The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–22

From resultssets to resultstables in Stata

Roger B. Newson
National Heart and Lung Institute, Imperial College London
London, United Kingdom
r.newson@imperial.ac.uk

Abstract. The `listtab` package supersedes `listtex`. It inputs a list of variables, and outputs them as a table in one of a range of formats, including TEX, LATEX, HTML, Microsoft Rich Text Format (RTF), or possibly future XML–based formats. It works with a team of 4 other packages, `sdecode`, `chardef`, `xrewide` and `ingap`. The `sdecode` package is an extension of `decode`, allowing the user to add prefixes and/or suffixes, and to output exponents as TEX, HTML or RTF superscripts. It is called by another package, `bmjcip`, to convert estimates, confidence limits and *p*–values from numeric to string. The `chardef` package is an extension of `char define`, allowing the user to define a characteristic for a list of variables, adding prefixes and/or suffixes. The `xrewide` package is an extension of `reshape wide`, allowing the user to store additional results in variable characteristics and/or local macros. The `ingap` package inserts gap observations into a dataset, to form gap rows when the dataset is converted to a table. Together, these packages form a general toolkit to convert Stata datasets (or resultssets) to tables, complete with row labels and column labels. Examples are demonstrated, using LATEX, and also using the `rtfutil` package to create RTF documents. This article uses data from the Avon Longitudinal Study of Parents and Children (ALSPAC), based at Bristol University, UK.

Keywords: st0001, table, resultsset, TEX, LATEX, HTML, XML, RTF, `listtab`, `listtex`, `sdecode`, `chardef`, `xrewide`, `ingap`, `bmjcip`, `rtfutil`, `keyby`, `addinby`.

1 Introduction

Statisticians and other scientists routinely produce tables of descriptive and inferential statistics. Unsurprisingly, Stata users have produced a large number of add–on packages to input the various Stata data and output structures and to output tables of statistics in a variety of formats, such as TEX, LATEX, Hypertext Markup Language (HTML), Extensible Markup Language (XML), Microsoft Rich Text Format (RTF), or delimited text tables for menu–driven manual incorporation into word processor diocuments. The packages for this purpose most frequently downloaded from SSC are Roy Wada's `outreg2`, Ben Jann's `estout`, John Luke Gallup's `outreg`, and Ian Watson's `tabout`. Some very inspiring presentations were given at the 2009 UK Stata Users' Meeting by Ben Jann (Jann (2009)), who gave a tutorial on `estout`, and by Adam Jacobs (Jacobs (2009)), who described some of his own programs to create XML–based documents for Open Document Format (ODF) packages.

My own method for table production was based initially on the `listtex` package, which has since been superseded by `listtab` for users of Stata Versions 10 or higher,

although the latest versions of both packages are still downloadable from SSC. Both of these packages input a list of variables from the Stata dataset in memory, and output a text table (in the Stata log and/or in a file) with 1 row per observation, derived by outputting the values of these variables, preceded by a `begin()` string option, separated by a `delimiter()` string option, and suffixed by an `end()` string option. Optionally, the user may specify a list of string header rows, to be output before the observation rows, using the `headlines()` option, and/or a list of string footer rows, to be output after the observation rows, using the `footlines()` option. This general format includes, as special cases, most table formats known to computer science, and probably a few yet to be invented. To save the user the trouble of specifying a long list of options, there is a `rstyle()` option, specifying one of a collection of row style names, such as `rstyle(tabular)` for LaTeX `tabular` environments (Lamport (1994)). A row style is defined as a vector of 4 string values, namely a `begin()` string, a `delimiter()` string, an `end()` string, and a `missnum()` string, to be output in the place of numeric missing values. All of these options default to the empty string for `listtab`. However, the `delimiter()` option defaults to the ampersand `&` for `listtex`, because I was thinking of TeX tables as the norm when I wrote `listtex`, and the `missnum()` option is probably not often used with either package, because numeric variables should usually be converted to string variables before being output to tables.

The `listtab`/`listtex` method is simple and versatile, if only the user already has a dataset in memory with 1 observation per table row. Fortunately, StataCorp, and other Stata users, had already developed a variety of programs, such as official Stata's `statsby` and the SSC package `parmest`, which could be used to produce such datasets (or resultssets). Some of these, and their use with `listtex`, are described in Newson (2003). Since that article was written, resultsset–processing has developed further, and additional Stata packages have become available.

This article aims to give an update of techniques for converting resultssets (and other datasets) to tables (or resultstables). The methods used can be summarized as "decode, label, list" (in the simplest cases), or as "decode, label, reshape, insert gaps, list" (in the most complicated cases).

1.1 Example resultsset: genotype frequencies in ALSPAC

The resultsset which we will use to demonstrate these techniques was produced using data from the Avon Longitudinal Study of Parents and Children (ALSPAC). For further information about this birth cohort study, refer to the study website at *http://www.alspac.bris.ac.uk*. Analyses of some of these data are discussed in Shaheen et al. (2010) and Henderson et al. (2010).

In the ALSPAC birth cohort study, 14060 children, born in and around Bristol, UK in 1991 and 1992, have been observed through progression to adulthood. Subsets of these children, and of their mothers, gave blood or other tissue samples, which were used to measure their genotypes with respect to 24 biallelic polymorphisms, using assay methods developed by molecular geneticists. An allele is a version of a gene, a biallelic

polymorphism is a set of 2 possible alleles for a gene, and the genotype of a study subject with respect to a polymorphism is the total set of alleles present in the 2 copies of the human genome present in each cell of that study subject, one copy from the subject's mother's egg, and the other copy from the subject's father's sperm. Genotypes with respect to a biallelic polymorphism may have values of the form GG, Gg or gg, where G is the first allele and g is the second allele.

Before measuring associations of the two alleles of a biallelic polymorphism with a disease, geneticists usually measure the frequencies of the 3 possible genotypes in the sample of subjects. An important population parameter of this frequency distribution is the geometric mean homozygote/heterozygote ratio, defined as $\sqrt{P_{GG}P_{gg}}/P_{Gg}$, where P_{GG}, P_{Gg} and P_{gg} are the proportions of subjects in the population with genotypes GG, Gg and gg, respectively. The sample estimate of this ratio is $\sqrt{N_{GG}N_{gg}}/N_{Gg}$, and the standard error of its log is $\sqrt{1/(4N_{GG}) + 1/(4N_{gg}) + 1/N_{Gg}}$, where N_{GG}, N_{Gg} and N_{gg} are the numbers of subjects in the sample with genotypes GG, Gg and gg, respectively. If the geometric mean homozygote/heterozygote ratio is 0.5, as would be the case under random mating (where the maternal egg allele and the paternal sperm allele are statistically independent), then the frequency distribution of the genotypes is said to be in Hardy–Weinberg equilibrium. The alternative possibilities are a tendency to inbreeding (where the ratio is greater than 0.5, indicating a tendency for mothers and fathers to contribute the same allele) and a tendency to outbreeding (where the ratio is less than 0.5, indicating a tendency for mothers and fathers to contribute different alleles). Justifications of these formulas can be found in Newson (2008a).

