Non-Numeric Values

Pedro Fonseca

Introduction to R

May 9, 2020

Introduction

- Statistical programming sometimes requires non-numeric values.
- Examples of non numerical values in R: characters and logical values.

What are logical values?

- Logical values can only have two values: TRUE or FALSE.
- Logical values in R can be abbreviated as T or F.

```
> foo <- TRUE
> foo
[1] TRUE

> bar <- F
> bar
[1] FALSE
```

Logical vectors and matrices

- A logical vector:

```
> baz <- c(TRUE, FALSE, FALSE, FALSE, TRUE, FALSE)
> baz
[1]  TRUE FALSE FALSE FALSE  TRUE FALSE
```

- A logical matrix:

```
> qux <- matrix(data = baz, nrow = 3, ncol = 2)
> qux
      [,1]  [,2]
[1,]  TRUE FALSE
[2,] FALSE  TRUE
[3,] FALSE FALSE
```

Basic logical operators

Operator	Interpretation
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Figure 1: Basic logical operators in R

Logical comparisons

- Logicals can be used to check relationships between values:

```
> 1 == 2
[1] FALSE

> 1>2
[1] FALSE

> (2-1) <= 2
[1] TRUE

> 1 != (2+3)
[1] TRUE
```

Logical comparisons

- Logicals can be used to check relationships between variables:

```
> x <- log(5, base = 10)
> y <- log(5, base = exp(1))

> y == x
[1] FALSE
> y >= x
[1] TRUE

> c(x, y)
[1] 0.698970 1.609438
```

Vectorized logical comparison


```
> vec_x <- c(1, 5, 6, 7, 3)
> vec_y <- c(1, 6, 8, 2, 1)

> vec_x == 5
[1] FALSE  TRUE FALSE FALSE FALSE
> vec_x != 5
[1]  TRUE FALSE  TRUE  TRUE  TRUE
> vec_x >= vec_y
[1]  TRUE FALSE FALSE  TRUE  TRUE
```

The ! operator

- "!" is the negation (NOT) operator

```
> !TRUE
[1] FALSE

> !FALSE
[1] TRUE

> !c(TRUE, FALSE, TRUE, TRUE, FALSE)
[1] FALSE  TRUE FALSE FALSE  TRUE
```

The ! operator


```
> x <- c(FALSE, TRUE)
> !x
[1]  TRUE FALSE
```

The ! operator


```
> vec_x <- c(1, 5, 6, 7, 3)
> vec_y <- c(1, 6, 8, 2, 1)

> vec_x >= vec_y
[1]  TRUE FALSE FALSE  TRUE  TRUE

> !(vec_x >= vec_y)
[1] FALSE  TRUE  TRUE FALSE FALSE
```

The ! operator


```
> vec_x <- c(1, 5, 6, 7, 3)
> vec_y <- c(1, 6, 8, 2, 1)

> vec_x == 5
[1] FALSE  TRUE FALSE FALSE FALSE
> !(vec_x == 5)
[1]  TRUE FALSE  TRUE  TRUE  TRUE

> vec_x == vec_y
[1]  TRUE FALSE FALSE FALSE FALSE
```

! vs *Factorial*

- Do not confuse "!" with the factorial function
- In R, the factorial function is *factorial*:

```
> factorial(5)
[1] 120
> 5 * 4 * 3 * 2 * 1
[1] 120
```

- Alternatively, can use the product function:

```
> prod(5:1)
[1] 120
> prod(5, 4, 3, 2, 1)
[1] 120
```

Subset with logicals


```
> myvec <- c(5, -2.3, 4, 4, 1)

> myvec[c(TRUE, FALSE, TRUE, FALSE, FALSE)]
[1] 5 4
```

- Recycling rules apply as usual:

  ```
  > myvec[c(TRUE, FALSE)]
  [1] 5 4 1
  ```

Subset with logicals


```
> mymat <- matrix(1:9, nrow = 3)
> mymat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> mymat[, c(TRUE, FALSE, TRUE)]
     [,1] [,2]
[1,]    1    7
[2,]    2    8
[3,]    3    9
```

