Data Structures and Subsetting

Completed by Leon Woltermann

Contents

NB: The worksheet has beed developed and prepared by Lincoln Mullen. Source: Lincoln A. Mullen, *Computational Historical Thinking: With Applications in R (2018–)*: http://dh-r.lincolnmullen.com.

The best way to learn R or computational history is to practice. These worksheets contain a series of questions designed to teach you about R or different computational methods. The worksheets are R Markdown documents that include text and code together. The places where you are expected to answer questions are marked like this.

`(@) Can you make a plot from this dataset?`

Beneath each question is a space to either create a code block or write an answer.

Aim of this worksheet

After completing this worksheet, you should have a basic familiarity with the most useful types of data structures in R (lists, data frames, and matrices). You should also be able to subset any of these data structures using the common subsetting operators (`[`, `[[`, and `$`).

You may find the chapters on data structures and subsetting from Hadley Wickham's *Advanced R* book to be helpful.

Subsetting vectors

In the worksheet on getting familiar with R you learned about vectors. Vectors hold a one-dimensional set of homogeneous values. By homogeneous we mean that that the values all have to be the same type (integer, numeric, logical, and so on) and you can't have both, say, numeric and logical values in the same vector. By one-dimensional we mean that the elements exist in order, and that they can be subset with a single value. For example, R includes a character vector of the letters in the English alphabet, helpfully called, `letters`.

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

The `letters` object is a character vector because everything inside of it is a character vector.

```
class(letters)
```

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

And it has twenty-six elements:

```
length(letters)
```

```
## [1] 26
```

We can **subset** a vector using the `[` operator (actually, it's a function). For instance, we can get the first element like this:

```
letters[1]
```

```
## [1] "a"
```

(1) How would you get the twenty-fifth element?

```
letters[25]
```

```
## [1] "y"
```

Note: In many programming languages counting starts with `0`; in `R`, however, counting starts with `1`.

We can get a range of values like so. Notice that we can get all the numbers one to five like this:

```
1:5
```

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

So, we can get the first five letters like this:

```
letters[1:5]
```

```
## [1] "a" "b" "c" "d" "e"
```

(2) Can you get the tenth through twelfth letters?

```
letters[10:12]
```

```
## [1] "j" "k" "l"
```

You can get an arbitrary subset by creating a numeric (or integer) vector. Here we get the first, tenth, and twelfth letters.

```
letters[c(1, 10, 12)]
```

```
## [1] "a" "j" "l"
```

We can also do this by creating a variable and using it to do the subsetting.

```
what_i_want <- c(1, 3, 5, 7)
letters[what_i_want]
```

```
## [1] "a" "c" "e" "g"
```

(3) Create a variable with the even numbers and then subset the `letters` variable to get the even letters (e.g., the second, fourth, etc.)

```
even_numbers <- c(2,4,6,8,10,12,14,16,18,20,22,24,26)
letters[even_numbers]
```

```
##  [1] "b" "d" "f" "h" "j" "l" "n" "p" "r" "t" "v" "x" "z"
```

(4) Bonus: can you use the `seq()` function (look it up with `?seq`) to get the even letters in a more clever way?

```
letters[seq(2,26,2)]
```

```
##  [1] "b" "d" "f" "h" "j" "l" "n" "p" "r" "t" "v" "x" "z"
```

In addition to values, vectors can also have names. For example, let's create a variable with the numbers 1 to 5, then give those values some names.

```
myvar <- 1:5
names(myvar) <- letters[1:5]
myvar
```

```
## a b c d e
## 1 2 3 4 5
```

Now we can also subset the vector based on the names:

```
myvar["c"]
```

```
## c
## 3
```

(5) Below is a vector of rankings for songs. Give the numeric vector names, which should be the titles of songs. Finally, subset the vector by the title of one of the songs to retrieve its ranking.

```
song_rankings <- c(10, 8.4, 6, 8.2, 4)
names(song_rankings) <- c("Dilemma", "Listen", "Californication", "Umbrella", "Summertime Sadness")
song_rankings["Californication"]
```

```
## Californication
##               6
```

