data on units belonging to groups (clustered data: individuals in households, workers in departments, etc.), the need may arise to create variables set to the value of an attribute of a specific member of the group. To exemplify: the name of the head of the household (identical for all members of the household) in a table of household member records; the salary of the department head (identical for all workers in the department) in a table of employees with department, function, salary as individual variables.

The purposes of adding such variables may be vastly different: the names of household heads may be helpful when matching the records of children of school-going age from household surveys to lists of students prepared by school administrators (who note, in the same student record, the parents' names); the salaries of department heads may be used to calculate wage differentials between supervisors and juniors in a financial analysis.

Regardless of substantive purpose, the formal challenge is to extract the value of an observed attribute in one member of the group - who is defined by yet another variable - and to write it into a new variable, in the records of all other group members. To revert to the first example: we look for the value of the variable "Name" in the member of the household defined, in the variable "Relation", as "household head". Once the household head's name has been found, it is then written, identically, into the variable "Name of household head" in all member records for this household.

This problem cropped up in the analysis of a large table of household members (over 110,000) resulting from a community baseline survey that a development NGO in Bangladesh had conducted. Two kinds of variables were needed: Monitoring associates performed manual matching with student lists. For quick and convenient side-by-side comparisons of names, they needed an extract of children of school-going age only, showing also the names of the parents.

Second, for the analysis of primary school completion rates, data on parental education was desired. The literacy status of household members was known and was used as a rough proxy of education level.

In both cases, the attributes (name, literacy) of the household head and of the first person listed as "wife" were used as approximations to the values of father and mother. The validity challenges are obvious, particularly for three-generation households and for children whose father or mother had died or moved away. In this note, however, the focus is on the formal data management side.

Extracting these values and transferring them to the proper place may prove, depending on skills, experience and applications, surprisingly difficult. In all but very small datasets, manual transfer is not an option; an algorithm for wholesale calculation is needed. At the time, the problem arose in a program monitoring unit that was collaborating on a spreadsheet (MS Excel) platform - as one will find in numerous development and relief NGOs that conduct surveys and manage extensive figurework on projects and programs. In the rest of the paper, I present solutions that can be generalized to other data management problems of the same formal type. Two solutions are offered for Excel users; I also present code in STATA, a statistical application widely used in econometric research. A zip file, available at this link, holds an Excel workbook, a demonstration Access database as well as a STATA dataset and a do-file. Readers may find it easier to follow the sections below by practicing with the demonstration files.

The data management challenge

At the risk of repetition, we work through the formal aspects of the problem using the example of extracting the parental literacy status. In short graphic mode, the challenge is to create two variables, "household head is literate" and "head's wife is literate", in the records of the children living in the same household. These serve as the basis for imputing father's and mother's literacy. Notionally, this is shown in this table:

Table 1: Converting an attribute of one HH member into a context attribute of another member

Household number	Household member ID	Name	Relation	Age	Literate	Household head is literate	Wife is literate
1	1	A	HH head	40	Yes		
1	2	B	Wife	35	No		
1	3	C	Daughter	15	Yes	Yes	No

In this scheme, C, the daughter, inherits the literacy of the household head as a context attribute in a new variable separate from her own literacy - "Household head is literate". This attribution is highlighted in the two yellow cells. On the assumption that A is C's father, this variable is the basis of imputing her father's literacy. Similarly for B, the household head wife, as the mother of C (orange cells).

Creating group-wise context variables

The creation of such household-context variables is an instance of a more general class of intra-group attribution. In formal terms, members of a given group inherit, as a group context variable, an attribute of an individual member who meets a certain condition.

In this type of data management problem, the challenge is to transfer to a context variable the value of an individual-level variable from one member of the group in point. This member can be defined by a unique attribute (e.g., there is only one head of household or department) or by the first occurrence in a subset of members who share the same attribute (the first recorded wife if there are several wives) or the last occurrence (e.g., the latest annual income if the group is the sequence of annual incomes in a given household or enterprise).

This situation is important to distinguish from another type of context variables that are often needed and used in data management and subsequent analysis. These are variables calculated as a function of the values of some original variable in all group members. The function could be the mean, sum, count, maximum, etc. of all values. The mean salary of all employees in the given department is a perfect example. In our simple 3-person household illustration, the mean literacy rate is 67 percent (two of the three members are literate); this is another, somewhat silly, example of a context variable based on the values of all group members.

