The Stata Journal (2018)
18, Number 3, pp. 503–516

Graphing each individual's data over time

Mark D. Chatfield
University of Queensland
Brisbane, Australia
m.chatfield@uq.edu.au

Abstract. Graphing each individual's data over time (in separate graphs) can be a worthwhile approach in exploring longitudinal and panel datasets. This especially applies for datasets where several variables change over time and where there are many possible time points, for example, administrative datasets and patient safety profiles in clinical trials. Studying a few individuals' graphs closely can provide insight into the nature and quality of the data, generate hypotheses, and inform data analysis. Selecting a few typical or unusual graphs can make for powerful presentations at meetings. I give examples of graphing a single variable and multiple variables over time for each individual, and I detail associated Stata coding tips and tricks.

Keywords: gr0074, longitudinal data, panel data, time series, graphics, twoway, scatter, putdocx, superimposition, xtline, patient profile, profile plot, developmental trajectory

1 Introduction

A good statistical principle is to familiarize yourself with and explore a new dataset before doing any analysis. Graphing raw data can be a useful way to explore the data. It can lead to a greater understanding of the data for you and your research team. For example, it can shed light on the nature and quality of the data (missing data patterns, outliers, etc.), and it can help you to choose the appropriate direction for your analysis, avoiding wasted time and frustration.

Cox (2010) discusses graphing data with an emphasis on comparisons between individuals or between groups of individuals. Graphing each individual's data over time (in separate graphs) may be a neglected approach in exploring datasets. By its nature, this approach emphasizes within- (rather than between-) individual differences. Studying graphs of a few individuals can deepen your understanding of the data and generate ideas as to what questions you might ask of the data and how you might analyze the data. In datasets with a lot of information on several individuals over time, graphing raw data can be challenging. Symbols and lines can overlay each other and make graphs hard to read. One solution is to produce a separate graph for each individual. In situations where little information exists (that is, just one or two variables) on a small number of individuals, this can satisfactorily be achieved using the `by()` option of `twoway`. In the rest of this article, I provide code with comments on how such graphs might be produced and collated in a more general setting.

1.1 Dataset used in examples

We will use data from a subsample of the National Longitudinal Survey of Labor Market Experience (Center for Human Resource Research 1989) on young women aged 14–26 years in 1968. Women were surveyed in each of the 21 years from 1968 to 1988, except for 1974, 1976, 1979, 1981, 1984, and 1986. `nlswork.dta`, often used in the Stata 15 [XT] *Longitudinal-Data/Panel-Data Reference Manual*, contains data on 4,711 women in years when they were employed, not enrolled in school and having completed their education, and made wages in excess of $1 per hour but less than $700 per hour. An individual woman in the dataset has one row of data for each year she was surveyed. The year is coded as a two-digit year ranging from 68 to 88. The dataset contains data on how wages and related factors change over time:

```
use http://www.stata-press.com/data/r15/nlswork.dta
xtset
xtdescribe
```

To reproduce figures 1 and 2, you will need to set the *Stata Journal* graph scheme:

```
set scheme sj
```

In Stata for Windows to create multiple graphs as tabs within one window (rather than as separate windows), you might want to use the setting

```
set autotabgraphs on
```

2 Simple example—Graphing a single variable over time for each individual

2.1 Preparatory work

As a first example, suppose we wish to produce graphs of wages over time for each individual. We might plot wages on the y axis (with linear scaling) and years on the x axis. We examine the distribution of wages. Because some wages are so much higher than the rest of the data, we truncate wages at $20 per hour.

```
generate wage = exp(ln_wage)
summarize wage, detail
generate wage_trunc = min(wage, 20) if !missing(wage)
label variable wage_trunc "Wage ($/hour)"
```

To demonstrate how to graph each individual's data over time in separate graphs, we do not need graphs for all 4,711 individuals in this dataset. We now restrict the dataset to the first six individuals.

```
keep if idcode <= 6
```

