Indexing, `subset()`, and missing values

BIOSTAT 213: Introduction to Computing in the R Software Environment

Monday, October 21, 2019

Take-home summary

I forgot to emphasize during the lecture that **when the indexing variable involves missing values, you should do one of the following: (1) add some `is.na()` code to the index as well, or (2) use `subset()`.**

Slide 65

Consider the following example from slide 65. In that example, the goal was to list all of the values of `bpxdi1` less than or equal to 30:

```
> ii$bpxdi1[!is.na(ii$bpxdi1)&(ii$bpxdi1<=30)]
 [1]  0 22  0 30  0  0 30 14  0  0  0  0  0  0  0
```

Notice the `!is.na(ii$bpxdi1)` portion of the code. This is necessary because there are missing values of `bpxdi1`:

```
> sum(!is.na(ii$bpxdi1))
[1] 4023
```

What would have happened had we not included `!is.na(ii$bpxdi1)` in the index? Let's try!

```
> ii$bpxdi1[ii$bpxdi1<=30]
  [1]  0 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [24] NA NA NA NA NA NA 22 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [47] NA NA NA NA NA NA NA NA NA NA NA NA  0 NA NA NA NA NA NA NA NA NA NA
 [70] NA 30 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [93] NA NA NA NA NA NA NA NA NA NA NA NA  0  0 30 NA 14 NA NA NA NA NA
[116] NA NA NA NA NA NA NA NA NA NA NA NA NA  0 NA NA NA NA NA NA  0 NA
[139] NA NA NA NA NA NA NA  0 NA NA NA NA NA NA NA NA NA NA NA  0 NA NA NA
[162] NA NA NA NA NA NA NA  0 NA NA NA NA NA NA NA NA NA  0 NA NA NA NA NA
[185] NA NA NA NA NA NA NA  0 NA NA NA NA NA NA NA NA NA
```

So, when we don't include `!is.na(ii$bpxdi1)` in the index, R returns the missing values of `bpxdi1` in the output.

As mentioned in the summary above, an alternate solution to adding `!is.na(ii$bpxdi1)` to the index is to instead use `subset()`:

```
> subset(ii$bpxdi1,ii$bpxdi1<=30)
 [1]  0 22  0 30  0  0 30 14  0  0  0  0  0  0  0
```

For this reason, `subset()` may be preferable to indexing in some cases.

Problem 8 from the in-class exercise

Indexing's strange treatment of missing values gets even stranger. Consider the following example:

```
> y<-1:5
> x<-c(1,0,1,0,1)
> is.na(x[4])<-TRUE
> y
[1] 1 2 3 4 5
> x
[1]  1  0  1 NA  1
> y[x==1]
[1]  1  3 NA  5
```

Here, the object we are indexing, `y`, does not have any missing values. Yet, when we try to index by `x==1`, we get missing values! Very strange!

To avoid the missing values, we have to—as highlighted in the summary above—include some `is.na()` code or use `subset()`:

```
> y[!is.na(x)&x==1]
[1] 1 3 5
> subset(y,x==1)
[1] 1 3 5
```

This is precisely the situation in problem 8. Note that the object we are indexing in this problem, `bmxbmi`, has no missing values. On the other hand, the indexing vector, `hs`, does have missing values:

```
> sum(is.na(ii$bmxbmi))
[1] 0
> sum(is.na(ii$hs))
[1] 4
```

As in the `y` and `x` example above, when we index using `hs`, we get missing values, even though there are no missing values of `bmxbmi`!

```
> mean(ii$bmxbmi[ii$hs=='History of smoking'])
[1] NA
```

This is strange, but consistent with the `y` and `x` example above. The solutions are to—again, highlighted in the summary above—include `!is.na()` or use `subset()`:

```
> mean(ii$bmxbmi[!is.na(ii$hs)&ii$hs=='History of smoking'])
[1] 29.86609
> mean(subset(ii$bmxbmi,ii$hs=='History of smoking'))
[1] 29.86609
```

A third option is the following:

```
> mean(ii$bmxbmi[ii$hs=='History of smoking'],na.rm=TRUE)
[1] 29.86609
```

However, for intuitiveness, I prefer either of the prior 2 solutions because the use of `na.rm=TRUE` here is surprising (since, as mentioned, `bmxbmi` does not actually have msising values).