The example resultsset to be used in this article contains 1 observation per polymorphism per DNA source ("Child" or "Maternal"), for each of 24 polymorphisms assayed, and data on the numbers of subjects (children or mothers) whose genotypes were assessed for the polymorphism and the numbers and percentages with each genotype, together with estimates and confidence intervals for the geometric mean homozygote/heterozygote ratios for 24 biallelic polymorphisms, and p–values for testing the hypothesis of Hardy–Weinberg equilibrium. The observations in the resultsset are sorted (and uniquely identified) by a key of 3 value–labelled numeric variables, `source` (indicating DNA source), `polygp` (indicating a polymorphism group corresponding to a particular protein specified by a particular gene), and `poly` (indicating the polymorphism). The resultsset was created using the SSC packages `xcontract`, `dsconcat` and `factmerg`, together with the `parmcip` module of the SSC package `parmest`. The methods used to create this resultsset are discussed in Newson (2008b), which also gives a plot of some of the confidence intervals. Methods for production of resultssets are discussed more generally in Newson (2003), and also in many more recent references, accessible by hypertext from the online help for the `parmest` package.

In this article, we will demonstrate how to convert these results into tables, in LaTeX (which Stata programmers frequently like to use), and in RTF (which produces Microsoft–ready documents, which their colleagues frequently demand). Section 2 explains how a table can be viewed as a special case of a Stata dataset. Sections 3–5 demonstrate increasingly advanced examples of table production in LaTeX. Section 6

describes the creation of tables in RTF documents. Finally, Section 7 outlines possible further extensions of the methods presented in this article. The example resultsset, together with the Stata do–files used to output the tables, can be downloaded (using Stata) as online supplementary material to this article.

2 Tables as Stata datasets

To create a table using `listtab`, we must first have a suitable Stata dataset in the memory, with 1 observation per table row.

In general, a well–formed Stata dataset should be designed to have 1 observation per *thing*, and data on *attributes_of_things*. For instance, the example resultsset has 1 observation per combination of DNA source ("Child" or "Maternal") and polymorphism, and data on genotype frequencies for that polymorphism in that DNA source, and on statistics derived from those frequencies. Specifically, a well–formed Stata dataset should be a table, as defined by the relational model of Date (1986). This model (in its essentials) assumes that a table (or dataset) is a mathematical function, whose domain is the set of existing combinations of key variable values (specifying the *things*), and whose range is the set of all possible combinations of non–key variable values (specifying the *attributes_of_things*). For a Stata dataset to conform to this model, it should be keyed (that is to say, its observations should be sorted *and* uniquely identified) by the values of a list of key variables. In the case of the example resultsset, these key variables are `source`, `polygp`, and `poly`.

Not all Stata datasets conform to the relational model. Some are not sorted, or are sorted by variables which do not uniquely identify the observations. Similarly, not all keyed Stata datasets are ideal for input to `listtab`. I would argue (from experience) that a Stata dataset input to `listtab` should be keyed, with key variables identifying the table rows in order of appearance on the page, possibly preceded in the key by key variables identifying the pages of a multi–page table. The dataset should also have the following features:

1. The variable list input to `listtab` should contain only string variables.

2. The first variable in the list should be a row label variable, whose value, in each observation, is determined from the values of the key variables in that observation.

3. The variables input to `listtab` should have one or more characteristics (see [P] **char**), containing the corresponding column labels for the output table.

The first of these features is a good idea because numeric variables nearly always have to be reformatted before inclusion in tables, in ways that are not provided by any official Stata format. The other two features are a good idea because tables in documents have row labels and column labels, both of which can contain justification, font specification, or other formatting information.

2.1 Making resultstables by breaking resultssets

A typical resultsset produced by `parmest`, `contract`, `collapse` or `statsby` contains mostly numeric variables, containing frequencies and/or sample statistics. Formatting resultssets for input to `listtab` is usually a destructive process, because these numeric variables are converted irreversibly to string variables. The process is even more destructive when the dataset is also subsetted and/or reshaped and/or provided with gap rows to make the table more readable. It is therefore usually a good idea to program this conversion process as a sequence of commands in a do–file, and to enclose these commands between a `preserve` statement and a `restore` statement (see [P] **preserve**). Stata programmers are frequently urged to avoid the use of `preserve` and `restore`, because they involve file processing and therefore consume computing resources. Unfortunately, in Stata Versions 1 to 12, the user may only have access to one dataset in memory at a time, and therefore frequently has no choice but to use `preserve` and `restore`. Fortunately, if a resultsset is small enough to be converted to a table, or even to a multi–page document of multi–page tables, then this resource intensity is not likely to be an important issue.

I usually follow a policy of writing a do–file to produce a keyed resultsset on disk, and then writing a second do–file to input this resultsset and to produce the tables (and plots), which frequently have to be revised iteratively before they have exactly the right look and feel.

2.2 Enforcing the relational model with `keyby` and `addinby`

Two useful packages to enforce and use the relational model in Stata are `keyby` and `addinby`, both downloadable from SSC. The program `keyby` is an extension of `sort` (see [D] **sort**), and sorts the dataset by a list of variables, and then checks that the dataset is keyed by these variables. The `keyby` package also contains a module `keybygen`, which sorts the data by a list of variables, and adds an additional variable, containing the pre–existing order of an observation within its by–group, to complete the key. Both `keyby` and `keybygen` re–order the key variables to be the first in the dataset, unless the user specifies a `noorder` option. The `addinby` package is an extension of `merge` (see [D] **merge**), and adds in extra variables to the dataset in memory from a second dataset, using a list of existing variables as a foreign key, assumed to be the primary key of the second dataset. Both `keyby` and `addinby` fail if the list of variables specified does not key the dataset that it is supposed to key, and also fail if these key variables contain missing values (unless the user specifies a `missing` option), and restore the original dataset in the event of failure (unless the user specifies a `fast` option).

The `keyby` and `addinby` packages are frequently used to produce a keyed resultsset to be stored on disk, before that resultsset is subsetted and/or decoded and/or reshaped and/or supplied with gap rows for input to `listtab`.

3 Simple tables: decode, label, list

Table 1: Volkswagen cars in the `auto` data

Make	*Mileage (mpg)*	*Weight (lbs)*	*Price*
VW Dasher	23	2,160	$7,140
VW Diesel	41	2,040	$5,397
VW Rabbit	25	1,930	$4,697
VW Scirocco	25	1,990	$6,850

Our first `listtab` examples are the creation of Tables 1 and 2. Table 1, a minimal example, is a modified version of Table 1 of Newson (2003), with 1 row for each Volkswagen car model in the `auto` data. Table 2, a more typical example, has 1 row per polymorphism in the example resultsset, labelled by the polymorphism group and the name of the individual polymorphism, and data on the total number of child subjects, the numbers of subjects with genotypes GG, Gg and gg (where G is the first allele and g is the second allele), and the estimates, confidence limits and p–values for the geometric mean homozygote/heterozygote ratios.

In both these cases, we start with a dataset (the `auto` dataset for Table 1 and the example resultsset for Table 2), and simply decode, label and list the variables we want to tabulate. The tools used to do this are `sdecode` (to decode the numbers and label the rows), `chardef` (to label the columns), and `listtab` (including the `listtab_vars` module) to output the table, as a LaTeX `tabular` environment, to the Stata log and/or the Results window and/or an output file, from which it was cut and pasted into the LaTeX version of this article.

