A common task in data analysis is to *merge* or *join* two tables according to shared *keys* or values. The operation is perhaps most commonly associated with relational databases and structured query language (SQL) but it’s just as useful in R with data frames.

Most joins are *equi-joins*, matching rows according to two columns having exactly equal values. These are easy to perfom in R using the base merge() function, the various join() functions in **dplyr** and the X[i] syntax of **data.table**.

But sometimes we need *non-equi joins* or \(\theta\)-joins, where the matching condition is an interval or a set of inequalities. Other situations call for a *rolling join*, used to link records according to their proximity in a time sequence.

How do you perform non-equi joins and rolling joins in R?

# Motivating example

A famous YouTuber is testing out a new marketing strategy, promoting specific videos on social media. They collect daily view counts of old videos and want to summarise these counts in the days following each promotion.

The tables promos and views are as follows. The data span a two-week period, but for certain days and certain videos, the view counts are missing.

## video promo\_date

|  |  |  |
| --- | --- | --- |
| A | 2020-04-01 | |
| A | 2020-04-07 | |
| B | 2020-04-03 | |
| B | 2020-04-08 | |
| C | 2020-04-03 | |
| **video view\_date views** | | |
| A | 2020-04-01 | 992 |
| A | 2020-04-02 | 3304 |
| A | 2020-04-03 | 1417 |
| A | 2020-04-04 | 191 |
| A | 2020-04-05 | 2366 |
| A | 2020-04-06 | 7318 |
| A | 2020-04-07 | 1570 |
| A | 2020-04-08 | 2051 |
| A | 2020-04-09 | 5958 |
| A | 2020-04-10 | 3574 |
| A | 2020-04-11 | 6724 |
| A | 2020-04-12 | 12043 |
| A | 2020-04-13 | 481 |
| A | 2020-04-14 | 286 |
| B | 2020-04-01 | 6270 |
| B | 2020-04-02 | 1549 |

**video view\_date views**

|  |  |  |
| --- | --- | --- |
| B | 2020-04-03 | 2376 |
| B | 2020-04-04 | 3111 |
| B | 2020-04-05 | 6228 |
| B | 2020-04-06 | 1852 |
| B | 2020-04-07 | 24314 |
| B | 2020-04-08 | 3329 |
| B | 2020-04-09 | 24980 |
| B | 2020-04-10 | 1118 |
| B | 2020-04-11 | 6057 |
| B | 2020-04-14 | 1400 |
| C | 2020-04-01 | 1156 |
| C | 2020-04-02 | 6435 |
| C | 2020-04-03 | 2847 |
| C | 2020-04-04 | 15093 |
| C | 2020-04-05 | 2488 |
| C | 2020-04-06 | 1773 |
| C | 2020-04-07 | 8782 |
| C | 2020-04-08 | 3687 |
| C | 2020-04-11 | 1963 |
| C | 2020-04-12 | 9510 |
| C | 2020-04-13 | 3891 |
| C | 2020-04-14 | 3282 |

What are the mean view counts on videos in the three days immediately following promotions for those videos?

I will show how you might accomplish this task using either non-equi joins or rolling joins in R.

# Crossing + filter with dplyr

The package **dplyr** has no function for joining on anything other than an equality relation. However, you can get the same results (possibly less efficiently) using an outer join or Cartesian product, followed by a filtering operation.

library(dplyr) views %>%

# Crossing full\_join(promos) %>% # Filter

filter(view\_date >= promo\_date, view\_date <= promo\_date + 3) %>%

# Aggregate group\_by(video) %>%

summarise(mean(views), sd(views), `n days` = n())

## video mean(views) sd(views) n days

A 2382 1832 8

## video mean(views) sd(views) n days

|  |  |  |  |
| --- | --- | --- | --- |
| B | 6131 | 7836 | 8 |
| C | 5550 | 6377 | 4 |

It would also be possible to compare promoted days with non-promoted days for each video, either by creating a binary indicator (instead of a filter) or by rejoining the table with the original dataset after the filter step.

