R programming

Lesson 3: Data Type & Data Structure

Xiaoqin Yang

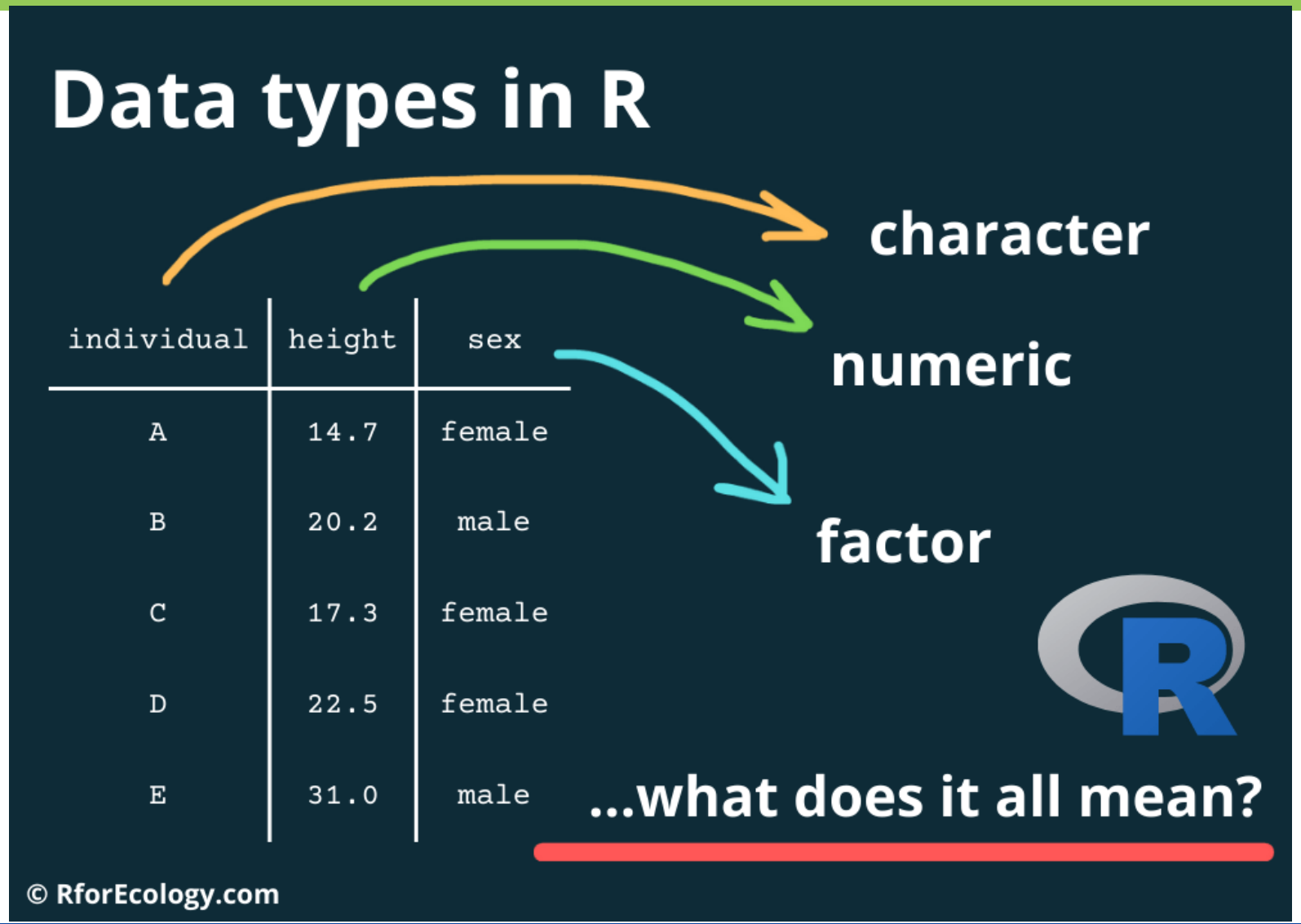
yangxiaoqin@suda.edu.cn

《R语言》课程成绩评分标准

- 总分100分
  - ✓ 平时成绩（过程化考核）：30%
  - ✓ 实验（含代码书写，伪代码注释，可视化效果）：30%
  - ✓ 期末考试：40%
- 平时成绩不含考勤
- 实验、过程化考核、考试请独立完成
- 过程化考核以及考试按照苏大纪律要求，违者必究
- 实验报告（包含代码）出现雷同，交由学工办以及学校教务部门处理
- 实验报告逾期按0分计算
- 不得使用ChatGPT等人工智能工具，一经发现按作弊论处
- 如无疫情等不可抗力之事由，考试形式为4道问答题及编程操作题
- 教师有考试形式及内容之决定权

- 善意提醒：
  - ✓ 违纪将交由学工及教务部门处理
  - ✓ 本学期过程化考核以及实验内容已全部更新……

What we should know?



https://www.rforecology.com/post/data-types-in-r/

Part 1 R Basic Data Types

Data Types

- In programming, data type is an important concept.
- Variables can store data of different types.
- There are many different data types with different purposes.

1 R Basic Data Types

I. character (a.k.a. string) - (e.g. "S", "R is good", "FALSE", "11.5")
II. numeric - (e.g. 11, 66.6, 919)
III. integer - (e.g. 1L, 66L, where the letter "L" declares it as an integer)
IV. complex - (e.g. 6 + 9i, where "i" is the imaginary part)
V. logical (a.k.a. boolean) - (TRUE or FALSE)

typeof(), class() & mode() functions in R

- typeof() determines the (R internal) type or storage mode of any object. Current values are the vector types "logical", "integer", "double", "complex", "character", "raw" and "list", "NULL", "closure" (function), "special" and "builtin" (basic functions and operators), "environment", "S4" (some S4 objects)

- class(): Abstract Type. In R, every object also has a class, which defines its abstract type. The terminology is borrowed from object-oriented programming. A single number could represent many different things: a distance, a point in time, or a weight, for example. All those objects have a mode of "numeric" because they are stored as a number, but they could have different classes to indicate their interpretation.

- mode(): Physical Type. In R, every object has a mode, which indicates how it is stored in memory: as a number, as a character string, as a list of pointers to other objects, as a function, and so forth.

Comparative table for data types & storage modes

- mode(x <- 5) # the storage mode of a number is numeric
- typeof(x) # by default numerics are double-precision floating-point numbers
- mode(y <- "test") # character strings are stored as character strings
- typeof(y) # ... and are character strings
- foo <- function(x) {x^2}
- mode(foo) # functions are stored as functions
- typeof(foo) # ... and declared as encapsulated chunk of code
- typeof(list) # but there are also functions which are only references to internal procedures (mostly written in C)
- mode(list) # ... which are nontheless stored as functions

