Part 6: Lists

Introduction to R

This assignment consists of six parts:

- Intro to Basics
- Vectors
- Matrices
- Factors
- Data frames
- Lists (this document)

Create a script called `hw02-6.R` and save it in your `hw02` folder.

After you complete each exercise, commit and push your R script to your remote repo. See Part 0 for instructions. Do *not* push this document.

6.1 Introduction to Lists

Lists are the most complex type of data object in R. You also will not use them much in this course so this exercise will be relatively brief.

A list in R is similar to a list of chores you have to do around the house. You may have to mow the lawn, wash and put away the dishes, and feed your pet. Each item on the list is unrelated to the others but they fit together logically on a list of chores you have to do.

In R, a list collects objects of different types into a single object. The collected objects can include vectors, matrices, data frames, and even lists. In fact, lists are used for the output of many R functions. The output contains information that you can extract with techniques similar to those you learned for vectors and data frames.

6.2 Create a list

The `list()` function in R creates a list from objects that you pass to the function. You can and should name each object as you pass it to the `list()` function. In the example below, `object1` will be named `name1`, `object2` will be named `name2`, and so on, as shown in the output.

```
object1 <- c(1:10)
object2 <- letters[1:6]
my_list <- list(name1 = object1, name2 = object2) #Repeat for each object

my_list
```

```
## $name1
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $name2
## [1] "a" "b" "c" "d" "e" "f"
```

Notice that the name of each object has `$` prefixed to it, followed by the contents of the object.

6.2 Instructions

Add the following code to your script.

- Create the four objects shown below: a numeric vector, a logical vector, a matrix, and a data frame. Use the code as shown.
- Use the `list()` function to create `the_list` with the objects `numeric_vector`, `logical_vector`, `letter_matrix`, and `chicks_df`. Use the following names for each object in the `list()` function.
  - Use the name `numbers` for `numeric_vector`
  - Use the name `boolean` for `logical_vector`
  - Use the name `letters` for `letter_matrix`
  - Use the name `chicks` for `chicks_df`
- Enter `the_list` on a line by itself to display the contents.
- Use the `str()` function to display the structure of the list.

```
# Numeric vector with numerics from 1 to 6
numeric_vector <- 1:6

# Logical vector with TRUE and FALSE
# The `rep` function replicates objects you pass to it.
# In this case, you're replicating TRUE FALSE four times.
logical_vector <- rep(c(TRUE, FALSE), 4)

# letter Matrix with the first nine letters of the English alphabet
letter_matrix <- matrix(LETTERS[1:9], ncol = 3)

# First 10 elements of the chickwts data frame
chicks_df <- chickwts[1:10,]

# Use the `list()` function to create `the_list` with the above objects. Use the names of the objects t


# Display the contents of `the_list`



#
```

If correct, the contents of `the_list` should like this this:

```
## $numbers
## [1] 1 2 3 4 5 6
##
## $boolean
## [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##
## $letters
##      [,1] [,2] [,3]
## [1,] "A"  "D"  "G"
## [2,] "B"  "E"  "H"
## [3,] "C"  "F"  "I"
##
```

```
## $chicks
##    weight      feed
## 1     179 horsebean
## 2     160 horsebean
## 3     136 horsebean
## 4     227 horsebean
## 5     217 horsebean
## 6     168 horsebean
## 7     108 horsebean
## 8     124 horsebean
## 9     143 horsebean
## 10    140 horsebean
```

The struction should look like this:

```
## List of 4
##  $ numbers: int [1:6] 1 2 3 4 5 6
##  $ boolean: logi [1:8] TRUE FALSE TRUE FALSE TRUE FALSE ...
##  $ letters: chr [1:3, 1:3] "A" "B" "C" "D" ...
##  $ chicks :'data.frame': 10 obs. of  2 variables:
##   ..$ weight: num [1:10] 179 160 136 227 217 168 108 124 143 140
##   ..$ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2
```

6.3 Extract elements from a list

You are by now familiar with using square brackets `[]` to extract items from vectors, matrices, and data frames. Lists are similar but with an important difference. In lists, `[]` returns the object name *and* the contents of the object. A pair of square brackets `[[]]` returns just the contents of the object.

