Basics of Calculations

Matrix Operations, Missing Data, Logical Operators, Truth Table and Control Structures

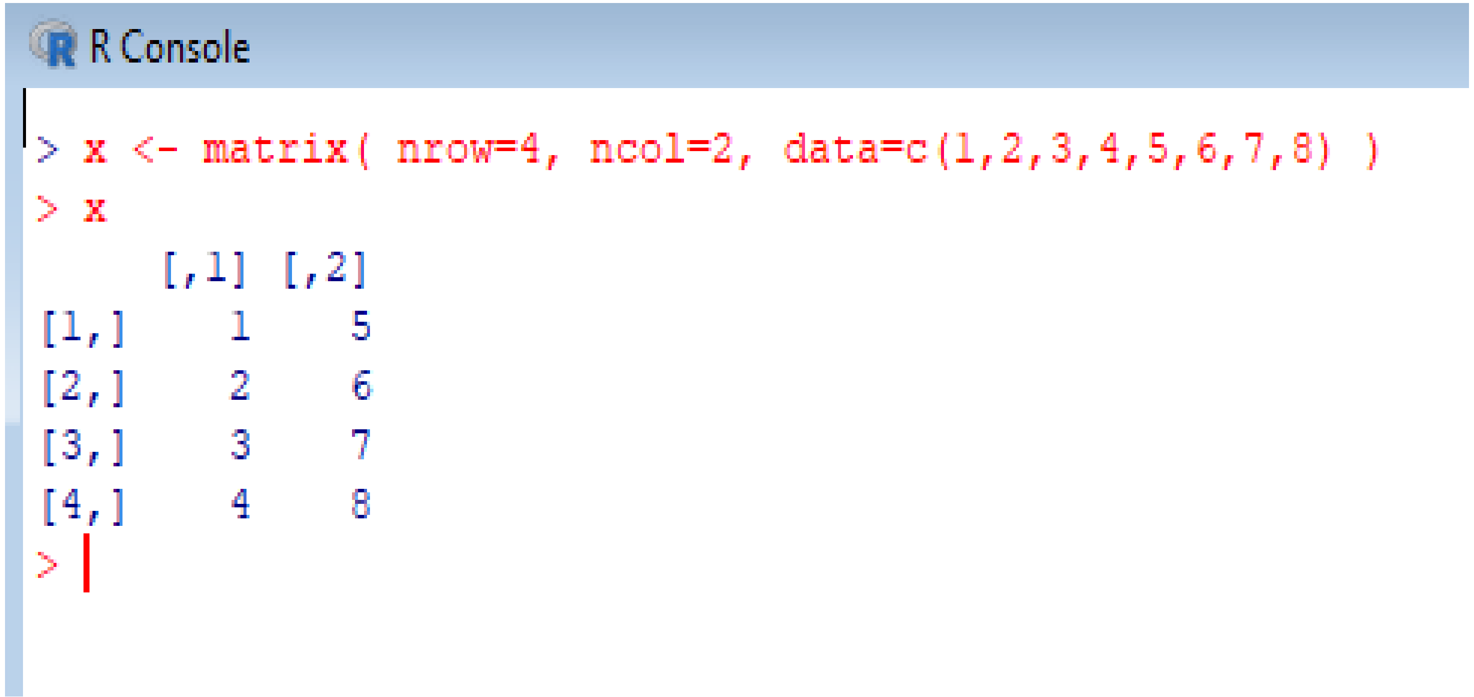
Kanahaiya Kumar [Data Mining Central]

Matrix Operations

In R, a 4 × 2-matrix

> x <- matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))

> x

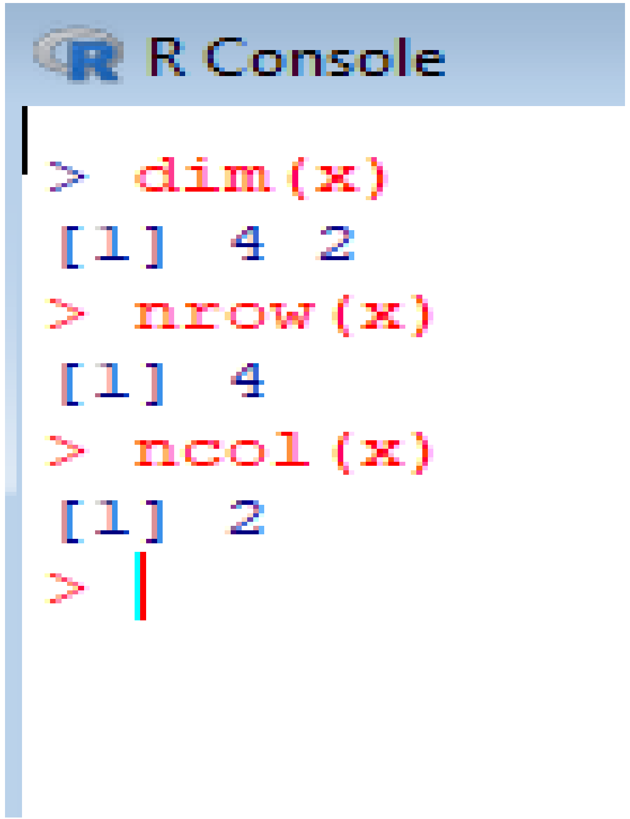


```
> x <- matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) )
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> 
```

Properties of Matrix

We can get specific properties of a matrix:

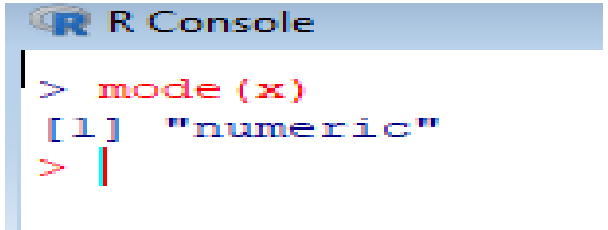
```
> dim(x) # tells the
[1] 4 2 dimension of matrix
> nrow(x) # tells
[1] 4 the number of rows
> ncol(x) # tells
[1] 2 the number of columns
```



```
> dim(x)
[1] 4 2
> nrow(x)
[1] 4
> ncol(x)
[1] 2
> 
```