2.2 Code for graphing a single variable over time for each individual

```
putdocx clear
putdocx begin, landscape
putdocx paragraph
levelsof idcode, local(idcodes) clean
foreach id of local idcodes {
   scatter wage_trunc year if idcode == `id´, connect(l) sort                ///
     ylabel(0 2 4 6 8 10 12 14 16 18 20 "≥20", angle(horizontal))            ///
     yscale(range(0 20.5))                                                   ///
     xlabel(68 "1968" 73 "1973" 78 "1978" 83 "1983" 88 "1988")               ///
     xmtick(68 69 70 71 72 73  75  77 78  80  82 83  85  87 88, grid notick) ///
     xscale(range(67.5 88.5))                                                ///
     title("id `id´")                                                        ///
     graphregion(color(white))                                               ///
     name(g`id´, replace)
     graph export "temp_graph.png", replace
     putdocx image "temp_graph.png"
}
putdocx save "Example 1. Individual Graphs.docx", replace
erase "temp_graph.png"
```

The result is six graphs, one of which is shown in figure 1.

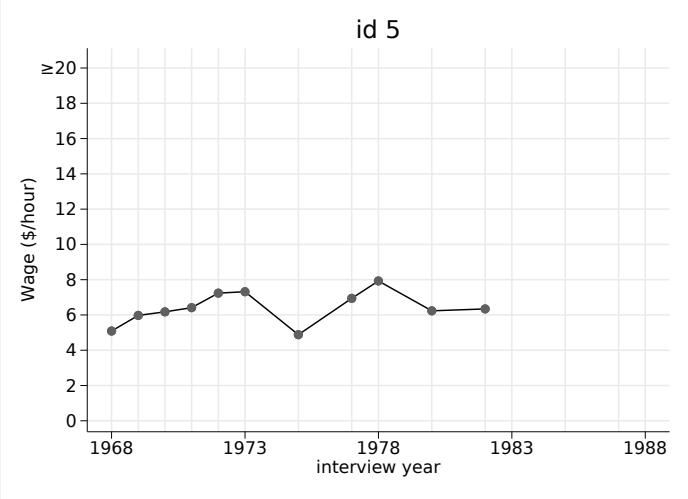

Figure 1. A graph of wage over time for one individual (`idcode == 5`)

2.3 Comments on code

Before explaining the meaning of the code, I will comment on how the code was developed. The first line of code in section 2.2 was not written first! Rather, I wrote code in the following order: First, I wrote code to produce a graph for one specific individual.

Second, I generalized and added to the code so that a separate graph would be produced for each individual. Finally, I added code to save the graphs into a `.docx` file. I now comment on the code in that order.

After deciding to plot an individual's data on a connected scatterplot, I wrote basic code for the first individual:

```
scatter wage_trunc year if idcode == 1, connect(l) sort
```

It is often desirable to have the same labels and scaling of axes for each graph. I decided upon the following options:

- y axis: the values to reflect the truncation of wages at \$20 per hour, and for major ticks, (horizontal) labels and grid lines to appear from 0 to ≥ 20 in steps of 2.

```
ylabel(0 2 4 6 8 10 12 14 16 18 20 "≥20", angle(horizontal))           ///
yscale(range(0 20.5))                                                  ///
```

- x axis: major ticks and labels every 5 years from 1968 to 1988 and grid lines to appear only for the 15 years of data collection.

```
xlabel(68 "1968" 73 "1973" 78 "1978" 83 "1983" 88 "1988")              ///
xmtick(68 69 70 71 72 73  75  77 78  80  82 83  85  87 88, grid notick) ///
xscale(range(67.5 88.5))                                               ///
```

  Note that the range of the axes needed to be expanded so that grid lines appeared at $y = 20$, $x = 68$, and $x = 88$.

In case some graphs get printed and as a personal preference, I changed the line and fill color and opacity of the outer graph region (which is blue in scheme `s2color`) to white:

```
graphregion(color(white))                                              ///
```

To produce a graph for each individual, I created a local macro containing all `idcodes`, used a loop to run the same code for each individual, and generalized the code (to refer no longer to `idcode == 1` but to `idcode == ‘id’`):

```
levelsof idcode, local(idcodes) clean
foreach id of local idcodes {
    (commands referring to `id´)
}
```

Note that while the option `clean` makes no difference in this example (because `idcode` is numeric), in other datasets, an ID variable will be string format (with no spaces), and the code in this article will work fine as long as the option `clean` is used to remove double quotes and `‘id’` is enclosed in double quotes where appropriate, for example, if `idcode == "‘id’"`.

