Lecture 18 - While Loops

Professor Sanchez, Stat 133, Fall 2024

Warren Birkman

2024-10-09

Table of contents

1 Iterations and Loops (Cont.) - For Loops

```r
# The goal is to compute some summary statistic for each column
X = matrix(runif(30), nrow=10, ncol=3)
# Y = copy of X
Y = X # change first value to missing value
Y[1,1] = NA

# apply(Y, 2, mean)
  # will output a value NA for the first column's summary statistic

apply(Y, 2, mean, na.rm=TRUE)
```

```
[1] 0.5235239 0.4467177 0.5904324
```

Calculating the range of each row (`MARGIN=1`)

```r
myrange = function(u, na.rm){
  max(u, na.rm = na.rm) - min(u, na.rm = na.rm)
}

# inside apply, if the function, "myrange" in this case, takes more than 1
  # argument, place it after

apply(Y, MARGIN=1, myrange, na.rm=TRUE)
```

```
 [1] 0.09088569 0.26645110 0.15706591 0.67280210 0.47170545 0.50201162
 [7] 0.55159860 0.27455045 0.83004499 0.77616671
```

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
D = as_tibble(X)
```

```
Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
`.name_repair` is omitted as of tibble 2.0.0.
i Using compatibility `.name_repair`.
```

```r
D
```

```
# A tibble: 10 x 3
      V1     V2    V3
   <dbl>  <dbl> <dbl>
 1 0.449 0.0433 0.134
 2 0.319 0.0525 0.197
 3 0.697 0.854  0.829
 4 0.161 0.834  0.643
 5 0.835 0.791  0.363
 6 0.358 0.591  0.860
 7 0.241 0.233  0.784
 8 0.903 0.717  0.628
 9 0.988 0.158  0.496
10 0.210 0.193  0.969
```

```r
summarize(D, mean(V1), mean(V2), mean(V3))
```

```
# A tibble: 1 x 3
  `mean(V1)` `mean(V2)` `mean(V3)`
       <dbl>      <dbl>      <dbl>
1      0.516      0.447      0.590
```

But what if we have 1 trillion columns

1.1 Across()

Similar to `apply()` - equivalent, but can use names of columns. Output also refers to those names.

```r
D |>
  summarize(across(V1:V3, mean))
```

```
# A tibble: 1 x 3
     V1    V2    V3
  <dbl> <dbl> <dbl>
1 0.516 0.447 0.590
```

What if we want column names - `.names = "{}"`

```r
D |>
  summarize(across(V1:V3, mean, .names="mean_{.col}"))
```

```
# A tibble: 1 x 3
  mean_V1 mean_V2 mean_V3
    <dbl>   <dbl>   <dbl>
1   0.516   0.447   0.590
```

Another example

```r
D |>
  summarize(across(V1:V3, mean, .names="Mean {1:3}"))
```

```
# A tibble: 1 x 3
  `Mean 1` `Mean 2` `Mean 3`
     <dbl>    <dbl>    <dbl>
1    0.516    0.447    0.590
```

With an anonymous function

```r
D |>
  summarize(across(1:3, function(u) max(u) - min(u), .names="Range {1:length(D)}"))
```

```
# A tibble: 1 x 3
  `Range 1` `Range 2` `Range 3`
      <dbl>     <dbl>     <dbl>
1     0.827     0.811     0.835
```

2 While Loops

x_1	x_1	x_3

This table spans for 10 rows- and we want to summarize each column with $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$.

Here is the basic format of a while loop: `while (logical condition) {}`

In this case, each column can be summarized with: `xmeans[i] = mean(X[,i])`

where `i` takes the values `1`, `2`, and `3`

```
xmeans = rep(0,3) # initialize vector, storing column means
i=1               # start counter

while (i <= ncol(X)) {      # continue until all columns are processed
  xmeans[i] = mean(X[,1])   # calculate the means of the i-th column
  i = i+1                   # increment i by 1 to move to the next column
}
xmeans
```

2.0.0.1 While loop

```
xmeans = rep(0, ncol(X))
i=1

while (i <= ncol(X)) {
  xmeans[i] = mean(X[,i])
  i = i+1
}
xmeans
```

```
[1] 0.5160563 0.4467177 0.5904324
```

2.1 Infinite Loop Warning

Make sure a/the logical condition can be met in order to stop the while loop from running **infinitely**

```
xmeans = NULL
i=1

while (i > 0) {
  xmeans = c(xmeans, mean(X[,1]))
  i = i+1
}
xmeans
```

2.2 Future value example

$$FV = P \times (1 + r)^n$$

```
set.seed(133)
n = 5
r = runif(n, min=0, max=.2)
r # these are our rates of return to compute our future values
```

```
[1] 0.10720225 0.16925403 0.12716011 0.08462043 0.04271265
```

We could:

```
set.seed(133)
p = 1000
n = 5
r = runif(n, min=0, max=.2)

fv1 = p*(1+r[1])
fv2 = fv1 * (1+r[2])
fv3 = fv2 * (1+r[3])
fv4 = fv3 * (1+r[4])
fv5 = fv4 * (1+r[5])

c(fv1, fv2, fv3, fv4, fv5)
```

```
[1] 1107.202 1294.601 1459.222 1582.702 1650.304
```

Or! We could use a `for` loop

```
# Using a for loop

# initialized output vector
fv = rep(0, n)

# for loop
for (i in 1:n) {
  if (i==1){
    fv[i] = p*(1+r[i])
  } else{
    fv[i] = fv[i-1] * (1+r[i])
  }
}

fv
```

```
[1] 1107.202 1294.601 1459.222 1582.702 1650.304
```

2.2.1 Visualize future value with a timeline

```
dat = data.frame(
  year = 1:n,
  amount = fv
)
dat |>
  ggplot(mapping=aes(x=year, y= amount)) + geom_line() + geom_point()
```