Matrices

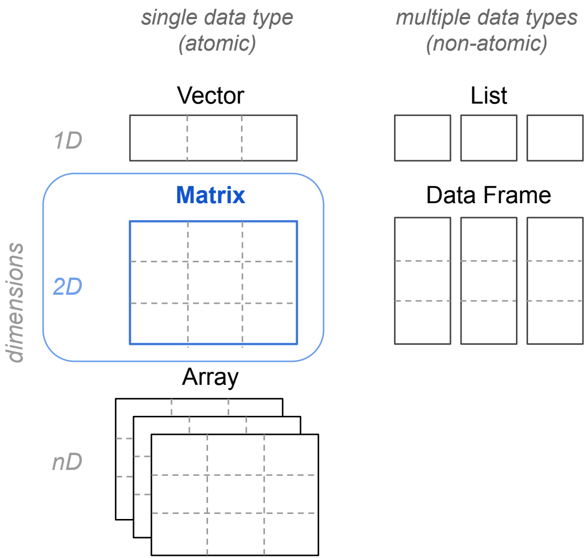
R Data Objects

Gaston Sanchez

STAT 33B, Fall 2025

About

- Matrices


single data type
(atomic)
multiple data types
(non-atomic)
Vector
List
1D
Matrix
Data Frame
2D
dimensions
Array
nD

From Vectors to Arrays

We can transform a vector in an **n-dimensional array** by giving it a dimensions attribute `dim`

```r
# positive: from 1 to 8
x <- 1:8

# adding 'dim' attribute
dim(x) <- c(2, 4)
x
```

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

From Vectors to Arrays

- a vector can be given a `dim` attribute
- a `dim` attribute is a numeric vector of length `n`
- R will reorganize the elements of the vector into `n` dimensions
- each dimension will have as many rows (or columns, etc.) as the `n`-th value of the `dim` vector

From Vectors to Arrays

```r
# dim attribute with 3 dimensions
dim(x) <- c(2, 2, 2)
x
```

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

From Vector to Matrix

A `dim` attribute of length 2 will convert a vector into a matrix

```
# vector to matrix
A <- 1:8
class(A)
```

```
## [1] "integer"
```

```
dim(A) <- c(2, 4)
class(A)
```

```
## [1] "matrix" "array"
```

When using `dim()`, R always fills up each matrix by columns.

From Vector to Matrix

To have more control about how a matrix is filled, (by rows or columns), we use the `matrix()` function:

```
# vector to matrix
A <- 1:8

matrix(A, nrow = 2, ncol = 4)
```

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

Matrices

- An R matrix provides a rectangular data object; i.e. to handle data in a two-dimensional array
- To create a matrix, give a vector to `matrix()` and specify number of rows and columns
- You can also assign row and column names to a matrix

Matrices

- R internally stores matrices as vectors.
- Which means that matrices are also atomic.
- Matrices in R are stored **column-major** (i.e. by columns).
- This is like Fortran, Matlab, and Julia, but not like C or Python (e.g. numpy)

Creating a Matrix

How do you create the following matrix?

```
##      [,1]       [,2]
## [1,] "Harry"    "Potter"
## [2,] "Ron"      "Weasley"
## [3,] "Hermione" "Granger"
```

Creating a Matrix

Here's one way to create the matrix of the previous slide

```r
# vector of names
hp <- c("Harry", "Ron", "Hermione",
        "Potter", "Weasley", "Granger")

# matrix filled up by columns
matrix(hp, nrow = 3)
```

```
##      [,1]       [,2]
## [1,] "Harry"    "Potter"
## [2,] "Ron"      "Weasley"
## [3,] "Hermione" "Granger"
```

Creating a Matrix

Here's another way to create the preceding matrix

```r
# vector of names
hp <- c("Harry", "Potter", "Ron", "Weasley",
        "Hermione", "Granger")

# matrix filled up by rows
matrix(hp, nrow = 3, byrow = TRUE)
```

```
##      [,1]       [,2]
## [1,] "Harry"    "Potter"
## [2,] "Ron"      "Weasley"
## [3,] "Hermione" "Granger"
```

Row and Column Names

R provides functions that let you set / get the names for either the rows or the columns (or both) of a matrix:

- `rownames()`
- `colnames()`
- `dimnames()`

Row and Column Names

```
set.seed(123)
A <- matrix(runif(12), nrow = 3, ncol = 4)
rownames(A) <- paste0("row", 1:nrow(A))
A
```

```
##           [,1]      [,2]      [,3]      [,4]
## row1 0.2875775 0.8830174 0.5281055 0.4566147
## row2 0.7883051 0.9404673 0.8924190 0.9568333
## row3 0.4089769 0.0455565 0.5514350 0.4533342
```

```
rownames(A)
```

```
## [1] "row1" "row2" "row3"
```

Row and Column Names

```r
set.seed(123)
B <- matrix(runif(12), nrow = 3, ncol = 4)
colnames(B) <- paste0("col", 1:ncol(B))
B
```

```
##           col1      col2      col3      col4
## [1,] 0.2875775 0.8830174 0.5281055 0.4566147
## [2,] 0.7883051 0.9404673 0.8924190 0.9568333
## [3,] 0.4089769 0.0455565 0.5514350 0.4533342
```

```r
colnames(B)
```

```
## [1] "col1" "col2" "col3" "col4"
```

