**Introduction**

Performing split-apply-combine operations on R data.frames and how this functionality was used within poorman – a base R replica of dplyr. In this post I’d like to talk about replicating dplyr’s join and filter join functionality, again using nothing but base.

**Joins**

First of all, let’s set up some data to use for our examples. Here I create two data.frames; one containing heights associated with names and IDs; the second containing weights also associated with some names and IDs. Some of these ID and name combinations are common to both tables (e.g. c("id" = 5, "name" = "C")) but other IDs are unique to one table or the other (e.g. c("id" = 1, "name" = "A")).

Note that sometimes a list of elements (here name and ID) are somtimes referred to as a tuple.

table1 <- data.frame(

id = c(1, 3, 5, 7, 9),

name = LETTERS[1:5],

height = c(1, 2, 2, 2, 2),

stringsAsFactors = FALSE

)

table1

# id name height

# 1 1 A 1

# 2 3 B 2

# 3 5 C 2

# 4 7 D 2

# 5 9 E 2

table2 <- data.frame(

id = c(2, 4, 5, 7),

name = LETTERS[1:4],

weight = c(2, 3, 4, 5),

stringsAsFactors = FALSE

)

table2

# id name weight

# 1 2 A 2

# 2 4 B 3

# 3 5 C 4

# 4 7 D 5

When working with data such as this, we may wish to join these two tables together such that we can more easily use data from both. In base, the function we use for joining tables together is called merge(). We will see how we can use this one function to perform inner, left, right and full joins.

**Inner Join**

An inner join selects records that have matching values in both tables within the columns we are joining by, returning all columns. For table1 and table2, we will be joining the tables by "id" and "name" since these are the common columns between both tables.

Note that depending on your circumstance you may not wish to join on all common columns. For example you may have two tables that have columns with the same name but actually contain different data.

When we perform the inner join therefore, we will expect a data.frame containing the records c("id" = 5, "name" = "C") and c("id" = 7, "name" = "D") since these records exist in both tables.

To perform an inner join, we use the merge() function with no additional parameters, giving the two tables to merge.

merge(x = table1, y = table2)

# id name height weight

# 1 5 C 2 4

# 2 7 D 2 5

When performing a join, we typically specify which columns we would like to join on. merge() is helpful in that it works out the common columns between the datasets with the following command.

by <- intersect(names(table1), names(table2))

by

# [1] "id" "name"

If we wanted, we could pass the column names explicitly.

merge(x = table1, y = table2, by = c("id", "name"))

# id name height weight

# 1 5 C 2 4

# 2 7 D 2 5

In poorman, we can use the inner\_join() function to perform this type of join.

table1 %>% inner\_join(table2)

# Joining, by = c("id", "name")

# id name height weight

# 1 5 C 2 4

# 2 7 D 2 5

**Left Join**

A left join will include all rows from table1 regardless of whether or not there is a matching record in table2. For those records which do not have a match, the left join will leave the cells as NA values. In base, we perform this type of join with the merge() function, specifying all.x = TRUE.

merge(x = table1, y = table2, all.x = TRUE)

# id name height weight

# 1 1 A 1 NA

# 2 3 B 2 NA

# 3 5 C 2 4

# 4 7 D 2 5

# 5 9 E 2 NA

In poorman, this type of join can be performed with left\_join().

table1 %>% left\_join(table2)

# Joining, by = c("id", "name")

# id name height weight

# 1 1 A 1 NA

# 2 3 B 2 NA

# 3 5 C 2 4

# 4 7 D 2 5

# 5 9 E 2 NA

**Right Join**

A right join is the opposite of a left join. Here, the merge() function returns all rows from table2 and the matched rows from table1; again filling any blank cells with NA. This time, we specify all.y = TRUE.

merge(x = table1, y = table2, all.y = TRUE)

# id name height weight

# 1 2 A NA 2

# 2 4 B NA 3

# 3 5 C 2 4

# 4 7 D 2 5

In poorman, this type of join can be performed with, you guessed it, right\_join().

table1 %>% right\_join(table2)

# Joining, by = c("id", "name")

# id name height weight

# 1 2 A NA 2

# 2 4 B NA 3

# 3 5 C 2 4

# 4 7 D 2 5

**Full Join**

A full join will join together all rows from two tables. Like the left and right joins, any rows that do not contain a match will contain NA in the cells that are missing. Here we specify all = TRUE within merge() to represent a full join.

