The Stata Journal (2008)
**8**, Number 4, pp. 557–568

# Speaking Stata: Distinct observations

Nicholas J. Cox
Department of Geography
Durham University
Durham City, UK
n.j.cox@durham.ac.uk

Gary M. Longton
Fred Hutchinson Cancer Research Center
Seattle, WA
glongton@fhcrc.org

**Abstract.** Distinct observations are those different with respect to one or more variables, considered either individually or jointly. Distinctness is thus a key aspect of the similarity or difference of observations. It is sometimes confounded with uniqueness. Counting the number of distinct observations may be required at any point from initial data cleaning or checking to subsequent statistical analysis. We review how far existing commands in official Stata offer solutions to this issue, and we show how to answer questions about distinct observations from first principles by using the `by` prefix and the `egen` command. The new `distinct` command is offered as a convenience tool.

**Keywords:** dm0042, distinct, by, egen, distinctness, uniqueness, data management

## 1 Introduction

A question common in data management is how many distinct observations there are in a dataset. Distinctness of observations can be determined with respect to either individual variables or several variables considered jointly. This question may arise at various stages in a project. Initial data checking or cleaning may include counting how many of the possible categories in a variable are represented in the data, or seeing how spiky a distribution is, say, because of rounding or digit preference. The number of distinct values may be a natural descriptor: How many different drugs have been prescribed to a patient? How many different products have been bought by a customer? How many different countries are trading partners of another? In many modeling exercises, researchers may need to determine the number of covariate classes or patterns present, or they may want to look more closely at the data given some puzzling results.

The question can be asked using different terminology, so the first task of this column is to discuss the varying language that researchers might use to ask this or related questions. Then we will consider how far the question is answered by existing Stata commands, outline how to answer it yourself from first principles, and introduce the `distinct` command as a convenience tool providing a direct answer.

## 2 Distinctness, duplication, and uniqueness

To Stata, the term *observation* has a very precise meaning: your dataset consists of observations and variables. Privately, say, because of previous training or exposure

 

to other software, you may continue to think of observations as rows, records, cases, subjects, objects, individuals, or in yet other ways, and of variables as columns, fields, attributes, properties, features, or in yet other ways. That is no problem so long as you realize that Stata makes no formal use of any of those terms for describing datasets. (Clearly, the terms rows and columns are used to describe matrices and tables.)

Distinctness, duplication, and uniqueness are different aspects of the similarity and difference of observations.

Suppose the values of some numeric variable are 1, 2, 2, 3, 3, 3, 4, 4, 4, 4. Then there are four *distinct values*: 1, 2, 3, and 4. Alternatively, there are, so far as this variable is concerned, four distinct observations because, for example, the second and third observations both containing the value 2 are identical in respect to this variable. Of these values, 2, 3, and 4 are duplicated in the data, meaning that each occurs twice or more.

The word *duplicates* taken literally implies occurring just twice, but in data management the original meaning has long been swept aside. There is a case on purely linguistic grounds for using an alternative term such as *replicates*, but here is an even stronger case against that: *replicates* already has an established meaning in the design of experiments, and in any case existing usage of duplicates seems so strongly entrenched that suggesting another term would be futile.

A more common cause of minor misunderstandings is that some people refer to the distinct values as *unique values*. Dictionaries, style guides, and usage guides typically insist that *unique* means occurring just once. They insist in vain because many people persist in using the word in weaker senses, including here with *unique* meaning distinctive. Harrell (2001) is just one example of a very careful writer on statistics who uses *unique* in this way. Indeed, as we shall see, such usage is also to be found in official Stata. Thus, in the example just given, 1, 2, 3, and 4 would be reported by many as the unique values of the variable in question, even though such distinct values could all be repeated in the data—and, in particular, 2, 3, and 4 are indeed repeated.

One logic behind that terminology is that if you remove all the duplicates from these data, then you are left with four distinct values, each of which occurs once. The Unix utility `uniq` does precisely that for lines in text files (see, e.g., Robbins and Beebe [2005]). Its widespread use may have contributed to the popularity of *unique*, meaning distinctive, among computer users.

If the battle over *unique* has long been lost, then *singleton* remains as a convenient term for values that really do occur once and only once.

