

# Converting numeric values to class "Date"

Mark Eisler and Ana Rabaza

March 28, 2026

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Convert numeric values representing year to class "Date"</b>	<b>1</b>
<b>3</b>	<b>Convert numeric values representing year and month to class "Date"</b>	<b>4</b>
<b>4</b>	<b>Convert numeric values representing ages to class "Date"</b>	<b>7</b>

## 1 Introduction

For each observation of a subject in a longitudinal study data set, the main **Transition** package functions `add_prev_date()`, `add_prev_result()` and `add_transitions()` all need to identify the previous observation for that same subject, if any. For compatibility with these **Transition** package functions, the timings of observations in a dataset, each referred to as a *timepoint*, should be coded within the data frame as a column of R class `"Date"`, representing calendar dates.

This vignette explains how timepoints represented by numeric values in data may easily be converted to class `"Date"`, using the R **base** package function `as.Date()`.

## 2 Convert numeric values representing year to class "Date"

We start by creating an example data frame of longitudinal data for three subjects, containing years 2018 to 2025 as numeric values and with observations having one of three possible ordinal values: –

```
> (df <- data.frame(
  subject = rep(1001:1003),
  timepoint = rep(2018:2025, each = 3),
  result = gl(3, 4, lab = c("good", "bad", "ugly"), ordered = TRUE)
))
```

	subject	timepoint	result
1	1001	2018	good
2	1002	2018	good
3	1003	2018	good
4	1001	2019	good
5	1002	2019	bad
6	1003	2019	bad
7	1001	2020	bad
8	1002	2020	bad
9	1003	2020	ugly
10	1001	2021	ugly
11	1002	2021	ugly
12	1003	2021	ugly
13	1001	2022	good
14	1002	2022	good
15	1003	2022	good
16	1001	2023	good
17	1002	2023	bad
18	1003	2023	bad
19	1001	2024	bad
20	1002	2024	bad
21	1003	2024	ugly
22	1001	2025	ugly
23	1002	2025	ugly
24	1003	2025	ugly

We convert the numeric values for year in the `timepoint` column to class "Date" using `as.Date()`, with consistent arbitrary values of January 1st for month and day: –

```
> (df <- transform(
  df,
  timepoint = as.Date(paste(timepoint, "01", "01", sep = "-"))
))
```

	subject	timepoint	result
1	1001	2018-01-01	good
2	1002	2018-01-01	good
3	1003	2018-01-01	good
4	1001	2019-01-01	good
5	1002	2019-01-01	bad
6	1003	2019-01-01	bad
7	1001	2020-01-01	bad
8	1002	2020-01-01	bad
9	1003	2020-01-01	ugly
10	1001	2021-01-01	ugly
11	1002	2021-01-01	ugly
12	1003	2021-01-01	ugly
13	1001	2022-01-01	good
14	1002	2022-01-01	good
15	1003	2022-01-01	good
16	1001	2023-01-01	good
17	1002	2023-01-01	bad

```

18  1003 2023-01-01  bad
19  1001 2024-01-01  bad
20  1002 2024-01-01  bad
21  1003 2024-01-01  ugly
22  1001 2025-01-01  ugly
23  1002 2025-01-01  ugly
24  1003 2025-01-01  ugly

```

We can now use the `add_prev_result()` function with default values for all but its first argument object—a `data.frame` (or a subclass thereof)— to add a column of results from the previous observation: –

```

> (df <- add_prev_result(df))

  subject  timepoint result prev_result
1     1001 2018-01-01  good          <NA>
2     1002 2018-01-01  good          <NA>
3     1003 2018-01-01  good          <NA>
4     1001 2019-01-01  good          good
5     1002 2019-01-01  bad           good
6     1003 2019-01-01  bad           good
7     1001 2020-01-01  bad           good
8     1002 2020-01-01  bad           bad
9     1003 2020-01-01  ugly          bad
10    1001 2021-01-01  ugly          bad
11    1002 2021-01-01  ugly          bad
12    1003 2021-01-01  ugly          ugly
13    1001 2022-01-01  good          ugly
14    1002 2022-01-01  good          ugly
15    1003 2022-01-01  good          ugly
16    1001 2023-01-01  good          good
17    1002 2023-01-01  bad           good
18    1003 2023-01-01  bad           good
19    1001 2024-01-01  bad           good
20    1002 2024-01-01  bad           bad
21    1003 2024-01-01  ugly          bad
22    1001 2025-01-01  ugly          bad
23    1002 2025-01-01  ugly          bad
24    1003 2025-01-01  ugly          ugly

