Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (qing.zhao@ufl.edu)

Daijiang Li (dli1@ufl.edu)

Denis Valle (drvalle@ufl.edu)

Class Five

`ifelse()`

Visualization

Problem to solve

Suppose that we have a vector `vec_a` with 5 elements:

```r
vec_a <- c(3, 2, -5, 7, 0)
```

Problem to solve

Suppose that we have a vector `vec_a` with 5 elements:

```r
vec_a <- c(3, 2, -5, 7, 0)
```

Now we want to take square root for positive and zero values while leave negative values alone.

How can we do this?

Thinking processes

We may do the following things step by step for each element:

1. Is it non-negative?

Thinking processes

We may do the following things step by step for each element:

1. Is it non-negative?
2. If yes, take the square root (recall the functions from last class).

Thinking processes

We may do the following things step by step for each element:

1. Is it non-negative?
2. If yes, take the square root (recall the functions from last class).
3. If no, return the original value.

Transform into R

```r
vec_a
```

```
## [1]  3  2 -5  7  0
```

```r
vec_a2 <- vector(mode = "numeric", length = length(vec_a)) # to hold results
for (i in 1:length(vec_a)){
  if (vec_a[i] >= 0){
    vec_a2[i] = sqrt(vec_a[i])
  } else {
    vec_a2[i] = vec_a[i]
  }
}
vec_a2
```

```
## [1]  1.732051  1.414214 -5.000000  2.645751  0.000000
```

Pros and Cons

Pros

- Intuitive
- Easy to follow

Pros and Cons

Pros

- Intuitive
- Easy to follow

Cons

- Multiple steps of coding
    - Create vector to hold results
    - Then the 3-steps procedure for each element
- Relatively slow (imagine millions of elements)
    - If you need to do for loops on large number of elements, remember to specify the `length` of the vector to hold results

Vectorization

A "whole project" thinking instead of "element-wise"

Vectorization

A "whole project" thinking instead of "element-wise"

In R, this means a function takes the whole vector as input (instead of one element each time the `if()` function takes within the `for()` loop above)

Vectorization

A "whole project" thinking instead of "element-wise"

In R, this means a function takes the whole vector as input (instead of one element each time the `if()` function takes within the `for()` loop above)

In R, vectorized functions are much faster

Vectorization

A "whole project" thinking instead of "element-wise"

In R, this means a function takes the whole vector as input (instead of one element each time the `if()` function takes within the `for()` loop above)

In R, vectorized functions are much faster

But `for` loops are still very important (see next class)

Vectorization

A "whole project" thinking instead of "element-wise"

In R, this means a function takes the whole vector as input (instead of one element each time the `if()` function takes within the `for()` loop above)

In R, vectorized functions are much faster

But `for` loops are still very important (see next class)

More information about vectorization

Back to our problem

Back to our problem

```
ifelse(test_expression, yes, no)
```

Back to our problem

```
ifelse(test_expression, yes, no)
```

```
ifelse(vec_a >= 0, sqrt(vec_a), vec_a)
```

```
## Warning in sqrt(vec_a): NaNs produced

## [1]  1.732051  1.414214 -5.000000  2.645751  0.000000
```

```
vec_a2
```

```
## [1]  1.732051  1.414214 -5.000000  2.645751  0.000000
```

This concludes Class 5, Section 1

Please continue on to the next video

Why warning message?

```r
(vec_test <- vec_a >= 0)
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
```

```r
(vec_yes <- sqrt(vec_a))
```

```
## Warning in sqrt(vec_a): NaNs produced
```

```
## [1] 1.732051 1.414214      NaN 2.645751 0.000000
```

```r
(vec_no <- vec_a)
```

```
## [1]  3  2 -5  7  0
```

```r
ifelse(vec_test, vec_yes, vec_no)
```

```
## [1]  1.732051  1.414214 -5.000000  2.645751  0.000000
```

Why warning message?

```r
(vec_test <- vec_a >= 0)
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
```

```r
(vec_yes <- sqrt(vec_a))
```

```
## Warning in sqrt(vec_a): NaNs produced
```

```
## [1] 1.732051 1.414214      NaN 2.645751 0.000000
```

```r
(vec_no <- vec_a)
```

```
## [1]  3  2 -5  7  0
```

```r
ifelse(vec_test, vec_yes, vec_no)
```

```
## [1]  1.732051  1.414214 -5.000000  2.645751  0.000000
```

- if the ith element of `vec_test` is `TRUE`
- then take the ith element of `vec_yes`
- otherwise, take the ith element of `vec_no`

More examples

Now, suppose instead of return the original value if it is negative, we want to return `NA`.

