Title

datasignature — Determine whether data have changed

Syntax

```
datasignature

datasignature set [, reset]

datasignature confirm [, strict]

datasignature report

datasignature set, saving(filename[, replace]) [reset]

datasignature confirm using filename [, strict]

datasignature report using filename

datasignature clear
```

Menu

Data > Other utilities > Manage data signature

Description

These commands calculate, display, save, and verify checksums of the data, which taken together form what is called a *signature*. An example signature is 162:11(12321):2725060400:4007406597. That signature is a function of the values of the variables and their names, and thus the signature can be used later to determine whether a dataset has changed.

`datasignature` without arguments calculates and displays the signature of the data in memory.

`datasignature set` does the same, and it stores the signature as a characteristic in the dataset. You should `save` the dataset afterward so that the signature becomes a permanent part of the dataset.

`datasignature confirm` verifies that, were the signature recalculated this instant, it would match the one previously set. `datasignature confirm` displays an error message and returns a nonzero return code if the signatures do not match.

`datasignature report` displays a full report comparing the previously set signature to the current one.

In the above, the signature is stored in the dataset and accessed from it. The signature can also be stored in a separate, small file.

`datasignature set, saving(`*filename*`)` calculates and displays the signature and, in addition to storing it as a characteristic in the dataset, also saves the signature in *filename*.

`datasignature confirm using` *filename* verifies that the current signature matches the one stored in *filename*.

`datasignature report using` *filename* displays a full report comparing the current signature with the one stored in *filename*.

In all the above, if *filename* is specified without an extension, `.dtasig` is assumed.

`datasignature clear` clears the signature, if any, stored in the characteristics of the dataset in memory.

Options

`reset` is used with `datasignature set`. It specifies that even though you have previously set a signature, you want to erase the old signature and replace it with the current one.

`strict` is for use with `datasignature confirm`. It specifies that, in addition to requiring that the signatures match, you also wish to require that the variables be in the same order and that no new variables have been added to the dataset. (If any variables were dropped, the signatures would not match.)

`saving(`*filename* [`, replace`]`)` is used with `datasignature set`. It specifies that, in addition to storing the signature in the dataset, you want a copy of the signature saved in a separate file. If *filename* is specified without a suffix, `.dtasig` is assumed. The `replace` suboption allows *filename* to be replaced if it already exists.

Remarks and examples stata.com

Remarks are presented under the following headings:

Using datasignature interactively

`datasignature` is useful in the following cases:

1. You and a coworker, separated by distance, have both received what is claimed to be the same dataset. You wish to verify that it is.

2. You work interactively and realize that you could mistakenly modify your data. You wish to guard against that.

3. You want to give your dataset to an assistant to improve the labels and the like. You wish to verify that the data returned to you are the same data.

4. You work with an important dataset served on a network drive. You wish to verify that others have not changed it.

Example 1: Verification at a distance

You load the data and type

```
. datasignature
  74:12(71728):3831085005:1395876116
```

Your coworker does the same with his or her copy. You compare the two signatures.

Example 2: Protecting yourself from yourself

You load the data and type

```
. datasignature set
  74:12(71728):3831085005:1395876116    (data signature set)
. save, replace
```

From then on, you periodically type

```
. datasignature confirm
  (data unchanged since 19feb2013 14:24)
```

One day, however, you check and see the message:

```
. datasignature confirm
  (data unchanged since 19feb2013 14:24, except 2 variables have been added)
```

You can find out more by typing

```
. datasignature report
  (data signature set on Monday 19feb2013 14:24)
  Data signature summary
   1. Previous data signature     74:12(71728):3831085005:1395876116
   2. Same data signature today   (same as 1)
   3. Full data signature today   74:14(113906):1142538197:2410350265
  Comparison of current data with previously set data signature
      variables                    number  notes
      ------------------------------------------------------------
      original # of variables          12  (values unchanged)
      added variables                   2  (1)
      dropped variables                 0
      ------------------------------------------------------------
      resulting # of variables         14
      (1) Added variables are agesquared logincome.
```

You could now either drop the added variables or decide to incorporate them:

```
. datasignature set
  data signature already set -- specify option reset
r(110)
. datasignature set, reset
  74:14(113906):1142538197:2410350265       (data signature reset)
```

Concerning the detailed report, three data signatures are reported: 1) the stored signature, 2) the signature that would be calculated today on the basis of the same variables in their original order, and 3) the signature that would be calculated today on the basis of all the variables and in their current order.