Subset with logicals


```
> mymat[c(TRUE, TRUE, FALSE), ]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8

> mymat[c(TRUE, TRUE, FALSE), c(TRUE, TRUE, FALSE)]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
```

Subset with logicals


```
> myvec <- c(5, -2.3, 4, 4, -1)

> myvec < 0
[1] FALSE  TRUE FALSE FALSE  TRUE
> myvec[myvec < 0]
[1] -2.3 -1.0

> myvec != 4
[1]  TRUE  TRUE FALSE FALSE  TRUE
> myvec[myvec != 4]
[1]  5.0 -2.3 -1.0
```

Overwriting with logicals


```
> mymat <- matrix(1:9, nrow = 3, byrow = TRUE)
> mymat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> mymat < 5
      [,1]  [,2]  [,3]
[1,]  TRUE  TRUE  TRUE
[2,]  TRUE FALSE FALSE
[3,] FALSE FALSE FALSE
```

Overwriting with logicals


```
> mymat[mymat < 5]
[1] 1 4 2 3

> mymat[mymat < 5] <- 0
> mymat
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    5    6
[3,]    7    8    9
```

Overwriting with logicals


```
> mymat <- matrix(1:9, nrow = 3, byrow = TRUE)
> mymat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> which(mymat < 5)
[1] 1 2 4 7

> mymat[which(mymat < 5)] <- 0 # same result as the
>       # previous slide
```

Introduction

- In R, missing values are represented by NA (not available)
- NAs are exceptional logical values.
- The *is.logical* testes whether or not values are logic:

```
> is.logical(54)
[1] FALSE
> is.logical(myvec)
[1] FALSE
> is.logical(TRUE)
[1] TRUE
> is.logical(FALSE)
[1] TRUE
> is.logical(NA)
[1] TRUE
```

Introduction

- In the previous lesson, we dealt with numerical values
- There is also a test function for numerics:

```
> is.numeric(54)
[1] TRUE
> is.numeric(myvec)
[1] TRUE
> is.numeric(TRUE)
[1] FALSE
> is.numeric(FALSE)
[1] FALSE
> is.numeric(NA)
[1] FALSE
```

Operations with NAs

- When NAs are present, caution is required!

```
> y <- c(1, 2, 3, NA)

> NA + 5
[1] NA
 > (NA + 3) > 0
[1] NA

> sum(1, 5, 6, NA, 7)
[1] NA
> mean(y)
[1] NA
```

Dealing with missing values

- Many functions have a *na.rm* argument. It is set to FALSE by default.

```
> y <- c(1, 2, 3, 7, 0.5, NA)
> y_without_nas <- y[-6]

> mean(y, na.rm = TRUE)
[1] 2.7
> mean(y_without_nas)
[1] 2.7
> sum(1, 5, 6, NA, 7, na.rm = TRUE)
[1] 19
> sum(1, 5, 6, 7)
[1] 19
```

Dealing with missing values

- Other option: *na.omit*

```
> y <- c(1, 2, 3, 7, 0.5, NA)
> na.omit(y)
[1] 1.0 2.0 3.0 7.0 0.5

> y_new <- na.omit(y)
> y_new
[1] 1.0 2.0 3.0 7.0 0.5
```

Dealing with missing values

- *na.omit* with matrices:

```
> (M <- matrix(c(1:3, NA, c(5, 9)), nrow = 3,
 byrow = TRUE))
     [,1] [,2]
[1,]    1    2
[2,]    3   NA
[3,]    5    9
>
> na.omit(M)
     [,1] [,2]
[1,]    1    2
[2,]    5    9
```

Dealing with missing values

- In matrices (and data frames) *na.omit* returns only complete rows!
- This can be useful in data analysis when only complete cases are to be considered.
- However, sometimes we have small samples and can't afford to throw away incomplete observations.
  - Solution: imputation

Testing for missing values


```
> y <- c(1, 2 , 3, NA)

> is.na(y)
[1] FALSE FALSE FALSE  TRUE

> !is.na(y)
[1]  TRUE  TRUE  TRUE FALSE
```

Testing for missing values

```
> M <- matrix(c(1:3, NA, c(5, 9)), 3, 2, TRUE)
> M
     [,1] [,2]
[1,]    1    2
[2,]    3   NA
[3,]    5    9

> is.na(M)
      [,1]  [,2]
[1,] FALSE FALSE
[2,] FALSE  TRUE
[3,] FALSE FALSE
```