Matrices

Vectors are one-dimensional and homogeneous. Matrices are two-dimensional and homogeneous, so they have the same kind of value, but have rows and columns as well. A matrix can be used for all kinds of problems in digital history. For now, let's imagine we have four cites, A, B, C, and D, and have measured the distances between them. For instance, the distance from A to B is 2. We can represent those distances as a matrix with 4 columns and 4 rows, where the names of the rows and columns are the cities.

```
city_distances <- matrix(c(0, 2, 8, 3, 2, 0, 6, 1, 8, 6, 0, 4, 3, 1, 4, 0),
                         nrow = 4, ncol = 4)
rownames(city_distances) <- LETTERS[1:4]
colnames(city_distances) <- LETTERS[1:4]
city_distances
```

```
##   A B C D
## A 0 2 8 3
## B 2 0 6 1
## C 8 6 0 4
## D 3 1 4 0
```

You can subset a matrix in the same way that you subset a vector. (Matrices *are* vectors—just vectors with two dimensions.) For instance, we can get the third element of the matrix.

```
city_distances[3]
```

```
## [1] 8
```

(6) Now get the fifth element of the matrix.

```
city_distances[5]
```

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

But matrices are more useful when we subset them by row and column. For instance, here is the value contained in the cell for the first row and third column.

```
city_distances[1, 3]
```

```
## [1] 8
```

(7) Now get the value for the third row and the first column.

```
city_distances[3,1]
```

```
## [1] 8
```

(8) What cities are we getting the distances for when we look for the third row and first column?

    Between city A and city C

If a matrix has row and column names, we can subset the vector by that. For instance, here is the distance between cities B and D.

```
city_distances["B", "D"]
```

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

(9) What is the distance between cities D and C?

```
city_distances["D", "C"]
```

```
## [1] 4
```

(10) What is the distance between city A and cities B, C, and D? (Hint: think back to the kinds of subsetting that we did with vectors.)

```
cities = LETTERS[2:4]
city_distances["A", cities]
```

```
## B C D
## 2 8 3
```

Data frames

The most useful data structure in R is the data frame. Think of a data frame like a spreadsheet that holds any kind of tabular data. It is two dimensional, like a matrix; unlike a matrix, it is heterogeneous in that the columns can hold different kinds of data. While other languages have add-on libraries that allow for data structures like this, in R data frames are a first class citizen.

Let's get a data frame from the `historydata` package. (If we also load `dplyr`, we will get nicer printing.)

```
library(historydata)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

We can use `str()` to get a different look at the data.

```
str(naval_promotions)
```

```
## tibble [5,705 x 5] (S3: tbl_df/tbl/data.frame)
##  $ id        : int [1:5705] 1 2 3 4 5 6 7 8 9 10 ...
##  $ name      : chr [1:5705] "Abbot, Joel" "Abbot, Trevett" "Abbott, Isaac" "Abbott, J. Francis" ...
##  $ generation: int [1:5705] 3 4 4 4 4 3 3 4 4 4 ...
##  $ rank      : Factor w/ 5 levels "midshipman","lieutenant",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ date      : chr [1:5705] "1812-06-18" "1848-10-13" "1820-05-10" "1837-12-27" ...
```

(11) How many rows and columns does the data frame have? What are the different types of vectors contained in it?

5 columns including id (integer), name (character), generation (integer), rank (factor), date (character) and 5705 rows.

We can use the `[` subset function to get access to rows and columns, just like we did with matrices. For instance, here we ask for just the first row and the columned named `"name"`:

```
naval_promotions[1, "name"]
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Abbot, Joel
```

We can also ask for the entire first row (note the comma):

```
naval_promotions[1, ]
```

```
## # A tibble: 1 x 5
##      id name        generation rank       date
##   <int> <chr>            <int> <fct>      <chr>
## 1     1 Abbot, Joel          3 midshipman 1812-06-18
```

(12) What is the name of the person in the tenth row? What was the date he became that rank?

```
naval_promotions[10,"name"]
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Abercrombie, J.B.
```

Because data frames are organized by column, it is possible to extract an entire column using the `$` function. Here, let's get just the names of the people. (We'll limit it using `head()` so we don't print out all of them.)

```
head(naval_promotions$name, 10)
```

```
##  [1] "Abbot, Joel"                "Abbot, Trevett"
##  [3] "Abbott, Isaac"              "Abbott, J. Francis"
##  [5] "Abbott, James W."           "Abbott, Thomas C."
##  [7] "Abbott, Walter"             "Abbott, William A."
##  [9] "Abercrombie, Alexander R." "Abercrombie, J.B."
```

The function `unique()` gives you only the unique values in a vector. For instance:

```
unique(c(1, 1, 1, 2, 3))
```

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

(13) What are all the different ranks contained in the `naval_promotions` dataset? (Hint: use `unique()`.)

