Lists, Apply, and Map

Data Wrangling and Husbandry

2/17/2020

Lists

Lists are a kind of vector, in which the individual elements do not need to be the same type, and in fact the elements can be vectors, matrices, and even lists. You can construct a list with the function `list()`.

```
example <- list(42, "B", TRUE, seq(1, 7, by = 2))
example
```

```
## [[1]]
## [1] 42
##
## [[2]]
## [1] "B"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1 3 5 7
```

```r
str(example)
```

```
## List of 4
##  $ : num 42
##  $ : chr "B"
##  $ : logi TRUE
##  $ : num [1:4] 1 3 5 7
```

You can get an individual element of a list with a construction like `example[[4]]`. `example[4]` is quite similar, but actually gives a list with just the one element. The single brackets can also be used to pick a range, for example `example[3:4]`, but double brackets can only be used for a single element. If the elements have names, you can also use the name.

```
names(example) <- c("First", "Next", "Hmm", "A_vector")
example[[4]]
```

```
## [1] 1 3 5 7
```

```
example[4]
```

```
## $A_vector
## [1] 1 3 5 7
```

```r
example[3:4]
```

```
## $Hmm
## [1] TRUE
##
## $A_vector
## [1] 1 3 5 7
```

```r
example[["A_vector"]]
```

```
## [1] 1 3 5 7
```

```r
example[[4]][3]
```

```
## [1] 5
```

```r
example$A_vector # same as example[["A_vector"]]
```

```
## [1] 1 3 5 7
```

```r
class(example[3:4])
```

```
## [1] "list"
```

```r
class(example[4])
```

```
## [1] "list"
```

```r
class(example[["A_vector"]])
```

```
## [1] "numeric"
```

```r
class(example[[4]][3])
```

```
## [1] "numeric"
```

```r
class(example$A_vector) # same as example[["A_vector"]]
```

```
## [1] "numeric"
```

(from Section 20.5.3 of *R for Data Science*)

As it turns out, a data frame in R is a list of vectors, all of the same length, but not necessarily of the same type.

```
library(babynames)
head(babynames)
```

year	sex	name	n	prop
1880	F	Mary	7065	0.0723836
1880	F	Anna	2604	0.0266790
1880	F	Emma	2003	0.0205215
1880	F	Elizabeth	1939	0.0198658
1880	F	Minnie	1746	0.0178884
1880	F	Margaret	1578	0.0161672

```
head(babynames[[3]])
```

```
## [1] "Mary"      "Anna"      "Emma"      "Elizabeth" "Mir
```

Apply

The `apply` function is part of a family of base R functions that themselves apply functions. If `x` is a matrix or data.frame, `apply(x, margin, fun, ...)` applies the function `fun()` to the rows (if `margin = 1`) or columns (if `margin = 2`) of `x`. Any additional arguments to `fun()` go at the end.

```
example2 <- data.frame(w = rnorm(10), x = rnorm(10, mean =
  y = rnorm(10, sd = 10), z = rnorm(10, mean = -2, sd = .5)
example2[2, 1] <- NA
apply(example2, 1, mean)
```

```
##  [1] -0.3162898         NA  3.0718658 -2.5516299  0.8334
##  [7]  1.3033323 -1.2963317 -2.0204565  1.7067011
```

```
apply(example2, 2, mean)
```

```
##          w          x          y          z
##         NA  1.9469522  0.5488749 -2.1294818
```

```r
apply(example2, 1, mean, na.rm = TRUE)
```

```
##  [1] -0.3162898 -1.4279094  3.0718658 -2.5516299  0.8334
##  [7]  1.3033323 -1.2963317 -2.0204565  1.7067011
```

```r
apply(example2, 2, function(x) mean(x) - 2)
```

```
##          w          x          y          z
##         NA -0.05304781 -1.45112507 -4.12948182
```

I often find `apply()` useful to count NAs by row and column.

```
apply(example2, 1, function(x) sum(is.na(x)))
```

```
##  [1] 0 1 0 0 0 0 0 0 0 0
```

```
apply(example2, 1, function(x) sum(is.na(x)))
```

```
##  [1] 0 1 0 0 0 0 0 0 0 0
```

