Modifying values

Priyadarsini Dasari

Changing values in place

Basically this section deals with modifying values inside of a dataset.

You can use R's notation system to modify values within an R object.

You can use an assignment operator to overwrite values in an object.

```
vec <- c(0, 0, 0, 0, 0, 0)
vec
##  0 0 0 0 0 0
```

```
vec[1]
##  0
```

```
vec[1] <- 1000
vec
## 1000    0    0    0    0    0
```

You can replace multiple values at once as long as the number of new values equals the number of selected values:

```
vec[c(1, 3, 5)] <- c(1, 1, 1)
vec
##  1 0 1 0 1 0


vec[4:6] <- vec[4:6] + 1
vec
## 1 0 1 1 2 1
```

You can also create values that do not yet exist in your object. R will expand the object to accommodate the new values:

```
vec[7] <- 0
vec
## 1 0 1 1 2 1 0
```

This provides a great way to add new variables to your data set.

```r
deck2$new <- 1:52


head(deck2)
##   face   suit value new
##   king spades    13   1
##  queen spades    12   2
##   jack spades    11   3
##    ten spades    10   4
##   nine spades     9   5
##  eight spades     8   6
```

```r
deck2$new <- NULL


head(deck2)
##   face   suit value
##   king spades    13
##  queen spades    12
##   jack spades    11
##    ten spades    10
##   nine spades     9
##  eight spades     8
```

For a game of War, aces are of highest value i.e. 14 compared to value 1. To play this game, the values of aces need to be changed from 1 to 14.

In a deck that was not shuffled, we know that aces appear after every 1 cars. We can use R's notation system modify these values.

```
deck2[c(13, 26, 39, 52), ]
##  face     suit value
##   ace   spades     1
##   ace    clubs     1
##   ace diamonds     1
##   ace   hearts     1
```

You can single out the *values* of the aces by subsetting the columns dimension or subset the column vector

```
deck2[c(13, 26, 39, 52), 3]
## 1 1 1 1


deck2$value[c(13, 26, 39, 52)]
## 1 1 1 1
```

Once we subset these values, we can replace them with new ones using R notations like c or <-

```
deck2$value[c(13, 26, 39, 52)] <- c(14, 14, 14, 14)


# or


deck2$value[c(13, 26, 39, 52)] <- 14
```

Now all the values of aces have been replaced in the dataset.

```
head(deck2, 13)
##    face    suit value
##    king spades    13
##   queen spades    12
##    jack spades    11
##     ten spades    10
##    nine spades     9
##   eight spades     8
##   seven spades     7
##     six spades     6
##    five spades     5
##    four spades     4
##   three spades     3
##     two spades     2
##     ace spades    14
```

The same technique will work whether data is stored in a vector, matrix, array, list, or data frame. Just identify values that you want to change with R's notation system, then change those values with R's assignment operator.

However, in the case where the virtual card deck was shuffled, we have to figure out where the aces are located and then change the values to suit the game. You can do that using logical sub-setting.

Logical subsetting

Table 7.1: R's Logical Operators

Operator	Syntax	Tests
`>`	`a > b`	Is a greater than b?
`>=`	`a >= b`	Is a greater than or equal to b?
`<`	`a < b`	Is a less than b?
`<=`	`a <= b`	Is a less than or equal to b?
`==`	`a == b`	Is a equal to b?
`!=`	`a != b`	Is a not equal to b?
`%in%`	`a %in% c(a, b, c)`	Is a in the group c(a, b, c)?

Each operator returns a TRUE or a FALSE. If you use an operator to compare vectors, R will do element-wise comparisons—just like it does with the arithmetic operators

```
1 > 2

## FALSE


1 > c(0, 1, 2)

## TRUE FALSE FALSE


c(1, 2, 3) == c(3, 2, 1)

## FALSE  TRUE FALSE
```

%in% is the only operator that does not do normal element-wise execution. %in% tests whether th value(s) on the left side are in the vector on the right side

```
1 %in% c(3, 4, 5)
## FALSE


c(1, 2) %in% c(3, 4, 5)
## FALSE FALSE


c(1, 2, 3) %in% c(3, 4, 5)
## FALSE FALSE  TRUE


c(1, 2, 3, 4) %in% c(3, 4, 5)
## FALSE FALSE  TRUE  TRUE
```

Note: Be careful not to confuse = with ==. = does the same thing as <-: it assigns a value to an object.

Example of changing values for aces:

Step 1: Count each value of ace either by using the $ notation or == operator and sum

```
deck2$face
##  "king"  "queen" "jack"  "ten"   "nine"
##  "eight" "seven" "six"   "five"  "four"
##  "three" "two"   "ace"   "king"  "queen"
##  "jack"  "ten"   "nine"  "eight" "seven"
##  "six"   "five"  "four"  "three" "two"
##  "ace"   "king"  "queen" "jack"  "ten"
##  "nine"  "eight" "seven" "six"   "five"
##  "four"  "three" "two"   "ace"   "king"
##  "queen" "jack"  "ten"   "nine"  "eight"
##  "seven" "six"   "five"  "four"  "three"
##  "two"   "ace"
```

```
deck2$face == "ace"
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE  TRUE
```

```
sum(deck2$face == "ace")
## 4
```

