Lesson 5: Blending data from multiple files and sources

Adam Zabell

12/27/2017

# Blending Data

## Simple *rbind* and *cbind*

Sometimes, you need data stored across more than one file. For example, managing the QC deviations across twelve separate months of reports. Rather than hold multiple printouts side-by-side to monitor a trend, **R** will allow you to read each file and knit them together either by row, or by column. For most circumstances, that combination involves adding rows to data set.

workingDir <- file.path(rootDir,"class\_data")  
january <- read\_csv(file.path(workingDir,"2017-01-06.csv"))

## Parsed with column specification:  
## cols(  
## batchName = col\_character(),  
## sampleName = col\_character(),  
## compoundName = col\_character(),  
## ionRatio = col\_double(),  
## response = col\_double(),  
## concentration = col\_double(),  
## sampleType = col\_character(),  
## expectedConcentration = col\_integer(),  
## usedForCurve = col\_logical(),  
## samplePassed = col\_logical()  
## )

as.data.frame(january[187195:187200,])

## batchName sampleName compoundName ionRatio response concentration  
## 1 b208048 s048052 morphine 1.228035 1.2129827 164.84355  
## 2 b208048 s048052 hydromorphone 0.000000 0.0000000 0.00000  
## 3 b208048 s048052 oxymorphone 1.281230 0.4470845 65.80569  
## 4 b208048 s048052 codeine 1.203730 1.4222678 230.23143  
## 5 b208048 s048052 hydrocodone 0.000000 0.0000000 0.00000  
## 6 b208048 s048052 oxycodone 0.000000 0.0000000 0.00000  
## sampleType expectedConcentration usedForCurve samplePassed  
## 1 unknown 0 FALSE FALSE  
## 2 unknown 0 FALSE TRUE  
## 3 unknown 0 FALSE TRUE  
## 4 unknown 0 FALSE TRUE  
## 5 unknown 0 FALSE TRUE  
## 6 unknown 0 FALSE TRUE

february <- read\_csv(file.path(workingDir,"2017-02-06.csv"))

## Parsed with column specification:  
## cols(  
## batchName = col\_character(),  
## sampleName = col\_character(),  
## compoundName = col\_character(),  
## ionRatio = col\_double(),  
## response = col\_double(),  
## concentration = col\_double(),  
## sampleType = col\_character(),  
## expectedConcentration = col\_integer(),  
## usedForCurve = col\_logical(),  
## samplePassed = col\_logical()  
## )

as.data.frame(february[1:5,])

## batchName sampleName compoundName ionRatio response concentration  
## 1 b593231 s231001 morphine 0 0 0  
## 2 b593231 s231001 hydromorphone 0 0 0  
## 3 b593231 s231001 oxymorphone 0 0 0  
## 4 b593231 s231001 codeine 0 0 0  
## 5 b593231 s231001 hydrocodone 0 0 0  
## sampleType expectedConcentration usedForCurve samplePassed  
## 1 blank 0 FALSE FALSE  
## 2 blank 0 FALSE TRUE  
## 3 blank 0 FALSE TRUE  
## 4 blank 0 FALSE TRUE  
## 5 blank 0 FALSE TRUE

twoMonths <- rbind(january,february)  
twoMonths[187195:187204,]

## # A tibble: 10 x 10  
## batchName sampleName compoundName ionRatio response concentration  
## <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 b208048 s048052 morphine 1.228035 1.2129827 164.84355  
## 2 b208048 s048052 hydromorphone 0.000000 0.0000000 0.00000  
## 3 b208048 s048052 oxymorphone 1.281230 0.4470845 65.80569  
## 4 b208048 s048052 codeine 1.203730 1.4222678 230.23143  
## 5 b208048 s048052 hydrocodone 0.000000 0.0000000 0.00000  
## 6 b208048 s048052 oxycodone 0.000000 0.0000000 0.00000  
## 7 b593231 s231001 morphine 0.000000 0.0000000 0.00000  
## 8 b593231 s231001 hydromorphone 0.000000 0.0000000 0.00000  
## 9 b593231 s231001 oxymorphone 0.000000 0.0000000 0.00000  
## 10 b593231 s231001 codeine 0.000000 0.0000000 0.00000  
## # ... with 4 more variables: sampleType <chr>,  
## # expectedConcentration <int>, usedForCurve <lgl>, samplePassed <lgl>

c(nrow(january),nrow(february),nrow(twoMonths))

## [1] 187200 187200 374400

There is an equivalent but less useful command in **R** which will append columns.

incomplete\_data <- tibble(sampleName="123456",  
 compoundName=c("morphine","hydromorphone","codeine","hydrocodone"),  
 concentration=c(34,35,44,45))  
additional\_columns <- tibble(expectedConcentration=c(20,20,20,20),  
 sampleType="standard")  
incomplete\_data <- cbind(incomplete\_data,additional\_columns)  
incomplete\_data

