ST512 Lab 1

Spring 2022

Getting started in R

The first order of business for this unit is to get up and running in R. You'll need to do several things. First, download and install R on your computer. You can find the link on the Computing Resources area of the course webpage. Second, most R users today interact with R through a front-end called RStudio. Download and install RStudio; again, the link is in the Computing Resources area of the course webpage.

This document that you are reading is created with software called R Markdown. R Markdown is a somewhat more advanced skill, so it's not the emphasis of this set of exercises. If you plan to use R professionally, however, R Markdown is a nice tool for preserving an archive of your analysis. It allows you to weave together text and R code, creating a file that will show others (including your future self!) exactly how the results that you generated were produced. We'll post the source code for this document online; students who are already familiar with R may wish to study the code used to generate this key to see how R Markdown works. You can run R Markdown from within R Studio. For more details on using R Markdown see http://rmarkdown.rstudio.com.

If you're completely new to R, you'll want to spend some time becoming acquainted with the working environment. The internet abounds with tools to help you learn R. On the course webpage, you'll find a document by Torfs & Brauer titled "A (very) short introduction to R' '. I find this especially helpful, even if it is becoming a bit dated. If you are completely new to R, read sections 3–9 of that document now. (Sections 1–2 cover installation. Sections 10–12 are more advanced than we'll need for ST 512, but do read them if you are so motivated.) If you have experience with R, browse sections 1–9 to see if it contains any new information that you didn't already know.

Reading in data

R works by holding objects in memory. There are many types of R objects, including vectors, arrays, and functions. The basic R object for storing data is a data frame. A data frame is very similar to a matrix, with one row for each subject and one (named) column for each observation.

It is possible to construct small data frames directly from the command line. However, it is usually more efficient to build the data set in another program (e.g., Excel), save it as a text file, and then read the text file in directly to R.

In this analysis we will consider a study in which researcher took measurements on 247 men and 260 women, for a total of $n = 507$ data points. These were primarily individuals in their twenties and early thirties, with a scattering of older men and women, all physically active (several hours of exercise a week). The measurements were taken at San Jose State University and at the U.S. Naval Postgraduate School in Monterey, California and in dozens of California health and fitness clubs. The data set "body' ' contains the following variables:

- Age (years)
- Body weight (kg)
- Height (cm)
- Gender

This data set can be found on the course webpage, in the Course Datasets folder.

In RStudio, there are two basic strategies for importing data. Both of these strategies begin with saving the data set to a text file on your computer. Once you have downloaded the data file and saved it in an appropriate place, there are two options for reading the data into memory. Both are described below.

Option 1: Read the data into RStudio by using the 'read.table' command at the command line. That is, in the "Console'' window of RStudio, enter the command below.

```
body <- read.table("body.txt", header = TRUE, stringsAsFactors = TRUE)
```

Note that in this case the file "body.txt'' is already in the working directory. (You can find the working directory by using the command

```
getwd()
```

If you've placed the file "body.txt" somewhere other than the active directory, then you'll need to specify the full pathname. For example, if you've saved the file in "C:/Users/ST512/body.txt'', then we would use

```
body <- read.table("C:/Users/ST512/body.txt", header = TRUE, stringsAsFactors = TRUE)
```

The optional argument header = TRUE is included in the read.table command above to tell R that the first line of the file is a "header" line that contains column names.

Option 2: Use the Import Dataset wizard. The latest release of RStudio gives you two options for reading data into memory: via a 'base' program, and via a more sophisticated program called 'readr'. I find the 'base' program to be easier to use, and 'readr' to be a bit clunky, but you are welcome to use whichever program you prefer. In any event, look for the 'Import Dataset' button on the Workspace tab (usually in the upper right) to launch the import wizard. Choose either 'From Text (base)...' or 'From Text (readr)...' to launch a dialog box based on either the 'base' or 'readr' program, respectively. If you use the 'base' program, the program should recognize the different features of the data file correctly. If you use the 'readr' program, you'll need to change the delimiter from Comma to Tab. Note also that the 'readr' program will not correctly recognize the gender variable as a factor, so you'll have to use the graphical interface to change the variable type from 'character' to 'factor'. When you do so, RStudio will prompt you to enter the 'factors'. This is a typo in the dialog box. The dialog box should prompt you to enter the 'levels' (a level is a unique value of a factor). In this case, the two levels are 'Male' and 'Female'.

