Matrices and Arrays

25-12-2024

In this we explains how to create and work with matrices and arrays in R.These are powerful data structures that are especially useful for handling data in statistical analysis.

Matrices

Matrices in R are collections of values (usually numbers) arranged in a rectangular grid of rows and columns. The elements of a matrix can be accessed by their row and column indices. You can create a matrix in R using the matrix() function.

1.You need to provide the data that will populate the matrix. This data is usually supplied as a vector. 2.You also need to specify the dimensions of the matrix (number of rows and columns) using the arguments nrow and ncol. 3.By default, R will fill the matrix column-wise, meaning the values in your data vector will be placed into the matrix column by column.

```
mymat <- matrix(1:12,nrow = 3,ncol = 4)
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

rbind and cbind of data in matrix

You can also create matrices by combining existing vectors using the rbind() and cbind() functions. 1.rbind() combines vectors as rows of the resulting matrix. 2.cbind() combines vectors as columns of the resulting matrix.

```
x <- 1:4
y <- 5:8

rbind(x,y)     # along row
```

```
##   [,1] [,2] [,3] [,4]
## x    1    2    3    4
## y    5    6    7    8
```

```
cbind(x,y)     # along column
```

```
##      x y
## [1,] 1 5
## [2,] 2 6
## [3,] 3 7
## [4,] 4 8
```

The dim(), nrow(), and ncol() functions can be used to get information about the dimensions of a matrix.

```
dim(mymat)     # dim of matrix
```

```
## [1] 3 4
```

```
nrow(mymat)   # no. or rows in matrix
```

```
## [1] 3
```

```
ncol(mymat)  # no. of column in a matrix
```

```
## [1] 4
```

Subsets, Extractions, and Replacements

1.You can access and modify elements within a matrix using subsetting.Subsetting in R uses square brackets [,] with the desired row and column indices separated by a comma. 2.To access a single element, specify its row and column index. For example, mymat would access the element in the 1st row and 3rd column of mymat.

```
mymat_1<-mymat[2,4]    # access single entry form matrix.
mymat_1
```

```
## [1] 11
```

3.To access an entire row, leave the column index blank. For example,mymat would access the second row of mymat.

```
mymat_2<-mymat[2,]    # for a entire second row.
mymat_2
```

```
## [1]  2  5  8 11
```

4.To access an entire column, leave the row index blank. For example, mymat[,4] would access the fourth column of mymat.

```
mymat_3<-mymat[,3]    # for a entire third column.
mymat_3
```

```
## [1] 7 8 9
```

To access a sub matrix, specify ranges of row and column indices. For example, mymat[1:2,2:3] would access the sub matrix containing the elements in rows 1 and 2 and columns 2 and 3 of mymat.

```
dim(mymat)
```

```
## [1] 3 4
```

```
mymat[2:3,3:4]        # for a subsetting of matrix.
```

```
##      [,1] [,2]
## [1,]    8   11
## [2,]    9   12
```

we can also modify the elements of the matrices.

```
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
mymat[2,4]<-34     # modify the entry in a matrix.
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   34
## [3,]    3    6    9   12
```

Diagonals and Identity matrices

The diag() function can be used to extract the diagonal elements of a matrix. It can also be used to create an identity matrix, a square matrix with ones on the diagonal and zeros elsewhere.To extract the diagonal, pass the matrix to diag(). For example, diag(mymat) would return a vector containing the diagonal elements of mymat.

```
diag(mymat)   # diagonal entries of the mymat.
```

```
## [1] 1 5 9
```

To create an identity matrix, pass the desired dimension to diag().For example, diag(3) would create a 3x3 identity matrix.

```
diag(4)       # for creating identity matrix.
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

Matrix Operations

1.Transpose (t()): The t() function transposes a matrix, interchanging its rows and columns.

```r
t(mymat)  # transpose of matrix.
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   34   12
```

2.Scalar Multiplication (*): You can multiply a matrix by a scalar (a single number) using the multiplication operator (*). This operation multiplies every element of the matrix by the scalar.

```r
2*mymat        # multiply a constant with a matrix.
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    8   14   20
## [2,]    4   10   16   68
## [3,]    6   12   18   24
```

3.Matrix Addition and Subtraction (+, -): Matrices of the same dimensions 4.can be added or subtracted using the + and - operators, respectively.

```r
mymat + mymat_3     # adding two matrices
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    8   11   14   17
## [2,]   10   13   16   42
## [3,]   12   15   18   21
```

```r
mymat - mymat_3     # subtraction two matrices
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -6   -3    0    3
## [2,]   -6   -3    0   26
## [3,]   -6   -3    0    3
```

5.Matrix Multiplication (%*%): You can multiply matrices using the %*% operator. The matrices must be conformable, meaning the number of columns in the first matrix must match the number of rows in the second matrix.

```r
x<-matrix(1:12, nrow = 4, ncol = 3)
dim(x)
```

```
## [1] 4 3
```

```r
x%*%mymat    # matrix multiplication
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   38   83  128  288
## [2,]   44   98  152  344
## [3,]   50  113  176  400
## [4,]   56  128  200  456
```

Matrix Inverse (solve()): For a square matrix, you can find its inverse (if it exists) using the solve() function. The inverse of a matrix A is denoted `A<sup> -1</sup>` and satisfies the property that A`%*% A<sup>-1</sup> = I`, where I is the identity matrix.

```r
z<-matrix(c(1,4,5,7,8,11,23,34,67), nrow = 3, ncol = 3)
solve(z)     # inverse of matrix
```

```
##              [,1]        [,2]       [,3]
## [1,] -0.375000000  0.50000000 -0.1250000
## [2,]  0.226851852  0.11111111 -0.1342593
## [3,] -0.009259259 -0.05555556  0.0462963
```

Multidimensional Arrays

Arrays are a generalization of matrices to more than two dimensions.You can think of them as "matrices with depth". You create arrays using the array() function, providing the data and the desired dimensions:

```r
myarray <- array(1:24, dim=c(2,3,4))
myarray
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

```r
#subsetting form array
myarray[2,2:3,3]
```

```
## [1] 16 18
```