R syntax guide

Richard Gonzalez
Psychology 613

August 27, 2015

This handout will help you get started with R syntax. There are obviously many details that I cannot cover in these short notes but these pages will help you get started using R. This handout parallels the SPSS handout I prepared for Psychology 613.

How to get R?

R is an open source, free statistics program. It can be downloaded at `http://www.R-project.org/.` Go the CRAN section of the website.

You will also want to download Rstudio, which provides a graphical interface to working with R, including an editor, output window, help window and plot window. Rstudio also interfaces with tools that help with version control (useful when multiple people are working on the same R project), tools for writing R packages, and tools for writing reports in R. Basically, Rstudio is an integrated development environment for working with R. Rstudio is open source and available at `http://www.rstudio.com/.`

R syntax

R syntax is relatively simple though not straightforward if you are new to programming. Syntax provides a concise way to tell people exactly which analyses you ran, easily recreate analyses, and run analyses more efficiently.

There is a package that provides an SPSS-like user interface called RCmdr. I recommend learning the syntax first. As with SPSS, there are many things that you will need to do that are not possible in the RCmdr interface so you will need to use syntax even if you use the "point and click" approach.

1. General tips

   (a) SPSS syntax is case sensitive. So "name" is not equal to "NAME" and not equal to "Name".

   (b) There is good documentation on each command. Type the name of the command with an initial question mark, as in `?hist` and you will see detailed information on the hist() function.

2. Entering data

   For small data sets you can enter data directly into vectors (aka columns).

```
heights <- c(62, 70, 59, 65, 68)
heights


## [1] 62 70 59 65 68
```

   The first line of the syntax creates a new variable named heights with five data points. The second line prints the contents of heights to the output window.

   For larger data sets you can save the data into a separate text file, which can be comma or tab delimited. For example, the command

```
data <- read.csv("FILENAME")
```

   can read in a csv file and store the contents into a new R object called data. If you type data $<-$ read.csv("FILENAME",header=T), the first row will be used as names for each column. Otherwise if you type read.csv("FILENAME"), you'll need to add column names in the R console window with

```
names(data) <- c("VAR1", "VAR2", "VAR3")
```

   where VAR1 corresponds to the first variable name, etc; there need to be as many variable names as there are columns in the object data.

   FILENAME is a file that resides in the current working directory, otherwise you'll need to specify the complete path so R can find the file as in `read.csv("C:\\working\\data\\FILENAME")`.

   I believe that on PC one can either use double back slashes `\\` or single forward slash `/` will work `c:/rich/working`. It seems that for Mac and Linux the forward slash format is the only one that works.

   The most common culprit of the R can't find the file error is that you didn't specify the path to the file so R doesn't know where the file is located.

   R can also read in an SPSS sav file, convenient of you've already created a data file where you named all variables, coded them, etc. You need to use the foreign library as in

```
library(foreign)
data <- read.spss("FILENAME")
```

3. Attaching data

   R distinguishes objects from their content. So if you read in a file with multiple columns of data and store it into an object called data, R only recognizes the object data not the

contents of data. In order to access the contents you either need to attach the object or refer to contents individually. To attach a data file you merely type the command attach(data). To refer to individual elements of data you type `data$content` as in `data$heights` if you want to access the heights column in data.

Here is an example. I will create an object data that has two columns, height and weight.

```
height<- c(62, 70, 59, 65, 68)
weight <- c(135, 180, 130, 155, 210)
data <- cbind(height,weight)
data
```

```
##      height weight
## [1,]     62    135
## [2,]     70    180
## [3,]     59    130
## [4,]     65    155
## [5,]     68    210
```

Right now there three objects active in R: the variable height, the variable weight, and the matrix of two columns data, which contains height in the first column and weight in the second column.

```
height
```

```
## [1] 62 70 59 65 68
```

```
weight
```

```
## [1] 135 180 130 155 210
```

```
data
```

```
##      height weight
## [1,]     62    135
## [2,]     70    180
## [3,]     59    130
## [4,]     65    155
## [5,]     68    210
```

Now let's delete height and weight using the remove command rm(), and print out the three objects again to see what happens.

```
rm(height,weight)
height

## Error in eval(expr, envir, enclos):  object 'height' not found

weight

## Error in eval(expr, envir, enclos):  object 'weight' not found

data

##      height weight
## [1,]     62    135
## [2,]     70    180
## [3,]     59    130
## [4,]     65    155
## [5,]     68    210
```

You can see the individual height and weight variables are no longer present, but the object data includes the height and weight variables as part of its contents.