R now has a data frame in memory called "body" that contains these data. Confirm this by using the ls() command (for 'list') to ask for a list of objects in memory. You should see something similar to:

```
ls()
```

```
## [1] "body"
```

In RStudio, you should see the data set listed in the Workspace window.

To confirm that the data were entered correctly, ask for a summary of "body". Try

```
summary(body)
```

```
##       age            weight           height         gender
##  Min.   :18.00   Min.   : 42.00   Min.   :147.2   Female:260
##  1st Qu.:23.00   1st Qu.: 58.40   1st Qu.:163.8   Male  :247
##  Median :27.00   Median : 68.20   Median :170.3
##  Mean   :30.18   Mean   : 69.15   Mean   :171.1
##  3rd Qu.:36.00   3rd Qu.: 78.85   3rd Qu.:177.8
##  Max.   :67.00   Max.   :116.40   Max.   :198.1
```

"Summary" is a useful, all-purpose command for learning about a data frame. You may note that the summary data provided for the quantitative variables (age, weight, height) differs from the summary data for gender, because gender here is treated as a categorical predictor. (We'll talk more about categorical predictors soon.)

Accessing individual variables in a data frame

The data frame 'body' contains four separate variables. The syntax for accessing each of these variables is

```
data.frame$variable
```

For example, if we want to find the average age of individuals in this data set, we can do so with the command

```
mean(body$age)
```

```
## [1] 30.18146
```

How do you think you would calculate the variance of the heights in these data? If you guessed

```
var(body$height)
```

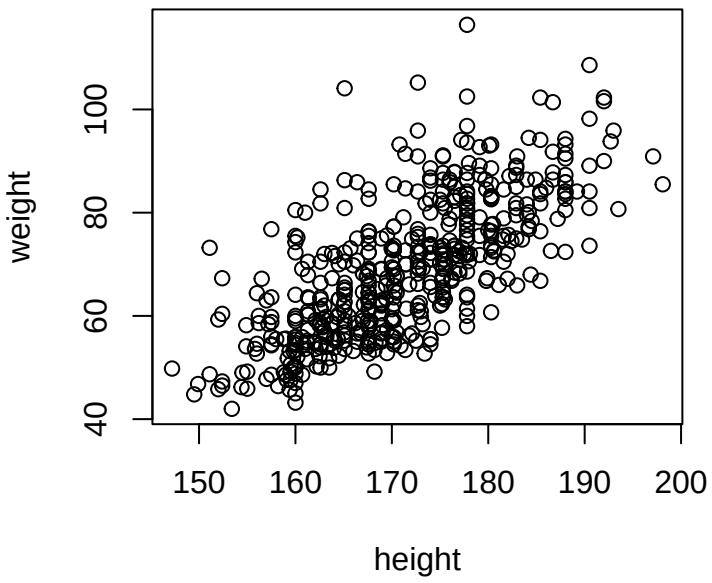
```
## [1] 88.49551
```

then well done!

Fitting a simple regression model

Now let's explore the relationship between height and weight. Let's first plot the relationship between height and weight, with height on the horizontal axis and weight on the vertical axis. Do so using the "plot'' command:

```
plot(weight ~ height, data = body)
```



In the "plot" command, the first argument is a formula in the form "y ~ x''. The second argument gives the data set in which the variables to be plotted are found. There are many ways to skin the proverbial cat in this case. For example, we also might have tried

```
plot(body$weight ~ body$height)
```

which would have worked just as well (why?).

