

# Solutions to Lab 7

## *Univariate Statistics with R*

This week we'll be merging data collected from a number of sources before running a simple correlation and a linear model. You should end up knowing a bit more about data manipulation, as well as about running statistical tests. As always, create a new project in RStudio.

### Aperitif: `merge()` and missing values

First of all, generate a data frame with IQ measurements for 5 people (Amos, Bill, Chas, Dave, and Earl). (**Hint:** Remember that IQ is a normally distributed variable with  $Mean = 100$  and  $SD = 15$ .)

```
### ANSWER ###
```

```
iq.f <- data.frame(name = factor(c('Amos', 'Bill', 'Chas', 'Dave', 'Earl')),  
  iq = rnorm(n = 5, mean = 100, sd = 15))
```

```
##   name      iq  
## 1 Amos  80.04486  
## 2 Bill 119.23918  
## 3 Chas 119.03401  
## 4 Dave 106.33162  
## 5 Earl  94.86945
```

Next, create another data frame with height measurements for 5 people (Bill, Dave, Earl, Fred, and Greg).

```
### ANSWER ###
```

```
ht.f <- data.frame(name = factor(c('Bill', 'Dave', 'Earl', 'Fred', 'Greg')),  
  height = rnorm(5, 175, 15))
```

```
##   name   height  
## 1 Bill 165.7051  
## 2 Dave 181.2402  
## 3 Earl 175.3291  
## 4 Fred 175.0135  
## 5 Greg 172.6889
```

You'll notice that the data frames you've created contain partially overlapping information (some of the names). If you wanted to test for a relationship between IQ and height<sup>1</sup> it would be useful to create one merged data frame. Perhaps predictably, the R function for this is `merge()`:

Try the following commands:

```
merge(iq.f, ht.f)
```

```
##   name      iq   height  
## 1 Bill 119.23918 165.7051  
## 2 Dave 106.33162 181.2402  
## 3 Earl  94.86945 175.3291
```

```
merge(iq.f, ht.f, all = T)
```

---

<sup>1</sup>This would be a bit silly with 5 observations (and a bit silly anyway!) but it's just an example.

```
##   name      iq   height
## 1 Amos  80.04486      NA
## 2 Bill 119.23918 165.7051
## 3 Chas 119.03401      NA
## 4 Dave 106.33162 181.2402
## 5 Earl  94.86945 175.3291
## 6 Fred      NA 175.0135
## 7 Greg      NA 172.6889
```

What do the different versions do? What would you do if the columns in each data frame had different names? (Try `?merge`).

Now create a new merged data frame (note, this command is slightly different again):

```
new.f <- merge(iq.f, ht.f, all.x = T)
new.f
```

```
##   name      iq   height
## 1 Amos  80.04486      NA
## 2 Bill 119.23918 165.7051
## 3 Chas 119.03401      NA
## 4 Dave 106.33162 181.2402
## 5 Earl  94.86945 175.3291
```

Calculate the means of the different relevant columns. *Revision: The first two calculations do the same thing (why?)*

```
mean(new.f$iq)
```

```
## [1] 103.9038
```

```
mean(new.f[,2])
```

```
## [1] 103.9038
```

Now calculate the mean `height`:

```
mean(new.f$height)
```

```
## [1] NA
```

## Missing Values

`mean(new.f$height)` doesn't give you a mean; instead it returns `NA`. If you completed last week's worksheet you will have encountered `NA` values and perhaps learned to deal with them. The fact that `R` returns `NA` rather than ignoring any `NAs` in the values given may seem dumb at first (why can't it ignore the missing values?) but in fact it's based on an important principle: *You should know your data and you should know that there are missing values, and tell R*. If it 'silently' dealt with missing values, you could end up with meaningless results through not checking your data properly.

For simple functions (like `mean()`, `sd()`, etc.) *the only way to get R to ignore NAs is as follows* (`na.rm` can be read as "NA remove"):

```
mean(new.f$height, na.rm = T)
```

```
## [1] 174.0915
```

For more complicated functions (like `lm()`, which we'll meet below) there is a general default behaviour called `na.omit` which deletes missing values prior to calculation.

