Subsetting with dplyr

## Learning Objectives

* Subset rows using filter or slice
* Subset columns using select

In a previous lesson, we reviewed how to subset vectors and data frames in R. Now we will learn how to perform similar operations on data frames using functions from the dplyr package. A short-term advantage of these functions is that they are faster to type, which facilitates interactive, exploratory analysis. However, the true advantage of learning these subsetting functions is to combine them with dplyr's more powerful capabilities.

Subsetting alone is not the most exciting task, but it provides a gentle introduction to the dplyr approach. We'll use subsetting to explore the data and learn about the general trends and to identify outliers. And we'll explore these ideas further as we learn more dplyr functions.

Before we begin we need to load the package.

library("dplyr")

Attaching package: 'dplyr'  
  
The following objects are masked from 'package:stats':  
  
 filter, lag  
  
The following objects are masked from 'package:base':  
  
 intersect, setdiff, setequal, union

### Subsetting rows with filter

Let's start by limiting our analysis to only include research articles. To subset the rows, we use the function filter. Thus to only include the research articles, we can run the following:

research <- filter(counts\_raw, articleType == "Research Article")

The first argument supplied to filter is the original data frame. The second argument is the criteria for filtering: articleType == "Research Article". This is the general format for all dplyr functions: the first argument is always the data frame followed by the arguments specific to that function. Also notice that we referred to the column name articleType without using quotation marks or indexing the data frame, i.e. counts\_raw$articleType. This is a convenience feature available for all dplyr functions.

## Non-standard evaluation

The functions in dplyr allow you to refer directly to column names using what is called non-standard evaluation. The function subset from base R uses a similar trick to reduce typing. Using non-standard evaluation is extremely convenient for interactive programming but introduces many complications when writing functions. See the chapter in [Advanced R](http://adv-r.had.co.nz/Computing-on-the-language.html) by Hadley Wickham to learn about this topic in general or the [dplyr documentation](https://cran.r-project.org/web/packages/dplyr/vignettes/nse.html) to learn how it specifically applies to dplyr.

How many research articles were published in PLOS journals in 2006?

research\_2006 <- filter(research, year == 2006)  
nrow(research\_2006)

[1] 873

And how many of the these articles from 2006 were mentioned in at least one tweet?

research\_2006\_tweet <- filter(research\_2006, backtweetsCount > 0)  
nrow(research\_2006\_tweet)

[1] 10

And we aren't limited to one filtering condition at a time. We can list an arbitrary number of filtering conditions separated by commas. For example, how many of these 2006 articles received at least one Facebook comment?

research\_2006\_fb <- filter(research, year == 2006,  
 facebookCommentCount > 0)  
nrow(research\_2006\_fb)

[1] 5

Furthermore, we can create even more complex filters using the & ("and") and | ("or") operators. How many of the articles in 2006 had at least one Tweet or Facebook comment?

research\_2006\_fb\_tweet <- filter(research, year == 2006,  
 facebookCommentCount > 0 |  
 backtweetsCount > 0)  
nrow(research\_2006\_fb\_tweet)

[1] 13

As we have seen, filter works with any expression that evaluates to a logical vector. Thus we can use grepl to search for patterns. How many of these articles with early social media coverage in 2006 involved infectious disease research?

research\_2006\_fb\_tweet\_disease <- filter(research, year == 2006,  
 facebookCommentCount > 0 |  
 backtweetsCount > 0,  
 grepl("Infectious Diseases",  
 plosSubjectTags))  
nrow(research\_2006\_fb\_tweet\_disease)

[1] 3

### Subsetting columns with select

Switching from rows to columns, dplyr provides the function select for subsetting the columns of a data frame. This is especially useful with a larger data set like ours which has 32 columns.

colnames(research)

