Introduction to R Software

Basics of Calculations
::::
Missing Data and Logical Operators

Shalabh

Department of Mathematics and Statistics

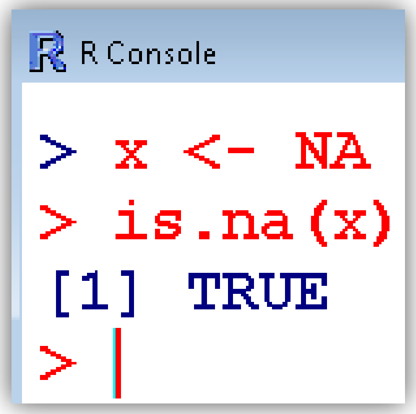
Indian Institute of Technology Kanpur

Missing data

R represents missing observations through the data value `NA`

We can detect missing values using `is.na`

```
> x <- NA     # assign NA to variable x
> is.na(x)    # is it missing?
 [1] TRUE
```

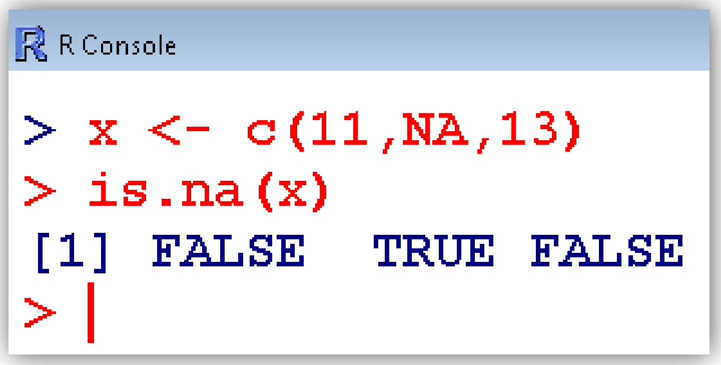


```
> x <- NA
> is.na(x)
[1] TRUE
> 
```

Missing data

Now try a vector to know if any value is missing?

```
> x <- c(11, NA, 13)

> is.na(x)

 [1] FALSE TRUE FALSE
```



```
> x <- c(11,NA,13)
> is.na(x)
[1] FALSE  TRUE FALSE
> |
```

Example : How to work with missing data

```
> x <- c(11,NA,13) # vector
```

```
> mean(x)
 [1] NA
```

$$\frac{11 + \text{NA} + 13}{2}$$

```
> mean(x, na.rm = TRUE) # NAs can be removed
 [1] 12
```

$$\frac{11+13}{2} = 12$$

The null object, called `NULL`, is returned by some functions and expressions.

Note that `NA` and `NULL` are not the same.

`NA` is a placeholder for something that exists but is missing.

`NULL` stands for something that never existed at all.

Logical Operators and Comparisons

The following table shows the operations and functions for logical comparisons (True or False).

`TRUE` and `FALSE` are reserved words denoting logical constants.

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation (not)

Logical Operators and Comparisons

Operator	Executions
`&, &&`	and
`|, ||`	or

- The <u>shorter form</u> performs element-wise comparisons in almost the same way as arithmetic operators.
- The <u>longer form</u> evaluates left to right examining only the first element of each vector. Evaluation proceeds only until the result is determined.
- The longer form is appropriate for programming control-flow and typically preferred in if clauses (conditional).

Logical Operators and Comparisons

TRUE and FALSE are reserved words denoting logical constants

Operator	Executions
`xor()`	either... or (exclusive)
`isTRUE(x)`	test if `x` is TRUE
`TRUE`	true
`FALSE`	false

Examples:

