Data Management

Anthony Chau

UCI Center for Statistical Consulting

2021/01/12 (updated: 2021-01-26)

Data Management

- Now, we'll address the **manipulation problem**: how to select and change slices of our data
- The focus is on data frames but other data structures will be discussed

Data management is a broad topic, so I'll focus on a few common tasks

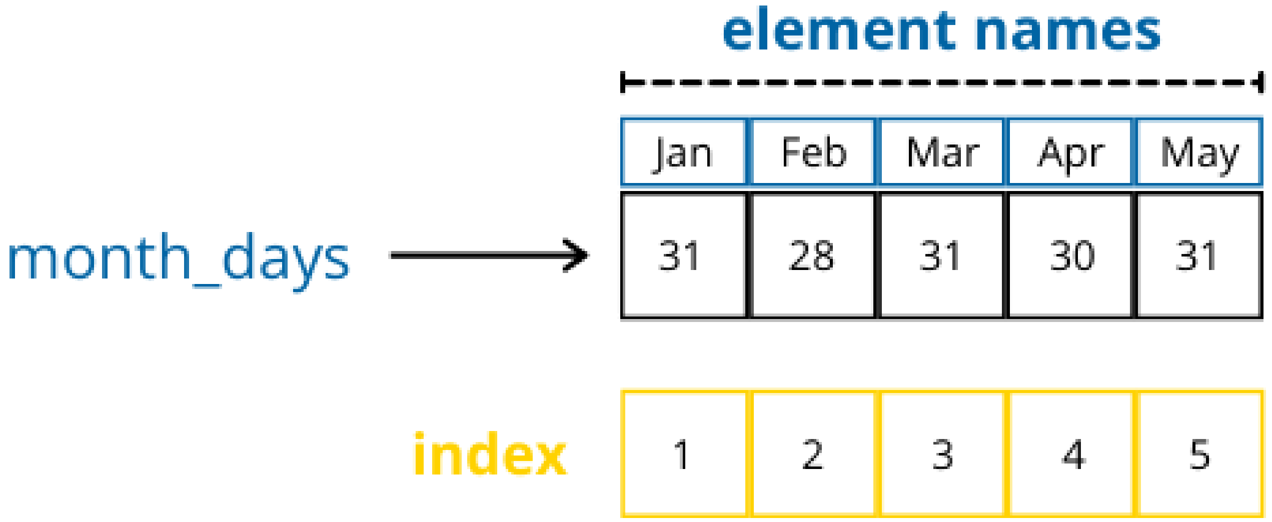
1. Select specific columns
2. Create a new column
3. Filter data given a condition
4. Rename column
5. Group data into subsets

Background

- R provides **subsetting** operators that allow us to select data in complex and useful ways
- **Subsetting** is the action of selecting specific pieces of our data
- How we subset data is dependent on the data type and data structure
- **There are three subsetting operators: `[[`, `[`, `$`**
- **The `[` and `[[` have another set of brackets**

Vector mental model

- Recall our vector mental model



Subset a vector with the [operator

- Within the brackets, we can provide a integer, character, or logical vector
- Supply an integer vector to select by index
- Supply an character vector to select by element name
- Supply an logical vector to select by condition

Subset a vector with integers

- Subset with positive or negative integers

```
x ← c(-1, 0, 2, 3)

# notice the use of c() inside the bracket for vector length > 1

# select the first element
x[1]
#> [1] -1

# select first and fourth element
x[c(1, 4)]
#> [1] -1  3

# exclude first and fourth element
x[c(-1, -4)]
#> [1] 0 2

# can't combine positive and negative indices
x[c(-1, 2)]
#> Error in x[c(-1, 2)]: only 0's may be mixed with negative subscripts
```

Subset a vector with element names

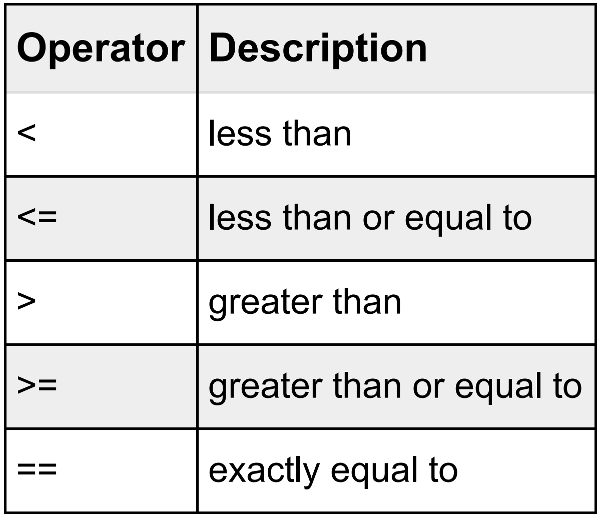
- Subset with element names

```
month_days ← c(31, 28, 31, 30, 31)
names(month_days) ← c("Jan", "Feb", "Mar", "Apr", "May")

# c() is not required for a length one subsetting vector
month_days["Feb"]
#> Feb
#>  28
month_days[c("Jan", "Apr")]
#> Jan Apr
#>  31  30
```

