Subsetting Data in R

Introduction to R for Public Health Researchers

Overview

We showed one way to read data into R using *read.csv*. In this module, we will show you how to:

1. Select specific elements of an object by an index or logical condition
2. Renaming columns of a `data.frame`
3. Subset rows of a `data.frame`
4. Subset columns of a `data.frame`
5. Add/remove new columns to a `data.frame`
6. Order the columns of a `data.frame`
7. Order the rows of a `data.frame`

Setup

We will show you how to do each operation in base R then show you how to use the `dplyr` package to do the same operation (if applicable).

Many resources on how to use `dplyr` exist and are straightforward:

- `https://cran.rstudio.com/web/packages/dplyr/vignettes/`
- `https://stat545-ubc.github.io/block009_dplyr-intro.html`
- `https://www.datacamp.com/courses/dplyr-data-manipulation-r-tutorial`

Select specific elements using an index

Often you only want to look at subsets of a data set at any given time. As a review, elements of an R object are selected using the brackets (`[` and `]`).

For example, `x` is a vector of numbers and we can select the second element of `x` using the brackets and an index (2):

```
x = c(1, 4, 2, 8, 10)
x[2]
```

```
[1] 4
```

Select specific elements using an index

We can select the fifth or second AND fifth elements below:

```
x = c(1, 2, 4, 8, 10)
x[5]
```

```
[1] 10
```

```
x[c(2,5)]
```

```
[1]  2 10
```

Subsetting by deletion of entries

You can put a minus (-) before integers inside brackets to remove these indices from the data.

```
x[-2] # all but the second
```

```
[1]  1  4  8 10
```

Note that you have to be careful with this syntax when dropping more than 1 element:

```
x[-c(1,2,3)] # drop first 3
```

```
[1]  8 10
```

```
# x[-1:3] # shorthand. R sees as -1 to 3
x[-(1:3)] # needs parentheses
```

```
[1]  8 10
```

Select specific elements using logical operators

What about selecting rows based on the values of two variables? We use logical statements. Here we select only elements of `x` greater than 2:

```
x
```

```
[1]  1  2  4  8 10
```

```
x > 2
```

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

```
x[ x > 2 ]
```

```
[1]  4  8 10
```

Select specific elements using logical operators

You can have multiple logical conditions using the following:

- `&` : AND
- `|` : OR

```
x[ x > 2 & x < 5 ]
```

```
[1] 4
```

```
x[ x > 5 | x == 2 ]
```

```
[1]  2  8 10
```

which function

The `which` functions takes in logical vectors and returns the index for the elements where the logical value is `TRUE`.

```
which(x > 5 | x == 2) # returns index
```

```
[1] 2 4 5
```

```
x[ which(x > 5 | x == 2) ]
```

```
[1]  2  8 10
```

```
x[ x > 5 | x == 2 ]
```

```
[1]  2  8 10
```

Creating a `data.frame` to work with

Here we create a toy data.frame named `df` using random data:

```
set.seed(2016) # reproducbility
df = data.frame(x = c(1, 2, 4, 10, 10),
                x2 = rpois(5, 10),
                y = rnorm(5),
                z = rpois(5, 6)
                )
tbl = as.tbl(df)
```

Renaming Columns

Renaming Columns of a `data.frame`: base R

We can use the `colnames` function to directly reassign column names of `df`:

```
colnames(df) = c("x", "X", "y", "z")
head(df)
```

```
   x  X           y  z
1  1  7 -0.2707606  6
2  2  6 -1.1179372  4
3  4 10 -1.3473558  7
4 10 13  0.4832675 10
5 10 13  0.1523950  5
```

```
colnames(df) = c("x", "x2", "y", "z") #reset
```

Renaming Columns of a `data.frame`: base R

We can assign the column names, change the ones we want, and then re-assign the column names:

```
cn = colnames(df)
cn[ cn == "x2"] = "X"
colnames(df) = cn
head(df)
```

```
   x  X           y  z
1  1  7 -0.2707606  6
2  2  6 -1.1179372  4
3  4 10 -1.3473558  7
4 10 13  0.4832675 10
5 10 13  0.1523950  5
```

```
colnames(df) = c("x", "x2", "y", "z") #reset
```

Renaming Columns of a `data.frame`: dplyr

```
library(dplyr)
```

Note, when loading `dplyr`, it says objects can be "masked". That means if you use a function defined in 2 places, it uses the one that is loaded in **last**.