3.1 Numeric to string conversion using `sdecode`

The decoding, and row labelling, is done by the `sdecode` package, which is a "super" version of `decode` (see [D] **decode**), and has a companion package `sencode`, a "super" version of `encode` (see [D] **encode**). Both packages are downloadable from SSC. The `sencode` package converts a string variable to a value–labelled numeric variable, and is used extensively to produce graphs, as a string variable must usually be converted to numeric before it can be used as an axis variable on a graph. By contrast, the `sdecode` package convertes a numeric variable (value–labelled or unlabelled) to a string variable, and is used extensively in producing tables, because numeric variables in tables are usually formatted to a specified number of decimal places or significant figures, and often are formatted to scientific notation with superscripts, and may have added parentheses (if they are confidence limits) or stars (if they are p–values), and may also have aditional prefixes or suffices, indicating justification (left, right or centre) and/or fonts (bold and/or italic and/or non-standard size).

To cater for all these possibilities, `sdecode` has some extra options not available for `decode`:

Table 2: Child genotype frequencies and homozygote/heterozygote ratios

Polymorphism (by group)	*N*	*GG*	*Gg*	*gg*	*Ratio*	*(95%*	*CI)*	*P*
AHR, rs2066853	8704	6811	1788	105	0.47	(0.43,	0.53)	.31
CYP1A1, rs1048943	8764	8205	544	15	0.64	(0.49,	0.84)	.062
CYP1A1, rs4646903	8587	6899	1567	121	0.58	(0.53,	0.65)	.0033
CYP2A6, rs1801272	8666	8203	454	9	0.60	(0.43,	0.84)	.3
CYP2A6, rs28399433	8691	7716	936	39	0.59	(0.49,	0.69)	.067
GCL, rs17883901	9389	7925	1401	63	0.50	(0.44,	0.58)	.9
GPX, rs713041	8611	2733	4168	1710	0.52	(0.50,	0.54)	.093
GSTM1, GSTM1 CNV	8399	585	3163	4651	0.52	(0.49,	0.55)	.14
GSTP1, rs947894	8692	3695	3937	1060	0.50	(0.48,	0.53)	.82
GSTT1, GSTT1 CNV	7591	2448	3653	1490	0.52	(0.50,	0.55)	.056
IGF, rs35767	9383	6579	2555	249	0.50	(0.47,	0.54)	.96
IGF, rs2854744	8489	2456	4202	1831	0.50	(0.48,	0.53)	.67
LTA, rs909253	8664	3397	4089	1178	0.49	(0.47,	0.51)	.34
MIF, rs755622	9500	6454	2757	289	0.50	(0.46,	0.53)	.79
MTNR1B, rs10830963	8584	4570	3357	657	0.52	(0.49,	0.54)	.24
Nrf2, rs1806649	8606	4889	3199	518	0.50	(0.47,	0.53)	.86
Nrf2, rs1962142	8763	7154	1528	81	0.50	(0.44,	0.56)	.95
Nrf2, rs2364723	8725	4047	3760	918	0.51	(0.49,	0.54)	.31
Nrf2, rs6706649	8731	6696	1920	115	0.46	(0.41,	0.51)	.086
Nrf2, rs6721961	9419	7534	1775	110	0.51	(0.46,	0.57)	.64
Nrf2, rs6726395	8692	2580	4323	1789	0.50	(0.48,	0.52)	.78
Nrf2, rs10183914	8671	3617	3975	1079	0.50	(0.47,	0.52)	.8
TNF, rs1800629	8685	5679	2699	307	0.49	(0.46,	0.52)	.53
UGB, rs3741240	8671	3602	4031	1038	0.48	(0.46,	0.50)	.079

1. The user has the choice of using a `generate()` option, specifying that a new string output variable will be generated, or using a `replace` option, specifying that the string output variable will replace the numeric input variable, inheriting its position in the dataset and its characteristics (see [P] **char**).

2. Unlabelled numeric values can be converted to string using formats, possibly specified using a `format()` option, as with the `tostring` command (see [D] **destring**).

3. The `esub()` option allows the user to convert format–generated string values containing embedded `e-` or `e+` substrings, indicating scientific notation, to a number of exponentiation formats. These include `esub(texsuper)` (converting exponents to TEX superscripts), `esub(htmlsuper)` (converting exponents to HTML superscripts), and `esub(rtfsuper)` (converting exponents to RTF superscripts).

4. The `ftrim` option alows the user to trim format–generated string values, removing spaces on the left and on the right.

5. The `xmlsub` option replaces output substrings `&`, < and > with the XML escape sequences `&`, `<` and `>`, respectively.

6. The `prefix()` and `suffix()` options allow the user to add a prefix and a suffix, respectively, to the output string values.

These extra options allow numeric variables to be converted to string equivalents suited to a wide variety of document formats. However, to make life easier for users, `sdecode` has been extended further by the provision of the `bmjcip` package, also downloadable from SSC, which is a front end for `sdecode`. `bmjcip` inputs up to 4 numeric variables, which it interprets as a p–value, an estimate with a p–value, an estimate with 2 confidence limits, or an estimate with 2 confidence limits and a p–value, depending on whether the user supplies 1, 2, 3 or 4 input variables, respectively. It decodes these variables, replacing them with their decoded string versions, prefixed and suffixed appropriately.

3.2 Characteristic assignment using `chardef`

As stated previously, users are advised to store column labels for tables in characteristics assigned to the corresponding variables output to `listtab` (see [P] **char**). To reduce the number and complexity of `char define` statements required, the `chardef` package, downloadable from SSC, has been provided to semi–automate this assignment. `chardef` assigns a user–specified characteristic for a list of variables, assigning a list of corresponding values, which may be prefixed and/or suffixed, using the `prefix()` option and/or the `suffix()` option. The characteristics may be cleared using the `charundef` module of the `chardef` package.

The aim of this mass–production (and mass–destruction) of variable characteristics is to enable the user to create header lines in local macros, to be input to the `headlines()` option of `listtab`. The `listtab` package includes a module `listtab_vars`, which inputs a list of variables that will later be output using `listtab`, together with a row style, and outputs a header line to a local macro. This header line contains a list of the variable names, the variable labels, the values of a named variable characteristic, or another attribute of the variables, separated by the `delimiter()` string of the row style, and prefixed and suffixed by the `begin()` and `end()` strings of the row style, respectively. This local macro can then be input to `listtab` as part of the `headlines()` string list option, to provide one or more rows of column labels in the output table.