`sapply()`, `vapply()`, and especially `lapply()` are generalizations of the `apply()` function. `sapply()` is a convenient way of applying a function to a list and (typically) getting a vector back:

```
sapply(example, length)
```

```
##    First     Next      Hmm A_vector
##        1        1        1        4
```

```
sapply(example2, length)
```

```
##  w  x  y  z
## 10 10 10 10
```

(Notice that we can use the `apply()` family of functions instead of using a `for` loop.)

map

The `map()` family of functions in the `purrr` library (loaded as part of the tidyverse) both generalizes and simplifies the `apply()` family.

An example:

```r
library(gapminder)

gapminder %>% split(.$year) # split returns a list by divi
```

```
## $`1952`
## # A tibble: 142 x 6
##    country     continent  year lifeExp      pop gdpPerca
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl
##  1 Afghanistan Asia       1952    28.8  8425333       779
##  2 Albania     Europe     1952    55.2  1282697      1601
##  3 Algeria     Africa     1952    43.1  9279525      2449
##  4 Angola      Africa     1952    30.0  4232095      3521
##  5 Argentina   Americas   1952    62.5 17876956      5911
##  6 Australia   Oceania    1952    69.1  8691212     10040
##  7 Austria     Europe     1952    66.8  6927772      6137
##  8 Bahrain     Asia       1952    50.9   120447      9867
##  9 Bangladesh  Asia       1952    37.5 46886859       684
## 10 Belgium     Europe     1952    68    8730405      8343
## # ... with 132 more rows
##
```

```
gapminder %>% split(.$year) %>%
  map(~ lm(lifeExp ~ log10(gdpPercap), weights = pop, data
```

```
## $`1952`
##
## Call:
## lm(formula = lifeExp ~ log10(gdpPercap), data = ., weigh
##
## Coefficients:
##       (Intercept)  log10(gdpPercap)
##             -12.81             19.75
##
##
## $`1957`
##
## Call:
## lm(formula = lifeExp ~ log10(gdpPercap), data = ., weigh
##
## Coefficients:
##       (Intercept)  log10(gdpPercap)
```

Details about `map(.x, .f)`

`map()` takes a list or vector in the first position (which might come via a pipe). For the second position, it takes

```
A function, formula, or atomic vector.
If a function, it is used as is.
If a formula, e.g. ~ .x + 2,
  it is converted to a function with two arguments,
  .x or . and .y. This allows you to create very
  compact anonymous functions with up to two inputs.
If character or integer vector, e.g. "y", it is converte
  to an extractor function, function(x) x[["y"]].
  To index deeply into a nested list, use multiple
  values; c("x", "y") is equivalent to z[["x"]][["y"]].
  You can also set .null to set a default to use instead
  of NULL for absent components.
```

```
map(example, length)
```

```
## $First
## [1] 1
##
## $Next
## [1] 1
##
## $Hmm
## [1] 1
##
## $A_vector
## [1] 4
```

```r
map(example, 1)
```

```
## $First
## [1] 42
##
## $Next
## [1] "B"
##
## $Hmm
## [1] TRUE
##
## $A_vector
## [1] 1
```

```r
map(example, 4, 2)
```