As many users of spreadsheet applications will know, this latter task is less demanding. In Microsoft Excel, the most popular such application, functions such as SUMIF, AVERAGEIF, COUNTIF painlessly deliver the results[1]. By contrast, the devices that produce context variables based on attributes of one group member are far less straightforward.

The strategy for creating group-wise context variables

In our substantive context - primary school completion - we want to create parental education variables with values for all adolescents in the sample. Pragmatically, one might be tempted to find an algorithm that calculates these variables only for the sample. However, the algorithm may be useful in other contexts as well. Thus, in order to make work simpler, and to avoid loss of generality to other applications, I recommend calculating the values for all records. Subsequently, one may choose to delete values in records in which they are meaningless, or to selectively recode them into new variables, or to ignore the problem and simply subset the cases in the estimation procedure.

The following, therefore, assumes that the algorithm calculates context variables in all records. Adjustments will be made later, as indicated above, according to analytic interests. In our context, for example, imputations are made from household head to father, and from first wife to mother, and adjustments are needed to take account of female household heads. These adjustments are context and analysis-specific; they are not part of the general mechanics of calculating the initial context variables.

A second assumption is that the dataset comes with a group identifier, that the identifier is the same for all members of a given group, and that it is unique between groups. Thus, no two households share the same household ID.

A third assumption is that the data have been cleaned, and spelling variants removed. This requirement extends to the group identifiers and to the attribute(s) that define the group member whose value in a substantive variable is to be transferred to all fellow group members. For example, in our application, "Wife" must be uniformly spelt; "Wife " with a trailing space would not be recognized and would produce an error.

[1] In certain more complex situations, a Pivot table may need to be created with the group name or identifier as the row field, and the - summed, counted, averaged, etc. - variable of interest as the data field, and a complex condition set in the report filter (formerly known as the "page field"). In a second step, the Pivot result is re-imported into the data table using the function VLOOKUP. This strategy is called for also when the names of groups have multiple spelling variants and may need cleaning or recoding before proceeding any further.

What the algorithm should do

At a minimum, the algorithm must

- identify the records that belong to a group (in our context: all the members of a given household)
- identify the group member whose attributes of interest will be transferred to the other members (here: e.g. the household head)
- read the values of those variables and write them to the correct addresses (cells at the intersection of group members and new context variables)
- produce missing values in these two situations: where there is no defining group member (e.g. no household head); where the substantive variable of the defining group member is missing (e.g., it is not known whether the household head is literate).

In addition, it is desirable that the algorithm can

- work with both numeric and alpha-numeric group identifiers and defining member attributes
- transfer variables of substantive interest that are text-based.

While the first desideratum is purely one of convenience, the second opens the algorithm to wider application. Notably, it allows members of a group to receive, in context variables, the proper names of particular other group members. For example, we can create and compute the variables "name of household head" and "name of the (first occurrence of) wife". Placing the names of parents directly in the children's records facilitates matching with external lists of students. In this case, records from a household head-centric list ("is son/daughter of household head") are made matchable to a child-centric list ("student - is son/daughter of Mr. X and Mrs. Y").

This catalogue of requirements and desiderata suggests that a sequence of commands, starting with the identification of groups and ending with the creation and filling of the new variables, be programmed and administered. This would be the classic scenario for a database macro.

However, whether a macro is the suitable tool for this task depends on the application:

- In a spreadsheet environment, the logic can be inverted. We first designate new variables (by writing their names to the top row of unused columns) and then populate the cells in these columns with formulas that extract the desired information.
- Similarly, working with a database application such as MS Access may entail the designation of new variables. However, these reside outside the original data table, in queries to be built with the help of a special language (Structured Query Language, SQL).

- In a statistical application such as STATA, data tables do not contain formulas; a macro-like bundle of commands does the job.

I demonstrate all three approaches.

What is in the demonstration files

The downloadable zip file contains demo material for each of the above-mentioned applications.