I inserted an individual's ID into the title of his or her graph:

```
title("id `id´")
```

To see multiple graphs within Stata as they are created (and re-created), I used the `name()` option:

```
name(g`id´, replace)
```

To put all graphs into a `.docx` file, I used the following code:

```
putdocx clear
putdocx begin, landscape
putdocx paragraph
(inside foreach loop)
graph export "temp_graph.png", replace
putdocx image "temp_graph.png"
putdocx save "Example 1. Individual Graphs.docx", replace
```

The first line closes an open document if one had previously been created (with `putdocx begin`) without saving, or else does nothing. The second line creates a fresh document for export. (Stata 15.1 can have only one document open and ready to be saved as a `.docx` file at any one time.) The third line adds a paragraph to the document, which is necessary so that images can then be added (images can be added only to paragraphs). Inside the `foreach` loop, the graph is written to a `.png` file before the `.png` file is immediately added to the document. For the next `idcode`, the `.png` file is overwritten, and the `.png` file is then added to the document. Finally, the document is closed and saved as "`Example 1. Individual Graphs.docx`", overwriting the `.docx` file if it already exists. See [P] **putdocx** or [P] **putpdf** for more on exporting graphs to a `.docx` or `.pdf` file.

Finally, having served its purpose, the `.png` file was erased from disk:

```
erase "temp_graph.png"
```

3 A more complicated example—Graphing multiple variables over time for each individual

3.1 Preparatory work

Suppose we wish to present the data on the following variables shown in the `describe` results:

```
use http://www.stata-press.com/data/r15/nlswork, clear
generate wage = exp(ln_wage)
keep idcode tenure hours union msp race grade age year wage
generate death_yr = 83 if idcode==5 // variable created for illustration
```

```
. describe
Contains data from http://www.stata-press.com/data/r15/nlswork.dta
  obs:        28,534                          National Longitudinal Survey.
                                                Young Women 14-26 years of age
                                                in 1968
 vars:            11                          27 Nov 2016 08:14
 size:       627,748
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
idcode          int     %8.0g                 NLS ID
year            byte    %8.0g                 interview year
age             byte    %8.0g                 age in current year
race            byte    %8.0g      racelbl    race
msp             byte    %8.0g                 1 if married, spouse present
grade           byte    %8.0g                 current grade completed
union           byte    %8.0g                 1 if union
tenure          float   %9.0g                 job tenure, in years
hours           int     %8.0g                 usual hours worked
wage            float   %9.0g
death_yr        float   %9.0g
-------------------------------------------------------------------------------
Sorted by: idcode  year
     Note: Dataset has changed since last saved.
```

The first thing to do is to discover which variables vary with year within an individual and which do not. In this example, because the dataset has been `xtset` and there are no string variables, one can use the command `xtsum`. Variables that do not change with year will have within standard deviation of 0. More generally, one can use the community-contributed command `distinct` (Cox and Longton 2008) to check whether variables are constant within an individual. The same numbers in the output below tell us that `grade` is constant within `idcode`. The only other variables that are constant within `idcode` are `race` and `death_yr`.

```
. distinct idcode
```

	total	distinct
idcode	28534	4711

```
. distinct idcode grade, joint missing
```

	total	distinct
(jointly)	28534	4711

Next, we consider the four time-varying continuous variables: `wage`, `tenure`, `hours`, and `age`. Which variable or variables shall we plot on which axis? Do we use log scales? Should we treat a continuous variable or variables the same as we would a string variable? It was following this last line of thought that, after some deliberation, I decided to add information in the numeric variable `age` as text near the x axis. (Age

does not correspond perfectly with year because a respondent is not interviewed exactly one year from the date of the last interview.) Next, I examined the values of the other three variables and decided to use linear scales and truncate large values. I also decided that `wage` and `tenure` could share the same y axis (ranging from 0 to 20).

```
summarize wage, detail
generate wage_trunc = min(wage, 20) if !missing(wage)
label variable wage_trunc "Wage ($/hour)"
summarize tenure, detail
label variable tenure_trunc "Tenure (years)"
generate tenure_trunc = min(tenure, 20) if !missing(tenure)
summarize hours, detail
generate hours_trunc = min(hours, 60) if !missing(hours)
label var hours_trunc "Usual hours worked"
```