Properties of Matrix

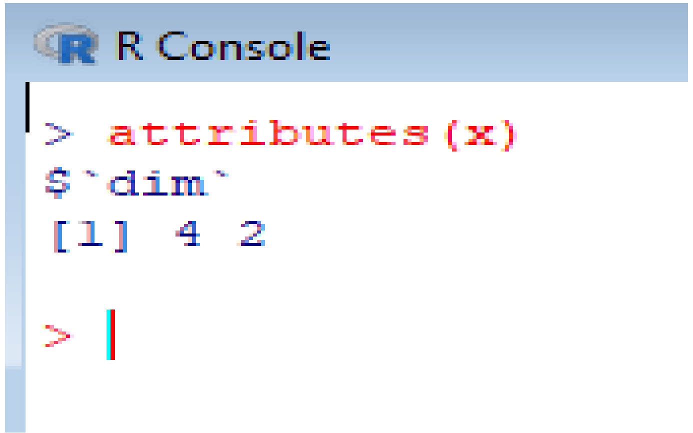
> mode(x) # Informs the type or storage mode of an object, e.g., numerical, logical etc.



```
> mode(x)
[1] "numeric"
>
```

attributes provides all the attributes of an object

> attributes(x) #Informs the dimension of matrix



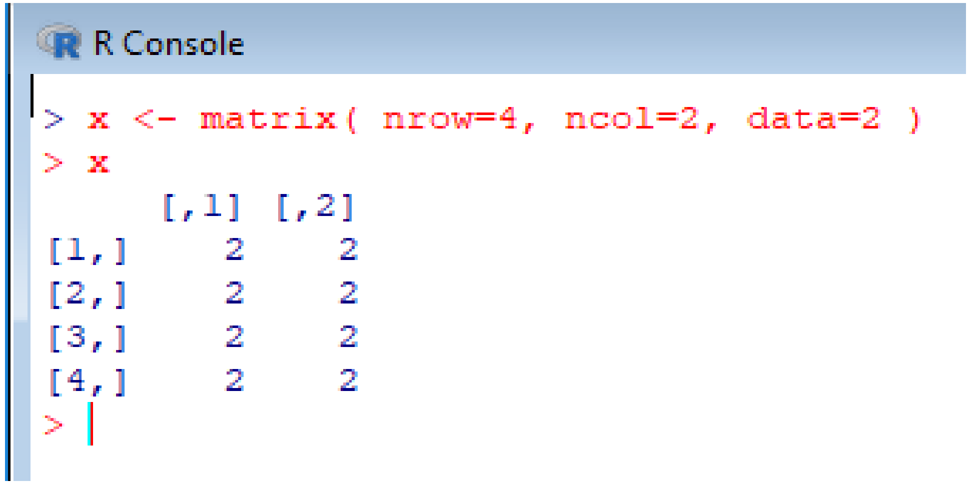
```
> attributes(x)
$`dim`
[1] 4 2

>
```

Matrix Operations

Assigning a specified number to all matrix elements:

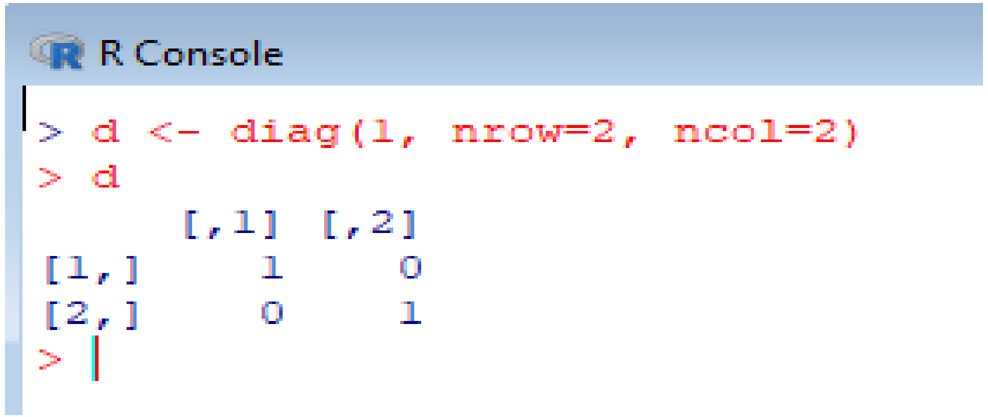
```
> x <- matrix( nrow=4, ncol=2, data=2 )
> x
```



```
> x <- matrix( nrow=4, ncol=2, data=2 )
> x
     [,1] [,2]
[1,]    2    2
[2,]    2    2
[3,]    2    2
[4,]    2    2
> 
```

Construction of a diagonal matrix, here the identity matrix of a dimension 2:

```
> d <- diag(1, nrow=2, ncol=2)
> d
```



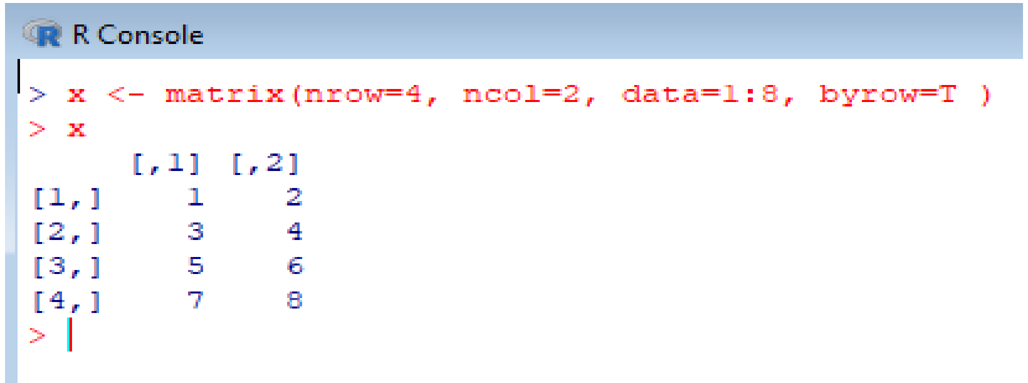
```
> d <- diag(1, nrow=2, ncol=2)
> d
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> 
```

Matrix Operations

Transpose of a matrix *X*: *X’*

> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T)

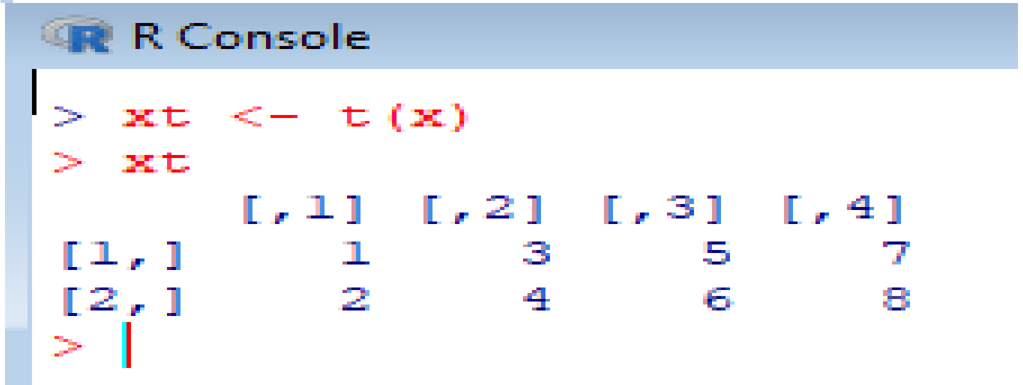
> x



```
> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T )
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> 
```

> xt <- t(x)

