Application of a Function Across Multiple Variables

Krishna Tateneni

Chicago RUG, Apr 2014

Let's start by creating a simple data frame

First, let's use sample() to create three numeric variables for our data frame:

```
> dataX = data.frame(V1 = sample(100,100,replace=TRUE),
                     V2 = sample(1000,100,replace=TRUE),
                     V3 = sample(10,100,replace=TRUE))
```

Then, we'll add a factor to create subgroups for analysis:

```
> dataX$grp = factor(rep(c("Grp1","Grp2"),50))
```

Finally, we'll add some missingness to our data:

```
> mpat = sample(100,100,replace=TRUE)
> dataX$V2[which(mpat<25 & grp=="Grp1")] = NA
> dataX$V2[which(mpat<15 & grp=="Grp2")] = NA
```

Our sample data frame has four variables, three numeric and one factor with two levels. Moreover, one of the numeric variables has missing values, and there is more missingness corresponding to one of the two factor levels.

A look at the sample data frame

```
> head(dataX)
  V1   V2 V3  grp
1 48  775  3 Grp1
2 88  314  7 Grp2
3 84  671  4 Grp1
4 49  706  8 Grp2
5 13   NA  2 Grp1
6 85  916  1 Grp2

> table(is.na(dataX$V2),dataX$grp)

       Grp1 Grp2
  FALSE   38   46
  TRUE    12    4
```

Number of valid cases for all variables

It's easy to get the number of missing values for one variable:
`sum(is.na(dataX$V2))` or even `table(is.na(dataX$V2))`

And thanks to colSums(), you don't need a loop for a whole set of variables!

```
X = matrix(dataX$V2,nrow=10)
colSums(is.na(X))
```

Now, let's write a function for that…

```
nvalid = function(x) {
  invisible(colSums(!is.na(x)))
}
```

"invisible" suppresses automatic output of the evaluation of the colSums expression.

```
> print(nvalid(dataX))
 V1  V2  V3 grp
100  84 100 100
```

colSums() versus apply()

apply() can be used instead of colSums():

```
apply(is.na(X),2,sum))
```

gives the same result as

```
colSums(is.na(X))
```

but colSums() is both easier to read and more efficient.

There are other functions like colSums():

```
rowSums(), colMeans(), and rowMeans()
```

Any that I don't know about?

In other situations, use apply().
And of course, there's Hadley Wickham's plyr package.
To paraphrase many wise speakers who've come before me:

"DON'T USE LOOPS!"

Missing values by subgroup

We can use by() to apply our function to slices of the sample data set:

```
by(dataX,dataX$grp,nvalid))
dataX$grp: Grp1
 V1  V2  V3 grp
 50  38  50  50
--------------------
dataX$grp: Grp2
 V1  V2  V3 grp
 50  46  50  50
```

And the result can be cleaned up with sapply:

```
sapply(by(dataX,dataX$grp,nvalid)), identity)
        Grp1   Grp2
   V1     50     50
   V2     38     46
   V3     50     50
  grp     50     50
```

Turning it all into a function…

```
grpsum = function(mydata,grpvar){
  byNs = by(mydata,grp,nvalid)
  Ns = sapply(byNs,identity)
  vars = names(mydata)
  result = as.data.frame(list(Var=vars,N=Ns))
  row.names(result) = NULL
  invisible(result)
}

> print(grpsum(dataX[,1:3],dataX$grp))
   Var N.Grp1 N.Grp2
1   V1     50     50
2   V2     38     46
3   V3     50     50
```

You can extend this to more complex functions

Here, we used `by` and `sapply` to apply our simple function `nvalid` to multiple variables and subgroups within our data frame.

You can use the same approach for simple built-in functions, such as `mean` or `sd`, or indeed more complex functions that you create.

Var	Label	Avg.1	Avg.2	Avg.3	N.1	N.2	N.3	ANOVA.F	ANOVA.p	Flags
A1	Multispecialty practice	0.53	0.43	0.41	131	69	86	3.2	0.1	3,1
Px	Procedure Volume	30.44	86.94	1.44	131	69	86	7.3	0.0	2,1|3,1|3,2
B1	Academic Center	0.08	0.12	0.06	131	69	86	0.1	0.7	
B2	Community Hospital	0.11	0.17	0.02	131	69	86	3.1	0.1	3,1|3,2
B3	Private Office	0.48	0.49	0.57	131	69	86	1.5	0.2	