Comparative table for data types & storage modes

```
##                      typeof(.)    mode(.)      class(.)
## NULL                 "NULL"       "NULL"       "NULL"
## 1                    "double"     "numeric"    "numeric"
## 1:1                  "integer"    "numeric"    "integer"
## 1i                   "complex"    "complex"    "complex"
## list(1)              "list"       "list"       "list"
## data.frame(x = 1) "list"          "list"       "data.frame"
## foo                  "closure"    "function"   "function"
## c                    "builtin"    "function"   "function"
## list                 "builtin"    "function"   "function"
## lm                   "closure"    "function"   "function"
## y ~ x                "language"   "call"       "formula"
## expression((1))   "expression" "expression" "expression"
## 1 < 3                "logical"    "logical"    "logical"
```

https://ds-pl-r-book.netlify.app/modes-types-classes-of-objects.html

1.1 character

- It stores character values or strings, which contains alphabets, numbers, and symbols.

```
> # A single string
> x <- 'this is the data’
> # A number
> y <- '123’
> # Create a vector of characters
> z <- c("These", "are", "characters")
```

```
> # A signle string
> x <- 'this is the data'
> # A number
> y <- '123'
> # Create a vector of characters
> z <- c("These", "are", "characters")
>
> typeof(x)
[1] "character"
> mode(y)
[1] "character"
> class(z)
[1] "character"
>
```

is.character() function

- is.character() Function is used to check if the object is of the form of a string/character or not.

```
> # R Program to illustrate
> # the use of is.character function
>
> # Creating a vector of mixed datatype
> x1 <- c("Hello", "Soochow University", 520)
>
> # Creating a vector of Numeric datatype
> x2 <- c(10, 20, 30)
>
> # Calling is.character() function
> is.character(x1)
[1] TRUE
> is.character(x2)
[1] FALSE
```

```
> # Create the vectors with different length
> vector1 <- c(1, 2, 3)
> vector2 <- c("SUDA", "Soochow", "Suzhou")
>
> # taking this vector as input
> result <- array(c(vector1, vector2), dim = c(3, 3, 2))
>
> # Calling is.character() function
> is.character(result)
[1] TRUE
> result
, , 1

     [,1] [,2]      [,3]
[1,] "1"  "SUDA"    "1"
[2,] "2"  "Soochow" "2"
[3,] "3"  "Suzhou"  "3"

, , 2

     [,1]      [,2] [,3]
[1,] "SUDA"    "1"  "SUDA"
[2,] "Soochow" "2"  "Soochow"
[3,] "Suzhou"  "3"  "Suzhou"

> mode(result)
[1] "character"
> class(result)
[1] "array"
```

```
> # R Program to illustrate
> # the use of is.character function
>
> # Create the vectors with different length
> vector1 <- c(1, 2, 3)
> vector2 <- c("SUDA", "Soochow", "Suzhou")
>
> # taking this vector as input
> result <- data.frame(vector1, vector2)
>
> # Calling is.character() function
> is.character(result)
[1] FALSE
> result
  vector1 vector2
1       1    SUDA
2       2 Soochow
3       3  Suzhou
> mode(result)
[1] "list"
> class(result)
[1] "data.frame"
>
```

```
> # Create the vectors with different length
> vector1 <- c(1, 2, 3)
> vector2 <- c("SUDA", "Soochow", "Suzhou")
>
> # taking these vectors as input
> result <- data.frame(vector1, vector2)
>
> # Calling is.character() function
> is.character(result)
[1] FALSE
>
```

1.2 numeric

- Numeric data consists of decimal values.
- However, the value doesn’t have to be decimal for the variable to be numeric.

- # Now assign a value of 11.6 to the variable num
- > num <- 11.6
- # Show the value of this variable num
- > num

```
> # Now assign a value of 11.6 to the variable num
> num <- 11.6
> # Show the value of this variable num
> num
[1] 11.6
>
```

Still numeric?

- # Now assign a value of "11.6" to the variable num1
- > num1 <- "11.6"
- # Show the value of this variable num1
- > num1
- > class(num1)
- > mode(num1)

```
> # Now assign a value of "11.6" to the variable num1
> num1 <- "11.6"
> # Show the value of this variable num1
> num1
[1] "11.6"
> class(num1)
[1] "character"
> mode(num1)
[1] "character"
```

numeric, but not decimal

- # assigns a value of 66 to the variable num2
- > num2 <- 66
- > num2
- # shows the class or type of the variable num2
- > class(num2)
- > mode(num2)

```
> #assigns a value of 66 to the variable num2
> num2 <- 66
> num2
[1] 66
> #shows the class or type of the variable num2
> class(num2)
[1] "numeric"
> mode(num2)
[1] "numeric"
```

is.numeric() function

- Check if the data is numeric by using the function is.numeric().

```
> # Is this number numeric? Yes!
> is.numeric(66)
[1] TRUE
>
> # Is this word numeric? No!
> is.numeric('Hello')
[1] FALSE
>
> # Create a numeric vector
> x <- c(3, 5, 6, 10.7)
>
> # Is our vector numeric? Yes!
> is.numeric(x)
[1] TRUE
```

Date is also numeric, but special

```
> x <- as.Date("2023-10-11")
>
> typeof(x)
[1] "double"
>
> class(x)
[1] "Date"
>
> mode(x)
[1] "numeric"
>
```

3. integer

- Do math with integers, which represent numbers without decimal places.
- Use the capital ‘L’ notation as a suffix to denote that a particular value is of the integer R data type.

Difference between integer and numeric

```
> # Create an integer vector
> x <- c(1L, 2L, 5L, 3L, 10L)
>
> # View vector
> x
[1]  1  2  5  3 10
>
> class(x)
[1] "integer"
>
> mode(x)
[1] "numeric"
>
> typeof(x)
[1] "integer"
>
> is.integer(x)
[1] TRUE
>
```

```
> # Create a numeric vector, without "L"
> x <- c(1, 2, 5, 3, 10)
>
> # View vector
> x
[1]  1  2  5  3 10
>
> class(x)
[1] "numeric"
>
> mode(x)
[1] "numeric"
>
> typeof(x)
[1] "double"
>
> is.integer(x)
[1] FALSE
>
```

4. complex

- Complex numbers aren't used much in R for data analysis, though they exist.
- These are just numbers with real and imaginary components (containing the number i, or the square root of -1).

```
> x <- 3+1i
> mode(x)
[1] "complex"
> typeof(x)
[1] "complex"
> class(x)
[1] "complex"
> is.complex(x)
[1] TRUE
>
```

5. logical

- The logical data type is also known as boolean data type. It have only two values: TRUE and FALSE.
- A logical value is often created via a comparison between variables.

```
> today <- 55
> today > 54
[1] TRUE
> today < 54
[1] FALSE
> today == 55
[1] TRUE
> is.numeric(today)
[1] TRUE
```

```
> yesterday <- "55"
> yesterday > 54
[1] TRUE
> yesterday < 54
[1] FALSE
> yesterday == 55
[1] TRUE
> is.numeric(yesterday)
[1] FALSE
```

```
> class(TRUE)
[1] "logical"
> mode(TRUE)
[1] "logical"
> typeof(FALSE)
[1] "logical"
>
>
```