This demonstration spreadsheet puts fictional data in a small number of household member records for a total of five households only. Two types of attributes are transferred to the context variables: the names of the household head and first wife as well as the head's gender (text variable) and a binary (1 or 0) variable for literacy (numeric variable). Missing-value and first-occurrence (two wives) situations are illustrated. The spreadsheet also shows the formula to adjust for female household heads, but, as emphasized above, this is a specific requirement of our substantive context, not an element of the basic lookup mechanics. It is included here for completeness only and is not discussed.

The data used in the Access and STATA demonstrations are the same as for Excel. In addition, the commands needed in STATA are packaged into a do-file.

Solutions for Excel users

We work with compound formulas to extract, in variables of interest, the values for the defining member within each group and place them, for all members of that group, in new context variables. In our example, each group is the set of persons enumerated in a given household, and the defining members are its head of households, and, for separate variables, members designated as "Wife". We exploit the fact that the combination of household ID and unique or first instance of kinship term determines one particular record.

I first give a detailed solution that relies on formulas directly written into the new variables in the data sheet itself. For those challenged by the formulas, I sketch a workaround that requires the creation of temporary additional sheets and a lookup operation.

Solution #1: Formulas directly operating in the new variables

Working in Excel 2007 and in its R1C1 cell reference style, we combine several functions in the lookup formulas, in this sequence of building from inside out: the concatenation operator (&, the ampersand), MATCH, INDEX, ISBLANK, and IF. The steps in building the formula are best followed in the sheet "FormulaBuild" of the demonstration workbook. In addition, I recommend consulting the Excel help for each of the functions. I adapted part of the formula from Dalgleis (Undated), who contributed the idea of using concatenated search terms and lookup arrays in MATCH and INDEX. The use of ISBLANK and IF remedies a much lamented peculiarity in Excel

2007 - its habit to flush blanks in Pivot tables and lookup results with zeros where blanks are appropriate. These parts of the formula may not be required in Excel 2003.

The generic version of the "engine" of the compound formula is

```
=INDEX([lookup array in substantive variable], MATCH([mixed reference to
group ID]&[Defining member attribute value],[concatenated group ID array
and defining attribute array],0))
```

as exemplified by the extraction of the household head's name:

```
=INDEX(R2C4:R28C4, MATCH(RC3&"Household head",R2C3:R28C3&R2C5:R28C5,0))
```

The formulas have to be entered as array formulas, by hitting Control + Shift + Enter. They are recognized in the formula bar by the curly brackets that Excels puts around them[2].

The lookup ranges can be named prior (recommended); named ranges make the formulas more intuitive and easier to write or adapt to other variables. The above example, using named ranges, looks like this:

```
=INDEX(Name, MATCH(RC3&"Household head",Hhcode&Relation,0))
```

The demonstration workbook has two sheets, "RangeReferences" (formulas using absolute range reference style for arrays) and "RangeNames" (formulas using named ranges). Both produce the same results.

Safeguard against incorrect zeros in Excel 2007

Annoyingly, the forced addition of the ISBLANK and IF functions, to safeguard against invalid zeros in Excel 2007, makes the formulas cumbersome. Example:

```
=IF(ISBLANK(INDEX(Name, MATCH(RC3&"Household head",
Hhcode&Relation,0)))=TRUE,"",INDEX(Name, MATCH(RC3&"Household head",
Hhcode&Relation,0)))
```

This complication slows down execution. In a household member data sheet with 110,000 records, calculating one of the household context variables by the full compound formula took about 30 minutes on an 8 GB RAM 2-processor Windows 7 machine. Users working with less well endowed machines may need to cut down the job into segments, each time pasting the formula to a part of the target range only. For data tables that are considerably smaller, speed and memory should be less of a problem. At any rate, once the user is satisfied with the result, it is recommended to replace the formulas with their values, through copy - paste special (values only).

[2] This is liable to confound users (including us initially), particularly among those who have developed a reflex for first selecting a target range for multi-cell output functions (e.g. FREQUENCY) and only then hitting the three keys. Our situation here is nearly the obverse; the arrays are in the input. Thus the formula is first written and entered as an array function for one cell in the new column. It can then be copied and pasted to the remainder of the target range. The first cell must not be part of the range to paste into.