```

Finally, we can format the class "Date" `timepoint` column to show just the year, as in the original data: –

```

> transform(df, timepoint = format(timepoint, "%Y"))

```

	subject	timepoint	result	prev_result
1	1001	2018	good	<NA>
2	1002	2018	good	<NA>
3	1003	2018	good	<NA>
4	1001	2019	good	good
5	1002	2019	bad	good
6	1003	2019	bad	good
7	1001	2020	bad	good
8	1002	2020	bad	bad
9	1003	2020	ugly	bad
10	1001	2021	ugly	bad
11	1002	2021	ugly	bad
12	1003	2021	ugly	ugly
13	1001	2022	good	ugly
14	1002	2022	good	ugly
15	1003	2022	good	ugly
16	1001	2023	good	good
17	1002	2023	bad	good
18	1003	2023	bad	good
19	1001	2024	bad	good
20	1002	2024	bad	bad
21	1003	2024	ugly	bad
22	1001	2025	ugly	bad
23	1002	2025	ugly	bad
24	1003	2025	ugly	ugly

### 3 Convert numeric values representing year and month to class "Date"

We create another example data frame of longitudinal data for two subjects, containing year and month from July 2024 to June 2025 as numeric values, and with observations having one of two possible ordinal values: –

```
> (df <- data.frame(
  subject = 1001:1002,
  year = rep(2024:2025, each = 12),
  month = rep(c(7:12, 1:6), each = 2),
  result = gl(2, 3, lab = c("low", "high"), ordered = TRUE)
))
```

	subject	year	month	result
1	1001	2024	7	low
2	1002	2024	7	low
3	1001	2024	8	low
4	1002	2024	8	high
5	1001	2024	9	high
6	1002	2024	9	high
7	1001	2024	10	low
8	1002	2024	10	low
9	1001	2024	11	low

10	1002	2024	11	high
11	1001	2024	12	high
12	1002	2024	12	high
13	1001	2025	1	low
14	1002	2025	1	low
15	1001	2025	2	low
16	1002	2025	2	high
17	1001	2025	3	high
18	1002	2025	3	high
19	1001	2025	4	low
20	1002	2025	4	low
21	1001	2025	5	low
22	1002	2025	5	high
23	1001	2025	6	high
24	1002	2025	6	high

We convert the numeric values for year and month to class "Date" using `as.Date()`, with a consistent arbitrary value of 1st for day of the month: –

```
> (df <- transform(
  df,
  timepoint = as.Date(paste(year, month, "01", sep = "-")),
  year = NULL,
  month = NULL
))
```

	subject	result	timepoint
1	1001	low	2024-07-01
2	1002	low	2024-07-01
3	1001	low	2024-08-01
4	1002	high	2024-08-01
5	1001	high	2024-09-01
6	1002	high	2024-09-01
7	1001	low	2024-10-01
8	1002	low	2024-10-01
9	1001	low	2024-11-01
10	1002	high	2024-11-01
11	1001	high	2024-12-01
12	1002	high	2024-12-01
13	1001	low	2025-01-01
14	1002	low	2025-01-01
15	1001	low	2025-02-01
16	1002	high	2025-02-01
17	1001	high	2025-03-01
18	1002	high	2025-03-01
19	1001	low	2025-04-01
20	1002	low	2025-04-01
21	1001	low	2025-05-01
22	1002	high	2025-05-01
23	1001	high	2025-06-01
24	1002	high	2025-06-01

We can now use the `add_transitions()` function with default values for all but the first argument to add a column of transitions: –

```
> (df <- add_transitions(df))
```

	subject	result	timepoint	transition
1	1001	low	2024-07-01	NA
2	1002	low	2024-07-01	NA
3	1001	low	2024-08-01	0
4	1002	high	2024-08-01	1
5	1001	high	2024-09-01	1
6	1002	high	2024-09-01	0
7	1001	low	2024-10-01	-1
8	1002	low	2024-10-01	-1
9	1001	low	2024-11-01	0
10	1002	high	2024-11-01	1
11	1001	high	2024-12-01	1
12	1002	high	2024-12-01	0
13	1001	low	2025-01-01	-1
14	1002	low	2025-01-01	-1
15	1001	low	2025-02-01	0
16	1002	high	2025-02-01	1
17	1001	high	2025-03-01	1
18	1002	high	2025-03-01	0
19	1001	low	2025-04-01	-1
20	1002	low	2025-04-01	-1
21	1001	low	2025-05-01	0
22	1002	high	2025-05-01	1
23	1001	high	2025-06-01	1
24	1002	high	2025-06-01	0

