**Introduction to R – Example**



Example: GPA data (GPA.R, gpa.txt, gpa.csv)

Suppose a random sample of size 20 was taken from the population of college students at a university. We would like to use the sample to examine the relationship between high school GPA (HS.GPA) and undergraduate College GPA (College.GPA).

Below is part of the code as it appears after being run in R. Note that I often need to fix the formatting to make it look “pretty” here. You are expected to do the same for any assignments!

> #########################################################

> # Simple data analysis example in R using the gpa data #

> # set #

> #########################################################

>

> # Read in the data

> gpa <- read.table(file = "C:\\data\\gpa.txt",

header = TRUE, sep = "")

> # Print data set

> gpa

HS.GPA College.GPA

1 3.04 3.10

2 2.35 2.30

3 2.70 3.00

4 2.55 2.45

5 2.83 2.50

6 4.32 3.70

7 3.39 3.40

8 2.32 2.60

9 2.69 2.80

10 2.83 3.60

11 2.39 2.00

12 3.65 2.90

13 2.85 3.30

14 3.83 3.20

15 2.22 2.80

16 1.98 2.40

17 2.88 2.60

18 4.00 3.80

19 2.28 2.20

20 2.88 2.60

> # This is how to read in a comma delimited file

> gpa4 <- read.csv(file = "gpa.csv")

> # Access parts of the data set

> names(gpa)

[1] "HS.GPA" "College.GPA"

> gpa$HS.GPA

[1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39

3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88

> gpa$College.GPA

[1] 3.10 2.30 3.00 2.45 2.50 3.70 3.40 2.60 2.80 3.60 2.00

2.90 3.30 3.20 2.80 2.40 2.60 3.80 2.20 2.60

> gpa[,1]

[1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39

3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88

> gpa[, "HS.GPA"]

[1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39 3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88

> gpa[1,1] # row 1 and column 1 value

[1] 3.04

> gpa[1:10,1] # first 10 observations of variable 1

[1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83

> # gpa[, c("HS.GPA", "College.GPA")] # Whole data set

> # Summary statistics for variables

> summary(gpa)

HS.GPA College.GPA

Min. :1.980 Min. :2.000

1st Qu.:2.380 1st Qu.:2.487

Median :2.830 Median :2.800

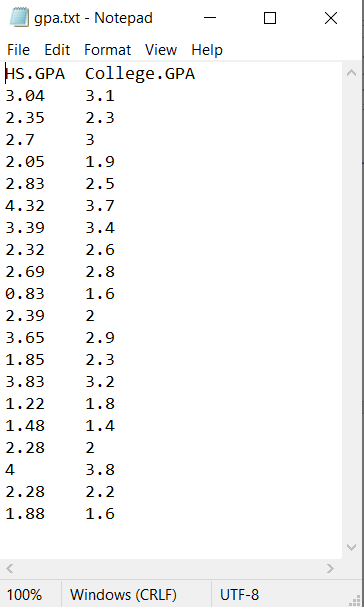
Mean :2.899 Mean :2.862

3rd Qu.:3.127 3rd Qu.:3.225

Max. :4.320 Max. :3.800

Notes:

* The # denotes a comment line in R.
* The gpa.txt file is an ASCII text file that looks like:

  
  
The read.table() function reads in the data and puts it into an object called gpa here. Notice the use of the “\\” between folder names. This needs to be used instead of “\”. Also, you can use “/” too. Since the variable names are at the top of the file, the header = TRUEoption is given. The sep = "" option specifies white space (spaces, tabs, …) is used to separate variable values. One can use sep = "," for comma delimited files with read.table() or the function read.csv() without the sep or header arguments.

* Another commonly used data format is an Excel file. The R Data Import/Export manual (select HELP > MANUALS (IN PDF)) provides options for how to read in Excel files; however, the manual says “The first piece of advice is to avoid doing so if possible!” Reasons for this recommendation are because of the different Excel file formats (.xls or .xlsx) and 32-bit vs. 64-bit driver issues.
* You can save data to a file outside of R by using the write.table() or write.csv() functions. Below is the code used to create a comma delimited file:

write.table(x = gpa, file = "gpa-out1.csv", quote =

FALSE, row.names = FALSE, sep =",")

write.csv(x = gpa, file = "gpa-out2.csv")

* The gpa object is an object type called a *data frame*.
* It is very important to learn how to access parts of a data frame. The code/output provides a number of examples. The most used way is through the following syntax:

Data frame $ variable

where spaces are removed and the actual names of the data frame and variable are used.