## sampleName compoundName concentration expectedConcentration sampleType  
## 1 123456 morphine 34 20 standard  
## 2 123456 hydromorphone 35 20 standard  
## 3 123456 codeine 44 20 standard  
## 4 123456 hydrocodone 45 20 standard

There are better ways of adding per-column data. Creating a new named vector directly into the existing data.frame is straightforward and shown below, but a true merge of two distinct data.frames needs more explanation.

incomplete\_data$batchName <- "batch01"  
incomplete\_data

## sampleName compoundName concentration expectedConcentration sampleType  
## 1 123456 morphine 34 20 standard  
## 2 123456 hydromorphone 35 20 standard  
## 3 123456 codeine 44 20 standard  
## 4 123456 hydrocodone 45 20 standard  
## batchName  
## 1 batch01  
## 2 batch01  
## 3 batch01  
## 4 batch01

# Joining Relational Data

The database example for this class has three different data.frames: one for batch-level information (calibration R2, instrument name); one for sample-level information (sample type, calculated concentration); and one for peak-level information (quant peak area, modification flag). Accessing the relationships across these three sources – reporting the quant and qual peak area of only the qc samples, for example – requires the tools of relational data. In the tidyverse, these tools are part of the **dplyr** package and involve three ‘families of verbs’ called *mutating joins*, *filtering joins*, and *set operations*, which in turn expect a unique key in order to correctly correlate the data.

jan.b <- read.xlsx(xlsxFile=file.path(workingDir,"2017-01-06.xlsx"))  
jan.p <- read\_tsv(file=file.path(workingDir,"2017-01-06.txt"))  
byBatch <- jan.b[jan.b$batchName=="b802253"|jan.b$batchName=="b252474",]  
bySample <- january[january$batchName=="b802253"|january$batchName=="b252474",]  
byPeak <- jan.p[jan.p$batchName=="b802253"|jan.p$batchName=="b252474",]

## Primary and foreign keys

A key is the variable in a data.frame – or combination of variables in a data.frame – that uniquely defines every row. In our data, *batchName* is present in each data.frame but always insufficient to define a specific row. In fact, no single column in our data operates as a key. We can build a key by combinging two (or three) columns.

byBatch$keyB <- paste(byBatch$batchName,byBatch$compoundName,sep=":")  
bySample$keyS <- paste(bySample$sampleName,bySample$compoundName,sep=":")  
byPeak$keyP <- paste(byPeak$sampleName,byPeak$compoundName,byPeak$chromatogramName,sep=":")

Doing this creates a **primary key**, which is the unique identifier for that data.frame. A **foreign key** by contrast would uniqely identify an item in another table. The following command (explained below) incompletely adds a foreign key to byPeak so that information in byBatch can be intuitively retreived. The second command completes the population of the foreign key variable, but only because the row order in byPeak follows a very specific format. There are safer ways of doing this, which require an understanding of set operations.

byPeak <- left\_join(byPeak,byBatch)

## Joining, by = c("batchName", "compoundName")

byPeak$keyB[is.na(byPeak$keyB)] <- byPeak$keyB[!is.na(byPeak$keyB)] # dangerous!

Exercise 1: Why didn’t the *left\_join* fully populate byPeak?

Exercise 2: Join the batch and sample data using only the batch-specific key  
Partially complete commands are commented out in the following code chunk. Since keyB is already built for one data.frame, creating this variable in bySample is the next step. How would you specify that only this variable which should be used for the join? Notice what that does for all of the variables in the joined data.frame.

byBatch$keyB

## [1] "b802253:morphine" "b802253:hydromorphone"  
## [3] "b802253:oxymorphone" "b802253:codeine"   
## [5] "b802253:hydrocodone" "b802253:oxycodone"   
## [7] "b252474:morphine" "b252474:hydromorphone"  
## [9] "b252474:oxymorphone" "b252474:codeine"   
## [11] "b252474:hydrocodone" "b252474:oxycodone"

#bySample$keyB <- paste( , ,sep=":")  
#exerciseTwo <- left\_join( , , )

## Set operations *union*, *intersect*, and *setdiff*

These three commands will return a vector which is the unduplicated combination of the two input vectors. *union(A,B)* includes all the values found in both A and B. *intersect(A,B)* returns only those values found in both A and B. *setdiff(A,B)* is order dependent, and returns the values of the first vector which are not also in the second vector.