Not a solution: A note on SUMPRODUCT

A number of Excel advice sites on the Web (e.g., Daveexcel 2006) recommend the use of the function SUMPRODUCT for what they sometimes call "multi-condition lookup" (of which our data management problem is an instance). This offers an elegant as well as parsimonious solution. However, it works only with variables of substantive interest that are numeric. Nor have I found a way to ensure that it returns blanks or error codes for missing values. Given these limitations, this function should not be used for the purpose on hand.

Solution #2: Importing values from temporary lookup tables

The core formula offered by Dalgleis (op.cit.), with its smart combination of concatenation, MATCH and INDEX, is the most elegant solution to this data management problem that I have come across so far in the spreadsheet context. It has the great advantage that it handles numeric as well as text variables.

However, its inner workings are not easy to understand. There are workarounds that do not require array formulas.

Possibly the simplest way to get there is by creating separate lookup tables (one for every defining group member attribute value, e.g. one for household heads, and a different one for wives). To do this safely, one ought to make a separate copy of the workbook for each defining value. These are auxiliary and may be deleted after the lookup operation has succeeded. We sort on the defining attribute, group ID and record ID (in this order). We then delete all records other than those with the defining value in point (e.g. all that are not household heads). We name a lookup range with the group ID in the leftmost column.

The lookup ranges in the temporary workbooks can either be copied into the master workbook or can be referenced while the workbooks stay open. Either way, we return to the master workbook and, in the newly named context variable, use the Excel function VLOOKUP to import the values of corresponding variables in the lookup range. For this, we use the group ID as the lookup value. VLOOKUP will find the first instance of the group ID (e.g. the record of the first wife if there are several) and look up the value of the variable to which we point it. The generic formula is:

```
=VLOOKUP([group ID], [temporary lookup range], [column number of
substantive variable in lookup range], FALSE)
```

which, in the sheet "UsingVlookup" of the demo workbook, works out for the name of the household head (see cell R2C9) as:

```
=VLOOKUP(RC3, HHheadLookup, 2, FALSE).
```

We repeat this for all variables of substantive interest. We then replace the VLOOKUP formulas with their values, in order to make them independent from the temporary lookup range. Error messages for missing records (e.g. because a household does not have a recorded head) should also be replaced by blanks (not yet done in the demo workbook). All cells with errors can conveniently be selected by pressing F5 - Special - Formulas - Errors.

Unfortunately, VLOOKUP in Excel 2007 follows the same habit of turning missing values into zeros. For statistical work, these illegal zeros are bad. Therefore, the formula should be wrapped up in the same IF and ISBLANK syntax as shown in the demonstration workbook. Again, the formulas become rather unwieldy. For an example, see the formula in R3C3 of the same sheet:

```
=IF(ISBLANK(VLOOKUP(RC3, HHheadLookup, 2, FALSE))=TRUE, "",
VLOOKUP(RC3, HHheadLookup, 2, FALSE))
```

To see why this is necessary, consider Household A. We have no information on the literacy of Shuvanti, the wife. If, in the formulas in R2C13:R5C13, IF and ISBLANK were not used, the result would be 0, meaning that these four individuals live in a household with an illiterate wife. This would be incorrect.

Creating the variables in MS Access

The methods demonstrated so far are for those who work in MS Excel. Others may have access to applications in which they can create the context variables faster. Notably, users of the database application MS Access might scorn convoluted formulas and point out that Access offers simpler solutions through queries. However, it is a fair guess that the majority of monitoring and research workers in poorer countries, while their computers may be configured with Access, do not have the skills to write the needed SQL code. The Excel solutions, while clumsier, appeal to a larger audience.

This section is meant for Access users with basic notions of writing SQL code. We demonstrate the result in two steps. Working with the downloadable demo database, in a first step we create a query to produce the context variables. In a second step, in order to obtain the same final results as in the Excel demonstration workbook, we build a second query off the first, adding variables for the putative literacy status of father and mother.

The code and explanations of the first step were provided by von Bünau (2010) and are closely followed here.

Step 1: Creating the context variables

As in the Excel solution, the challenge is to create, for each household member record, variables holding the name of the household head, the name of the first wife, the gender of the household head, and the literacy status of household head and first wife.

