How can I convert data from person-level to person-period format?

Authored by

stats writer

June 30, 2024

Converting data from person-level to person-period format involves restructuring individual-level data into a format that represents the same person over multiple time periods. This process allows for the analysis and comparison of data at the individual level over time. It typically requires the aggregation of data for each individual, such as demographic information or behavior patterns, into a single record for each time period. This format is commonly used in longitudinal studies and can provide valuable insights into changes and trends over time for a particular population. The conversion process may involve merging multiple datasets, creating new variables, and applying statistical techniques to ensure data accuracy. Overall, converting data from person-level to person-period format is a crucial step in conducting in-depth and comprehensive analyses of individual-level data.

How can I convert from person-level to person-period? | R FAQ

In studies of survival or modeling discrete-time events, one compact way to store data is in what may be called, "person-level" or
generally "observation-level". For example, you could have three variables,
one indicating the observation, one indicating the time period the event occurred
or the last follow-up period and one indicating whether the observation was censored.
The book, Applied Longitudinal Data Analysis.
has an example of this sort of data in chapter 10. Teachers are observed in a school
district for a 12 year period, with the event being whether or not they left their job.

Another way to store the same data would be in what could be called, "person-period".
Here, there is a separate row for each person for each period they were observed. The event
occurs in the last period they were observed unless the observation was censored. The question
then is, how can a dataset be converted from one format to the other? For demonstration, we
will use the dataset on teachers from the book mentioned above.

Basics

We define a custom function that will convert back and forth for us.
Note that this is just one of many possible ways to do this in R. In this section
we describe how to use the function and give examples. The next section ("Details")
explains how it works and what happens at each step of the way.

The PLPP function takes five arguments. The first, data is the
data set to be converted. The second, id is the name of

the variable containing
the identifier for each observation. The third, period is the name of the variable
that indicates how many periods the person or observation was in. The fourth, event
is the name of the variable that indicates whether the event occurred or not or whether
the observation was censored (depending on which direction you are converting). The
fifth, direction indicates whether the function should go from person-level to
person-period or from person-period to person-level. There are two options, "period"
to go to person-period or "level" to go to person-level. Now let's try it out.
For the examples that follow to work, you need to source the function into R.

Person-Level Person-Period Converter Function PLPP

Now we can use the function to convert between person-level and person-period

datasets.

Read in the person-level dataset

teachers

id t censor

19 20 3 0

125 126 12 0

128 129 12 1

Uses PLPP to convert to person-period and store in object, 'tpp'

tpp

id t censor

95 20 1 0

96 20 2 0

97 20 3 1

740 126 1 0

741 126 2 0

742 126 3 0

743 126 4 0

744 126 5 0

745 126 6 0

746 126 7 0

747 126 8 0

748 126 9 0

749 126 10 0

750 126 11 0

751 126 12 1

760 129 1 0

761 129 2 0

762 129 3 0

763 129 4 0

764 129 5 0

765 129 6 0

766 129 7 0

767 129 8 0

768 129 9 0

769 129 10 0

770 129 11 0

771 129 12 0

Because there is an example dataset in person-period form provided

from the book, we can read that dataset in and compare our results with

that one.

Read in person-period dataset

teachers.pp

id period event

95 20 1 0

96 20 2 0

97 20 3 1

740 126 1 0

741 126 2 0

742 126 3 0

743 126 4 0

744 126 5 0

745 126 6 0

746 126 7 0

747 126 8 0

748 126 9 0

749 126 10 0

750 126 11 0

751 126 12 1

760 129 1 0

761 129 2 0

762 129 3 0

763 129 4 0

764 129 5 0

765 129 6 0

766 129 7 0

767 129 8 0

768 129 9 0

769 129 10 0

770 129 11 0

771 129 12 0

Aside from different column names, they look identical. In fact
if we were to change the names, we could whether the two datasets are
equal using the all.equal function.

colnames(tpp)
TRUE

The PLPP function can also convert from a person-period to a
person-level dataset. Here is an example going the opposite direction.

