3. More Data structures in R

Principles of Data Science with R

Dr. Uma Ravat

PSTAT 10

Summary

- Data types (character, double, integer, logical)
- Data structures
  - Scalars
  - Vectors
  - more next time

Post-Lecture To DO

1. Review the lecture again
2. Write down a summary of today's lecture. Include all functions we went over and a short description of what each function does.

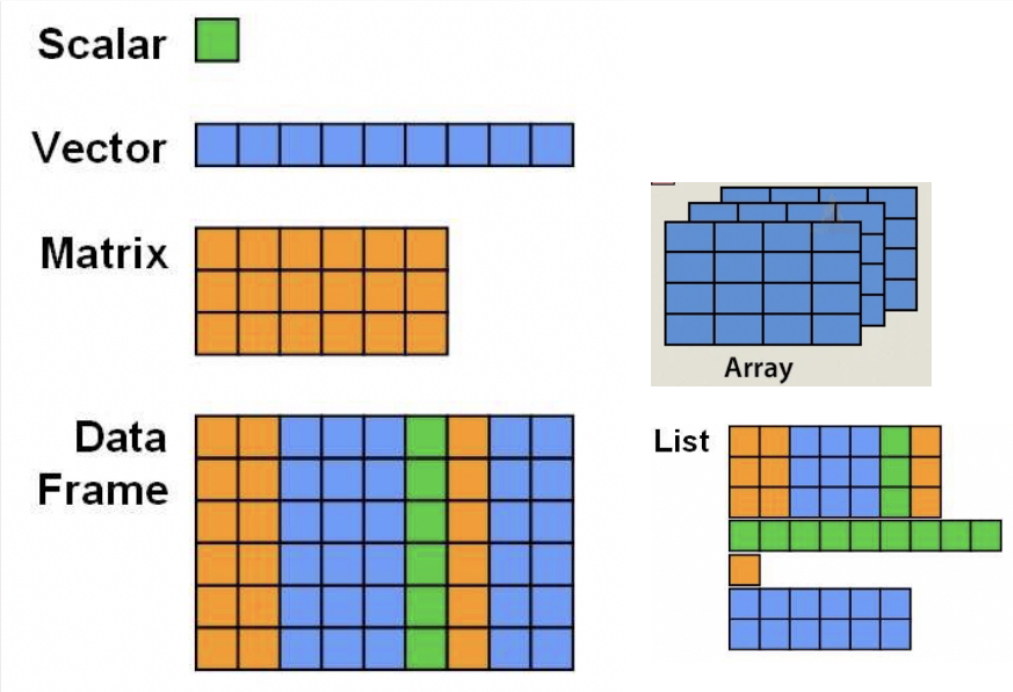
You will be asked to do this to your homework.

Next we will see. . .

- More data structures
  1. matrix
  2. array
  3. factor
  4. logical operators, strings

Matrices, Arrays, and Lists

- Data with dimensionality
- Often times, a vector (1-D) is not enough.



Matrix : two dimensional vector

Matrix : two dimensional vector

- All elements must be of the same data type
- ordered as [row, column]
- `?matrix`: syntax and arguments
- **Syntax :** matrix(data, nrow, ncol, byrow, dimnames)

Description

`matrix` creates a matrix from the given set of values.

`as.matrix` attempts to turn its argument into a matrix.

`is.matrix` tests if its argument is a (strict) matrix.

Usage

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)

as.matrix(x, ...)
## S3 method for class 'data.frame'
as.matrix(x, rownames.force = NA, ...)

is.matrix(x)
```

Arguments

`data`	an optional data vector (including a list or `expression` vector). Non-atomic classed `R` objects are coerced by `as.vector` and all attributes discarded.
`nrow`	the desired number of rows.
`ncol`	the desired number of columns.
`byrow`	logical. If `FALSE` (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.
`dimnames`	A `dimnames` attribute for the matrix: `NULL` or a `list` of length 2 giving the row and column names respectively. An empty list is treated as `NULL`, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.
`x`	an `R` object.
`...`	additional arguments to be passed to or from methods.

Create a summary of what we did

1. Create matrix
   - Get matrix dimensions
   - Assigning row and column names
   - Get row and column names
   - Creating a matrix by binding to rows or columns
2. Subseting
   - Using [ ] : provide an index to each dimension.
   - order in a matrix is given as [row , col]
   - **Omitting** an index returns all elements in that dimension
   - Subsetting by using row/column names

   Add anything else that we did

```
(x <- matrix(1:9, nrow=3, ncol=3))
```

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

Always print your object in exercises you submit. If you create an object but don't print it, the grader can't verify you successfully did the required task.

What is the result of `x[1:2, -3]`? *Express your answer in words*

Recycling

Create a 4×5 matrix `mx` of integers 1 through 17 (inclusive). Print the matrix.