is.logical() functions

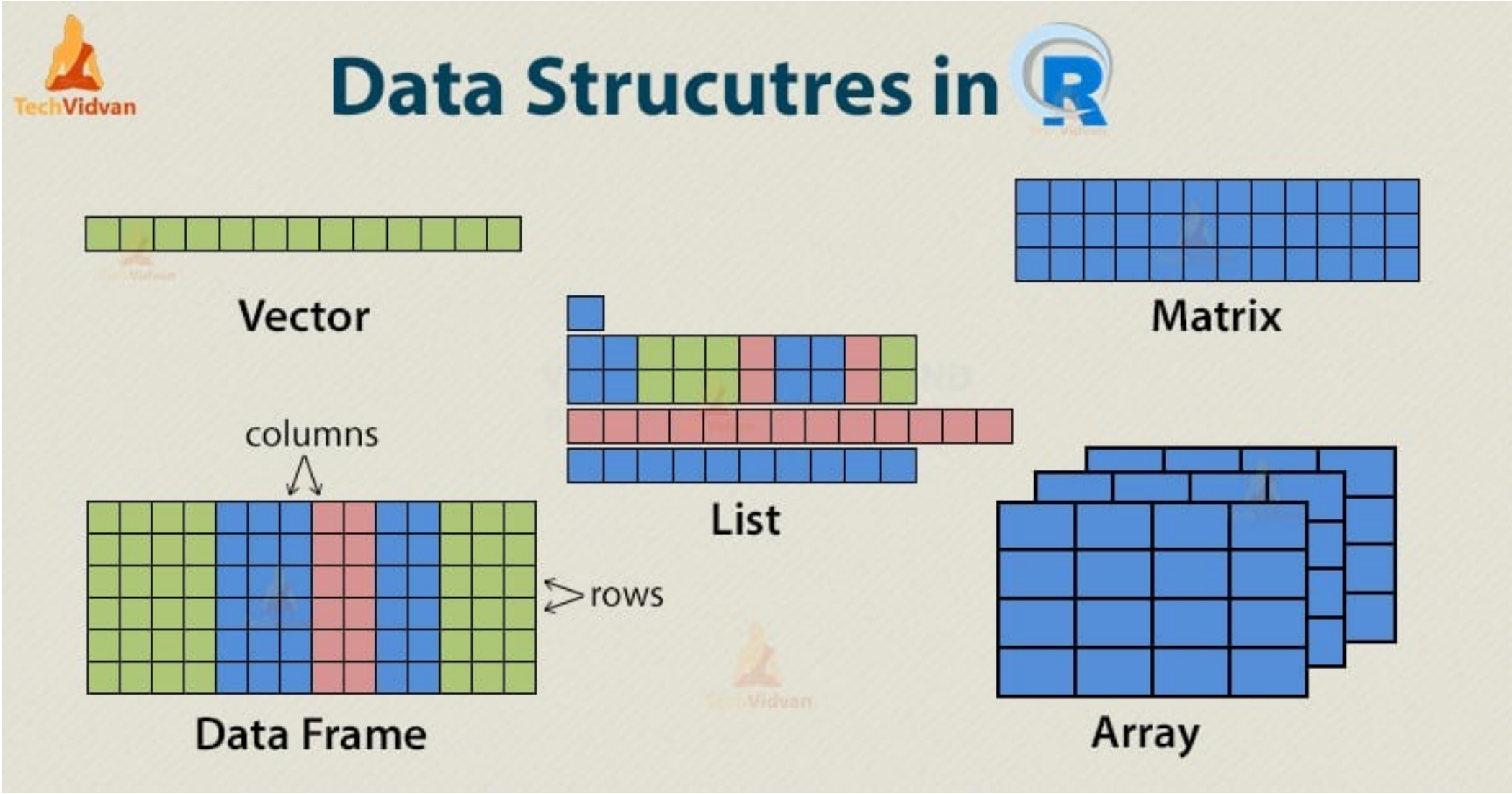
```
> class(TRUE)
[1] "logical"
> mode(TRUE)
[1] "logical"
> typeof(FALSE)
[1] "logical"
>
>
> is.logical(TRUE)
[1] TRUE
> is.logical("TRUE")
[1] FALSE
```

Part 2 R Basic Data Structures

R Basic Data Structures

I. Vector
II. Matrix
III. Data frame
IV. Array
V. List
VI. Factor

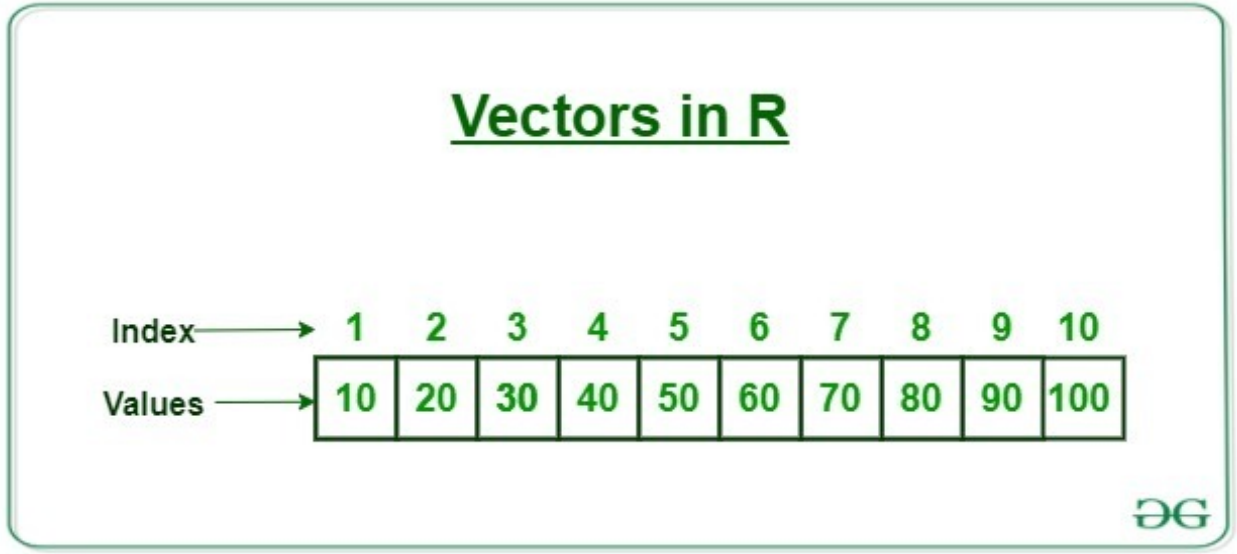
Overall viewpoint



https://techvidvan.com/tutorials/r-data-structures/

1. Vector

- Vectors are single-dimensional, homogeneous data structures.
- It contains elements of the same type.
- The data types can be logical, integer, double, character, complex or raw.



https://www.geeksforgeeks.org/r-vector/

Creating a vector using the c() function.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> mode(x)
[1] "numeric"
> class(x)
[1] "numeric"
> length(x)
[1] 5
> dim(x)
NULL
>
```

```
> x <- c(1, 5.4, TRUE, "hello")
> typeof(x)
[1] "character"
> mode(x)
[1] "character"
> class(x)
[1] "character"
> length(x)
[1] 4
> dim(x)
NULL
>
```

Creating a vector using : operator

```
> # create vector x
> x <- 1:7
> x
[1] 1 2 3 4 5 6 7
> # create vector y
> y <- 2:-2
> y
[1]  2  1  0 -1 -2
>
```

Creating a vector using seq() function

seq {base} R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

```
> seq(1, 3, by=0.2) # specify step size
 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
>
> seq(1, 5, length.out=4) # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
>
>
```

How to access Elements of a Vector?

- Using integer vector as index

```
> x <- 1:7
> # access 3rd element
> x[3]
[1] 3
> # access 2nd and 4th element
> x[c(2, 4)]
[1] 2 4
> # access all but 1st element
> x[-1]
[1] 2 3 4 5 6 7
> # cannot mix positive and negative integers
> x[c(2, -4)]
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
>
```

- Using logical vector as index

```
> x <- 1:5
> x[c(TRUE, FALSE, FALSE, TRUE)]
[1] 1 4 5
> # filtering vectors based on conditions
> x[x < 0]
integer(0)
> x[x > 0]
[1] 1 2 3 4 5
>
>
```

- Using character vector as index

```
> x <- c("first"=3, "second"=0, "third"=9)
> names(x)
[1] "first"  "second" "third"
> x["second"]
second
     0
>
```

How to modify a vector in R?

```
> x <- c(-3,-2,-1,0,1,2)
> # modify 2nd element
> x[2] <- 0; x
[1] -3  0 -1  0  1  2
> # modify elements less than 0
> x[x<0] <- 5; x
[1] 5 0 5 0 1 2
> # truncate x to first 4 elements
> x <- x[1:4]; x
[1] 5 0 5 0
>
```

How to delete a vector in R?

- Delete a vector by simply assigning a NULL to it.