Look at the structure for `my_list`, the example list made above.

```
## List of 2
##  $ name1: int [1:10] 1 2 3 4 5 6 7 8 9 10
##  $ name2: chr [1:6] "a" "b" "c" "d" ...
```

If exact the first object in the list (`name1`) using a single pair of square brackets (`[]`), you get the object name *and* the contents. In most cases, you will need *only* the contents so use two pairs of square brackets `[[]]`. `name1` is included with the single brackets but not with two pairs, as shown with this example.

```
# Extract the first object with one pair of brackets
my_list[1]

# Extract the first object with two pairs of brachets
my_list[[1]]

# The same applies if you use the name of the object in backets.
# Here, extract the second object.
my_list["name2"]
my_list[["name2"]]
```

```
## $name1
##  [1]  1  2  3  4  5  6  7  8  9 10
##
##  [1]  1  2  3  4  5  6  7  8  9 10
## $name2
## [1] "a" "b" "c" "d" "e" "f"
```

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

Recall that the object name is prefixed with a `$`. Recall too that you can use the `$` to extract a variable from a data frame with the `$` symbol. You can do the same with a list.

```
my_list$name1
```

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

Typing the `$` and the object name saves typing: `$name1` versus `[["name"]]`. *If the items in a list are not named, you have to use the pairs of square brackets.*

Often, you need one element from an object in the list. You use combinations of `$` and square brackets, as necessary. Look at the structure of `the_list` you made with your code above.

```
str(the_list)
```

```
## List of 4
##  $ numbers: int [1:6] 1 2 3 4 5 6
##  $ boolean: logi [1:8] TRUE FALSE TRUE FALSE TRUE FALSE ...
##  $ letters: chr [1:3, 1:3] "A" "B" "C" "D" ...
##  $ chicks :'data.frame': 10 obs. of  2 variables:
##   ..$ weight: num [1:10] 179 160 136 227 217 168 108 124 143 140
##   ..$ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2
```

If you need the letter "D" for example from the `letters` object, you could do this one of three ways.

- `the_list[[3]][4]` `# Get fourth element from third object`
- `the_list[["letters"]][4]` `# Get fourth element from named object`
- `the_list$letters[4]` `# Use the dollar sign`

The third method is clear and requires less typing. **Work smart, not hard.**

Look again at the structure of `the_list`. How would you get the fifth element from the `weight` column of the `chicks` data frame, which is the fourth object in the list?

Remember that you can use the `$` to exact inforamtion from data frames and you use the `$` to extracte named objects from lists. You can use `$` multiple times to get the information you need.

```
# Get the fifth element from the weights column of the chicks object.
the_list$chicks$weight[5]
```

```
## [1] 217
```

This will be the primary way you will use lists in this course so get comfortable with these methods to get the elements you need from a list.

6.3 Instructions

I said above that many functions create lists. One such function is `t.test()`, which performs a statistical comparison of the means of two groups. You'll compare the average weights of chicks raised on two different types of feed. The two values most often reported from a t-test are the "t" statistic and the probability ("p value") that the null hypothesis is true (i.e., the two groups have the same mean).

Add code to your script to do the following. The first two lines create the data vectors you will compare. The third line runs the t-test. Do not worry if you don't understand right now what the t-test is doing (although you should understand if you took BI 163).

- Add the line `hb_chicks <- chickwts$weight[1:10]`. This vector holds weights of 10 chicks raised on horsebean.

- Add the line `ls_chicks <- chickwts$weight[11:20]`. This vector holds weights of 10 chicks raised on linseed.
- Add this line to create a list object with the results of the t-test. `chicks_t <- t.test(hb_chicks, ls_chicks, var.equal = TRUE)`
- Run the `str()` function on `chicks_t` to see the structure of the list that is returned from the analysis. The first line should say `list of 10`.
- Use the most efficient method to display the numeric value contained in the `statistic` object. The `statistic` is the t-value.
- Use the most efficient method to display the numeric `p.value`. This is the probability that the null hypothesis is supported.

It is also common to display the upper and lower 95% confidence limits of the difference between the two means. The `conf.int` object is a vector with two values. The first value is the lower limit. The second value is the upper limit.

- Add code to display the lower and upper confidence limits on two separate lines. In other words, extract and display the individual elements, not the vector.

```
# Add the line to create the horsebean vector
hb_chicks <- chickwts$weight[1:10]

# Add the line to create the linseed vector
ls_chicks <- chickwts$weight[11:20]

# Create a list object with the results of the t-test.
chicks_t <- t.test(hb_chicks, ls_chicks, var.equal = TRUE)

# Run the `str()` function on `chicks_t`


# Display the numeric value of the `statistic` object from `chicks_t`


# Display the numeric value of the `p.value` object from `chicks_t`


# Display the lower confidence limit


# Display the upper confidence limit


# Create a text string called `Whew` with the value "I did it!"

#
```