Now let's fit a simple regression model with height as the predictor and weight as the response. The basic program for fitting regression models (both simple regression and multiple regression) is "lm' ', which stands for [l]inear [m]odel. The basic format for a call to lm is to supply a formula that specifies the model first, and the data set that contains the variables second. So, to fit our model, we would use

```
slr1 <- lm(weight ~ height, data = body)
```

In this case, we want to keep the model in memory so that we can analyze it later without having to refit it. So, we have stored the fitted model in memory under the name "slr1' '. The ever-handy summary command tells us a bit about our model fit:

```
summary(slr1)
```

```
##
## Call:
## lm(formula = weight ~ height, data = body)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.743  -6.402  -1.231   5.059  41.103
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -105.01125    7.53941  -13.93   <2e-16 ***
## height         1.01762    0.04399   23.14   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.308 on 505 degrees of freedom
## Multiple R-squared:  0.5145, Adjusted R-squared:  0.5136
## F-statistic: 535.2 on 1 and 505 DF,  p-value: < 2.2e-16
```

Here we find a variety of useful information about our model. For example, the "Coefficients" table lists the least-squares estimates of the intercept and slope:

$$\hat{\beta}_0 = -105.0 \quad \hat{\beta}_1 = 1.018$$
$$s_{\hat{\beta}_0} = 7.5 \quad s_{\hat{\beta}_1} = 0.044$$

Again, we don't pay too much mind to the estimate of the intercept here. While we need to know the value of the estimate to construct the regression line, it's obviously silly to think about the weight of a person that is 0 cm tall, and so we don't try to assign a scientific interpretation to the intercept.

We also get t-tests for whether each of these parameters equals 0. In this case, we see that the t-test for the null hypothesis that the slope equals 0 vs. the two-sided alternative that the slope is not equal to 0 gives a t-statistic of $t_{505} = 23.14$, which corresponds to a very small p-value, less than 2×10^{-16}! Thus, not surprisingly, there is overwhelming evidence of a significant linear association between height and weight.

Let's suppose that we wanted to calculate a 90% confidence interval for the slope. The "summary" already gives us the estimate and the standard error, so all we need now is the appropriate critical value. We can find this critical value from the on-line calculator discussed in lecture (you'll find a link on the course webpage). Alternatively, we can use the "qt" function in R. In this case, the function is called "qt" because we want to find a [q]uantile from a [t]-distribution. If we want a 90% confidence interval, then we'll need to find the quantile that cuts off a $\frac{10\%}{2}$ or 5% tail, which we find as

```
qt(.05, df = 505, lower = FALSE)
```

```
## [1] 1.647877
```

Thus the confidence interval is

$$\begin{aligned} \widehat{\beta_1} \pm t_{n-2,\alpha/2} \times s_{\widehat{\beta_1}} &= 1.018 \pm 1.648 \times 0.044 \\ &= (0.945, 1.090) \end{aligned}$$

Or we could get the confidence intervals directly using `confint` function

```
confint(slr1, level = 0.9)
```

```
##                      5 %       95 %
## (Intercept) -117.435270 -92.587238
## height         0.945132   1.090102
```