* The summary() function summarizes the information stored within an object. Different object types will produce different types of summaries. Examples later in the class will be given where thesummary()function did produce a different type of summary.

Scatter plot of the GPAs

> #Simple plot

> plot(x = gpa$HS.GPA, y = gpa$College.GPA)



> #Better plot

> plot(x = gpa$HS.GPA, y = gpa$College.GPA, xlab = "HS

GPA", ylab = "College GPA", main = "College GPA vs. HS

GPA", xlim = c(0,4.5), ylim = c(0,4.5), col = "red",

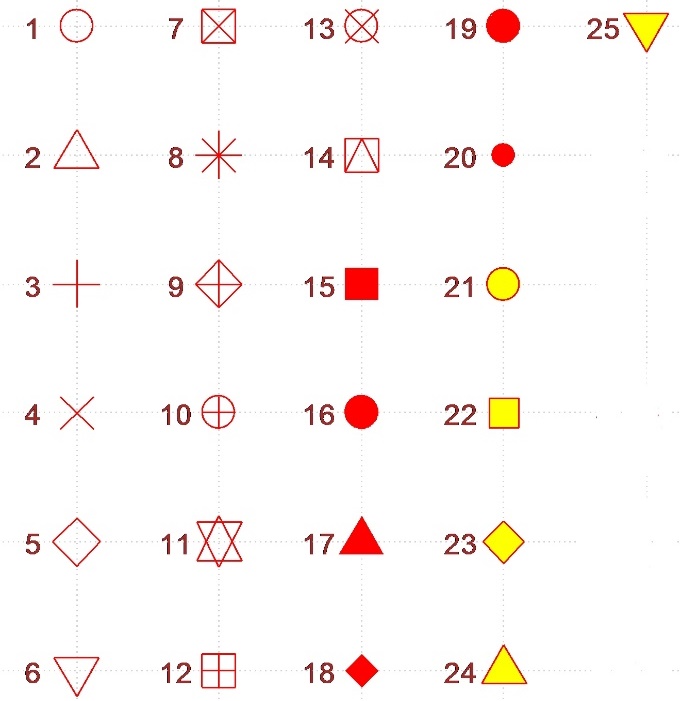
pch = 1, cex = 1.0, panel.first = grid(col = gray", lty

= "dotted"))



Notes:

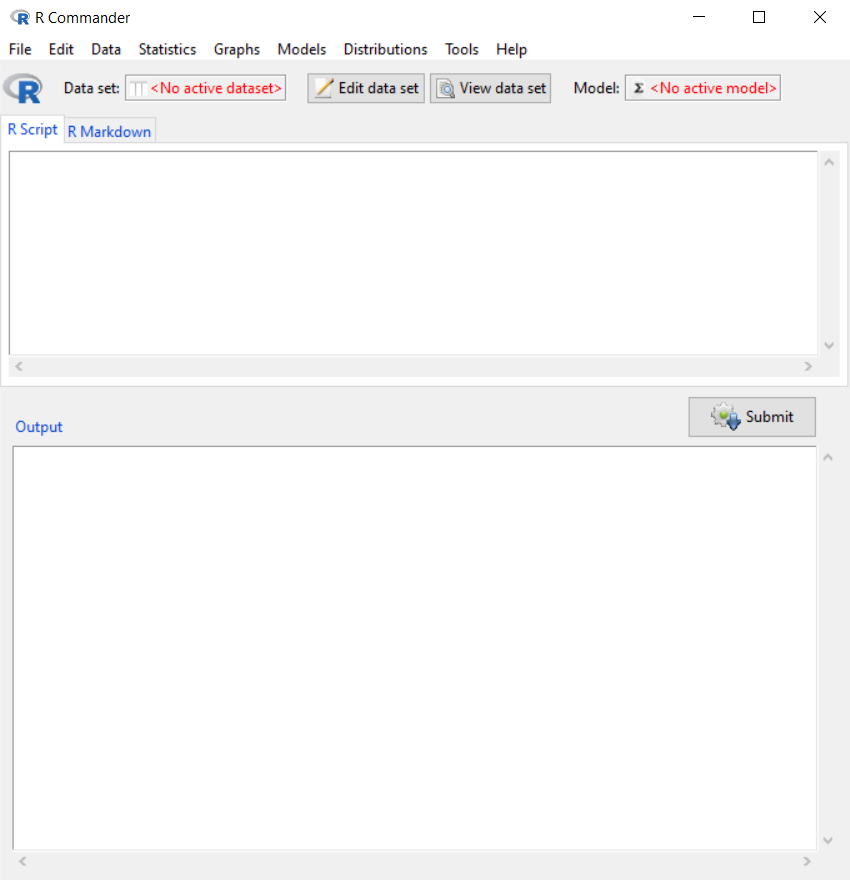
* The plot()function creates a two dimensional plot of data. Here are descriptions of its arguments:
  + x specifies what is plotted for the x-axis.
  + y specifies what is plotted for the y-axis.
  + xlab and ylab specify the x-axis and y-axis labels, respectively.
  + main specifies the main title of the plot.
  + xlim andylim specifythe x-axis and y-axis limits, respectively. Notice the use of the c() function.
  + col specifies the color of the plotting points. Run the colors() function to see what possible colors can be used. Also, you can see <https://github.com/EarlGlynn/colorchart/wiki/Color-Chart-in-R> for the colors from colors().
  + pch specifies the plotting characters. Below is a list of possible characters.