Imputation of missing values


```
> M <- matrix(c(1:3, NA, c(5, NA)), 2, 3, TRUE)
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   NA    5   NA

> is.na(M)
      [,1]  [,2]  [,3]
[1,] FALSE FALSE FALSE
[2,]  TRUE FALSE  TRUE
```

Imputation of missing values


```
> M[is.na(M)] <- 0
> M

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    0    5    0
```

- Replacing missing values with zeros can result in severe underestimation of the actual values. Sometimes it better to replace NAs with an estimate of its value.

Imputation of missing values


```
> M <- cbind(
    y  = c(3, NA, 7, 1),
    x1 = c(1, 7,  9, 6),
    x2 = c(6, 7,  9, NA)
 )

> M
      y x1 x2
[1,]  3  1  6
[2,] NA  7  7
[3,]  7  9  9
[4,]  1  6 NA
```

Imputation of missing values


```
> M[, "y"]
[1]  3 NA  7  1

> is.na(M[, "y"])
[1] FALSE  TRUE FALSE FALSE

> M[is.na(M[, "y"]), "y"] <- median(M[, "y"],
 na.rm = TRUE)

> M[is.na(M[, "x2"]), "x2"] <- median(M[, "x2"],
 na.rm = TRUE)
```

Imputation of missing values

```
> M
     y x1 x2
[1,] 3  1  6
[2,] 3  7  7
[3,] 7  9  9
[4,] 1  6  7
```

Counting missing values

- The *table* function is useful here:

```
> y <- c(1, 2 , 3, NA, 6, NA, 9, NA)

> is.na(y)
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE

> table(is.na(y))
FALSE  TRUE
    5     3
```

Counting missing values

- *table* also works with matrices:

```
> M <- rbind(
    col1 <- c(2, 5, 7),
    col2 <- c(NA, 5, 7),
    col3 <- c(2, 5, NA)
  )

> table(is.na(M))

FALSE  TRUE
    7     2
```

Counting missing values

- Alternative:

```
> y <- c(1, 2 , 3, NA, 6, NA, 9, NA)
> sum(is.na(y))
[1] 3

> M <- rbind(
    col1 <- c(2, 5, 7),
    col2 <- c(NA, 5, 7),
    col3 <- c(2, 5, NA)
 )
> sum(is.na(M))
[1] 2
```

Coercion of logical values

- Why does *sum(is.na())* Work? R coerces logical values to numerical values if you use them in a context where numeric values are expected.
- How?

```
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
```

- The *as.numeric* function forces the coercion to numeric.

Coercion of logical values


```
> y <- c(1, 2 , 3, NA, 6, NA, 9, NA)
> is.na(y)
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> as.numeric(is.na(y))
[1] 0 0 0 1 0 1 0 1
> sum(as.numeric(is.na(y)))
[1] 3
```

- The *as.numeric* function here is redundant since R coerces the inputs of *sum* to numerical if possible.
- The same happens with matrices.

Removing NAs with the *which* function


```
> y <- c(1, 2 , 3, NA, 6, NA, 9, NA)
```

- How many NAs in y?

```
> length(which(is.na(y)))
[1] 3
```

- In which positions are the positions with non-NA values of y?

```
> which(is.na(y))
[1] 4 6 8
```

- Subset y to extract non-missing values only:

```
> y[which(!is.na(y))] # notice the "!"
[1] 1 2 3 6 9
```

Logical comparison with more than one condition

Operator	Interpretation	Results
&	AND (element-wise)	TRUE & TRUE is TRUE TRUE & FALSE is FALSE FALSE & TRUE is FALSE FALSE & FALSE is FALSE
&&	AND (single comparison)	Same as & above
|	OR (element-wise)	TRUE|TRUE is TRUE TRUE|FALSE is TRUE FALSE|TRUE is TRUE FALSE|FALSE is FALSE
||	OR (single comparison)	Same as | above
!	NOT	!TRUE is FALSE !FALSE is TRUE

