Basic Course on R: Manipulating / Selecting Data

Karl Brand* and Elizabeth Ribble†

18-24 May 2017

Contents

*brandk@gmail.com
†emcclel3@msudenver.edu

1 Manipulating / Selecting Data

1.1 Indexing

Select data from vectors, arrays, lists, rows/columns in matrices and data.frames, using

- Explicit numeric selection
- Named element selection
- Logical and relational filtering

1.1.1 Explicit numeric selection

We can specify positions or *indices* i in numbers of the data we wish to select using square brackets which are the 'extract' function, i.e., `object[i]`.

```
x <- c(11, 10, 12, 15)
x[2]

## [1] 10

y <- c(13, 17, 21, 18)
y[c(2,3)]

## [1] 17 21

m <- matrix(c(x, y), ncol=2)
m

##      [,1] [,2]
## [1,]   11   13
## [2,]   10   17
## [3,]   12   21
## [4,]   15   18

m[, 1]

## [1] 11 10 12 15

m[c(2, 4), 1:2]

##      [,1] [,2]
## [1,]   10   17
## [2,]   15   18
```

We can also use negative indices to remove/exclude elements we do not want:

```
x[-2]

## [1] 11 12 15

m[, -c(1, 3)]

## [1] 13 17 21 18
```

1.1.2 Named element selection

Here we make use of the `names` attribute:

```
names(x) <- c("a", "b", "c", "d")
x

##  a  b  c  d
## 11 10 12 15

str(x)

##  Named num [1:4] 11 10 12 15
##  - attr(*, "names")= chr [1:4] "a" "b" "c" "d"

x["b"]

##  b
## 10

row.names(m) <- LETTERS[1:4]
m

##   [,1] [,2]
## A   11   13
## B   10   17
## C   12   21
## D   15   18

m["A",]

## [1] 11 13
```

Note that this is case sensitive and that `LETTERS` is different from `letters`.

We can even combine the use of numbers and names:

```
m["A", 2]

##  A
## 13
```

Let's look at an example where row names are gene symbols and column names are sample IDs:

```
mygenes <- data.frame(samp1 = c(33, 22, 12),
                      samp2 = c(44, 111, 13),
                      samp3 = c(33, 53, 65))
row.names(mygenes) <- c("CRP", "BRCA1", "HOXA")
mygenes

##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65

mygenes["CRP", ]

##     samp1 samp2 samp3
## CRP    33    44    33

mygenes[, "samp1"]

## [1] 33 22 12

mygenes[, c("samp1", "samp3")]

##       samp1 samp3
## CRP      33    33
## BRCA1    22    53
## HOXA     12    65

mygenes["HOXA", "samp2"]

## [1] 13
```

Note that we can also change the names of a `data.frame` in the same way we would do so for a `list` using the `names` attribute:

```
names(mygenes) <- c("samp10", "samp20", "samp30")
mygenes

##       samp10 samp20 samp30
## CRP       33     44     33
## BRCA1     22    111     53
## HOXA      12     13     65

## but let's change it back...
names(mygenes) <- c("samp1", "samp2", "samp3")
```

Note that the `colnames` function also performs the same job for data frames.

1.1.3 Logical and relational filtering

Expressions like `<`, `<=`, `|`, and `!=` can also be used to select data:

```
x

##  a  b  c  d
## 11 10 12 15

y

## [1] 13 17 21 18

keep <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
x[keep]

##    a    b <NA>
##   11   10   NA

x[y>=18]

##  c  d
## 12 15
```

Why?

```r
y>=18
```

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

```r
which(y>=18)
```

```
## [1] 3 4
```

```r
x[x<=11 | x==15]
```

```
##  a  b  d
## 11 10 15
```

```r
x[x!=10]
```

```
##  a  c  d
## 11 12 15
```

This works for factors as well:

```r
gender <- factor(c("M", "M", "F", "F"))
gender
```

```
## [1] M M F F
## Levels: F M
```

```r
males <- gender[gender=="M"]
levels(males)
```

```
## [1] "F" "M"
```

1.2 Other useful functions

Besides square brackets (`[`, `[[`), other useful functions exist for selecting data: `duplicated`, `match()`, `%in%`, `grep`, `is.na` and `$`.