3.3 The code to create Table 1

Having introduced the packages to be used, we can now present the Stata code that combines them to produce Table 1. If we assume that the `auto` dataset is in the memory, then this code (and its output to the Stata log) is as follows:

```
. preserve
. keep if word(make,1) == "VW"
(70 observations deleted)
. sdecode mpg, replace
```

```
. sdecode weight, replace
. sdecode price, replace prefix("\\$")
. chardef make mpg weight price, char(varname) prefix("\textit{") suffix("}") /
> //
>   values("Make" "Mileage (mpg)" "Weight (lbs)" "Price")
. listtab_vars make mpg weight price, substitute(char varname) rstyle(tabular)
> local(h1)
. listtab make mpg weight price, type rstyle(tabular) ///
>   headlines("\begin{tabular}{rrrr}" "\hline" "`h1´" "\hline") ///
>   footlines("\hline" "\end{tabular}")
\begin{tabular}{rrrr}
\hline
\textit{Make}&\textit{Mileage (mpg)}&\textit{Weight (lbs)}&\textit{Price}\\
\hline
VW Dasher&23&2,160&\$7,140\\
VW Diesel&41&2,040&\$5,397\\
VW Rabbit&25&1,930&\$4,697\\
VW Scirocco&25&1,990&\$6,850\\
\hline
\end{tabular}
. restore
```

We start by using `preserve` to save a copy of the original `auto` dataset, which in this case is not a resultsset. We then discard the data on all non–Volkswagen car models using `keep`, and use `sdecode` to convert the numeric variables `mpg`, `weight` and `price` to string variables, prefixed in the case of `price` by a LaTeX dollar sign. (Note that we do not need to create a row label variable, because the existing variable `make` will play that role, and also the role of a key for the dataset.) We then specify column labels using `chardef`, which assigns the characteristic `varname` of the variables `make`, `mpg`, `weight` and `price` with a list of values specified by the `values()` option, prefixed by `\textit{` and suffixed by `}`, so the column labels will be italicized in a LaTeX environment. Then, we use the `listtab_vars` module of `listtab` to combine the `varname` characteristics of the variables `make`, `mpg`, `weight` and `price` into an output local macro `h1`, containing a header row for the table, using the option `rtf(tabular)` to specify that the header row will be in the LaTeX `tabular` row style, and the option `substitute(char varname)` to specify that the cells in the header row will be taken from the variable characteristic `varname`, instead of from the variable names. After this, we use `listtab`, with a `headlines()` option to specify table headlines including the local macro `h1`, and a `footlines()` options to specify table footlines. The `listtab` command outputs the variables `make`, `mpg`, `weight` and `price` to the Stata log in the form of some alien–looking output, which readers may recognize as a LaTeX `tabular` environment, ready to be cut and pasted into a LaTeX document to produce Table 1. Finally, the `restore` command restores the original `auto` data.

3.4 The code to create Table 2

As a less trivial example, we can now present the Stata code to produce Table 2. If we assume that the example resultsset is in the memory, then this code (and its output to the Stata log) is as follows:

```
. preserve
. keep if source==1
(24 observations deleted)
. keep source polygp poly N _freq* homhet min* max* p
. format homhet min* max* %8.2f
. bmjcip homhet min* max* p, esub(texsuper) prefix($) suffix($)
. foreach Y of var N _freq* {
  2.   sdecode `Y´, replace esub(texsuper) prefix($) suffix($)
  3. }
. sdecode polygp, gene(S_1)
. sdecode poly, gene(S_2)
. gene row=S_1+", "+S_2
. chardef row N _freq* homhet min* max* p, ///
>   char(varname) prefix("\textit{") suffix("}") ///
>   values("Polymorphism (by group)" N GG Gg gg Ratio "(95\%" "CI)" P)
. listtab_vars row N _freq* homhet min* max* p, rstyle(tabular) ///
>   substitute(char varname) local(h1)
. listtab row N _freq* homhet min* max* p using texdemo1_2.tex, ///
>   replace rstyle(tabular) ///
>   headlines("\begin{tabular}{lrrrrrrrl}" "\hline" "`h1´" "\hline") ///
>   footlines("\hline" "\end{tabular}")
. restore
```

We start by using `preserve` to save a copy of the original resultsset (with data on genotype frequencies in children and mothers), and keeping only the subset `source==1`, because the variable `source` is 1 for child results and 2 for maternal results. We then reset the formats of the homozygote/heterozygote ratios, and their confidence limits, to `%8.2f` (implying 2 decimal places), and use `bmjcip` to decode the ratios, and their confidence limits and p–values, from numeric variables to string variables. We then use a `foreach` loop to decode the variables `N`, `_freq0`, `_freq1` and `_freq2` to string variables. For all these decoded variables, we use the options `prefix($)` and `suffix($)` to output the numbers in LaTeX math mode, which was not really necessary, but might have been necessary if there had been any exponentiated numbers (such as very small p–values or astronomical sample numbers). We then label the table rows, using `sdecode` to decode the key variables `polygp` and `poly` to new variables `S_1` and `S_2`, respectively, and combining their values to produce a new row label variable `row`. We then label the table columns, which are the variable `row` and the decoded statistical variables, using `chardef`, which assigns the characteristic `varname` to contain italicized table header cells, in the way earlier demonstrated in the creation of Table 1. We can now use `listtab_vars` to input the table column variables and output a header row to the local macro `h1`, again in the way demonstrated in the creation of Table 1. Having created this header row, we can use `listtab`, which again uses the local macro `h1` as part of its `headlines()` option. The `listtab` command creates a LaTeX `tabular` environment, even more alien–looking than the one for Table 1, and (fortunately) does not type it to the Stata log, but writes it to the file `texdemo1_2.tex`. This file may be cut and pasted into a LaTeX document, or included using `\input`, to produce Table 2. Finally, the `restore` command restores the original resultsset, with the original numeric statistical variables, and observations for child and maternal results, ready to be used to produce

Table 3: Child and maternal homozygote/heterozygote ratios

	Child:				*Maternal:*			
Polymorphism	*N*	*Ratio*	*(95%*	*CI)*	*N*	*Ratio*	*(95%*	*CI)*
AHR, rs2066853	8704	0.47	(0.43,	0.53)	8100	0.57	(0.52,	0.63)
CYP1A1, rs1048943	8764	0.64	(0.49,	0.84)	8130	0.54	(0.38,	0.75)
CYP1A1, rs4646903	8587	0.58	(0.53,	0.65)	7990	0.51	(0.45,	0.57)
CYP2A6, rs1801272	8666	0.60	(0.43,	0.84)	8088	0.61	(0.44,	0.84)
CYP2A6, rs28399433	8691	0.59	(0.49,	0.69)	8055	0.49	(0.41,	0.60)
GCL, rs17883901	9389	0.50	(0.44,	0.58)	5644	0.47	(0.39,	0.56)
GPX, rs713041	8611	0.52	(0.50,	0.54)	8015	0.49	(0.47,	0.51)
GSTM1, GSTM1 CNV	8399	0.52	(0.49,	0.55)	7497	0.51	(0.49,	0.55)
GSTP1, rs947894	8692	0.50	(0.48,	0.53)	8000	0.48	(0.46,	0.50)
GSTT1, GSTT1 CNV	7591	0.52	(0.50,	0.55)	6674	0.51	(0.49,	0.54)
IGF, rs35767	9383	0.50	(0.47,	0.54)	7434	0.53	(0.49,	0.57)
IGF, rs2854744	8489	0.50	(0.48,	0.53)	6215	0.50	(0.47,	0.52)
LTA, rs909253	8664	0.49	(0.47,	0.51)	8073	0.49	(0.46,	0.51)
MIF, rs755622	9500	0.50	(0.46,	0.53)	7397	0.51	(0.47,	0.55)
MTNR1B, rs10830963	8584	0.52	(0.49,	0.54)	7987	0.49	(0.47,	0.52)
Nrf2, rs1806649	8606	0.50	(0.47,	0.53)	8095	0.52	(0.49,	0.55)
Nrf2, rs1962142	8763	0.50	(0.44,	0.56)	8119	0.49	(0.43,	0.56)
Nrf2, rs2364723	8725	0.51	(0.49,	0.54)	8093	0.50	(0.47,	0.52)
Nrf2, rs6706649	8731	0.46	(0.41,	0.51)	8071	0.51	(0.46,	0.56)
Nrf2, rs6721961	9419	0.51	(0.46,	0.57)	5630	0.50	(0.43,	0.57)
Nrf2, rs6726395	8692	0.50	(0.48,	0.52)	8061	0.51	(0.49,	0.53)
Nrf2, rs10183914	8671	0.50	(0.47,	0.52)	8037	0.52	(0.49,	0.54)
TNF, rs1800629	8685	0.49	(0.46,	0.52)	8114	0.51	(0.48,	0.55)
UGB, rs3741240	8671	0.48	(0.46,	0.50)	7952	0.51	(0.49,	0.54)

further resultstables.

