Subsetting

Anh Le

August 12, 2014

1 Data types

```
x <- c(1, 2, 3, 4)
x[] # Returns the original vector

## [1] 1 2 3 4

x[0] # Returns a zero-length vector

## numeric(0)
```

If the vector is named, we can

```
y <- setNames(x, letters[1:4])
y[c("d", "c", "a")]

## d c a
## 4 3 1
```

1.1 Logical vectors

If the logical vector is shorter than the subsetted vector, it will be recycled.

A missing value in the index always yields a missing value in the output

```
x <- 1:4
x[c(T, T, NA, F)]

## [1]  1  2 NA
```

2 Matrices and arrays

You can also subset higher-dimensional data structures with an integer matrix (or, if named, a character matrix). Each row in the matrix specifies the location

of one value, where each column corresponds to a dimension in the array being subsetted. This means that you use a 2 column matrix to subset a matrix, a 3 column matrix to subset a 3d array, and so on. The result is a vector of values:

```
vals <- outer(1:5, 1:5, FUN="paste", sep=",")
select <- matrix(ncol=2, byrow=TRUE, c(
  1, 1,
  3, 1,
  2, 4))
vals[select]

## [1] "1,1" "3,1" "2,4"
```

3 Data frame

Data frame can be subsetted both as a list (when given a single vector) or as a matrix (when given two vectors)

```
df <- data.frame(x=1:3, y=3:1, z=letters[1:3])

# An important difference if you select a single column:
# select like matrix simplifies (return a 1-d vector)
str(df[ , "y"])

##  int [1:3] 3 2 1

# select like a list does not (return a data.frame of 1 column)
str(df["y"])

## 'data.frame': 3 obs. of  1 variable:
##  $ y: int  3 2 1
```

4 Exercises

1. Fix each of the following common data frame subsetting errors (easy)
2. Why does `x <- 1:5; x[NA]` yield five missing values?

```
x <- 1:5
x[NA]

## [1] NA NA NA NA NA

x[NA_real_]

## [1] NA
```

This is because by default `NA` is logical, which gets recycled since x has longer length. `NA_real_` is not logical and does not get recycled, hence returning one `NA` only.

3. What does `upper.tri()` return? How does subsetting a matrix with it work? Do we need any additional subsetting rules to describe its behavior?

```
x <- outer(1:5, 1:5, FUN="*")
x

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    4    6    8   10
## [3,]    3    6    9   12   15
## [4,]    4    8   12   16   20
## [5,]    5   10   15   20   25

x[upper.tri(x)]

##  [1]  2  3  6  4  8 12  5 10 15 20

upper.tri(x)

##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] FALSE  TRUE  TRUE  TRUE  TRUE
## [2,] FALSE FALSE  TRUE  TRUE  TRUE
## [3,] FALSE FALSE FALSE  TRUE  TRUE
## [4,] FALSE FALSE FALSE FALSE  TRUE
## [5,] FALSE FALSE FALSE FALSE FALSE
```

4. Why does `mtcars[1:20]` return a error? How does it differ from the similar `mtcars[1:20, ]`?

   `mtcars[1:20]` tries to select column 1 to 20. There aren't that many columns.

5. Implement your own function that extracts the diagonal entries from a matrix

```
m <- outer(1:5, 1:6, FUN="paste", sep=",")
my_diag <- function(m) {
  idx <- 1:min(dim(m))
  m[cbind(idx, idx)]
}
diag(m)
```

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

my_diag(m)

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

6. What does `df[is.na(df)] <- 0` do? How does it work?

   Replace any missing value (NA) with 0. `is.na(df)` returns a logical matrix with the same dimension as df.

```
df <- data.frame(x=1:3, y=c(4,NA,2))
is.na(df)

##           x     y
## [1,] FALSE FALSE
## [2,] FALSE  TRUE
## [3,] FALSE FALSE

class(is.na(df))

## [1] "matrix"
```

5 Subsetting operators

Because it can return only a single value, you must use `[[` with either a single positive integer or a string.

