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

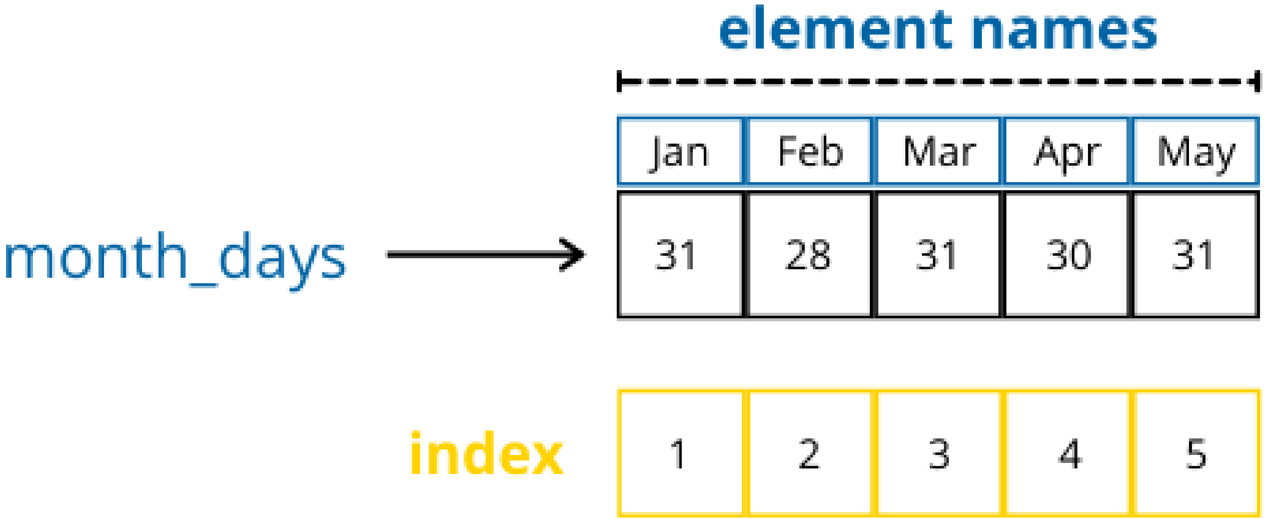
1. Select specific columns
2. Create a new column
3. Filter data given a condition
4. Rename column
5. Group data into subsets

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



Jan	Feb	Mar	Apr	May
31	28	31	30	31
1	2	3	4	5

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
```

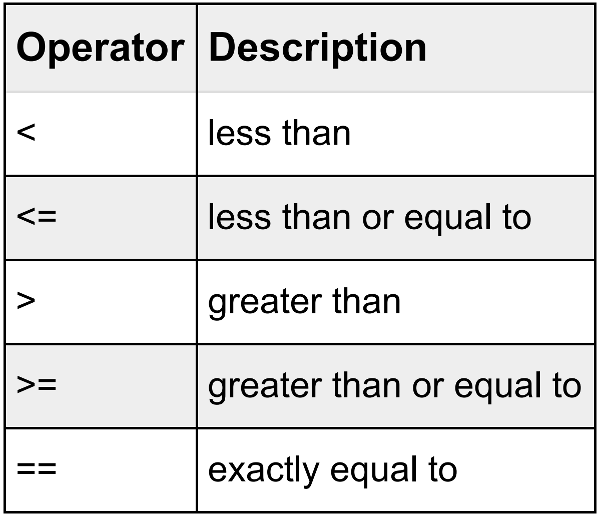
Subset a vector with element names

- Subset with element names

```r
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)



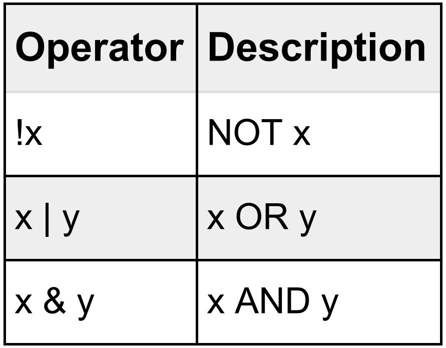
Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to

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

```r
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"]]`

```
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.

```
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 subsetting features from vectors, lists, and matrices
- When subsetting a data frame with a single index `df[]` is similar to list subsetting
- Subsetting a data frame with two indices `df[row, column]` is similar to matrix subsetting

Subset a data frame - single index

- Subsetting a data frame with a single index selects the columns of a data frame