```
> x <- c(-3,-2,-1,0,1,2)
> x <- NULL
> x
NULL
> x[4]
NULL
>
```

2. Matrix

Example of different matrix dimension

2x2 matrix	column 1	column 2
row 1	1	2
row 2	3	4

3x3 matrix	column 1	column 2	Column 3
row 1	1	2	3
row 2	4	5	6
row 3	7	8	9

5x2 matrix	column 1	column 2
row 1	1	2
row 2	3	4
row 3	5	6
row 4	7	8
row 5	9	10

- Matrices are two-dimensional, homogeneous data structures.
- It is similar to vector but additionally contains the dimension attribute.

https://www.guru99.com/r-matrix-tutorial.html

How to Create a Matrix in R

- A matrix can be created with the function matrix(). Following is a function to create a matrix in R which takes three arguments:

- Syntax:
- matrix(data, nrow, ncol, byrow = FALSE)

- Arguments:

✓ data: The collection of elements that R will arrange into the rows and columns of the matrix

✓ nrow: Number of rows.

✓ ncol: Number of columns.

✓ byrow: The rows are filled from the left to the right. We use `byrow = FALSE` (default values).

How to construct a matrix?

```
> # Construct a matrix with 5 rows that contain
> # the numbers 1 up to 10 and byrow = TRUE
> matrix_a <- matrix(1:10, byrow=TRUE, nrow=5, ncol=2)
> matrix_a
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
>
```

```
> # Same result is obtained by providing only one dimension.
> matrix_a <- matrix(1:10, byrow=TRUE, nrow=5)
> matrix_a
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
>
>
```

Here ncol is not provided.

byrow=FALSE OR byrow=TRUE

fill matrix row-wise

```
> # byrow=TRUE
> matrix_a <- matrix(1:10, byrow=TRUE, nrow=5)
> matrix_a
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
>
>
```

fill matrix column-wise

```
> # byrow=FALSE
> matrix_a <- matrix(1:10, byrow=FALSE, nrow=5)
> matrix_a
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
>
>
```

The row and column numbers?

```
> x <- matrix(1:12, nrow = 3,
+     dimnames = list(c("X","Y","Z"),
+                     c("A","B","C","D")))
>
> dim(x)
[1] 3 4
>
> nrow(x)
[1] 3
>
> ncol(x)
[1] 4
>
```

How to name the rows and columns?

```
> # Name the rows and columns of matrix during creation
> # by passing a two-element list to the argument dimnames:
>
> x <- matrix(1:9, nrow = 3,
+     dimnames = list(c("X","Y","Z"),
+                     c("A","B","C")))
>
> x
  A B C
X 1 4 7
Y 2 5 8
Z 3 6 9
>
```

colnames() and rownames()

- These names can be accessed or changed with the following functions: colnames() and rownames()

```
> # access column names and rownames
> colnames(x)
[1] "A" "B" "C"
> rownames(x)
[1] "X" "Y" "Z"
>
> # change column names
> colnames(x) <- c("C1","C2","C3")
> # change row names
> rownames(x) <- c("R1","R2","R3")
> x
   C1 C2 C3
R1  1  4  7
R2  2  5  8
R3  3  6  9
>
```

Attributes of a matrix

```
> is.matrix(x)
[1] TRUE
>
> dim(x)
[1] 3 3
>
> attributes(x)
$dim
[1] 3 3

$dimnames
$dimnames[[1]]
[1] "R1" "R2" "R3"

$dimnames[[2]]
[1] "C1" "C2" "C3"
```

```
> class(x)
[1] "matrix" "array"
>
> mode(x)
[1] "numeric"
>
> typeof(x)
[1] "integer"
```

Create Matrix Using dim()

```
> # define a vector
> x <- c(1,2,3,4,5,6)
>
> # define the row and column numbers
> dim(x) <- c(2,3)
>
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

```
> x <- c(1,2,3,4,5,6)
> class(x)
[1] "numeric"
>
> dim(x) <- c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> class(x)
[1] "matrix" "array"
>
```

Create Matrix Using cbind() and rbind()

```
> # creating a matrix by using functions cbind() and rbind()
> # as in column bind and row bind.
>
> cbind(c(1,2,3),c(4,5,6))
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
>
> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```

How to access Elements of a matrix?

- Elements of a matrix could be accessed using the square bracket [] indexing method
- Syntax: var[row, column]
- Here row and column are vectors

```
> # define the matrix
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)
> # select rows 1 & 2 and columns 2 & 3
> x[c(1,2),c(2,3)]
     [,1] [,2]
[1,]    2    3
[2,]    5    6
>
> # leaving column field blank will select entire columns
> x[c(3,2),]
     [,1] [,2] [,3]
[1,]    7    8    9
[2,]    4    5    6
>
> #  leaving row as well as column field blank will select entire matrix
> x[,]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>
> # select all rows except first
> x[-1,]
     [,1] [,2] [,3]
[1,]    4    5    6
[2,]    7    8    9
```

- If the matrix returned after indexing is a row matrix or column matrix, the result is given as a vector!
- This behavior can be avoided by using the argument drop = FALSE while indexing.

```
>
> x[1,]
[1] 1 2 3
> class(x[1,])
[1] "numeric"
>
> x[c(1,2),]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> class(x[c(1,2),])
[1] "matrix" "array"
>
```

```
> # now the result is a 1X3 matrix rather than a vector
> x[1,,drop=FALSE]
     [,1] [,2] [,3]
[1,]    1    2    3
>
> class(x[1,,drop=FALSE])
[1] "matrix" "array"
>
```

Using logical vector as index

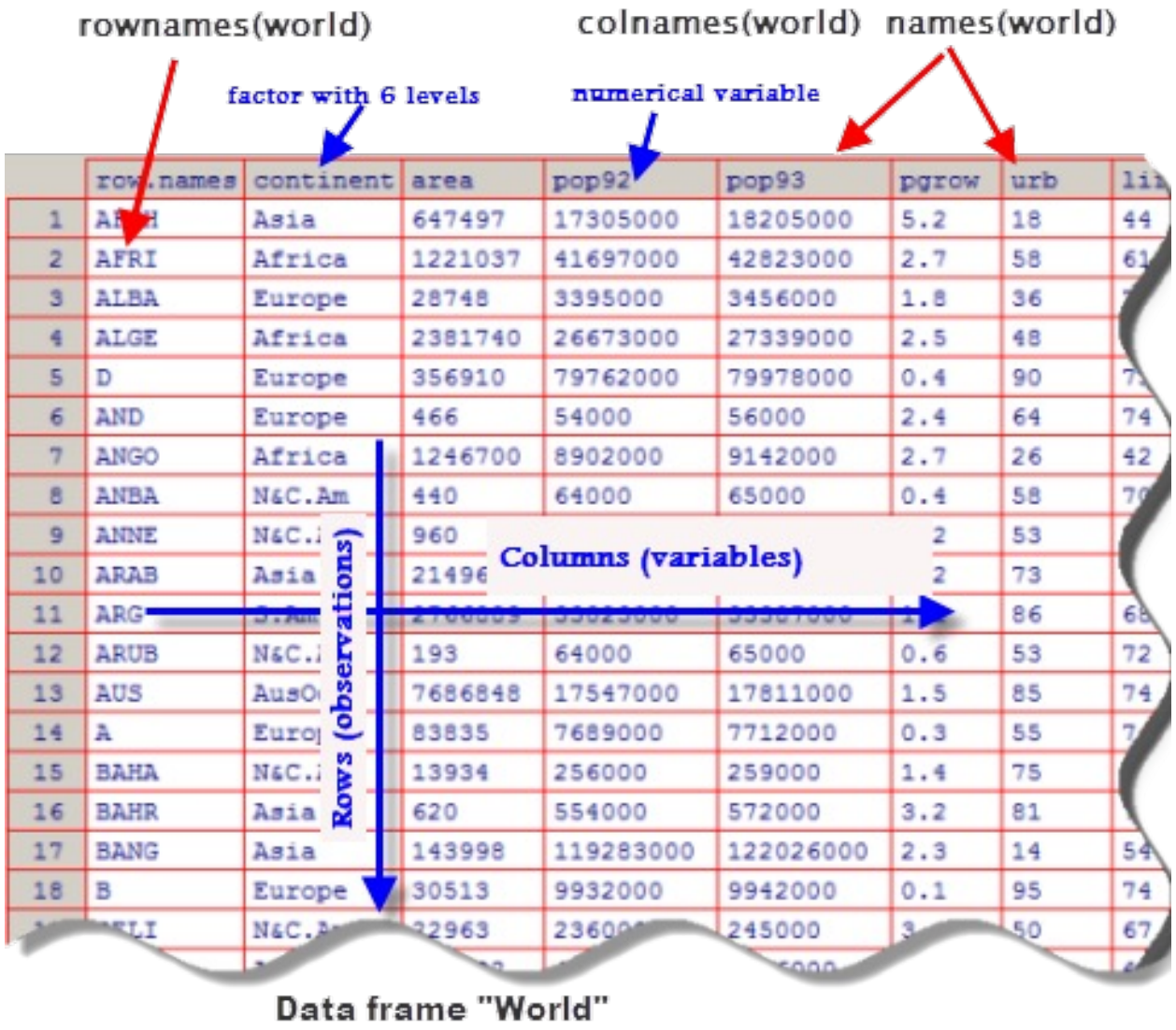
- Two logical vectors can be used to index a matrix.
- Can be mixed with integer vectors.