Now consider distinctness determined jointly for two variables. Suppose the observations are 1 and `"a"`, 2 and `"b"`, 2 and `"b"`, 3 and `"c"`, 3 and `"c"`, 3 and `"d"`, 4 and `"c"`, 4 and `"d"`, and 4 and `"d"`. Then, as far as these two variables are concerned, there are six distinct observations: 1 and `"a"`, 2 and `"b"`, 3 and `"c"`, 3 and `"d"`, 4 and `"c"`, and 4 and `"d"`. Considering the variables individually, there are four distinct values for the first variable and four for the second. Clearly, the same principles of considering variables individually and jointly extend to three or more variables.

# 3 Official Stata commands

How could official Stata commands be used to report on the distinctness of observations or related questions?

For concreteness, let us imagine, as above, a dataset with one numeric variable, `n`, and one string variable, `s`, which might be entered as follows:

```
. input n str1 s
            n        s
 1. 1   "a"
 2. 2   "b"
 3. 2   "b"
 4. 3   "c"
 5. 3   "c"
 6. 3   "d"
 7. 4   "c"
 8. 4   "d"
 9. 4   "d"
10. end
```

The `contract` command (see [D] **contract**) will reduce such a dataset to a new dataset consisting of distinct observations (as determined for one or more variables) and their frequencies. If that is the form of dataset you seek, `contract` is ideal. But destroying a dataset to find out its structure is an especially drastic method, which will not appeal to those who agree with Gandalf in *The Lord of the Rings* that to break a thing to find out what it is is to leave the path of wisdom. (The precept would rule out much of the natural sciences too, but let that be.)

The `inspect` command (see [D] **inspect**) will report the number of unique values, meaning distinct values, for numeric variables such as `n`—namely, four—and return it to the user as `r(N_unique)`. However, it gives up if that number exceeds 99, and it ignores string variables, such as `s`, by treating them all as missing.

The `codebook` command (see [D] **codebook**) will report four unique values for both `n` and `s`, but the number of unique values is not returned to the user except by display within the Results window.

The `duplicates` command (see [D] **duplicates**) gives various methods for identifying and dealing with duplicates in data. However, its dedication to those tasks leads to the neglect of singletons, except indirectly through a mention by `duplicates report` or by tagging with `duplicates tag`. A dataset could be reduced to singleton observations with `duplicates drop`, but we have already questioned the wisdom of such a destructive tactic.

The `isid` command (see [D] **isid**) will report via error messages that neither variable uniquely identifies the observations.

The `tabulate` command (see [R] **tabulate oneway**) will tabulate the distinct values of a variable and (as deserves to be better known) will leave the number of rows in a table behind in memory as `r(r)`. Thus a quick way to count categories (and do nothing else) is

```
. quietly tabulate n
. display r(r)
4
```

There are, however, limits to the size of tables that `tabulate` will produce. Because the limits depend on Stata flavor and version, see `help limits` for those limits that apply to you.

The `levelsof` command (see [P] **levelsof**) will display a list of the distinct values (levels) of a variable. This command thus introduces yet another term, which will at least be familiar to those well schooled in the design of experiments. The user can save that list to a local macro and then use `display` to count the number of levels on the fly and show it:

```
. levelsof n, local(N)
1 2 3 4
. display `: word count `N´´
4
```

See [P] **macro** if the latter syntax is new to you. However, `levelsof` does not guarantee either to display or to count all distinct levels if their number is very large.

Although graphical commands lie off the main path we are following here, it is worth mentioning that graphical display is often effective at showing spikiness of distributions. In addition to the more obvious bar charts and histogram commands documented in the *Graphics Reference Manual*, we draw attention to [R] **spikeplot**.

In short, although several commands offer partial solutions on the number of distinct observations, they all show clear limitations in one or more ways:

- The number of distinct values or levels that can be handled.
- Whether several variables can be compared without repeating a command several times.
- Whether distinct observations can be determined jointly for two or more variables.
- Whether the information must be extracted from a mass of other output.
- Whether the information is left behind in memory for later use, especially automated use.
- Whether it is necessary to destroy the dataset to find out more about it.
- Whether both numeric and string variables can be handled.

That leaves plenty of scope for other approaches and dedicated commands.

## 4 From first principles

Let's now back up and show how to count the number of distinct observations for ourselves from first principles. The basic tactic is first to sort the data into groups of distinct observations and then to count those groups.

This process requires an understanding of the `by` prefix construct. A self-contained tutorial was given in an earlier column (Cox 2002). We will need the facts that under `by` *varlist*`:`,

- `_n` is defined as the observation number within each distinct group defined by `by` *varlist*`:`. Thus `_n` starts over at 1 each time a new group is encountered.
- `_N` is interpreted as the number of observations within each distinct group defined by `by` *varlist*`:`. It is equally the observation number of the last observation in each such group. (If there are 10 observations in a group, the last is obviously the 10th.)

