

# 22S:164:Lab5

Sep. 27, 2007

## 1. Missing Data

In `alr3` library, the data files use the question mark “?” as a place holder for missing values. And R indicates missing values as NA. So, some packages in R may not recognize this as a missing value indicator. Therefore, you may need to change the question mark to NA.

- To read a data file with missing value indicated by “?”, use the argument `na.strings` in the `read.table` command to cover the “?” to the missing value indicator. For example, we want to read a data file called `sleep1.txt` with missing value indicated by “?” into R, the following command will work: 

```
> sleep1 <- read.table("sleep1.txt",header=TRUE, na.strings="?")
```

- There are several functions for working with missing value indicators. `is.na` serves two purposes, to set elements of a vector or array to NA, and to test each element of the vector or array to be NA, and return either TRUE or FALSE. For example, try the following commands regarding to `is.na`:

```
> a <- c(2,4,5,6,8) # set a to be the vector (2,4,5,6,8)
> is.na(a)<-c(1,5) # set element 1 and 5 of a to NA
> a # print a
> is.na(a) # for each element of a, test if a[i]=NA? return T or F
```

- `na.action` argument tells the command what to do if missing data indicators are present. The default option `na.action=na.omit` tells the command to delete all cases with missing data. Another option `na.action =na.fail` will prevent the command from executing at all with missing data. Type

```
> library(alr3);data(sleep1)
> m1 <- lm(log(SWS)~log(BodyWt)+log(Life)+log(GP),data=sleep1,na.action=na.omit)
which will delete 12 cases with missing values. Then fit
> m2 <- lm(log(SWS)~log(BodyWt)+log(GP),data=sleep1,na.action=na.omit)
```

You can't actually compare the two fits via an analysis of variance because they are based on different cases. To guarantee that two models are fitted with the same cases, use the `subset` argument to `lm`.

```
> m3 <- lm(log(SWS)~log(BodyWt)+log(GP),data=sleep1,subset=complete.cases(sleep1))
```

## 2. Bootstrap

In R, the `alr3` package provides a function called `boot.case` to deal with bootstrap. The `boot.case` has three arguments: the first argument is the regression model you want to fit in, the second argument is the number of bootstraps, the last argument is called `f`, the name of a function to be evaluated on each bootstrap. By default, the function `f` is `coef`, which will return the coefficient estimates on each bootstrap.

- Try the following example:

```
> data(transact)
> m1 <- lm(Time ~ T1 +T2,data=transact)
> betahat.boot <- boot.case(m1, B=999,f=coef)
```

The command returns a matrix with `B` rows and number of columns equal to the number of coefficients. The `i`th row gives the estimates from the `i`th bootstrap replication.

- The following command will give the bootstrap stand errors.

```
> apply(betahat.boot,2,sd)
```

- To get bootstrap 95% confidence intervals, try the following commands:
 

```
> cl<-function(x) quantile(x,c(.025,.975))
> apply(betahat.boot,2,cl)
```

### 3. Write Your Own Function

To write your own function in R, the safest way is to write the code in a text editor outside of R and save it in a text file. Then you can use the `source` function in R to read in the function and assign it to an R object.

- The structure of a function is as follows:

```
function(<arguments>)
{
  <body of function>
  <object to return>
}
```

- For example, we will write a function that does the following:

1. Accepts two vectors as arguments
2. if the two vectors are numeric and have the same length, return a list that contains the summaries of both vectors.
3. Otherwise display the message “Arguments must be two numeric vectors of equal length” and exit.

- Use your text editor to create a plain text file named “sum\_xy.R” that contains the following code.

```
Function(x,y)
{
  if (!is.numeric(x)|!is.numeric(y)|length(x)!=length(y))
    cat("Arguments must be two numeric vectors of equal length\n")
  else {
    sumx <- summary(x)
    sumy <- summary(y)
    list(statsx=sumx,statsy=sumy)
  }
}
```

- Now read it into an R function by entering the following in R:

```
> sum.xy <- source("sum_xy.R")$value
```

- Test your function by creating different types of vectors and using them as arguments. For example,

```
> x <- c(1:5); y <- c(12,8,4,6,7); z <- c("Mary","Lucy")
> sum.xy(x,y)
> sum.xy(x,z)
```