Aside: logical operators

- R has built-in **logical operators** (operators used to evaluate whether a condition is true or false)



Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to

Logical operators example

```r
x <- c(-10, -1, 0, 2, 3)

# remember vector recycling: (-10, -1, 0, 2, 3) > (0, 0, 0, 0, 0)
# x > 0 returns a logical vector of the same length as z
x > 0
#> [1] FALSE FALSE FALSE  TRUE  TRUE

# select and return all elements greater than 0
x[x > 0]
#> [1] 2 3

# select and return all elements less than or equal to 0
x[x <= 0]
#> [1] -10  -1   0

# select and return all elements equal to -10
x[x == -10]
#> [1] -10

# select and return all elements greater than 5
x[x > 5]
#> numeric(0)
```

Aside: boolean operators

- R has built-in **boolean operators** (operators used to chain together multiple logical expressions)

Operator	Description
!x	NOT x
x | y	x OR y
x & y	x AND y

`!` boolean operator example

- `!` reverses the logical value (`TRUE` becomes `FALSE`, `FALSE` becomes TRUE)

```
# assume that foods are either fruits or vegetables
x <- c("apple", "spinach", "broccolli", "blueberry", "carrot")

fruit <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
fruit
#> [1]  TRUE FALSE  TRUE  TRUE FALSE

# return fruits
x[fruit]
#> [1] "apple"     "broccolli" "blueberry"

# reverse each logical value in fruit
vegetable <- !fruit
vegetable
#> [1] FALSE  TRUE FALSE FALSE  TRUE

# return vegetables
x[vegetable]
#> [1] "spinach" "carrot"
```

| and & boolean operator example

- `!` evaluates to TRUE if at least one logical expression is true
- `&` evaluates to TRUE if and only if all logical expressions are true
- Use parentheses to separate logical expressions

```
x ← c(-100, -67, -34, 0, 5, 7, 15, 15, 50)

# return elements that are equal to -100 OR elements greater than 0
x[(x == -100) | (x > 0)]
#> [1] -100    5    7   15   15   50
# return elements that are less than or equal to 0 or elements greater than 0
x[(x ⩽ 0) | (x > 0)]
#> [1] -100  -67  -34    0    5    7   15   15   50

# return elements that are less than or equal AND greater than or equal to 0
x[(x ⩽ 0) & (x ⩾ 0)]
#> [1] 0
# return elements that are less than or equal to zero AND greater than 0
x[(x ⩽ 0) & (x > 0)]
#> numeric(0)
```

Subset a vector given a condition

- Subset with logical vectors

```r
mascots <- c("Peter", "Tommy", "King Triton",
             "Josephine", "Oski", "King Triton")
names(mascots) <- c("UCI", "USC", "UCSD", "UCLA", "UCB", "UCSD")
uc_campus <-c(TRUE, FALSE, TRUE, TRUE, TRUE, TRUE)
# select elements that equal "Peter" or elements that equal "Peter"
mascots[mascots == "Peter" | mascots == "King Triton"]
#>           UCI          UCSD          UCSD
#>       "Peter" "King Triton" "King Triton"

# select non-UC campuses
!uc_campus
#> [1] FALSE  TRUE FALSE FALSE FALSE FALSE
mascots[!uc_campus]
#>     USC
#> "Tommy"

# select elements with element name "UCSD"
mascots[names(mascots) == "UCSD"]
#>          UCSD          UCSD
#> "King Triton" "King Triton"
```

6 ways to subset a vector

Method	Behavior	Example	Result	Notes
Positive Integers	Select elements at the specified index	x[c(1, 4)] x[c(1, 1)]	*Return first and fourth element* *Return first element twice*	Duplicate indices return duplicate values Real numbers truncated to integers
Negative Integers	Exclude elements at the specified index	x[c(-1, -4)] x[c(-2, 2)]	*Exclude first and fourth element* *Error - not possible*	Can't mix positive and negative integer indices
Logical Vectors	Select elements when logical value is TRUE	x[c(TRUE, FALSE, TRUE)] x[x > 0]	*Return first and third element* *Return elements that are greater than 0*	
Nothing	Return the original vector	x[]	*Return the original vector*	Not that useful for vectors
Zero	Return a zero-length vector	x[0]	*Return empty numeric vector*	
Character Vectors	Select elements with matching names	x[c("a", "c", "d")]	*Return elements with element names: "a", "c", "d"*	Vector must have element names

