Lists and Iterations

Evan Bowman

2023-03-24

Working with Lists

Because of their generality, lists (or list-like objects) are often the output of many statistical procedures in R. The file `fpout.RDS` in the data folder contains sample output from using fitPoly, a statistical method to quantify properties of locations on the genome.

1. Use `read_rds()` and a relative path to read this data into R.

```
fitPoly <- read_rds("../data/fpout.RDS")
```

2. Use appropriate functions to obtain the length of the list and then the vector of the names of the elements in the list.

```
# Length of List
length(fitPoly)
```

```
## [1] 7
```

```
# Vector of names of the elements
names(fitPoly)
```

```
## [1] "log"          "modeldata"    "allmodeldata" "scores"       "diploscores"
## [6] "time"         "prop_miss"
```

3. The `diploscores` element does not provide any information. Remove it from the list in one step.

```
fitPoly <- fitPoly[-5]

names(fitPoly)
```

```
## [1] "log"          "modeldata"    "allmodeldata" "scores"       "time"
## [6] "prop_miss"
```

4. The `scores` element contains the output most users would want. The variables in `scores` called `P0`, `P1`, `P2`, `P3`, `P4`, `P5`, and `P6` contain "posterior probabilities" for each individual for values `0`, `1`, `2`, `3`, `4`, `5`, and `6` (respectively).

- Calculate the posterior mean for each individual and add this as a new variable in the `scores` data frame.

```
fitPoly$scores <- fitPoly$scores %>%
  mutate(posterior_mean = (P0 * 0) + (P1 * 1) + (P2 * 2) + (P3 * 3) + (P4 * 4) + (P5 * 5) + (P6 * 6), .a
head(fitPoly$scores)
```

```
##   marker MarkerName SampleName     ratio P0            P1           P2
## 2      1        SNP          2 0.9450980  0  0.000000e+00 0.000000e+00
## 3      1        SNP          3 0.9186047  0  1.513089e-70 1.630727e-32
## 4      1        SNP          4 0.9976387  0 4.965532e-143 6.960499e-85
## 5      1        SNP          5 1.0000000  0 3.237313e-161 5.236498e-99
## 6      1        SNP          6 0.9202756  0  2.644688e-71 5.145367e-33
## 7      1        SNP          7 0.9037620  0  4.181303e-64 2.965385e-28
##              P3           P4           P5            P6 posterior_mean maxgeno
## 2 5.487698e-278 1.792528e-95 1.000000e+00 1.765903e-145       5.000000       5
## 3  1.688654e-15 8.489692e-05 9.999151e-01  2.727826e-18       4.999915       5
## 4  2.023616e-54 1.173498e-31 8.163299e-14  1.000000e+00       6.000000       6
## 5  7.915965e-66 1.252725e-40 5.265544e-20  1.000000e+00       6.000000       6
## 6  7.908401e-16 5.665944e-05 9.999433e-01  6.710451e-18       4.999943       5
## 7  1.067884e-12 2.639557e-03 9.973604e-01  1.287973e-21       4.997360       5
##        maxP geno
## 2 1.0000000    5
## 3 0.9999151    5
## 4 1.0000000    6
## 5 1.0000000    6
## 6 0.9999433    5
## 7 0.9973604    5
```

5. Operating on elements within a list.

- Use a {purrr} function to identify the names of the columns/variables in the `scores` data frame that are *not* of type double.

```
names(keep(fitPoly$scores, ~ !is_double(.)))
```

```
## [1] "marker"     "MarkerName" "SampleName"
```

- Add a new atomic vector element called `col_means` **to the list** that contains just the column means of all the *double* variables in the `scores` data frame.

```
fitPoly$score_col_means <- map_dbl(fitPoly$scores[-1:-3], mean, na.rm = T)
fitPoly$score_col_means
```

```
##          ratio             P0             P1             P2             P3
##   9.110837e-01  1.141826e-236   5.571926e-24   3.159587e-06   1.528993e-02
##             P4             P5             P6 posterior_mean        maxgeno
##   2.535163e-01   5.074307e-01   2.237600e-01   4.939654e+00   4.951049e+00
##           maxP           geno
##   9.780702e-01   5.095652e+00
```