We also need an understanding that true and false conditions evaluate numerically to 1 and 0, respectively, which is also explained in the tutorial cited above. Thus the expression `_n == 1` is evaluated as 1 whenever `_n` is indeed 1, and it is evaluated as 0 otherwise (i.e., if `_n` is 2, 3, or any higher integer).

Let's illustrate with the simplest example, computing the number of distinct observations on a single variable, by using the `auto` dataset. We `sort` by that variable, and then we tag the first observation within each distinct group. Read in the data and use the variable `rep78`:

```
. sysuse auto, clear
(1978 Automobile Data)
. by rep78, sort: generate nvals = _n == 1
```

The new variable, `nvals`, is 1 whenever a value is first in its group and is 0 otherwise. If this were indeed the problem, we should now just type

```
. count if nvals
    6
```

which would display the count, also returned as `r(N)`. This is a contracted but equivalent version of

```
. count if nvals == 1
    6
```

because the expression `nvals == 1` evaluates to 1 precisely when `nvals` is 1. The usefulness of `count` was detailed in another column (Cox 2007a).

A refinement here, important for programmers or if memory is tight, is to spell out to `generate` that `nvals` should be created as a byte variable.

You may be puzzled by the result of 6, particularly because `tabulate rep78` shows just 5 categories. The mysterious 6th value is missing; more on that later.

It is unimportant here that `rep78` happens to be numeric and even more unimportant that it happens to contain just integer values. `sort` can group all kinds of variables, numeric or string or both, into clusters of distinct observations. We count the number of groups by counting the number of times a new group starts, by counting the number of times an observation is first in its group. Another systematic way to do that is to count the number of times a group finishes, by counting the number of times an observation is last in its group, using the expression `_n == _N` rather than `_n == 1`. Because groups could be as small as one member, those two methods are the only possibilities.

To extend this problem to others, we need another technique. We calculate the running or cumulative sum, and thus count the 1s, because the 0s make no difference, and pick up the last value as our answer:

```
. replace nvals = sum(nvals)
. replace nvals = nvals[_N]
```

The variable `nvals` now contains the number of distinct observations. `display` of any value of `nvals`, say, the first or the last value, shows the number of distinct values.

As already explained, we might want to define distinct observations of a variable with respect to two or more variables. To Stata, this is the same problem. The only trick needed is to change *varlist* fed to `by:`.

Suppose we wish to calculate the number of distinct observations as defined by the combinations of `foreign` and `rep78`:

```
. drop nvals
. by foreign rep78, sort: generate nvals = _n == 1
. replace nvals = sum(nvals)
. replace nvals = nvals[_N]
```

If you do this for the `auto` data, you will find that there are 10 distinct values of `rep78` and `foreign` combined, that is, every possible pair except the pair 1 and Foreign and the pair 2 and Foreign. This can also be shown directly by typing `tabulate foreign rep78, miss`.

Alternatively, we might want to calculate the number of distinct observations of `rep78` separately for each value of `foreign`.

```
. drop nvals
. by foreign rep78, sort: generate nvals = _n == 1
. by foreign: replace nvals = sum(nvals)
. by foreign: replace nvals = nvals[_N]
```

If you do this for the `auto` data, you will find that there are six distinct values of `rep78` when `foreign` is 0 (domestic cars), namely, 1, 2, 3, 4, 5, and `.` (missing); and four distinct values of `rep78` when `foreign` is 1 (foreign cars), namely, 3, 4, 5, and `.` (missing). This result can also be shown directly by typing `by foreign: tabulate rep78, miss`.

Two complications not tackled in detail here are likely to arise in practice. First, as already exemplified, computations of the kind above include missing values, whether `.`, `.a`, `.b`, ..., `.z` for numeric variables or `""` for string variables, just like any other value. It is quite possible that you do not want missing values included, and if so, you need to exclude them from the computation, say, by typing `if !missing(foreign, rep78)`. Second, you might have further restrictions on the computation, to be specified by `if` or `in`.

Readers familiar with `egen` (see [D] **egen**) may know of an essentially equivalent approach. We can tag just one in any group of observations in *varlist* by using `egen, tag()`. Tagging here means giving a value of 1 to one observation in a group of identical observations and 0 to all the others that may exist in that group. The logic is that whenever a group is identical, then for many problems we need to look at only one observation because the others are, indeed, identical. This is useful, for example, in graphing, where there may be no point to plotting exactly the same coordinates again and again.