We can access the contents of data by using the dollar sign syntax. We first need to save data as a a data.frame object in R. You can see that the variable height by itself is not accessible but `data$height` is accessible.

```
data <- data.frame(data)
height

## Error in eval(expr, envir, enclos):  object 'height' not found

data$height

## [1] 62 70 59 65 68

weight

## Error in eval(expr, envir, enclos):  object 'weight' not found

data$weight

## [1] 135 180 130 155 210
```

Another way to access the contents of data is to attach the data object

```
height

## Error in eval(expr, envir, enclos):  object 'height' not found

weight

## Error in eval(expr, envir, enclos):  object 'weight' not found

attach(data)
height

## [1] 62 70 59 65 68

weight

## [1] 135 180 130 155 210
```

We can detach an object through the detach command and then height is no longer available.

```
height

## [1] 62 70 59 65 68

detach(data)
height

## Error in eval(expr, envir, enclos):  object 'height' not found
```

Another way to access contents is through the with() command

```
height

## Error in eval(expr, envir, enclos):  object 'height' not found

with(data,print(height))

## [1] 62 70 59 65 68
```

There is much more to cover on these issues but you can learn them later.

4. Labeling variables

   The best way to label variables in R is to use descriptive variable names such as HeightInches. There is no limit to the number of characters in a variagle name.

5. Labeling values of a variable

   It is a good idea to label codes used for group membership. In R one way to do that is directly in the data entry, e.g., writing out "male" or "female" rather than use codes such as 1 or 2.

   If you already have codes in the data, you can recode with commands such as

```
#creates a variable gender with five 1s and five 2s
gender <- c(rep(1,5),rep(2,5))
gender

##  [1] 1 1 1 1 1 2 2 2 2 2

gendernew <- ifelse(gender==1,"male","female")
gendernew

##  [1] "male"   "male"   "male"   "male"   "male"   "female" "female" "female"
##  [9] "female" "female"
```

6. Defining missing values

   Missing values are entered in the data file as NA. If codes such as 999 are in data you will need to change those codes to NA. For example,

```
#make a variable x with numbers 1 to 5 for illustration
x <- 1:5
x

## [1] 1 2 3 4 5

x <- c(x,999)
x

## [1]   1   2   3   4   5 999

#taking a mean of this variable is meaningless because the 999 is treated as data
#rather than missing
mean(x)

## [1] 169

x[x==999] <- NA
x

## [1]  1  2  3  4  5 NA
```

```
#some commands don't like NA and don't report back anything if there are missing data
mean(x)

## [1] NA

mean(x,na.rm=T)

## [1] 3
```

7. Performing transformations

   Transformation are easy in R

```
x.sq <- x^2
x.sqrt <- sqrt(x)
x.log <- log(x)
```

8. Obtaining descriptive statistics for continuous variables

   Common R commands for descriptive statistics are mean(), median(), var(), max(), min(), range(), where you enter variable name inside the parentheses as in mean(x).

   Issues with missing data are one of the most common questions people have when learning R. So to reiterate the previous section, some R commands are fussy about missing data. If you want the command to ignore missing data then you have to tell the command explicitly to ignore missing data through the na.rm=TRUE argument.

```
test <- c(1,2,3,4,5)
mean(test)

## [1] 3

test <- c(test, NA)
test

## [1]  1  2  3  4  5 NA

mean(test)

## [1] NA

mean(test, na.rm=TRUE)

## [1] 3
```

9. Getting frequencies for categorical variables

```
gendernew

##  [1] "male"   "male"   "male"   "male"   "male"   "female" "female" "female"
##  [9] "female" "female"

table(gendernew)

## gendernew
## female   male
##      5      5
```

10. Obtaining a cross-tabulation between two categorical variables

    Simply include two variables in the table command. For example, the command

```
x <- c(1,1,1,1,2,2,2,3,4,4)
table(x,gendernew)

##    gendernew
## x   female male
##   1      0    4
##   2      2    1
##   3      1    0
##   4      2    0
```

    gives a count of variable x separately for males and females.