Next, we consider the two time-varying categorical variables `union` and `msp`. We examine the values and decide the letters "`U`" for `union`, "`M`" for married, and "`-`" for neither in a union nor married would be nice on a graph.

```
tabulate union, missing
generate lab_union = "U" if union==1
replace lab_union = "-" if union==0
tabulate msp, missing
generate lab_married = "M" if msp == 1
replace lab_married = "-" if msp == 0
```

Let us now consider the nontime-varying variables. We decide to draw a vertical line on an individual's graph corresponding to `death_yr` if that information exists. We can present information on `grade` and `race` in a title near to the individual's `idcode`. Because the values of `race` are labeled, we create a variable containing the value labels for `race`, which will make adding this information to the graphs easier.

```
decode race, generate(race_str)
```

As in the simple example, the dataset is now restricted to the first six individuals.

```
keep if idcode <= 6
```

3.2 Code for a graph of multiple variables over time for each individual

```
generate y_union = 23
generate y_married = 26
generate y_age = 0
putdocx clear
putdocx begin, landscape
putdocx paragraph
levelsof idcode, local(idcodes) clean
foreach id of local idcodes {
     local xline ""
     local xlinetext ""
```

```
        summarize death_yr if idcode == `id´
        if r(N) > 0 {
             local death_year = r(min)
             if r(min) <= 88 {
                  local xline "xline(`death_year´)"
                  local xlinetext `"text(21 `death_year´ "Died")"´
             }
        }
        levelsof race_str if idcode == `id´, local(race_info) clean
        levelsof grade if idcode == `id´, local(grade_info)
        twoway (scatter wage_trunc tenure_trunc year, connect(l l) sort         ///
               clwidth(thick) msymbol(O T))                                     ///
          (scatter y_union y_married year, msymbol(i i)                         ///
               mlabel(lab_union lab_married) mlabposition(0 0))                 ///
          (scatter y_age year, msymbol(i) mlabel(age) mlabposition(6)           ///
               mlabgap(*0.1) mlabcolor(gs8))                                    ///
          (scatter hours_trunc year, yaxis(2) connect(l) sort msymbol(Sh)       ///
               clpattern(dash))                                                 ///
          if idcode == `id´,                                                    ///
          xlabel(68 "1968" 73 "1973" 78 "1978" 83 "1983" 88 "1988", notick      ///
               labsize(small) labgap(*6))                                       ///
          xmtick(68 69 70 71 72 73 75 77 78 80 82 83 85 87 88, grid notick)     ///
          `xline´ `xlinetext´                                                   ///
          xscale(range(67.5 88.5))                                              ///
          plotregion(margin(zero))                                              ///
          xtitle("")                                                            ///
          text(-1 66 "Age", size(small) color(gs8))                             ///
          text(-2.7 66 "Year", size(small))                                     ///
          graphregion(color(white))                                             ///
          legend(order(1 - "         " 6  2 ) span cols(3) )                    ///
          ytitle("● Wage ($/hour)       " "▲ Tenure (years)     ")              ///
          ytitle("□ Usual hours worked      ", axis(2))                         ///
          title("id `id´") subtitle("(`race_info´, grade `grade_info´)")        ///
          ylabel(0 2 4 6 8 10 12 14 16 18 20 "≥20" 23 "Union?" 26 "Married?",   ///
               angle(horizontal))                                               ///
          ylabel(0 6 12 18 24 30 36 42 48 54 60 "60+", axis(2)                  ///
               angle(horizontal))                                               ///
          yscale(range(0 27)) yscale(range(0 81) axis(2))                       ///
          name(g`id´, replace)
        graph export "graph.png", replace
        putdocx image "graph.png"
}
putdocx save "Example 2. Individual Graphs.docx", replace
erase "graph.png"
```

The result is six graphs, one of which is shown in figure 2.