Row and Column Names

```r
set.seed(123)
C <- matrix(runif(12), nrow = 3, ncol = 4)
rownames(C) <- paste0("row", 1:nrow(C))
colnames(C) <- paste0("col", 1:ncol(C))
C
```

```
##           col1      col2      col3      col4
## row1 0.2875775 0.8830174 0.5281055 0.4566147
## row2 0.7883051 0.9404673 0.8924190 0.9568333
## row3 0.4089769 0.0455565 0.5514350 0.4533342
```

```r
dimnames(C)
```

```
## [[1]]
## [1] "row1" "row2" "row3"
##
## [[2]]
## [1] "col1" "col2" "col3" "col4"
```

Row and Column Names

```r
set.seed(123)
D <- matrix(runif(12), nrow = 3, ncol = 4)
dimnames(D) <- list(
  paste0("row", 1:nrow(D)),
  paste0("col", 1:ncol(D))
)
D
```

```
##           col1      col2      col3      col4
## row1 0.2875775 0.8830174 0.5281055 0.4566147
## row2 0.7883051 0.9404673 0.8924190 0.9568333
## row3 0.4089769 0.0455565 0.5514350 0.4533342
```

Recycling

Recycling rules also apply to matrices

```
x <- letters[1:4]
X <- matrix(x, nrow = 4, ncol = 3)
X
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "a"
## [2,] "b"  "b"  "b"
## [3,] "c"  "c"  "c"
## [4,] "d"  "d"  "d"
```

Recycling

```r
# "empty" matrices
mat_chr <- matrix("", nrow = 4, ncol = 3)

mat_num <- matrix(0, nrow = 4, ncol = 3)

mat_lgl <- matrix(NA, nrow = 4, ncol = 3)
```

