Data Types and Data Structures in R

Learning Objectives

1. Know the common R data types: numeric, character, logical
2. Know the common R data structures: vector, matrix, array, data frame, list
3. Give column and row names to various data structures
4. Know how elements within objects are indexed
5. Find the dimensions of objects
6. Convert between data types

We will working extensively with objects in R. An object is where you store various information. For example, to store the numeric value of 30 into an object called `age`

```
age<-30 ##recommended way
age=30
```

The value 30 is assigned to the object called `age`. There are two ways to assign a value to an object:

1. using `<-`
2. using `=`.

I highly recommend using `<-` instead of `=`.

There are a few rules in naming objects:

1. it must begin with a letter or `.`
2. if begins with `.`, the `.` cannot be followed by a number
3. only letters, numbers, `.`, and `_` can be used.

One important thing is to know the data type and structure associated with the object you are operating on. Having a good understanding of data types and structures will be useful since how you operate on objects depends on the object's data type and structure.

1. Common R data types: numeric, character, logical

There are a number of data types in R. The ones that we will be working with most often are

1. numeric
2. character
3. logical

Numeric

A numeric type is a number. The `age` object that we just created is a numeric type. We can confirm by using `class()` or `is.numeric()`

```
class(age) ##or
```

```
## [1] "numeric"
```

```
is.numeric(age)
```

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

Character

A character type is associated with text. For example, we want to assign the character `Monday` to an object called `today`

```
today<-"Monday"
```

Characters must be surrounded by quotation marks, to distinguish characters from objects. If you leave out the quotation marks

```
today<-Monday
```

returns an error message as R thinks `Monday` is the name of another object, which we have not declared.

To verify the data type associated with `today`

```
class(today) ##or
```

```
## [1] "character"
```

```
is.character(today)
```

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

Logical

A logical type takes on two values: `TRUE` or `FALSE`. Logical types are useful when we want certain conditions to be satisfied. To assign `TRUE` to an object, we can type `TRUE` or just `T` (likewise, `F` can be used in place of `FALSE`)

```
yes_or_no<-TRUE ##or
yes_or_no2<-T
is.logical(yes_or_no)
```

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

```
class(yes_or_no2)
```

```
## [1] "logical"
```

The following are symbols to express logic in R:

- `==`: equal to
- `!=`: not equal to
- `>`: greater than
- `>=`: greater than or equal to
- `<`: less than
- `<=`: less than or equal to
- `&`: and
- `|`: or

Look at the following examples

```
(2+1)==3
```

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

```
(2+1)!=3
```

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

```
(5+7)>12
```

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

```
(5+7)>=12
```

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

```
(2+1==3)&(2+1==4)
```

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

```
(2+1==3)|(2+1==4)
```

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

There are other data types such as integer and complex which we will not cover in this document. We will next move on to data structures.

2. Common R data structures: vector, matrix, array, data frame, list

We will explore the most common R data structures:

1. vector
2. matrix
3. array
4. data frame
5. list

Vector

A vector is the most basic and common data structure. A vector contains a collection of elements, and all elements in a vector must be of the same type. Technically speaking, the object `age` that we created is a vector, with just one element, the numeric value 30. To create a vector with more than one element, use `c()`. For example, to create a numeric vector with five numeric elements

```r
Age<-c(30,18,19,20,21)
Age
```

```
## [1] 30 18 19 20 21
```

Note that R is case sensitive, so `Age` and `age` are considered different objects.

To create a character vector

```r
Day_of_week<-c("Monday","Wednesday","Saturday","Sunday","Monday")
Day_of_week
```

```
## [1] "Monday"    "Wednesday" "Saturday"  "Sunday"    "Monday"
```

To create a logical vector

```r
logic<-c(TRUE,T,FALSE,F,T)
logic
```

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

We can check the type associated with vector `Age`

```r
is.vector(Age)
```

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

```r
is.numeric(Age)
```

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

```r
is.character(Age)
```

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

```r
is.logical(Age)
```

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

If you end up mixing elements of various types in a vector, R will coerce all the elements to be of a single type. The order is character $>$ numerical $>$ logical. For example, a vector with one character element and one numerical element will coerce all elements to the character type, since character is higher in order. Consider the following examples

```r
try<-c(3.1, "Tuesday")
class(try)
```

```
## [1] "character"
```

```r
try2<-c(T, "Friday")
class(try2)
```

```
## [1] "character"
```

```r
try3<-c(3.1, F)
class(try3)
```

```
## [1] "numeric"
```

Matrix

A matrix is a two-dimensional $r \times c$ object, where r and c denote the number of rows and columns respectively. A matrix can be viewed as an extension of a vector, or as side by side vectors. To create a matrix, we use the `matrix()` function, specify the values of the elements, as well as the number of rows and columns. For example, the following produces a 3×2 matrix

```r
mat.A<-matrix(c(4,3,6,1,2,1), nrow=3, ncol=2)
mat.A
```

```
##      [,1] [,2]
## [1,]    4    1
## [2,]    3    2
## [3,]    6    1
```

Just like vectors, all elements in a matrix must be of the same type. If not, R will coerce all elements to be the same type, in the same order: character $>$ numerical $>$ logical.

Array

An array is a three-dimensional $r \times c \times h$ object. We can think of an array as a bunch of $r \times c$ matrices. We use the `array()` function to create an array, for example

```
arr.A<-array(0, dim=c(2,3,2))
arr.A
```

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

creates a 2×3 matrix twice, with all elements being 0.

