looping

Nicholus Tint Zaw

2022-08-31

For loops

ref: https://r4ds.had.co.nz/iteration.html#iteration

Let's say we wnat to calculate the mediam value for each vector from the tibble.

```
# create tilbble

df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

median(df$a)
```

```
## [1] 0.1471264
```

```
median(df$b)
```

```
## [1] -0.08551792
```

```
median(df$c)
```

```
## [1] 0.290721
```

```
median(df$d)
```

```
## [1] 0.7591783
```

Let apply loop instead of individual command for each vector.

```
output <- vector("double", ncol(df))  # 1. output

for (i in seq_along(df)) {            # 2. sequence
  output[[i]] <- median(df[[i]])      # 3. body
}

output
```

```
## [1]  0.14712636 -0.08551792  0.29072096  0.75917827
```

Let's do some exercise!

1. Compute the mean of every column in `mtcars`.
2. Determine the type of each column in `nycflights13::flights`.
3. Compute the number of unique values in each column of `iris`.

```
# Compute the mean of every column in `mtcars`.
mtcars_means <- vector("double", ncol(mtcars))  # 1. output

for (i in seq_along(mtcars)) {            # 2. sequence
  mtcars_means[[i]] <- mean(mtcars[[i]])        # 3. body
}

mtcars_means
```

```
##  [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##  [7]  17.848750   0.437500   0.406250   3.687500   2.812500
```

```
# Determine the type of each column in `nycflights13::flights`.
flight_type <- vector("double", ncol(nycflights13::flights))  # 1. output

for (i in seq_along(mtcars)) {            # 2. sequence
  flight_type[[i]] <- class(nycflights13::flights[[i]])        # 3. body
}

flight_type
```

```
##  [1] "integer"   "integer"   "integer"   "integer"   "integer"   "numeric"
##  [7] "integer"   "integer"   "numeric"   "character" "integer"   "0"
## [13] "0"         "0"         "0"         "0"         "0"         "0"
## [19] "0"
```

```
# Compute the number of unique values in each column of `iris`.
iris_unique <- vector("double", ncol(iris))  # 1. output

for (i in seq_along(iris)) {            # 2. sequence
  iris_unique[[i]] <- length(unique(iris[[i]]))        # 3. body
}

iris_unique
```

```
## [1] 35 23 43 22  3
```

Unknown output length

```
means <- c(0, 1, 2)

output <- double()

for (i in seq_along(means)) {
```

```r
  n <- sample(100, 1)
  output <- c(output, rnorm(n, means[[i]]))
}

str(output)
```

```
##  num [1:150] 0.337 -0.7714 -0.2679 -0.0389 -0.0212 ...
```

```r
out <- vector("list", length(means))

for (i in seq_along(means)) {
  n <- sample(100, 1)
  out[[i]] <- rnorm(n, means[[i]])
}

str(out)
```

```
## List of 3
##  $ : num [1:60] -1.0219 -1.1595 -0.0119 -0.9022 -1.1861 ...
##  $ : num [1:74] 0.702 0.671 2.686 1.053 -0.872 ...
##  $ : num [1:34] 2.702 0.193 2.04 2.378 1.01 ...
```

```r
str(unlist(out))
```

```
##  num [1:168] -1.0219 -1.1595 -0.0119 -0.9022 -1.1861 ...
```

Unknown sequence length

```r
flip <- function() sample(c("T", "H"), 1)

flips <- 0
nheads <- 0

while (nheads < 3) {

  if (flip() == "H") {
    nheads <- nheads + 1
  } else {
    nheads <- 0
  }

  flips <- flips + 1
}

flips
```

```
## [1] 3
```

any shortcuts?

Of course! yes.

```
df %>% map_dbl(mean)
```

```
##           a           b           c           d
##  0.23648559 -0.09541687  0.26864670  0.65762533
```

```
df %>% map_dbl(median)
```

```
##           a           b           c           d
##  0.14712636 -0.08551792  0.29072096  0.75917827
```

```
df %>% map_dbl(sd)
```

```
##         a         b         c         d
## 1.5092289 0.9605085 1.3701721 0.8636614
```

Other function from `map` functions from `purrr` package.

