Video 9: Subsetting 2D structures

Stats 102A

Miles Chen, PhD

Subsetting Matrices and arrays

Subsetting Matrices and arrays

You can subset higher-dimensional structures in three ways:

- With multiple vectors. (Most common method)
- With a single vector.
- With a matrix. (least common)

The most common way of subsetting matrices (2d) and arrays (>2d) is a simple generalisation of 1d subsetting: you supply a 1d index for each dimension, separated by a comma. Blank subsetting is now useful because it lets you keep all rows or all columns.

Let’s create a matrix

```r
a <- matrix(1:9, nrow = 3)
colnames(a) <- c("A", "B", "C")
a
```

```
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

Subsetting the matrix

```
1 a[1:2, ] # first and second rows
```

```
     A B C
[1,] 1 4 7
[2,] 2 5 8
```

```
1 a[c(T, F, T), c("B", "A")] # first and third rows, columns b and a
```

```
     B A
[1,] 4 1
[2,] 6 3
```

```
1 a[0, -2] # no rows, all but the second column
```

```
     A C
```

Subsetting a matrix simplifies by default

By default, single square bracket subsetting [will simplify the results to the lowest possible dimensionality. We will later discuss preservation to avoid this.

```
1 is.vector(a) # matrix is not a vector
```

```
[1] FALSE
```

```
1 a[1,] # no longer a matrix
```

```
A B C
1 4 7
```

```
1 is.vector(a[1,]) # when you subset the row, it becomes a vector
```

```
[1] TRUE
```

Matrices are atomic vectors

Because matrices and arrays are implemented as vectors with special attributes, you can subset them with a single vector. In that case, they will behave like a vector. Arrays in R are stored in column-major order:

```
(vals <- matrix(LETTERS[1:25], nrow = 5))
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] "A"  "F"  "K"  "P"  "U"
[2,] "B"  "G"  "L"  "Q"  "V"
[3,] "C"  "H"  "M"  "R"  "W"
[4,] "D"  "I"  "N"  "S"  "X"
[5,] "E"  "J"  "O"  "T"  "Y"
```

```
# select the 8th and 9th values in the vector
vals[c(8, 9)]
```

```
[1] "H" "I"
```

Subset a matrix with a matrix

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:

```
select <- matrix(ncol = 2, byrow = TRUE, c(
  1, 5,   ## select the values at the coordinates (1,5)
  3, 1,   ## value at coord (3, 1), third row, 1st col
  2, 3,
  1, 1
))
vals[select]
```

```
[1] "U" "C" "L" "A"
```

Subset a 3d array with a matrix with 3 columns

```
# generalizes to three dimensions
(ar <- array(1:12, c(2,3,2)))
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Subset a 3d array with a matrix with 3 columns

```
select_ar <- matrix(ncol = 3, byrow = TRUE, c(
  1,2,1,
  2,3,2))
ar[select_ar]
```

```
[1]  3 12
```

Subsetting Data Frames

Data frames

Data frames possess the characteristics of both lists and matrices: if you subset with a single vector, they behave like lists; if you subset with two vectors, they behave like matrices.

```
df <- data.frame(x = 1:4, y = 4:1, z = letters[1:4])
df
```

```
  x y z
1 1 4 a
2 2 3 b
3 3 2 c
4 4 1 d
```

```
df[df$y %% 2 == 0, ]  # choose the rows where this logical statemer
```

```
  x y z
1 1 4 a
3 3 2 c
```

Subsetting Data Frames like a list

```r
# Select columns like you would a list:
df[c("x", "z")]
```

```
  x z
1 1 a
2 2 b
3 3 c
4 4 d
```

Subsetting Data Frames like a matrix

```r
# select columns like a matrix
df[ , c("x", "z")]
```

```
  x z
1 1 a
2 2 b
3 3 c
4 4 d
```

Subsetting Data Frames to select rows

To select the rows, you can provide a vector before the comma.

Here we choose the first and third rows.

```
df[c(1, 3), ]  # note the comma
```

```
  x y z
1 1 4 a
3 3 2 c
```

Subsetting Data Frames to select rows

If you leave out the comma, it will try to subset the data frame like a list.

