R Programming: Week 1

Mike Talley

2/20/2022

Coercion of Different Data Types

R will not stop you from entering different data types into a vector, but when you go to operate on the vector, R will have to "coerce" the data into being a uniform data type. This inevitably can cause issues. One way to solve this is to use a list. Lists are a vector that can contain multiple data types, and are very important in R.

```
x <- 0:6    # Create a vector of numbers
class(x)
```

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

```
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```
as.logical(x)
```

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

```
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

```
## An example of coercion not working.
x <- c("a", "b", "c")
as.numeric(x)         #No way to make this work.
```

```
## Warning: NAs introduced by coercion
```

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

```
as.logical(x)
```

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

```
as.complex(x)
```

```
## Warning: NAs introduced by coercion
```

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

```
## A list
x <- list(1, "a", TRUE, 1 + 4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Binding Objects

We can bind objects or vectors using cbind() and rbind()

Factors

Factors are used to represent categorical data. They are essentially integers that have labels. It's often better to use a factor such as Male / Female, rather than an integer of 1 / 2. Below you'll see that R specifies the levels of this vector as "no" and "yes".

R will inherently use the first level alphabetically as the baseline level. If you want R to use a different level ("yes" for example) as the baseline, you need to specifically tell it that.

```
x <- factor(c("yes","yes","no","yes","no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

```
table(x)    #Gives a frequency count
```

```
## x
##  no yes
##   2   3
```

```
unclass(x)  #It will assign an integer to each value. Not necessary, but useful to know.
```

```
## [1] 2 2 1 2 1
## attr(,"levels")
## [1] "no"  "yes"
```

```
x <- factor(c("yes","yes","no","yes","no"),
            levels = c("yes", "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: yes no
```

Missing Values

is.na() is used to test objects to see if they are NA, while is.nan() will test for NaN. There can be both integer and character NA, so it still does have a class assigned to it. Also, NaN is a NA, but the converse is not always true. NaN are mathmatical values while NA is everything else.

Data Frames

These are most commonly used for storing tabular data. They CAN store different classes of objects, like a list. Be wary of coercion. They are also special in that they have an attribute called "row.names".

Reading in Data

read.table and read.csv are the two common ways of importing data. While most datasets do not require you to input many of the arguments, explicitly doing so can often help large datasets run faster. Using the colClasses argument can sometimes make the code run twice as fast.

Dputing and Dumping Data

A little more complicated. But dput and dump help create textual files to store the data. Can help prevent corruption down the road.

Connections

Can be useful for creating a connection to a certain file or object or location. A useful part of this is con <- url(), where you can connect to a webpage and read data from it.

Subsetting

[returns an object of the same class, but can select more than one element.

[[extracts a singular element, but the class may not be the same.

$ is used to extract elements of a list of df by name.

A benefit of [[is that it can extract computed objects, while $ can only select a literal name that exists in the list. It can also use sequences within it.

```
x <- list(foo = 1:4, bar = 0.6, baz = "hello")
name <- "foo"
x[[name]]     ## A computed index for 'foo'
```

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

```
x$name        ## 'name' is not in this list.
```

```
## NULL
```

```
x$foo         ##Call on foo explicitly.
```

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

```
x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
x[[c(1, 3)]]    ## Third object of first list.
```

```
## [1] 14
```

```
x[[1]][[3]]     ## Same
```

```
## [1] 14
```

```
x[[c(2,1)]]     ## First object of second list.
```

```
## [1] 3.14
```

You can subset a matrix by specifying row and column (where the element intersects both), or by specifying a row OR column and it will return a vector. It will usually return a value or vector, not a matrix, with no dimensions. If for some reason you wanted to preserve those dimensions, you would use the "drop" command, to tell it not to drop those dimensions.

```
x <- matrix(1:6, 2, 3)
x[1, ]
```

```
## [1] 1 3 5
```

```
x[1, drop = FALSE]
```

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

You can also use a partial match when subsetting, which can be useful for time saving if you are typing away rapidly. aardvark is an annoying word to type. To do this, you either need to use $, or to be very specific with [[.

```
x <- list(aardvark = 1:5)
x$a         ## Partial match
```

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

```
x[["a"]]    ## The double bracket doesn't natrually do a partial match
```

```
## NULL
```

```
x[["a", exact = FALSE]]
```

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

Removing Missing or NA Values

Most realistic data has a lot of missing values. Often times we need to remove those. A solution is to create a logical vector that tells you where the NAs are, allowing you to remove them via subsetting.

We can also use the complete.cases() function to help.

```
x <- c(1, 2, NA, 4, NA, 5)
bad <- is.na(x)
x[!bad]           ## Return elements that are not bad.
```

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

```
x <- c(1, 2, NA, 4, NA, 5)
y <- c("a", "b", NA, "d", NA, "f")
good <- complete.cases(x, y)
good
```

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

```
x[good]
```

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

```
y[good]
```

```
## [1] "a" "b" "d" "f"
```

```
airquality[1:6, ]   ## This has missing values
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```r
good <- complete.cases(airquality)
airquality[good, ][1:6, ]   ## Subset out missing values
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```