More examples

Now, suppose instead of return the original value if it is negative, we want to return `NA`.

```r
ifelse(vec_a >= 0, sqrt(vec_a), NA)
```

```
## Warning in sqrt(vec_a): NaNs produced
```

```
## [1] 1.732051 1.414214       NA 2.645751 0.000000
```

More examples

Now, suppose instead of return the original value if it is negative, we want to return NA.

```
ifelse(vec_a >= 0, sqrt(vec_a), NA)
```

```
## Warning in sqrt(vec_a): NaNs produced
```

```
## [1] 1.732051 1.414214       NA 2.645751 0.000000
```

```
sqrt(ifelse(vec_a >= 0, vec_a, NA))
```

```
## [1] 1.732051 1.414214       NA 2.645751 0.000000
```

```
# No warning any more
```

More examples

Now, suppose instead of return the original value if it is negative, we want to return NA.

```r
ifelse(vec_a >= 0, sqrt(vec_a), NA)
```

```
## Warning in sqrt(vec_a): NaNs produced
```

```
## [1] 1.732051 1.414214       NA 2.645751 0.000000
```

```r
sqrt(ifelse(vec_a >= 0, vec_a, NA))
```

```
## [1] 1.732051 1.414214       NA 2.645751 0.000000
```

```r
# No warning any more
```

The order of functions matters for warning.

In general, it is better to keep the warning message.

More examples

`ifelse()` works for whole matrix too

```r
mat_a <- matrix(data = c(1:6, -2, -5, 0), nrow = 3, ncol = 3)
mat_a
```

```
##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]    2    5   -5
## [3,]    3    6    0
```

```r
ifelse(mat_a > 0, sqrt(mat_a), mat_a)
```

```
## Warning in sqrt(mat_a): NaNs produced
```

```
##          [,1]     [,2] [,3]
## [1,] 1.000000 2.000000   -2
## [2,] 1.414214 2.236068   -5
## [3,] 1.732051 2.449490    0
```

More examples

Suppose we have exam scores for students, and we want to assign pass (>= 60) or fail (<60) to each of them.

```r
set.seed(123)
scores <- data.frame(names = letters,
                     score = runif(n = 26, min = 30, max = 100))
head(scores, n = 3)
```

```
##   names    score
## 1     a 50.13043
## 2     b 85.18136
## 3     c 58.62838
```

How do we add another column that will indicate pass and fail?

More examples

```
str(scores)
```

```
## 'data.frame':	26 obs. of  2 variables:
##  $ names: Factor w/ 26 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ score: num  50.1 85.2 58.6 91.8 95.8 ...
```

```
(pass_fail <- ifelse(scores$score >= 60, "pass", "fail")) # recycle
```

```
##  [1] "fail" "pass" "fail" "pass" "pass" "fail" "pass" "pass" "pass" "pass"
## [11] "pass" "pass" "pass" "pass" "fail" "pass" "fail" "fail" "fail" "pass"
## [21] "pass" "pass" "pass" "pass" "pass" "pass"
```

```
scores$status <- pass_fail
head(scores, n = 6)
```

```
##   names    score status
## 1     a 50.13043   fail
## 2     b 85.18136   pass
## 3     c 58.62838   fail
## 4     d 91.81122   pass
## 5     e 95.83271   pass
```

More examples

What about if we want also assign "A" (>= 85), "B" (>= 75), "C" (>= 60) to those who passed the exam?