Thus the number of distinct values of `rep78` can be obtained by typing

```
. egen tag = tag(rep78)
. count if tag
```

and the number of distinct values of `foreign` and `rep78` jointly, by typing

```
. egen tag = tag(foreign rep78)
. count if tag
```

and the number of distinct values of `rep78` within categories of `foreign`, by typing

```
. egen tag = tag(foreign rep78)
. egen nvals = total(tag), by(foreign)
. tabdisp foreign, cell(nvals)
```

For more on `tabdisp`, see [P] **tabdisp** or Cox (2003a). Note, however, that the `tag()` function of `egen` ignores missing values unless the `missing` option is specified. For most problems, that will be the right way around.

## 5 Adding a twist

We have seen that counting distinct observations can be broken down into easy steps. Let's add a twist to the problem, by considering data collected in sequence, and a desire to count the number of distinct values seen so far in the sequence. This situation arises in several contexts. One is that many birdwatchers like to monitor the number of distinct species they have observed during a career or since the beginning of the current year. The example will underline that although distinctness may be of interest from different viewpoints, Stata solutions remain based on the same basic principles.

For example, suppose we have these data:

```
. clear
. input x
            x
 1. 1
 2. 2
 3. 2
 4. 3
 5. 1
 6. 3
 7. 4
 8. 1
 9. 5
10. 3
11. end
```

The number of distinct values increases from 1 at observation 1 (where 1 first occurs), to 2 at observation 2 (where 2 first occurs), to 3 at observation 4 (where 3 first occurs), and so forth.

As in previous problems, the key is to use the `by` prefix. All we need to do is tag the first occurrence of each distinct value and then count those first occurrences in sequence.

The `by` prefix goes hand in hand with sorting. We should keep a record of the current order of observations because we will want to return to it. If the dataset already includes a time, or some other identifier indicating sequence, we can use that. Otherwise, `generate` a variable recording current order:

```
. generate order = _n
```

If your dataset is really big, that should be

```
. generate long order = _n
```

We will sort into groups of `x` and ensure that within those groups the original order of observations is followed. Then we tag the first occurrence of each value of `x`. This process can all be telescoped into one statement:

```
. by x (order), sort: generate first = _n == 1
```

That statement can be thought of as a condensed version of

```
. sort x order
. by x: generate first = _n == 1
```

The sort order is first by `x` and then by `order`. Then, within groups of `x`, the first observation is tagged as 1; all others within the same group are tagged as 0.

After that, we need to `sort` back to the original order. Then we need a cumulative or running sum of `first`, because the number of distinct values seen so far is equal to the number of first occurrences seen so far.

```
. sort order
. replace first = sum(first)
```

`order` has served its purpose, so type

```
. drop order
```

What do we have now?

```
. list x first
```

| | x | first |
|---|---|---|
| 1. | 1 | 1 |
| 2. | 2 | 2 |
| 3. | 2 | 2 |
| 4. | 3 | 3 |
| 5. | 1 | 3 |
| 6. | 3 | 3 |
| 7. | 4 | 4 |
| 8. | 1 | 4 |
| 9. | 5 | 5 |
| 10. | 3 | 5 |

`first` is no longer a good variable name, so you might want to `rename` it, but the problem is solved. A good feature of the solution is that it extends readily to counting the number of distinct combinations of values seen so far, in exactly the same manner as in the previous section: you just change the varlist fed to `by:`.

## 6 Examining distinct observations

Counting distinct observations is all very well, but at some point you will want to look at the values concerned. For individual variables, that should already be clear: use `tabulate` or `levelsof`, as convenient. If either breaks down because you have too many levels, then you can always use `egen, tag()` and `list` the tagged values.

For variables considered jointly, the problem may seem a little harder but yields to similar approaches. After using `egen, tag()` to create a tagged variable, you can list tagged observations to again show the distinct combinations. After using `egen, group() label` to produce a new composite variable, you can use `tabulate` to count those distinct combinations. A convenience command, `groups`, was described in an earlier column (Cox 2003b). See also Cox (2007b) about generating composite categorical variables.

## 7 The distinct command

A new version of the `distinct` command to report the number of distinct observations is published formally with this column. (Earlier versions publicly available and down-

loadable from the Statistical Software Components archive by using the `ssc` command date back to 2002.)

## 7.1 Syntax

`distinct` [ *varlist* ] [ *if* ] [ *in* ] [ `, missing abbrev(#) joint` ]

`by` is allowed; see [U] **11.1.10 Prefix commands**.