> xt



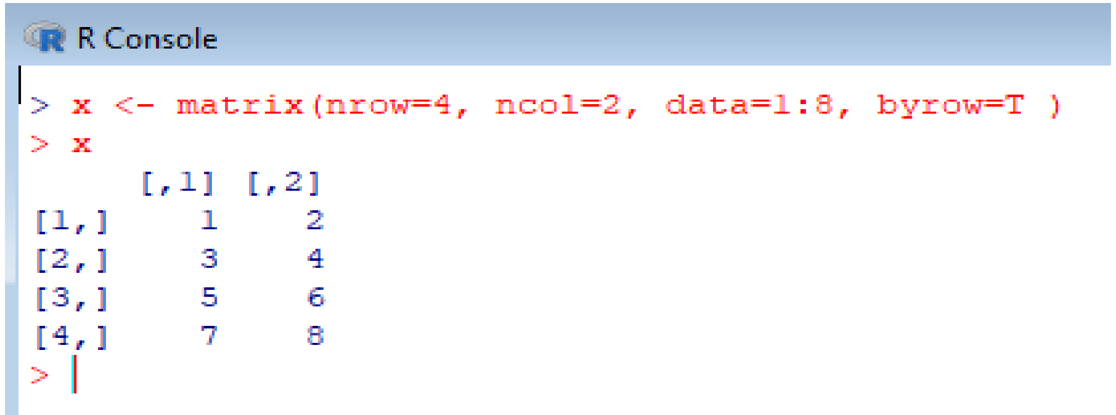
```
> xt <- t(x)
> xt
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> 
```

Matrix Operations

Multiplication of a matrix with a constant

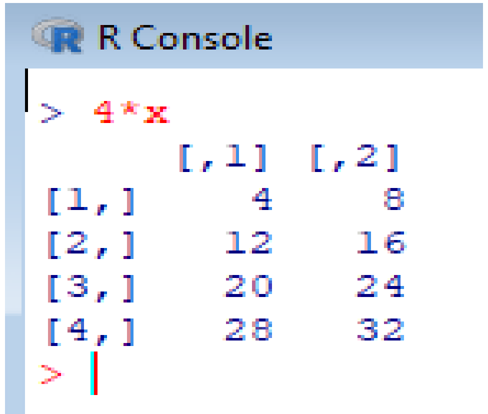
```
> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T )
```

```
> x
```



```
> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T )
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> 
```

```
> 4*x
```



```
> 4*x
     [,1] [,2]
[1,]    4    8
[2,]   12   16
[3,]   20   24
[4,]   28   32
> 
```

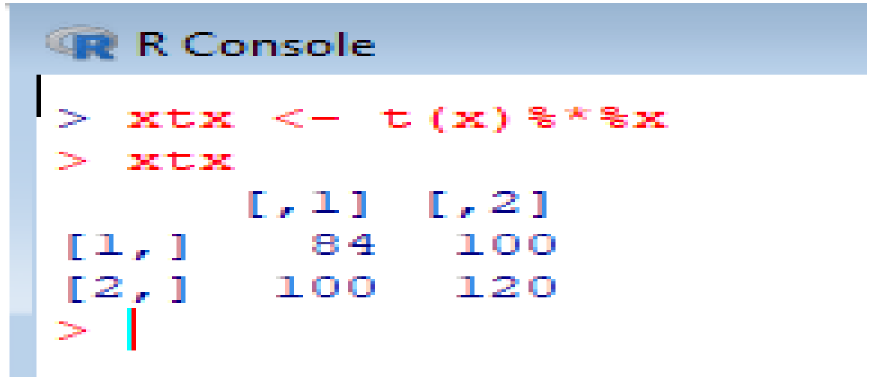
Matrix Operations

Matrix multiplication: operator %*%

Consider the multiplication of X’ with X

> xtx <- t(x) %% x*

> xtx



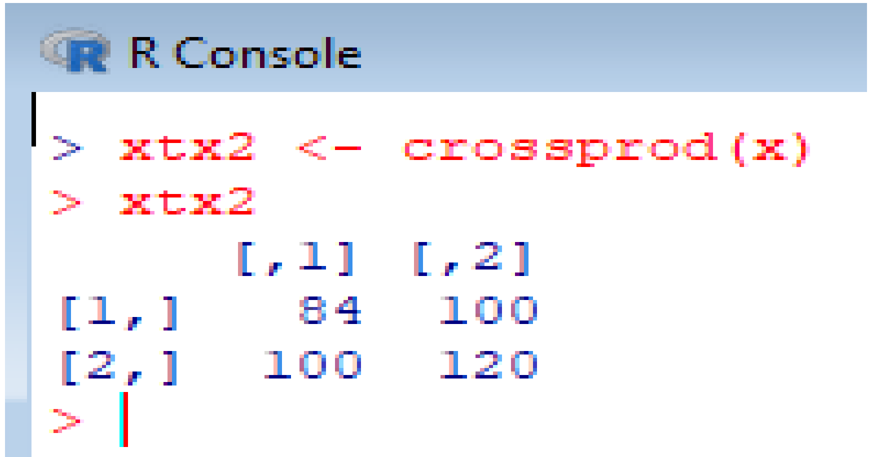
```
> xtx <- t(x)%*%x
> xtx
     [,1] [,2]
[1,]   84  100
[2,]  100  120
> 
```

Cross product of a matrix X, X’X, with a function crossprod

> xtx2 <- crossprod(x)

> xtx2

Note: Command crossprod()executes the multiplication faster than the conventional method with t(x)%*%x



```
> xtx2 <- crossprod(x)
> xtx2
     [,1] [,2]
[1,]   84  100
[2,]  100  120
> 
```

Matrix Operations

Addition and subtraction of matrices (of same dimensions) can be executed with the usual operators + and -

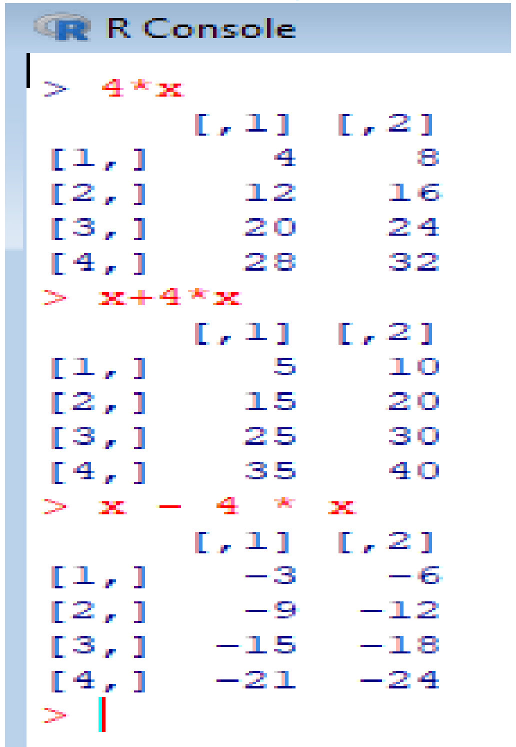
> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T)