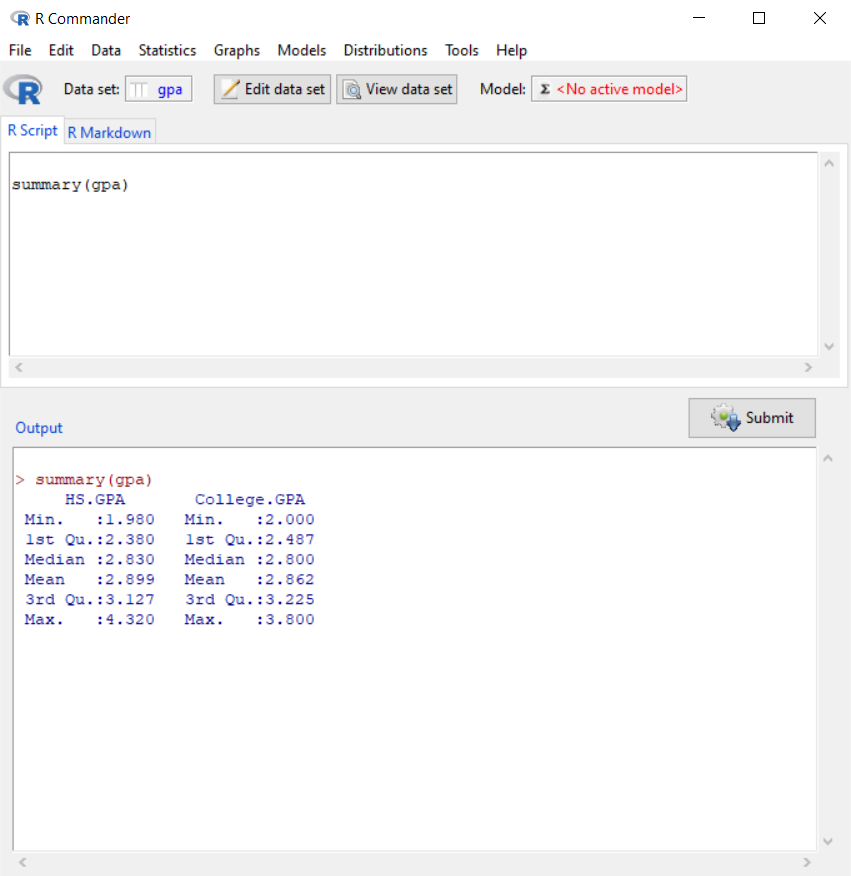
* + cex specifies the height of the plotting characters. The value 1.0 is the default.
  + panel.first = grid()specifies grid lines will be plotted. The line types can be specified as follows: 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash". These line type specifications can be used in other functions.
  + The par()function’s Help contains more information about the different plotting options!
* The plot can be brought into Word easily. In R, make sure the plot window is the current window and then select FILE > COPY TO THE CLIPBOARD > AS A METAFILE. Select the PASTE button in Word to paste it.