```
unique(naval_promotions$rank)
```

```
## [1] midshipman        lieutenant        master_commandant captain
## [5] left_service
## Levels: midshipman lieutenant master_commandant captain left_service
```

(14) How many different people are contained in the `naval_promotions` dataset?

```
length(unique(naval_promotions$name))
```

```
## [1] 3354
```

(15) Why is that number different than the number of rows?

Maybe because some people have dublicate entries.

(16) What is the earliest and latest date in the dataset? (Hint: you may find some combination of `sort()`, `head()`, `tail()`, `range()`, and `as.Date()` to be useful. Don't forget about `na.rm = TRUE` as appropriate.)

```
earliest_latest <- range(sort(as.Date(naval_promotions$date)))
print(paste("The earliest date is", earliest_latest[1]))
```

```
## [1] "The earliest date is 1794-06-04"
```

```
print(paste("The latest date is", earliest_latest[2]))
```

```
## [1] "The latest date is 1905-02-12"
```

We will work much, much more with data frames.

Lists

Another very useful kind of data structure is the list. A list can hold values of any type, including vectors, data frames, and even other lists. For instance, we can create a list that holds several different kinds of information:

```
our_class <- list(
  title = "Intro to R",
  year = 2016,
  books = c("Basics of R", "Get Awesome at R"),
  students = c("Adam", "Betsy", "Cynthia", "David")
)
str(our_class)
```

```
## List of 4
##  $ title   : chr "Intro to R"
##  $ year    : num 2016
##  $ books   : chr [1:2] "Basics of R" "Get Awesome at R"
##  $ students: chr [1:4] "Adam" "Betsy" "Cynthia" "David"
```

We can get just part of the list using the `[` function that we've become used to. For instance, to get just the title:

```
our_class["title"]
```

```
## $title
## [1] "Intro to R"
```

But notice that the returned value is a list, not a character vector

```
is.list(our_class["title"])
```

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

```
is.character(our_class["title"])
```

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

R has another subset operator, `[[`. The single bracket (`[`) gives us what we asked for inside a list; the double bracket (`[[`) simplifies the list to give us the vector (or whatever) we asked for.

```
our_class[["title"]]
```

```
## [1] "Intro to R"
```

```
is.list(our_class[["title"]])
```

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

```
is.character(our_class[["title"]])
```

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

(17) Using the `our_class` list, get just the year (as a list). Get it as a numeric vector.

```
our_class["year"]
```

```
## $year
## [1] 2016
```

```
our_class[["year"]]
```

```
## [1] 2016
```

(18) Using the `our_class` list, get the class title, the book list, and the year (as a list). (Hint: remember the different kinds of subsetting we did with `[`.)

```
our_class[["title"]]
```

```
## [1] "Intro to R"
```

```
our_class[["books"]]
```

```
## [1] "Basics of R"      "Get Awesome at R"
```

```
our_class["year"]
```

```
## $year
## [1] 2016
```

R also lets us use the `$` operator we used to get columns of a data frame.

```
our_class$title
```

```
## [1] "Intro to R"
```

(19) Get the students vector from `our_class`:

```
our_class[["students"]]
```

```
## [1] "Adam"    "Betsy"   "Cynthia" "David"
```

(20) Is the `$` equivalent to `[` or `[[`?

$ is equal to [[because it returs a vector and not a list

(21) Create a list that models a historic event. What parts of the event are worth keeping track of? Also extract certain parts of that list.

```
election <- list(
  title="First General Election",
  country="Indonesia",
  year="1955",
  parties= c("PNI", "NU", "PKI", "Masyumi"),
  percentages= c(22, 18, 16, 20)
)
```

(22) You can use `$`, `[`, and `[[` on both data frames and lists. What is the relationship between a data frame and a list? (Hint: use `is.list()` and `is.data.frame()` on a list and a data frame.)

```
is.list(election)
```

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

```
is.data.frame(election)
```

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

```
is.list(naval_promotions)
```

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

```
is.data.frame(naval_promotions)
```

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

Subsetting with logical vectors

We have done subsetting above using numeric and character vectors. We can also do subsetting using logical vectors. Let's create a sample dataset of the heights of soldiers.

```
set.seed(3929)
heights <- rnorm(20, mean = 170)
names(heights) <- letters[1:20]
heights
```

```
##        a        b        c        d        e        f        g        h 
## 168.8234 170.0480 170.5908 169.8695 169.5483 169.1309 169.8232 168.9089 
##        i        j        k        l        m        n        o        p 
## 172.8877 167.5170 169.5843 169.2370 171.3657 169.3305 170.4878 170.8695 
##        q        r        s        t 
## 170.7869 169.9028 169.8254 170.4796
```

We can find all the soldiers who are taller than average like this. First let's compare all the heights to the mean height.