[1] "doi" "pubDate"   
 [3] "journal" "title"   
 [5] "articleType" "authorsCount"   
 [7] "f1000Factor" "backtweetsCount"   
 [9] "deliciousCount" "pmid"   
[11] "plosSubjectTags" "plosSubSubjectTags"   
[13] "facebookShareCount" "facebookLikeCount"   
[15] "facebookCommentCount" "facebookClickCount"   
[17] "mendeleyReadersCount" "almBlogsCount"   
[19] "pdfDownloadsCount" "xmlDownloadsCount"   
[21] "htmlDownloadsCount" "almCiteULikeCount"   
[23] "almScopusCount" "almPubMedCentralCount"   
[25] "almCrossRefCount" "plosCommentCount"   
[27] "plosCommentResponsesCount" "wikipediaCites"   
[29] "year" "daysSincePublished"   
[31] "wosCountThru2010" "wosCountThru2011"

Let's create two new new data frames that are a subset of the original research. article\_info will contain columns describing the article, and metrics will contain the altmetrics and citation data. First we'll select some columns that specifically describe the article. To do this, we simply list each column we want separated by commas.

article\_info <- select(research, doi, pubDate, journal, title,  
 articleType, authorsCount)  
colnames(article\_info)

[1] "doi" "pubDate" "journal" "title"   
[5] "articleType" "authorsCount"

Note again that we did not need to use quotation marks when writing the columns names. However, since these columns are adjacent in the data frame, we can write the command in an even more compact way. Using a colon, :, we can specify the first and last column we want, and all intervening columns will be selected as well.

article\_info <- select(research, doi:authorsCount)  
colnames(article\_info)

[1] "doi" "pubDate" "journal" "title"   
[5] "articleType" "authorsCount"

Now we'll select the columns which contain the count data. Luckily we do not have to type all of those columns! select has multiple special functions that help subset columns (see ?select for all the options). One versatile special function is contains, which works similar to grep. It selects a column if it contains the search string provided. We can use it to select all the columns that contain the pattern "Count".

metrics <- select(research, contains("Count"))  
colnames(metrics)

[1] "authorsCount" "backtweetsCount"   
 [3] "deliciousCount" "facebookShareCount"   
 [5] "facebookLikeCount" "facebookCommentCount"   
 [7] "facebookClickCount" "mendeleyReadersCount"   
 [9] "almBlogsCount" "pdfDownloadsCount"   
[11] "xmlDownloadsCount" "htmlDownloadsCount"   
[13] "almCiteULikeCount" "almScopusCount"   
[15] "almPubMedCentralCount" "almCrossRefCount"   
[17] "plosCommentCount" "plosCommentResponsesCount"  
[19] "wosCountThru2010" "wosCountThru2011"

## Case sensitivity

By default, most of the special functions for select are not case sensitive. Thus we could have written "count" and received the same result. In general, it is better to be as specific as possible. If we needed to differentiate between "Count" and "count", we could set the argument ignore.case = FALSE when calling contains.

This is almost what we want. The first problem is that the column authorsCount was included. We can specifically exclude a column by inserting a minus sign before it.

metrics <- select(research, contains("Count"), -authorsCount)  
colnames(metrics)

[1] "backtweetsCount" "deliciousCount"   
 [3] "facebookShareCount" "facebookLikeCount"   
 [5] "facebookCommentCount" "facebookClickCount"   
 [7] "mendeleyReadersCount" "almBlogsCount"   
 [9] "pdfDownloadsCount" "xmlDownloadsCount"   
[11] "htmlDownloadsCount" "almCiteULikeCount"   
[13] "almScopusCount" "almPubMedCentralCount"   
[15] "almCrossRefCount" "plosCommentCount"   
[17] "plosCommentResponsesCount" "wosCountThru2010"   
[19] "wosCountThru2011"

The second problem is that we are missing the columns f1000Factor and wikipediaCites. Let's include them by listing them.

metrics <- select(research, contains("Count"), -authorsCount,  
 f1000Factor, wikipediaCites)  
colnames(metrics)

[1] "backtweetsCount" "deliciousCount"   
 [3] "facebookShareCount" "facebookLikeCount"   
 [5] "facebookCommentCount" "facebookClickCount"   
 [7] "mendeleyReadersCount" "almBlogsCount"   
 [9] "pdfDownloadsCount" "xmlDownloadsCount"   
[11] "htmlDownloadsCount" "almCiteULikeCount"   
[13] "almScopusCount" "almPubMedCentralCount"   
[15] "almCrossRefCount" "plosCommentCount"   
[17] "plosCommentResponsesCount" "wosCountThru2010"   
[19] "wosCountThru2011" "f1000Factor"   
[21] "wikipediaCites"

Notice that this also rearranges the columns.