Figure 2: The "and", "or" and "not" operators in R

Logical comparison with more than one condition

- Examples:

```
> (6 < 4) || (3 != 1)
[1] TRUE

> (6 < 4) || (3 == 1)
[1] FALSE

> (6 < 4) && (3 != 1)
[1] FALSE

> (6 > 4) && (3 >= 1)
[1] TRUE
```

Logical comparison with more than one condition

- Examples:

```
> y <- c(1, 5 ,3, 1, 2, 9)

> (y > 2) & (y < 6)
[1] FALSE  TRUE  TRUE FALSE FALSE FALSE

> (y > 2) | (y < 6)
[1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Logical comparison with more than one condition

- Examples:

```
> y <- c(1, 5 ,3, 1, 2, 9)

> y[(y > 2) & (y < 6)]
[1] 5 3

> y[(y > 2) | (y < 6)]
[1] 1 5 3 1 2 9
```

Logical comparison with more than one condition

- You can chain as many comparisons as you want:

```
> y[y > 2]
[1] 5 3 9

> y[(y > 2) | (y < 6)]
[1] 1 5 3 1 2 9

> y[((y > 2) | (y < 6)) & (y != 2)]
[1] 1 5 3 1 9
```

- When performing multiple comparisons, parentheses are recommended!

The *any* and *all* functions

- Given a set of logical vectors, is at least one of the values true?
- Given a set of logical vectors, are all of the values true?

```
> y <- c(1, 5 , 3, NA, 2, 9)

> any(y > 2, y < 6, !is.na(y))
[1] TRUE

> all(y > 2, y < 6, !is.na(y))
[1] FALSE
```

Exclusive OR

- *xor* indicates element-wise exclusive OR

```
> y <- c(1, 5 ,3, 1, 2, 9)

> xor(y>2, y<6)
[1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE
```

Logical comparison with more than one condition

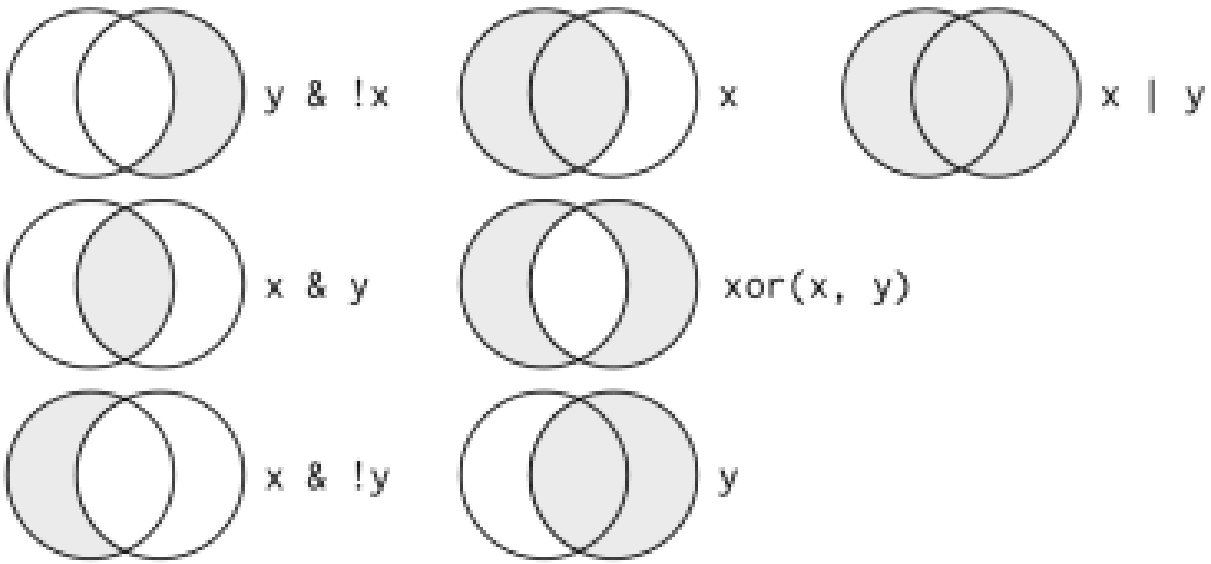

Figure 3: A visual perspective of logical operators