Note that the names passed to the period and event arguments changed from
before because we overwrote the column names in tpp with those in
teachers.pp.

Convert from person-period to person-level
t2
id period event
19 20 3 0
125 126 12 0
128 129 12 1

In fact, as before, we might test that t2 is equal to teachers.
We can do this, all we need to do is change the column names.

colnames(t2)
TRUE

We could go one step farther that the conversion back and forth

is the same. We convert teachers to period and back to level
and compare with unconverted teachers
all.equal(PLPP(data = PLPP(teachers, "id", "t", "censor", "period"),
"id", "t", "censor", "level"), teachers)

TRUE

Details

The basic concept of how the conversion works is straightforward, but the
code to do it is a bit tricky. First, we will look at a list of what needs to be
accomplished to go from person-level to person-period.

To show each step of the function, I will show the log from an
interactive debugging session (this allows me to step through the function
line by line playing with what is there at each step).

debug(PLPP)

```
test
debugging in: PLPP(teachers, "id", "t", "censor",
"period")
debug: {
stopifnot(is.matrix(data) || is.data.frame(data))
stopifnot(c(id, period, event) %in% c(colnames(data),
1:ncol(data)))
if (any(is.na(data))) {
stop("PLPP cannot currently handle missing data in the
id, period, or event variables")
}
switch(match.arg(direction), period = {
index ## The first thing we see is an echo of the parsed
function definition
Browse> ## note that in the parsed version, comments
and spaces are removed
Browse> ## now R will proceed through the function
line by line
Browse> ## the first part of the function is just a series
of logical checks
Browse> ## to make sure that the data is in an
acceptable format and all the
Browse> ## arguments are appropriate for the function
Browse> ## it is not fool proof, but it helps to ensure
```

that the results are
Browse> ## accurate and trustworthy. We can also list everything in the function
Browse> ls()
"data" "direction" "event" "id" "period"
Browse> ## This shows us all the objects currently defined inside the function
Browse> ## environment. These are recognizable as all the arguments
Browse> class(data)
"data.frame"
Browse> direction
"period"
Browse> event
"censor"
Browse> id
"id"
Browse> period
"t"
Browse> ## So the data is a data frame and we can see that all the argument variables
Browse> ## contain the values that we set them to (as they should)
Browse>

debug: stopifnot(is.matrix(data) || is.data.frame(data))
Browse> ## this is the first check, data must be a matrix or data frame
Browse>
debug: stopifnot(c(id, period, event) %in% c(colnames(data), 1:ncol(data)))
Browse> ## the second check, the values passed to id, period, and event
Browse> ## must be in the column names of the data or in the column indices
Browse>
debug: if (any(is.na(data))) {
stop("PLPP cannot currently handle missing data in the id, period, or event variables")
}
Browse> ## this checks whether any of id, period, or event contain missing values
Browse> ## if there were missing values, the function would return the error message
Browse> ## written above
Browse> ## is.na() returns TRUE/FALSE for each cell, any() says, are there *any* TRUE
Browse>
debug: NULL

Browse> ## NULL is returned because the condition for the if() statement was not met
Browse> ## this means that R will skip over the expression inside the braces { }
Browse> ## and continue evaluation
Browse>
debug: switch(match.arg(direction), period = {
index ## Now the debugger is showing us the next line of code that is going to
Browse> ## be debugged. On the surface, this may seem like more than one line
Browse> ## but it is all one call to the function, switch() so it is one line to R
Browse> ## in order to examine each step more closely, we will need to turn on
Browse> ## debugging for the switch function too (or everything in there would fly by)
Browse> debug(switch)
Browse> ## now we can continue
Browse>
debug: index ## notice we are now debugging inside of switch()
Browse> ## so the first thing that happened is based on the argument, direction

