# Iterations and functionals

Fall 2022

October 15 2022

## Why repeat ourselves?

```
tinydata <- tribble(
    ~case, ~x, ~y, ~z,
    "a", 5, 3, -2,
    "b", 7, 1, -5,
    "c", 9, 12, -3
)</pre>
```

#### Find the mean of each columns

```
mean(tinydata$x)
[1] 7
```

```
mean(tinydata$y)
[1] 5.333333
```

```
mean(tinydata$z)
[1] -3.333333
```

#### **Iteration**

# Iteration is the process of repeating the same action over and over again

## Multiple ways to do in R

- loops using for, while, etc
- vectorized functions that apply the same function to every element of a vector
- functional functions that apply the same function to elements in a vector, matrix, data frame, or list

## for loop

A way to iterate through a series of items stored as data object in R.

```
items <- c("grapes","bananas","chocolate","bread")
for(i in items){
   print(i)
}
[1] "grapes"
[1] "bananas"
[1] "chocolate"
[1] "bread"</pre>
```

```
i <- items[1]
print(i)
[1] "grapes"</pre>
```

```
i <- items[2]
print(i)
[1] "bananas"</pre>
```

# for loop components

the for() function is used to specify

- what object we're drawing from and
- what object we are writing to

## for loop components

```
The brackets {}
```

• Inside the brackets we house the code that is going to happen each iteration

## for loops tinydata

Let's iterate calculation of column means:

```
my_means <- rep(NA, 3)  # initialize an empty vector
my_means
[1] NA NA NA</pre>
```

```
for (i in 1:3) {  # three columns to get the mean for
  my_means[i] <- mean(tinydata[[i+1]])  # mean of col. i+1 (skip col. 1)
}
my_means
[1] 7.000000 5.333333 -3.333333</pre>
```

# For loops: index vector

• seq\_along(df) index based on columns of data frame

```
seq_along(tinydata)
[1] 1 2 3 4
```

Can also use

```
1:ncol(tinydata)
[1] 1 2 3 4
```

```
1:nrow(tinydata)
[1] 1 2 3
```

#### Function for conditional evaluation

• if x is numeric then standardize, else just return x

```
standardize <- function(x, ...){  # ... used for arbitrary number of arguments
  if (is.numeric(x)){  # condition
     (x - mean(x, ...))/sd(x, ...)  # if TRUE, standardize
  } else{  # else (FALSE)
     x  # return x unchanged
  }
}</pre>
```

```
standardize(c(2,4,6,8, 10))
[1] -1.2649111 -0.6324555 0.0000000 0.6324555 1.2649111
```

```
standardize(c(2,4,6,8, "10"))
[1] "2" "4" "6" "8" "10"
```

```
standardize(c(2,4,6,8, NA), na.rm = TRUE)
[1] -1.1618950 -0.3872983 0.3872983 1.1618950 NA
```

## Standardizing tinydata

```
# allocate storage in a new data frame
scaled_tinydata <- tinydata %>%
  mutate(
    x = NA,
    y = NA,
    z = NA
)
```

```
scaled_tinydata
# A tibble: 3 × 4
  case x y z
  <chr> <lgl> <lgl> <lgl>
1 a NA NA NA
2 b NA NA NA
3 c NA NA NA
```

```
for (i in seq_along(tinydata)){
    scaled_tinydata[, i] <- standardize(tinydata[[i]])
}</pre>
```

#### Vectorization

- A vectorized function will apply the same operation (function) to each element of a vector.
  - avoid loops by applying operations to each element of a vector

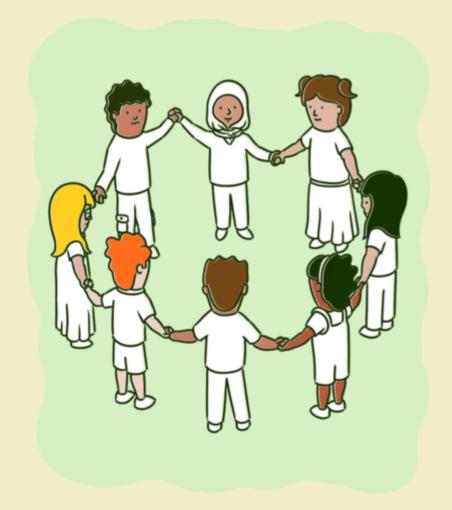
#### Vectorization

```
x <- c(1, 10, 100, 1000, 100000, 100000)
log10(x)  # log10 is a vectorized function
[1] 0 1 2 3 4 5</pre>
```

The for loop version

```
out <- rep(NA, length(x))
for (i in 1:length(x)){
  out[i] <- log10(x[i])
  }
out
[1] 0 1 2 3 4 5</pre>
```

# Group Activity 1



- Let's go over to maize server/ local Rstudio and our class moodle
- Get the class activity 15.Rmd file
- Work on activity 1
- Ask me questions

## **Functionals**

A functional function will apply the same operation (function) to each element of a vector, matrix, data frame or list.