11. Producing a histogram

    hist(x)

12. Creating side-by-side boxplots

```
boxplot(x~gender)
```

    will produce separate boxplots by gender.

13. Generating a scatterplot.

    One can specify horizontal and vertical variables in the plot using this command syntax (put the variable you want on the horizontal axis, i.e., the predictor, first then the variable you want on the vertical axis second)

```
plot(x,y),
```

or equivalently, one can use the structural model notation (in this case vertical variable is listed first then the horizontal variable, or the predictor),

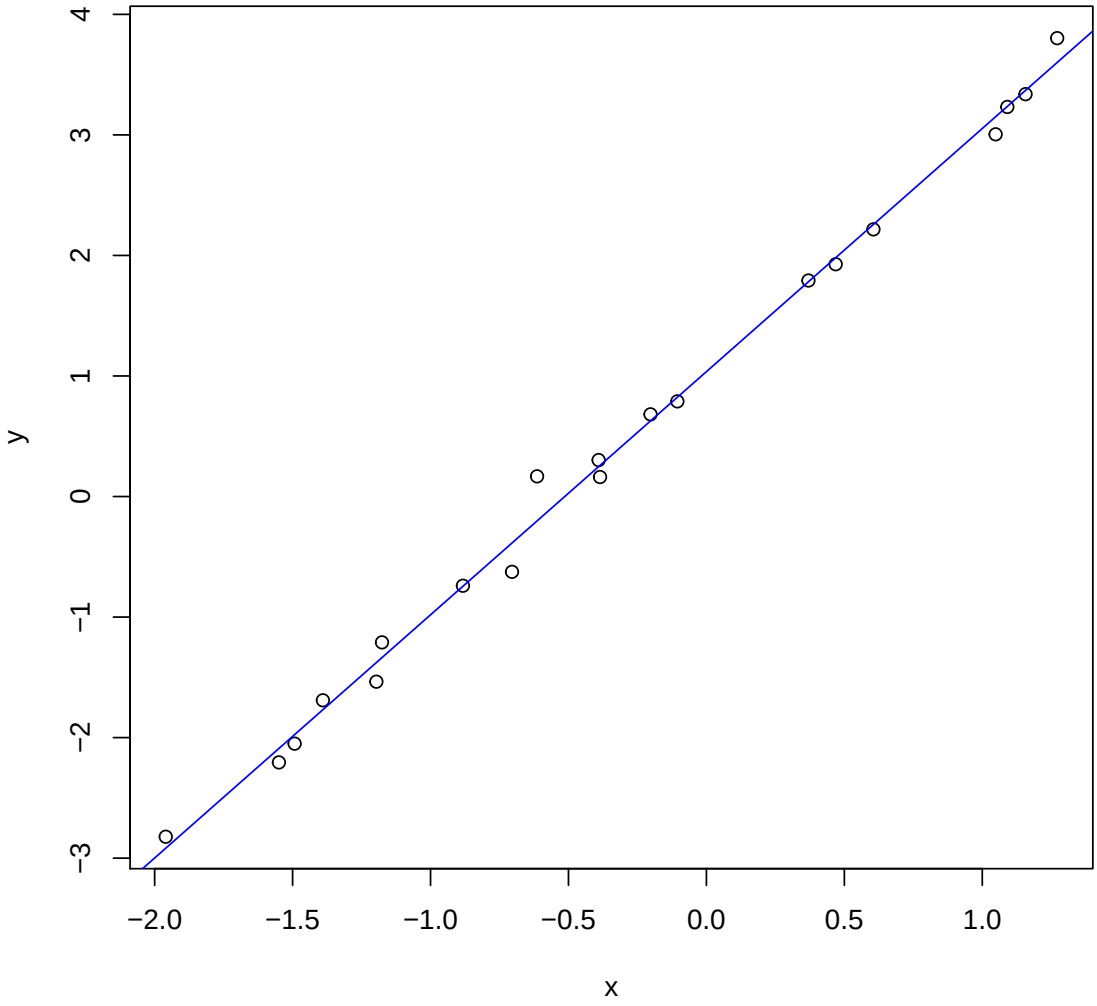
```
plot(y ~ x).
```

The command ?plot privdes more information on the plot command.