4 Parallel tables: decode, label, reshape, list

In our next example, we demonstrate the production of Table 3. Unlike Table 2, Table 3 does not include the individual genotype frequencies or the p–values. However, it does include sample numbers, homozygote/heterozygote ratios and confidence limits both for child genotypes and for maternal genotypes. The child and maternal results are displayed side by side, in the same table rows. This can be done by using the official Stata `reshape` command (see [D] **reshape**), or the SSC package `xrewide`.

4.1 Reshaping tables using `reshape` and `xrewide`

The `reshape` command reshapes a dataset in memory from a long form to a wide form (using its `reshape wide` syntax), or from a wide form to a long form (using its `reshape long` syntax). If we think of tables as Stata datasets, then we can think of using `reshape wide` to convert two versions of Table 2 (the child and maternal versions) to Table 3. Alternatively, we can think of using `reshape long` to convert Table 2 to a table form similar to some of those created by the `esttab` package (Jann (2007)), in which the

sample numbers, ratios, confidence limits and p–values are stacked vertically. (This might be done using an integer–valued `j()` variable named `stat`, with value labels like "N", "Ratio", "95% CI", and "p".)

The package `xrewide` is an extended version of `reshape wide`, which stores additional results in variable characteristics and/or local macros. These additional results can be useful, if the dataset is output to a table using `listtab`. To store these results, `xrewide` has the following options, in addition to those available with `reshape wide`:

1. The `cjvalue()` and `cjlabel()` options specify the names of variable characteristics, to be assigned to the generated reshaped output variables, and to contain the corresponding values and value labels, respectively, of the `j()` variable.
2. The `vjvalue()` and `vjlabel()` options specify the names of subsets of the input variables to be reshaped, whose corresponding output variables will be assigned the variable characteristics named by `cjvalue()` and `cjlabel()`, respectively. (By default, all output variables are assigned these characteristics.)
3. The `pjvalue()` and `sjvalue()` options specify a prefix and suffix, respectively, to be added to the variable characteristic named by `cjvalue()`. And there are also `pjlabel()` and `sjlabel()` options, specifying a prefix and a suffix, respectively, for the `cjlabel` variable characteristic.
4. The `xmlsub` option specifies that XML escape sequence substitutions will be performed for the characters `&`, `<` and `>` in the `cjlabel()` variable characteristic.
5. Finally, the `lxjk()` and `lxkj()` options specify the names of local macros, to be assigned with full lists of the generated reshaped output variables. These lists are sorted primarily by the corresponding `j()`–value and secondarily by the corresponding input variable, in the case of the `lxjk()` macro, or sorted primarily by the corresponding input variable and secondarily by the corresponding `j()`–value, in the case of the `lxkj()` macro.

These extra saved results, stored in variable characteristics and/or local macros, can then be passed to `listtab` and `listtab_vars`, to save the user the trouble of manually typing column header rows and/or variable lists. They can save a lot of work if a colleague suddenly decides that 3 columns of confidence intervals were wanted instead of 2, or suddenly decides to relabel the columns.

4.2 The code to create Table 3

This code, and its Stata log output, is as follows:

```
. preserve
. keep source polygp poly N homhet min* max*
. format homhet min* max* %8.2f
. bmjcip homhet min* max*, esub(texsuper) prefix($) suffix($)
```

```
. sdecode N, replace esub(texsuper) prefix($) suffix($)
. sdecode polygp, gene(S_1)
. sdecode poly, gene(S_2)
. gene row=S_1+", "+S_2
. chardef row N homhet min* max*, ///
>   char(varname) prefix("\textit{") suffix("}") ///
>   values("Polymorphism" N Ratio "(95\%" "CI)")
. chardef row N homhet min* max*, ///
>   char(justify) values(l r r r r)
. xrewide N homhet min* max*, i(polygp poly) j(source) ///
>   cjlabel(varname2) vjlabel(N) pjlabel("\textit{") sjlabel(":}") ///
>   lxjk(nonrowvars)
```

Data	long	->	wide
Number of obs.	48	->	24
Number of variables	10	->	13
j variable (2 values)	source	->	(dropped)
xij variables:			
	N	->	N1 N2
	homhet	->	homhet1 homhet2
	min95	->	min951 min952
	max95	->	max951 max952

```
. listtab_vars row `nonrowvars´, rstyle(tabular) ///
>   substitute(char varname) local(h1)
. listtab_vars row `nonrowvars´, rstyle(tabular) ///
>   substitute(char varname2) local(h2)
. listtab_vars row `nonrowvars´, ///
>   substitute(char justify) local(just)
. listtab row `nonrowvars´ using texdemo1_3.tex, replace rstyle(tabular) ///
>   headlines("\begin{tabular}{`just´}" "\hline" "`h2´" "`h1´" "\hline") ///
>   footlines("\hline" "\end{tabular}")
. restore
```

We start, as before, by preserving the input resultsset, keeping only the key variables and the variables to be tabulated. and formatting the estimates and confidence limits to 2 decimal places. We then use `bmjcip`, as before, to decode the estimates and confidence limits (this time without p–values), and use `sdecode` to create the row label variable `row` as before. Note that, at this point, the dataset is keyed by the variables `source`, `polygp` and `poly`, and the 2 observations for each polymorphism with different values of `source` (containing child and maternal statistics respectively) will have the same value of `row`. We then use `chardef` to assign values to 2 variable characteristics, whose names are `varname` and `justify`, containing, respectively, the variable header cell and the variable justification of the variable in the table (`l` for left–justified, `r` for right–justified). We can then use `xrewide` to reshape the dataset to a wide format, keyed by the variables `polygp` and `poly`, and with 2 sets of statistical variables, the first containing the sample numbers, estimates and confidence limits for child genotypes, and the second containing the sample numbers, estimates and confidence limits for maternal genotypes. Note that the characteristics `varname` and `justify`, assigned to the input statistical variables to be reshaped, are inherited automatically by the corresponding generated reshaped output

variables. The output variables `N1` and `N2` also have a second characteristic `varname2`, as specified by the `cjlabel()` option, containing the value labels of the `j()`–variable `source`, prefixed and suffixed by the `pjlabel()` and `sjlabel()` options to be italicized and terminated with colons in LaTeX. The option `lxjk(nonkeyvars)` assigns a local macro `nonkeyvars` with the variable list

```
N1 homhet1 min951 max951 N2 homhet2 min952 max952
```

containing the reshaped statistical variables, in the order in which they are to be output. We then use `listtab_vars`, with the option `rstyle(tabular)`, to create header rows in local macros `h1` and `h2`, containing the variable characteristics `varname` and `varname2`, respectively, and then use `listtab_vars`, with no `rstyle()` option, to create an unprefixed, unsuffixed and unseparated list of `justify` characteristics in the local macro `just`, with the value `lrrrrrrrr`, indicating that the first column of Table 3 will be left–justified and the other columns of Table 3 will be right–justified. The `listtab` command then creates a LaTeX `tabular` environment, and writes it to the file `texdemo1_3.tex`, where it can be copied and pasted, or `\input`, into a table in a LaTeX document, which appears as Table 3. Finally, we restore the original resultsset to the memory.