> x

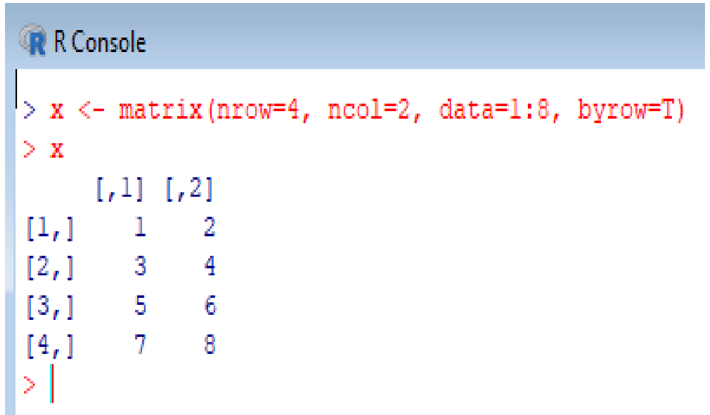
*> 4*x*

*> x + 4*x*

*> X – 4*x*



```
> 4*x
     [,1] [,2]
[1,]    4    8
[2,]   12   16
[3,]   20   24
[4,]   28   32
> x+4*x
     [,1] [,2]
[1,]    5   10
[2,]   15   20
[3,]   25   30
[4,]   35   40
> x - 4 * x
     [,1] [,2]
[1,]   -3   -6
[2,]   -9  -12
[3,]  -15  -18
[4,]  -21  -24
> 
```



```
> x <- matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> 
```

Matrix Operations

Access to rows, columns or submatrices:

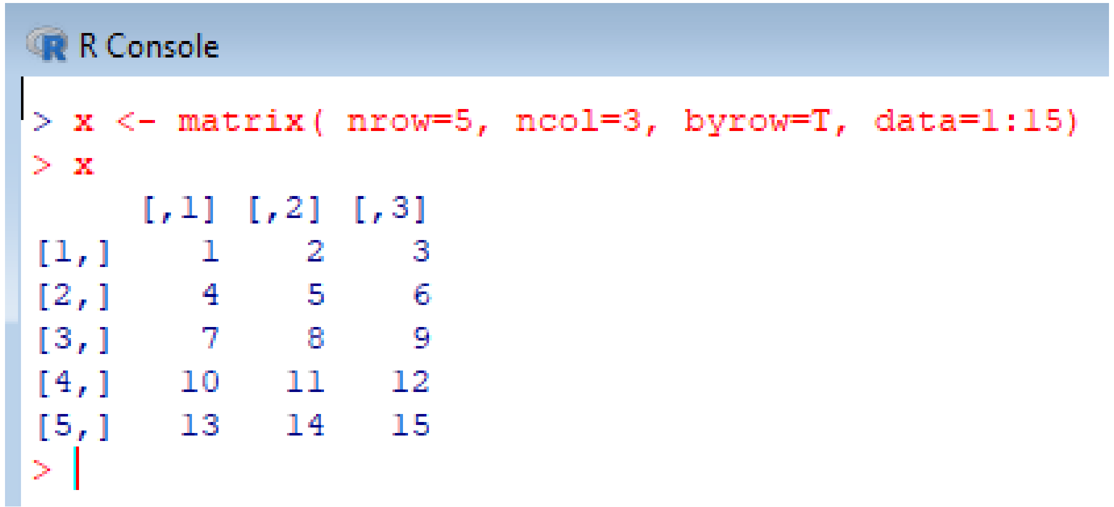
> x <- matrix(nrow=5, ncol=3, byrow=T, data=1:15)

> x

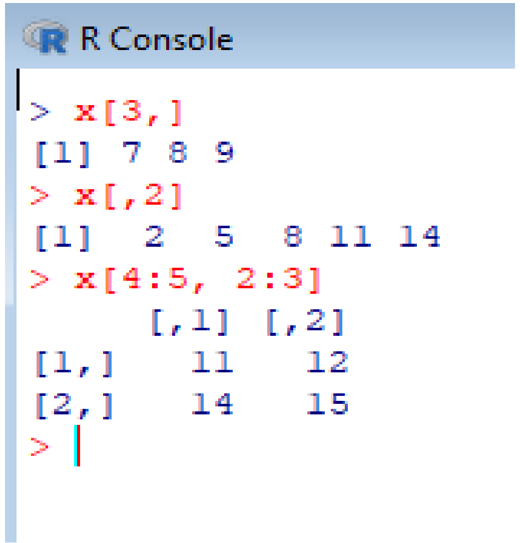
> x[3,]

> x[,2]

> x[4:5, 2:3]



```
> x <- matrix( nrow=5, ncol=3, byrow=T, data=1:15)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
[5,]   13   14   15
> 
```



```
> x[3,]
[1] 7 8 9
> x[,2]
[1]  2  5  8 11 14
> x[4:5, 2:3]
     [,1] [,2]
[1,]   11   12
[2,]   14   15
> 
```

Matrix Operations

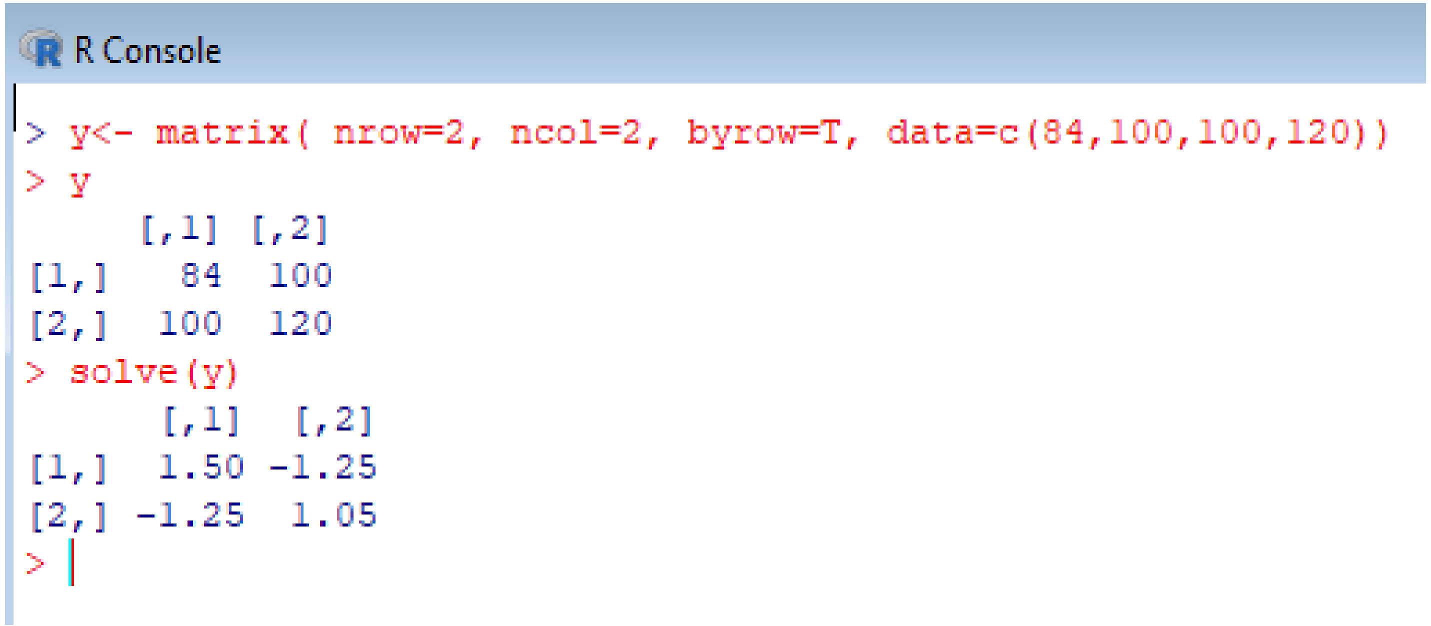
Inverse of a matrix:

solve() finds the inverse of a positive definite matrix

> y<- matrix(nrow=2, ncol=2, byrow=T, data=c(84,100,100,120))