6. Extract elements from a list.

- Demonstrate three different ways to extract the `col_means` element from the list. The extracted element must *not be a list*.

```
# 1st Method: Extracting with $
fitPoly$score_col_means
```

```
##          ratio             P0             P1             P2             P3
##   9.110837e-01  1.141826e-236   5.571926e-24   3.159587e-06   1.528993e-02
##             P4             P5             P6 posterior_mean         maxgeno
##   2.535163e-01   5.074307e-01   2.237600e-01   4.939654e+00   4.951049e+00
##           maxP           geno
##   9.780702e-01   5.095652e+00
```

```
typeof(fitPoly$score_col_means)
```

```
## [1] "double"
```

```
# 2nd Method: Extract using [[list_number]]
fitPoly[[7]]
```

```
##          ratio             P0             P1             P2             P3
##   9.110837e-01  1.141826e-236   5.571926e-24   3.159587e-06   1.528993e-02
##             P4             P5             P6 posterior_mean         maxgeno
##   2.535163e-01   5.074307e-01   2.237600e-01   4.939654e+00   4.951049e+00
##           maxP           geno
##   9.780702e-01   5.095652e+00
```

```
typeof(fitPoly[[7]])
```

```
## [1] "double"
```

```
#3rd Method: pluck function
pluck(fitPoly, "score_col_means")
```

```
##          ratio             P0             P1             P2             P3
##   9.110837e-01  1.141826e-236   5.571926e-24   3.159587e-06   1.528993e-02
##             P4             P5             P6 posterior_mean         maxgeno
##   2.535163e-01   5.074307e-01   2.237600e-01   4.939654e+00   4.951049e+00
##           maxP           geno
##   9.780702e-01   5.095652e+00
```

```
typeof(pluck(fitPoly, "score_col_means"))
```

```
## [1] "double"
```

- Show two ways to extract the third element of `col_means`

```r
# 1st Method: Extracting with both $ and []
fitPoly$score_col_means[3]
```

```
##           P1
## 5.571926e-24
```

```r
typeof(fitPoly$score_col_means[3])
```

```
## [1] "double"
```

```r
# 2nd Method: Pluck function
pluck(fitPoly, "score_col_means", 3)
```

```
## [1] 5.571926e-24
```

```r
typeof(pluck(fitPoly, "score_col_means", 3))
```

```
## [1] "double"
```

For Loops

Consider the recursive sequence defined by

$$x_n = x_{n-1} + \frac{|x_{n-3} - x_{n-2}|}{4}.$$

In words, element n is the sum of element $n-1$ and the absolute value of the difference between between elements $n-3$ and $n-2$, divided by four. For example, if we let $x_1 = 3$, $x_2 = 1$, and $x_3 = 10$, then x_4 is

$$x_4 = 10 + \frac{|3-1|}{4} = 10.5.$$

1. Write a function called `calcn()` that takes as input a vector `x` with the first three elements of this sequence (x_1, x_2, x_3) and an integer `n` denoting the final element of the sequence to calculate.

- `calcn(x, n)` should return the value of the $\texttt{n}^{\text{th}}$ element.
- Include error checking to ensure both inputs are of the correct length and class and `n` is greater than 0.

```r
calcn <- function(x, n) {
  stopifnot(length(x) == 3 & is.numeric(x))
  stopifnot(n > 0 & is.integer(n))

nums <- vector(mode = "integer", length = n)
for (i in seq_along(nums)) {
  if (i <= 3) {
    nums[i] <- x[i]
} else {
```

```
  nums[i] <- nums[i-1] + ((abs(nums[i-3] - nums[i-2])) / 4)
  }
}
  return(nums[n])
}
```

- **Evaluate your function at the following inputs**:

```
calcn(c(11, 1, 130), 1000L)
```

```
## [1] 176.3333
```

```
calcn(c(11, 1, 130), 1L)
```

```
## [1] 11
```

```
calcn(c(7, 3, 20), 8L)
```

```
## [1] 26.625
```

Question: Lists, For-loops, and `map_*()`

Lists are often used to save simulation output. You can then extract individual elements from the lists using for-loops.