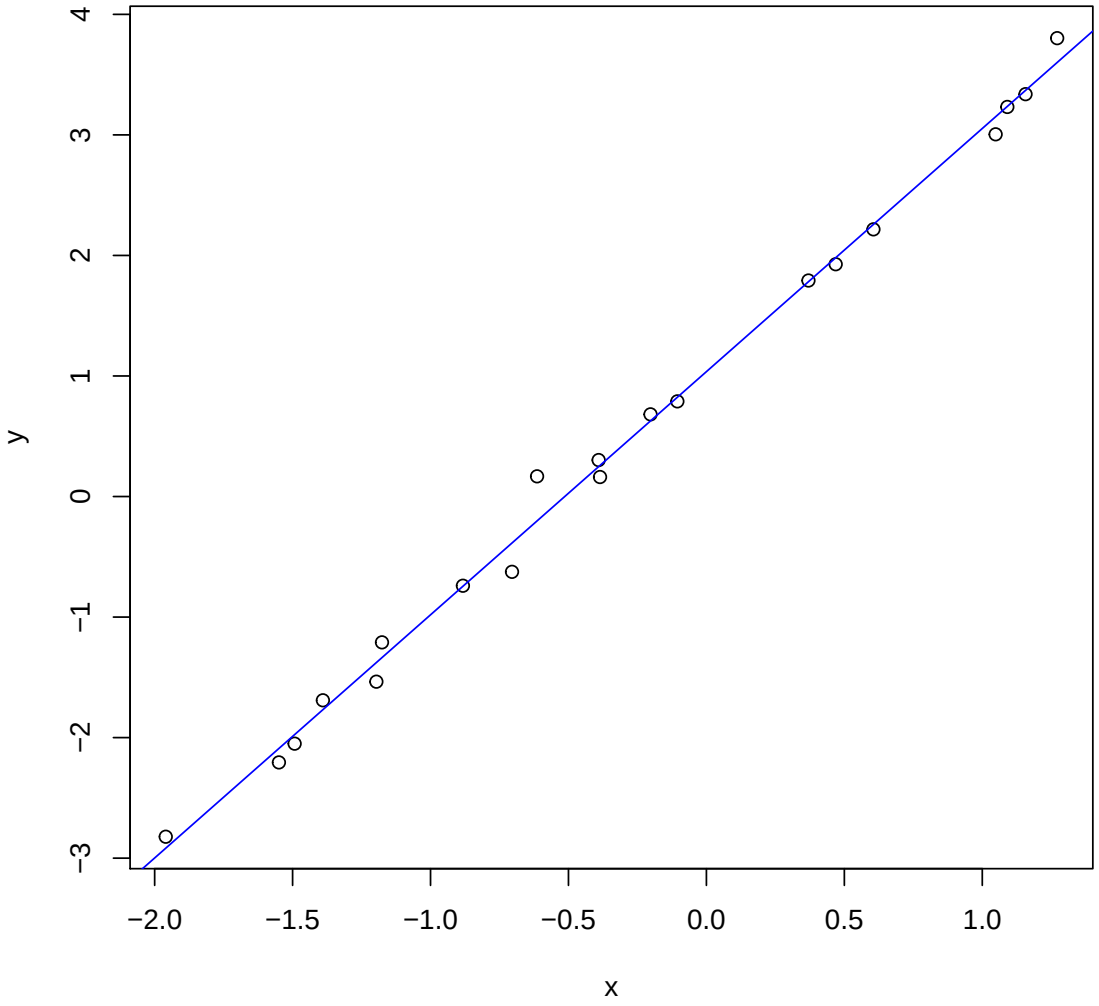
14. Adding a line to a plot

    The abline() command can add a regression line to an existing plot.

```
#first make up some data
x <- rnorm(20)
x

##  [1]  0.4686083 -1.3902273  0.3698862 -1.5495246  1.2714747 -0.7046330
##  [7]  0.6054740  1.0484313 -0.3906246 -0.2026367 -1.4925535 -1.9597786
## [13]  1.0907776 -0.3855741 -0.8826151 -1.1962779 -0.6138209  1.1567586
## [19] -1.1759474 -0.1052528

y <- x*2 + 1 + rnorm(20,0,.2)
y

##  [1]  1.9270544 -1.6904395  1.7920012 -2.2058168  3.8025641 -0.6246542
##  [7]  2.2165078  3.0048067  0.3025044  0.6817174 -2.0502310 -2.8223069
## [13]  3.2318877  0.1617660 -0.7401589 -1.5360202  0.1674599  3.3379471
## [19] -1.2098149  0.7894958

#generate plot
plot(x,y)
#superimpose regression line
abline(lm(y~x),col="blue")
```




```
#same plot using structural model notation
plot(y ~ x)
#superimpose regression line
abline(lm(y~x),col="blue")
```