# Non-equi joins with sqldf

The **sqldf** package lets you query R data frames with SQL, as if you were working with a relational database. The result of the query is another data frame, and performance is sometimes better than equivalent R functions.

library(sqldf)

sqldf('SELECT v.video, view\_date, views FROM views v

JOIN promos p

ON v.video = p.video AND

view\_date BETWEEN promo\_date AND promo\_date + 3'

) %>%

group\_by(video) %>%

summarise(mean(views), sd(views), `n days` = n())

## video mean(views) sd(views) n days

|  |  |  |  |
| --- | --- | --- | --- |
| A | 2382 | 1832 | 8 |
| B | 6131 | 7836 | 8 |
| C | 5550 | 6377 | 4 |

The aggregation step could also be written in SQL, but it makes sense only to use SQL where it is absolutely needed, and to use native R functions for everything else.

Having access to this functionality is very powerful, but has the obvious disadvantage that you need to learn a bit of SQL to understand the syntax.

# Non-equi joins with data.table

The high-performance data manipulation package **data.table** now (as of v1.9.8) supports non- equi joins.

Non-equi joins are made possible with the X[i] merging syntax and the on argument. It’s slightly less flexible than the equivalent SQL, because you can’t just write promo\_date + 3 in the inequality: instead it needs to be an explicit column in the table. You also can’t use the infix

%between% operator, so two inequalities have to do instead. Otherwise the syntax is similar. Like in SQL, a prefix is used to disambiguate the column names: here it’s x.name.

library(data.table) setDT(views)

setDT(promos)[, promo\_end := promo\_date + 3]

## video promo\_date promo\_end

A 2020-04-01 2020-04-04

## video promo\_date promo\_end

|  |  |  |
| --- | --- | --- |
| A | 2020-04-07 | 2020-04-10 |
| B | 2020-04-03 | 2020-04-06 |
| B | 2020-04-08 | 2020-04-11 |
| C | 2020-04-03 | 2020-04-06 |

views[promos,

.(video, views, x.view\_date), # Non equi join:

on = .(video,

view\_date >= promo\_date, view\_date <= promo\_end)

][, # Chain into aggregate:

.(mean = mean(views), sd = sd(views), .N), by = video]

**video mean sd N**

|  |  |  |
| --- | --- | --- |
| A | 2382 1832 | 8 |
| B | 6131 7836 | 8 |
| C | 5550 6377 | 4 |

# Rolling joins with data.table

This particular example, since it involves a time variable, is even simpler using a *rolling join*. The concept is a bit confusing, but essentially it attributes records in one table with the most recent preceding records in the second table.

When performing rolling joins in **data.table**, one of the joining time columns gets dropped, which can make it hard to identify your records if they don’t have an explicit ID. To mitigate this, we will copy each date column to the name join\_date and join on that.

views[, join\_date := view\_date] promos[, join\_date := promo\_date]

setkey(views, video, join\_date) setkey(promos, video, join\_date) promos[views, roll = TRUE]

## video promo\_date join\_date view\_date views

|  |  |  |  |
| --- | --- | --- | --- |
| A | 2020-04-01 | 2020-04-01 2020-04-01 | 992 |
| A | 2020-04-01 | 2020-04-02 2020-04-02 | 3304 |
| A | 2020-04-01 | 2020-04-03 2020-04-03 | 1417 |
| A | 2020-04-01 | 2020-04-04 2020-04-04 | 191 |
| A | 2020-04-01 | 2020-04-05 2020-04-05 | 2366 |
| A | 2020-04-01 | 2020-04-06 2020-04-06 | 7318 |
| A | 2020-04-07 | 2020-04-07 2020-04-07 | 1570 |
| A | 2020-04-07 | 2020-04-08 2020-04-08 | 2051 |
| A | 2020-04-07 | 2020-04-09 2020-04-09 | 5958 |