In this case, we get the first and third columns and all the rows.

```
1 df[c(1, 3)]  # comma is missing
```

```
  x z
1 1 a
2 2 b
3 3 c
4 4 d
```

Data Frames with named columns

There's an important difference if you select a single column: matrix subsetting simplifies by default, list subsetting does not.

```
str(df["x"])  # preserves: remains a data frame
```

```
'data.frame':   4 obs. of  1 variable:
 $ x: int  1 2 3 4
```

```
str(df[ , "x"])  # simplifies: becomes a vector
```

```
 int [1:4] 1 2 3 4
```

Data Frames with named columns

```
1 str(df$x)  # dollar sign always simplifies: becomes a vector
```

```
int [1:4] 1 2 3 4
```

```
1 str(df[["x"]]) # simplifies: becomes a vector
```

```
int [1:4] 1 2 3 4
```

Simplifying vs. preserving

When you subset, R often simplifies the result to an atomic vector or a form different from the original form. We saw this with the data frames in previous slides.

The next few slides cover simplifying vs preserving behaviors in R for different data types

Simplifying vs. preserving: Atomic vectors

For atomic vectors simplifying removes names

```
x <- c(a = 1, b = 2)
x[1]  # preserving: keeps names
```

```
a
1
```

```
x[[1]] # simplifying: drops names
```

```
[1] 1
```

Simplifying vs. preserving: Lists

Simplifying return the object inside the list, not a single element list.

```
y <- list(a = 1, b = 2)
str(y[1])  # preserving: still a list
```

```
List of 1
 $ a: num 1
```

```
str(y[[1]]) # simplifying: is a vector
```

```
 num 1
```

Simplifying vs. preserving: Factors

Factors are a special case. For factors, simplifying is achieved by using the argument `drop = TRUE` inside the square brackets. It drops any unused levels.

```r
z <- factor(c("a", "b"))
z[1] # preserving: keeps levels that do not appear
```

```
[1] a
Levels: a b
```

```r
z[1, drop = TRUE] # simplifying: drops levels that no longer appear
```

```
[1] a
Levels: a
```

Simplifying vs. preserving: Matrices and arrays

If subsetting a matrix or array results in a dimension with a length 1, it will drop that dimension. For example, when you subset a row from a matrix, R will return an atomic vector rather than a 1 x n matrix. If you want to preserve the matrix structure, use the `drop = FALSE` argument inside the square brackets.

```
a <- matrix(1:4, nrow = 2)
a[1, , drop = FALSE] # preserving: keeps matrix structure
```

```
     [,1] [,2]
[1,]    1    3
```

```
a[1, ] # simplifying: returns an atomic vector
```

```
[1] 1 3
```

Simplifying vs. preserving: Data Frames

Data Frames are lists, so when you subset one like a list, the simplifying vs preserving rules apply.

```
df <- data.frame(a = 1:2, b = 1:2)
str(df[1]) # preserving: a single square bracket returns a data fra
```

```
'data.frame':   2 obs. of  1 variable:
 $ a: int  1 2
```

```
str(df[[1]]) # simplifying : double brackets returns a vector
```

```
 int [1:2] 1 2
```

Examples of subsetting

```
1 head(mtcars, 10)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

What's wrong?

```
1 mtcars[mtcars$cyl <= 5]
```

```
Error in `[.data.frame`(mtcars, mtcars$cyl <= 5): undefined columns selected
```

The fix

```
# need to specify selection of rows with a comma
mtcars[mtcars$cyl <= 5, ]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

What's wrong?

```
1 mtcars[mtcars$cyl == 4 | 6, ]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4

The fix

```r
# the mtcars$cyl == 4 | 6
# on the left is a logical vector
# on the right of 'or |' is the number 6 which gets coerced to TRUE
# so it returns TRUE for everything

# the or operator has to be between two logical vectors
mtcars[mtcars$cyl == 4 | mtcars$cyl == 6, ]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

What's wrong?

```
1 mtcars[1:13]
```

```
Error in `[.data.frame`(mtcars, 1:13): undefined columns selected
```

Don’t forget the comma!

```
1 mtcars[1:13,] # without the comma, it tried to select the first 13
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3