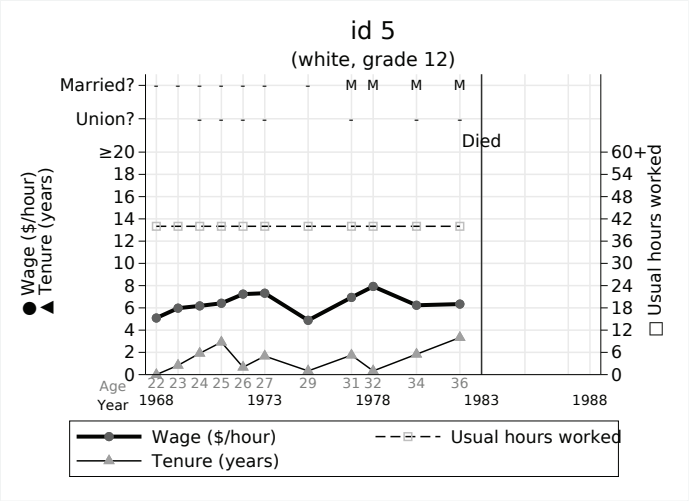

Figure 2. A graph of wage and other factors over time for one individual (`idcode == 5`)

3.3 Comments on code

I will now comment on code that is additional to that used in the simple example.

I plotted the variables `wage_trunc` and `tenure_trunc` on the same y axis:

```
(scatter wage_trunc tenure_trunc year, connect(l l) sort         ///
      clwidth(thick) msymbol(O T))                               ///
```

Using the same y axis, I added invisible markers with visible marker labels for the two time-varying categorical variables `union` and `msp`. I specified particular heights on the y axis for this information so that it appeared above the presentation of the continuous variables `wage_trunc` and `tenure_trunc` (which range from 0 to 20). I substituted text for labels on the y axis.

```
generate y_union = 23
generate y_married = 26
(scatter y_union y_married year, msymbol(i i)                      ///
     mlabel(lab_union lab_married) mlabposition(0 0))              ///
   , ylabel(0 2 4 6 8 10 12 14 16 18 20 "≥20" 23 "Union?" 26 "Married?",   ///
         angle(horizontal))                                        ///
```

In the same way, I added invisible markers with visible labels underneath for `age` (at $y = 0$), which gave the impression that two scales (`age` and `year`) were simultaneously used on the x axis. To make the graph tidy, I used some options that included eliminating the margin between the axes and the plot region.

```
generate y_age = 0
(scatter y_age year, msymbol(i) mlabel(age) mlabposition(6)          ///
     mlabgap(*0.1) mlabcolor(gs8))                                   ///
   , plotregion(margin(zero))                                        ///
     xtitle("")                                                      ///
     text(-1 66 "Age", size(small) color(gs8))                       ///
     text(-2.7 66 "Year", size(small))                               ///
```

I plotted the variable `hours_trunc` on the y axis on the right by specifying `yaxis(2)`. This axis was labeled and scaled in such a way that the grid lines for the first y axis nicely corresponded with labels on the second y axis. (Note that unless `axis(2)` is specified, Stata thinks y-axis options apply to the first axis.)

```
(scatter hours_trunc year, yaxis(2) connect(l) sort msymbol(Sh)      ///
     clpattern(dash))                                                ///
   , ylabel(0 6 12 18 24 30 36 42 48 54 60 "60+", axis(2)            ///
        angle(horizontal))                                           ///
     yscale(range(0 27)) yscale(range(0 81) axis(2))                 ///
```