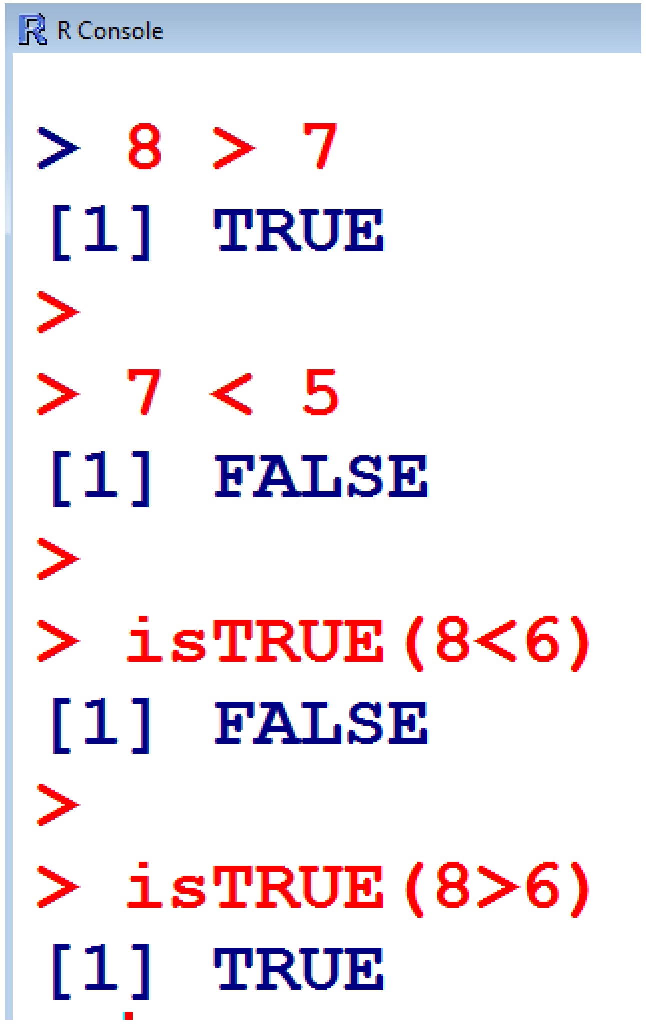
```
> 8 > 7
[1] TRUE

> 7 < 5
[1] FALSE
```

Is 8 less than 6?

```
> isTRUE(8<6)
[1] FALSE
```

Is 8 greater than 6?

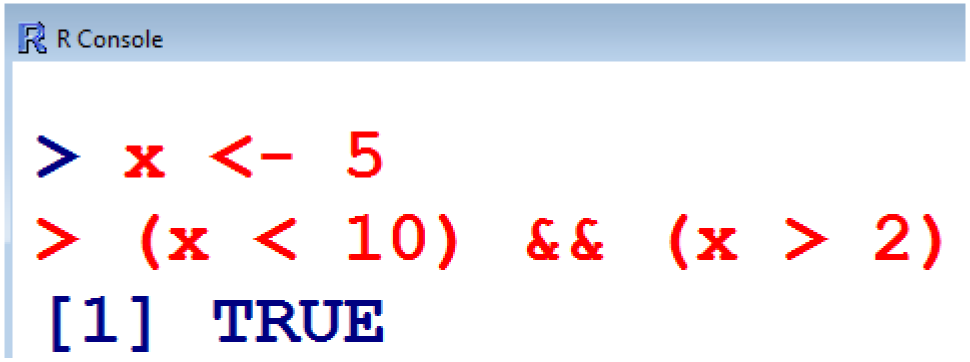
```
> isTRUE(8>6)
[1] TRUE
```



```
> 8 > 7
[1] TRUE
>
> 7 < 5
[1] FALSE
>
> isTRUE(8<6)
[1] FALSE
>
> isTRUE(8>6)
[1] TRUE
```

Examples:

```
> x <- 5
> (x < 10) && (x > 2)      # && means AND
[1] TRUE
```



```
> x <- 5
> (x < 10) && (x > 2)
[1] TRUE
```

Examples:

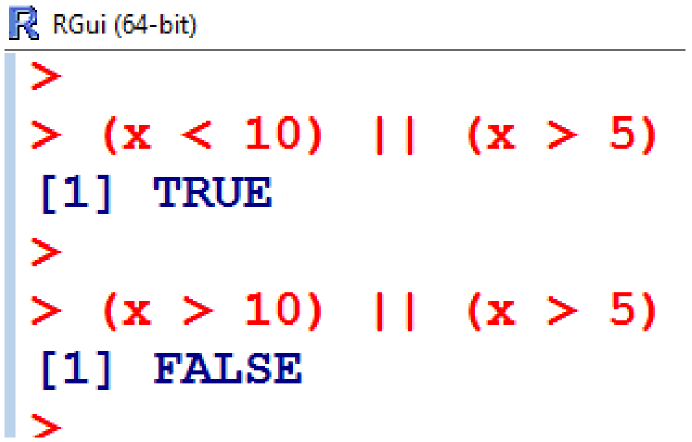
```
> x <- 5
```

Is **x** less than 10 or **x** is greater than 5 ?

```
> (x < 10) || (x > 5)       # || means OR
[1] TRUE
```