**video promo\_date join\_date view\_date views**

|  |  |  |  |
| --- | --- | --- | --- |
| A | 2020-04-07 | 2020-04-10 2020-04-10 | 3574 |
| A | 2020-04-07 | 2020-04-11 2020-04-11 | 6724 |
| A | 2020-04-07 | 2020-04-12 2020-04-12 | 12043 |
| A | 2020-04-07 | 2020-04-13 2020-04-13 | 481 |
| A | 2020-04-07 | 2020-04-14 2020-04-14 | 286 |
| B | NA | 2020-04-01 2020-04-01 | 6270 |
| B | NA | 2020-04-02 2020-04-02 | 1549 |
| B | 2020-04-03 | 2020-04-03 2020-04-03 | 2376 |
| B | 2020-04-03 | 2020-04-04 2020-04-04 | 3111 |
| B | 2020-04-03 | 2020-04-05 2020-04-05 | 6228 |
| B | 2020-04-03 | 2020-04-06 2020-04-06 | 1852 |
| B | 2020-04-03 | 2020-04-07 2020-04-07 | 24314 |
| B | 2020-04-08 | 2020-04-08 2020-04-08 | 3329 |
| B | 2020-04-08 | 2020-04-09 2020-04-09 | 24980 |
| B | 2020-04-08 | 2020-04-10 2020-04-10 | 1118 |
| B | 2020-04-08 | 2020-04-11 2020-04-11 | 6057 |
| B | 2020-04-08 | 2020-04-14 2020-04-14 | 1400 |
| C | NA | 2020-04-01 2020-04-01 | 1156 |
| C | NA | 2020-04-02 2020-04-02 | 6435 |
| C | 2020-04-03 | 2020-04-03 2020-04-03 | 2847 |
| C | 2020-04-03 | 2020-04-04 2020-04-04 | 15093 |
| C | 2020-04-03 | 2020-04-05 2020-04-05 | 2488 |
| C | 2020-04-03 | 2020-04-06 2020-04-06 | 1773 |
| C | 2020-04-03 | 2020-04-07 2020-04-07 | 8782 |
| C | 2020-04-03 | 2020-04-08 2020-04-08 | 3687 |
| C | 2020-04-03 | 2020-04-11 2020-04-11 | 1963 |
| C | 2020-04-03 | 2020-04-12 2020-04-12 | 9510 |
| C | 2020-04-03 | 2020-04-13 2020-04-13 | 3891 |
| C | 2020-04-03 | 2020-04-14 2020-04-14 | 3282 |

The syntax promos[views, roll=TRUE] means “which promotion immediately preceded each viewing date?” Conversely, views[promos, roll=TRUE] means “which viewing dates immediately preceded each promotion?”

In this case, we want something *close* to the former, but we’re only interested in promos in the past 3 days, whereas by default it’ll extend back into the depths of time looking for the last one, regardless of how long ago.

By changing roll = TRUE to roll = 3 the join will fail to match when the join\_date differs by more than three days between the two tables. If we wanted to go in the opposite direction in time, we could use roll = -Inf to search forwards in time for future promotions, and roll =

-3 for only those in the following three days.

promos[views, roll = 3]

