Two-Dimensional Data in R

PS239T

06 September, 2019

2D Data Structures in R

In this markdown file, we build off our new knowledge about one-dimensional data to create and alter new kinds of objects: matrices and dataframes.

1. Matrices introduces matrices, data structures for storing 2d data that is all the same class.
2. Dataframes teaches you about the dataframe, the most important data structure for storing data in R, because it stores different kinds of (2d) data.

1. Matrices

Matrices are created when we combine multiple vectors that all have the same class (e.g., numeric). This creates a dataset with rows and columns. By definition, if you want to combine multiple classes of vectors, you want a dataframe. You can coerce a matrix to become a dataframe, and vice-versa, but as with all vector coercions, the results can be unpredictable, so be sure you know how each variable (column) will convert.

```
m <- matrix(nrow = 2, ncol = 2)
m
```

```
##      [,1] [,2]
## [1,]   NA   NA
## [2,]   NA   NA
```

```
dim(m)
```

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

Matrices are filled column-wise.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```

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

Other ways to construct a matrix

```
m <- 1:10
dim(m) <- c(2, 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
dim(m) <- c(5, 2)
m
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

You can transpose a matrix (or dataframe) with `t()`

```
m <- 1:10
dim(m) <- c(2, 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
t(m)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

Another way is to bind columns or rows using `cbind()` and `rbind()`.

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
# or
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

You can also use the `byrow` argument to specify how the matrix is filled. From R's own documentation:

```
mdat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3, byrow = TRUE,
               dimnames = list(c("row1", "row2"),
                               c("C.1", "C.2", "C.3")))
mdat
```

```
##      C.1 C.2 C.3
## row1   1   2   3
## row2  11  12  13
```

Notice that we gave `names` to the dimentions in `mdat`.

```
dimnames(mdat)
```

```
## [[1]]
## [1] "row1" "row2"
##
## [[2]]
## [1] "C.1" "C.2" "C.3"
```

```
rownames(mdat)
```

```
## [1] "row1" "row2"
```

```
colnames(mdat)
```

```
## [1] "C.1" "C.2" "C.3"
```

2. Dataframes

A data frame is a very important data type in R. It's pretty much the **de facto** data structure for most tabular data and what we use for statistics.

2a. Creation

You create a data frame using `data.frame()`, which takes named vectors as input:

```
vec1 <- 1:3
vec2 <- c("a", "b", "c")
df <- data.frame(vec1, vec2)
df
```

```
##   vec1 vec2
## 1    1    a
## 2    2    b
## 3    3    c
```

```
str(df)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ vec1: int  1 2 3
##  $ vec2: Factor w/ 3 levels "a","b","c": 1 2 3
```

Beware: `data.frame()`'s default behaviour which turns strings into factors. Remember to use `stringAsFactors = FALSE` to suppress this behaviour as needed:

```
df <- data.frame(
  x = 1:3,
  y = c("a", "b", "c"),
  stringsAsFactors = FALSE)
str(df)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

In reality, we rarely type up our datasets ourselves, and certainly not in R. The most common way to make a data.frame is by calling a file using `read.csv` (which relies on the `foreign` package), `read.dta` (if you're using a Stata file), or some other kind of data file input.

2b. Structure and Attributes

Under the hood, a data frame is a list of equal-length vectors. This makes it a 2-dimensional structure, so it shares properties of both the matrix and the list.

```r
vec1 <- 1:3
vec2 <- c("a", "b", "c")
df <- data.frame(vec1, vec2)

str(df)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ vec1: int  1 2 3
##  $ vec2: Factor w/ 3 levels "a","b","c": 1 2 3
```

This means that a dataframe has `names()`, `colnames()`, and `rownames()`, although `names()` and `colnames()` are the same thing.

** Summary **

- Set column names: names() in data frame, colnames() in matrix
- Set row names: row.names() in data frame, rownames() in matrix

```r
vec1 <- 1:3
vec2 <- c("a", "b", "c")
df <- data.frame(vec1, vec2)

# these two are equivalent
names(df)
```

```
## [1] "vec1" "vec2"
```

```r
colnames(df)
```

```
## [1] "vec1" "vec2"
```

```r
# change the colnames
colnames(df) <- c("Number", "Character")
df
```

```
##   Number Character
## 1      1         a
## 2      2         b
## 3      3         c
```

```r
names(df) <- c("Number", "Character")
df
```

```
##   Number Character
## 1      1         a
## 2      2         b
## 3      3         c
```

```r
# change the rownames
rownames(df)
```

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

```r
rownames(df) <- c("donut", "pickle", "pretzel")
df
```

```
##         Number Character
## donut        1         a
## pickle       2         b
## pretzel      3         c
```

The `length()` of a dataframe is the length of the underlying list and so is the same as `ncol()`; `nrow()` gives the number of rows.