Is **x** greater than 10 or **x** is greater than 5 ?

```
> (x > 10) || (x > 5)
[1] FALSE
```



```
>
> (x < 10) || (x > 5)
[1] TRUE
>
> (x > 10) || (x > 5)
[1] FALSE
>
```

Examples:

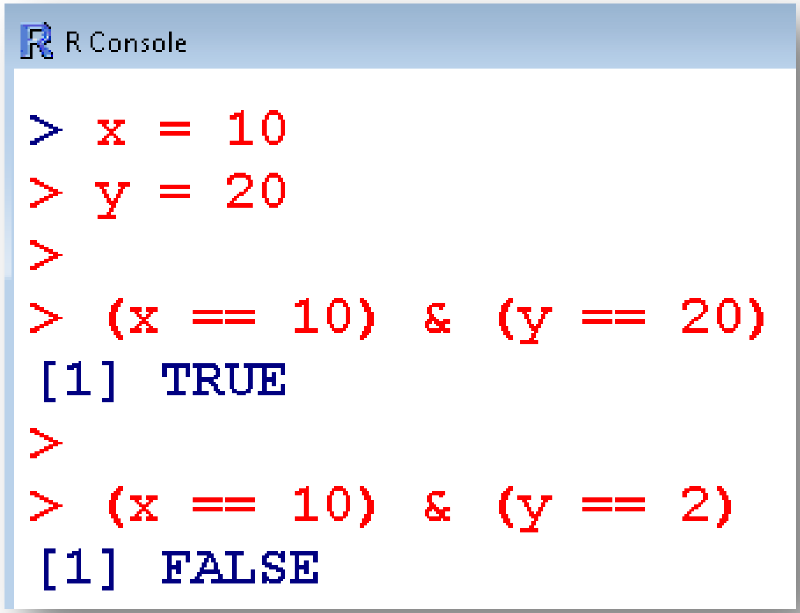
```
> x = 10
> y = 20
```

Is **x** equal to 10 and is **y** equal to 20?

```
> (x == 10) & (y == 20)        # == means exactly
                                     equal to

[1] TRUE
```

Is **x** equal to 10 and is **y** equal to 2?

```
> (x == 10) & (y == 2)
[1] FALSE
```


R Console

```
> x = 10
> y = 20
>
> (x == 10) & (y == 20)
[1] TRUE
>
> (x == 10) & (y == 2)
[1] FALSE
```

Examples:

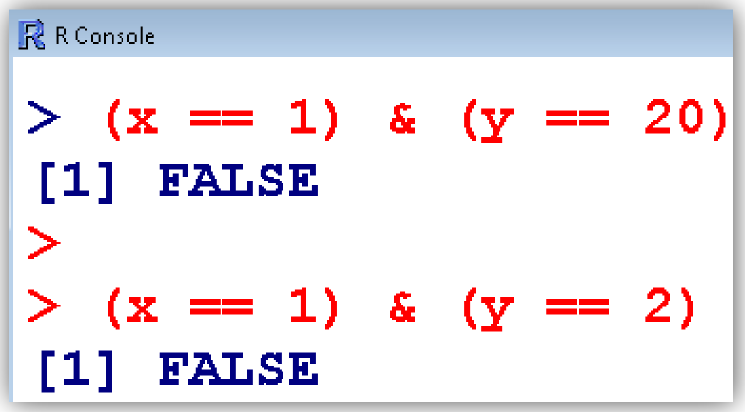
```
> x = 10
> y = 20
```

Is x equal to 1 and is y equal to 20?

```
> (x == 1) & (y == 20)        # == means exactly
                               equal to
[1] FALSE
```

Is x equal to 1 and is y equal to 2?

```
> (x == 1) & (y == 2)
[1] FALSE
```



```
> (x == 1) & (y == 20)
[1] FALSE
>
> (x == 1) & (y == 2)
[1] FALSE
```