To select e.g. rows that are not duplicated:

```r
mm <- matrix(c(x, x, y, y), nrow=4, byrow=T)
```

```
mm
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   10   12   15
## [2,]   11   10   12   15
## [3,]   13   17   21   18
## [4,]   13   17   21   18
```

```
mm[!duplicated(mm), ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   10   12   15
## [2,]   13   17   21   18
```

The above can also be done with `unique`, but the use of `duplicated` might be more appropriate in more complex situations:

```
unique(mm)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   10   12   15
## [2,]   13   17   21   18
```

Calling `match` returns the position of the first match of its first argument in the second argument:

```
match(c("a", "b"), c("a", "c", "a", "b", "a", "b"))
```

```
## [1] 1 4
```

whereas `%in%` tells you whether the elements of the first argument appear in the second argument:

```
c("a", "b", "d") %in% c("a", "c", "a", "b", "a", "b")
```

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

Recall our data frame mygenes:

```
mygenes
```

```
##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65
```

```
is.data.frame(mygenes)

## [1] TRUE
```

Note that since `mygenes` is a data frame, it is therefore also an array, which means we can select by the name of the elements in the array:

```
mygenes[match(c("samp1", "samp3"), colnames(mygenes))]

##       samp1 samp3
## CRP      33    33
## BRCA1    22    53
## HOXA     12    65

mygenes[colnames(mygenes)%in%c("samp1", "samp4")]

##       samp1
## CRP      33
## BRCA1    22
## HOXA     12
```

However, in this case we could just use the names...

```
mygenes[c("samp1", "samp3")]

##       samp1 samp3
## CRP      33    33
## BRCA1    22    53
## HOXA     12    65
```

But this gives an error:

```
mygenes[c("samp1", "samp30")])

## Error: <text>:1:30: unexpected ')'
## 1: mygenes[c("samp1", "samp30")])
##                                 ^
```

where this does not:

```
mygenes[colnames(mygenes)%in%c("samp1", "samp30")]

##       samp1
## CRP      33
## BRCA1    22
## HOXA     12
```

We can also use functions like `grep` to search for the names we are interested in:

```
mygenes[grep(2, names(mygenes))]

##       samp2
## CRP      44
## BRCA1   111
## HOXA     13

mygenes[grep("A", row.names(mygenes)), ]

##       samp1 samp2 samp3
## BRCA1    22   111    53
## HOXA     12    13    65
```

If we want to find or exclude data with missing values, we can use `is.na`:

```
z <- c(1:4, NA, 5:10)
z

##  [1]  1  2  3  4 NA  5  6  7  8  9 10

is.na(z)

##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

which(is.na(z))

## [1] 5

z[!is.na(z)]

##  [1]  1  2  3  4  5  6  7  8  9 10
```

Double brackets `[[` and `$` extract *single* elements of objects, whereas `[` can extract multiple elements. Recall that a data frame is a special list, where each column is an element of a list:

```
## retrieve element
mygenes[[3]]

## [1] 33 53 65
```

```
## retrieve element
mygenes$samp3

## [1] 33 53 65

## subset
mygenes[3]

##       samp3
## CRP      33
## BRCA1    53
## HOXA     65
```

These work for lists as well:

```
mygenelist <- list(mygenes = mygenes, mygenes2 = mygenes * 2)
mygenelist

## $mygenes
##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65
##
## $mygenes2
##       samp1 samp2 samp3
## CRP      66    88    66
## BRCA1    44   222   106
## HOXA     24    26   130

## retrieve list element
mygenelist[[1]]

##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65

mygenelist$mygenes

##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65
```

```
## subset list
mygenelist[1]

## $mygenes
##       samp1 samp2 samp3
## CRP      33    44    33
## BRCA1    22   111    53
## HOXA     12    13    65
```

And we can even combine `$` with `[` :

```
mygenelist$mygenes[2]

##       samp2
## CRP      44
## BRCA1   111
## HOXA     13
```

The `$` notation is useful for accessing elements of objects output by functions, e.g. a *t*-test:

```
tt <- t.test(x, y)
names(tt)

## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"

tt$p.value

## [1] 0.04342819

tt$estimate

## mean of x mean of y
##     12.00     17.25
```