> y

> solve(y)

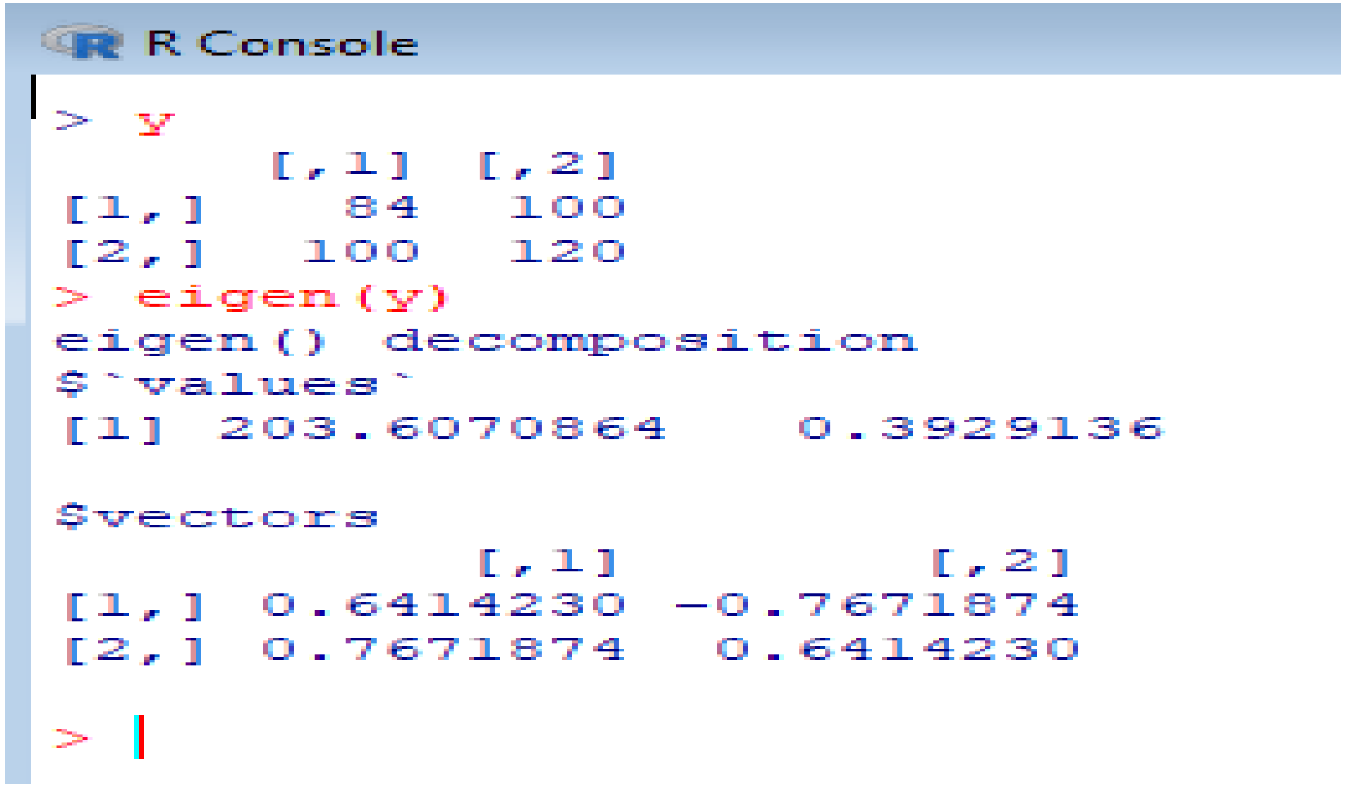


```
> y<- matrix( nrow=2, ncol=2, byrow=T, data=c(84,100,100,120))
> y
     [,1] [,2]
[1,]   84  100
[2,]  100  120
> solve(y)
      [,1]  [,2]
[1,]  1.50 -1.25
[2,] -1.25  1.05
> 
```

Matrix Operations

Eigen Values and Eigen Vectors:

eigen() finds the eigen values and eigen vectors of a positive definite matrix



```
> y
     [,1] [,2]
[1,]   84  100
[2,]  100  120
> eigen(y)
eigen() decomposition
$`values`
[1] 203.6070864   0.3929136

$vectors
          [,1]       [,2]
[1,] 0.6414230 -0.7671874
[2,] 0.7671874  0.6414230

> 
```

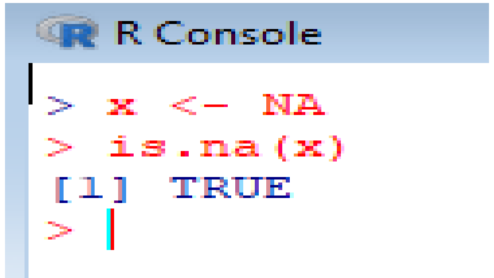
Missing Data

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?

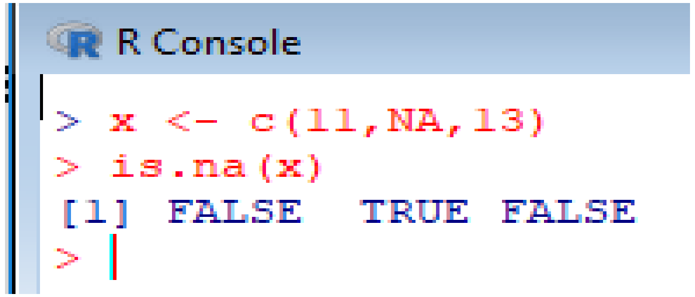


```
> x <- NA
> is.na(x)
[1] TRUE
>
```

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

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

> is.na(x)



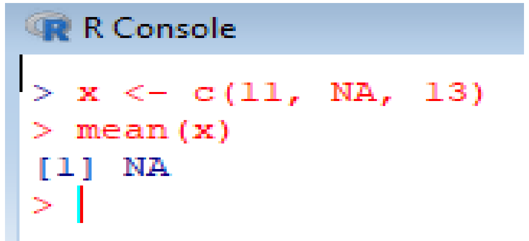
```
> x <- c(11,NA,13)
> is.na(x)
[1] FALSE  TRUE FALSE
>
```

Missing Data

How to work with missing data?