Browse> ## the switch function jumps to the code after period or the code after level
Browse> ## it will only pick one, and what is not chosen, is not evaluated
Browse> direction
"period"
Browse> ## since direction == "period", we are in the 'period' track
Browse> ## the first line (above) of the period track created an index
Browse> ## this is used to replicate each row of the original dataset as many times as
Browse> ## necessary.
Browse> ## for 1 through n rows of the dataset, it repeats the row number
Browse> ## the number of times in the variable period
Browse> ## so for example
Browse> data
1 2 1 1 12
Browse> ## the first row will be repeated 1 time, the second 2 times the third 1 time
Browse> ## the fourth 1 time and the fifth 12 times
Browse> ## on a small scale this looks like:
Browse> rep(1:5, data)

1 2 2 3 4 5 5 5 5 5 5 5 5 5 5

Browse> ## These values are then stored in the object, index

Browse> ls()

"data" "direction" "event" "id" "period"

Browse>

debug: idmax index

1 2 2 3 4 5 5 5 5 5 5 5 5 5 5 6 7 7

Browse> ## note that the first 20 elements of index match the first few from our example above

Browse> ## next we create an object called 'idmax'

Browse> ## this contains the cumulative sum of the period variable in the original data

Browse> ## this will serve as an index of the row numbers in the person-period data

Browse> ## where the event should occur (unless it is censored)

Browse> ## again just for the first 5 rows of the person-level data

Browse> cumsum(data)

1 3 4 5 17

Browse> ## compare this with the index

Browse> index

1 2 2 3 4 5 5 5 5 5 5 5 5 5 5 6 7 7

Browse> ## the first row is repeated once, so its event occurs in row 1 of the person-period data
Browse> ## the second row is repeated twice, 1 + 2 = 3 so the event for row 2 of the person-level
Browse> ## appears in row 3 of the person-period, and so on and so forth for all the rows
Browse> ## Now for the event, we want it to be 1 if the event occurs or 0 otherwise
Browse> ## However, if the observation is censored, (1 for censored, 0 for not censored)
Browse> ## then the event should not occur
Browse> ## in short, for a given observation, the last period should contain the event
Browse> ## if it is not censored
Browse> ## R stores TRUE/FALSE internally as 1/0, so if we want an event (1) if censored is FALSE (0)
Browse> ## we can just negate the value in the censored variable to use as the value for the event
Browse> ## variable. This is what the next line of code does
Browse>
debug: reve ## now we create the person-period data, 'dat', using the index we created (replicates rows)
Browse>

debug: dat ## the next line of code creates a sequence (seq_along) of period from the
Browse> ## person-period data, _for each ID_
Browse> ## For example, for the first row, it would just be 1 (since that was only repeated once)
Browse> ## for the second row, it would be 1 2 (since that row was repeated twice)
Browse> ## in particular, the variable will be an integer class starting at 1 going to whatever the
Browse> ## number of times a particular ID was repeated in the person-period data
Browse> ## (this should always be the same as the value in the period variable from the
Browse> ## person level data, because that is what _determined_ how many times the rows were repeated
Browse> ## in the person-period dataset)
Browse>
debug: dat ## now since events only occur (if they occur at all) in the last
Browse> ## period of observation for a given ID, we will set all events equal to 0
Browse>
debug: dat ## now using the idmax index (which contains the rows of the last period for each ID)

Browse> ## and the reve object which is just the negation of whether an ID was censored
Browse> ## we can overwrite the relevant rows of the event variable in the new person-period
Browse> ## dataset with the appropriate values (TRUE/FALSE)
Browse> ## however, since we set everything to 0, it is currently an integer variable
Browse> ## and since we are only replacing a few rows of it, the TRUE/FALSE values will
Browse> ## be upgraded from logical (a lower class) to integer (a higher class)
Browse> ## which means they will be converted to their underling 1/0 values
Browse> ## basically R is quietly doing
Browse> reve
TRUE TRUE TRUE TRUE FALSE TRUE FALSE TRUE TRUE TRUE
Browse> as.integer(reve)
1 1 1 1 0 1 0 1 1 1
Browse>
debug: dat ## once that is done the 'dat' variable contains the finalized person-period data
Browse> ## the only thing left to do is since rows were