## video promo\_date join\_date view\_date views

|  |  |  |  |
| --- | --- | --- | --- |
| A | 2020-04-01 | 2020-04-01 2020-04-01 | 992 |
| A | 2020-04-01 | 2020-04-02 2020-04-02 | 3304 |
| A | 2020-04-01 | 2020-04-03 2020-04-03 | 1417 |
| A | 2020-04-01 | 2020-04-04 2020-04-04 | 191 |
| A | NA | 2020-04-05 2020-04-05 | 2366 |
| A | NA | 2020-04-06 2020-04-06 | 7318 |
| A | 2020-04-07 | 2020-04-07 2020-04-07 | 1570 |
| A | 2020-04-07 | 2020-04-08 2020-04-08 | 2051 |
| A | 2020-04-07 | 2020-04-09 2020-04-09 | 5958 |
| A | 2020-04-07 | 2020-04-10 2020-04-10 | 3574 |
| A | NA | 2020-04-11 2020-04-11 | 6724 |
| A | NA | 2020-04-12 2020-04-12 | 12043 |
| A | NA | 2020-04-13 2020-04-13 | 481 |
| A | NA | 2020-04-14 2020-04-14 | 286 |
| B | NA | 2020-04-01 2020-04-01 | 6270 |
| B | NA | 2020-04-02 2020-04-02 | 1549 |
| B | 2020-04-03 | 2020-04-03 2020-04-03 | 2376 |
| B | 2020-04-03 | 2020-04-04 2020-04-04 | 3111 |
| B | 2020-04-03 | 2020-04-05 2020-04-05 | 6228 |
| B | 2020-04-03 | 2020-04-06 2020-04-06 | 1852 |
| B | NA | 2020-04-07 2020-04-07 | 24314 |
| B | 2020-04-08 | 2020-04-08 2020-04-08 | 3329 |
| B | 2020-04-08 | 2020-04-09 2020-04-09 | 24980 |
| B | 2020-04-08 | 2020-04-10 2020-04-10 | 1118 |
| B | 2020-04-08 | 2020-04-11 2020-04-11 | 6057 |
| B | NA | 2020-04-14 2020-04-14 | 1400 |
| C | NA | 2020-04-01 2020-04-01 | 1156 |
| C | NA | 2020-04-02 2020-04-02 | 6435 |
| C | 2020-04-03 | 2020-04-03 2020-04-03 | 2847 |
| C | 2020-04-03 | 2020-04-04 2020-04-04 | 15093 |
| C | 2020-04-03 | 2020-04-05 2020-04-05 | 2488 |
| C | 2020-04-03 | 2020-04-06 2020-04-06 | 1773 |
| C | NA | 2020-04-07 2020-04-07 | 8782 |
| C | NA | 2020-04-08 2020-04-08 | 3687 |
| C | NA | 2020-04-11 2020-04-11 | 1963 |
| C | NA | 2020-04-12 2020-04-12 | 9510 |
| C | NA | 2020-04-13 2020-04-13 | 3891 |
| C | NA | 2020-04-14 2020-04-14 | 3282 |

Since the default in **data.table**’s X[i] merge syntax is nomatch = NA, we get all of the views back, with the column promo\_date equal to the date of the last promotion (in the last three days), or NA if no such promotion was found. If we set nomatch = 0 then these non-matching values are dropped from the result.

So the full operation to calculate the summary figures is

promos[views, roll = 3, nomatch = 0

][j = .(mean = mean(views), sd = sd(views), .N), by = video]

## video mean sd N

|  |  |  |
| --- | --- | --- |
| A | 2382 1832 | 8 |
| B | 6131 7836 | 8 |
| C | 5550 6377 | 4 |

If the intervals needed to be different lengths for each of the campaigns (i.e. not all equal to 3), then you would probably want a non-equi join rather than a rolling join in this case.

# Extensions

Arguably, for these examples you’d want to compare the view counts in promotional periods with those outside promotional periods. This does not require a special kind of join; rather you perform the non-equi or rolling join as above and then wrangle the output accordingly.

If the non-matching rows are filtered out, you need to re-join with the original dataset. Otherwise, you need to use some sort of indicator variable for whether the viewing date falls within a promotional period or not.

In **dplyr**:

promos %>% full\_join(views) %>%

mutate(promo = between(view\_date - promo\_date, 0, 3)) %>% group\_by(video, view\_date) %>%

summarise(promo = any(promo),

views = unique(views)) %>% group\_by(promo) %>%

summarise(mean(views), sd(views), n = n())

## promo mean(views) sd(views) n

|  |  |  |
| --- | --- | --- |
| FALSE | 5637 | 5772 18 |
| TRUE  In **data.table**: | 4515 | 5790 20 |

promos[views, roll = 3][

j = .(mean = mean(views), sd = sd(views), .N), by = .(promo = !is.na(promo\_date))]

## promo mean sd N

TRUE 4515 5790 20

FALSE 5637 5772 18