## 7.2 Description

The `distinct` command displays the number of distinct observations with respect to the variables in *varlist*. By default, each variable is considered separately so that the number of distinct observations for each variable is reported; the number of distinct observations is the same as the number of distinct values. Optionally, variables can be considered jointly so that the number of distinct groups defined by the values of variables in *varlist* is reported.

By default, missing values are not counted. *varlist* can contain both numeric and string variables.

## 7.3 Options

`missing` specifies that missing values are to be included in counting distinct observations.

`abbrev(#)` specifies that variable names are to be displayed abbreviated to at most # characters. This option has no effect with `joint`.

`joint` specifies that distinctness is to be determined jointly for the variables in *varlist*.

## 7.4 Saved results

`distinct` saves the following in `r()`:

Scalars

| | |
|---|---|
| `r(ndistinct)` | distinct count (for last variable, or jointly considered group of variables, and, if specified, last `by` group) |
| `r(N)` | number of observations (for last variable, or jointly considered group of variables, and, if specified, last `by` group) |

## 7.5 Examples

The `distinct` command is intended simply as a reporting tool. Above all, it does not tackle the creation of variables; the logic of sections 4 and 5 indicates how to do that. But `distinct` does allow examination of several variables (both individually and

jointly), subdivision according to a `by` *varlist*:, and exclusion or inclusion of missing values. Here it is in action with the `auto` data:

```
. sysuse auto, clear
(1978 Automobile Data)
. distinct
                │        Observations
                │     total   distinct
────────────────┼──────────────────────
          make  │        74         74
         price  │        74         74
           mpg  │        74         21
         rep78  │        69          5
      headroom  │        74          8
         trunk  │        74         18
        weight  │        74         64
        length  │        74         47
          turn  │        74         18
  displacement  │        74         31
    gear_ratio  │        74         36
       foreign  │        74          2
. distinct for rep78, joint
        Observations
      total   distinct
         69         8
. by foreign, sort: distinct rep78
──────────────────────────────────────────────────────────────────────────────
-> foreign = Domestic
        │        Observations
        │     total   distinct
────────┼──────────────────────
 rep78  │        48          5
──────────────────────────────────────────────────────────────────────────────
-> foreign = Foreign
        │        Observations
        │     total   distinct
────────┼──────────────────────
 rep78  │        21          3
. by foreign, sort: distinct rep78, missing
──────────────────────────────────────────────────────────────────────────────
-> foreign = Domestic
        │        Observations
        │     total   distinct
────────┼──────────────────────
 rep78  │        52          6
──────────────────────────────────────────────────────────────────────────────
-> foreign = Foreign
        │        Observations
        │     total   distinct
────────┼──────────────────────
 rep78  │        22          4
```

## 8 Conclusions

Questions about counting distinct or unique observations continue to arise on Statalist and at the Stata Users Group meetings. Hence, we have attempted to review the topic concisely yet comprehensively in this column. As is often the case, existing official commands do offer at least partial solutions to the question. An approach from first principles with the `by` prefix and associated tricks or an equivalent approach with the `egen` command are alternatives that can be tuned to both simple and more challenging versions of the problem. Finally, the `distinct` command is offered as another small tool for the data manager's Stata toolbox.

## 9 Acknowledgments

The `distinct` command grew out of one originally posted to Statalist by Patrick Royston for Stata 4. Steven Samuels made helpful comments on an earlier version of one section.

## 10 References

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.

———. 2003a. Speaking Stata: Problems with tables, Part I. *Stata Journal* 3: 309–324.

———. 2003b. Speaking Stata: Problems with tables, Part II. *Stata Journal* 3: 420–439.

———. 2007a. Speaking Stata: Making it count. *Stata Journal* 7: 117–130.

———. 2007b. Stata tip 52: Generating composite categorical variables. *Stata Journal* 7: 582–583.

Harrell Jr., F. E. 2001. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.

Robbins, A., and N. H. F. Beebe. 2005. *Classic Shell Scripting*. Sebastopol, CA: O'Reilly.

#### About the authors

Nicholas Cox is a statistically minded geographer at Durham University. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored 15 commands in official Stata. He wrote several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*.

Gary M. Longton is a biostatistician at the Fred Hutchinson Cancer Research Center in Seattle, Washington. In addition to providing statistical support for investigators at the Center from a variety of disciplines, he has written several Stata programs implementing statistical methods developed by Margaret Pepe for the evaluation of medical screening and diagnostic tests.