One can superimpose multiple lines or ablines onto an existing plot. This is helpful to illustrate multiple groups or multiple regression models.

15. Splitting observations into strata, so that analyses will be repeated for each stratum separately.

    The by() command is helpful for this.

```
x <- rnorm(20)
gender <- c(rep("male",10),rep("female",10))
mean(x)

## [1] 0.08666227

by(x,gender,mean)
```

```
## gender: female
## [1] -0.04544627
## ------------------------------------------------------------
## gender: male
## [1] 0.2187708
```

16. Using "if then" logic

    The general syntax is "if (logical condition) commands else commands".

```
x <- 5
if (x==5) print("hello") else print("good bye")


## [1] "hello"
```

    It is possible to include multiple commands by using curly brackets as in

```
if (x==5) {
  print("hello")
  hist(y)
  median(z)
} else {
  print("good bye")
  table(gender)
}
```

    I formatted the spacing and line breaks to make it easier to read. R does not care about spaces and line breaks.

    If there is no else, it can be omitted. So fine to just say the following if you only want something to happen if the condition is TRUE.

```
x <- 6
if (x==5) print("hello")
```

17. Reading characters in a data file

    Fine to have characters or strings, just be sure to include them in quotation marks.

```
names <- c("rich","josh","brian")
names


## [1] "rich"  "josh"  "brian"


names <- c(names, "first last")
names


## [1] "rich"       "josh"       "brian"      "first last"
```

18. Factors

I recommend defining as factors() variables that will be used as factors in analyses. For example, you may have a factor that takes on levels low, medium and high. You code the three groups as 1, 2, and 3, respectively. Some R functions will not know whether the variable coded 1, 2 or 3 should be treated as a numeric variable or as a factor. Would a mean of a factor make any sense? In a regression, should the 1, 2, and 3 be treated as a single numeric predictor or as a factor with two orthogonal contrasts?

```
#set up a variable with three groups, 10 subjects each
group <- c(rep(1,10),rep(2,10),rep(3,10))
group

##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3

#does a mean make sense here?
mean(group)

## [1] 2

group.factor <- factor(group,labels=c("low","medium","high"),levels=1:3)
group.factor

##  [1] low    low    low    low    low    low    low    low    low    low
## [11] medium medium medium medium medium medium medium medium medium medium
## [21] high   high   high   high   high   high   high   high   high   high
## Levels: low medium high

#mean no longer computes, which makes sense
mean(group.factor)

## Warning in mean.default(group.factor):  argument is not numeric or logical:
returning NA

## [1] NA

#contrasts are defined for the factor but not the variable
contrasts(group.factor)

##        medium high
## low         0    0
## medium      1    0
## high        0    1

contrasts(group)
```