## Starter: A useful function

In Lab 5, you learnt to write functions; in Lab 6, you learnt about logical subsetting (where items in a vector or matrix are selected according to the logical values T or F). Can we put these together here?

Write a function called `outliers()` to test whether any of the elements passed to it are greater than, or less than, `x` standard deviations from the mean (where `x` is a value passed to the function). You should use a template something like the following:

```
outliers <- function(obs, x = 2.5) {  
  # code goes here  
}
```

```
### ANSWER ###
```

```
## one possible solution  
outliers <- function(obs, x = 2.5) {  
  # the following line returns TRUE if outlier, FALSE otherwise (for each element of 'obs')  
  return(abs(obs - mean(obs, na.rm = T)) > (sd(obs, na.rm = T) * x))  
}
```

**Tip:** Your function takes two values: `obs` (a vector or matrix of observations), and `x`, as above. `x = 2.5` passes a *default value* to `x`: If you don't specify an explicit value, the default will be used. This means that you can use `outliers(vec)` to find elements in a vector more than 2.5 sds from the mean; or `outliers(vec, 2)` if you prefer 2 sds from the mean.

Now write the code to complete the function:

**Tip:** You solved most of this problem last week, when you looked for outliers' in a matrix. You would have written code something like the following: `mat[mat > mean(mat) + 2 * sd(mat)] <- NA`

Here, you're being asked to produce a function that does something similar to the code inside the `[]` above. You want to use `x` rather than 2 so that the number of standard deviations can be arbitrary, and you need to take into account that your input could include NAs.

You also need to think about detecting values that are '`x`' standard deviations *below* the mean, in addition to those above it.

**Pro tip:** A neat solution to this problem might involve using the `abs()` function which converts values to their absolute value. See `?abs`, also for more on what an absolute value is, look here <https://www.mathsisfun.com/numbers/absolute-value.html>.

Test your `outliers` function with the following code: Does it correctly identify the 2 outliers?

```
vec <- rnorm(20, 100, 15)  
vec[sample(length(vec), 2)] <- 250 # create two outliers at random  
vec # inspect vector to find outliers  
  
## [1] 81.97338 78.22557 111.28514 105.89443 84.47922 250.00000 88.74325  
## [8] 77.34254 250.00000 103.48145 110.94137 71.44722 101.80677 124.33607  
## [15] 101.96495 82.49247 113.96429 90.96551 86.66524 101.54428  
  
which(outliers(vec)) # check outliers function  
  
## [1] 6 9
```

## Main Course: Some data manipulation

The aim of this exercise is for you to load some data, get it into a suitable format for analysis, and perform (for now) a simple correlation, and a linear regression. *NB., most of this exercise is about getting data into shape: Doing the stats is easy!.*

Download the data from the LEARN (lab7.Rdata), and load it into R.

**Tip:** This data is in R format, not .csv (because it contains several data frames). If you know how, save it into your project folder. Use one of the commands below to load it into R:

```
load('lab7.Rdata') # if your data is in your project folder
load(file.choose()) # or if you want to use the GUI
```

**Tip:** The first thing you should do when you've loaded your data is look at the **Environment** tab in Rstudio (top right), or type `ls()`, to find out what new objects you've loaded.

**Pro tip:** `ls()` returns a list of all the objects (variables and functions) in your workspace. That means it's very useful if you want to delete everything and start again: You can use `rm(list = ls())` (or if you like pointing and clicking, just click on the broom in RStudio's Environment tab). `rm()` is the function to remove things.

You have data from three different Universities on some students in their statistics classes. Each University (or School) has provided you with the same information; unfortunately, they have provided it in slightly different formats. Your task is to assemble all of the information into *one* data frame called `schools`, suitable for further analysis.

The data should consist of a unique student identifier, and for each student, the school they're in, their IQ, an exam score, and their gender. Unfortunately the records are not all complete (in particular, some exam scores are missing), and there may be other errors.

**Tip:** You will definitely want to use `merge()` to tackle this (be careful with the `all` arguments!). You may also be able to use `rbind()` to bind things row-wise as an alternative for some (but not all) merges. You should also be thinking about your indexing skills from last week's lab. Below are some things you might want to think about:

- are the observations complete?
- are there any typos?
- do the column names match?
- are there any unlikely values, or outliers?