repeated
Browse> ## the rownames will be rather strange (orig.number_of_repeats)
Browse> ## for example 1.1
Browse> ## 2.1 and 2.2, etc.
Browse> ## we will simply set them to NULL so that R will use the default
Browse> ## which is just an index from 1 to the total number of rows
Browse>
debug: rownames(dat) ## (note that we jumped out of the switch debugging because it was done)
Browse> ## now we can just return the object 'dat' which is the converted dataset
Browse>
debug: return(dat)
Browse> ## and we are done
Browse>
exiting from: PLPP(teachers, "id", "t", "censor", "period")

Now we will do an abbreviated version of the same, going from person-period
to person-level. Only the differences will be highlighted

during the debugging.

test2
debugging in: PLPP(teachers.pp, "id", "period", "event", "level")
debug: {
stopifnot(is.matrix(data) || is.data.frame(data))
stopifnot(c(id, period, event) %in% c(colnames(data), 1:ncol(data)))
if (any(is.na(data))) {
stop("PLPP cannot currently handle missing data in the id, period, or event variables")
}
switch(match.arg(direction), period = {
index ## Same as before
Browse>
debug: stopifnot(is.matrix(data) || is.data.frame(data))
Browse>
debug: stopifnot(c(id, period, event) %in% c(colnames(data), 1:ncol(data)))
Browse>
debug: if (any(is.na(data))) {
stop("PLPP cannot currently handle missing data in the

id, period, or event variables")
}
Browse>
debug: NULL
Browse>
debug: switch(match.arg(direction), period = {
index ## Now we are entering switch() again, this time it will go to the
Browse> ## level track because
Browse> direction
"level"
Browse> ## the argument direction matches a different value
Browse> ## the first thing we do is create a small temporary dataset
Browse> ## it just contains the rows necessary for the conversion (i.e., period and id)
Browse> ## the reason is that in theory, the data could have had 100s or 1000s of columns
Browse> ## containing other variables measured along with the period and event
Browse> ## we are only going to extract one row for each ID, the final period it was observed
Browse> ## but to do this, we need to subset the

dataset by ID
Browse> ## rather than subset a potentially enormous dataset
Browse> ## we make a small copy, then once the row indices are known
Browse> ## the final data returned is just the appropriate rows from the full data
Browse>
debug: tmp ## Now that we have our small working data set, we need to find the row indices
Browse> ## that contain the highest value of period for each ID
Browse> ## we use the by() function which applies the function we specify
Browse> ## to the tmp data, subset by each unique id
Browse> ## the final results (which are in a special class called 'by' that
Browse> ## is not unlike a list, but is not quite a list) are converted to a vector
Browse> ## using the as.vector() function
Browse>
debug: index ## now we can create the person-level data set by just extracting the relevant rows
Browse> ## using our index variable

Browse>

debug: dat index

1 3 4 5 17 18 30 31 33 35 42 54 55 67 79 81 93 94 97 99

Browse> ## in fact this should look quite familiar to us

Browse> ## it is the same as idmax from the other direction created with cumsum

Browse> ## the final step is to convert from whether or not an event occurred in the final

Browse> ## observed period to whether the observation is censored

Browse> ## just like we negated censoring to create the event, we can negate the event to create

Browse> ## the censoring

Browse> ## the only new trick is that because we are replacing the entire column

Browse> ## if we want 1/0s instead of TRUE/FALSE, we will need to be explicit

Browse> ## this is done using the as.integer() function

Browse>

debug: dat ## now as before we will reset the rownames to their default by setting to NULL

Browse>

debug: rownames(dat) ## and we can return 'dat' which is the person-level dataset created from

Browse> ## the person-period data
Browse>
debug: return(dat)
Browse>
exiting from: PLPP(teachers.pp, "id", "period", "event", "level")

Finally, to avoid entering debugging everytime you use the function (until you close and
restart R), use the undebug function.

undebug(PLPP)
undebug(switch)

and from person-level to person-period, that takes you there and back again.