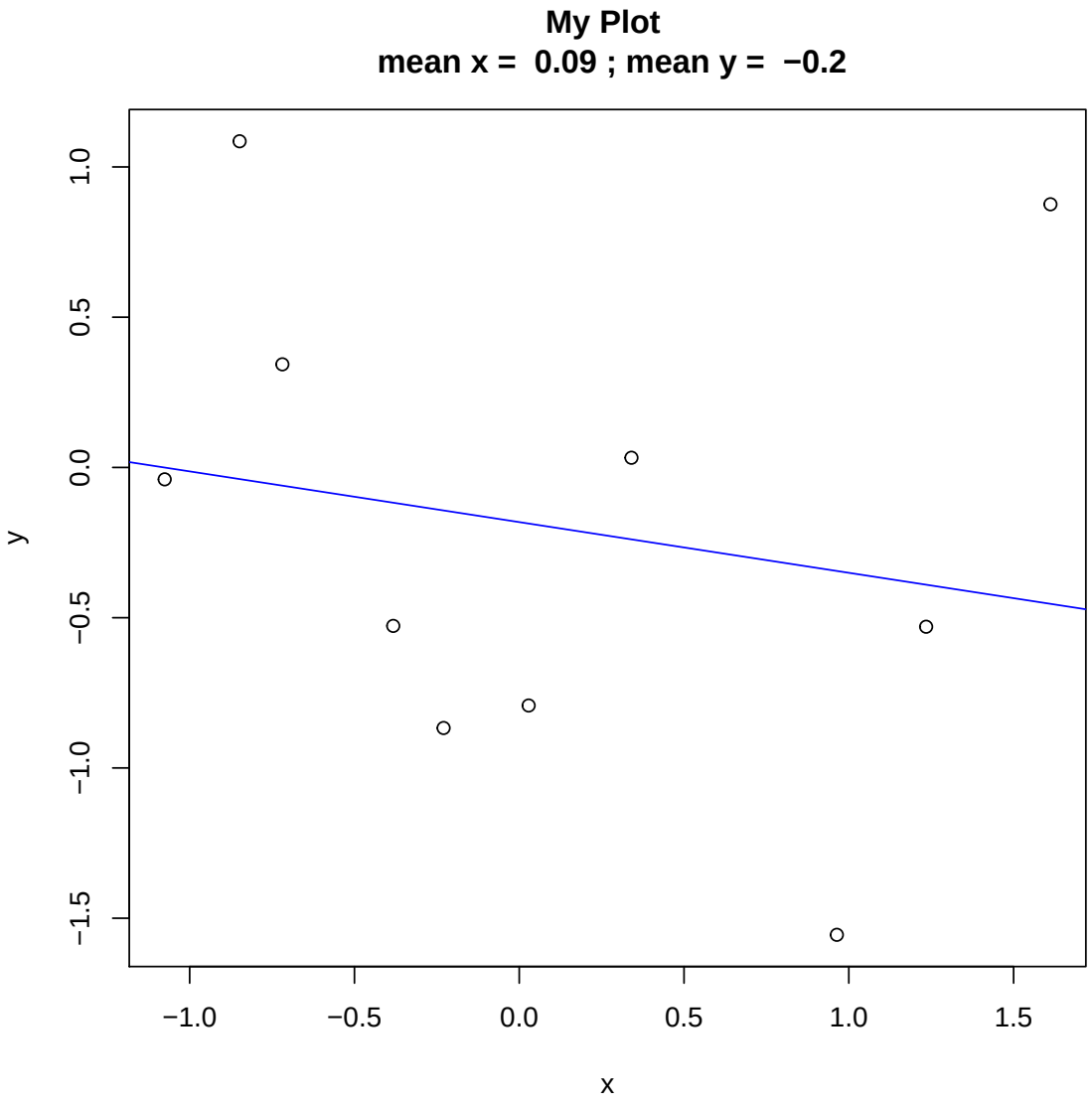
```
## Error in contrasts(group):  contrasts apply only to factors

#R chooses dummy contrasts by default, you can specify your own orthogonal set as in
newset <- cbind(c(1,0,-1),c(1,-2,1))
colnames(newset) <- c("C1:low v high", "C2:medium v average low and high")
contrasts(group.factor) <- newset
contrasts(group.factor)

##        C1:low v high C2:medium v average low and high
## low                1                                1
## medium             0                               -2
## high              -1                                1
```