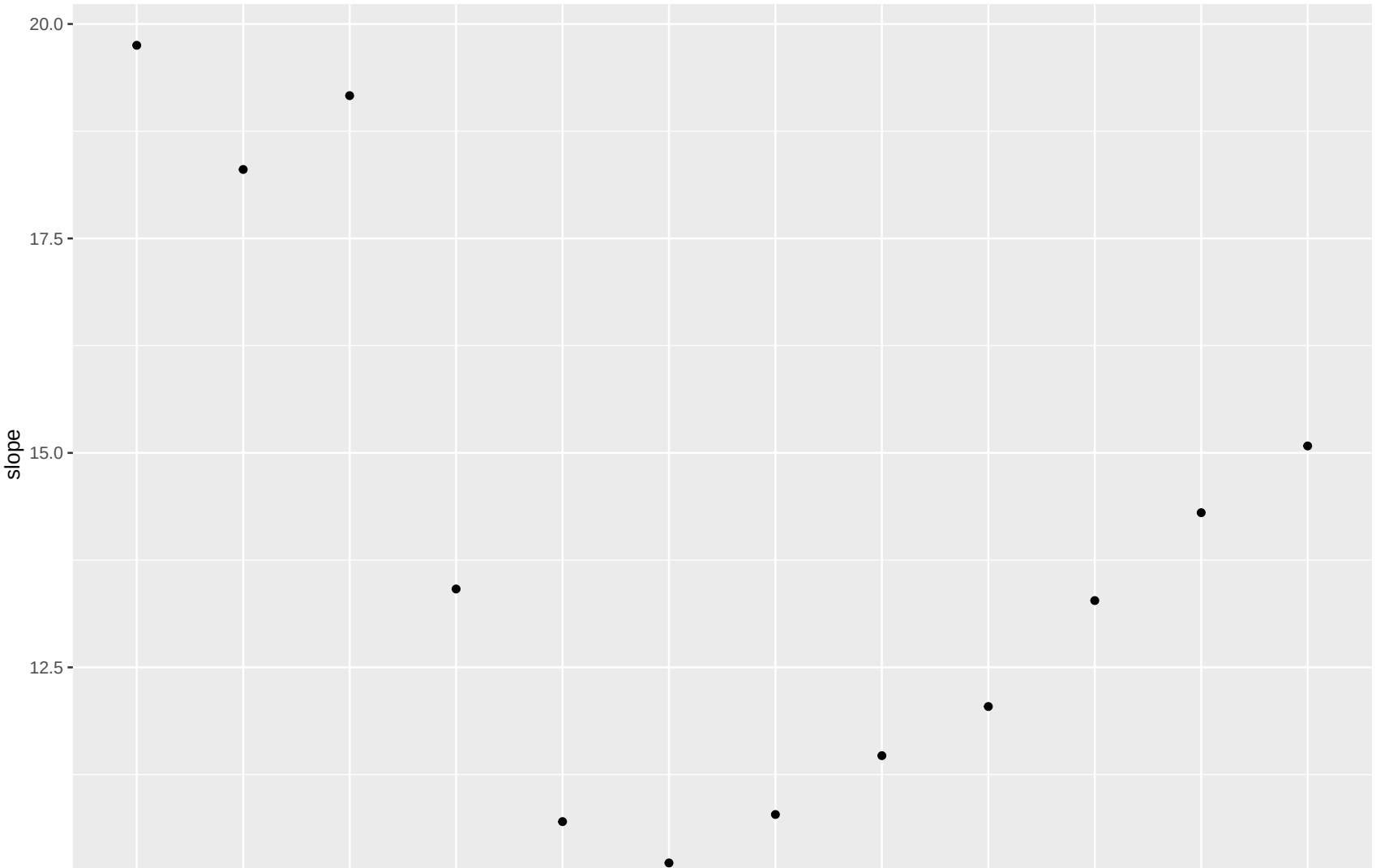
```
## $First
## NULL
##
## $Next
## NULL
##
## $Hmm
## NULL
##
## $A_vector
## [1] 7
```

- map() always returns a list.
- map_lgl() returns a logical vector
- map_int() an integer vector
- map_dbl(), a double vector
- map_chr(), a character vector

```
gapminder %>% split(.$year) %>%
  map( ~ lm(
    lifeExp ~ log10(gdpPercap),
    weights = pop,
    data = .
  )) %>%
  map_dbl(~ coef(.x)[2])
```

```
##      1952      1957      1962      1967      1972      1977
## 19.75138 18.30392 19.16484 13.41348 10.70174 10.21975 10
##      1992      1997      2002      2007
## 12.04351 13.27848 14.30282 15.08101
```

```
gapminder %>% split(.$year) %>%
    map(~ lm(lifeExp ~ log10(gdpPercap), weights = pop, dat
    map_dbl(~ coef(.x)[2]) %>%
    tibble(year = names(.), slope = .) %>%
    ggplot(aes(year, slope)) + geom_point()
```



In class exercises:

- `mtcars` is a built-in data.frame. Use the `map()` functions to get the mean of each column. Can you get those means in the form of a numeric vector?
- Use the `map()` functions to find the number of unique values for each column
- what does `map(1:5, rnorm)` give and why?

