Basic Course on R: Manipulating Data Practical Answers

Karl Brand* and Elizabeth Ribble†

18-24 May 2017

Contents

*brandk@gmail.com

†emcclel3@msudenver.edu

1 Manipulating / Selecting Data

1.1 Answer the following without typing the commands into R. Use ? if you're not sure what the object is or what the function does.

1.1.1 What is

```
length(letters)
```

26

1.1.2 What is

```
length(letters==LETTERS)
```

26

1.1.3 What is

```
which(letters %in% c("a", "d"))
```

1, 4

1.1.4 What is

```
which(c("a", 7, "d") %in% letters)
```

1, 3

1.1.5 What is

```
letters[LETTERS > "W"]
```

x, y, z

1.1.6 What is

```
letters[!LETTERS > "C"]
```

a, b, c

1.1.7 What is

```
seq(1, 20, 3)
```

1, 4, 7, 10, 13, 16, 19

1.1.8 Why is `x` filled in the way it is? Hint: read about the arguments for `matrix`!

```
x <- matrix(8:11, nrow = 6, ncol = 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    8   10    8   10
## [2,]    9   11    9   11
## [3,]   10    8   10    8
## [4,]   11    9   11    9
## [5,]    8   10    8   10
## [6,]    9   11    9   11
```

`x` looks this way because we defined the number of rows to be 6 and the number of columns to be 4, the default `byrow` is FALSE so the array is filled columnwise, and the numbers `8:11` are repeated until the array is filled.

1.1.9 What are

```
x + 4
x + x
2 * x
x/c(2, 3, 4, 5)
x[, 3] + 2*x[, 2]
nrow(x)
x[x[, 3]>10, ]
```

- 4 added to each value of `x`
- 2 multiplied by each value of `x`
- 2 multiplied by each value of `x`
- columns of x are divided by the vector c(2, 3, 4, 5); the vector is recycled so that the first and third columns are still the same and the second and fourth columns are still the same
- a single vector with values from the third column of x added to two times the second column of x

- 6
- the 4th row of x, since it's the only row where the third column is greater than 10

1.2 Use R to answer the following.

1.2.1 Create a vector (using `c()`) called `a` (i.e. assign it to an object called `a`) with four elements which are the integers 5 to 8 (inclusive).

```
a <- 5:8
```

1.2.2 Display element 2 of `a`.

```
a[2]

## [1] 6
```

1.2.3 Display element 4 of `a`.

```
a[4]

## [1] 8
```

1.2.4 Calculate the product of elements 2 and 4 of `a`.

```
a[2] * a[4]

## [1] 48
```

1.2.5 Assign the integers 3 and 4 to object `b` and use `b` to select elements 3 and 4 of object `a`.

```
b <- c(3,4)
# or
b <- 3:4
a[b]

## [1] 7 8
```

1.2.6 Display every element of `a` except element 2.

```
a[-2]

## [1] 5 7 8
```

1.2.7 Display every element of `a` except elements 3 and 4.

```
a[-c(3:4)]

## [1] 5 6
```

1.2.8 Display only those elements of `a` that are greater than or equal to 6.

```
a[a>=6]

## [1] 6 7 8
```

1.2.9 Display only those elements of `a` that are not equal to 7.

```
a[a!=7]

## [1] 5 6 8
```

1.2.10 Use the `list` function to create an object `ab` which is a list of the two objects `a` and `b`.

```
ab <- list(a, b)
```

1.2.11 Display `ab`.

```
ab

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

1.2.12 Change the names of the elements in `ab` to "a" and "b".

```
names(ab) <- c("a", "b")
```

1.2.13 Display `ab` again. What has changed?

```
ab


## $a
## [1] 5 6 7 8
##
## $b
## [1] 3 4
```

Displaying `ab` now shows `$a` and `$b` in place of the original `[[1]]` and `[[2]]`. This indicates how to select and subset the list when we have names versus having no names.

1.2.14 Create this matrix `m`:

```
m <- matrix(1:9, nrow=3, byrow=T)
m


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

Why are the numbers 1, 2, and 3 in the first row and not the first column?

Because we set the argument `byrow` equal to TRUE.

1.2.15 Display the element on the second row and second column of `m`.

```
m[2, 2]


## [1] 5
```

1.2.16 Display only the 2nd row of `m`.

```
m[2, ]


## [1] 4 5 6
```

1.2.17 Display only the 3rd column of `m`.

```
m[, 3]

## [1] 3 6 9
```

1.2.18 Display only the 2nd and 3rd columns of `m`. Do so in two different ways.

```
m[, 2:3]

##      [,1] [,2]
## [1,]    2    3
## [2,]    5    6
## [3,]    8    9

m[, -1]

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

If you want to save your work: save your R session and/or source code!