More examples

What about if we want also assign "A" (>= 85), "B" (>= 75), "C" (>= 60) to those who passed the exam?

```r
# nested ifelse
vec_levels <-
  ifelse(scores$score < 60, "fail",
       ifelse(scores$score < 75, "C",
              ifelse(scores$score < 85, "B", "A")))
vec_score <- scores$score # to make it easier to check
names(vec_score) <- vec_levels # in pratice, you should make it as a new column
vec_score
```

```
##     fail        A     fail        A        A     fail        C        A
## 50.13043 85.18136 58.62838 91.81122 95.83271 33.18895 66.96738 92.46933
##        C        C        A        C        B        C     fail        A
## 68.60045 61.96303 96.97833 61.73339 77.42994 70.08434 37.20473 92.98775
##     fail     fail     fail        A        A        B        C        A
## 47.22614 32.94417 52.95445 96.81526 92.26775 78.49624 74.83548 99.59888
##        B        B
## 75.89941 79.59713
```

More examples

What about if we want also assign "A" (>= 85), "B" (>= 75), "C" (>= 60) to those who passed the exam?

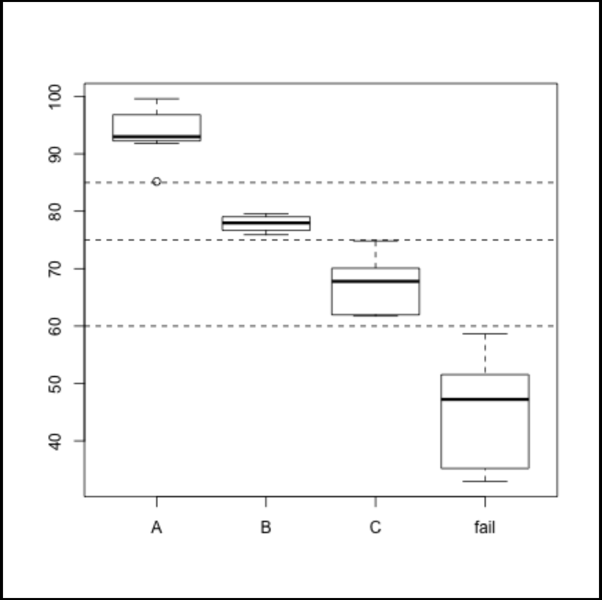
```r
# nested ifelse
vec_levels <-
  ifelse(scores$score < 60, "fail",
       ifelse(scores$score < 75, "C",
              ifelse(scores$score < 85, "B", "A")))
vec_score <- scores$score # to make it easier to check
names(vec_score) <- vec_levels # in pratice, you should make it as a new column
vec_score
```

```
##     fail        A     fail        A        A     fail        C        A
## 50.13043 85.18136 58.62838 91.81122 95.83271 33.18895 66.96738 92.46933
##        C        C        A        C        B        C     fail        A
## 68.60045 61.96303 96.97833 61.73339 77.42994 70.08434 37.20473 92.98775
##     fail     fail     fail        A        A        B        C        A
## 47.22614 32.94417 52.95445 96.81526 92.26775 78.49624 74.83548 99.59888
##        B        B
## 75.89941 79.59713
```

If needed, you can also assign "D" (e.g. >= 50, < 60), "E" (< 50) to those who failed by replacing `"fail"` with another `ifelse()`.

More examples

```
# "A" (>= 85), "B" (>= 75), "C" (>= 60)
boxplot(vec_score ~ vec_levels)
abline(h = c(60, 75, 85), lty = 2)
```



More examples

Be careful when use it with Dates and factors

```r
(vec_x <- factor(c("b", "a", "d", "e")))
```

```
## [1] b a d e
## Levels: a b d e
```

```r
(vec_y <- ifelse(vec_x == "a", vec_x, NA))
```

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

```r
as.numeric(vec_x)
```

```
## [1] 2 1 3 4
```

More examples

Be careful when use it with Dates and factors

```r
(vec_x <- factor(c("b", "a", "d", "e")))
```

```
## [1] b a d e
## Levels: a b d e
```

```r
(vec_y <- ifelse(vec_x == "a", vec_x, NA))
```

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

```r
as.numeric(vec_x)
```

```
## [1] 2 1 3 4
```

```r
vec_x2 <- as.character(vec_x)
ifelse(vec_x2 == "a", vec_x2, NA)
```

```
## [1] NA  "a" NA  NA
```