The general approach is probably to fix up (or merge) the data from each individual school, before merging the complete dataset together (NB., you can only merge two things at a time...)

```
### ANSWER ###
```

```
#### SOME STEPS YOU MIGHT TAKE
```

```
summary(schoolA)
```

##	id	school	gender	iq	exam
##	s023	: 1	a: 1	female:25	Min. : 68.0
##	s028	: 1	A:62	male :38	1st Qu.: 90.5
##	s038	: 1			Median : 99.0
##	s059	: 1			Mean : 101.3
##	s103	: 1			3rd Qu.: 110.5
##	s105	: 1			Max. : 250.0
##	(Other)	:57			NA's : 14

```

# maximum IQ of 250 looks suspect!
schoolA$iq[schoolA$iq > 180]

## [1] 250

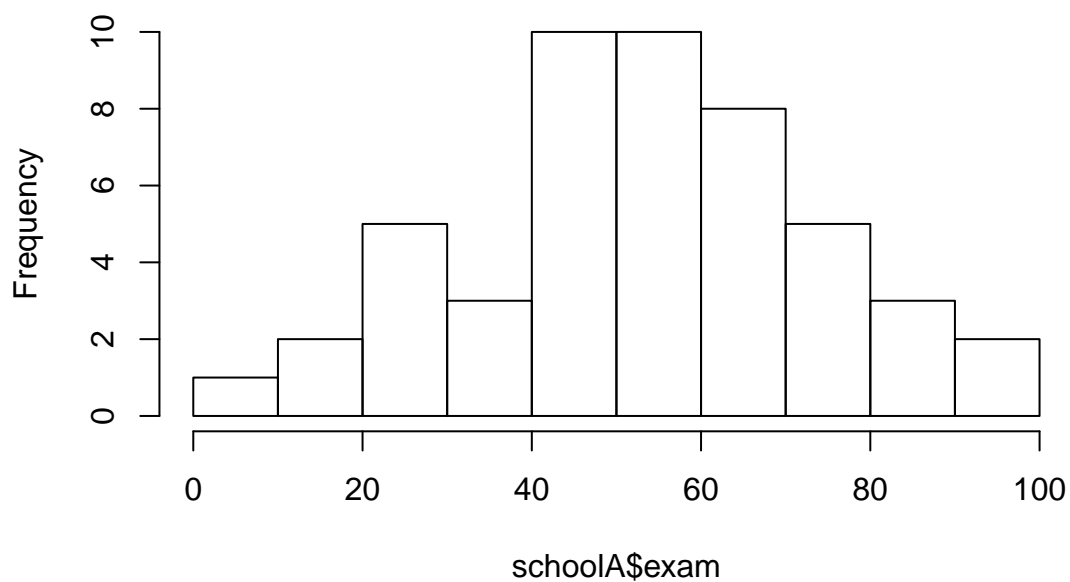
# there's just one; fix it
schoolA$iq[schoolA$iq > 180] <- NA
# exam range looks very broad; is the 3 unusual?
which(outliers(schoolA$exam))

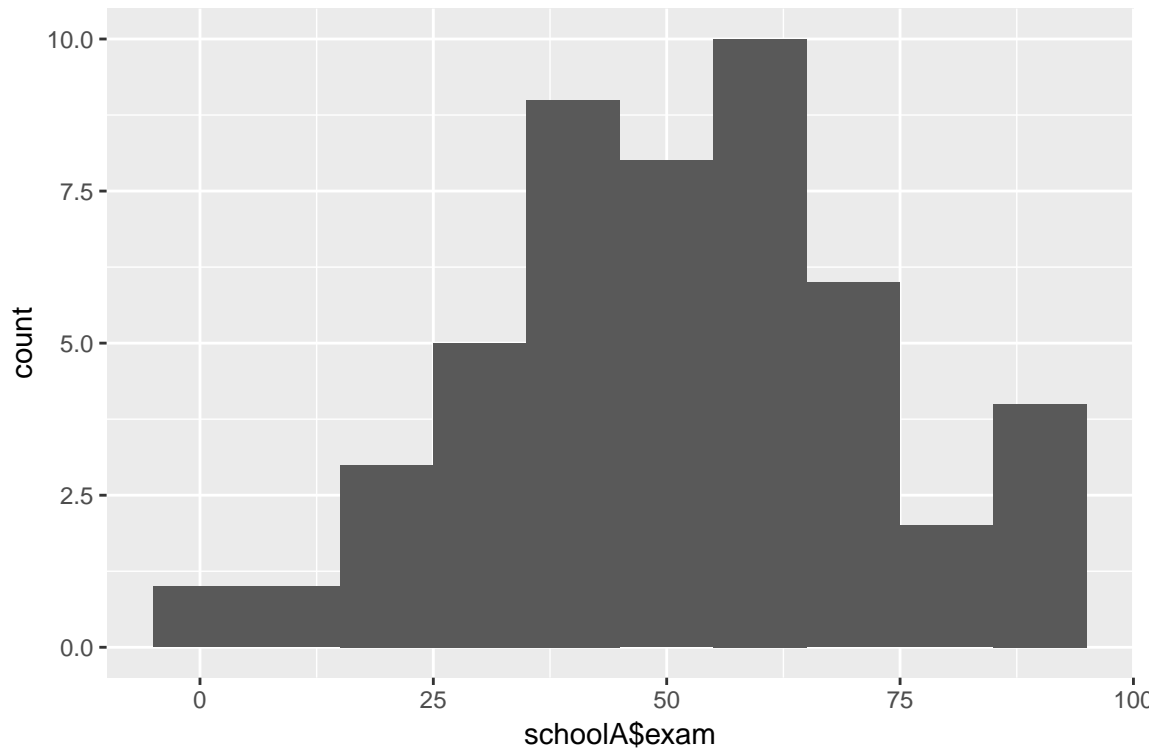
## integer(0)

# actually there are no outliers at 2.5sd; just a wide spread, see
hist(schoolA$exam) # or:
ggplot2::qplot(schoolA$exam, binwidth = 10)