Just like vectors and matrices, all elements in an array need to be the same type. If not, R will coerce the elements in the same manner as vectors and matrices.

Data frame

Data frames are probably the data structure that we will be dealing with most often. A data frame is similar to a matrix, except that a data frame does not require all elements to be the same type. Whenever we read in a data file into R, the elements are stored as a data frame. Within each column, all elements must be the same data type. But data type may vary between columns. To combine vectors of the same length to a data frame

```
Data<-data.frame(Age,Day_of_week,logic)
Data
```

```
##   Age Day_of_week logic
## 1  30      Monday  TRUE
## 2  18   Wednesday  TRUE
## 3  19    Saturday FALSE
## 4  20      Sunday FALSE
## 5  21      Monday  TRUE
```

List

Lists act as containers, and can contain objects of different types and structures. Use `list()` to create a list, for example

```
big<-list(mat.A, Day_of_week, Age, age, logic)
big
```

```
## [[1]]
##      [,1] [,2]
## [1,]    4    1
## [2,]    3    2
## [3,]    6    1
```

```
##
## [[2]]
## [1] "Monday"    "Wednesday" "Saturday"  "Sunday"    "Monday"
##
## [[3]]
## [1] 30 18 19 20 21
##
## [[4]]
## [1] 30
##
## [[5]]
## [1]  TRUE  TRUE FALSE FALSE  TRUE
```

3. Give column and row names to objects

There are different functions to supply names to rows and columns, depending on the data structure:

- `names()` for vectors, data frames, and lists
- `rownames()` and `colnames()` for matrices and data frames
- `dimnames()` for arrays

For example

```
names(Data)<-c("Friends", "Pets", "Logic?")
Data
```

```
##   Friends      Pets Logic?
## 1      30    Monday   TRUE
## 2      18 Wednesday   TRUE
## 3      19  Saturday  FALSE
## 4      20    Sunday  FALSE
## 5      21    Monday   TRUE
```

4. How elements within objects are indexed

Knowing how elements within objects are indexed is very important, since we often want to extract specific elements from an object. The way elements within objects are indexed differ based on the data structure:

- vectors: `[i]` for the ith element
- matrices and data frames: `[i,j]` for element in the ith row, jth column
- arrays: `[i,j,k]` for element in the ith row, jth column, kth level
- lists: `[[i]]` for the ith object

For example

```
##extract a specific element in vector
Age[3] ##third element
```

```
## [1] 19
```

```r
##extract a specific element in matrix
mat.A[2,1] ##row 2, column 1
```

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

```r
##extract object 1 and 2 from list
big[[1]]
```

```
##      [,1] [,2]
## [1,]    4    1
## [2,]    3    2
## [3,]    6    1
```

```r
big[[2]]
```

```
## [1] "Monday"    "Wednesday" "Saturday"  "Sunday"    "Monday"
```

5. Dimensions of objects

Another important thing we need to know is to find the dimension associated with objects. Again, we use different functions based on the data structure:

- `length()` for vectors
- `dim()`, `nrow()`,ncol()‘ for matrices, arrays, and data frames

```r
##number of elements in vector
length(Day_of_week)
```

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

```r
##number of rows and cols in matrix, array, data frame
dim(mat.A)
```

```
## [1] 3 2
```

```r
nrow(mat.A)
```

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

```r
ncol(mat.A)
```

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

```r
dim(arr.A)
```

```
## [1] 2 3 2
```

6. Convert between data types

Convert numeric to character

Sometimes, depending on how we wish to analyze the data, we may need to convert between data types. For example, the vector `Age()` contains integer values and are numeric. We can apply methods for both quantitative and categorical data when we have a discrete variable. So, to apply methods for categorical data on `Age()`, we need to convert its type to character

```
Age2<-as.character(Age)
Age2
```

```
## [1] "30" "18" "19" "20" "21"
```

Notice the values of `Age2` are in quotation marks. The quotation marks indicate that these values are characters and not numeric.

The `mean()` function can be applied to `Age` since it is numeric, but not the `Age2`, since it is a character and not numeric

```
mean(Age)
```

```
## [1] 21.6
```

```
mean(Age2) ##error
```

```
## Warning in mean.default(Age2): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

Factor

The `factor()` function is useful when we want to order the character elements in a specific manner. By default, the character elements are ordered in alphabetical order

```
DOW<-factor(Day_of_week)
DOW
```

```
## [1] Monday    Wednesday Saturday  Sunday    Monday
## Levels: Monday Saturday Sunday Wednesday
```

Notice the order of the levels in the output are in alphabetical order. To change the order, we `factor()`

```
DOW<-factor(Day_of_week,levels = c("Sunday","Monday","Wednesday","Saturday"))
DOW
```

```
## [1] Monday    Wednesday Saturday  Sunday    Monday
## Levels: Sunday Monday Wednesday Saturday
```

So now the order is based on what we supplied to `factor()`.

Convert numeric to factor

To convert from numeric to factor

```
Age_ord<-factor(Age)
Age_ord
```

```
## [1] 30 18 19 20 21
## Levels: 18 19 20 21 30
```

By default, R arranges numeric from smallest to largest during the conversion to a factor.

Convert factor to numeric

Suppose we want to convert `Age_ord` back to numeric

```
Age3<-as.numeric(as.character(Age_ord))
Age3
```

```
## [1] 30 18 19 20 21
```

Note that we need to first convert `Age_ord` to a character, then convert to numeric.