```
heights > mean(heights)
```

```
##     a     b     c     d     e     f     g     h     i     j     k     l     m 
## FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE 
##     n     o     p     q     r     s     t 
## FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

Notice that we get a logical vector as a result. We can use that within the `[` operator to get just the soldiers who are taller than average.

```
heights[heights > mean(heights)]
```

```
##        b        c        i        m        o        p        q        t 
## 170.0480 170.5908 172.8877 171.3657 170.4878 170.8695 170.7869 170.4796
```

(23) Which soldiers are taller than 170 cm?

```
heights[heights > 170]
```

```
##        b        c        i        m        o        p        q        t 
## 170.0480 170.5908 172.8877 171.3657 170.4878 170.8695 170.7869 170.4796
```

This kind of subsetting also works for data frames. Here we get all the officers from the first generation. (Notice the comma.)

```
first_gen <- naval_promotions[naval_promotions$generation == 1, ]
head(first_gen, 10)
```

```
## # A tibble: 10 x 5
##       id name                generation rank       date      
##    <int> <chr>                    <int> <fct>      <chr>     
##  1    76 Archer, John                 1 lieutenant 1798-11-08
##  2   117 Bainbridge, William          1 lieutenant 1798-08-03
##  3   127 Baker, Thomas                1 lieutenant 1798-05-25
##  4   141 Ballard, John                1 lieutenant 1798-10-02
##  5   168 Barron, James                1 lieutenant 1798-03-09
##  6   180 Barton, Jeremiah             1 lieutenant 1798-06-08
##  7   259 Blair, George                1 lieutenant 1799-03-13
##  8   434 Burns, James                 1 lieutenant 1798-10-29
##  9   453 Byrne, Gerald                1 lieutenant 1799-06-17
## 10   474 Campbell, James              1 lieutenant 1799-09-20
```

(24) Can you get just the promotions to captain?

```
captain <- naval_promotions[naval_promotions$rank == "captain", ]
head(captain, 10)
```

```
## # A tibble: 10 x 5
##       id name                  generation rank    date      
##    <int> <chr>                      <int> <fct>   <chr>     
##  1     1 Abbot, Joel                    3 captain 1850-10-03
##  2    14 Adams, Henry A.                3 captain 1855-09-14
##  3    15 Adams, Henry A., Jr.           4 captain 1877-03-28
##  4    29 Alden, James                   4 captain 1863-01-02
##  5    56 Almy, John J.                  4 captain 1865-03-03
##  6    58 Ammen, Daniel                  4 captain 1866-07-25
##  7    73 Angus, Samuel                  2 captain 1816-04-27
##  8    82 Armstrong, James               3 captain 1841-09-08
##  9    84 Armstrong, James F.            4 captain 1866-09-27
## 10    85 Armstrong, William M.          3 captain 1855-03-24
```

(25) Can you get the promotions from 1800? (Hint: you will first have to make the date column a date object. Try looking up the documentation for `?lubridate::ymd` and `?lubridate::year`.)

```
naval_promotions_temp <- naval_promotions
naval_promotions_temp$date <- lubridate::ymd(naval_promotions_temp$date)
```

```
## Warning: 2 failed to parse.
```

```
naval_promotions_temp$date <- lubridate::year(naval_promotions_temp$date)
naval_promotions_temp[naval_promotions_temp$date == 1800,]
```

```
## # A tibble: 190 x 5
##       id name                  generation rank        date
##    <int> <chr>                      <int> <fct>      <dbl>
##  1    31 "Aldrick, Samuel"              2 midshipman  1800
##  2    45 "Allen, Samuel "               2 midshipman  1800
##  3    47 "Allen, William H."            2 midshipman  1800
##  4    66 "Anderson, Thomas O."          2 midshipman  1800
##  5    72 "Angier, Charles"              2 midshipman  1800
##  6   154 "Barker, William W."           2 midshipman  1800
##  7   195 "Beale, Thomas T."             2 midshipman  1800
##  8   203 "Beck, Daniel"                 2 midshipman  1800
##  9   227 "Bennett, Edward"              2 midshipman  1800
## 10   233 "Bentley, Peter E."            2 midshipman  1800
## # ... with 180 more rows
```