See `?ifelse` for example of Dates

Summary

The results of `ifelse()` have the same length as the input `test`, but attributes may not been preserved

Summary

The results of `ifelse()` have the same length as the input `test`, but attributes may not been preserved

Values are selected from `vec_yes` and `vec_no`

Summary

The results of `ifelse()` have the same length as the input `test`, but attributes may not been preserved

Values are selected from `vec_yes` and `vec_no`

If `vec_yes` and `vec_no` are too short, they are recycled

Summary

The results of `ifelse()` have the same length as the input `test`, but attributes may not been preserved

Values are selected from `vec_yes` and `vec_no`

If `vec_yes` and `vec_no` are too short, they are recycled

NA in the input give NA in the output

This concludes Class 5, Section 2

Please continue on to the next video

Use `ifelse` when plotting

Use `ifelse` when plotting

Main idea: use `ifelse` to create a variable that will be mapped to plot elements such as color (`col`), shape (`pch`), line type (`lty`), etc.

Another exam!

Suppose we have scores of a second exam for all students

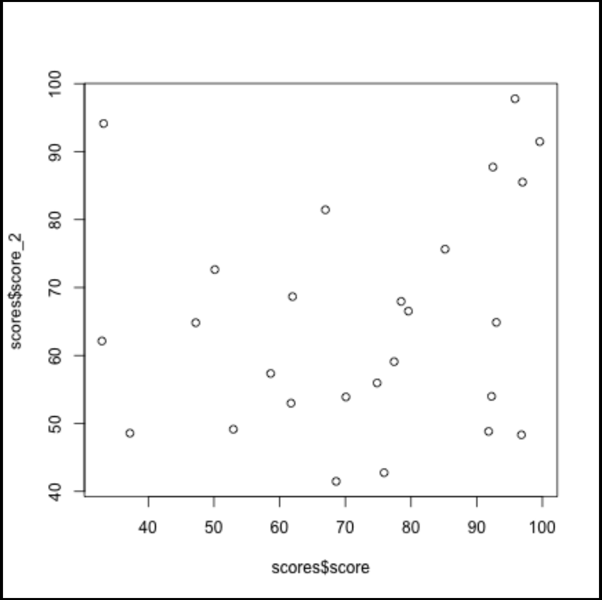
```r
scores$score_2 <- runif(n = nrow(scores), min = 40, max = 100)
head(scores)
```

```
##   names    score status  score_2
## 1     a 50.13043   fail 72.64396
## 2     b 85.18136   pass 75.64852
## 3     c 58.62838   fail 57.34958
## 4     d 91.81122   pass 48.82682
## 5     e 95.83271   pass 97.78145
## 6     f 33.18895   fail 94.13794
```

Plot two scores

Now, let's plot scores of exam 1 against scores of exam 2.

```
plot(x = scores$score, y = scores$score_2)
```