Finally, we can format the class "Date" timepoint column to show just the month and year, as in the original data: –

```
> transform(df, timepoint = format(timepoint, "%b-%Y"))
```

	subject	result	timepoint	transition
1	1001	low	Jul-2024	NA
2	1002	low	Jul-2024	NA
3	1001	low	Aug-2024	0
4	1002	high	Aug-2024	1
5	1001	high	Sep-2024	1
6	1002	high	Sep-2024	0
7	1001	low	Oct-2024	-1
8	1002	low	Oct-2024	-1
9	1001	low	Nov-2024	0
10	1002	high	Nov-2024	1
11	1001	high	Dec-2024	1
12	1002	high	Dec-2024	0
13	1001	low	Jan-2025	-1
14	1002	low	Jan-2025	-1

15	1001	low	Feb-2025	0
16	1002	high	Feb-2025	1
17	1001	high	Mar-2025	1
18	1002	high	Mar-2025	0
19	1001	low	Apr-2025	-1
20	1002	low	Apr-2025	-1
21	1001	low	May-2025	0
22	1002	high	May-2025	1
23	1001	high	Jun-2025	1
24	1002	high	Jun-2025	0

## 4 Convert numeric values representing ages to class "Date"

We inspect the first 22 rows of the Blackmore data, which includes numeric values for age in years rather than dates: –

```
> head(Blackmore, 22)
```

	subject	age	exercise	group
1	100	8.00	2.71	patient
2	100	10.00	1.94	patient
3	100	12.00	2.36	patient
4	100	14.00	1.54	patient
5	100	15.92	8.63	patient
6	101	8.00	0.14	patient
7	101	10.00	0.14	patient
8	101	12.00	0.00	patient
9	101	14.00	0.00	patient
10	101	16.67	5.08	patient
11	102	8.00	0.92	patient
12	102	10.00	1.82	patient
13	102	12.00	4.75	patient
14	102	15.08	24.72	patient
15	103	8.00	1.04	patient
16	103	10.00	2.90	patient
17	103	12.00	2.65	patient
18	103	14.08	6.86	patient
19	104	8.00	2.75	patient
20	104	10.00	6.62	patient
21	104	12.00	0.29	patient
22	104	15.42	12.37	patient

We shall use the `add_prev_date()` function to add a column of previous test dates. For the `timepoint` argument, we convert the age values to class "Date" using `as.Date()` and an arbitrary "origin" of 1st January 2000<sup>1</sup>, to which we add the age in days<sup>2</sup> calculated as `365.25 * age` (in years).

```
> Blackmore <- transform(
  Blackmore,
  timepoint = as.Date("2000-01-01") + round(365.25 * age)
)
> head(Blackmore, 22)
```

	subject	age	exercise	group	timepoint
1	100	8.00	2.71	patient	2008-01-01
2	100	10.00	1.94	patient	2009-12-31
3	100	12.00	2.36	patient	2012-01-01
4	100	14.00	1.54	patient	2014-01-01
5	100	15.92	8.63	patient	2015-12-03
6	101	8.00	0.14	patient	2008-01-01
7	101	10.00	0.14	patient	2009-12-31
8	101	12.00	0.00	patient	2012-01-01
9	101	14.00	0.00	patient	2014-01-01
10	101	16.67	5.08	patient	2016-09-02
11	102	8.00	0.92	patient	2008-01-01
12	102	10.00	1.82	patient	2009-12-31
13	102	12.00	4.75	patient	2012-01-01
14	102	15.08	24.72	patient	2015-01-30
15	103	8.00	1.04	patient	2008-01-01
16	103	10.00	2.90	patient	2009-12-31
17	103	12.00	2.65	patient	2012-01-01
18	103	14.08	6.86	patient	2014-01-30
19	104	8.00	2.75	patient	2008-01-01
20	104	10.00	6.62	patient	2009-12-31
21	104	12.00	0.29	patient	2012-01-01
22	104	15.42	12.37	patient	2015-06-03