5 Gapped tables: decode, label, reshape, insert gaps, list

Table 3 can be criticized because too much horizontal space is allocated in the first column to repetitive displays of the polymorphism groups (particularly the group "Nrf2"), so that a small font has to be used. Table 4 might be viewed as an improvement on Table 3, because each polymorphism group starts with a gap row, in which the polymorphism group name is displayed in bold type and left–justified, so it does not need to be repeated in the following rows. The first column therefore needs less horizontal space, allowing the other columns to have more. The gap rows make the table less dense and more clear, at the price of using more vertical space. To insert gap rows into tables, we use the `ingap` package.

5.1 Inserting gap rows using `ingap`

The `ingap` package is downloadable from SSC, and was described briefly in Newson (2003). It is a comprehensive package for inserting user–selected numbers of gap observations into user–selected positions in a dataset or by–group, with user–selected values assigned to the table row variable.

However, most users, most of the time, will probably only want to insert a single gap observation at the beginning of each of a number of by–groups, and to set the row variable, in each of these new observations, to be equal to a pre–existing gap–row string variable, whose value will be constant within each by–group. The syntax for doing this is:

`bysort` *keyvarlist_1 gaprowvarname* `(` *keyvarlist_2* `) : ingap ,`

`rowlabel(`*rowvarname*`) grexpression(`*gaprowvarname*`)`

`neworder(`*newordvarname*`)`

where *keyvarlist_1* is a list of key variables defining the by–groups which will each have a single gap observation added at the beginning, *gaprowvarname* is the name of the pre–existing gap–row string variable, *keyvarlist_2* is a list of key variables defining the order of pre–existing observations within the by–group, *rowvarname* is the name of the pre–existing string row label variable, and *newordvarname* is the name of a new variable to be generated by `ingap`, containing, in each observation, the sequential order of that observation within the by–group, once the gap observations have been added. The dataset is usually keyed by the variables *keyvarlist_1* and *keyvarlist_2* before the execution of `ingap`, and is keyed by *keyvarlist_1*, *gaprowvarname* and *newordvarname* after the execution of `ingap`. Therefore, the variable *newordvarname* will be 1 for the gap observation in each by–group defined by the *keyvarlist_1*, and have integer values starting from 2 in the remaining observations in each by–group.

The gap–row variable specified by *gaprowvarname* is defined, usually using `sdecode`, as a function of the variables in *keyvarlist_1*, and will therefore be constant within each by–group.

For instance, in our example resultsset, once the decoding, labelling and reshaping have been done, and a gap row variable `gaprow` has been defined as a function of the polymorphism group variable `polygp`, the command

```
. bysort polygp gaprow (poly): ingap, rowlabel(row) grexpression(gaprow) ///
>   neworder(rowseq)
```

will insert a gap row at the beginning of each by–group defined by `polygp`, and will set the value of the string variable `row` in that gap row to be equal to the value of `gaprow` in that by–group, and will create a new integer–valued variable `rowseq`, equal to 1 in the gap observation and to integers starting at 2 in the other observations in the by–group, in the ascending order defined by the variable `poly`. The dataset will then be keyed by `polygp`, `gaprow` and `rowseq`.

5.2 The code to create Table 4

This code to do this in the example resultsset, and its Stata log, is as follows:

```
. preserve
. keep source polygp poly N homhet min* max*
. format homhet min* max* %8.2f
. bmjcip homhet min* max*, esub(texsuper) prefix($) suffix($)
. sdecode N, replace esub(texsuper) prefix($) suffix($)
. sdecode poly, gene(row) prefix("\hfill ")
. chardef row N homhet min* max*, ///
>   char(varname) prefix("\textit{") suffix("}") ///
>   values("Polymorphism (by group)" N Ratio "(95\%" "CI)")
```

```
. chardef row N homhet min* max*, ///
>   char(justify) values("p{1in}" r r r r)
. xrewide N homhet min* max*, i(polygp poly) j(source) ///
>   cjlabel(varname2) vjlabel(N) pjlabel("\textit{") sjlabel(":}") ///
>   lxjk(nonrowvars)
Data                                long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                        48   ->      24
Number of variables                    8   ->      11
j variable (2 values)             source   ->   (dropped)
xij variables:
                                       N   ->   N1 N2
                                  homhet   ->   homhet1 homhet2
                                   min95   ->   min951 min952
                                   max95   ->   max951 max952
-----------------------------------------------------------------------------
. sdecode polygp, gene(gaprow) prefix("\textbf{") suffix(":}\hfill")
. bysort polygp gaprow (poly): ingap, rowlabel(row) grexpression(gaprow) ///
>   neworder(rowseq)
. listtab_vars row `nonrowvars´, rstyle(tabular) ///
>   substitute(char varname) local(h1)
. listtab_vars row `nonrowvars´, rstyle(tabular) ///
>   substitute(char varname2) local(h2)
. listtab_vars row `nonrowvars´, ///
>   substitute(char justify) local(just)
. listtab row `nonrowvars´ using texdemo1_4.tex, replace rstyle(tabular) ///
>   headlines("\begin{tabular}{`just´}" "\hline" "`h2´" "`h1´" "\hline") ///
>   footlines("\hline" "\end{tabular}")
. restore
```

Once again, we start by preserving the original resultsset, and proceed as for Table 3 up to `xrewide`, except that the row label variable `row` now depends only on `poly` and not on `polygp`, and the characteristic `justify`, for this row variable, is set to `p{1in}`, implying that this row variable will be neither left–justified nor right–justified, but will have a fixed width of 1 inch in the table, with justification depending on the value of the variable `row`. In the original variable `row`, before the addition of the gap observations, the values are all prefixed with `\hfill` , which causes the values to be right–justified. Once `xrewide` has executed, producing a wide dataset keyed by `polygp` and `poly` as before, we use `sdecode` to produce the gap row variable `gaprow`, containing the decoded value of `polygp`, prefixed by `\textbf{` and suffixed by `:}\hfill`, which implies a bold font, appended colons, and left–justification. We then execute `ingap`, adding the gap observations. After that, we can use `listtab_vars` 3 times as before, use `listtab` once as before (outputting the `tabular` environment to the file `texdemo1_4.tex`), and restore the original resultsset. The `tabular` environment can then be pasted or `\input` into a LaTeX document to form Table 4.

6 Creation of RTF documents using `rtfutil`

`listtab` can create tables in many formats other than LaTeX. In particular, it can be used with the SSC package `rtfutil` to create tables in Microsoft Rich Text Format (RTF), described in plain language by Burke (2003).

RTF tables are less simple to produce than LaTeX tables, for 2 reasons:

1. It is not usually a good idea to cut and paste a RTF table into a RTF document using a text editor (although it can be done). This is because RTF documents (especially those produced using Microsoft Word) may contain a lot of incomprehensible code, making it difficult to guess where the table should be pasted. If it is pasted in the wrong place, then Microsoft Word will often fail uninformatively when it tries to open the document.
2. There is no unique RTF row style, defined by `begin()`, `delimiter()`, `end()` and `missnum()` options for `listtab`. This is because, in RTF tables, the number of cells per row, their widths, and other formatting features are defined separately for each row, as part of the `begin()` string and/or the `end()` string.