Renaming Columns of a `data.frame`: dplyr

For example, if we print `filter`, then we see at the bottom `namespace:dplyr`, which means when you type `filter`, it will use the one from the `dplyr` package.

```
filter
```

```
function (.data, ...)
{
    filter_(.data, .dots = lazyeval::lazy_dots(...))
}
<environment: namespace:dplyr>
```

Renaming Columns of a `data.frame`: dplyr

A `filter` function exists by default in the `stats` package, however. If you want to make sure you use that one, you use `PackageName::Function` with the colon-colon ("`::`") operator.

```
head(stats::filter,2)
```

```
1 function (x, filter, method = c("convolution", "recursive
2     sides = 2L, circular = FALSE, init = NULL)
```

This is important when loading many packages, and you may have some conflicts/masking:

Renaming Columns of a data.frame: dplyr

To rename columns in dplyr, you use the rename command

```
df = dplyr::rename(df, X = x2)
head(df)
```

```
   x  X           y  z
1  1  7 -0.2707606  6
2  2  6 -1.1179372  4
3  4 10 -1.3473558  7
4 10 13  0.4832675 10
5 10 13  0.1523950  5
```

```
df = dplyr::rename(df, x2 = X) # reset
```

Subsetting Columns

Subset columns of a `data.frame`:

We can grab the `x` column using the `$` operator.

```
df$x
```

```
[1]  1  2  4 10 10
```

Subset columns of a `data.frame`:

We can also subset a `data.frame` using the bracket `[, ]` subsetting.

For `data.frames` and matrices (2-dimensional objects), the brackets are `[rows, columns]` subsetting. We can grab the `x` column using the index of the column or the column name ("`x`")

```
df[, 1]
```

```
[1]  1  2  4 10 10
```

```
df[, "x"]
```

```
[1]  1  2  4 10 10
```

Biggest difference between `tbl` and `data.frame`:

Mostly, `tbl` (tibbles) are the same as `data.frames`, except they don't print all lines. When subsetting only one column using brackets, a `data.frame` will return a vector, but a `tbl` will return a `tbl`

```
df[, 1]
```

```
[1]  1  2  4 10 10
```

```
tbl[, 1]
```

```
# A tibble: 5 × 1
      x
  <dbl>
1     1
2     2
3     4
4    10
5    10
```

Subset columns of a `data.frame`:

We can select multiple columns using multiple column names:

```
df[, c("x", "y")]
```

```
   x           y
1  1 -0.2707606
2  2 -1.1179372
3  4 -1.3473558
4 10  0.4832675
5 10  0.1523950
```

Subset columns of a `data.frame`: dplyr

The `select` command from `dplyr` allows you to subset

```
select(df, x)
```

```
   x
1  1
2  2
3  4
4 10
5 10
```

Select columns of a `data.frame`: dplyr

The `select` command from `dplyr` allows you to subset columns of

```
select(df, x, x2)
```

```
   x x2
1  1  7
2  2  6
3  4 10
4 10 13
5 10 13
```

```
select(df, starts_with("x"))
```

```
   x x2
1  1  7
2  2  6
3  4 10
4 10 13
```

Subsetting Rows

Subset rows of a `data.frame` with indices:

Let's select **rows** 1 and 3 from `df` using brackets:

```
df[ c(1, 3), ]
```

```
  x x2          y z
1 1  7 -0.2707606 6
3 4 10 -1.3473558 7
```

Subset rows of a `data.frame`:

Let's select the rows of `df` where the `x` column is greater than 5 or is equal to 2. Without any index for columns, all columns are returned:

```
df[ df$x > 5 | df$x == 2, ]
```

```
   x x2           y  z
2  2  6 -1.1179372  4
4 10 13  0.4832675 10
5 10 13  0.1523950  5
```

Subset rows of a `data.frame`:

We can subset both rows and colums at the same time:

```
df[ df$x > 5 | df$x == 2, c("y", "z")]
```

```
           y  z
2 -1.1179372  4
4  0.4832675 10
5  0.1523950  5
```

Subset rows of a `data.frame`: dplyr

The command in `dplyr` for subsetting rows is `filter`. Try `?filter`

```
filter(df, x > 5 | x == 2)
```

```
   x x2           y  z
1  2  6 -1.1179372  4
2 10 13  0.4832675 10
3 10 13  0.1523950  5
```

Note, no `$` or subsetting is necessary. R "knows" `x` refers to a column of `df`.

Subset rows of a `data.frame`: dplyr

By default, you can separate conditions by commas, and `filter` assumes these statements are joined by `&`

```
filter(df, x > 2 & y < 0)
```

```
  x x2         y z
1 4 10 -1.347356 7
```

```
filter(df, x > 2, y < 0)
```

```
  x x2         y z
1 4 10 -1.347356 7
```

Combining `filter` and `select`

You can combine `filter` and `select` to subset the rows and columns, respectively, of a `data.frame`:

```
select(filter(df, x > 2 & y < 0), y, z)
```

```
          y z
1 -1.347356 7
```

In `R`, the common way to perform multiple operations is to wrap functions around each other in a nested way such as above

Assigning Temporary Objects

One can also create temporary objects and reassign them:

```
df2 = filter(df, x > 2 & y < 0)
df2 = select(df2, y, z)
```

Piping - a new concept

There is another (newer) way of performing these operations, called "piping". It is becoming more popular as it's easier to read:

```
df %>% filter(x > 2 & y < 0) %>% select(y, z)
```

```
           y z
1 -1.347356 7
```

It is read: "take df, then filter the rows and then select `y`, `z`".