To use the `add_prev_date()` function, we need to provide a `result` argument in one of two permissible formats—an ordered factor, or binary data with values of either 1 or 0; note that the `exercise` column is neither of these. Since we shall not be using the values of the `exercise` column for this demonstration, we simply add a dummy `result` column with values all integer 0: –

```
> Blackmore <- transform(Blackmore, result = 0L)
```

We can now use `add_prev_date()` with default values for all but the first argument: –

```
> Blackmore <- add_prev_date(Blackmore)
```

---

<sup>1</sup>This is equivalent to assuming all subjects were born on the 1st January 2000, which is permissible so long as these dates are not used for any purpose other than that shown here.

<sup>2</sup>This works because class "Date" is represented internally in days and has a method for the `+` operator that returns a date.

```
> head(Blackmore, 22)
```

	subject	age	exercise	group	timepoint	result	prev_date
1	100	8.00	2.71	patient	2008-01-01	0	<NA>
2	100	10.00	1.94	patient	2009-12-31	0	2008-01-01
3	100	12.00	2.36	patient	2012-01-01	0	2009-12-31
4	100	14.00	1.54	patient	2014-01-01	0	2012-01-01
5	100	15.92	8.63	patient	2015-12-03	0	2014-01-01
6	101	8.00	0.14	patient	2008-01-01	0	<NA>
7	101	10.00	0.14	patient	2009-12-31	0	2008-01-01
8	101	12.00	0.00	patient	2012-01-01	0	2009-12-31
9	101	14.00	0.00	patient	2014-01-01	0	2012-01-01
10	101	16.67	5.08	patient	2016-09-02	0	2014-01-01
11	102	8.00	0.92	patient	2008-01-01	0	<NA>
12	102	10.00	1.82	patient	2009-12-31	0	2008-01-01
13	102	12.00	4.75	patient	2012-01-01	0	2009-12-31
14	102	15.08	24.72	patient	2015-01-30	0	2012-01-01
15	103	8.00	1.04	patient	2008-01-01	0	<NA>
16	103	10.00	2.90	patient	2009-12-31	0	2008-01-01
17	103	12.00	2.65	patient	2012-01-01	0	2009-12-31
18	103	14.08	6.86	patient	2014-01-30	0	2012-01-01
19	104	8.00	2.75	patient	2008-01-01	0	<NA>
20	104	10.00	6.62	patient	2009-12-31	0	2008-01-01
21	104	12.00	0.29	patient	2012-01-01	0	2009-12-31
22	104	15.42	12.37	patient	2015-06-03	0	2012-01-01

Finally, to be consistent with the original data, we can calculate a `prev_age`<sup>3</sup> column from the `prev_date` column, which itself can be removed along with the now superfluous `timepoint` and `result` columns: –

```
> Blackmore <- transform(
  Blackmore,
  prev_age = round(
    as.integer(prev_date - as.Date("2000-01-01")) / 365.25, 2
  ),
  timepoint = NULL, result = NULL, prev_date = NULL
)
> head(Blackmore, 22)
```

	subject	age	exercise	group	prev_age
1	100	8.00	2.71	patient	NA
2	100	10.00	1.94	patient	8
3	100	12.00	2.36	patient	10
4	100	14.00	1.54	patient	12
5	100	15.92	8.63	patient	14
6	101	8.00	0.14	patient	NA
7	101	10.00	0.14	patient	8

<sup>3</sup>Note that by default, R formats the `age` column to two decimal places because some ages in the `Blackmore` dataset are not whole numbers of years. These non-whole number ages are always the last observation for an individual. Consequently, all “previous ages” are indeed whole numbers and R formats the `prev_age` column without showing any decimal places.

8	101	12.00	0.00	patient	10
9	101	14.00	0.00	patient	12
10	101	16.67	5.08	patient	14
11	102	8.00	0.92	patient	NA
12	102	10.00	1.82	patient	8
13	102	12.00	4.75	patient	10
14	102	15.08	24.72	patient	12
15	103	8.00	1.04	patient	NA
16	103	10.00	2.90	patient	8
17	103	12.00	2.65	patient	10
18	103	14.08	6.86	patient	12
19	104	8.00	2.75	patient	NA
20	104	10.00	6.62	patient	8
21	104	12.00	0.29	patient	10
22	104	15.42	12.37	patient	12