When several continuous variables are plotted on y axes, it can be challenging to communicate which variables are which and to which axis they relate. I attempted to achieve this by using a legend with the two variables associated with the left y axis appearing in the left side of the legend and with the one variable associated with the right y axis appearing in the right side of the legend. Perhaps unnecessarily so, I also used the option `ytitle()`, which does not add symbols and connecting lines as nicely as `legend()`. To find the circle, triangle, and square symbols, please see the *Appendix*.

```
legend(order(1 - "         " 6  2 ) span cols(3) )                   ///
ytitle("● Wage ($/hour)       " "▲ Tenure (years)     ")             ///
ytitle("□ Usual hours worked      ", axis(2))                        ///
```

To add a vertical line for `death_yr`, I used the following code. It is important that the local macros `‘xline’` and `‘xlinetext’` do not contain information from the previous ID, hence the first two lines above resetting the macro to be empty. If `death_yr` is not missing for an individual and is no later than 1988, then the macros are not empty, and they will produce a vertical line at that time, with the text “Died”. It is a good idea to verify that death information from individual 5 has not accidentally been carried forward and plotted on the graph of individual 6.

```
local xline ""
local xlinetext ""
summarize death_yr if idcode == `id´
if r(N) > 0 {
     local death_year = r(min)
     if r(min) <= 88 {
          local xline "xline(`death_year´)"
          local xlinetext `"text(21 `death_year´ "Died")"´
     }
}
twoway ..., `xline´ `xlinetext´
```

Finally, to add the information from i) a string and ii) a numeric nontime-varying variable to the subtitle, I used the following code. I used the `clean` option of `levelsof`

so that the resulting local macro did not contain compound double quotes for string variables.

```
levelsof race_str if idcode == `id´, local(race_info) clean
levelsof grade if idcode == `id´, local(grade_info)
twoway ..., subtitle("(`race_info´, grade `grade_info´)")
```

4 Concluding remarks

I provided examples to share thought processes and coding that will be useful for many datasets, especially datasets where several variables change over time and where there are many possible time points (for example, some events and measures can take place on any day). Exploration of administrative datasets and patient safety profiles in clinical trials are two examples. Much is possible using Stata's two-way graphs and a spoonful of creativity. More detail on possibilities can be found in Stata's PDF documentation: [G-2] **graph twoway** and [G-2] **graph twoway scatter**. I provide a few more tips in the *Appendix*. Experimentation is important to see what is effective for a particular dataset and a particular audience.

Such graphs can take a little time to explain to your collaborators, and looking closely at a few graphs can also take a bit of time. However, my experience is such that it is time well spent for you and your collaborators (study a dozen or so graphs together if you can). One can often see many individual stories, and one can select a few typical or unusual graphs for a powerful presentation at meetings. I hope this article encourages and empowers you to take a little time to explore such datasets graphically. What will be revealed as you do? And what ideas will be generated?

5 Acknowledgments

I am grateful for suggestions on drafts of the article by Satomi Okano, Karen Hay, and Gunter Hartel.

6 References

Center for Human Resource Research. 1989. National Longitudinal Survey of Labor Market Experience, Young Women 14–26 years of age in 1968. Ohio State University.

Cox, N. J. 2010. Speaking Stata: Graphing subsets. *Stata Journal* 10: 670–681.

Cox, N. J., and G. M. Longton. 2008. Speaking Stata: Distinct observations. *Stata Journal* 8: 557–568.

A Appendix

Some other useful tips:

1. Append rather than merge. Sometimes, information on an individual lies in different datasets. Provided that a common ID variable appears in these datasets, the datasets can be combined for the purpose of creating graphs using `append` rather than `merge`. This will be a relief for situations where the data are not easy to merge.

2. To produce graphs for a subset of (approximately 1%) of randomly chosen individuals, type

```
set seed 2476247
generate random_number = runiform()
levelsof idcode if random_number < 0.01, local(idcodes)
```

3. You can use the command `quietly` to suppress some output in the Results window.

4. How to find symbols not on the keyboard?

   For the greater than or equals to symbol:

```
display ustrunescape("\u2265")
```

   For the black circle:

```
display ustrunescape("\u25CF")
```

   For the black triangle:

```
display ustrunescape("\u25B2")
```

   For the white square:

```
display ustrunescape("\u25A1")
```

   For more symbols, see
   https://www.rapidtables.com/code/text/unicode-characters.html or
   http://unicode.org/charts/#symbols.

