Writing and Applying Functions - Text Strings

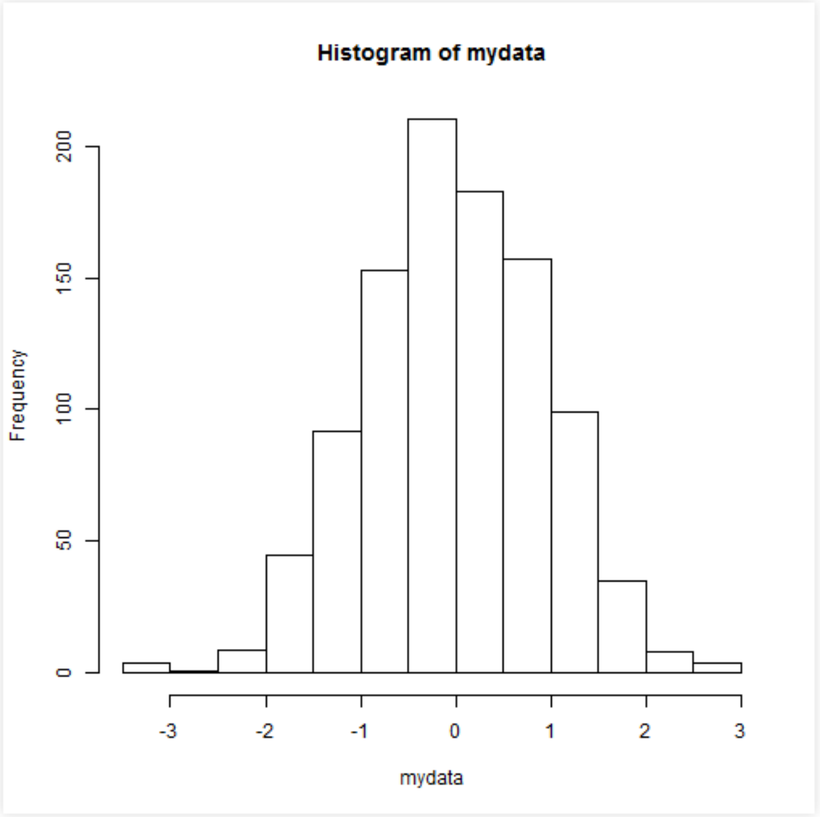
Richard Upton
5 August 2015

Functions

- Functions are a key element of R
- Functions take an input, process it and return an output
- The function output can be assigned to another object
- R has base functions and functions from installed packages
- Writing your own functions unleashes the power of R!

Using Functions

```
> mydata <- rnorm(1000)
> hist(mydata)
```



Using Functions

Are there arguments for **hist** that give us more control?

Using Functions

Arguments to functions can be supplied by name or by order

Try to limit using order when more than 1 argument

The first argument of **hist** is “x”, the data
The second argument is “breaks”

This argument can have multiple formats

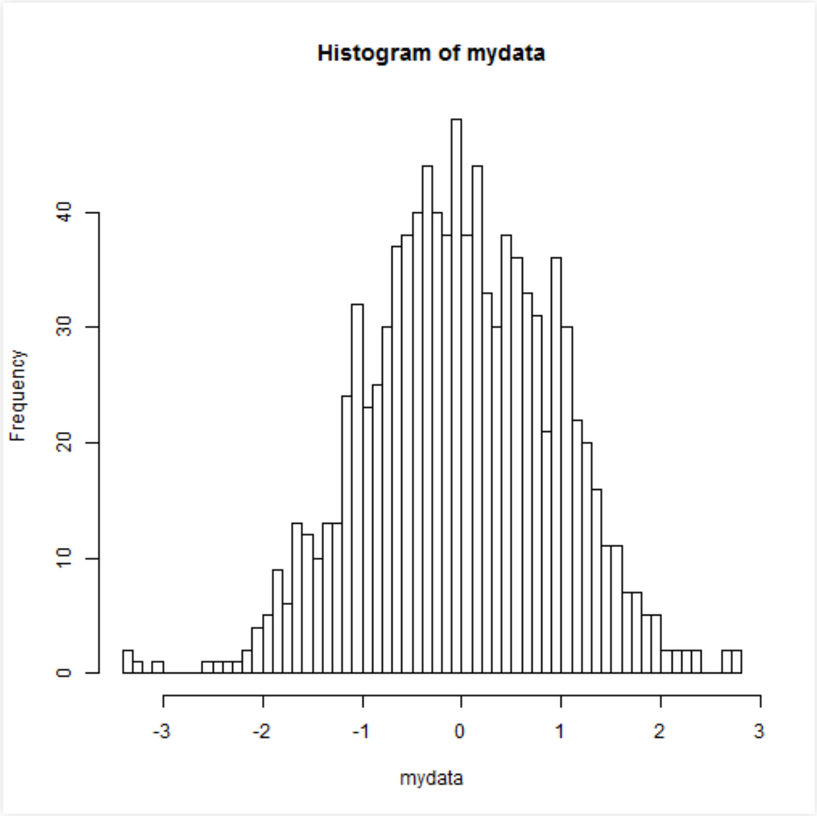
- A text string giving the cell method
- A number giving the number of cells

