The Stata Journal (2022)
22, Number 1, pp. 224–230 DOI: 10.1177/1536867X221083928

Stata tip 145: Numbering weeks within months

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

1 This day is in which week of its month?

A user wanted a variable to indicate in which week of its month a daily date fell. That question is a challenge both to imagine different definitions of weeks within months and to produce Stata code for each interpretation. It serves as a reminder that whatever date and time problems have been solved, there are still plenty more. See, for example, Cox (2010, 2012a,b, 2018b, 2019) for some previous notes in this territory.

Even if you lack interest in this specific question, it serves as an example for showing problem-solving skills in using Stata.

In what follows, we go no further than a standard Western calendar in which the months of each year are January to December. Much more on calendars can be found in standard references such as Blackburn and Holford-Strevens (1999) and Reingold and Dershowitz (2018). Unlike years, days, and months, weeks have no physical (meaning, astronomical) basis, but they do have many different associations and implications, ranging from mythological and religious to economic and cultural (for example, Henkin [2021]).

Surprisingly or not, Stata's built-in week functions are unlikely to be part of the answer. Stata's definition of a week is idiosyncratic. Week 1 always starts on 1 January, week 2 always starts on 8 January, and so on with 7-day weeks, until week 52 ends on 31 December and is 8 or 9 days long depending on whether the year is a leap year. There is no week 53 in this scheme. Stata's definition has one distinct advantage, which is that weeks always nest inside years without ever spanning two years. Otherwise, this definition seems rarely used outside Stata, and the rest of this tip is based on the assumption that you are using some different definition of a week.

We will work with a sandbox dataset with daily dates for the first three months of 2021. It is mental arithmetic to check that 2021 was not a leap year so that $90 = 31 + 28 + 31$ observations will give us the right dataset size for those months.


```
. set obs 90
Number of observations (_N) was 0, now 90.
. generate ddate = mdy(12, 31, 2020) + _n
. format ddate %td
. list if inlist(_n, 1, _N)

     +-----------+
     |     ddate |
     |-----------|
  1. | 01jan2021 |
 90. | 31mar2021 |
     +-----------+

. generate mdate = mofd(ddate)
. format mdate %tm
```

For more on `_n` and `_N` if needed, see `help _variables`.

2 Daily and monthly dates introduced

Let me first recount some basics to help readers quite new to handling dates like these. If you are already broadly familiar with dates, you can skip and skim to the next section.

In Stata, dates are held as integers. Stata holds daily and monthly dates as integers, with the origin 0 as the first possible date in 1960. We have already used `mdy()` as a way to get daily dates from month, day, and year components. `daily("31 Dec 2020", "DMY")` is an example of one of several other ways to get started if daily dates arrive as strings. What is `mdy(12, 31, 2020)`? Let's use `display` to find out:

```
. display mdy(12, 31, 2020)
22280
```

It is 22280 on a scale in which 1 January 1960 is 0. Stata really does not expect mental arithmetic from you to work out in reverse that 22280 means 31 December 2020, which leads to the next point.

Date display formats show dates conventionally. It is the job of date display formats to show you what a date means conventionally. A format has nothing to do with what is stored—that remains here integers, like 22280—but, as said, only with what is displayed when you ask for it. Again, `display` shows this for small examples:

```
. display %td 22280
31dec2020
```

`%td` is the default daily date display format. Although a format can be used with `display` of individual values, it is more common, as shown earlier, to use the `format` command to associate variables with display formats.

Special date functions are key. I used the function `mofd()` (think `m`onthly date `of` `d`aily date) to go from daily dates to monthly dates, and then I applied `%tm` as the default monthly date format. To generate monthly date variables in Stata form directly, which is a different problem, I most often use one of two functions. `ym()` assembles a monthly

date from numeric year and month components, just as `mdy()` assembles a daily date from numeric month, day, and year components. `monthly()` is for processing string arguments such as `"Jan 2021"`, much as `daily()` does for daily dates in string form.

```
. display ym(2021, 1)
732
```

shows that, regardless of whatever display format is applied elsewhere, the monthly date for January 2021 is to Stata an integer, 732. Recall the principle that monthly dates too are stored as integers counting from 0, which is the first possible date in 1960, here January 1960.

Changing the date format does not change the date. It is often misunderstood, so let's underline that changing the date display format is never a way to convert from one kind of date to another (Cox 2012c). Short of setting up your own calculation, you need a conversion function such as `mofd()`.