Finally, the summary of the model fit also gives the coefficient of determination as $R^2 = 0.5145$. If we wish, we can find the sums-of-squares breakdown that leads to that value using the "anova'' command:

```
anova(slr1)
```

```
## Analysis of Variance Table
##
## Response: weight
##            Df Sum Sq Mean Sq F value    Pr(>F)
## height      1  46370   46370  535.21 < 2.2e-16 ***
## Residuals 505  43753      87
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This output doesn't give us the total sum-of-squares, but it's easy to find:

$$SSE = 43753$$
$$SST = SSE + SSM = 43753 + 46370 = 90123$$

To make another connection, recall that the mean squared error (MSE) serves as a good estimate of the residual variance, σ_ε^2. We usually write this estimate as s_ε^2. So, in this case, our estimate of the residual variance is

$$s_\varepsilon^2 = \frac{43753}{507 - 2} = 86.6.$$

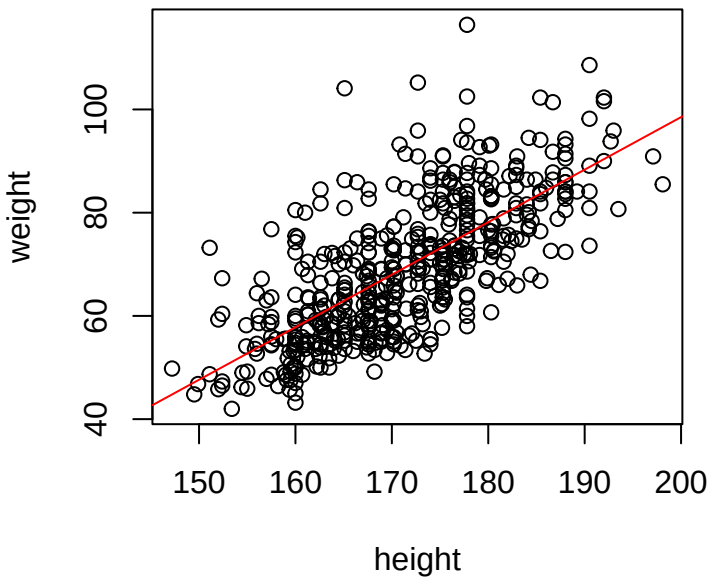
To find the residual standard deviation, just take the square root, or the root mean-squared error (RMSE)

$$s_\varepsilon = \sqrt{86.6} = 9.31.$$

This is a helpful number, because it gives us an idea of how much variation there is in the weights of people at any given height. In the "summary" of the model, this is the quantity that R labels as the "residual standard error".

We can also use our model fit to overlay a fitted regression line onto the scatter plot of the two variables. First we'll recreate the scatter plot using the `plot'' command, and then we'll overlay the regression line using the`abline'' command.

```
plot(weight ~ height, data = body)
abline(slr1, col = "red")
```



Change of units

A good way to build understanding of the parameters in a regression model is to consider how the estimates change if the units of either variable change. So far, we've been working with metric units: body weight in kilograms and height in centimeters. Let's suppose that we convert to US customary units, with body weight in pounds and height in inches. Remember that 1 kg is equivalent to 2.2 lbs, and 2.54 cm is equivalent to 1 inch.

First we'll create a new variable (weight.lb) that is the body weight in pounds and a new variable (height.in) that is height in inches. Each of these commands creates a new variable within the data frame 'body' by applying a mathematical operation to a variable in that data frame.

```
body$weight.lb <- body$weight * 2.2
body$height.in <- body$height / 2.54
```

Ask for a `summary` of the data frame "body" to confirm that you have added these new variables correctly.

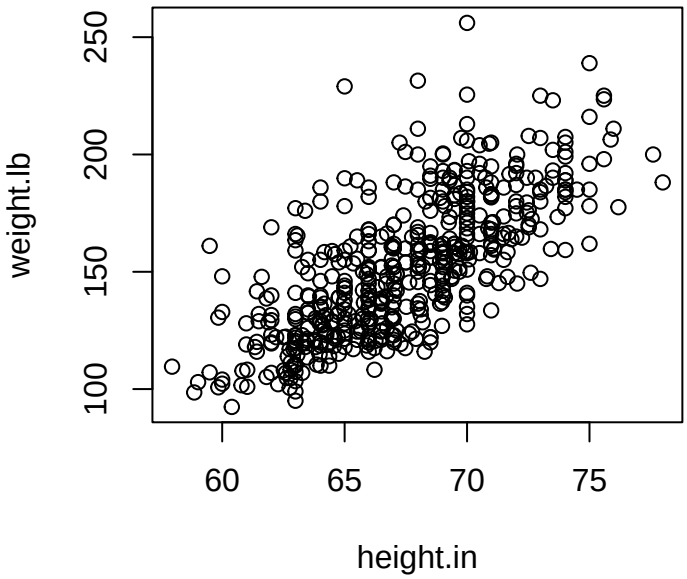
```
summary(body)
```