Consider the t-test, often used to test whether or not the true mean of a population for some observations is 0. We can use the following code to simulate data from a Normal (0,1) distribution, and then use a t-test to test if the true mean is 0:

```
set.seed(123)
x <- rnorm(n = 10, mean = 0, sd = 1)
tout <- t.test(x)
```

1. The output of `t.test()` is a list-like object. Use one function to show how many elements are in the list along with their names and class.

```
str(tout)
```

```
## List of 10
##  $ statistic  : Named num 0.247
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 9
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 0.81
##  $ conf.int   : num [1:2] -0.608 0.757
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num 0.0746
##   ..- attr(*, "names")= chr "mean of x"
##  $ null.value : Named num 0
```

```
##   ..- attr(*, "names")= chr "mean"
##  $ stderr      : num 0.302
##  $ alternative: chr "two.sided"
##  $ method      : chr "One Sample t-test"
##  $ data.name   : chr "x"
##  - attr(*, "class")= chr "htest"
```

2. Write a for-loop to perform the following three operations on iteration `i`:

    1. Draw 10 random observations from a normal distribution with mean 0 and standard deviation 2.
    2. Run a t-test on these 10 observations.
    3. Save the output of the t-test as the `i`th element in a list called `tlist`.

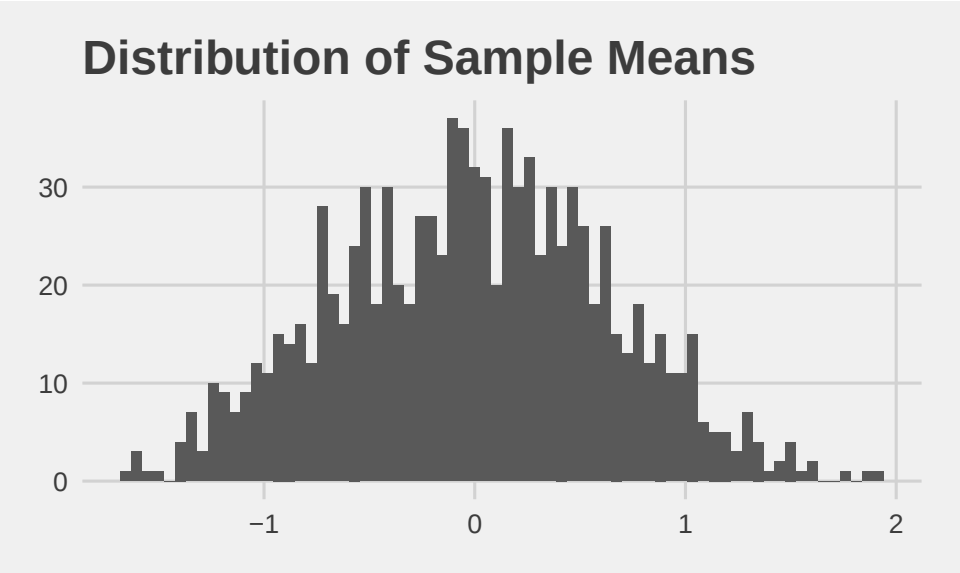
- Set the seed to 1 and run for 1000 iterations.

```
set.seed(1)
tlist <- vector(mode = "list", length = 1000)
for (i in 1:1000) {
 x <- rnorm(10, mean = 0, sd = 2)
 tout <- t.test(x)
 tlist[[i]] <- tout
}
```

3. Use the appropriate `purrr:map_*` function to extract the sample mean from each test (from 2) and pipe to an appropriate plot to show the sampling distribution of the sample mean.

- Hint: Make sure the data going into ggplot is a data frame (tibble)

```
tlist %>%
map_dbl(~.$estimate[[1]]) %>%
  tibble(mean=.) %>%
  ggplot(aes(x = mean))+
  geom_histogram(bins = 70) +
  ggthemes::theme_fivethirtyeight()+
  ggtitle("Distribution of Sample Means")
```



4. The p-value is an important quantity in statistics. Use a for-loop to extract the p-values from each test in part 2 and combine these p-values into a single atomic vector called `pvec_f`. Show the first 6 values.

```
pvec_f <- vector(mode = "double", length = 1000)
for (i in 1:1000) {
  pvec_f[[i]] <- tlist[[i]]$p.value
}
head(pvec_f)
```

```
## [1] 0.6052327 0.4806056 0.6686761 0.6480420 0.4945143 0.6653161
```

5. Use the appropriate `purrr:map_*` function to extract the p-values from each test in part 2 and combine these p-values into a single atomic vector called `pvec_m`. Show the first 6 values.

```
pvec_m <- tlist %>%
  map_dbl(~ .$p.value)
head(pvec_m)
```

```
## [1] 0.6052327 0.4806056 0.6686761 0.6480420 0.4945143 0.6653161
```