- base-R: apply family of commands
- purrr package: map family of commands



# apply family of commands

R has a family of commands that apply a function to different parts of a vector, matrix or data frame

### lapply(X, FUN):

applies the FUN to each element in the vector/list X. Returns a list with length equal to that of the vector

sapply(X, FUN):

works like lapply but returns a vector (so FUN can only return one value)

## apply family of commands

R has a family of commands that apply a function to different parts of a vector, matrix or data frame

#### apply(matrix, MARGIN, FUN):

applies the function FUN to the matrix. MARGIN given (1=row, 2=column, c(1,2)=rows and cols). Returns an atomic vector or matrix

## tapply(x,y,FUN)):

applies FUN to atomic vector (variable) x for each group in categorical variable y. Returns an atomic vector with a  $\dim names$  attribute

## purrr package

powerful package for iteration with the same functionality as apply commands, but more readable (according to Hadley)

map(.x, .f) maps the function .f to elements in the vector/list .x

## lapply with tinydata

```
lapply(tinydata, FUN = mean)
$case
[1] NA

$x
[1] 7

$y
[1] 5.333333

$z
[1] -3.333333
```

• a 3x4 data frame is summarized in a list of length 4.

- R sees tinydata as a list whose elements are column vectors (variables)
- the FUN is applied to each list element
- a list is returned
- length is the number of variables in the data frame

### map

In purrr, the map function is equivalent to lapply

```
library(purrr)
map(tinydata, .f = mean)
$case
[1] NA

$x
[1] 7

$y
[1] 5.333333

$z
[1] -3.333333
```

## sapply with tinydata

Output is an atomic vector (simplify)

• a 3x4 data frame is summarized in a vector of length 4.

## map\_dbl

map\_dbl is equivalent to sapply

## map\_df

map\_df returns a data frame instead of a vector

No equivalency in base-R apply!

## Iterate or dplyr?!

- summarize\_all, summarize\_if, summarize\_at are all options that apply .funs to columns of a data frame
- if option needs a logical function that determines which columns to apply the .funs to

# functionals: single function that mutates

standardize function gives us a list of standardized values

```
tinydata
A tibble: 3 \times 4
 case x y
 <chr> <dbl> <dbl> <dbl>
                   -2
1 a
   7 1 -5
2 b
              12 -3
3 c
```

```
lapply(tinydata, FUN = standardize)
 $case
 [1] "a" "b" "c"
 $x
 \lceil 1 \rceil -1 \quad 0 \quad 1
 $y
 [1] -0.3982161 -0.7395442 1.1377602
$z
\lceil 1 \rceil
      0.8728716 - 1.0910895 0.2182179
```

• a 3x4 data frame is mutated to a list of 4 vectors of length 3 each

## lapply with tinydata

Using dplyr::bind\_cols converts the list to a data frame with variables equal to list entries

# map\_df

In purrr, the map\_df is equal to lapply + bind\_cols:

a 3x4 data frame is mutated to standardized 3x4 data frame

## Iterate or dplyr?!

- mutate\_all, mutate\_if, mutate\_at are all options that apply .funs to columns of a data frame
- if option needs a logical function that determines which columns to apply the .funs to

## applying multiple functions

Let's get the 0.1 and 0.9 quantile for variables in tinydata

```
quantile(tinydata$x, probs = c(.1, .9))
10% 90%
5.4 8.6
```

```
quantile(tinydata$y, probs = c(.1, .9))
  10%  90%
  1.4  10.2
```

```
quantile(tinydata$z, probs = c(.1, .9))
10% 90%
-4.6 -2.2
```

• the function output is a vector of length 2 (same lengths as probs)

## map\_df: getting quantiles

```
tinydata %>%
  select_if(is.numeric) %>%  # only numeric columns
  map_df(
    .f = quantile,  # function to apply to cols
    probs = c(.1, .9)) # extra function arguments
# A tibble: 3 × 2
    `10%` `90%`
    <dbl> <dbl>
1    5.4    8.6
2    1.4    10.2
3    -4.6    -2.2
```

## map\_df: getting quantiles

Can use .id to record the variable names from tinydata:

map\_df options

# There are two types of map\_df

- map\_dfr: row binds the list created by map
  - entries in the list are rows in the data frame
- map\_dfc: column binds the list created by map
  - entries in the list are columns in the data frame

## Iterate or dplyr?!

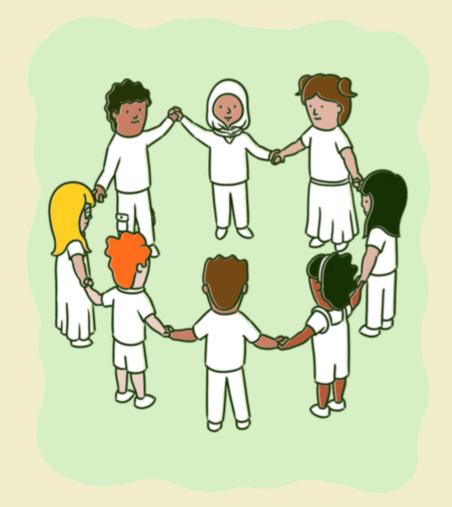
- summarize\_all, summarize\_if, summarize\_at can work with functions like quantile that return multiple values.
- the form of the output is a transposed version of map\_df

- rows = 0.1 and 0.9 quantiles
- cols = variables

# Iterate or dplyr?!

We need to manually add a percentile variable to help us ID the value in each row

# Group Activity 2



- Let's go over to maize server/ local Rstudio and our class moodle
- Get the class activity 15.Rmd file
- Work on activity 2
- Ask me questions