Northeastern University

CS6020: Collecting, Storing, and Retrieving Information

Basic Data Shaping

Basic Data Shaping

R DATA TYPES

Lesson Objectives

- After completing this lesson, you are able to:
  - explain the difference between mode and class of an R data object
  - convert between different data types
  - assess the mode and class of an object
  - create vectors

Modes and Classes of Objects

- R stores everything as objects:

Object	Description	Example Values
Constant	a numeric value	8, -3, 3.14
Text	a string of characters	"CS6020"
NULL	null reference	NULL
NA	missing value	NA
NaN	not a number	NaN
Inf	infinity	Inf

Variable Naming Rules

- Objects are generally assigned to variables using the **<-** operator.
- Variables are identifiers and are subject to the following naming rules:
  - must start with letter followed by upper and lower case letters, digits, period, and underscore (_)
  - no special characters: #, @, &, %, ^, $, ~, *
- Examples: `rangeValue`, `i3`, `open_date`

Identifier Naming Conventions

- Most R programmers follow the camel case or period separated naming conventions for variables and functions.
- Function naming:
  - lower case followed by mixed upper lower case without underscores
- Variable naming:
  - lower case followed by mixed upper lower case without underscores
  - lower case with period separator

Example Identifiers

- Functions:
  - `startProcessingDataSet()`
  - `loadVals()`
- Variables and Parameters:
  - `startDate`
  - `val`
  - `start.date`

Attributes of R Objects

- Objects in R have attributes:
  - mode
    - numeric, character, logical
  - class
- The mode of objects must match when they are combined in an operation.

Inspecting the Mode

- The `mode()` function returns the mode of an object.

```
> num.obj <- seq(from=1,to=10,by=2)
> mode(num.obj)
[1] "numeric"
> logical.obj <- c(TRUE,TRUE,FALSE,TRUE)
> mode(logical.obj)
[1] "logical"
> character.obj <- c("h","e","l","l","o")
> mode(character.obj)
[1] "character"
```

Important Facts About Mode

- R stores numeric objects as either 32-bit integers or double-precision floating point numbers.
- If an R object contains both numeric and logical elements, the mode of the objects is numeric and all logical elements are converted to numeric values with TRUE = 1 and FALSE = 0.
- If an R object contains character and numeric or logical elements, it is converted to character mode.

Mode Conversions

- Examples of mode conversion in mixed mode objects:

```
> nl <- c(1,3,TRUE,9,FALSE)
> nl
[1] 1 3 1 9 0
> mode(nl)
[1] "numeric"
>
> cnl <- c(1,TRUE,"x")
> cnl
[1] "1"    "TRUE" "x"
> mode(cnl)
[1] "character"
```

Alternative Mode Testing Functions

- Instead of using `mode()`, you can also use:
  - `is.numeric()`
  - `is.logical()`
  - `is.character()`

```
> cnl
[1] "1"    "TRUE" "x"
> is.logical(cnl)
[1] FALSE
> is.character(cnl)
[1] TRUE
```

Functions as Objects

- Functions are treated as objects in R and have the mode `function`.

```
> mode(mean)
[1] "function"
> is.function(mean)
[1] TRUE
```

Class of an Object

- The class of an object determines what can be done with the object while the mode indicates how it is stored.
- The class is assessed through the `class()` function.

```
> class(cnl)
[1] "character"
> class(mean)
[1] "function"
```

Categorical Data as Factors

- Categorical data can be stored as numeric, character, or the special "factor" object mode.
- Storing categorical values as factor uses less storage and allows for statistical analysis.
- Use the `as.factor()` function to convert a character set to factors.

Example: Converting to Factor

```
> semesters <- c("Fall","Spring","Summer")
> mode(semesters)
[1] "character"
> semester.factor <- as.factor(semesters)
> mode(semester.factor)
[1] "numeric"
> semester.factor
[1] Fall   Spring Summer
Levels: Fall Spring Summer
> class(semester.factor)
[1] "factor"
```

Vectors

- A vector is any single value or any collection of values either numeric, logical, or character.
- The simplest way to create a vector is to use the `c()` function.

```
> # create a vector of numeric values
> num.vec <- c(1,3,5,8,13,21)
> num.vec
[1]  1  3  5  8 13 21
> mode(num.vec)
[1] "numeric"
> class(num.vec)
[1] "numeric"
> is.vector(num.vec)
[1] TRUE
```

Mixed Mode Vectors

- A vector with numeric and character values will be converted as a character vector.

```
> # mixed mode vector
> mixed.vec <- c(1,3,5,8,"thirteen")
> mixed.vec
[1] "1"        "3"        "5"        "8"
"thirteen"
> mode(mixed.vec)
[1] "character"
> is.vector(mixed.vec)
[1] TRUE
```

Combining Vectors

- Vectors can be combined into a single vector using the `c()` function.

```
> mixed.vec
[1] "1"        "3"        "5"        "8"        "thirteen"
> num.vec
[1]  1  3  5  8 13 21
> new.vec <- c(num.vec,mixed.vec)
> new.vec
 [1] "1"        "3"        "5"        "8"        "13"       "21"
 [7] "1"        "3"        "5"        "8"        "thirteen"
> mode(new.vec)
[1] "character"
```

Named Vectors

- The elements in the vector can be named.

```
> # create named vector
> named.vec <-
c(S1="Fall",S2="Spring",S3="Summer")
> named.vec
      S1        S2        S3
  "Fall" "Spring" "Summer"
```

Single Values

- Single values (scalars) are in fact vectors with a single element.

```
> x <- 99
> mode(x)
[1] "numeric"
> class(x)
[1] "numeric"
> is.vector(x)
[1] TRUE
```

Mode Conversion Functions

- While some conversion are automatic, specific conversions can be forced using the family of `as.`*`mode`*`()` functions:
  - `as.numeric()`
  - `as.logical()`
  - `as.character()`
  - `as.data.frame()`
  - `as.list()`
  - …

Conversion Example

- This example converts a vector of numeric values into a vector of logical values.
- Note that any non-zero value is TRUE while only 0 is FALSE.

```
> n.vec <- c(-99,0,99)
> l.vec <- as.logical(n.vec)
> n.vec
[1] -99   0  99
> l.vec
[1]  TRUE FALSE  TRUE
> mode(l.vec)
[1] "logical"
```

Missing Values

- During conversion, some values may not be convertible and are replaced with the special `NA` value.

```
> char.vec <- c("1","3","five","7")
> mode(char.vec)
[1] "character"
> char.vec
[1] "1"    "3"    "five" "7"
> num.vec <- as.numeric(char.vec)
Warning message:
NAs introduced by coercion
> num.vec
[1]  1  3 NA  7
```

Getting Help with `help()`

- To get help for any function use the `help()` function.

```
> help(seq)
```

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)
```

Arguments

`...`	arguments passed to or from methods.
`from, to`	the starting and (maximal) end values of the sequence.

Summary

- In this lesson, you learned that:
  - single and a collection of values are treated as vectors in R
  - each R object has a mode and a class
  - the mode determines the storage of the object
  - integers are 32-bit values
  - some mode conversions happen automatically while others can be forced

Summary, Review, & Questions…