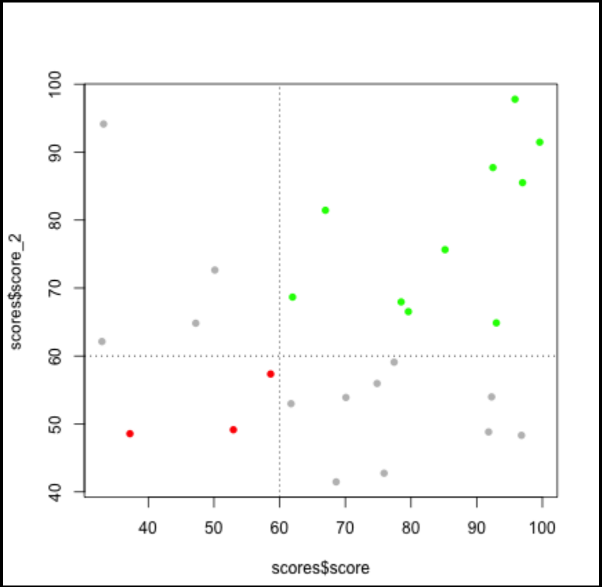


Task

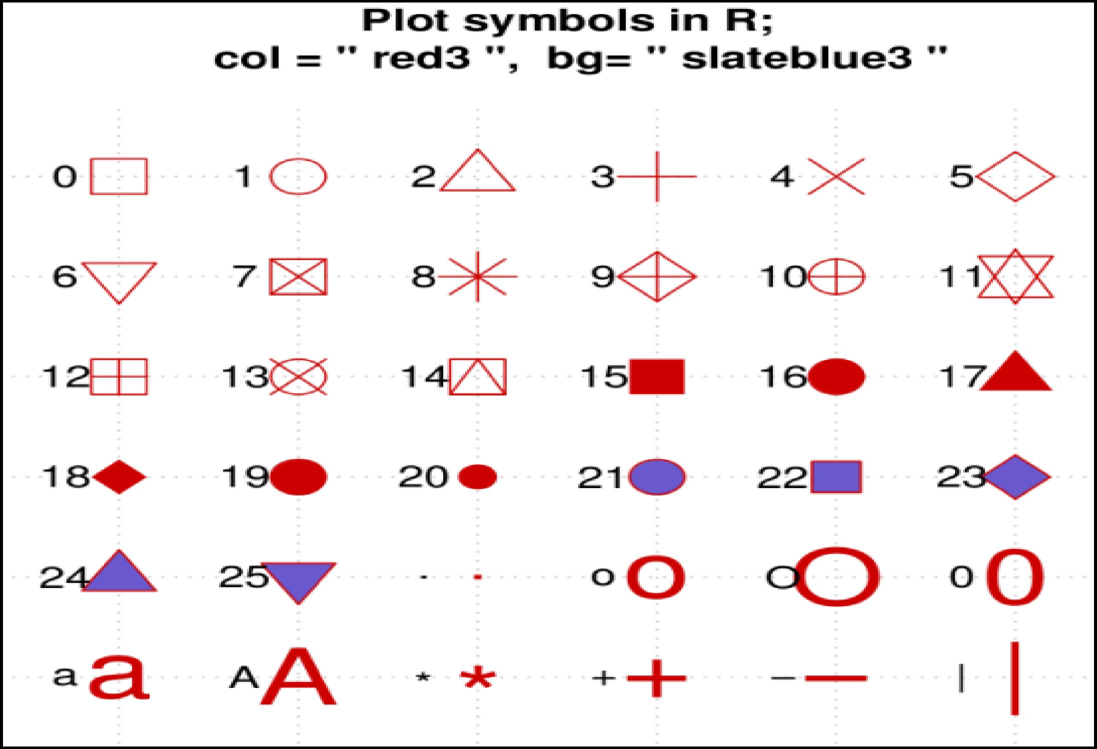
Color those points (students) with both exams failed as "red", both exam passed as "green", and one exam failed as "gray".

Conditional color

```
def_color <- ifelse(scores$score < 60 & scores$score_2 < 60, "red",
            ifelse(scores$score >= 60 & scores$score_2 >= 60, "green", "gray"))
plot(x = scores$score, y = scores$score_2, col = def_color, pch = 16)
abline(h = 60, lty = 3); abline(v = 60, lty = 3)
```

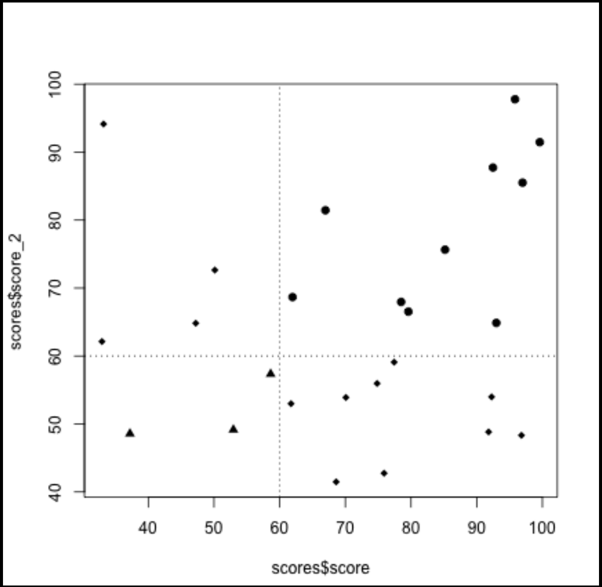


Shape


Plot symbols in R;
col = " red3 ", bg= " slateblue3 "
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
.
o
O
0
a
A
*
+
-
|