The original data is held in a table named "DemoData". Opening it, we use, in the menu tab "Create", the query wizard to create a simple query holding all the fields of DemoData. We name this query "AddGroupVariables". In the ViewTab (under the Home menu), we bring up its SQL view and replace the existing code with this:

```
SELECT

D.*,

(SELECT Name FROM DemoData WHERE Hhcode=D.Hhcode AND Relation='Household head') AS NameHHhead,

(SELECT TOP 1 Name FROM DemoData WHERE Hhcode=D.Hhcode AND Relation='Wife') AS NameWife,

(SELECT  Gender FROM DemoData WHERE Hhcode=D.Hhcode AND Relation='Household head') AS GenderHHhead,

(SELECT IsLiterate FROM DemoData WHERE Hhcode=D.Hhcode AND Relation='Household head') AS LiterateHHHead,

(SELECT TOP 1 IsLiterate FROM DemoData WHERE Hhcode=D.Hhcode AND Relation='Wife') AS LiterateWife

FROM DemoData AS D;
```

The SQL construction chosen here for adding group-level variables is the *correlated subquery*. A correlated subquery is a subquery that is nested in an outer query from which it references fields; this is what makes it *correlated*. In our example, we use subqueries to add computed fields to a SELECT statement, and the correlation is the link between households and their members. In order for this operation to be well defined, the subquery may only return one column and, more importantly, one row. The latter condition can only be verified at execution time, and will result in an error if violated.

The following is annotated SQL code, abbreviated to one example each of a *SELECT* and of a *SELECT TOP 1* statement. All syntactic elements are in green. Comments refer to the line above.

```
SELECT D.*,
```

This is the beginning of the outer query: D.* means that all fields from the table with alias D shall be returned. The alias is defined at the end of the outer query (FROM clause). This is necessary to differentiate the tables that the inner and the outer queries operate on, because they have the same name here, because e.g. household heads are in the same table as household members.

```
(SELECT IsLiterate FROM DemoData WHERE Hhcode=D.Hhcode AND
Relation='Household head') AS LiterateHHHead,
```

This is the correlated subquery, in parenthesis. The correlated subquery appears on the field list of the SELECT statement and the name of the computed field is defined by AS. Note that the correlation between the subquery and the outer query is created by the condition in the WHERE clause, where fields of both tables appear.

```
(SELECT TOP 1 IsLiterate FROM DemoData WHERE Hhcode=D.Hhcode AND
Relation='Wife') AS LiterateWife
```

This is the SELECT TOP 1-type correlated subquery. Since it is the last element on the field list, it is not followed by a comma. The TOP 1 qualifier truncates the result set to the first row; this is to avoid an error when there is more than one wife in a household. Note that the ordering of the rows was done at the time of creating the data table, not in this query operation. Thus the substantive selection of a household member as "first wife" is extraneous to the query.

```
FROM DemoData AS D;
```

This is the end of the enclosing outer query. It is here that the alias for the DemoData table is defined that is used in the correlated subqueries, to differentiate inner and outer table sources.

Step 2: Creating the parental literacy variables

To repeat, this step is not an essential part of the creation of variables based on subgroup-specific attribute values, and in many contexts will not be needed. We document it here in order to present the same result as in the Excel demonstration file. A second motivation is to exemplify the use of IIf-expressions.

As in Step 1, we first create a simple query. This time it is based, not on the initial data table, but on the query that we just created: qryAddGroupVariables. We import all its variables into the new query, which we name "qryAddFatherMotherLiterate". As before, we replace the SQL code. In this case, with:

```
SELECT

qryAddGroupVariables.RecNo, qryAddGroupVariables.Hhcode,
qryAddGroupVariables.Name, qryAddGroupVariables.Relation,
qryAddGroupVariables.Gender, qryAddGroupVariables.Age,
qryAddGroupVariables.IsLiterate,

qryAddGroupVariables.LiterateHHHead, qryAddGroupVariables.NameHHhead,
qryAddGroupVariables.NameWife, qryAddGroupVariables.GenderHHhead,
qryAddGroupVariables.LiterateHHHead, qryAddGroupVariables.LiterateWife,

IIf([GenderHHhead]="Male",[LiterateHHHead],) AS FatherBelievedLiterate,

IIf([GenderHHhead]="Female",[LiterateHHHead],[LiterateWife]) AS
MotherBelievedLiterate

FROM qryAddGroupVariables;
```

The first block after SELECT simply repeats the original variables. The second block contains the six variables created in the first query.