1. map() makes a list.
2. map_lgl() makes a logical vector.
3. map_int() makes an integer vector.
4. map_dbl() makes a double vector.
5. map_chr() makes a character vector.

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(function(df) lm(mpg ~ wt, data = df))


unique(mtcars$cyl)
```

```
## [1] 6 4 8
```

```
summary(models[[1]])
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1513 -1.9795 -0.6272  1.9299  5.2523
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   39.571      4.347   9.104 7.77e-06 ***
## wt            -5.647      1.850  -3.052   0.0137 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.332 on 9 degrees of freedom
## Multiple R-squared:  0.5086, Adjusted R-squared:  0.454
## F-statistic: 9.316 on 1 and 9 DF,  p-value: 0.01374
```

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(~lm(mpg ~ wt, data = .))


models %>%
  map(summary) %>%
  map_dbl(~.$r.squared)
```

```
##         4         6         8
## 0.5086326 0.4645102 0.4229655
```

Base R function

apply

```
# create sample data
sample_matrix <- matrix(C<-(1:10),nrow=3, ncol=10)

print( "sample matrix:")
```

```
## [1] "sample matrix:"
```

```
sample_matrix
```

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

```
# Use apply() function across row to find sum
print("sum across rows:")
```

```
## [1] "sum across rows:"
```

```
apply(sample_matrix, 1, sum)
```

```
## [1] 55 55 55
```

```
# use apply() function across column to find mean
print("mean across columns:")
```

```
## [1] "mean across columns:"
```

```
apply( sample_matrix, 2, mean)
```

```
##  [1] 2.000000 5.000000 8.000000 4.333333 4.000000 7.000000 6.666667 3.000000
##  [9] 6.000000 9.000000
```

lapply

```
# create sample data
names <- c("priyank", "abhiraj","pawananjani", "sudhanshu","devraj")
print( "original data:")
```

```
## [1] "original data:"
```

```
names
```

```
## [1] "priyank"     "abhiraj"     "pawananjani" "sudhanshu"   "devraj"
```

```
# apply lapply() function
print("data after lapply():")
```

```
## [1] "data after lapply():"
```

```
lapply(names, toupper)
```

```
## [[1]]
## [1] "PRIYANK"
##
## [[2]]
## [1] "ABHIRAJ"
##
## [[3]]
## [1] "PAWANANJANI"
##
## [[4]]
## [1] "SUDHANSHU"
##
## [[5]]
## [1] "DEVRAJ"
```

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
# compute the list mean for each list element
lapply(x, mean)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535125
##
## $logic
## [1] 0.5
```

```
lapply(out, mean)
```

```
## [[1]]
## [1] -0.1740571
##
## [[2]]
## [1] 1.121729
##
## [[3]]
## [1] 1.75156
```

sapply

```
x1 <- list(
  c(0.27, 0.37, 0.57, 0.91, 0.20),
  c(0.90, 0.94, 0.66, 0.63, 0.06),
  c(0.21, 0.18, 0.69, 0.38, 0.77)
)
x2 <- list(
  c(0.50, 0.72, 0.99, 0.38, 0.78),
  c(0.93, 0.21, 0.65, 0.13, 0.27),
  c(0.39, 0.01, 0.38, 0.87, 0.34)
)

threshold <- function(x, cutoff = 0.8) x[x > cutoff]

x1 %>% sapply(threshold) %>% str()
```

```
## List of 3
##  $ : num 0.91
##  $ : num [1:2] 0.9 0.94
##  $ : num(0)
```

```
x2 %>% sapply(threshold) %>% str()
```

```
##  num [1:3] 0.99 0.93 0.87
```

tapply

```r
# print head of diamonds dataset
print(" Head of data:")
```

```
## [1] " Head of data:"
```

```r
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```r
unique(diamonds$cut)
```

```
## [1] Ideal     Premium   Good      Very Good Fair
## Levels: Fair < Good < Very Good < Premium < Ideal
```

```r
# apply tapply function to get average price by cut
print("Average price for each cut of diamond:")
```

```
## [1] "Average price for each cut of diamond:"
```

```r
tapply(diamonds$price, diamonds$cut, mean)
```

```
##      Fair      Good Very Good   Premium     Ideal
##  4358.758  3928.864  3981.760  4584.258  3457.542
```