There is also an argument “main” for a plot title

Functions can have default values for arguments

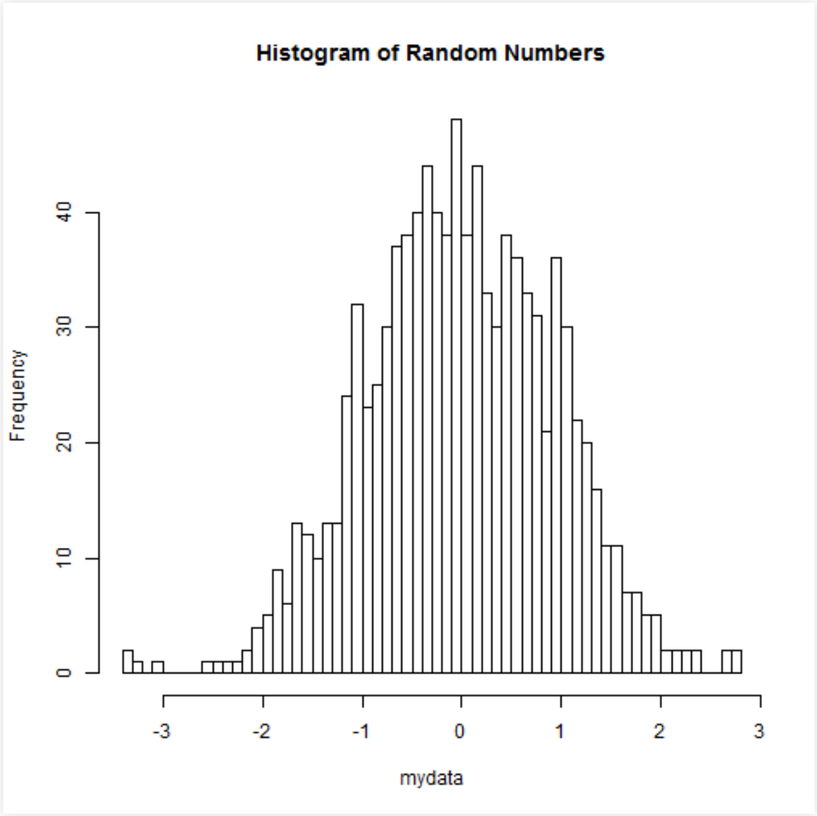
Function arguments by order

```
> hist(mydata, 50)
```



Function arguments by name

```
> hist(x=mydata, breaks=50, main="Histogram of Random Numbers")
```



Writing Functions

Functions in R take the following format:

```
addtwo <- function(x) {
   y <- x+2
   y }
```

- x is the function argument - the input to the function
- all the calculations take place between the curly brackets
- objects outside a function can be used in the function "environment"
- objects assigned inside a function are local to the function "environment"
- y is the output (result of the last line of function)

Testing the function

```
test <- c(2,3,4)
result <- addtwo(test)

result
```

```
[1] 4 5 6
```

Global and local variables

```
addtwo <- function(x) {
   y <- x+2
   y }
```

```
test <- c(2,3,4)
result <- addtwo(test)

#y
```

y is not found in the global environment; it's local to addtwo

Default argument values

```
addnum <- function(x, num=4) {
   y <- x+num
   y }
```