```
> x[c(TRUE,FALSE,TRUE),c(TRUE,TRUE,FALSE)]
     [,1] [,2]
[1,]    1    2
[2,]    7    8
>
> x[c(TRUE,FALSE),c(2,3)]
     [,1] [,2]
[1,]    2    3
[2,]    8    9
>
```

Using character vector as index

```
> # create a matrix with specified values and column names
> x <- matrix(c(4, 6, 1, 8, 0, 2, 3, 7, 9),
+     nrow = 3, ncol = 3, byrow = TRUE,
+     dimnames = list(NULL, c("A", "B", "C")))
>
> # subset the matrix by selecting the "A" column
> x[,"A"]
[1] 4 8 3
>
> # subset the matrix by selecting rows that are TRUE and columns "A" and "C"
> x[TRUE, c("A", "C")]
     A C
[1,] 4 1
[2,] 8 2
[3,] 3 9
>
> # subset the matrix by selecting rows 2 to 3 and columns "A" and "C"
> x[2:3, c("A", "C")]
     A C
[1,] 8 2
[2,] 3 9
>
```

3. Data frame



	row.names	continent	area	pop92	pop93	pgrow	urb	li
1	AR H	Asia	647497	17305000	18205000	5.2	18	44
2	AFRI	Africa	1221037	41697000	42823000	2.7	58	61
3	ALBA	Europe	28748	3395000	3456000	1.8	36	
4	ALGE	Africa	2381740	26673000	27339000	2.5	48	
5	D	Europe	356910	79762000	79978000	0.4	90	
6	AND	Europe	466	54000	56000	2.4	64	74
7	ANGO	Africa	1246700	8902000	9142000	2.7	26	42
8	ANBA	N&C.Am	440	64000	65000	0.4	58	
9	ANNE	N&C.A	960			2	53	
10	ARAB	Asia	21496			2	73	
11	ARG	S.Am					86	
12	ARUB	N&C.A	193	64000	65000	0.6	53	72
13	AUS	AusO	7686848	17547000	17811000	1.5	85	74
14	A	Europ	83835	7689000	7712000	0.3	55	
15	BAHA	N&C.A	13934	256000	259000	1.4	75	
16	BAHR	Asia	620	554000	572000	3.2	81	
17	BANG	Asia	143998	119283000	122026000	2.3	14	54
18	B	Europe	30513	9932000	9942000	0.1	95	74
	ELI	N&C.A	22963	236	245000	3	50	67

- Data Frames are data displayed in a format as a table.
- Data Frames can have different types of data inside it.
- Each column should have the same type of data.

http://www.unige.ch/ses/sococ/cl/r/concepts/dataframe.e.html

https://www.w3schools.com/r/r_data_frames.asp

How to define a data frame?

```
> x <- data.frame(
+     SN = c(1, 2, 3, 4),
+     Age = c(21, 15, 17, 18),
+     Name = c("John", "Dora", "Lucy", "Lily"),
+     gender = c("M","F","F","F"))
>
> # print the data frame
> print(x)
  SN Age Name gender
1  1  21 John      M
2  2  15 Dora      F
3  3  17 Lucy      F
4  4  18 Lily      F
>
> # check the type of x
> print(typeof(x))
[1] "list"
>
> # check the class of x
> print(class(x))
[1] "data.frame"
>
> # check the mode of x
> print(mode(x))
[1] "list"
>
```

The structure of the data frame

```
> str(x)
'data.frame':  4 obs. of  4 variables:
 $ SN     : num  1 2 3 4
 $ Age    : num  21 15 17 18
 $ Name   : chr  "John" "Dora" "Lucy" "Lily"
 $ gender: chr  "M" "F" "F" "F"
>
```

factor or not?

- data.frame() function converts character vectors into factors.
- Pass the argument stringsAsFactors=TRUE.

```
> x <- data.frame(
+      SN = c(1, 2, 3, 4),
+      Age = c(21, 15, 17, 18),
+      Name = c("John", "Dora", "Lucy", "Lily"),
+      gender = c("M","F","F","F"),
+      stringsAsFactors = TRUE)
>
> str(x)
'data.frame':  4 obs. of  4 variables:
 $ SN     : num  1 2 3 4
 $ Age    : num  21 15 17 18
 $ Name   : Factor w/ 4 levels "Dora","John",..: 2 1 4 3
 $ gender: Factor w/ 2 levels "F","M": 2 1 1 1
>
```

How to Access Components of a Data Frame?

- Just like a list, use either [, [[or $ operator to access columns of data frame.

```
> # access the "Name" column using different methods
> print(x["Name"])
  Name
1 John
2 Dora
3 Lucy
4 Lily
> print(x$Name)
[1] "John" "Dora" "Lucy" "Lily"
> print(x[["Name"]])
[1] "John" "Dora" "Lucy" "Lily"
> print(x[[3]])
[1] "John" "Dora" "Lucy" "Lily"
>
```

- Or like a matrix

```
> # select rows 2 and 3 of x
> x[2:3, ]
  SN Age Name gender
2  2  15 Dora      F
3  3  17 Lucy      F
>
> # select rows with Age greater than 15
> x[x$Age > 15, ]
  SN Age Name gender
1  1  21 John      M
3  3  17 Lucy      F
4  4  18 Lily      F
>
> # select the Name column of rows 2 to 3
> x[2:3, "Name"]
[1] "Dora" "Lucy"
>
```

How to modify a Data Frame in R?

- Data frames can be modified like we modified matrices through reassignment.