```
(mx <- matrix(1:17, 4, 5)) # recycling
```

```
## Warning in matrix(1:17, 4, 5): data length [17] is not a
## multiple of the number of rows [4]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14    1
## [3,]    3    7   11   15    2
## [4,]    4    8   12   16    3
```

YT02 :

1. Subset `mx` to create the following matrix:

```
result
```

```
##      [,1] [,2] [,3]
## [1,]    1    5   17
## [2,]    4    8    3
```

2. Set all entries of `mx` greater than 10 to zero.

Arrays : Higher ($>$2) dimensional vectors

Arrays : Higher (>2) dimensional vectors

- **Arrays** are matrices in layers.
  - ordered as [row, column, layer/level]

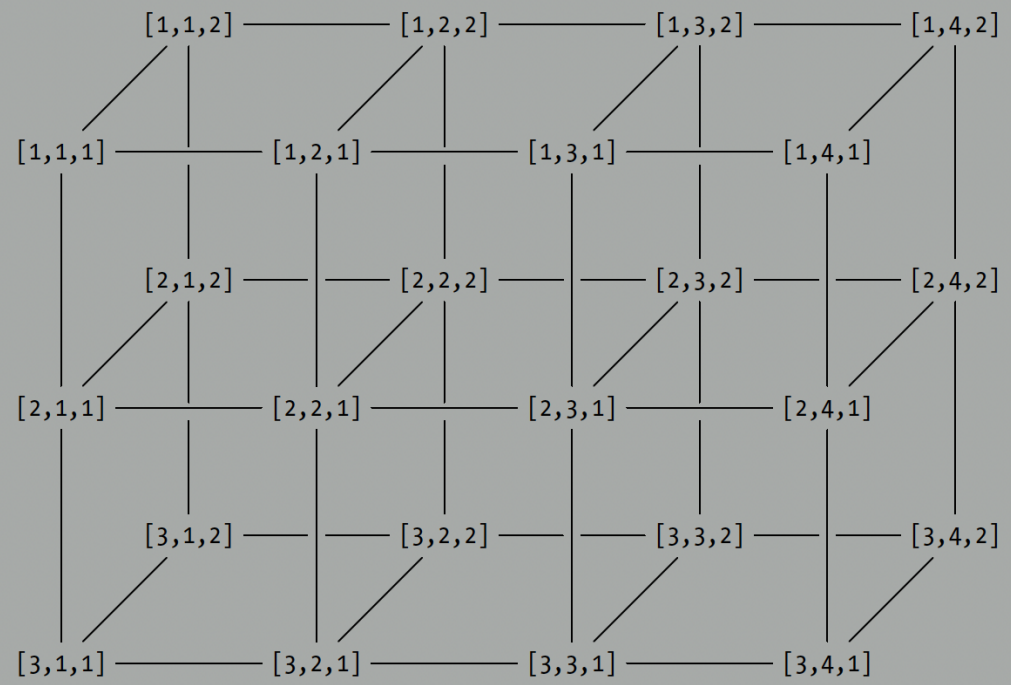


Figure 3-3: A conceptual diagram of a 3 × 4 × 2 array. The index of each element is given at the corresponding position. These indexes are provided in the strict order of `[row,column,layer]`

Syntax: Array creation

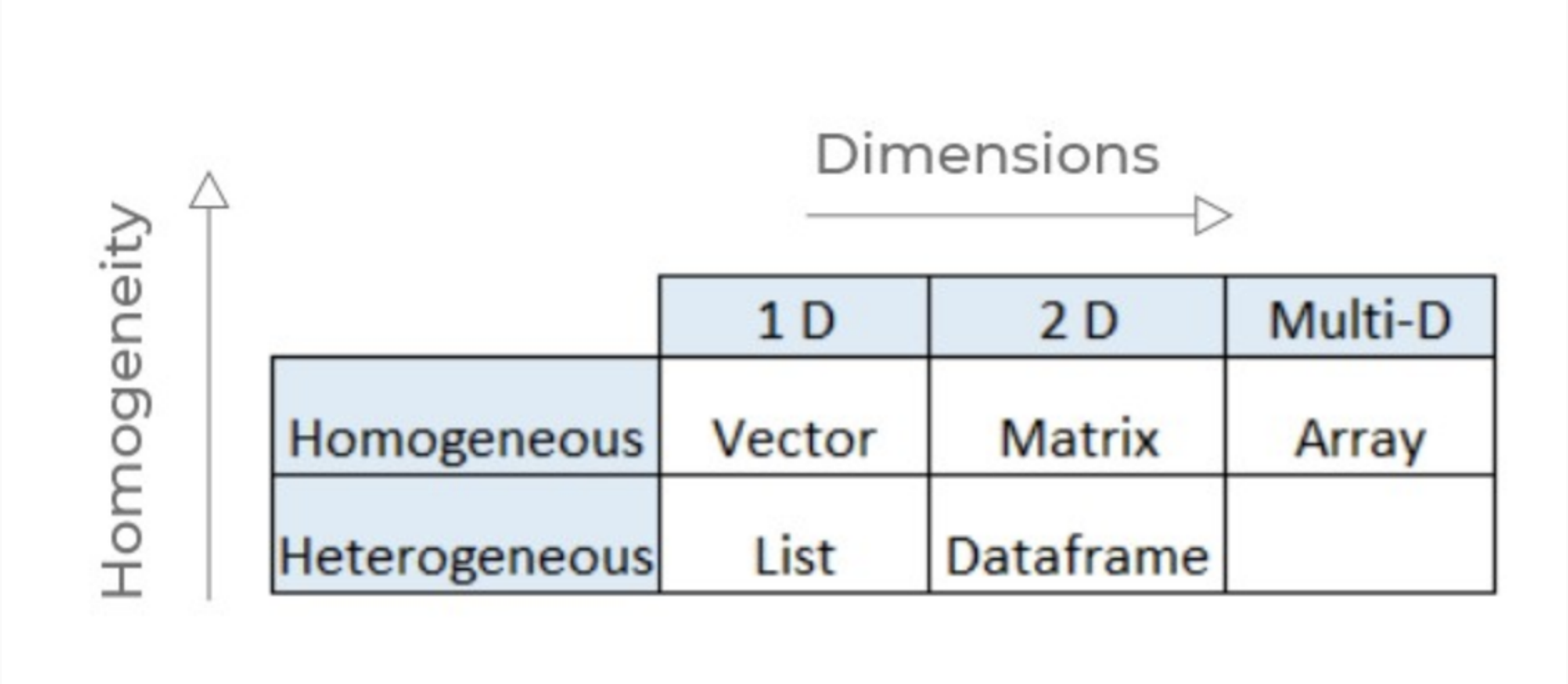
array(data, dim, dimnames)