Unless a value for *num* is given, it will default to 4

```
test <- c(2,3,4)
result <- addnum(test)
result
```

```
[1] 6 7 8
```

```
result <- addnum(test, num=6)
result
```

```
[1]  8  9 10
```

Applying Functions

Often, we need to *apply* a function by a factor:

- Mean of data by Treatment group
- Maximum concentration by Subject
- Percent of baseline by Subject
- AUC by Treatment and Age
- Summarising demographic data

Applying Functions

We have some options:
The **apply** family of functions in the base
Effective but a little cluncky

The **doBy** package:
Easy syntax list outputs may need further processing

The **plyr** and **dplyr** packages
Easy syntax (but different to doBy)
Returns dataframes, lists and more

Applying Functions - Example data

```
> exampledata <- read.csv("data_apply.csv")
> exampledata
```

```
   ID DOSE AUC
1   1    5  34
2   1    5  42
3   1   10  60
4   1   10  58
5   2    5  43
6   2    5  35
7   2   10  74
8   2   10  65
9   3    5  28
10  3    5  27
11  3   10  51
12  3   10  49
```

Applying Functions - the doBy package

```
> library(doBy)
> summarydata <- summaryBy(AUC~ID, data=exampledata, FUN=mean)
> summarydata
```

```
  ID AUC.mean
1  1    48.50
2  2    54.25
3  3    38.75
```

Applying Functions - the doBy package

```
> library(doBy)
> summarydata <- summaryBy(AUC~DOSE, data=exampledata, FUN=mean)
> summarydata
```

```
  DOSE AUC.mean
1    5    34.83
2   10    59.50
```

Applying Functions - the doBy package

```
> library(doBy)
> summarydata <- summaryBy(AUC~ID+DOSE, data=exampledata, FUN=mean)
> summarydata
```

```
  ID DOSE AUC.mean
1  1    5     38.0
2  1   10     59.0
3  2    5     39.0
4  2   10     69.5
5  3    5     27.5
6  3   10     50.0
```

Text Strings

Text is delineated by quotes: “my text string”
Text makes an object of class *character*
Text strings are important in R as:

- object names
- working directory paths
- file names
- data

R has a number of tools for manipulating text strings

Paste to join text strings

```
string1 <- "PhaseI"
string2 <- "Control"
filename <- paste(string1,string2,".csv", sep="")
filename
```

```
[1] "PhaseIControl.csv"
```

```
write.csv(exampledata, file=filename)
```

the *sep* argument defaults to a single space.
It's set to “no space” here.

Text matching in R

There are a number of R methods for manipulating text
Many are based on pattern matching using “regular” expressions
regex are a common standard in many computing systems
regex are a compact, powerful and confusing instructions for text matching
See ?regex for an introduction
See ?glob2rx for a Windows wildcard format
gsub will get the basics done

gsub to edit text strings

First argument is old text
Second argument is new text

```
  filename
```

```
[1] "PhaseIControl.csv"
```

```
  filename2 <- gsub("PhaseI","PhaseIaVolunteer",filename)
  filename2
```

```
[1] "PhaseIaVolunteerControl.csv"
```

Splitting text strings

We have some options:

The **strsplit** function in the base package
Effective but a little cluncky

The **stringr** package (Hadley Wickham again)
Handy for splitting strings into columns
This is a common task in pharmacometrics

String splitting

```
> library(stringr)
> exampledata <- read.csv("data_string.csv")
> exampledata
```

```
    TIME CONC
1 10_min 0.56
2 30_min 0.92
3 1_hour 0.42
4 4_hour 0.61
```

String splitting

The TIME column is split into 2 columns at "_"

```
> newcols <- str_split_fixed(exampledata$TIME,"_",2)
> exampledata$TIMEnumber <- as.numeric(newcols[,1])
> exampledata$TIMEtext <- newcols[,2]
> exampledata
```

```
    TIME CONC TIMEnumber TIMEtext
1 10_min 0.56         10      min
2 30_min 0.92         30      min
3 1_hour 0.42          1     hour
4 4_hour 0.61          4     hour
```