Are there any point-and-click methods to use in R to do the analyses here? Yes – Rcmdr (short for “R Commander”) is a package that allows for some point-and-click calculations. This package does not come downloaded with the initial installation of R so you will need to install it (will take a little bit of time because it automatically installs other packages as well). Once the package is installed, simply use library(package = Rcmdr) to start it. Below is what it looks like:



One of the nice things about R Commander is that you can use it to help learn the code through using its point-and-click interface. To begin, you need to specify the data set of interest. Since gpa already exists in my current R session, I choose this data set by selecting DATA > ACTIVE DATA SET > SELECT ACTIVE DATASET. Now the “Data set: gpa” is shown toward the top of the R Commander window. To find summary statistics like we did before, select STATISTICS > SUMMARIES > ACTIVE DATA SET to produce the following:



Notice how R uses the summary() function just like we did before to print summary statistics for each variable. The SCRIPT window keeps track of this code. If you would like to save this code, select FILE > SAVE SCRIPT.

Explore the menus on your own to examine the resources available! Note that HELP > INTRODUCTION TO THE R COMMANDER within R Commander opens a PDF file on getting started with it.

Next, let’s estimate a regression model to the data using high school GPA to predict college GPA

> mod.fit <- lm(formula = College.GPA ~ HS.GPA, data = gpa)

> #Brief look of what is inside of mod.fit

> mod.fit

Call:

lm(formula = College.GPA ~ HS.GPA, data = gpa)

Coefficients:

(Intercept) HS.GPA

1.0869 0.6125

> #See the names of all components in the object

> names(mod.fit)

[1] "coefficients" "residuals" "effects"

[4] "rank" "fitted.values" "assign"

[7] "qr" "df.residual" "xlevels"

[10] "call" "terms" "model"

> mod.fit$coefficients

(Intercept) HS.GPA

1.0868795 0.6124941

> round(mod.fit$residuals[1:5],2)

1 2 3 4 5

0.15 -0.23 0.26 -0.20 -0.32

> #Put some of the components into a data.frame object

> save.fit <- data.frame(gpa, C.GPA.hat =

round(mod.fit$fitted.values,2), residuals =

round(mod.fit$residuals,2))

> #Print contents of save.fit

> save.fit

HS.GPA College.GPA C.GPA.hat residuals

1 3.04 3.10 2.95 0.15

2 2.35 2.30 2.53 -0.23

3 2.70 3.00 2.74 0.26

4 2.55 2.45 2.65 -0.20

5 2.83 2.50 2.82 -0.32

6 4.32 3.70 3.73 -0.03

7 3.39 3.40 3.16 0.24

8 2.32 2.60 2.51 0.09

9 2.69 2.80 2.73 0.07

10 2.83 3.60 2.82 0.78

11 2.39 2.00 2.55 -0.55

12 3.65 2.90 3.32 -0.42

13 2.85 3.30 2.83 0.47

14 3.83 3.20 3.43 -0.23

15 2.22 2.80 2.45 0.35

16 1.98 2.40 2.30 0.10

17 2.88 2.60 2.85 -0.25

18 4.00 3.80 3.54 0.26

19 2.28 2.20 2.48 -0.28

20 2.88 2.60 2.85 -0.25

> #Summarize the information stored in mod.fit

> summary(object = mod.fit)

Call:

lm(formula = College.GPA ~ HS.GPA, data = gpa)

Residuals:

Min 1Q Median 3Q Max

-0.55074 -0.25086 0.01633 0.24242 0.77976

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.0869 0.3666 2.965 0.008299 \*\*

HS.GPA 0.6125 0.1237 4.953 0.000103 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3437 on 18 degrees of freedom

Multiple R-squared: 0.5768, Adjusted R-squared: 0.5533

F-statistic: 24.54 on 1 and 18 DF, p-value: 0.0001027

Notes:

* The lm() function fits the simple linear regression model. The results are stored in an object called mod.fit. Notice the use of the ~ to separate the dependent and independent variables. If there were multiple independent variables, the + sign can be used to separate them.
* By entering the mod.fit object name only, R prints a little information about what is inside of it.
* To see a list of the components inside of mod.fit, use the names()function. To access a component of an object, use the $ sign with the name of the component.
* To create your own data frame, you can use the data.frame() function. Notice how you can specify the names of columns inside of the data.frame().
* The summary()function can be used with themod.fit object to summarize the contents of mod.fit. Notice the different results from what we had before with summary()!
* The estimated regression model is



> #########################################################

> # Put regression line on plot

> # Open a new graphics window

> dev.new(width = 8, height = 6, pointsize = 10)

> # 1 row and 2 columns of plots

> par(mfrow = c(1,2))

> # Same scatter plot as before

> plot(x = gpa$HS.GPA, y = gpa$College.GPA, xlab = "HS

GPA", ylab = "College GPA", main = "College GPA vs.