We can now focus on the title problem.

3 Week 1 is days 1 to 7, and so on

The simplest definitions of each week of the month are that week 1 is days 1 to 7, week 2 days 8 to 14, and so on. On these definitions, there are usually 5 weeks in each month, and when that is true, the last week is 1, 2, or 3 days long depending on whether the month is 29, 30, or 31 days long. The exceptions are when the month is February in a nonleap year, so the number of days in the month is then 28, and there are 4 complete weeks.

Extracting day of month from a daily date is a common and fundamental need, so you should expect there to be a function provided to yield this directly. It is `day()`, which yields one of 1 to 31 as a result. We need to map days to week numbers, which could be

```
. generate wofm = ceil(day(ddate)/7)
```

That is it. From the inside outward, we read off `day(ddate)` and then divide by 7. So the result of that will be a decimal equivalent of 1/7, 2/7, 3/7, 4/7, 5/7, 6/7, 7/7, 8/7, and so on, up to 31/7. The `ceil()` function rounds up and is what we need: think "ceiling". It leaves integers unchanged and otherwise yields the next higher integer. That is, 1/7 to 7/7 round up to 1, 8/7 to 14/7 round up to 2, and so forth. `ceil()` and its sibling function `floor()` are often overlooked. I seize every opportunity to riff about them (Cox 2003, 2011, 2018a).

You might want to take that more slowly. You could put the result of `day(ddate)` in a new variable first:

```
. generate day = day(ddate)
```

Then, you could divide and round up:

```
. generate wofm = ceil(day/7)
```

Indeed, the last operation could also be split into two steps. You should certainly rewrite this way if you need a separate variable holding day of the month. It is also a good idea if the code is thereby clearer to you or anyone else needing to understand it.

It is prudent to check results. A programmer can rarely be too careful, but looking at a table is easy.

```
. tabulate mdate wofm
```

	wofm					
mdate	1	2	3	4	5	Total
2021m1	7	7	7	7	3	31
2021m2	7	7	7	7	0	28
2021m3	7	7	7	7	3	31
Total	21	21	21	21	6	90

That is as it should be. There are 4 complete weeks and 3 days left over for January and March (31 days), and there are 4 complete weeks for February (28 days in a nonleap year). An even more careful check would be to see whether the values 1 to 5 occur when they should and whether a leap year example works too.

4 A week starts on a Sunday, and so on

Another class of definitions might include a week starting on Sunday and ending on Saturday or starting on Monday and ending on Sunday. Do not rule out a weird-seeming definition that some group uses somewhere. My own university for many years had a convention that teaching weeks ran Thursday to Wednesday in the first term of the academic year and Monday to Friday in the other terms. The offset was to accommodate welcomes and whatnot at the beginning of each academic year. Admitting different start days may seem dismaying: do we need code for seven cases? Fortunately, one trick covers all.

Stata has a day-of-week function, `dow()`, that is handy here. `dow()` returns 0 for Sundays, 1 for Mondays, and so on, through to 6 for Saturdays. Let's take the case of starting on Sundays.

```
. bysort mdate (ddate): generate WOFM = sum(dow(ddate) == 0)
```

That is the essence. Ensuring that observations are in the right order, we use `bysort:` to calculate separately within each month. The code that follows the colon counts the number of Sundays so far in the month. `dow(date)` is equal to 0 when it is Sunday and not equal to 0 (because it is an integer from 1 to 6) on other days. When `dow(ddate) == 0` is true, then 1 is returned; when it is false, then 0 is returned. `sum()` is a key function that returns the running or cumulative sum of those 1s and 0s. The equivalent function in Mata is called `runningsum()`. In some statistical contexts, cumulative sums

are called "cusums", a term with some advantages and some disadvantages; either way, Stata does not use it generally, but note a distinct `cusum` command with a different goal.

If you are fuzzy about what `bysort:` does precisely, start with its help file or Cox (2002).

This is perhaps the trickiest point of the entire tip, so let us follow through what happens. At the start of each month, `dow(ddate) == 0` will be 1 or 0, and the running sum will be the same. As we look at each successive day in the same month, `dow(ddate) == 0` will alternate between 1s on Sundays and 0s on any other day. Correspondingly, the running sum will be bumped up by 1 on any Sunday and by 0 on any other day, meaning that the running sum will stay the same.