```
> x <- data.frame(
+     SN = c(1, 2),
+     Age = c(21, 15),
+     Name = c("John", "Dora")
+ )
>
> # print the initial data frame
> print(x)
  SN Age Name
1  1  21 John
2  2  15 Dora
```

```
> # update the Age value in the first row to 20
> x[1, "Age"] <- 20
>
> # print the updated data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
>
>
>
```

Adding Components to Data Frame

```
> x <- data.frame(
+     SN = c(1, 2),
+     Age = c(20, 15),
+     Name = c("John", "Dora")
+ )
>
> # print the initial data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
>
>
> # create a new row and bind it to the data frame
> new_row <- list(SN = 1, Age = 16, Name = "Paul")
> x <- rbind(x, new_row)
>
> # print the updated data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
3  1  16 Paul
```

Step1: create a list with same elements
list() function

Step2: insert this new record into the initial data frame
rbind() function

```
> x <- data.frame(
+   SN = c(1, 2),
+   Age = c(20, 15),
+   Name = c("John", "Dora")
+ )
>
> # print the initial data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
>
> # add a new column "State" to the data frame using cbind()
> x <- cbind(x, State = c("NY", "FL"))
>
> # print the updated data frame
> print(x)
  SN Age Name State
1  1  20 John    NY
2  2  15 Dora    FL
>
```

Step1 prepare a vector

Step2 insert as a column cbind()

What could cbind() function create?

```
> SN = c(1, 2)
> Age = c(20, 15)
> Name = c("John", "Dora")
>
> x <- cbind(SN, Age, Name)
>
> x
     SN  Age  Name
[1,] "1" "20" "John"
[2,] "2" "15" "Dora"
> class(x)
[1] "matrix" "array"
> is.data.frame(x)
[1] FALSE
>
> ?cbind
> mode(x)
[1] "character"
>
```

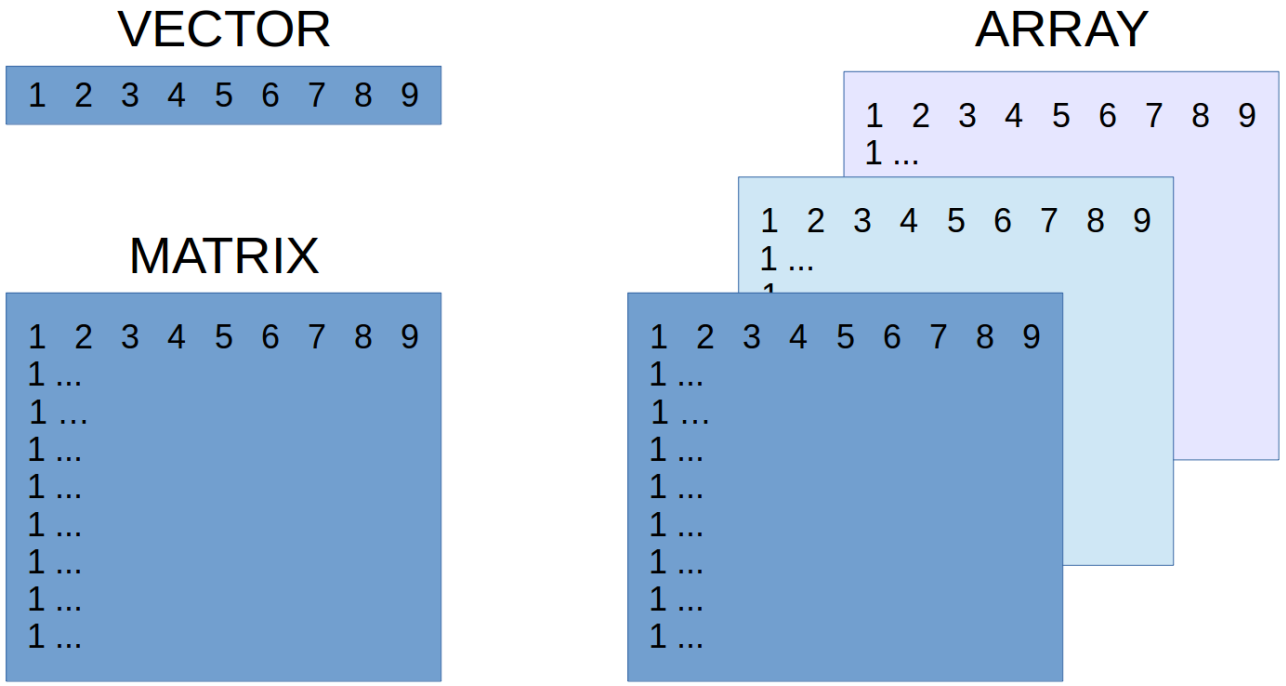
How to remove one column?

```
> x <- data.frame(
+     SN = c(1, 2),
+     Age = c(20, 15),
+     Name = c("John", "Dora"),
+     State = c("NY", "FL")
+ )
>
> # print the initial data frame
> print(x)
  SN Age Name State
1  1  20 John    NY
2  2  15 Dora    FL
>
> # remove the "State" column from the data frame
> x$State <- NULL
>
> # print the updated data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
```

```
> x <- data.frame(
+     SN = c(1, 2),
+     Age = c(20, 15),
+     Name = c("John", "Dora"),
+     State = c("NY", "FL")
+ )
>
> # print the initial data frame
> print(x)
  SN Age Name State
1  1  20 John    NY
2  2  15 Dora    FL
>
> # select the first three columns
> x <- x[,1:3]
> # print the updated data frame
> print(x)
  SN Age Name
1  1  20 John
2  2  15 Dora
>
```

4. Array

- Arrays are three dimensional, homogeneous data structures.



creating with the use of array() function

Syntax:

array(data, dim = (nrow, ncol, nmat), dimnames=names)

- ✓ nrow : Number of rows
- ✓ ncol : Number of columns
- ✓ nmat : Number of matrices of dimensions nrow * ncol
- ✓ dimnames : Default value = NULL.

```
> # arranges data from 2 to 13
> # in two matrices of dimensions 2x3
> arr = array(2:13, dim = c(2, 3, 2))
> print(arr)
, , 1

     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7

, , 2

     [,1] [,2] [,3]
[1,]    8   10   12
[2,]    9   11   13

>
```

```
> vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> vec2 <- c(10, 11, 12)
>
> # elements are combined into a single vector,
> # vec1 elements followed by vec2 elements.
> arr = array(c(vec1, vec2), dim = c(2, 3, 2))
> print (arr)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

```
> row_names <- c("row1", "row2")
> col_names <- c("col1", "col2", "col3")
> mat_names <- c("Mat1", "Mat2")
>
> # the naming of the various elements
> # is specified in a list and
> # fed to the function
> arr = array(2:14, dim = c(2, 3, 2),
+             dimnames = list(row_names,
+                             col_names, mat_names))
> print (arr)
, , Mat1

     col1 col2 col3
row1    2    4    6
row2    3    5    7

, , Mat2

     col1 col2 col3
row1    8   10   12
row2    9   11   13

>
```

Where is the number 14?