Matrix Manipulation

Subsetting: Bracket Notation

opening bracket

closing bracket

`mat[i,j]`

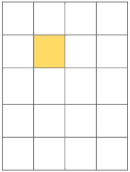
matrix object

row index vector

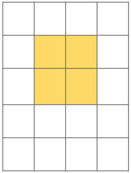
column index vector

Subsetting Cells

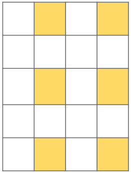
`mat[2,2]`



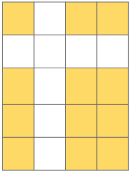
`mat[2:3,2:3]`



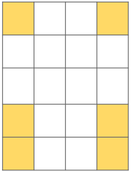
`mat[c(1,3,5), c(2,4)]`



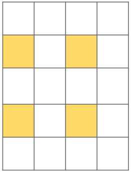
`mat[-2,-2]`



`mat[-(2:3), -(2:3)]`

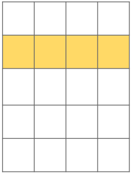


`mat[-c(1,3,5), -c(2,4)]`

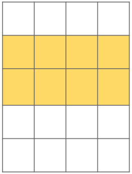


Subsetting Rows

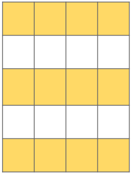
`mat[2, ]`



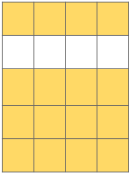
`mat[2:3, ]`



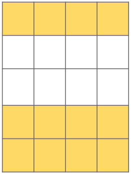
`mat[c(1,3,5), ]`



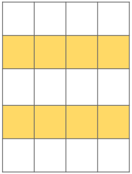
`mat[-2, ]`



`mat[-(2:3), ]`

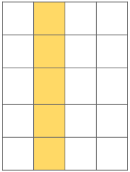


`mat[-c(1,3,5), ]`

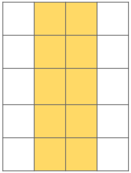


Subsetting Columns

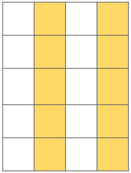
`mat[ ,2]`



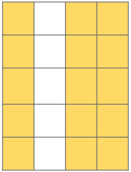
`mat[ ,2:3]`



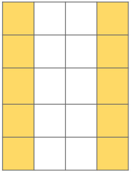
`mat[ ,c(2,4)]`



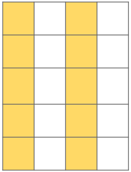
`mat[ ,-2]`

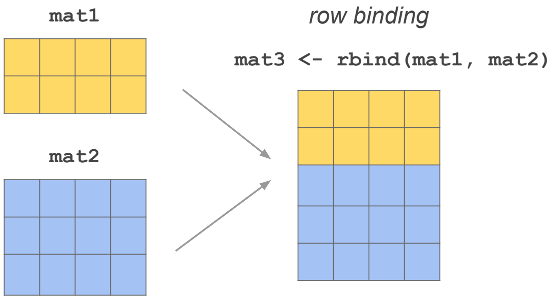


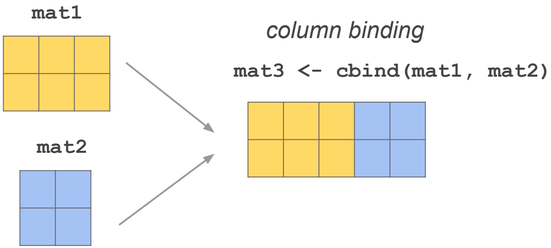
`mat[ ,-(2:3)]`



`mat[ ,-c(2,4)]`




mat1
row binding
mat3 <- rbind(mat1, mat2)
mat2


mat1
column binding
mat3 <- cbind(mat1, mat2)
mat2

`apply()` function

```r
X = matrix(rep(1:3, each = 5), nrow = 5, 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
## [3,]    1    2    3
## [4,]    1    2    3
## [5,]    1    2    3
```

```r
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
## [3,]    1    2    3
## [4,]    1    2    3
## [5,]    1    2    3
```

```r
# sum of elements in each row (MARGIN = 1)
apply(X, MARGIN = 1, FUN = sum)
```

```
## [1] 6 6 6 6 6
```

```r
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
## [3,]    1    2    3
## [4,]    1    2    3
## [5,]    1    2    3
```

```r
# sum of elements in each column (MARGIN = 2)
apply(X, MARGIN = 2, FUN = sum)
```

```
## [1]  5 10 15
```

```r
Y = matrix(1:15, nrow = 5, 3)
Y
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```r
# mean of elements in each row (MARGIN = 1)
apply(Y, MARGIN = 1, FUN = mean)
```

```
## [1]  6  7  8  9 10
```

```r
Y
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```r
# mean of elements in each column (MARGIN = 2)
apply(Y, MARGIN = 2, FUN = mean)
```

```
## [1]  3  8 13
```

```r
set.seed(123)
Z = matrix(runif(15), nrow = 5, 3)
Z
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.2875775 0.0455565 0.9568333
## [2,] 0.7883051 0.5281055 0.4533342
## [3,] 0.4089769 0.8924190 0.6775706
## [4,] 0.8830174 0.5514350 0.5726334
## [5,] 0.9404673 0.4566147 0.1029247
```

```r
# median of elements in each row (MARGIN = 1)
apply(Z, MARGIN = 1, FUN = median)
```

```
## [1] 0.2875775 0.5281055 0.6775706 0.5726334 0.4566147
```

```r
Z
```

```
##              [,1]      [,2]      [,3]
## [1,] 0.2875775 0.0455565 0.9568333
## [2,] 0.7883051 0.5281055 0.4533342
## [3,] 0.4089769 0.8924190 0.6775706
## [4,] 0.8830174 0.5514350 0.5726334
## [5,] 0.9404673 0.4566147 0.1029247
```

```r
# median of elements in each column (MARGIN = 2)
apply(Z, MARGIN = 2, FUN = median)
```

```
## [1] 0.7883051 0.5281055 0.5726334
```

```r
# descriptive stats of elements in each row (MARGIN = 1)
apply(Z, MARGIN = 1, FUN = summary)
```

```
##              [,1]      [,2]      [,3]      [,4]      [,5]
## Min.    0.0455565 0.4533342 0.4089769 0.5514350 0.1029247
## 1st Qu. 0.1665670 0.4907198 0.5432738 0.5620342 0.2797697
## Median  0.2875775 0.5281055 0.6775706 0.5726334 0.4566147
## Mean    0.4299891 0.5899149 0.6596555 0.6690286 0.5000022
## 3rd Qu. 0.6222054 0.6582053 0.7849948 0.7278254 0.6985410
## Max.    0.9568333 0.7883051 0.8924190 0.8830174 0.9404673
```

```r
# descriptive stats of elements in each column (MARGIN = 2)
apply(Z, MARGIN = 2, FUN = summary)
```

```
##              [,1]      [,2]      [,3]
## Min.    0.2875775 0.0455565 0.1029247
## 1st Qu. 0.4089769 0.4566147 0.4533342
## Median  0.7883051 0.5281055 0.5726334
## Mean    0.6616689 0.4948262 0.5526592
## 3rd Qu. 0.8830174 0.5514350 0.6775706
## Max.    0.9404673 0.8924190 0.9568333
```

Matrix Algebra

Operators

Operator	Description	Example
`+`	addition	`A + B`
`-`	subtraction	`A - B`
`*`	elementwise product	`A * B`
`%*%`	matrix product	`A %*% B`
`t()`	transpose	`t(A)`
`det()`	determinant	`det(A)`
`diag()`	extract diagonal	`diag(A)`
`solve()`	inverse	`solve(A)`

Note: make sure the dimensions of matrices are conformable when using an operator or some calculation on them.

Other Functions

Function	Description
`upper.tri()`	upper triangular part of a matrix
`lower.tri()`	upper triangular part of a matrix
`eigen()`	eigenvalue decomp.
`svd()`	singular value decomp.
`lu()`	Triangular decomposition
`qr()`	QR decomposition
`chol()`	Cholesky decomposition