`datasignature confirm` knew that new variables had been added because signature 1 was equal to signature 2. If some variables had been dropped, however, `datasignature confirm` would not be able to determine whether the remaining variables had changed.

Example 3: Working with assistants

You give your dataset to an assistant to have variable labels and the like added. You wish to verify that the returned data are the same data.

Saving the signature with the dataset is inadequate here. Your assistant, having your dataset, could change both your data and the signature and might even do that in a desire to be helpful. The solution is to save the signature in a separate file that you do not give to your assistant:

```
. datasignature set, saving(mycopy)
  74:12(71728):3831085005:1395876116       (data signature set)
  (file mycopy.dtasig saved)
```

You keep file `mycopy.dtasig`. When your assistant returns the dataset to you, you `use` it and compare the current signature to what you have stored in `mycopy.dtasig`:

```
. datasignature confirm using mycopy
  (data unchanged since 19feb2013 15:05)
```

By the way, the signature is a function of the following:

- The number of observations and number of variables in the data
- The values of the variables
- The names of the variables
- The order in which the variables occur in the dataset
- The storage types of the individual variables

The signature is not a function of variable labels, value labels, notes, and the like.

Example 4: Working with shared data

You work on a dataset served on a network drive, which means that others could change the data. You wish to know whether this occurs.

The solution here is the same as working with an assistant: you save the signature in a separate, private file on your computer,

```
. datasignature set, saving(private)
  74:12(71728):3831085005:1395876116       (data signature set)
  (file private.dtasig saved)
```

and then you periodically check the signature by typing

```
. datasignature confirm using private
  (data unchanged since 15mar2013 11:22)
```

Using datasignature in do-files

`datasignature confirm` aborts with error if the signatures do not match:

```
. datasignature confirm
  data have changed since 19feb2013 15:05
r(9);
```

This means that, if you use `datasignature confirm` in a do-file, execution of the do-file will be stopped if the data have changed.

You may want to specify the `strict` option. `strict` adds two more requirements: that the variables be in the same order and that no new variables have been added. Without `strict`, these are not considered errors:

```
. datasignature confirm
  (data unchanged since 19feb2013 15:22)
. datasignature confirm, strict
  (data unchanged since 19feb2013 15:05, but order of variables has changed)
r(9);
```

and

```
. datasignature confirm
  (data unchanged since 19feb2013 15:22, except 1 variable has been added)
. datasignature confirm, strict
  (data unchanged since 19feb2013 15:22, except 1 variable has been added)
r(9);
```

If you keep logs of your analyses, issuing `datasignature` or `datasignature confirm` immediately after loading each dataset is a good idea. This way, you have a permanent record that you can use for comparison.

Interpreting data signatures

An example signature is `74:12(71728):3831085005:1395876116`. The components are

1. `74`, the number of observations;
2. `12`, the number of variables;
3. `71728`, a checksum function of the variable names and the order in which they occur; and
4. `3831085005` and `1395876116`, checksum functions of the values of the variables, calculated two different ways.

Two signatures are equal only if all their components are equal.

Two different datasets will probably not have the same signature, and it is even more unlikely that datasets containing similar values will have equal signatures. There are two data checksums, but do not read too much into that. If either data checksum changes, even just a little, the data have changed. Whether the change in the checksum is large or small—or in one, the other, or both—signifies nothing.

The logic of data signatures

The components of a data signature are known as checksums. The checksums are many-to-one mappings of the data onto the integers. Let's consider the checksums of `auto.dta` carefully.