Accessing entire matrices

- The elements can be accessed by using indexes of the corresponding elements.

```
> vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> vec2 <- c(10, 11, 12)
> row_names <- c("row1", "row2")
> col_names <- c("col1", "col2", "col3")
> mat_names <- c("Mat1", "Mat2")
> arr = array(c(vec1, vec2), dim = c(2, 3, 2),
+               dimnames = list(row_names,
+                               col_names, mat_names))
>
> # accessing matrix 1 by index value
> print ("Matrix 1")
[1] "Matrix 1"
> print (arr[,,1])
     col1 col2 col3
row1    1    3    5
row2    2    4    6
>
> # accessing matrix 2 by its name
> print ("Matrix 2")
[1] "Matrix 2"
> print(arr[,,"Mat2"])
     col1 col2 col3
row1    7    9   11
row2    8   10   12
```

```
> arr
, , Mat1

     col1 col2 col3
row1    1    3    5
row2    2    4    6

, , Mat2

     col1 col2 col3
row1    7    9   11
row2    8   10   12
```

Accessing specific rows and columns of matrices

```
> vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> vec2 <- c(10, 11, 12)
> row_names <- c("row1", "row2")
> col_names <- c("col1", "col2", "col3")
> mat_names <- c("Mat1", "Mat2")
> arr = array(c(vec1, vec2), dim = c(2, 3, 2),
+             dimnames = list(row_names,
+                             col_names, mat_names))
>
> # accessing matrix 1 by index value
> print ("1st column of matrix 1")
[1] "1st column of matrix 1"
> print (arr[, 1, 1])
row1 row2
   1    2
>
> # accessing matrix 2 by its name
> print ("2nd row of matrix 2")
[1] "2nd row of matrix 2"
> print(arr["row2",,"Mat2"])
col1 col2 col3
   8   10   12
>
```

```
> arr
, , Mat1

     col1 col2 col3
row1    1    3    5
row2    2    4    6

, , Mat2

     col1 col2 col3
row1    7    9   11
row2    8   10   12

>
```

Accessing elements individually

```
> vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> vec2 <- c(10, 11, 12)
> row_names <- c("row1", "row2")
> col_names <- c("col1", "col2", "col3")
> mat_names <- c("Mat1", "Mat2")
> arr = array(c(vec1, vec2), dim = c(2, 3, 2),
+       dimnames = list(row_names, col_names, mat_names))
>
> # accessing matrix 1 by index value
> print ("2nd row 3rd column matrix 1 element")
[1] "2nd row 3rd column matrix 1 element"
> print (arr[2, "col3", 1])
[1] 6
>
> # accessing matrix 2 by its name
> print ("2nd row 1st column element of matrix 2")
[1] "2nd row 1st column element of matrix 2"
> print(arr["row2", "col1", "Mat2"])
[1] 8
>
```

```
> arr
, , Mat1

     col1 col2 col3
row1    1    3    5
row2    2    4    6

, , Mat2

     col1 col2 col3
row1    7    9   11
row2    8   10   12
```

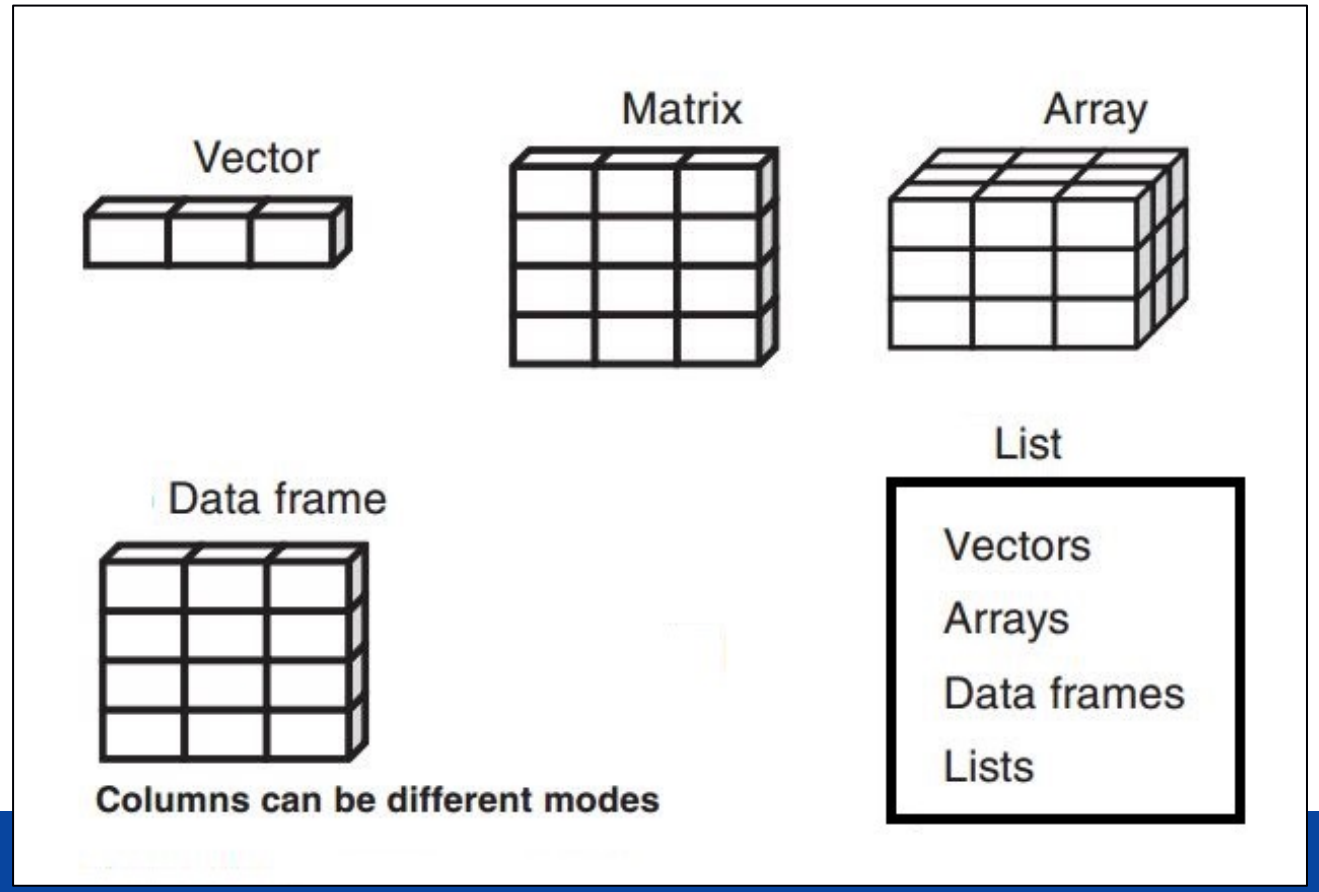
Accessing subset of array elements

```
> row_names <- c("row1", "row2")
> col_names <- c("col1", "col2", "col3", "col4")
> mat_names <- c("Mat1", "Mat2")
> arr = array(1:15, dim = c(2, 4, 2),
+       dimnames = list(row_names, col_names, mat_names))
>
> # print elements of both the rows and columns 2 and 3 of matrix 1
> print (arr[, c(2, 3), 1])
     col2 col3
row1    3    5
row2    4    6
>
```

```
> arr
, , Mat1

     col1 col2 col3 col4
row1    1    3    5    7
row2    2    4    6    8

, , Mat2

     col1 col2 col3 col4
row1    9   11   13   15
row2   10   12   14    1
```

5. List

- If a vector has elements of different types, it is called a list in R programming.
- A list is a flexible data structure that can hold elements of different types, such as numbers, characters, vectors, matrices, and even other lists.



https://rpubs.com/sheikh/list

How to create a list in R programming?

```
> # create a list using the list() function.
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
> x
$a
[1] 2.5

$b
[1] TRUE

$c
[1] 1 2 3

>
> typeof(x)
[1] "list"
>
> mode(x)
[1] "list"
>
> class(x)
[1] "list"
```

```
> str(x)
List of 3
 $ a: num 2.5
 $ b: logi TRUE
 $ c: int [1:3] 1 2 3
>
>
```

```
> # tags are optional
> x <- list(2.5,TRUE,1:3)
> x
[[1]]
[1] 2.5

[[2]]
[1] TRUE

[[3]]
[1] 1 2 3

>
```

How to access components of a list?