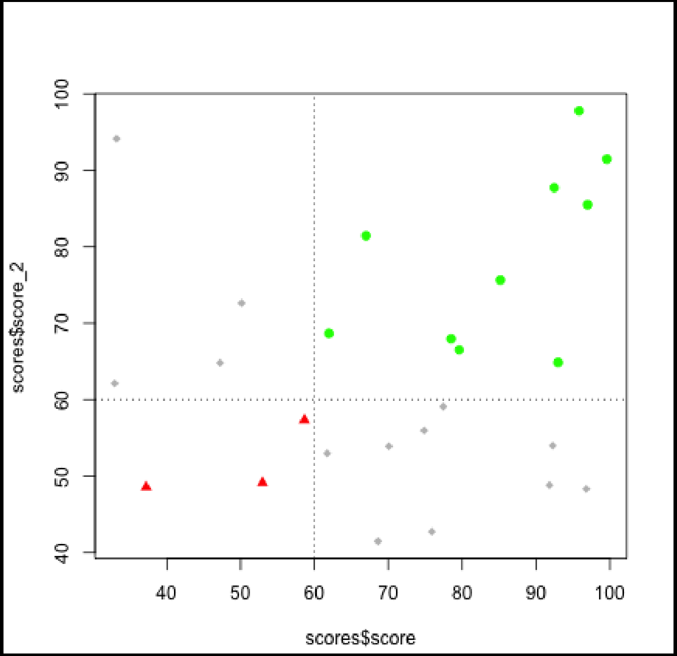
Conditional shape

```
def_shape <- ifelse(scores$score < 60 & scores$score_2 < 60, 17,
            ifelse(scores$score >= 60 & scores$score_2 >= 60, 19, 18))
plot(x = scores$score, y = scores$score_2, pch = def_shape)
abline(h = 60, lty = 3); abline(v = 60, lty = 3)
```



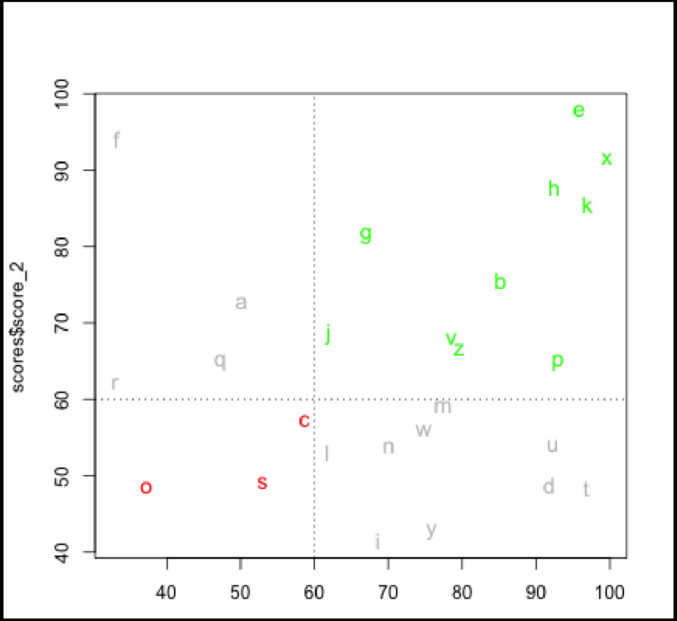
Conditional color and shape

```r
plot(x = scores$score, y = scores$score_2, pch = def_shape, col = def_color)
abline(h = 60, lty = 3); abline(v = 60, lty = 3)
```



Use text instead of points

```r
plot(x = scores$score, y = scores$score_2, type = "n")
text(x = scores$score, y = scores$score_2, labels = scores$names, col = def_color, cex = 1.2)
abline(h = 60, lty = 3); abline(v = 60, lty = 3)
```



Thank you and see you next class