   Rather than copying and pasting the symbol from the Results window into the do-file, users might instead create a local macro containing the symbol and use the local macro in titles and labels. (Some symbols do not display properly in the Results window or do-file where typically the Courier New font is used and they appear as a box.)

5. An option that may be useful when there are many time-varying continuous variables to be plotted is to plot a variable or variables on what looks like another graph above or below the current graph. For example, the following code added into the right places in the second example will plot `hours_trunc` using the first y axis in the as-yet-unused space where y ranges from 30–50. Figure 3 shows one

of the graphs produced. Another way of creating similar graphs would be to use `graph combine`.

```
generate hours_trunc_trick = 30 + hours_trunc/3
(scatter hours_trunc_trick year, connect(l) sort msymbol(sh)             ///
      clpattern(dash))                                                   ///
   , ylabel(0 2 4 6 8 10 12 14 16 18 20 "≥20" 23 "Union?" 26 "Married?"  ///
         30 "0" 40 "30" 50    "60+", angle(horizontal))                  ///
     yline(30, lcolor(gs8)) ysize(6) legend(off)                         ///
     ytitle("● Wage ($/hour)                      □ Usual hours worked" ///
         "▲ Tenure (years)                            ")               ///
```

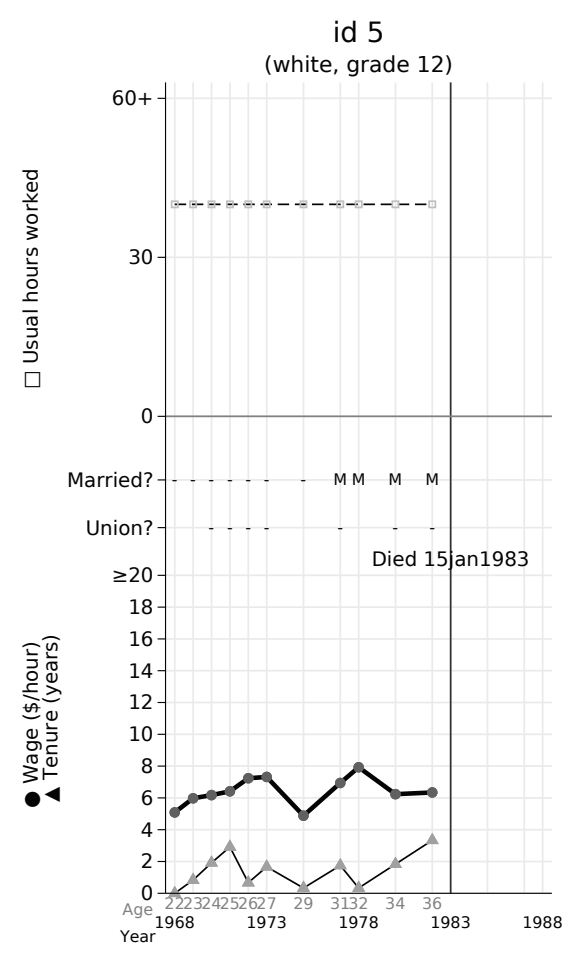

Figure A.3. A graph of wage and other factors over time for one individual (`idcode == 5`), plotting `hours_trunc` above the other data on the one y axis

6. To create a local macro consisting of a formatted date (so a date can be added into the graph as text), type

```
levelsof dod if idcode == `id´, local(dodnum)
local dodformatted : display %td `dodnum´
```

I used the above code (with the below code) in creating figure 3.

```
generate dod = 8415 // variable created for illustration
local xlinetext `"text(21 `death_year´ "Died `dodformatted´")"´
```

About the author

Mark Chatfield is an applied biostatistician and has enjoyed using Stata almost exclusively for 15 years. Work was done on this article while working in the Statistics Unit, QIMR Berghofer Medical Research Institute, Brisbane, Australia.