merge(x = table1, y = table2, all = TRUE)

# id name height weight

# 1 1 A 1 NA

# 2 2 A NA 2

# 3 3 B 2 NA

# 4 4 B NA 3

# 5 5 C 2 4

# 6 7 D 2 5

# 7 9 E 2 NA

Finally poorman provides full\_join() to do this type of join.

table1 %>% full\_join(table2)

# Joining, by = c("id", "name")

# id name height weight

# 1 1 A 1 NA

# 2 2 A NA 2

# 3 3 B 2 NA

# 4 4 B NA 3

# 5 5 C 2 4

# 6 7 D 2 5

# 7 9 E 2 NA

**Filter Joins**

For filter joins we will look at some slightly different data.

table1 <- data.frame(

pupil = rep(1:3, each = 2),

test = rep(c("A", "B"), 3),

score = c(60, 70, 65, 80, 85, 70),

stringsAsFactors = FALSE

)

table1

# pupil test score

# 1 1 A 60

# 2 1 B 70

# 3 2 A 65

# 4 2 B 80

# 5 3 A 85

# 6 3 B 70

table2 <- table1[c(1, 3, 4), ]

table2

# pupil test score

# 1 1 A 60

# 3 2 A 65

# 4 2 B 80

**Semi-Join**

A semi-join is slightly different to the other types of joins we have seen thus far. We describe a semi-join as a “filter join”, since a semi-join returns the rows in table1 where the join column tuples in table1 are also found in table2. So we still specify the column names that we wish to “join” by (in this example it’s c("pupil", "test")), which is why this is considered a join but we are actually performing a sort of filter on table1.

by <- c("pupil", "test")

Since we are looking for the rows in table1 that are also in table2, we will be using [.data.frame to filter table1 by the matching rows. To do this, we will take advantage of base::interaction().

interaction(table1[, by])

# [1] 1.A 1.B 2.A 2.B 3.A 3.B

# Levels: 1.A 2.A 3.A 1.B 2.B 3.B

As you can see, what this function does is compute a vector of factor levels which represents the interaction of the given columns. So the number before the . is the pupil and the letter after the . is the test they took. If we do this for both tables, we can figure out which levels are %in% both tables. As we can see below, the results of %in% returns a logical vector that we can use to filter table1.

interaction(table2[, by])

# [1] 1.A 2.A 2.B

# Levels: 1.A 2.A 1.B 2.B

rows <- interaction(table1[, by]) %in% interaction(table2[, by])

rows

# [1] TRUE FALSE TRUE TRUE FALSE FALSE

So here we can see that levels 1.A, 2.A and 2.B appear in both table1 and table2. So now we can perform our filter.

table1[rows, ]

# pupil test score

# 1 1 A 60

# 3 2 A 65

# 4 2 B 80

Semi-join functionality is afforded by poorman using the semi\_join() function.

table1 %>% semi\_join(table2)

# Joining, by = c("pupil", "test", "score")

# pupil test score

# 1 1 A 60

# 2 2 A 65

# 3 2 B 80

**Anti-Join**

An anti-join is slightly different to a semi-join in that it returns all the rows from table1 that *do not* appear in table2 when “joining” on the join columns. Using our previously defined rows variable, we can take the inverse of this logical vector using !.

rows <- !rows

rows

# [1] FALSE TRUE FALSE FALSE TRUE TRUE

table1[rows, ]

# pupil test score

# 2 1 B 70

# 5 3 A 85

# 6 3 B 70

poorman provides a copy of anti-join functionality using anti\_join().

table1 %>% anti\_join(table2)

# Joining, by = c("pupil", "test", "score")

# pupil test score

# 1 1 B 70

# 2 3 A 85

# 3 3 B 70

**Conclusion**

As we can see, the join and filter join functionality provided by dplyr is also available in base, you just need to know the parameter details or how to perform the filters efficiently. For your convenience, these functions are now available in poorman using the same dplyr API: inner\_join(), left\_join(), right\_join(), full\_join(), semi\_join() and anti\_join(). See poorman::joins or ?poorman::filter\_joins for more details.