A <- rep(seq(1,10),2)  
B <- seq(2,20,2)  
union(A,B)

## [1] 1 2 3 4 5 6 7 8 9 10 12 14 16 18 20

intersect(A,B)

## [1] 2 4 6 8 10

setdiff(A,B)

## [1] 1 3 5 7 9

setdiff(B,A)

## [1] 12 14 16 18 20

These commands are good for checking matches between two vectors, and we can use them to rebuild the byPeak$keyB foreign key without the risk of incorrect naming.

byPeak <- jan.p %>% # reset the byPeak data.frame, add byPeak$keyP  
 filter(batchName %in% c("b802253","b252474")) %>%  
 unite(keyP, sampleName, compoundName, chromatogramName, sep=":", remove=FALSE)  
  
allNames <- unique(byPeak$compoundName)  
byPeak$analyte <- NA  
for(name in allNames[1:6]) {  
 compoundPairIdx <- grep(name, allNames)  
 theCompound <- intersect(allNames[compoundPairIdx], name)  
 theInternalStandard <- setdiff(allNames[compoundPairIdx], name)  
 byPeak$analyte[byPeak$compoundName==theInternalStandard] <- theCompound  
 byPeak$analyte[byPeak$compoundName==theCompound] <- theCompound  
}  
  
byPeak$keyB <- paste(byPeak$batchName,byPeak$analyte,sep=":") # alternative build of keyB

## Mutating join to add columns

Mutating joins operate in much the same way as the set operations, but on data.frames instead of vectors, and with one critical difference: repeated values are retained. We took advantage of this earlier when using the left\_join command, so that the byBatch$keyB got repeated for both the Quant and the Qual peak entries in byPeak.

There are four kinds of mutating joins, differing in how the rows of the source data.frames are treated. In each case, the matching columns are identified by column name and only one is kept, with row order remaining consistent with the principle (usually the left) source. All non-matching columns are returned, and which rows are returned depends on the type of join. An *inner\_join(A,B)* only returns rows from A which have a column match in B. The *full\_join(A,B)* returns every row of both A and B, using an NA in those columns which don’t have a match. The *left\_join(A,B)* returns every row of A, and either the matching value from B or an NA for columns with don’t have a match. Finally, the *right\_join(A,B)* returns every row of B, keeping the order of B, with either the matching value from columns in A or an NA for columns with no match.

Because these commands can duplicate rows, the potential for breaking things is pretty significant if the key isn’t unique. Here are two examples, one where you do – and one where you do not – want that duplication:

goodDuplication <- inner\_join(x=bySample[,c(1:3,7)],  
 y=byBatch[,c(1:6)],  
 by=c("batchName","compoundName"))  
badDuplication <- inner\_join(x=bySample[,c(1:3,7)],  
 y=byBatch[,c(1:6)],  
 by=c("compoundName"))

Having built the byBatch primary key, and correctly included it as a foreign key in byPeak, correctly joining them into a single data.frame is straightforward.

byPeakWide <- left\_join(x=byPeak,y=byBatch)

## Joining, by = c("batchName", "compoundName", "keyB")

## Filtering join to check the overlap

We created the byBatch$keyB explicitly, but it was effectively present already thanks to the *batchName* and *compoundName* columns. The compound naming scheme in byPeak remains problematic since the internal standard isn’t identified in byBatch or bySample, but we fixed this using a new column *analyte*. We could have accomplished the same effect in fewer steps using the semi\_join and anti\_join commands. The *semi\_join(A,B)* returns all rows of A where there is a match from B, but keeps only the columns of A, and does not duplicate a row if there are multiple matches. The *anti\_join(A,B)* is the inverse, returning all rows from A where there is no match from B. To create the ‘analyte’ column as before:

byBatch <- jan.b %>% # reset the byBatch data.frame  
 filter(batchName %in% c("b802253","b252474"))  
byPeak <- jan.p %>% # reset the byPeak data.frame, add byPeak$analyte  
 filter(batchName %in% c("b802253","b252474")) %>%  
 mutate(analyte=compoundName)  
  
noMatch <- anti\_join(byPeak,byBatch)

## Joining, by = c("batchName", "compoundName")

unique(byPeak$analyte)

## [1] "morphine" "hydromorphone" "oxymorphone"   
## [4] "codeine" "hydrocodone" "oxycodone"   
## [7] "morphine-13C" "hydromorphone-d3" "oxymorphone-d3"   
## [10] "codeine-d6" "hydrocodone-d3" "oxycodone-d3"

byPeak$analyte <- sub("-.\*$", "", byPeak$analyte)   
  
noMatch <- anti\_join(byPeak,byBatch,by=c("batchName","analyte"="compoundName"))  
justMatch <- semi\_join(byPeak,byBatch,by=c("batchName","analyte"="compoundName"))  
c(nrow(noMatch),nrow(justMatch),nrow(byPeak))

## [1] 0 2496 2496

Exercise 3: Join the batch and peak data  
Start from the reset data.frames built in the prior code chunk, so the keyB and keyP variables are not present. Partially complete commands are commented out in the following code chunk.

#exerciseThree <- left\_join( , ,by=c( ) )

# Summary

* **rbind** and **cbind** add rows (or columns) to an existing data.frame
* **union**, **intersect**, and **setdiff** return a combination of two vectors
* Relational data merges two data.frames on the common columns, called keys
  + A primary key is a unique identifier for every row in a data.frame (the presence of keyB in byBatch)
  + A foreign key is a unique identifier for another data.frame (the presence of keyB in byPeak)
* **inner\_join**, **full\_join**, **left\_join**, and **right\_join** are mutating joins which add columns
* **semi\_join** and **anti\_join** are filtering joins which check for overlap