### Keeping it simple

We have observed how we can more conveniently perform complex subsetting operations with dplyr. But what if we want to do something simple like subset rows or columns by position?

This is very simple for columns. Instead of naming the columns we want, we can use the column numbers with select. Thus the following are equivalent.

head(select(research, journal))

journal  
1 pbio  
2 pbio  
3 pbio  
4 pbio  
5 pbio  
6 pbio

head(select(research, 3))

journal  
1 pbio  
2 pbio  
3 pbio  
4 pbio  
5 pbio  
6 pbio

For rows, we need to use a different function, slice. We can subset the first three rows of article\_info like this:

slice(article\_info, 1:3)

doi pubDate journal  
1 10.1371/journal.pbio.0000001 2003-10-13 pbio  
2 10.1371/journal.pbio.0000002 2003-11-17 pbio  
3 10.1371/journal.pbio.0000005 2003-08-18 pbio  
 title  
1 A Functional Analysis of the Spacer of V(D)J Recombination Signal Sequences  
2 Viral Discovery and Sequence Recovery Using DNA Microarrays  
3 The Transcriptome of the Intraerythrocytic Developmental Cycle of Plasmodium falciparum  
 articleType authorsCount  
1 Research Article 6  
2 Research Article 14  
3 Research Article 6

Therefore we can use dplyr whether our subsetting operations are basic or complex.

### Challenges

## How much did altmetrics numbers change by 2009?

How many articles were published in 2009? How many of these had at least one Tweet or Facebook comment? How many were in at least one Mendeley library (mendeleyReadersCount)?

research\_2009 <- filter(research, year == 2009)  
nrow(research\_2009)

[1] 6036

research\_2009\_fb\_tweet <- filter(research, year == 2009,  
 facebookCommentCount > 0 |  
 backtweetsCount > 0)  
nrow(research\_2009\_fb\_tweet)

[1] 830

research\_2009\_mendeley <- filter(research, year == 2009,  
 mendeleyReadersCount > 0)  
nrow(research\_2009\_mendeley)

[1] 5078

## What are people reading but not citing?

One potential use of altmetrics data is recognizing articles that are widely read among the scientific community but are not cited as highly as similarly influential papers. Compile a data set named low\_cite that contains the journal, title, and year of each research article that meets the following criteria:

* Published in 2008 or prior (cite)
* Has more than 1,000 pdf downloads (pdfDownloadsCount)
* Is contained in more than 15 Mendeley libraries (mendeleyReadersCount)
* Has fewer than 10 citations as of 2011 (wosCountThru2011)

How many articles did you find?

summary(research$pdfDownloadsCount)

Min. 1st Qu. Median Mean 3rd Qu. Max.   
 0.0 152.0 261.0 432.5 481.0 35720.0

summary(research$mendeleyReadersCount)

Min. 1st Qu. Median Mean 3rd Qu. Max.   
 0.000 1.000 2.000 4.128 5.000 171.000

summary(research$wosCountThru2011)

Min. 1st Qu. Median Mean 3rd Qu. Max.   
 0.00 2.00 7.00 14.54 16.00 737.00

low\_cite <- filter(research, pdfDownloadsCount > 1000,  
 mendeleyReadersCount > 15,  
 year < 2009, wosCountThru2011 < 10)  
low\_cite <- select(low\_cite, journal, title, year)  
nrow(low\_cite)

[1] 25