Data Frames, Lists, and Indexing

Bio724D: Fall 2023

2023-09-11

Review: Vectors

Structure

- Homogeneous – all items in a vector are of the same type
- Ordered – each item has a position in the vector

Indexing

- For a vector of length n, the indices are $1 \dots n$ (1-indexing)
- Indexing occurs using single brackets `[]`

```
x <- c(2, 4, 6, -99, NA)
x[1]
x[length(x)]  # robust way to get last element
```

Question: What is the type of the “NA” item in the vector example above?

Data Frames

Data Frames represent tabular data. You can think of the columns of a data frame as an ordered collection of vectors.

Columns

- Columns of a data frame represent variables
- Columns must have names
- Every item in a given column is of the same data type
- Each column can have a different data type
- Each column must be of the same number of rows

Rows

- Rows represent observations/entities
- Each row is of the same length
- A row is heterogeneous collection – a data frame with a single observation

Constructing a data frame from scratch

```r
name <- c("Paul", "Maira", "Peter", "Beatriz")
grade <- c("C", "A", "B", "A-")
age <- c(40, 12, 17, 52)

example.df <- data.frame(name = name,
                         age = age,
                         grade = grade)
example.df
```

```
     name age grade
1    Paul  40     C
2   Maira  12     A
3   Peter  17     B
4 Beatriz  52    A-
```

Data frames: Detail

Shape

```
dim(example.df)   # number of rows and columns
nrow(example.df)  # number of rows
ncol(example.df)  # number of columns
```

Column names

```
names(example.df)
```

Data frames: Indexing by position

- Every element in a data frame is indexed by a row and a column position

```
example.df[3, 2]  # row, column
```

- Get a single column by integer position

```
example.df[2]
```

- Get multiple columns using a vector of indices

```
example.df[c(1,3)]
```

- Get a single row by integer position (not comma)

```
example.df[2,]  # row 2
```

- Multiple rows uing vector of indices

```
example.df[c(1,3), ]  # note comma
```

Data frames: Indexing rows and columns simultaneously

You can simultaneously index both rows and columns:

```r
example.df[c(1,3), c("name", "age")]
```

Data frames: Column name indexing

Indexing columns by name:

```r
example.df["grade"]
```

You can get multiple columns at a time by indexing with a vector of column names

```r
example.df[c("name","grade")]
```

$ operator

The $ operator followed by the name of a column **returns a vector** representing the values in the corresponding data frame column:

```
example.df$grade
```

- Note that when using the $ operator you don’t have to put the name of the column in quotes unless there are spaces in the name

Double bracket indexing

The columns of a data frame can be accessed by double bracket indexing:

```
example.df[[1]]
```

Like the `$` operator this returns a vector.

Boolean indexing

Both vectors and lists can be “Boolean indexed” – given an indexing vector of logical (`TRUE`/`FALSE`) values, Boolean indexing returns all the elements where the indexing vector is `TRUE`

- Vector example

```r
x <- c(1, 2, 3, 4)
x[c(TRUE, FALSE, FALSE, TRUE)]
```

```
[1] 1 4
```

- Data frame example

```r
example.df[c(TRUE,FALSE,TRUE,FALSE), ]
```

```
   name age grade
1  Paul  40     C
3 Peter  17     B
```

Boolean indexing, continued

Boolean indexing is often used to filter or subset data

- Example: subsetting the rows of a data frame

```
# get all rows of data frame where persons age > 18
is.adult <- example.df$age > 18
is.adult
```

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

```
example.df[is.adult, ]
```

```
     name age grade
1    Paul  40     C
4 Beatriz  52    A-
```

Usually we’d write the above example like so:

```
example.df[example.df$age > 18, ]
```

We’ll see a cleaner syntax and more example of Boolean indexing and filtering when we introduce the dplyr package

Lists

Lists are the most flexible built-in data structure in R.

- Unlike vectors and data frames which have constraints on what they contain and the size of the respective elements, lists can contain arbitrary objects of any type and size (even other lists)

```r
bob <- list('Bob', 16, 27707)

selena <- list('Selena', 'Montgomery', 17, 91324)

people <- list(bob, selena)

people
```

List elements can have names

```r
bob <- list(first_name="Bob",
            last_name="Gimli",
            age=16,
            zip=27707)
```

The names of list elements can be accessed with the `names()` function similar to the columns in a data frame

```r
names(bob)
```

Indexing Lists

- Single brackets always **return a list** containing the element at index `i`

```
bob[1]
typeof(bob[1])
```

- Use double brackets (`[[]]`)to return the element at index `i`

```
bob[[1]]
typeof(bob[[1]])
```

- If the list has named objects they can be accessed via the `$` operator

```
bob$last_name
```

- String indexing also works with lists

```
bob[c("last_name", "first_name")]
```

Question: What happens when you index with an integer index or a name that doesn’t exist?

Functions often return lists

Many R functions that need to return multiple values of different types will return lists (or things that act like lists).

Example

```
# draws a histogram but also return a list object
# with useful information about what was drawn
h <- hist(rnorm(100))
```

```
names(h)
## [1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"

# get break points for each bin and respective counts
h$breaks
## [1] -3 -2 -1  0  1  2  3
h$counts
## [1]  2 15 28 36 17  2
```

Setting values in data structures using indexing

The items in vectors, data frames, and lists can all be set or changed using the indexing operations described previously.

- Vector example

```r
v1 <- c(2, 4, 6, 8)
v1
## [1] 2 4 6 8
v1[3] <- -99
v1
## [1]   2   4 -99   8
```

Setting values in data structures using indexing, cont.

- Data frame example:

```
example.df
##      name age grade
## 1    Paul  40     C
## 2   Maira  12     A
## 3   Peter  17     B
## 4 Beatriz  52    A-
```

```
example.df$middle_initial <- c("M", "M", "B", "S")
example.df
##      name age grade middle_initial
## 1    Paul  40     C              M
## 2   Maira  12     A              M
## 3   Peter  17     B              B
## 4 Beatriz  52    A-              S
```