The first of these problems is solved using the `handle()` option of `listtab`, which allows `listtab` to output to an existing open file, with a file handle, as created by the `file` utility of Stata (see [P] **file**). The `rtfutil` package has a module `rtfopen`, which opens a RTF file using `file open` and outputs the beginning of a RTF document, and a module `rtfclose`, which outputs the end of a RTF document and closes the file using `file close`. The `listtab` commands to create tables can then be inserted between the `rtfopen` and `rtfclose` commands.

The second of these problems is solved using the `rtfrstyle` module of `rtfutil`, which creates RTF row styles. It inputs a list of variables to be output by `listtab`, with their column widths and other formatting information, and outputs to a list of local macros specified in a `local()` option, which will contain the `begin()`, `delimiter()`, `end()`, and possibly `missnum()` strings that will later be input to `listtab`. Column widths are specified using the `cwidths()` option of `rtfrstyle`, and are expressed in twips (1440 twips = 1 inch), which are the units used internally by RTF documents.

It is a good idea for all commands between a `rtfopen` command and a `rtfclose` command to be enclosed in a `capture noisily` block. This ensures that, if any of these commands fails, then the RTF file will automatically be closed, and will be available for inspection by the user.

6.1 The code to create a RTF version of Table 4

In this example, we start with the example resultsset, and create a RTF document `rtfdemo1.rtf`, containing a RTF version of Table 4. The code is enclosed in a document–generating block, starting with a `tempname` command to create a file handle name followed by a `rtfopen` command followed by the beginning of a `capture noisily` block,

and ending with a `rtfclose` command preceded by the end of the `capture noisily` block.

```
. *
>  Beginning of document-generating block
> *;
. tempname rtfb1;
. rtfopen `rtfb1´ using "rtfdemo1.rtf", replace
>   margins(1000 1000 1000 1000) template(fnmono1);
. capture noisily {;
. *
>  Create table
> *;
. preserve;
. keep source polygp poly N homhet min* max*;
. format homhet min* max* %8.2f;
. bmjcip homhet min* max*, esub(rtfsuper)
>   pref("\qr{") suff("}");
. sdecode N, replace esub(rtfsuper) pref("\qr{") suff("}");
. sdecode poly, gene(row) pref("\qr{") suff("}");
. chardef row N homhet min* max*, char(varname) pref("\qr{\i ") suff("}")
>   val("Polymorphism (by group)" N Ratio "(95%" "CI)");
. xrewide N homhet min* max*, i(polygp poly row) j(source)
>   cjlab(varname2) vjlab(N) pjlab("\ql{\i ") sjlab(" DNA:}") lxjk(nonrow);
Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       48   ->     24
Number of variables                   8   ->     11
j variable (2 values)            source   ->   (dropped)
xij variables:
                                      N   ->   N1 N2
                                 homhet   ->   homhet1 homhet2
                                  min95   ->   min951 min952
                                  max95   ->   max951 max952
-----------------------------------------------------------------------------
. sdecode polygp, gene(gaprow) pref("\ql{\b ") suff(":}");
. bysort polygp gaprow (poly): ingap, row(row) grex(gaprow) neword(rowseq);
. rtfrstyle row `nonrow´, cwidths(1500 1000) local(b d e);
. listtab_vars row `nonrow´, sub(char varname) b(`b´) d(`d´) e(`e´) local(h1);
. listtab_vars row `nonrow´, sub(char varname2) b(`b´) d(`d´) e(`e´) local(h2);
. file write `rtfb1´
>   "{\pard\b"
>   " Geometric mean homozygote/heterozygote ratios"
>   " for biallelic polymorphisms in child and maternal DNA"
>   "\par}"
>   _n;
. listtab row `nonrow´, handle(`rtfb1´) b(`b´) d(`d´) e(`e´)
>   head("`h2´" "`h1´");
. restore;
. };
. rtfclose `rtfb1´;
. *
>  End of document-generating block
> *;
```

Inside the `capture noisily` block, the commands up to `ingap` are mostly similar to the corresponding commands to create Table 4, except that the prefixes and suffixes are

RTF prefixes and suffixes. The prefix–suffix pair `\ql{` and `}` specifies left–justification, the prefix–suffix pair `\qr{` and `}` specifies right–justification, and both pairs can be used with or without the added `\i` or `\b`, appended to the prefix, to specify an italic or bold font, respectively. The `rtfrstyle` command inputs the variables to be output by `listtab`, together with the option `cwidths(1500 1000)` to specify that the first column will have width 1500 twips and that all other columns will have widths 1000 twips, and the `local()` option to specify that the `begin()`, `delimiter()` and `end()` strings will be stored in the local macros `b`, `d` and `e`, respectively. These local macros are used by `listtab_vars` to create the header rows, stored in the local macros `h1` and `h2`, and by `listtab` to output the table to the RTF file, after the RTF table heading has been output using a `file write` command. The RTF file produced, with name `rtfdemo1.rtf`, is part of the supplemental online material for this article, and can be downloaded using Stata.

7 Possible extensions

The team of packages `listtab`, `sdecode`, `chardef`, `xrewide` and `ingap` can be used with document formats other than TEX and RTF, such as HTML, and possibly XML–based formats yet to be invented. The packages are good at enforcing the nested block structure commonly used in such formats, because each `prefix()` option of `sdecode` or `chardef` can have its `suffix()` option, each `pjlabel()` option of `xrewide` can have its `sjlabel()` option, each `begin()` option of `listtab` can have its `end()` option, and each `headlines()` option of `listtab` can have its `footlines()` option. And each `rtfopen` command (followed by the beginning of a `capture noisily` block) can have its `rtfclose` command (preceded by the end of the same `capture noisily` block).

It is possible to produce parallel column groups nested in larger parallel column groups, using multiple calls to `xrewide`. Similarly, it is possible to produce gap–prefixed by–groups nested in larger gap–prefixed by–groups, using multiple calls to `ingap`. The smaller by–groups might be preceded by gap row labels in a smaller bold font, while the larger by–groups might be preceded by gap row labels in a larger bold font.

It is also possible to produce multi–page tables in a document, by putting a `listtab` command with an `if` qualifier inside a program loop. Alternatively, we can put the whole set of commands from `preserve` to `restore` inside a program loop, including a `keep if` command to select the subset of observations for a page.

It should also be possible to write user–friendly front–end packages, using the low–level programming toolkit described here, to produce standard documents, in particular formats, containing standard–format tables. So far, I have not done this, because my colleagues and collaborators seem to prefer highly–customized RTF resultstables to standard–format RTF resultstables.

Finally, a Stata program that inputs a resultsset to output resultstables to TEX, RTF and other documents can also output resultsplots of the same results to the same documents, produced using Stata graphics commands (see [G] **graph**). In particular, the

`rtfutil` package includes a `rtflink` module to include graphical output files, such as Encapsulated PostScript (EPS) files produced using `graph export`, as linked objects in a RTF document. The user may then convert these linked objects to embedded objects using Microsoft Word. I find that this possibility is a major advantage of a resultsset–based solution over the purely table–based solutions used by `estout`, `outreg`, `outreg2` and `tabout`.