Adding/Removing Columns

Adding new columns to a `data.frame`: base R

You can add a new column, called `newcol` to `df`, using the `$` operator:

```
df$newcol = 5:1
df$newcol = df$x + 2
```

Removing columns to a `data.frame`: base R

You can remove a column by assigning to `NULL`:

```
df$newcol = NULL
```

or selecing only the columns that were not `newcol`:

```
df = df[, colnames(df) != "newcol"]
```

Adding new columns to a `data.frame`: base R

You can also "**c**olumn **bind**" a `data.frame` with a vector (or series of vectors), using the `cbind` command:

```
cbind(df, newcol = 5:1)
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      5
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      3
4 10 13  0.4832675 10      2
5 10 13  0.1523950  5      1
```

Adding columns to a data.frame: dplyr

The mutate function in dplyr allows you to add or replace columns of a data.frame:

```
mutate(df, newcol = 5:1)
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      5
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      3
4 10 13  0.4832675 10      2
5 10 13  0.1523950  5      1
```

```
print({df = mutate(df, newcol = x + 2)})
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      3
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      6
```

Removing columns to a `data.frame`: dplyr

The `NULL` method is still very common.

The `select` function can remove a column with a minus (`-`), much like removing rows:

```
select(df, -newcol)
```

```
   x x2           y  z
1  1  7 -0.2707606  6
2  2  6 -1.1179372  4
3  4 10 -1.3473558  7
4 10 13  0.4832675 10
5 10 13  0.1523950  5
```

Removing columns to a `data.frame`: dplyr

Remove `newcol` and `y`

```
select(df, -one_of("newcol", "y"))
```

```
   x x2  z
1  1  7  6
2  2  6  4
3  4 10  7
4 10 13 10
5 10 13  5
```

Ordering columns

Ordering the columns of a `data.frame`: base R

We can use the `colnames` function to get the column names of `df` and then put `newcol` first by subsetting `df` using brackets:

```
cn = colnames(df)
df[, c("newcol", cn[cn != "newcol"]) ]
```

```
  newcol  x x2          y  z
1      3  1  7 -0.2707606  6
2      4  2  6 -1.1179372  4
3      6  4 10 -1.3473558  7
4     12 10 13  0.4832675 10
5     12 10 13  0.1523950  5
```

Ordering the columns of a `data.frame`: dplyr

The `select` function can reorder columns. Put `newcol` first, then select the rest of columns:

```
select(df, newcol, everything())
```

```
  newcol  x x2          y  z
1      3  1  7 -0.2707606  6
2      4  2  6 -1.1179372  4
3      6  4 10 -1.3473558  7
4     12 10 13  0.4832675 10
5     12 10 13  0.1523950  5
```

Ordering rows

Ordering the rows of a `data.frame`: base R

We use the `order` function on a vector or set of vectors, in increasing order:

```
df[ order(df$x), ]
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      3
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      6
4 10 13  0.4832675 10     12
5 10 13  0.1523950  5     12
```

Ordering the rows of a `data.frame`: base R

The `decreasing` argument will order it in decreasing order:

```
df[ order(df$x, decreasing = TRUE), ]
```

```
   x x2           y  z newcol
4 10 13  0.4832675 10     12
5 10 13  0.1523950  5     12
3  4 10 -1.3473558  7      6
2  2  6 -1.1179372  4      4
1  1  7 -0.2707606  6      3
```

Ordering the rows of a `data.frame`: base R

You can pass multiple vectors, and must use the negative (using `-`) to mix decreasing and increasing orderings (sort increasing on `x` and decreasing on `y`):

```
df[ order(df$x, -df$y), ]
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      3
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      6
4 10 13  0.4832675 10     12
5 10 13  0.1523950  5     12
```

Ordering the rows of a `data.frame`: dplyr

The `arrange` function can reorder rows By default, `arrange` orders in ascending order:

```
arrange(df, x)
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      3
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      6
4 10 13  0.4832675 10     12
5 10 13  0.1523950  5     12
```

Ordering the rows of a `data.frame`: dplyr

Use the `desc` to arrange the rows in descending order:

```
arrange(df, desc(x))
```

```
   x x2           y  z newcol
1 10 13  0.4832675 10      12
2 10 13  0.1523950  5      12
3  4 10 -1.3473558  7       6
4  2  6 -1.1179372  4       4
5  1  7 -0.2707606  6       3
```

Ordering the rows of a `data.frame`: dplyr

It is a bit more straightforward to mix increasing and decreasing orderings:

```
arrange(df, x, desc(y))
```

```
   x x2           y  z newcol
1  1  7 -0.2707606  6      3
2  2  6 -1.1179372  4      4
3  4 10 -1.3473558  7      6
4 10 13  0.4832675 10     12
5 10 13  0.1523950  5     12
```

Transmutation

The `transmute` function in `dplyr` combines both the `mutate` and `select` functions. One can create new columns and keep the only the columns wanted:

```
transmute(df, newcol2 = x * 3, x, y)
```

```
  newcol2  x           y
1       3  1 -0.2707606
2       6  2 -1.1179372
3      12  4 -1.3473558
4      30 10  0.4832675
5      30 10  0.1523950
```