Step 2: Build a logical test to spot the ace value, single out the ace value by using the logical test and change the ace values using assignment notation.

```
deck3$face == "ace"
```

```
deck3$value[deck3$face == "ace"]
##  1 1 1 1
```

Logical subsetting is a powerful technique because it lets you quickly identify, extract, and modify individual values in your data set. When you work with logical subsetting, you do not need to know *where* in your data set a value exists. You only need to know how to describe the value with a logical test.

```
deck3$value[deck3$face == "ace"] <- 14


head(deck3)
##  face     suit value
## queen    clubs    12
##  king    clubs    13
##   ace   spades    14  # an ace
##  nine    clubs     9
## seven   spades     7
## queen diamonds    12
```

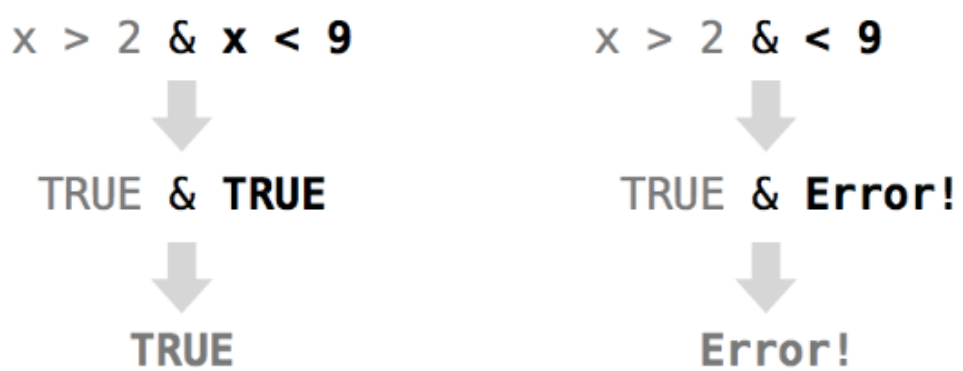
Boolean operators

Boolean Operators: combine multiple logical tests together into a single test.

Table 7.2: Boolean operators

Operator	Syntax	Tests
`&`	`cond1 & cond2`	Are both `cond1` and `cond2` true?
`|`	`cond1 | cond2`	Is one or more of `cond1` and `cond2` true?
`xor`	`xor(cond1, cond2)`	Is exactly one of `cond1` and `cond2` true?
`!`	`!cond1`	Is `cond1` false? (e.g., `!` flips the results of a logical test)
`any`	`any(cond1, cond2, cond3, ...)`	Are any of the conditions true?
`all`	`all(cond1, cond2, cond3, ...)`	Are all of the conditions true?

It is easy to forget to put a complete test on either side of a Boolean operator. In English, it is efficient to say "Is *x* greater than two and less than nine?" But in R, you need to write the equivalent of "Is *x* greater than two and *is x* less than nine?"



When used with vectors, Boolean operators will follow the same element-wise execution as arithmetic and logical operators:

```
a <- c(1, 2, 3)
b <- c(1, 2, 3)
c <- c(1, 2, 4)


a == b
##  TRUE TRUE TRUE


b == c
##  TRUE  TRUE FALSE


a == b & b == c
##  TRUE  TRUE FALSE
```

Example to locate queen of spades in the deck using Boolean operator, select the value and change its value to suit a game of hearts:

```
deck4$face == "queen" & deck4$suit == "spades"
##  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  FALSE FALSE FALSE
```

```
queenOfSpades <- deck4$face == "queen" & deck4$suit == "spades"
```

```
deck4[queenOfSpades, ]
##  face   suit value
## queen spades     0

deck4$value[queenOfSpades]
## 0
```

```
deck4$value[queenOfSpades] <- 13

deck4[queenOfSpades, ]
##  face   suit value
## queen spades     13
```

Missing information

In blackjack, each number card has a value equal to its face value. Each face card (king, queen, or jack) has a value of 10. Finally, each ace has a value of 11 or 1, depending on the final results of the game. It is hard to decide what value to give the aces because their exact value will change from hand to hand. At the end of each hand, an ace will equal 11 if the sum of the player's cards does not exceed 21. Otherwise, the ace will equal 1. The actual value of the ace will depend on the other cards in the player's hand. This is a case of missing information. At the moment, you do not have enough information to assign a correct point value to the ace cards. Vey common problem in data science.

The NA character is a special symbol in R. It stands for “not available” and can be used as a placeholder for missing information. R will treat NA exactly as you should want missing information treated.

```
1 + NA
## NA
```

```
NA == 1
## NA
```

Missing values can help you work around holes in your data sets, but they can also create some frustrating problems. Such as taking an average of 1000 observations. Even if one value is NA, the answer is NA.

Can identify if a value is NA using is.na function

```
is.na(NA)
## TRUE


vec <- c(1, 2, 3, NA)
is.na(vec)
## FALSE FALSE FALSE  TRUE
```

For the blackjack case, we can set all the ace values to NA thus helping in scoring an ace correctly after it's final value is determined.

```
deck5$value[deck5$face == "ace"] <

head(deck5, 13)
##    face    suit value
##    king spades    10
##   queen spades    10
##    jack spades    10
##     ten spades    10
##    nine spades     9
##   eight spades     8
##   seven spades     7
##     six spades     6
##    five spades     5
##    four spades     4
##   three spades     3
##     two spades     2
##     ace spades    NA
```