The next statement creates the variable *"FatherBelievedLiterate"*. It uses the database version of the IF-function (IIf) to condition its value on the gender of the household head. Note that at this point we no longer need the syntax of correlated subqueries. For our second query works off the first (statement at the end: "*FROM qryAddGroupVariables*"*)*, in which *GenderHHhead* and *LiterateHHHead* have already been computed for each record. This variable then takes the value of *LiterateHHHead* if the individual in point lives in a household with a male head (*IIf([GenderHHhead]="Male"*). Else, it is left blank. Note that the instruction to leave it blank when the head is female is given by NOT writing anything between the comma and the closing parenthesis in

```
IIf([GenderHHhead]="Male",[LiterateHHHead],)
```

Some users might be tempted to write *IIf([GenderHHhead]="Male",[LiterateHHHead],"").* However, inserting this "" in the falsepart element of the clause will turn FatherBelievedLiterate into a string variable. In this case, for analysis purposes, the numeric format is preferable.

The following statement creates the variable *"MotherBelievedLiterate"*. The logic is similar. If the household head is female, the value is taken from the variable *LiterateHHHead*. Else from *LiterateWife*. Both of these variables are already contained in the first query.

Comparison with the Excel operation

The Access operation entails, besides the original data table, the creation of two queries, which are not part of the data table. In Excel, the new context variables can be created within the same original data table. However, this advantage is minimal; in Access the transfer of all the original data from the table to the first query, and hence to the second, is trivial. The Access solution has the added benefits that there are no error results (which can occur in Excel and need to be deleted in an extra step), and no special safeguards are needed against erroneous zeros. Also, the Access SQL statements are less complex than the Excel formulas.

Creating the variables in STATA

STATA users are not numerous among monitoring and reporting staff in non-profits, development NGOs or micro-finance organizations, but they form a substantial community among researchers. The logic of proceeding in STATA is different from Excel. Rather than working from formulas built in the new variables, the process is driven by a sequence of commands that reorder the records, create auxiliary variables (which may be deleted afterwards) and then only define and calculate the new variables.

Statistical programs are very "conscientious" about missing values, and thus the annoying safeguards that Excel 2007 necessitates in this regard are not an issue. On the other hand, as long as the formulas have not been replaced by their results (values only), Excel updates formula results automatically when the data are changed. STATA does not have this ability.

The STATA commands used for the purpose are listed in a macro file (a so-called do-file) in the zipped demo files package as well as in the appendix. The mainstay devices are the ability to tag the first among observations that meet certain conditions (the egen tag command) and the ability to reference the preceding observation (subscripting with [_n-1]). The do-file is written for our specific household survey dataset, but it is straightforward how to adapt it for analogous tasks.

Final thoughts

Between data capture and cleaning on one hand and analysis, interpretation, reporting and dissemination on the other, data management fulfills a critical bridging function. The boundaries on both sides are fluid and may only be of academic interest. Data managers bring defectively formatted data into a shape amenable to analysis; they accelerate data cleaning where frequent identical impurities can be purged with global commands. Anticipating the analyst's needs, they generate variables through various operations. They reformat, recode, summarize and query the raw data.

Unresolved challenges in data management detract from the quality of the data or from the analytic potential. Delays in resolving them compress the time left for the subsequent operations. Resistant challenges may ultimately become showstoppers.

A generic solution, taught by a specific example

This paper addresses a specific data management challenge that at the time bedeviled parts of the author's survey data analysis. A solution was found much later. On second thought, its basic structure appeared to respond to a whole class of data management problems - how to create variables that are based on subgroup-specific attribute values. It seems productive to discuss this class of problems in a broad manner that adds a generic solution to the toolbox of the data management community.

The solution is presented by way of example, using a small extract of the original data, yet making it obvious how the algorithms can be adapted to other situations. I work through the same problem using three different applications in which survey and statistical data are often managed: a spreadsheet, a database and a statistical application.