19. Defining functions

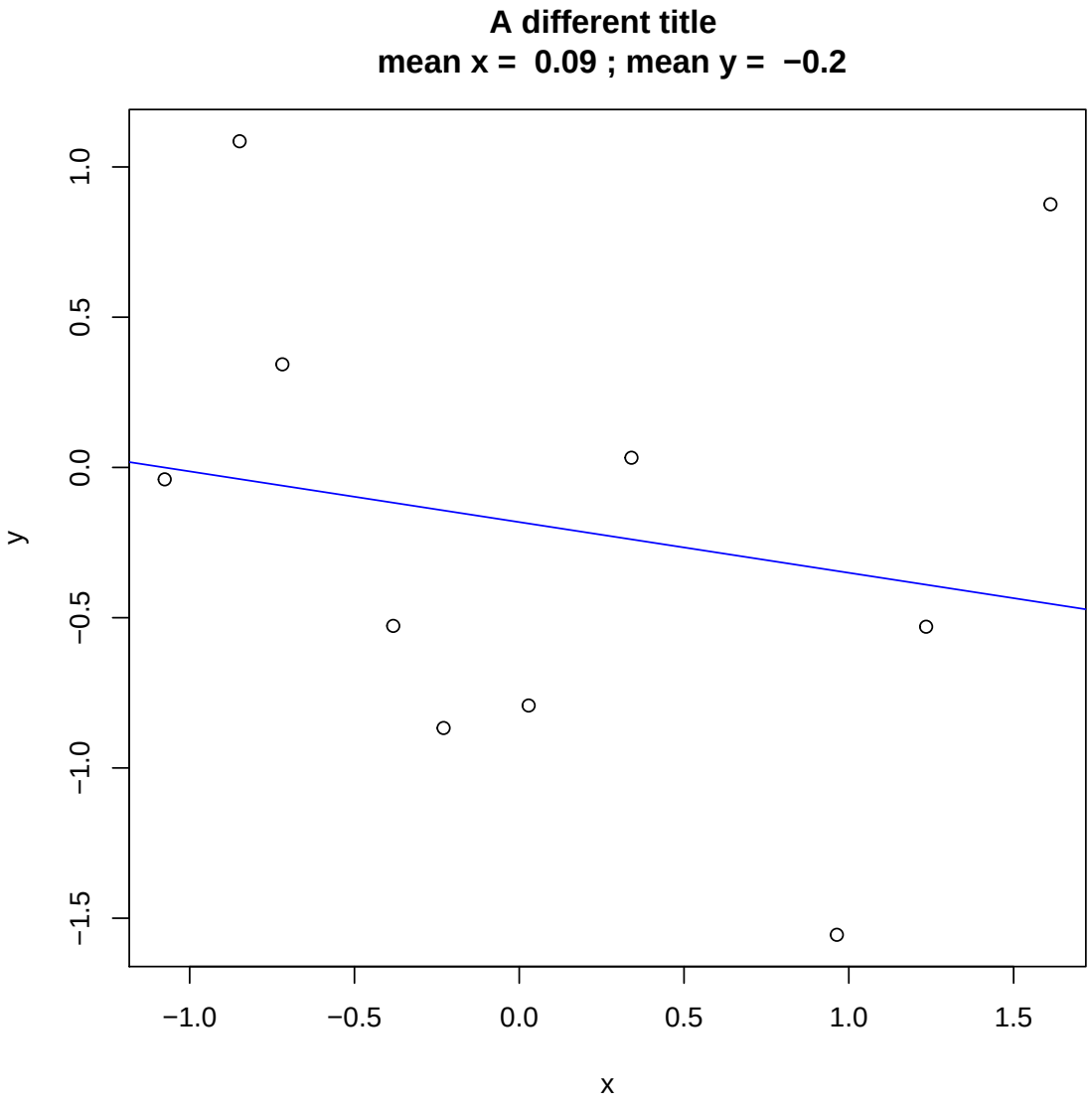
    A convenient feature in R is that you can collect several commands into a function that you can then use repeatedly.

    This function illustrates the basic outline. It takes two variables, produces a scatterplot, superimposes the regression line, and prints the means in title. This example illustrates how to setup default action (such as the title) and make it something that you can edit as needed in future function calls.

```
#a simple function; 2nd and 3rd line print the variance of each variance
myplot <- function(x,y,title.firstline="My Plot") {
print(var(x))
print(var(y))
plot(x,y)
abline(lm(y~x),col="blue")
title(paste(title.firstline,"\n","mean x = ", round(mean(x),2),"; mean y = ", round(mean(
}

#now create some data for illustration
var1 <- rnorm(10)
var2 <- rnorm(10)
myplot(var1,var2)

## [1] 0.8527829
## [1] 0.6702595
```




```
#same function if I want to use new variables, or a different first line
myplot(var1,var2,title.firstline="A different title")


## [1] 0.8527829
## [1] 0.6702595
```


A different title
mean x = 0.09 ; mean y = −0.2
y
x
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5