- Integer, logical or character vectors can be used for indexing.

```
>
> x <- list(name = "John", age = 19, speaks = c("English", "French"))
>
> # access elements by name
> x$name
[1] "John"
> x$age
[1] 19
> x$speaks
[1] "English" "French"
>
```

```
> x[c(1, 2)]
$name
[1] "John"

$age
[1] 19

> x[-2]
$name
[1] "John"

$speaks
[1] "English" "French"
```

```
> # access elements by logical index
> x[c(TRUE, FALSE, FALSE)]
$name
[1] "John"

>
> # access elements by character index
> x[c("age", "speaks")]
$age
[1] 19

$speaks
[1] "English" "French"
```

```
> x <- list(name = "John", age = 19, speaks = c("English", "French"))
>
> # access element by name using single bracket []
> x["age"]
$age
[1] 19


>
> # check the type of the result (single bracket returns a list)
> typeof(x["age"])
[1] "list"
>
> # access element by name using double bracket [[]]
> x[["age"]]
[1] 19
>
> # check the type of the result (double bracket returns the content)
> typeof(x[["age"]])
[1] "double"
>
```

```
> x <- list(name = "John", age = 19, speaks = c("English", "French"))
>
> # access element by exact matching using $
> x$name
[1] "John"
>
> # access element by partial matching using $
> x$age
[1] 19
>
> # access element by partial matching using $
> x$speaks
[1] "English" "French"
>
> # create a list with similar tags
> y <- list(n = "Alice", a = 25, s = c("Spanish", "Italian"))
>
> # access element by partial matching using $
> y$n
[1] "Alice"
>
> # access element by partial matching using $
> y$a
[1] 25
>
> # access element by partial matching using $
> y$s
[1] "Spanish" "Italian"
>
```

How to modify a list in R?

```
> x <- list(name = "John", age = 19, speaks = c("English", "French"))
>
> # access element by double brackets [[]] and update its value
> x[["name"]] <- "Clair"
>
> # print the updated list
> x
$name
[1] "Clair"

$age
[1] 19

$speaks
[1] "English" "French"

>
```

How to add components to a list?

```
> x <- list(name = "Clair", age = 19, speaks = c("English", "French"))
>
> # assign a new element to the list using double brackets [[]]
> x[["married"]] <- FALSE
>
> # print the updated list
> x
$name
[1] "Clair"

$age
[1] 19

$speaks
[1] "English" "French"

$married
[1] FALSE

>
```

How to delete components from a list?

```
> # delete a component by assigning NULL to it.
> x <- list(name = "Clair", age = 19, speaks = c("English", "French"))
>
> # remove an element from the list using double brackets [[]]
> x[["age"]] <- NULL
>
> # print the structure of the updated list
> str(x)
List of 2
 $ name  : chr "Clair"
 $ speaks: chr [1:2] "English" "French"
>
> # remove an element from the list using $ notation
> x$name <- NULL
>
> # print the structure of the updated list
> str(x)
List of 1
 $ speaks: chr [1:2] "English" "French"
>
```

6. Factor

- Factor is a data structure used for fields that takes only a predefined, finite number of values (categorical data).
- A factor could be created using the function factor().

How to create a factor in R?

- How to change the sequence of the levels?

```
> x <- factor(c("single", "married", "married", "single"))
> print(x)
[1] single  married married single
Levels: married single
>
> x <- factor(c("single", "married", "married", "single"),
+     levels = c("single", "married", "divorced"))
> print(x)
[1] single  married married single
Levels: single married divorced
>
>
```

How to access components of a factor?

- Just like accessing a vector

```
> x <- factor(c("single", "married", "married", "single"))
> print(x)
[1] single  married married single
Levels: married single
>
> print(x[3])
[1] married
Levels: married single
> print(x[c(2, 4)])
[1] married single
Levels: married single
> print(x[-1])
[1] married married single
Levels: married single
> print(x[c(TRUE, FALSE, FALSE, TRUE)])
[1] single single
Levels: married single
>
>
```

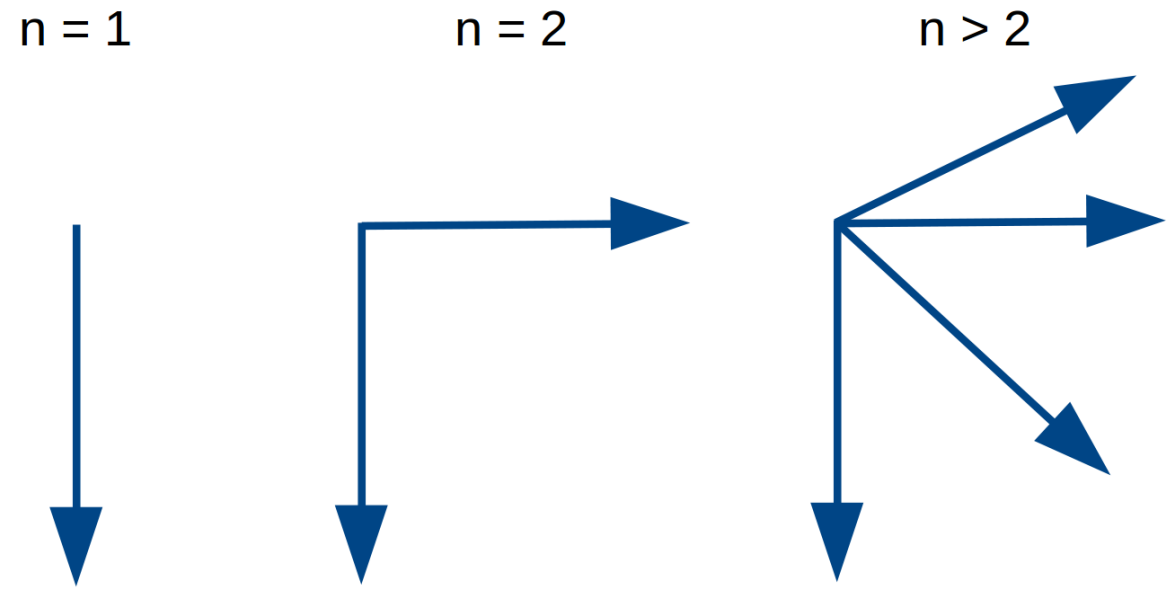
How to modify a factor?

```
>
> x <- factor(c("single", "married", "married", "single"),
+     levels = c("single", "married", "divorced"))
> print(x)
[1] single  married married single
Levels: single married divorced
>
> x[2] <- "divorced"
> print(x)
[1] single   divorced married  single
Levels: single married divorced
>
> x[3] <- "widowed"
Warning message:
In `[<-.factor`(`*tmp*`, 3, value = "widowed") :
  invalid factor level, NA generated
> print(x)
[1] single   divorced <NA>     single
Levels: single married divorced
>
```

- A workaround to this is to add the value to the levels, it's important!

```
> x <- factor(c("single", "divorced", "widowed", "single"),
+     levels = c("single", "married", "divorced"))
> print(x)
[1] single   divorced <NA>     single
Levels: single married divorced
>
> levels(x) <- c(levels(x), "widowed")
> x[3] <- "widowed"
> print(x)
[1] single   divorced widowed  single
Levels: single married divorced widowed
>
>
```

Data objects have different dimensions



Vector

Factor

List

Matrix

Dataframe

Array

Data objects can be homogenous or heterogenous

HOMOGENEOUS
(elements are only 1 type)

Vector

Matrix

Array

HETEROGENEOUS
(elements can be different)

Dataframe

List

http://www.simonqueenborough.info/R/basic/data-types.html

Functions you should know

- is.numeric(), is.character(), is.matrix(), is.data.frame() ….
- mode(), typeof(), class() ….
- dim(), length(), str(), cbind(), rbind(), names() ….
- matrix(), data.frame(), list(), array(), c() ….

请认真复习
独立完成作业

Please review carefully and complete the assignment independently