```
mascots <- data.frame(name = c("Peter Anteater", "Josephine Bruin",
                               "King Triton","Tommy Trojan"),
                      age = c(56, 101, 60, 140), gpa = c(4, 3.9, 3.87, 3.7),
                      residence = c("Irvine", "Los Angeles",
                                    "San Diego", "Los Angeles"))

mascots[2]
#>   age
#> 1  56
#> 2 101
#> 3  60
#> 4 140
is.data.frame(mascots[2])
#> [1] TRUE
```

Subset a data frame - single index

- All methods of vector subsetting are valid for data frames

```
# subset with double vector
mascots[c(1,3)]
#>              name  gpa
#> 1  Peter Anteater 4.00
#> 2 Josephine Bruin 3.90
#> 3     King Triton 3.87
#> 4    Tommy Trojan 3.70
is.data.frame(mascots[c(1,3)])
#> [1] TRUE
# subset with character vector
mascots[c("age","gpa")]
#>   age  gpa
#> 1  56 4.00
#> 2 101 3.90
#> 3  60 3.87
#> 4 140 3.70
is.data.frame(mascots[c("age","gpa")])
#> [1] TRUE
```

Subset a data frame - double index

- Subsetting a data frame with two indices selects the rows and columns of a data frame

```
# select second and third row; all columns
mascots[c(2,3), ]
#>                 name age  gpa   residence
#> 2 Josephine Bruin 101 3.90 Los Angeles
#> 3     King Triton  60 3.87   San Diego
is.data.frame(mascots[c(2,3), ])
#> [1] TRUE
# select all rows; name and residence columns
mascots[, c("name", "residence")]
#>                 name   residence
#> 1  Peter Anteater      Irvine
#> 2 Josephine Bruin Los Angeles
#> 3     King Triton   San Diego
#> 4    Tommy Trojan Los Angeles
is.data.frame(mascots[, c("name", "residence")])
#> [1] TRUE
```

Subset a data frame - double index

- Note that selecting only one column with double index subsetting does not return a data frame - this can be a source of downstream problems

```
# returns a data frame
mascots["age"]
#>   age
#> 1  56
#> 2 101
#> 3  60
#> 4 140
is.data.frame(mascots["age"])
#> [1] TRUE

# returns a vector
mascots[, "age"]
#> [1]  56 101  60 140
is.data.frame(mascots[, "age"])
#> [1] FALSE
```

Applications of subsetting

- Knowing the mechanics of subsetting opens up a variety of new operations we can perform on our data
- We will focus on the following operations for data frames

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

Select specific columns

- Select columns with `[`, `[[` or `$`

```
df ← data.frame(a = c(1:3), b = c(4:6), c = c(7:9), d = c(10:12))

df[, c("a", "d")]
#>   a  d
#> 1 1 10
#> 2 2 11
#> 3 3 12
df[c("b", "c")]
#>   b c
#> 1 4 7
#> 2 5 8
#> 3 6 9
# use drop = FALSE to return a data frame
df[, c("b"), drop=FALSE]
#>   b
#> 1 4
#> 2 5
#> 3 6
df[["b"]]
#> [1] 4 5 6
df$b
#> [1] 4 5 6
```

Create a new column

- Create a new column by combining subsetting and assignment
- On the left hand side of the assignment operator, supply the name of your new column
- On the right hand side of the assignment operator, In general, apply some transformation on your existing columns to derive a new column

```
# raw data: weight ~ pounds, height ~ meters
df ← data.frame(height_m = c(1.7, 1.65, 1.9, 1.8, 1.73, 1.7),
                 weight_lbs = c(151, 149, 187, 183, 175, 178),
                 age = c(22, 20, 21, 19, 20, 19),
                 sex = c("Male", "Male", "Female",
                         "Male", "Male", "Female"),
                 home_state = c("CA", "AZ", "TX", "CA",
                                "FL", "NY"),
                 school = c("USC", "UCI", "UCLA",
                            "USC", "Chapman", "UCSD"))

df
#>   height_m weight_lbs age    sex home_state  school
#> 1     1.70        151  22   Male         CA     USC
#> 2     1.65        149  20   Male         AZ     UCI
#> 3     1.90        187  21 Female         TX    UCLA
#> 4     1.80        183  19   Male         CA     USC
#> 5     1.73        175  20   Male         FL Chapman
#> 6     1.70        178  19 Female         NY    UCSD
```

Create a new column - simple case

- Note that after creating a new column, it is immediately available for use