Subset a list with [

- Subsetting a list with `[` will always return a list
- Just like vectors, you can supply a vector when using `[`

```
l ← list(letter = "a", number = 1,
          boolean = TRUE, ones_vector = c(1,1,1),
          my_list = list(1,2,3))

l[1]
#> $letter
#> [1] "a"
is.list(l[1])
#> [1] TRUE

l["boolean"]
#> $boolean
#> [1] TRUE
is.list(l["boolean"])
#> [1] TRUE
```

Subset a list with `[`

- All the ways to subset a vector carry through when subsetting a list with `[`

```
l ← list(letter = "a", number = 1,
          boolean = TRUE, num_vector = c(1,2,3),
          my_list = list(1,2))

# vectors allowed
l[c("boolean", "number")]
#> $boolean
#> [1] TRUE
#>
#> $number
#> [1] 1
is.list(l[c("boolean", "number")])
#> [1] TRUE

# negative integers allowed
l[c(-2, -3, -4, -5)]
#> $letter
#> [1] "a"
is.list(l[c(-2, -3, -4, -5)])
#> [1] TRUE
```

Subset a list with `[[`

- Subsetting a list with `[[` returns a single element in the list (the element *could* be a list)
- When using `[[`, you can supply a single positive integer, a single element name, or a vector
- If you use a vector with `[[`, you will subset recursively

```
l ← list(letter = "a", number = 1, boolean = TRUE, num_vector = c(1,2,3))

l[[1]]
#> [1] "a"
is.list(l[[1]])
#> [1] FALSE
l[["boolean"]]
#> [1] TRUE
is.list(l[["boolean"]])
#> [1] FALSE

# l[[c(4,3)]] = l[[4]][[3]]
l[[c(4, 3)]]
#> [1] 3
# no negative integers
l[[-2]]
#> Error in l[[-2]]: invalid negative subscript in get1index <real>
```

Subset a list with $

- Subsetting a list with `$` is a shorthand for subsetting with `[[`
- `l$element_name` = `l[["element_name"]]`

```r
l ← list(letter = "a", number = 1, boolean = TRUE, ones_vector = c(1,1,1))

l$letter
#> [1] "a"
is.list(l$letter)
#> [1] FALSE

l[["letter"]]
#> [1] "a"
is.list(l[["letter"]])
#> [1] FALSE
```

Subset a matrix with [

- Subsetting a matrix with `[` is similar to subsetting a vector with `[`
- Since a matrix is 2-dimensional, we select rows and columns with `m[row, column]`
- Then, we can provide a vector for each dimension to select specific rows and columns.

```r
m <- matrix(1:16, nrow = 4, ncol = 4)
colnames(m) <- c("a", "b", "c", "d")
m
#>      a b  c  d
#> [1,] 1 5  9 13
#> [2,] 2 6 10 14
#> [3,] 3 7 11 15
#> [4,] 4 8 12 16

# first row, second column
m[1, 2]
#> b
#> 5

# first and third row; column a and column c
m[c(1, 3), c("a", "c")]
#>      a  c
#> [1,] 1  9
#> [2,] 3 11
```

Subset a matrix with [

- Syntax to **select all rows**, `m[, columns]`
- Syntax to **select all columns**, `m[rows, ]`

```
m ← matrix(1:16, nrow = 4, ncol = 4)
colnames(m) ← c("a", "b", "c", "d")
m
#>      a b  c  d
#> [1,] 1 5  9 13
#> [2,] 2 6 10 14
#> [3,] 3 7 11 15
#> [4,] 4 8 12 16

# all rows; first and third column
m[, c(1, 3)]
#>      a  c
#> [1,] 1  9
#> [2,] 2 10
#> [3,] 3 11
#> [4,] 4 12

# first and second row; all columns
m[c(1, 4), ]
#>      a b  c  d
#> [1,] 1 5  9 13
#> [2,] 4 8 12 16
```

Subset a matrix with [

- Subset a matrix with a single vector
- Each element in a matrix is stored in column-major order

column major order:

start at top-left corner -> move down a column -> start at top of adjacent column

```
m ← matrix(1:16, nrow = 4, ncol = 4)
colnames(m) ← c("a", "b", "c", "d")
m
#>      a b  c  d
#> [1,] 1 5  9 13
#> [2,] 2 6 10 14
#> [3,] 3 7 11 15
#> [4,] 4 8 12 16

# select first element and third element
m[c(1,3)]
#> [1] 1 3
```

Subset a data frame

- Subsetting a data frame combines aspects