```
A = array(data = 1:18, dim = c(3, 3, 2))
```

What we did

1. Create array
    - Get array dimensions
    - Assigning row, column, layer names
    - Get row, column, layer names 4 Subsetting
    - Using [ ]
    - Subsetting by using `dimnames`
2. `apply` function
   Add anything else that we did

Data structures in R



Factor

Factor

Factor is a special vector in R used to store data that belongs to fixed and known set of possible values(categories)

Example:

- Ordinal
  - Grades: A, B, C
  - Month of the year : Jan, Feb, Mar, ... Dec
- Nominal
  - Sex: Male, Female
  - Color of Hair : Brown, Black, Blonde, Red, Other

A factor consists of

1. a set of **values**
2. a set of **valid levels** (the different categories)
    - the levels can be **ordered** (ordinal) or **unordered** (nominal)

Ordered factor: Two options in `R`

1. `ordered()` function for ordinal data.
2. `factor()` function with the argument `ordered = TRUE`

Aside: Working with Logical Operators, strings

Aside: Working with Logical Operators, strings

Recall: Logical datatype: `TRUE`, `FALSE`

Single comparison operator	Interpretation
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
5+6 == 4
```

```
## [1] FALSE
```

Comparing Two Logical Values

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

```r
x <- c(TRUE, FALSE, TRUE)
y <- c(TRUE, TRUE, TRUE)

# Using &
x & y  # Element-wise comparison
```

```
## [1]  TRUE FALSE  TRUE
```

```r
# Using &&,can only do a single comparision
x[1] && y[1]
```

```
## [1] TRUE
```

```r
# try x && y !

# use case in stats
y <- 1 + (x <- stats::rpois(50, lambda = 1.5) / 4 - 1)
x[(x > 0) & (x < 1)]    # all x values between 0 and 1
```

```
## [1] 0.25 0.25
```

```r
if (any(x == 0) || any(y == 0)) "zero encountered"
```

which()

`which()` function returns index positions that satisfy a logical condition

For example, looking at the `state.name` dataset from `datasets` package, we see that "Alaska" is the second element.

```
which(state.name == "Alaska")
```

```
## [1] 2
```

Strings

```
my_string <- c("This is a string")
my_string
```

```
## [1] "This is a string"
```

```
nchar(my_string)
```

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

```
substr(my_string, start = 3, stop = 9)
```

```
## [1] "is is a"
```

```r
sub(pattern="is",replacement="was",x=my_string)
```

```
## [1] "Thwas is a string"
```

```r
gsub(pattern="is",replacement="was",x=my_string)
```

```
## [1] "Thwas was a string"
```

Predefined Constants in R

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
month.abb
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

questions you should be able to answer

- What are the different data types and data structures in `R`?
- What are differences in each of these
- How do I create these, access, update data within the various data structures?

Post-Lecture To DO

1. Review the lecture again
2. Write down a summary of today's lecture. Include all functions we went over and a short description of what each function does.

You will be asked to do this to your homework.

Summary:

More data structures

- matrices and arrays. (Textbook Ch3)
- Factors (Textbook Chapter 4)
- Logical datatye and operator (Textbook Chapter 4)

Maintain a glossary of functions used.

Next we will see. . .

- Even more Data structures
  - list
  - data frame
  - Working with strings

Learning Programming is HARD!

E. Kale Edmiston PhD
@EKaleEdmiston

Follow

A friend/colleague who is an excellent programmer offhandedly told me the other day that coding is 90% googling error messages & 10% writing code. Until this point, I thought that all the time I spent googling error messages meant I was bad at coding. What a perspective change!

8:12 AM - 4 Jan 2019

151 Retweets **1,069** Likes

27 151 1.1K