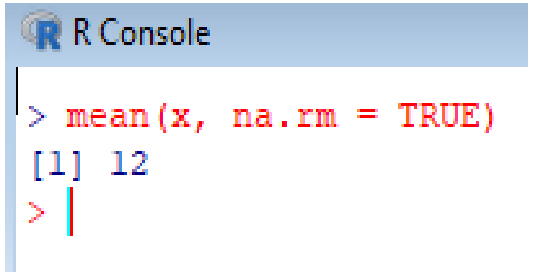
> x <- c(11,NA,13) # vector

> mean(x) $\frac{11+NA+13}{2}$



```
> x <- c(11, NA, 13)
> mean(x)
[1] NA
> 
```

> mean(x, na.rm = TRUE) # NAs can be removed



```
> mean(x, na.rm = TRUE)
[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 & 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 to*
==	*Exactly equal to*
!=	*Not equal to*
!	*Negation (Not)*
&, &&	*And*
|, ||	*or*

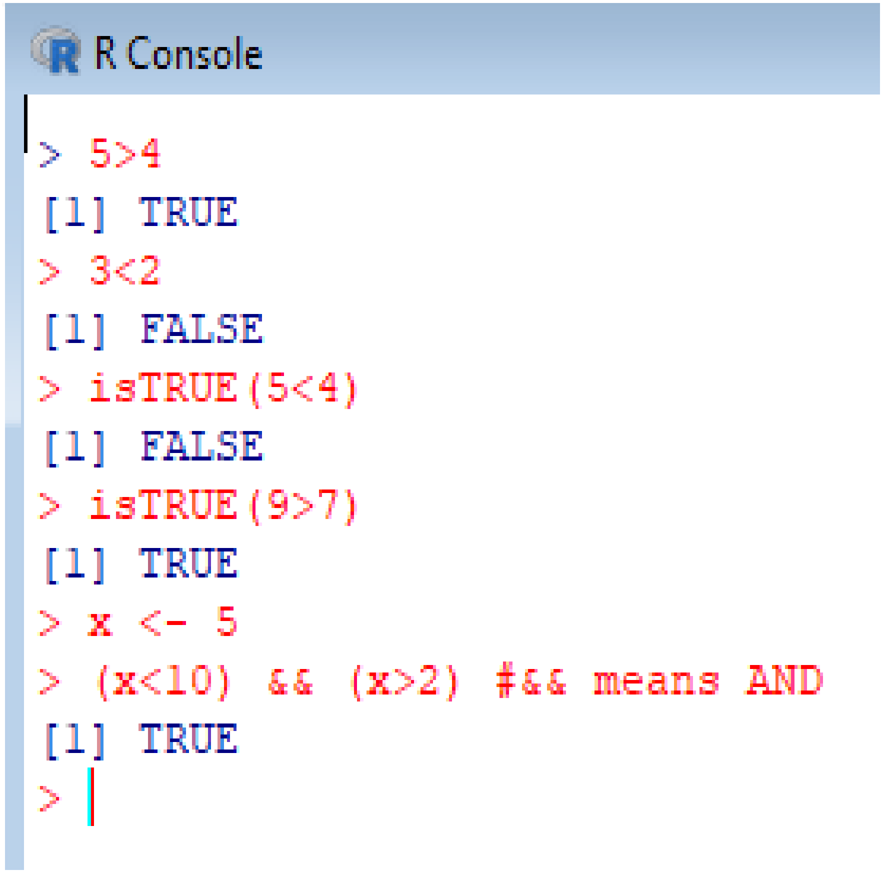
Logical Operators & Comparisons

The shorter form performs element-wise comparisons in almost the same way as arithmetic operators.

- The longer form 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 & Comparisons

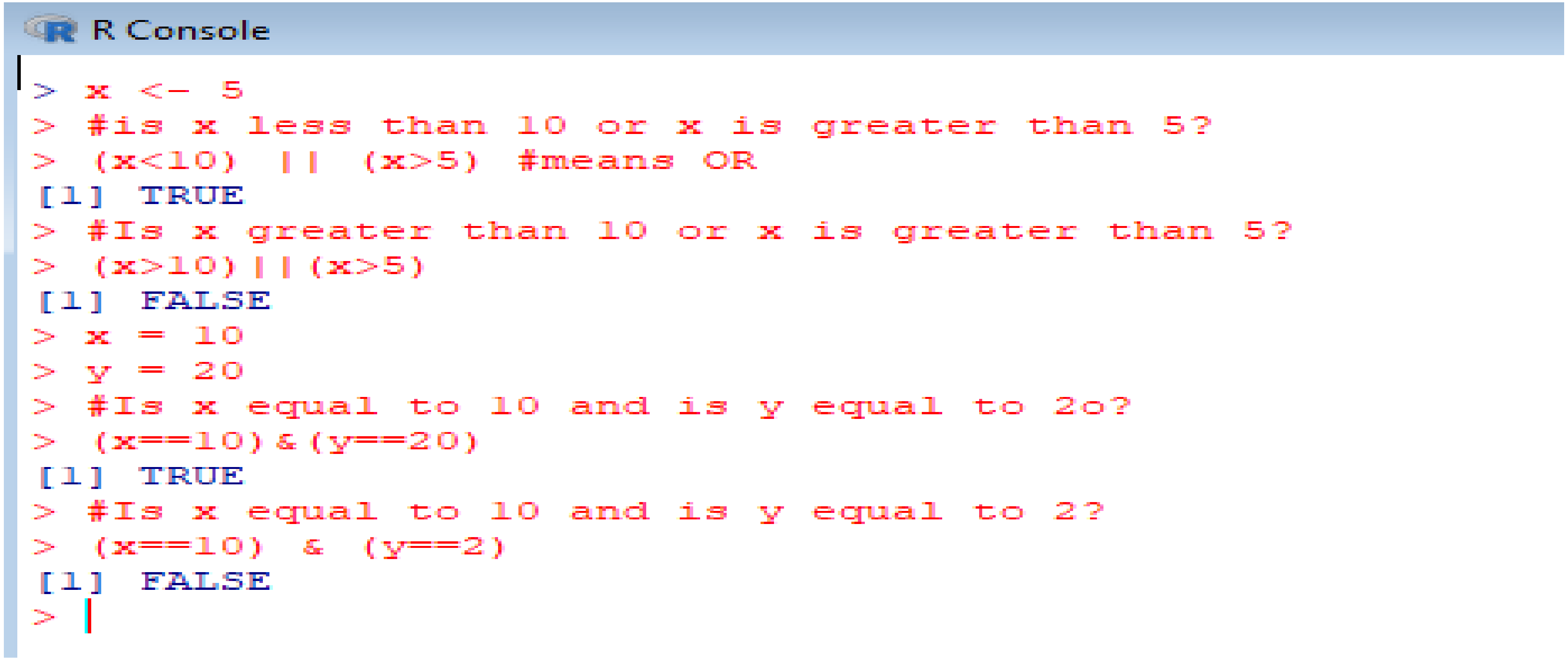
TRUE and FALSE are reserved words denoting logical constants