The data portion of `auto.dta` contains 38,184 bytes. There are 256^{38184} such datasets or, equivalently, 2^{305472}. The first checksum has 2^{48} possible values, and it can be proven that those values are equally distributed over the 2^{305472} datasets. Thus there are $2^{305472}/2^{48} - 1 = 2^{305424} - 1$ datasets that have the same first checksum value as `auto.dta`. The same can be said for the second checksum. It would be difficult to prove, but we believe that the two checksums are conditionally independent, being based on different bit shifts and bit shuffles of the same data. Of the $2^{305424} - 1$ datasets that have the same first checksum as `auto.dta`, the second checksum should be equally distributed over them. Thus there are about $2^{305376} - 1$ datasets with the same first and second checksums as `auto.dta`.

Now let's consider those $2^{305376} - 1$ other datasets. Most of them look nothing like `auto.dta`. The checksum formulas guarantee that a change of one variable in 1 observation will lead to a change in the calculated result if the value changed is stored in 4 or fewer bytes, and they nearly guarantee it in other cases. When it is not guaranteed, the change cannot be subtle—"Chevrolet" will have to change to binary junk, or a double-precision 1 to −6.476678983751e+301, and so on. The change will be easily detected if you `summarize` your data and just glance at the minimums and maximums. If the data look at all like `auto.dta`, which is unlikely, they will look like a corrupted version.

More interesting are offsetting changes across observations. For instance, can you change one variable in 1 observation and make an offsetting change in another observation so that, taken together, they will go undetected? You can fool one of the checksums, but fooling both of them simultaneously will prove difficult. The basic rule is that the more changes you make, the easier it is to create a dataset with the same checksums as `auto.dta`, but by the time you've done that, the data will look nothing like `auto.dta`.

Stored results

`datasignature` without arguments and `datasignature set` store the following in `r()`:

Macros	
`r(datasignature)`	the signature

`datasignature confirm` stores the following in `r()`:

Scalars	
`r(k_added)`	number of variables added
Macros	
`r(datasignature)`	the signature

`datasignature confirm` aborts execution if the signatures do not match and so then returns nothing except a return code of 9.

`datasignature report` stores the following in `r()`:

Scalars	
`r(datetime)`	`%tc` date–time when set
`r(changed)`	`.` if `r(k_dropped)` $\neq 0$, otherwise `0` if data have not changed, `1` if data have changed
`r(reordered)`	`1` if variables reordered, `0` if not reordered, `.` if `r(k_added)` $\neq 0$ | `r(k_dropped)` $\neq 0$
`r(k_original)`	number of original variables
`r(k_added)`	number of added variables
`r(k_dropped)`	number of dropped variables
Macros	
`r(origdatasignature)`	original signature
`r(curdatasignature)`	current signature on same variables, if it can be calculated
`r(fulldatasignature)`	current full-data signature
`r(varsadded)`	variable names added
`r(varsdropped)`	variable names dropped

`datasignature clear` stores nothing in `r()` but does clear it.

`datasignature set` stores the signature in the following characteristics:

Characteristic	
`_dta[datasignature_si]`	signature
`_dta[datasignature_dt]`	`%tc` date–time when set in `%21x` format
`_dta[datasignature_vl1]`	part 1, original variables
`_dta[datasignature_vl2]`	part 2, original variables, if necessary
etc.	

To access the original variables stored in _dta[datasignature_vl1], etc., from an ado-file, code

```
    mata: ado_fromlchar("vars", _dta", "datasignature_vl")
```

Thereafter, the original variable list would be found in ‘vars’.

Methods and formulas

datasignature is implemented using _datasignature; see [P] **_datasignature**.

Reference

Gould, W. W. 2006. Stata tip 35: Detecting whether data have changed. *Stata Journal* 6: 428–429.

Also see

[P] **_datasignature** — Determine whether data have changed

[P] **signestimationsample** — Determine whether the estimation sample has changed