Subsetting and Sorting

Contents

Subsetting - a quick review

Creating a data frame to work with

```
set.seed(13435)
X <-
        data.frame(
        "var1" = sample(1:5),
        "var2" = sample(6:10),
        "var3" = sample(11:15)
        )
X <- X[sample(1:5), ]
X$var2[c(1, 3)] = NA
X
```

```
  var1 var2 var3
1    2   NA   15
4    1   10   11
2    3   NA   12
3    5    6   14
5    4    9   13
```

Get the first column

```
X[, 1]
```

```
[1] 2 1 3 5 4
```

Get the first row

```
X[1, ]
```

```
  var1 var2 var3
1    2   NA   15
```

Get the first variable in the first row and first column

```
X[1, 1]
```

```
[1] 2
```

Get (row) variable by name

```
X[, "var2"]
```

```
[1] NA 10 NA  6  9
```

```
X[2, "var2"]
```

```
[1] 10
```

Get for a row subset a variable by name

```
X[1:2, "var2"]
```

```
[1] NA 10
```

Get data with logical AND conditions

```
X[(X$var1 >= 3 & X$var3 >= 13), ]
```

```
  var1 var2 var3
3    5    6   14
5    4    9   13
```

Get data with logical OR conditions

```
X[(X$var1 >= 3 | X$var3 >= 13), ]
```

```
  var1 var2 var3
1    2   NA   15
2    3   NA   12
3    5    6   14
5    4    9   13
```

Dealing with missing values

Problem: Subsetting on NAs

Subsetting on NAs will not produce the actual rows:

```
X[(X$var2 > 8),]
```

```
     var1 var2 var3
NA     NA   NA   NA
4       1   10   11
NA.1   NA   NA   NA
5       4    9   13
```

Solution: Subsetting with `which`

You can use the behavior of the `which` function, which takes only TRUE indices.

```
X[which(X$var2 > 8),]
```

```
  var1 var2 var3
4    1   10   11
5    4    9   13
```

Sorting, ordering, arrange of rows

Sort: Ordering of just one variable

```
sort(X$var1)
```

```
[1] 1 2 3 4 5
```

```
sort(X$var1, decreasing = TRUE)
```

```
[1] 5 4 3 2 1
```

```
sort(X$var2) # NAs are not included
```

```
[1]  6  9 10
```

```
sort(X$var2, na.last = TRUE)
```

```
[1]  6  9 10 NA NA
```

Order: Ordering of a row in a data frame

```
X[order(X$var1),]
```

```
  var1 var2 var3
4    1   10   11
1    2   NA   15
2    3   NA   12
5    4    9   13
3    5    6   14
```

Order: Ordering of multiple variables of a data frame

The following example is not a good one, as there aren't any ties where you can see that the second variable is ordered inside the first

```
X[order(X$var1,X$var3),]
```

```
  var1 var2 var3
4    1   10   11
1    2   NA   15
2    3   NA   12
5    4    9   13
3    5    6   14
```

Arrange: Ordering with the arrange command of dplyr

{r label = "order-with-arrange"}‘ library("dplyr") arrange(X, var1) ### Arrange: Sorting in descending order

```
library("dplyr")
arrange(X, desc(var1))
```

```
  var1 var2 var3
1    5    6   14
2    4    9   13
3    3   NA   12
4    2   NA   15
5    1   10   11
```

Adding row and columns

Adding directly at the end of the data frame

```
X$var4 <- rnorm(5)
X
```

```
  var1 var2 var3       var4
1    2   NA   15  0.1875960
4    1   10   11  1.7869764
2    3   NA   12  0.4966936
3    5    6   14  0.0631830
5    4    9   13 -0.5361329
```

Add column to the right with cbind

```
Yr <- cbind(X,rnorm(5)) # it has to be the same dimension as the data frame
Yr
```

```
  var1 var2 var3       var4    rnorm(5)
1    2   NA   15  0.1875960  0.62578490
4    1   10   11  1.7869764 -2.45083750
2    3   NA   12  0.4966936  0.08909424
3    5    6   14  0.0631830  0.47838570
5    4    9   13 -0.5361329  1.00053336
```

Add column to the left with cbind

```
Yl <- cbind(rnorm(5), X) # it has to be the same dimension as the data frame
Yl
```

```
    rnorm(5) var1 var2 var3       var4
1  0.5439561    2   NA   15  0.1875960
4  0.3304796    1   10   11  1.7869764
2 -0.9710917    3   NA   12  0.4966936
3 -0.9446847    5    6   14  0.0631830
5 -0.2967423    4    9   13 -0.5361329
```

Add row at the end with rbind

```
Xr <- rbind(X, c(0, 4, 22, NA)) # it has to be the same dimension as the data frame
Xr
```

```
  var1 var2 var3       var4
1    2   NA   15  0.1875960
4    1   10   11  1.7869764
2    3   NA   12  0.4966936
3    5    6   14  0.0631830
5    4    9   13 -0.5361329
6    0    4   22         NA
```

Add row to the top with rbind

```
Yl <- rbind(c(0, 4, 22, NA), X) # it has to be the same dimension as the data frame
Yl
```

```
   var1 var2 var3       var4
1     0    4   22         NA
11    2   NA   15  0.1875960
4     1   10   11  1.7869764
2     3   NA   12  0.4966936
3     5    6   14  0.0631830
5     4    9   13 -0.5361329
```

Subsetting with lists

Lists are a very generic datatype. It can hold vectors, strings, matrices, models, list of other list. Lists can reference data using $ (if the elements are named), or using [], or [[]]. For instance, if there is a list "mylist", with a list of "myname" then:

1. [], mylist["myname"] returns a list of elements
2. [[]], mylist$myname, mylist[["myname"]] returns the original class (vector, matrix etc.)

The most important distinction between [, [[and $ is that the [can select more than one element whereas the other two select a single element. ### Creating a list to work with

```
mylist <-
        list(letters = c("A", "b", "c"),
        numbers = 1:3,
        matrix(1:25, ncol = 5),
        text = list(text1 = c("This", "is", "a", "test"),
             text2 = c("That", "is", "another", "test")))
mylist
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3

[[3]]
     [,1] [,2] [,3] [,4] [,5]
```

```
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

$text
$text$text1
[1] "This" "is"   "a"    "test"

$text$text2
[1] "That"    "is"      "another" "test"
```

There are two different forms of subsetting with lists

Mode 1: Subsetting returns a list

```
mylist[1] # returns a list
```

```
$letters
[1] "A" "b" "c"
```

```
class(mylist[1])
```

```
[1] "list"
```

```
mylist["letters"] # returns a list
```

```
$letters
[1] "A" "b" "c"
```

```
class(mylist["letters"])
```

```
[1] "list"
```

Mode 2: Subsetting returns data of the class of the subsetted datatype

```
mylist[[1]] # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

```
class(mylist[[1]])
```

```
[1] "character"
```

```
mylist$letters # returns vector
```

```
[1] "A" "b" "c"
```

```
class(mylist$letters)
```

```
[1] "character"
```

```
mylist[["letters"]] # returns the vector 'letters
```

```
[1] "A" "b" "c"
```

```
class(mylist[["letters"]])
```

```
[1] "character"
```