How does that pan out in the sandbox dataset? Again, a cross-tabulation is simple.

```
. tabulate mdate WOFM
```

	WOFM					
mdate	0	1	2	3	4	Total
2021m1	2	7	7	7	7	31
2021m2	6	7	7	7	1	28
2021m3	6	7	7	7	4	31
Total	14	21	21	21	12	90

	WOFM	
mdate	5	Total
2021m1	1	31
2021m2	0	28
2021m3	0	31
Total	1	90

In 2021, the first Sundays in each month were on 3 January, 7 February, and 7 March. Correspondingly, each month started with an incomplete week, which is coded 0, zero values being recorded because no Sundays had been observed so far. As it happens, August 2021 started on a Sunday, so no week 0 would be defined then.

Now twists on the idea should seem easy.

Starting on Monday or any other day of the week just means testing for a different result from `dow()`. If weeks start on Mondays, we are testing whether `dow(ddate) == 1`. So all seven cases for different starts and stops to the week yield to essentially the same trick.

Wanting the numbering of weeks to start at 1 just means adding 1 in calculating `WOFM` whenever the count starts at 0. Indeed, although starting the count at 0 may seem natural to anyone who knows more than a little mathematics, in the outside world of (say) medical or business reporting, it is much more likely that numbering starts at 1.

```
. by mdate: replace WOFM = WOFM + 1 if WOFM[1] == 0
(3 real changes made)
```

Notice the conditioning there. We want to bump up the counter if and only if the counter starts at 0.

5 Small morals

Given the extraordinary range of calendars and calendar practices across countries and subject areas, there could be yet other definitions. Yet we stop there and close with some small but standard morals to be drawn from the tale.

Dates are complicated. The entire rigmarole of dates (to say nothing of times too, but those are not an issue here) is a complicated mess. From your early education onward, you have been exposed repeatedly to the rules you usually need to know, but handling dates in software like Stata poses a new set of challenges. Some software handles dates through one or more distinct data or variable types. Stata's decision that dates and times are just integers simplifies much, but it still implies details about formats and functions that may need mastering. `help datetime` remains the best place to start.

Functions are helpful. The key to success here is knowing your functions, whether they are date functions that are what you need, such as `day()` or `dow()`, or other more general functions that help, such as `sum()` or `ceil()`. Functions remain the unsung heroes of Stata (Cox 2011).

Sandboxes are useful. Often, I witness people who have read large and complicated datasets into memory and are struggling to find the syntax they need. While that is the immediate problem, it can be a good idea to back up and create a small sandbox dataset, as we did here. You can get results quickly and compare easily with whatever you know to be the right answer. Other neglected ideas include using `display` to work with individual values and Mata as a glorified calculator (examples in Cox [2018a, 2019]).

References

Blackburn, B., and L. Holford-Strevens. 1999. *The Oxford Companion to the Year*. Oxford: Oxford University Press.

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102. https://doi.org/10.1177/1536867X0200200106.

———. 2003. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447. https://doi.org/10.1177/1536867X0400300413.

———. 2010. Stata tip 68: Week assumptions. *Stata Journal* 10: 682–685. https://doi.org/10.1177/1536867X1101000409.

———. 2011. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471. https://doi.org/10.1177/1536867X1101100308.

———. 2012a. Stata tip 111: More on working with weeks. *Stata Journal* 12: 565–569. https://doi.org/10.1177/1536867X1201200316.

———. 2012b. Stata tip 111: More on working with weeks, erratum. *Stata Journal* 12: 765. https://doi.org/10.1177/1536867X1201200416.

———. 2012c. Stata tip 113: Changing a variable's format: What it does and does not mean. *Stata Journal* 12: 761–764. https://doi.org/10.1177/1536867X1201200415.

———. 2018a. Speaking Stata: From rounding to binning. *Stata Journal* 18: 741–754. https://doi.org/10.1177/1536867X1801800311.

———. 2018b. Stata tip 130: 106610 and all that: Date variables that need to be fixed. *Stata Journal* 18: 755–757. https://doi.org/10.1177/1536867X1801800312.

———. 2019. Speaking Stata: The last day of the month. *Stata Journal* 19: 719–728. https://doi.org/10.1177/1536867X19874247.

Henkin, D. M. 2021. *The Week: A History of the Unnatural Rhythms That Made Us Who We Are*. New Haven, CT: Yale University Press.

Reingold, N. M., and N. Dershowitz. 2018. *Calendrical Calculations: The Ultimate Edition*. Cambridge: Cambridge University Press.