```r
vec1 <- 1:3
vec2 <- c("a", "b", "c")
df <- data.frame(vec1, vec2)

# these two are equivalent - number of columns
length(df)
```

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

```r
ncol(df)
```

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

```r
# get number of rows
nrow(df)
```

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

```r
# get number of both columns and rows
dim(df)
```

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

2c. Testing and coercion

To check if an object is a dataframe, use `class()` or test explicitly with `is.data.frame()`:

```r
class(df)
```

```
## [1] "data.frame"
```

```r
is.data.frame(df)
```

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

You can coerce an object to a dataframe with `as.data.frame()`:

- A vector will create a one-column dataframe.
- A list will create one column for each element; it's an error if they're not all the same length.
- A matrix will create a data frame with the same number of columns and rows as the matrix.

2d. Combining dataframes

You can combine dataframes using `cbind()` and `rbind()`:

```r
df <- data.frame(
  x = 1:3,
  y = c("a", "b", "c"),
  stringsAsFactors = FALSE)

cbind(df, data.frame(z = 3:1))
```

```
##   x y z
## 1 1 a 3
## 2 2 b 2
## 3 3 c 1
```

```r
rbind(df, data.frame(x = 10, y = "z"))
```

```
##    x y
## 1  1 a
## 2  2 b
## 3  3 c
## 4 10 z
```

When combining column-wise, the number of rows must match, but row names are ignored. When combining row-wise, both the number and names of columns must match. (If you want to combine rows that don't have the same columns, there are other functions / packages in R that can help.)

It's a common mistake to try and create a dataframe by `cbind()`ing vectors together. This doesn't work because `cbind()` will create a matrix unless one of the arguments is already a dataframe. Instead use `data.frame()` directly:

```r
bad <- (cbind(x = 1:2, y = c("a", "b")))
bad
```

```
##      x   y
## [1,] "1" "a"
## [2,] "2" "b"
```

```r
str(bad)
```

```
##  chr [1:2, 1:2] "1" "2" "a" "b"
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "x" "y"
```

```r
good <- data.frame(x = 1:2, y = c("a", "b"),
  stringsAsFactors = FALSE)
good
```

```
##   x y
## 1 1 a
## 2 2 b
```

```r
str(good)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ x: int  1 2
##  $ y: chr  "a" "b"
```

The conversion rules for `cbind()` are complicated and best avoided by ensuring all inputs are of the same type.

Exercises

1. Create a 3x2 data frame called `basket`. The first column should contain the names of 3 fruits. The second column should contain the price of those fruits.

```r
fruit = c("starfruit", "boysenberries", "lychee")
price = c(.89, .99, 1.00)

basket <- data.frame(_____, ______) # add your vectors here
class(basket)
```

```r
data.frame(fruit = c("starfruit", "boysenberries", "lychee"), price = c(.89, .99, 1.00))

basket

names(basket)
colnames(basket)
```

2. Now give your dataframe appropriate column and row names.

```r
names(basket) <- c("name", "price")
names(basket)
```

3. Add a third column called `color`, that tells me what color each fruit is.

```r
color = c("_______", "_______", "_______")

data.frame(basket, color)

cbind(basket, color)
```

Other objects

Missing values are specified with `NA`, which is a logical vector of length 1. `NA` will always be coerced to the correct type if used inside `c()`

```r
x <- c(NA, 1)
x
```

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

```r
typeof(NA)
```

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

```r
typeof(x)
```

```
## [1] "double"
```

`Inf` is infinity. You can have either positive or negative infinity.

```r
1/0
```

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

```r
1/Inf
```

```
## [1] 0
```

`NaN` means Not a number. It's an undefined value.

```r
0/0
```

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

3. Quiz

You can check your answers in answers.

1. What are the three properties of a vector, other than its contents?
2. What are the four common types of atomic vectors?

3. What are attributes? How do you get them and set them?
4. How is a list different from an atomic vector? How is a matrix different from a data frame?

Answers

1. The three properties of a vector are type (or class), length, and attributes.
2. The four common types of atomic vector are logical, integer, double (sometimes called numeric), and character. The two rarer types are complex and raw.
3. Attributes allow you to associate arbitrary additional metadata to any object. You can get and set individual attributes with `attr(x, "y")` and `attr(x, "y") <- value`; or get and set all attributes at once with `attributes()`.
4. The elements of a list can be any type (even a list); the elements of an atomic vector are all of the same type. Similarly, every element of a matrix must be the same type; in a data frame, the different columns can have different types.