8 Acknowledgements

I would like to thank Ben Jann of the Eidgenössiche Technische Hochschule (ETH), Zurich, Switzerland and Adam Jacobs of Dianthus Medical Limited, London, UK for their presentations at the 2009 UK Stata User Meeting, which inspired a lot of the packages and updates presented in this article. I would also like to thank Philippe Van Kerm of CEPS/INSTEAD, Luxemburg, for suggesting that a `handle()` option for `listtex` (predecessor to `listtab`) would be a good idea, Ian White of the MRC Biostatistics Unit, Cambridge, UK, for suggesting that something like the `listtab_vars` module of `listtab` would be a good idea, and Lindsey Peters of the United Kingdom Health Protection Agency (HPA) for some very helpful feedback about RTF table rows output using the `rtfutil` package. I would also like to thank Nicholas J. Cox, of Durham University, UK, for coining the term resultsset to describe a Stata dataset of results.

We would also like to thank our collaborators in the ALSPAC Study Team (Institute of Child Health, University of Bristol, Bristol, UK) for allowing the use of their data in this paper. In particular, I would like to thank my ALSPAC collaborators John Holloway, John Henderson and Seif Shaheen for helping me to define the gene labels in Tables 2 to 4 of this article. The whole ALSPAC Study Team comprises interviewers, computer technicians, laboratory technicians, clerical workers, research scientists, volunteers and managers who continue to make the study possible. The ALSPAC study could not have been undertaken without the co–operation and support of the mothers and midwives who took part, or the financial support of the Medical Research Council, the Department of Health, the Department of the Environment, the Wellcome Trust and other funders. The ALSPAC study is part of the WHO–initiated European Longitudinal Study of Pregnancy and Childhood (ELSPAC). My own work at Imperial College London is financed by the UK Department of Health.

9 References

Burke, S. M. 2003. *RTF Pocket Guide*. Beijing: O'Reilly.

Date, C. J. 1986. *An Introduction to Database Systems. Volume 1*. 4th ed. Reading, MA: Addison–Wesley.

Henderson, A. J., R. B. Newson, M. Rose-Zerelli, S. M. Ring, J. W. Holloway, and S. O. Shaheen. 2010. Maternal Nrf2 and gluthathione–S–transferase polymorphisms

do not modify associations of prenatal tobacco smoke exposure with asthma and lung function in school–aged children. *Thorax* 65: 897–902.

Jacobs, A. 2009. Improving the output capabilities of Stata with Open Document Format XML. 15th UK Stata Users Group meeting proceedings. http://www.stata.com/meeting/uk09/abstracts.html.

Jann, B. 2007. Making regression tables simplified. *Stata Journal* 7(2): 227–244.

———. 2009. Recent developments in output processing. 15th UK Stata Users Group meeting proceedings. http://www.stata.com/meeting/uk09/abstracts.html.

Lamport, L. 1994. *LaTeX: A Document Preparation System*. 2nd ed. Reading, MA: Addison–Wesley.

Newson, R. 2003. Confidence intervals and *p*–values for delivery to the end user. *Stata Journal* 3(3): 245–269.

Newson, R. B. 2008a. Asymptotic distributions of linear combinations of logs of multinomial parameter estimates.
http://www.imperial.ac.uk/nhli/r.newson/papers.htm.

———. 2008b. `parmest` and extensions. 14th UK Stata Users Group meeting proceedings. http://www.stata.com/meeting/uk08/abstracts.html.

Shaheen, S. O., R. B. Newson, S. M. Ring, M. J. Rose-Zerilli, J. W. Holloway, and A. J. Henderson. 2010. Prenatal and infant acetaminophen exposure, antioxidant gene polymorphisms, and childhood asthma. *Journal of Allergy and Clinical Immunology* 128(6): 1141–1148.

About the author

Roger B. Newson is a Lecturer in Medical Statistics at the National Heart and Lung Institute, Imperial College London, UK, working principally in asthma research. He has written a number of Stata packages, which are frequently used together, and which define a dialect of the Stata language. These packages can all be installed as a group, in versions compatible with Stata Versions 9, 10, 11 or 12, using Stata do–files (one for each Stata version), which users can download from
http://www.imperial.ac.uk/nhli/r.newson/stata.htm.

Table 4: Child and maternal homozygote/heterozygote ratios (with gaps)

Polymorphism (by group)	*Child: N*	*Ratio*	*(95% CI)*		*Maternal: N*	*Ratio*	*(95% CI)*	
AHR:								
rs2066853	8704	0.47	(0.43,	0.53)	8100	0.57	(0.52,	0.63)
CYP1A1:								
rs1048943	8764	0.64	(0.49,	0.84)	8130	0.54	(0.38,	0.75)
rs4646903	8587	0.58	(0.53,	0.65)	7990	0.51	(0.45,	0.57)
CYP2A6:								
rs1801272	8666	0.60	(0.43,	0.84)	8088	0.61	(0.44,	0.84)
rs28399433	8691	0.59	(0.49,	0.69)	8055	0.49	(0.41,	0.60)
GCL:								
rs17883901	9389	0.50	(0.44,	0.58)	5644	0.47	(0.39,	0.56)
GPX:								
rs713041	8611	0.52	(0.50,	0.54)	8015	0.49	(0.47,	0.51)
GSTM1:								
GSTM1 CNV	8399	0.52	(0.49,	0.55)	7497	0.51	(0.49,	0.55)
GSTP1:								
rs947894	8692	0.50	(0.48,	0.53)	8000	0.48	(0.46,	0.50)
GSTT1:								
GSTT1 CNV	7591	0.52	(0.50,	0.55)	6674	0.51	(0.49,	0.54)
IGF:								
rs35767	9383	0.50	(0.47,	0.54)	7434	0.53	(0.49,	0.57)
rs2854744	8489	0.50	(0.48,	0.53)	6215	0.50	(0.47,	0.52)
LTA:								
rs909253	8664	0.49	(0.47,	0.51)	8073	0.49	(0.46,	0.51)
MIF:								
rs755622	9500	0.50	(0.46,	0.53)	7397	0.51	(0.47,	0.55)
MTNR1B:								
rs10830963	8584	0.52	(0.49,	0.54)	7987	0.49	(0.47,	0.52)
Nrf2:								
rs1806649	8606	0.50	(0.47,	0.53)	8095	0.52	(0.49,	0.55)
rs1962142	8763	0.50	(0.44,	0.56)	8119	0.49	(0.43,	0.56)
rs2364723	8725	0.51	(0.49,	0.54)	8093	0.50	(0.47,	0.52)
rs6706649	8731	0.46	(0.41,	0.51)	8071	0.51	(0.46,	0.56)
rs6721961	9419	0.51	(0.46,	0.57)	5630	0.50	(0.43,	0.57)
rs6726395	8692	0.50	(0.48,	0.52)	8061	0.51	(0.49,	0.53)
rs10183914	8671	0.50	(0.47,	0.52)	8037	0.52	(0.49,	0.54)
TNF:								
rs1800629	8685	0.49	(0.46,	0.52)	8114	0.51	(0.48,	0.55)
UGB:								
rs3741240	8671	0.48	(0.46,	0.50)	7952	0.51	(0.49,	0.54)