```
> 5>4
[1] TRUE
> 3<2
[1] FALSE
> isTRUE(5<4)
[1] FALSE
> isTRUE(9>7)
[1] TRUE
> x <- 5
> (x<10) && (x>2) #&& means AND
[1] TRUE
> 
```

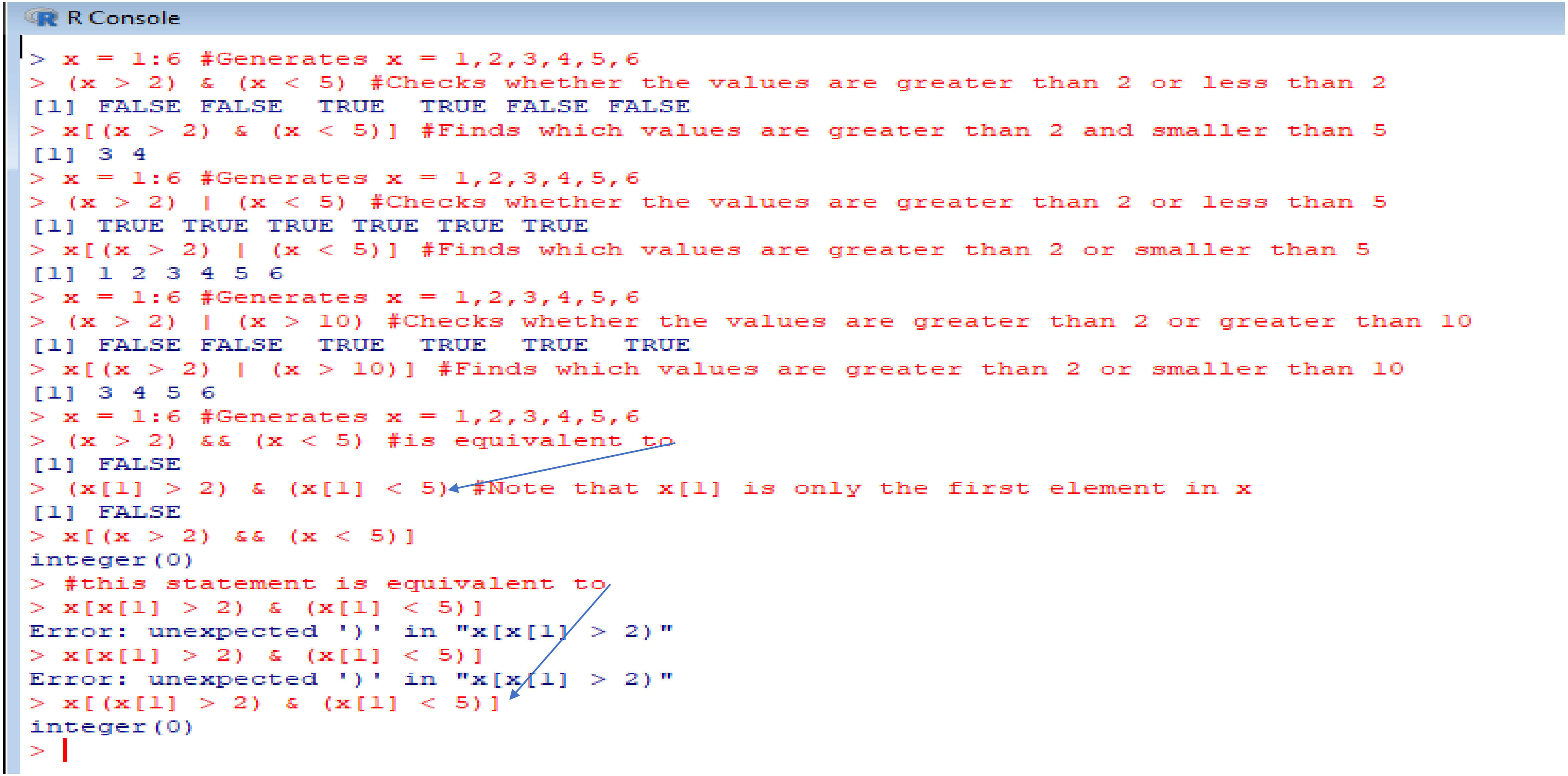
Operator	Executions
xor()	*Either…or (exclusive)*
isTRUE(x)	*Test if x is TRUE*
TRUE	*true*
FALSE	*false*

Logical Operators & Comparisons



```
> 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
> y = 20
> #Is x equal to 10 and is y equal to 2o?
> (x==10)&(y==20)
[1] TRUE
> #Is x equal to 10 and is y equal to 2?
> (x==10) & (y==2)
[1] FALSE
> 
```

Logical Operators & Comparisons



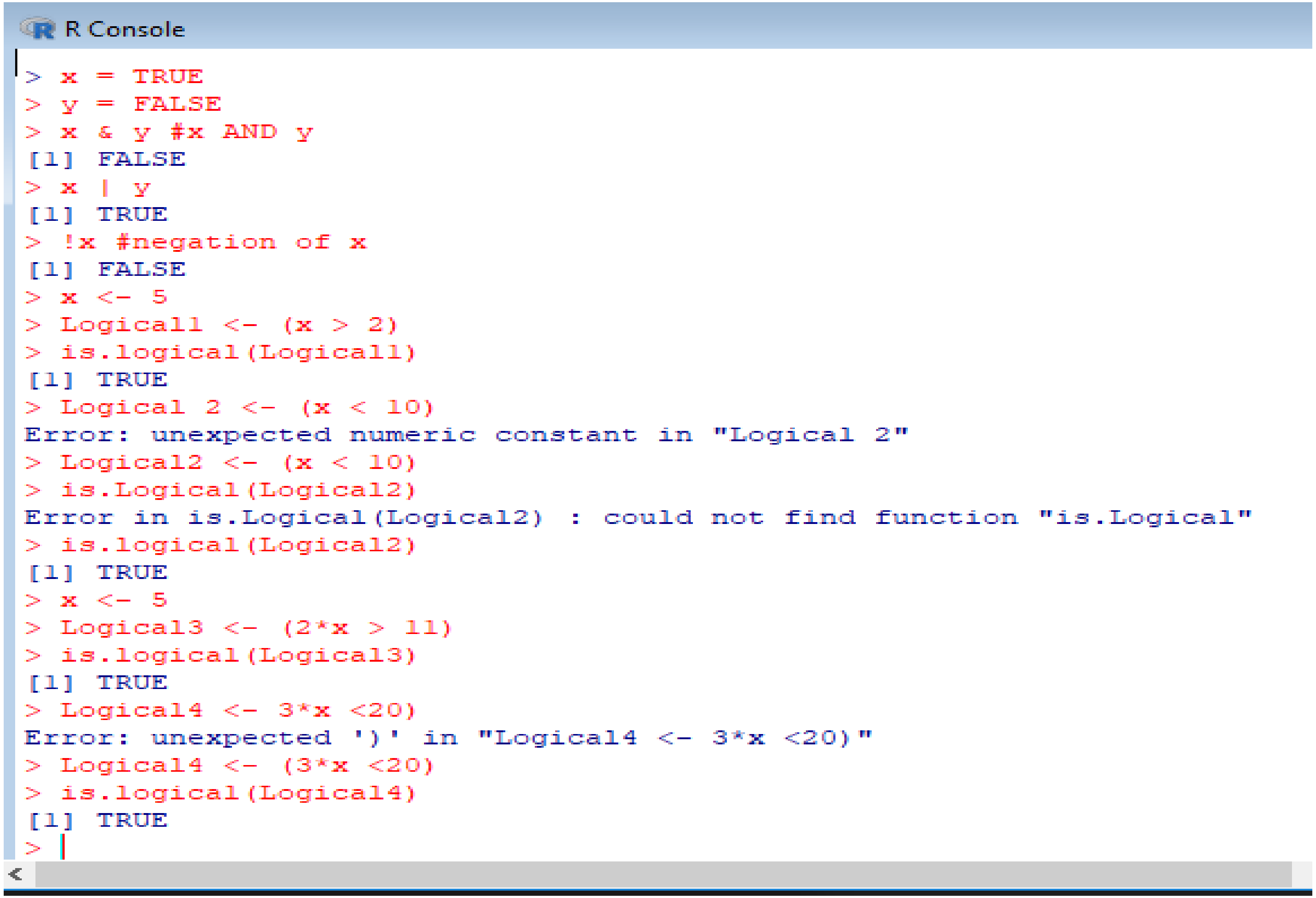
```
> x = 1:6 #Generates x = 1,2,3,4,5,6
> (x > 2) & (x < 5) #Checks whether the values are greater than 2 or less than 2
[1] FALSE FALSE  TRUE  TRUE FALSE FALSE
> x[(x > 2) & (x < 5)] #Finds which values are greater than 2 and smaller than 5
[1] 3 4
> x = 1:6 #Generates x = 1,2,3,4,5,6
> (x > 2) | (x < 5) #Checks whether the values are greater than 2 or less than 5
[1] TRUE TRUE TRUE TRUE TRUE TRUE
> x[(x > 2) | (x < 5)] #Finds which values are greater than 2 or smaller than 5
[1] 1 2 3 4 5 6
> x = 1:6 #Generates x = 1,2,3,4,5,6
> (x > 2) | (x > 10) #Checks whether the values are greater than 2 or greater than 10
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> x[(x > 2) | (x > 10)] #Finds which values are greater than 2 or smaller than 10
[1] 3 4 5 6
> x = 1:6 #Generates x = 1,2,3,4,5,6
> (x > 2) && (x < 5) #is equivalent to
[1] FALSE
> (x[1] > 2) & (x[1] < 5) #Note that x[1] is only the first element in x
[1] FALSE
> x[(x > 2) && (x < 5)]
integer(0)
> #this statement is equivalent to
> x[x[1] > 2) & (x[1] < 5)]
Error: unexpected ')' in "x[x[1] > 2)"
> x[x[1] > 2) & (x[1] < 5)]
Error: unexpected ')' in "x[x[1] > 2)"
> x[(x[1] > 2) & (x[1] < 5)]
integer(0)
> 
```

Truth Tables

Example of Standard logical operations

Statement 1 :: (x)	Statement 2 :: (y)	Outcome :: x and y	Outcome :: x or y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Truth Tables



