Contents

1 Introduction to R

Generally speaking, R was built for statistical analysis, with many built-in functions. Similar functionality in Python must be imported from another library.

In future weeks, we will be seeing more probability distribution related functions.

2 The `cat()` and `print()` functions

The `cat()` function (short for `concatenate`) and the `print()` function can be used to display information to the user. Both, however, work slightly differently.

We will not frequently use the `cat()` or `print()` functions, because we generally see the output immediately.

```r
cat('abc')
print('xyz')
```

3 Assignment operator (<− or =)

<− and = can both be used to assign values to variables in R. For consistency I will just use <−.

```r
x = 1
y <- 2
cat('x has the value', x, ', y has the value', y)
```

```
## x has the value 1 , y has the value 2
```

4 Vectors (`c()`)

Vectors are created using the `c()` function. When a range of consecutive numbers is desired, we can also use `start:`end to get a vector of all numbers between `start` and end, inclusive of both ends.

```r
v1 <- c(1, 3, 5, 7, 9, 11)
v2 <- 1:6*2   # create the vector (1, 2, 3, 4, 5, 6) and multiply each
↪  element by 2.
cat('v1 has the value', v1, ', v2 has the value', v2)
```

```
## v1 has the value 1 3 5 7 9 11 , v2 has the value 2 4 6 8 10 12
```

You can see the length of a vector using `length()`:

```r
cat('v1 has length', length(v1))
```

```
## v1 has length 6
```

5 `seq()` and `rep()`

`seq()` and `rep()` are two functions to form vectors.

There is not much to say about `seq()` - its most basic use is to form numerical vectors. It is short for `seq`uence generation. A similar function in Python is `range()`, but they are not precisely the same. The ending number here is always included, and the step size (and thus the resulting vector) can be a float (e.g. `1.3`).

```r
# Basic usage: seq(from, to, by)
seq(5, 14, 3)
```

```
## [1]  5  8 11 14
```

```r
# There is also the length.out parameter
seq(5, 14, length.out=11)
```

```
##  [1]  5.0  5.9  6.8  7.7  8.6  9.5 10.4 11.3 12.2 13.1 14.0
```

`rep()` on the other hand, `rep`eats things passed to it. It can be passed *vectors* to be repeated, as well as the number of times to be repeated. The number of elements in x must match the number of elements in `times`.

```r
# Basic usage: rep(x, times)
rep(12, 3)
```

```
## [1] 12 12 12
```

```r
rep(10:13, 3)
```

```
##  [1] 10 11 12 13 10 11 12 13 10 11 12 13
```

```r
rep(11:14, 1:4)
```

```
##  [1] 11 12 12 13 13 13 14 14 14 14
```

You can, of course, compose the two, and repeat more than just numbers.

```r
rep(c("a", "b", "c"), seq(1, 5, 2))
```

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

6 Matrices

Matrices in R are similar to matrices in math. They can be created in several ways.

6.1 `dim()`

`dim()` tells us the dimension of an object. A vector has no dimension, only length. We can assign a dimension to a vector to make it a matrix.

```r
dim(v1)
```

```
## NULL
```

```r
dim(v1) <- c(3, 2)
v1      # now a 3x2 matrix
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    3    9
## [3,]    5   11
```

6.2 `matrix()`

We can create matrices directly with `matrix()`. Specify the parameter(s) `ncol`, `nrow`, and/or `byrow` to determine how the matrix should be created.

```r
matrix(1:10, nrow = 2)
```

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

```
matrix(1:10, ncol = 2)
```

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

```
matrix(1:10, ncol = 5, byrow=T)
```

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

6.3 `rbind()` and `cbind()`

Matrices can also be made using `rbind()` and `cbind()` on vectors or other matrices.

```
a <- matrix(1:10, nrow = 2)
b <- matrix(11:20, nrow = 2)
cbind(a, b)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    3    5    7    9   11   13   15   17    19
## [2,]    2    4    6    8   10   12   14   16   18    20
```

```
rbind(a, b)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
## [3,]   11   13   15   17   19
## [4,]   12   14   16   18   20
```

```r
rbind(1:5, 6:10)
```

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

6.4 Getting help (? and `help()`)

Many functions in R have documentation on them, i.e. what the function does, how the function is used, the parameters it accepts. If you ever forget what a function does or how to use it, in the R console you can use the ? operator or the `help()` function.

For example, typing `?matrix` or `help(matrix)` *without the parentheses* will bring up the documentation page for the `matrix()` function.