```
##       age            weight          height         gender      weight.lb
##  Min.   :18.00   Min.   : 42.00   Min.   :147.2   Female:260   Min.   : 92.4
##  1st Qu.:23.00   1st Qu.: 58.40   1st Qu.:163.8   Male  :247   1st Qu.:128.5
##  Median :27.00   Median : 68.20   Median :170.3                Median :150.0
##  Mean   :30.18   Mean   : 69.15   Mean   :171.1                Mean   :152.1
##  3rd Qu.:36.00   3rd Qu.: 78.85   3rd Qu.:177.8                3rd Qu.:173.5
##  Max.   :67.00   Max.   :116.40   Max.   :198.1                Max.   :256.1
##    height.in
##  Min.   :57.95
##  1st Qu.:64.49
##  Median :67.05
##  Mean   :67.38
```

```
##  3rd Qu.:70.00
##  Max.   :77.99
```

We can plot weight in pounds vs. height in inches:

```
plot(weight.lb ~ height.in, data = body)
```



Compare this plot to our previous plots.

Now we'll fit a regression where `weight.lb` is the response and `height.in` is the predictor.

```
slr2 <- lm(weight.lb ~ height.in, data = body)
summary(slr2)
```

```
##
## Call:
## lm(formula = weight.lb ~ height.in, data = body)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -41.236 -14.084  -2.709  11.129  90.426
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -231.0248    16.5867  -13.93   <2e-16 ***
## height.in      5.6864     0.2458   23.14   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.48 on 505 degrees of freedom
## Multiple R-squared:  0.5145, Adjusted R-squared:  0.5136
## F-statistic: 535.2 on 1 and 505 DF,  p-value: < 2.2e-16
```

It is instructive to compare the estimates from this fit with the estimates when the variables are reported in metric units. For example, consider the slope. Our original estimate of the slope was $\beta_1 = 1.018$ kg / cm. To convert this to a slope measured in pounds per inch, we could have used

$$\begin{aligned}\widehat{\beta}_1 &= 1.018 \text{ kg / cm } \times (2.2 \text{ lb } / 1 \text{ kg }) \times (2.54 \text{ cm } / 1 \text{ in }) \\ &= 5.69 \text{ lb / in}\end{aligned}$$

which reproduces the answer shown in the R output. Think about this for a moment. We are saying that a rate of 1.018 kg / cm is the same as a rate of 5.69 lbs / in. Convince yourself that those two rates are the same. Can you perform a similar unit conversion for the intercept? For the standard deviation of the errors? How about for R^2?

Prediction

Finally, we'll illustrate how to use the regression model to make predictions. Here, we'll revert back to the model in metric units. Let's predict the weight of an individual who is 180 cm tall. Of course, point estimates of a prediction (a "single best guess'') are easily obtained by hand. To obtain a point prediction of the weight of a 180 cm tall individual, just use the equation for the fitted regression line:

$$\begin{aligned}\hat{y}^* &= \hat{\beta}_0 + \hat{\beta}_1 x^* \\ &= -105.01 + (1.018)(180) \\ &= 78.2 \text{ kg.}\end{aligned}$$

Confidence and prediction intervals are harder to obtain by hand, so we'll rely on the software to compute them for us. To make predictions using regression models in R, you need to create a new data frame that contains the values of the predictors for the new observations. (Note that this new data frame can be used to make many predictions at once, not just one.) Create this new data frame with the functions below.

```
new.data <- data.frame(height = 180)
```

Now, use the "predict" command to make the desired prediction:

```
predict(slr1, interval = "prediction", newdata = new.data, level = .95)
```

```
##        fit      lwr      upr
## 1 78.15977 59.83849 96.48104
```

To generate a confidence interval instead of a prediction interval, we would use

```
predict(slr1, interval = "confidence", newdata = new.data, level = .95)
```

```
##        fit     lwr      upr
## 1 78.15977 77.0438 79.27573
```

Try adjusting the "level" to generate a 99% prediction interval, or a 99% confidence interval instead.