```
> x = TRUE
> y = FALSE
> x & y #x AND y
[1] FALSE
> x | y
[1] TRUE
> !x #negation of x
[1] FALSE
> x <- 5
> Logical1 <- (x > 2)
> is.logical(Logical1)
[1] TRUE
> Logical 2 <- (x < 10)
Error: unexpected numeric constant in "Logical 2"
> Logical2 <- (x < 10)
> is.Logical(Logical2)
Error in is.Logical(Logical2) : could not find function "is.Logical"
> is.logical(Logical2)
[1] TRUE
> x <- 5
> Logical3 <- (2*x > 11)
> is.logical(Logical3)
[1] TRUE
> Logical4 <- 3*x <20)
Error: unexpected ')' in "Logical4 <- 3*x <20)"
> Logical4 <- (3*x <20)
> is.logical(Logical4)
[1] TRUE
>
```

Control Structure in R

Control structures in R

Allow you to control the flow of execution of a series of R expressions. Basically, control structures allow you to put some “logic” into your R code, rather than just always executing the same R code every time.

Control structures allow you to respond to inputs or to features of the data and execute different R expressions accordingly.

Control Structure in R

Commonly used control structures are:

if and ***else*****:** testing a condition and acting on it

for**:** execute a loop a fixed number of times

while**:** execute a loop while a condition is true

repeat**:** execute an infinite loop

break**:** break the execution of a loop

next**:** skip an iteration of loop

Return: functions to do some processing and return back the result

if and else

Syntax

if (condition) {executes commands if condition is TRUE}
if (condition) {executes commands if condition is TRUE}
else { executes commands if condition is FALSE }

Please note:

• The condition in this control statement may not be vector valued and if so, only the first element of the vector is used.

• The condition may be a complex expression where the logical operators "and" (&&) and "or" (||) can be used.

if and else

Example:

> x <- 5

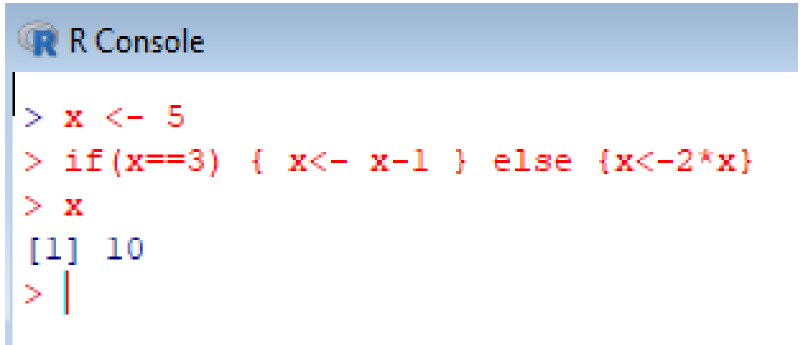
if (x==3) { x <- x-1 } else { x <- 2*x }

Interpretation:

- If x = 3, then execute x = x – 1.
- If x ≠ 3, then execute x = 2*x.

In this case, x = 5, so x ≠ 3. Thus x = 2*5

> x



```
> x <- 5
> if(x==3) { x<- x-1 } else {x<-2*x}
> x
[1] 10
> 
```

if and else

> x <- 3

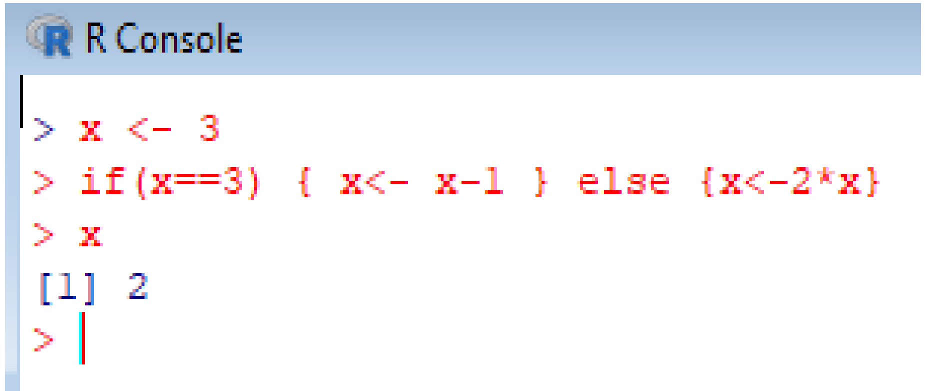
> if (x==3) { x <- x-1 } else { x <- 2*x }

Interpretation:

- If x = 3, then execute x = x – 1.
- If x ≠ 3, then execute x = 2*x.

In this case, x = 3, so x = 3 – 1

> x



```
> x <- 3
> if(x==3) { x<- x-1 } else {x<-2*x}
> x
[1] 2
> 
```