More examples

```
devtools::install_github("jennybc/repurrrsive")
```

```
##
##       checking for file ‘/tmp/RtmpLZCfcR/remotes554711468
##    -  preparing ‘repurrrsive’:
##       checking DESCRIPTION meta-information ...  v  check
##    -  checking for LF line-endings in source and make fil
##    -  checking for empty or unneeded directories
##    -  looking to see if a ‘data/datalist’ file should be
##    -  building ‘repurrrsive_1.0.0.9000.tar.gz’
##
##
```

```
library(repurrrsive) # a library of examples
```

```
data(package = "repurrrsive")
```

Table 2: Data sets in repurrrsive

Item	Title
discog	Sharla Gelfand's music collection
gap_nested	Gapminder data frame in various forms
gap_simple	Gapminder data frame in various forms
gap_split	Gapminder data frame in various forms
gh_repos	GitHub repos
gh_users	GitHub users
got_chars	Game of Thrones POV characters
sw_films	Entities from the Star Wars Universe
sw_people	Entities from the Star Wars Universe
sw_planets	Entities from the Star Wars Universe
sw_species	Entities from the Star Wars Universe
sw_starships	Entities from the Star Wars Universe
sw_vehicles	Entities from the Star Wars Universe
wesanderson	Color palettes from Wes Anderson movies

```
sw_people[[1]]
```

```
## $name
## [1] "Luke Skywalker"
##
## $height
## [1] "172"
##
## $mass
## [1] "77"
##
## $hair_color
## [1] "blond"
##
## $skin_color
## [1] "fair"
##
## $eye_color
## [1] "blue"
##
```

How many films has each character been in?

1. First calculate for one character
2. Make it recipe
3. Use `map()` to calculate it for all characters

```r
luke <- sw_people[[1]]
length(luke$films)
```

```
## [1] 5
```

```r
rey <- sw_people[[83]]
length(rey$films)
```

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

The recipe is then

```
~ length(.x$films)
```

Charlotte Wickham suggests thinking of `.x` as `purrr`'s pronoun

And now using map()

```
map(sw_people, ~ length(.x$films))
```

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

```
map_int(sw_people, ~ length(.x$films))
```

```
##  [1] 5 6 7 4 5 3 3 1 1 6 3 2 5 4 1 3 3 1 5 5 3 1 1 2 1 2
## [39] 1 1 2 1 1 3 1 1 1 3 3 3 2 2 2 1 3 2 1 1 1 2 2 1 1 2
## [77] 1 1 2 2 1 1 1 1 1 1 3
```

```
map_int(sw_people, ~ length(.x$films)) %>% table()
```

1	2	3	4	5	6	7
46	18	13	2	5	2	1

```
sw_people[map_int(sw_people, ~ length(.x$films)) == 7]
```

```
## [[1]]
## [[1]]$name
## [1] "R2-D2"
##
## [[1]]$height
## [1] "96"
##
## [[1]]$mass
## [1] "32"
##
## [[1]]$hair_color
## [1] "n/a"
##
## [[1]]$skin_color
## [1] "white, blue"
##
## [[1]]$eye_color
## [1] "red"
```

```
sw_people[map_int(sw_people, ~ length(.x$films)) == 7] %>%
```

```
## [1] "R2-D2"
```

It would have been nice to have the list have names

```
sw_people <- sw_people %>% set_names(map_chr(sw_people, "na
```

In class exercise

1. Create a character vector of hair color for Star Wars characters.
2. Create a table of hair color, sorted from most common to least common
3. Using the `sw_films` list, determine which film has the most characters.