```
# convert weight to kg
df[["weight_kg"]] ← df$weight_lbs / 2.205
df
#>   height_m weight_lbs age    sex home_state  school weight_kg
#> 1     1.70        151  22   Male         CA     USC  68.48073
#> 2     1.65        149  20   Male         AZ     UCI  67.57370
#> 3     1.90        187  21 Female         TX    UCLA  84.80726
#> 4     1.80        183  19   Male         CA     USC  82.99320
#> 5     1.73        175  20   Male         FL Chapman  79.36508
#> 6     1.70        178  19 Female         NY    UCSD  80.72562
# compute bmi
df[["bmi"]] ← df$weight_kg / (df$height_m)^2
df
#>   height_m weight_lbs age    sex home_state  school weight_kg      bmi
#> 1     1.70        151  22   Male         CA     USC  68.48073 23.69575
#> 2     1.65        149  20   Male         AZ     UCI  67.57370 24.82046
#> 3     1.90        187  21 Female         TX    UCLA  84.80726 23.49231
#> 4     1.80        183  19   Male         CA     USC  82.99320 25.61518
#> 5     1.73        175  20   Male         FL Chapman  79.36508 26.51779
#> 6     1.70        178  19 Female         NY    UCSD  80.72562 27.93274
```

Aside: conditional values with `ifelse()`

- Often, you want to create a new column given some condition in your existing columns
- The `ifelse()` function allows us to return a value given some condition
- The arguments to `ifelse()` are:
  - `test`: a logical expression that evaluates to `TRUE` or `FALSE`
  - `yes`: the value returned if test is `TRUE`
  - `no`: the value returned if test is `FALSE`

```r
x <- c(-2, -5, 0, -2, 3, 4)

ifelse(test = x < 0 & x != 0,
       yes = "Negative",
       no = "Positive")
#> [1] "Negative" "Negative" "Positive" "Negative" "Positive" "Positive"
```

Create a new column on a condition

- Let's create a column to indicate if a student is from out-of-state

```
# create an out-of-state column
df["out_of_state"] ←
  ifelse(test = df$home_state ≠ "CA",
         yes = "Yes",
         no = "No")

df
#>    height_m weight_lbs age    sex home_state  school weight_kg       bmi out
#> 1      1.70        151  22   Male         CA     USC  68.48073  23.69575
#> 2      1.65        149  20   Male         AZ     UCI  67.57370  24.82046
#> 3      1.90        187  21 Female         TX    UCLA  84.80726  23.49231
#> 4      1.80        183  19   Male         CA     USC  82.99320  25.61518
#> 5      1.73        175  20   Male         FL Chapman  79.36508  26.51779
#> 6      1.70        178  19 Female         NY    UCSD  80.72562  27.93274
```

Filter data given a condition

- Filter data by combining logical expressions with subsetting
- Often, we would like to see a subset of our data given some condition
- The general syntax to filter using subsetting operator:

```
df[logical_expression, selected_columns]
```

Filter data given a condition

- Let's view all the students who are older than 20

```
df[df$age > 20, ]
#>   height_m weight_lbs age    sex home_state school weight_kg      bmi out_
#> 1      1.7        151  22   Male         CA    USC  68.48073 23.69575
#> 3      1.9        187  21 Female         TX   UCLA  84.80726 23.49231
```

- Let's view all the male students

```
df[df$sex == "Male", ]
#>   height_m weight_lbs age  sex home_state  school weight_kg      bmi out_o
#> 1     1.70        151  22 Male         CA     USC  68.48073 23.69575
#> 2     1.65        149  20 Male         AZ     UCI  67.57370 24.82046
#> 4     1.80        183  19 Male         CA     USC  82.99320 25.61518
#> 5     1.73        175  20 Male         FL Chapman  79.36508 26.51779
```

Select and filter data with `subset()`

- It turns out there is a built-in function `subset()` that allows you to select and filter data all in the same function

```
subset(x = df, subset = sex == "Male", select = c("home_state", "school"))
#>   home_state  school
#> 1         CA     USC
#> 2         AZ     UCI
#> 4         CA     USC
#> 5         FL Chapman

# default to return all columns
subset(x = df, subset = age > 20)
#>   height_m weight_lbs age    sex home_state school weight_kg      bmi out_
#> 1      1.7        151  22   Male         CA    USC  68.48073 23.69575
#> 3      1.9        187  21 Female         TX   UCLA  84.80726 23.49231
```

Rename column

- Rename columns by subsetting the column names of a data frame and assigning new names

```r
df2 <- data.frame(EmaIl = c("chaua3@uci.edu",
                            "peteranteater@uci.edu"),
                  School.name = c("UCI", "UCI"))

names(df2)
#> [1] "EmaIl"      "School.name"

# first, select the column names you want to change
names(df2)[names(df2) == "EmaIl"]
#> [1] "EmaIl"

# then, assign the old name with a new name
names(df2)[names(df2) == "EmaIl"] <- "email"

names(df2)
#> [1] "email"      "School.name"
```

Group data into subsets