

Biometry. Lecture 13

Alexey Shipunov

Minot State University

March 11, 2015



- 1 One-dimensional data
 - R functions



```
> setwd("<working folder>")  
or  
"Change dir"  
in menu!
```

On Mac, be sure that startup option is working: `getwd()`
(`getwd()` checks if R is in working folder, `dir()` checks the folder
content)



One-dimensional data

R functions



- `shapiro.test()` is good but it is hard to apply if for data frames, and output is not very helpful.
- We will create the user function which run Shapiro-Wilks test with better output.



What is a function

```
> Sum <- function(a, b)
+ {
+ a + b
+ }
> Sum(2, 3)
```

Function is a piece of code which may run independently. All R commands are functions. Please note that every functions requires two parts: **arguments** in *round brackets* and **body** in *curly brackets*. It is too boring to enter functions line-by-line. Instead, it is better to copy function from external editor. If function contains mistake(s), one may use `fix()` command.



Let us create a simple useful function

R has no command for coefficient of variation, we will create it ourselves:

```
> CV <- function(x)
+ {
+   (sd(x) / mean(x)) * 100
+ }
> CV(trees[,3])
> CV(trees$Volume)
> sapply(trees, CV)
```

We can then run `fix(CV)` and add `round(..., 2)` function to make numbers more readable.



“for” will do the dirty work for you

R has special command for calculating mean of every column but how to calculate median of every column? One possible solution will employ “for” construction:

```
> for (i in 1:ncol(trees))  
{  
  print(median(trees[,i]))  
}
```

How to use `sapply()` to do the same?



shapiro.test() surgery

```
> set.seed(1)
> shapiro.test(rnorm(1000))
> res <- shapiro.test(rnorm(1000))
> str(res)
> res$p.value
```

Output from a test is a list!



Normality function, version 1

```
> Normality1 <- function(data.f)
+ {
+   result <- data.frame(var=names(data.f),
+   p.value=rep(0, ncol(data.f)),
+   normality=is.numeric(names(data.f)))
+   for (i in 1:ncol(data.f))
+     {
+       data.sh <- shapiro.test(data.f[, i])$p.value
+       result[i, 2] <- round(data.sh, 5)
+       result[i, 3] <- (data.sh > .05)
+     }
+   return(result)
+ }
Normality1(trees)
```

Complicated stuff! Please listen to explanations carefully. `return()` is needed because otherwise function will output nothing.



Normality1(): using the source code

```
> download.file("http://ashipunov.info/data/normality1.r",  
+ "normality1.r")  
> file.edit("normality1.r") # if you need it  
> source("normality1.r")  
> Normality11(trees) # the source specifies slightly  
# different name
```



Normality function, version 2

```
> Normality2 <- function(data.f, p=.05)
+ {
+   nn <- ncol(data.f)
+   result <- data.frame(var=names(data.f),
+   p.value=numeric(nn),
+   normality=logical(nn))
+   for (i in 1:nn)
+     {
+       data.sh <- shapiro.test(data.f[, i])$p.value
+       result[i, 2:3] <- list(round(data.sh, 5), data.sh > p)
+     }
+   return(result)
+ }
> Normality2(trees)
```

Features: has default value of argument and uses faster empty object creation (through `numeric()` and `logical()` creators).



ifelse() selector makes function smart

Will work best from inside a function:

```
> ifelse(is.numeric(1:3), mean(1:3), "ERROR!")
> CV1 <- function(x)
+ {
+   ifelse(is.numeric(x), (sd(x) / mean(x)) * 100, "ERROR!")
+ }
> CV1(iris[,1])
> CV1(iris[,5])
> sapply(iris, CV1)
```

CV1() function will check its argument if it is numeric



Normality function, version 3

```
> Normality3 <- function(df, p=.05)
+ {
+   sapply(df, function(.x)
+     ifelse(shapiro.test(.x)$p.value > p,
+     "NORMAL", "NOT NORMAL"))
+ }
> Normality3(trees)
> Normality3(log(trees+1))
```

Features: shorter, faster, vectorized and applicable to modified data



Finishing...

Save your commands!

(`savehistory(<todaysdate>.r)` or File -> Save as... on Mac)



Summary: most important commands

- `for(){}—the cycle`
- `ifelse(check, yes, no)—selector`
- `binom.test()` and `prop.test()`—tests for the equality of proportions
- `t.test()`—paired and non-paired two-sample parametric test
- `wilcox.test()`—paired and non-paired two-sample non-parametric test
- `response ~ factor`—if factor has exactly two levels, this is a model formula for two-sample test
- `%in%`—selection operator



For Further Reading



A. Shipunov.

Biometry [Electronic resource].

2012—onwards.

Mode of access:

http://ashipunov.info/shipunov/school/biol_240



A. Shipunov, and others.

Visual statistics. Use R!

Ongoing translation from Russian.