HS GPA", xlim = c(0,4.5), ylim = c(0,4.5), col =

"red", pch = 1, cex = 1.0, panel.first = grid(col =

"gray", lty = "dotted"))

> # Puts the line y = a + bx on the plot

> abline(a = mod.fit$coefficients[1], b =

mod.fit$coefficients[2], lty = "solid", col =

"blue", lwd = 2)

> # Same scatter plot as before

> plot(x = gpa$HS.GPA, y = gpa$College.GPA, xlab = "HS

GPA", ylab = "College GPA", main = "College GPA vs. HS

GPA", xlim = c(0,4.5), ylim = c(0,4.0),

panel.first=grid(col = "gray", lty = "dotted"))

> # Add line

> curve(expr = mod.fit$coefficients[1] +

mod.fit$coefficients[2]\*x, xlim =

c(min(gpa$HS.GPA),max(gpa$HS.GPA)), col= "blue", add =

TRUE, lwd = 2)



Notes:

* The dev.new() function can be used to open a new plotting window.
* Theabline() function can be used to draw straight lines on a plot. In the format used here, the line y = a + bx was drawn where a was the  (intercept) and b was the  (slope).
* In the second plot, the curve()function was used to draw the line on the plot. This was done to have the line within the range of the high school GPA values.

Below is a function written to help automate the analysis.

my.reg.func <- function(x, y, data) {

*# Fit the simple linear regression model and save the*

*results in mod.fit*

mod.fit <- lm(formula = y ~ x, data = data)

*#Open a new graphics window - do not need to*

dev.new(width = 6, height = 6, pointsize = 10)

*# Same scatter plot as before*

plot(x = x, y = y, xlab = *"x"*, ylab = *"y"*, main = *"y*

*vs. x"*, panel.first=grid(col = *"gray"*, lty =

*"dotted"*))

*# Plot model*

curve(expr = mod.fit$coefficients[1] +

mod.fit$coefficients[2]\*x, xlim = c(min(x),max(x)),

col = *"blue"*, add = TRUE)

*# This is the object returned*

mod.fit

}

Run the function:

> save.it <- my.reg.func(x = gpa$HS.GPA, y =

gpa$College.GPA, data = gpa)



> names(save.it)

[1] "coefficients" "residuals" "effects"

[4] "rank" "fitted.values" "assign"

[7] "qr" "df.residual" "xlevels"

[10] "call" "terms" "model"

> summary(save.it)

Call:

lm(formula = y ~ x, data = data)

Residuals:

Min 1Q Median 3Q Max

-0.55074 -0.25086 0.01633 0.24242 0.77976

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.0869 0.3666 2.965 0.008299 \*\*

x 0.6125 0.1237 4.953 0.000103 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3437 on 18 degrees of freedom

Multiple R-squared: 0.5768, Adjusted R-squared: 0.5533

F-statistic: 24.54 on 1 and 18 DF, p-value: 0.0001027

Final notes:

* Typing a function name only, like sd, will show you the actual code that is used by the function to do the calculations! This can be useful when you want to know more about how a function works or if you want to create your own function by modifying the original version. Note that for new users of R, reading the code can often be difficult. However, remember that you could execute each line one-by-one to see what it does!
* Please remember to always use the Help if you do not understand a particular function!
* To get specific x-axis or y-axis tick marks on a plot, use the axis()function. For example,

#Note that xaxt = "n" tells R to not give any labels on the

# x-axis (yaxt = "n" works for y-axis)

plot(x = gpa$HS.GPA, y = gpa$College.GPA, xlab = "HS GPA",

ylab = "College GPA", main = "College GPA vs. HS GPA",

xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =

"red", pch = 1)

#Major tick marks

axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.5))

#Minor tick marks

axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.1), tck

= 0.01, labels = FALSE)

* A large community of R users exist which has led to many sources for help outside of R.
  + A simple web search frequently leads to question/answer websites, such as Stack Overflow, that provide the needed help. In particular for Stack Overflow, a web page is available at <http://stackoverflow.com/tags/r/info> that lists questions/answers tagged with R.
  + There are a large number of blogs devoted to the use of R. The R-bloggers website at <http://www.r-bloggers.com> serves as an aggregator for many of these blogs.
  + There are active listservs (<http://www.r-project.org/mail.html>) to answer submitted questions.