Solutions for workers who are not professional data managers

For professional data managers (of whom I am not one), this class of problems and its (potentially numerous) solutions must be fairly basic and hardly worth the ink spilled over so many pages. However, in less well endowed organizational contexts - such as in the monitoring, research and evaluation units of development NGOs -, a clear grasp of the challenge cannot be assumed, and even less so ready solutions. Spreadsheet applications - foremost MS Excel - are the most common platform on which such groups manage data. Their typical users have limited data management skills, but in larger organizations some may be found with intermediate skills. Because of the widespread use of Excel, informal and cross-departmental learning on this platform is more common that among Access users or users of statistical programs. The Excel solutions are therefore deliberately given more space in this paper. Data managers conversant with Access or with STATA may find the solutions given in these applications more straightforward and preferable to Excel's.

References

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

Dalgleis, D. (Undated). "Pivot Table : First Function." The Daily Reviewer Retrieved 13 October 2010, from http://thedailyreviewer.com/office/view/pivot-table-first-function-112366088.

Daveexcel. (2006, 11 Sept 2006). "Lookup Two Criteria using SUMPRODUCT."   Retrieved 13 October 2010, from http://www.excelforum.com/excel-general/580001-help-with-multiple-conditions-with-lookup-table.html.

von Bünau, P. (2010). Correlated Subqueries for Group-Level Variables. E-mail to Aldo Benini [29 December 2010]. Berlin, Technical University of Berlin.

8 November 2010 / Revised 30 December 2010

Appendix

STATA do-file

```
* Do-file to create new variables holding, for all members within a
given household, attributes of particular members (household head,
first wife)

* use [path]\Benini_STATAdemo_VariablesOnSubgroupAttributes_101108.dta,
clear

* variable name   type   format      label      variable label
* ----------------------------------------------------------------------
* recno           byte   %8.0g                  Record number

* hhcode          str1   %9s                    Household identifier
[the group identifier]

* name            str11  %11s                   Name
[variable of substantive interest]

* relation        str15  %15s                   Relation
[defining attribute of group member]

* gender          str6   %9s                    Gender
[variable of substantive interest]

* age             byte   %8.0g                  Age
[not used in this example]

* isliterate      byte   %8.0g                  Is literate
[variable of substantive interest]
* ----------------------------------------------------------------------


* Make sure initial sort order can be re-created, even if no record
number variable exists.
* Create temporary record number:
gen temprecno = _n

* Create tags for defining members, create variables and compute values
by household.
* Sort on the defining attribute, group ID and record ID (in this
order)

* 1. For household heads:
* Create tag for household head [to be on safe side, do not assume
unique head in household]
gsort  hhcode relation temprecno
egen tagHHhead = tag(hhcode) if relation == "Household head"

* Create variables of interest:
gsort  hhcode -tagHHhead
gen nameHHhead =  name if  tagHHhead
```

```
replace  nameHHhead =  nameHHhead[_n-1] if  tagHHhead==0 &  hhcode ==
hhcode[_n-1]

gen genderHHhead =  gender if  tagHHhead
replace  genderHHhead =  genderHHhead[_n-1] if  tagHHhead==0 &  hhcode
==  hhcode[_n-1]

gen literateHHhead =   isliterate if  tagHHhead
replace  literateHHhead =  literateHHhead[_n-1] if  tagHHhead==0 &
hhcode ==  hhcode[_n-1]

* 2. For first wives:
* Create tag for first wife:
gsort  hhcode relation temprecno
egen tagFirstWife = tag(hhcode) if relation == "Wife"

* Create variables of interest:
gsort  hhcode -tagFirstWife
gen nameFirstWife =  name if   tagFirstWife
replace   nameFirstWife=   nameFirstWife[_n-1] if   tagFirstWife==0 &
hhcode ==  hhcode[_n-1]

gen literateFirstWife =   isliterate if   tagFirstWife
replace   literateFirstWife=    literateFirstWife[_n-1] if
tagFirstWife==0 &  hhcode ==  hhcode[_n-1]

sort  temprecno
drop  temprecno tag*

* Label new variables as convenient
describe
codebook  nameHHhead genderHHhead literateHHhead nameFirstWife
literateFirstWife

* Save as appropriate.

* End of do-file
* -----------------------------------------------------------------------
```

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.