7 Data frames (`data.frame`)

Matrices are limited to having data all of the same type (e.g. number, text, boolean) Data frames are just matrices, which allow for a mix of data types. They can be created with the `data.frame()` function on any matrix or vector.

```r
df1 <- data.frame(1:5)
df1
```

```
## # A tibble: 5 x 1
##    X1.5
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5
```

```r
a
```

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

```
df2 <- data.frame(a)
df2
```

```
## # A tibble: 2 x 5
##      X1    X2    X3    X4    X5
##   <int> <int> <int> <int> <int>
## 1     1     3     5     7     9
## 2     2     4     6     8    10
```

The rows are individual observations, the columns are the data of each observation. By default, the columns are assigned the names `X1, X2, ..., Xn` for each column, and the rows are assigned numerical labels. We can see the column names, and assign new names using the `names()` function.

```
names(df2)
```

```
## [1] "X1" "X2" "X3" "X4" "X5"
```

```
names(df2) <- c('age', 'height', 'weight', 'grade', 'money')
df2
```

```
## # A tibble: 2 x 5
##     age height weight grade money
##   <int>  <int>  <int> <int> <int>
## 1     1      3      5     7     9
## 2     2      4      6     8    10
```

We can also change the row names if desired using `row.names()`.

```
row.names(df2)
```

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

```
row.names(df2) <- c('caleb', 'you')
df2
```

```
## # A tibble: 2 x 5
##     age height weight grade money
##   <int>  <int>  <int> <int> <int>
## 1     1      3      5     7     9
## 2     2      4      6     8    10
```

Now, to describe this data frame, I will say that this data frame has 2 observations (rows), and 5 data points (columns) for each observation. The interpretation of this data frame with these row and column names should come naturally.

In this course, in general, we will not be creating data frames or matrices directly. Instead, most data will be imported using a `read` function, which we will see in the lab, or imported from another library, which we will see in a few weeks.

8 Indexing vectors, matrices, and data frames

To get a single value in a vector, matrix, or data frame, we can use indexing with square brackets `[]`. Indexing in R is extremely similar to indexing in Python, but unlike Python, which is 0-indexed, R is 1-indexed, so the first entry in a list always has index 1. Vectors in R are analogous to lists in Python (that do not contain other lists), and require only one indexing.

```
x <- 1:5*10
x
```

```
## [1] 10 20 30 40 50
```

```
x[2]            # Get the second element
```

```
## [1] 20
```

For matrices and data frames, as they have two dimensions (rows and columns), we have the option to get rows, columns, or individual cells. The square bracket notation is the same, but we now have the option to index along the rows or columns or both. Take note of the below examples, noting the the presence of the commas and blank entries.

```
df2['age']      # with no comma, selects columns, return as *data frame*.
```

```
## # A tibble: 2 x 1
##     age
##   <int>
## 1     1
## 2     2
```

```
                # or with comma, first is row, second is column.
df2[, 'age']    # selects all rows, and only column 'age', return as
↪  *vector*.
```

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

```
df2['caleb',]   # selects only row 'caleb', and all columns, return as
↪  *data frame*.
```

```
## # A tibble: 1 x 5
##     age height weight grade money
##   <int>  <int>  <int> <int> <int>
## 1     1      3      5     7     9
```

In general, the indexing is done as `thing_to_index[rows, columns]`. Columns in data frames can also be accessed using a $ operator.

```
df2$age         # equivalent to df2[, 'age'], return as vector.
```

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

To select multiple rows or multiple columns, you can pass in vectors as indexes. Selections can also be made using numerical indices. You can see that the below have the exact same output as using the named indices above.

```
df2[1:3]
```

```
## # A tibble: 2 x 3
##     age height weight
##   <int>  <int>  <int>
## 1     1      3      5
## 2     2      4      6
```

```
df2[, c(2, 5)]
```

```
## # A tibble: 2 x 2
##   height money
##    <int> <int>
## 1      3     9
## 2      4    10
```

```
df2[1, ]
```

```
## # A tibble: 1 x 5
##     age height weight grade money
##   <int>  <int>  <int> <int> <int>
## 1     1      3      5     7     9
```

Indexing for vectors and matrices is extremely similar. As an exercise, index the following values from the below matrix, using the `c()` function if necessary.

```r
m1 <- matrix(1:20, nrow=4, byrow=T)
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

```r
# Index the following from the matrix m1:
# The entry 13
# The entries 1, 2, 3, 4, and 5.
# The entries 4, 9, 14, and 19.
# The entries 2, 7.
# The entries 1, 3, 16, and 18.

# For example, to index the entry 12
# which is the 3rd row, 2nd column.
m1[3, 2]
```

```
## [1] 12
```