If you do supply a vector it indexes recursively

```
l <- list(a = list(b = list(c = list(d = 1))))
l[[c("a", "b", "c", "d")]]

## [1] 1

l[["a"]][["b"]][["c"]][["d"]] # Same

## [1] 1
```

6 Simplifying vs preserving subsetting

Simplifying subsets returns the simplest possible data structure that can represent the output, and is useful interactively because it usually gives you what you

want. Preserving subsetting keeps the structure of the output the same as the input, and is generally better for programming because the result will always be the same type. Omitting `drop = FALSE` when subsetting matrices and data frames is one of the most common sources of programming errors.

7 Subsetting and assignment

You can't combine integer indices with NA but you can combine logical indices with NA (where it is treated as FALSE)

WEIRD: PUT IN PULL REQUEST

```
x <- c(1, 2, 3, 4)
x[c(1, NA)] <- c(10, 20)

## Error:  NAs are not allowed in subscripted assignments

x

## [1] 1 2 3 4

x[c(T, F, NA, NA)] <- c(10, 20, 30, 40)

## Error:  NAs are not allowed in subscripted assignments

x[c(T, F, NA, NA)] <- c(10)
x

## [1] 10  2  3  4
```

Notice the following behavior when we do subset assignement out of bound. It will lengthen the vector. Missing values are filled in.

```
x <- 1:4
x[c(F, F, F, F, F, T)] <- 10
x

## [1]  1  2  3  4 NA 10
```

Subsetting with nothing can be useful in conjunction with assignment because it will preserve the original object class and structure. Example:

```
# Here mtcars remains a data frame
mtcars[] <- lapply(mtcars, as.integer)
# Here mtcars becomes a lsit
mtcars <- lapply(mtcars, as.integer)
```

With lists, you can use subsetting + assignment + NULL to remove components from a list. To add a literal NULL to a list, use [and list(NULL):

```
l <- list(a=1, b=2)
l[["b"]] <- NULL
str(l)

## List of 1
##  $ a: num 1

l <- list(a=1)
l["b"] <- list(NULL)
str(l)

## List of 2
##  $ a: num 1
##  $ b: NULL
```

8 Applications

8.1 Lookup table

```
x <- c("m", "m", "u", "f", "m", "f", "f", "f")
lookup <- c("m"="Male", "f"="Female", "u"=NA)
lookup[x]

##        m        m        u        f        m        f        f        f
##   "Male"   "Male"       NA "Female"   "Male" "Female" "Female" "Female"

unname(lookup[x])

## [1] "Male"   "Male"   NA       "Female" "Male"   "Female" "Female" "Female"

# Or with fewer output values
lookup <- c("m"="Known", "f"="Known", "u"="Unknown")
lookup[x]

##         m         m         u         f         m         f         f
##   "Known"   "Known" "Unknown"   "Known"   "Known"   "Known"   "Known"
##         f
##   "Known"
```

8.2 Matching and merging by hand (integer subsetting)

```r
grades <- c(1, 2, 2, 3, 1)
info <- data.frame(
  grade = 3:1,
  desc = c("Excellent", "Good", "Poor"),
  fail = c(F, F, T)
)

# Using match
info[match(grades, info$grade), ]
```

```
##     grade      desc  fail
## 3       1      Poor  TRUE
## 2       2      Good FALSE
## 2.1     2      Good FALSE
## 1       3 Excellent FALSE
## 3.1     1      Poor  TRUE
```

```r
# Using rownames
rownames(info) <- info$grades
info[as.character(grades), ]
```

```
##     grade      desc  fail
## 1       3 Excellent FALSE
## 2       2      Good FALSE
## 2.1     2      Good FALSE
## 3       1      Poor  TRUE
## 1.1     3 Excellent FALSE
```

8.3 Random samples/bootstrap (integer subsetting)

```r
df <- data.frame(x = rep(1:3, each = 2), y = 6:1, z = letters[1:6])

# Randomly reorder
df[sample(nrow(df)), ]
```

```
##   x y z
## 1 1 6 a
## 2 1 5 b
## 3 2 4 c
## 4 2 3 d
## 6 3 1 f
## 5 3 2 e
```

```r
# Select 3 random rows
df[sample(nrow(df), 3), ]
```

```
##   x y z
## 3 2 4 c
## 2 1 5 b
## 4 2 3 d

# Select 6 bootstrap replicates
df[sample(nrow(df), 6, replace=T), ]

##     x y z
## 3   2 4 c
## 2   1 5 b
## 2.1 1 5 b
## 1   1 6 a
## 3.1 2 4 c
## 1.1 1 6 a
```