```

**Histogram of schoolA\$exam**





```
# Look at the levels of school: 'a' and 'A'.....
# there's an 'a' where there should be an 'A'
schoolA$school[schoolA$school == 'a'] <- 'A'
# (can be useful to drop levels, although merge() should cope with this)
schoolA <- droplevels(schoolA)
```

```
# There are more IQ than exam entries (see number of NAs);
# We just have to throw away IQinfo for which there's no exam info etc.
schoolB <- merge(schoolB.IQ, schoolB.exam)
summary(schoolB)
```

```
##      id      iq      school  gender      exam
## s001 : 1  Min.   : 58.00  B:48  female:24  Min.   : 15.00
## s039 : 1  1st Qu.: 92.25      male :24  1st Qu.: 39.75
## s078 : 1  Median :102.00                      Median : 51.00
## s133 : 1  Mean   : 99.77                      Mean   : 53.29
## s136 : 1  3rd Qu.:108.00                      3rd Qu.: 66.00
## s144 : 1  Max.   :130.00                      Max.   :100.00
## (Other):42
```

```
# everything else looks hunky dory
```

```
# can now merge schools A&B: all = T is important, think about why!
schools <- merge(schoolA, schoolB, all = T)
```

```
summary(schoolC)
```

```
##      id  school  gender      IQ      exam
## s007 : 1  C:40  female:20  Min.   : 64.00  Min.   :  5.00
## s010 : 1      male :20  1st Qu.: 94.75  1st Qu.: 48.75
```

```
## s011 : 1 Median : 99.00 Median : 64.50
## s035 : 1 Mean : 99.95 Mean : 62.95
## s041 : 1 3rd Qu.:108.75 3rd Qu.: 75.50
## s055 : 1 Max. :135.00 Max. :104.00
## (Other):34
```

```
# column name is "IQ" rather than "iq"
colnames(schoolC)[4] <- 'iq'
# in doing random checking...
which(outliers(schoolC$exam))
```

```
## [1] 35
```

```
schoolC$exam[35]
```

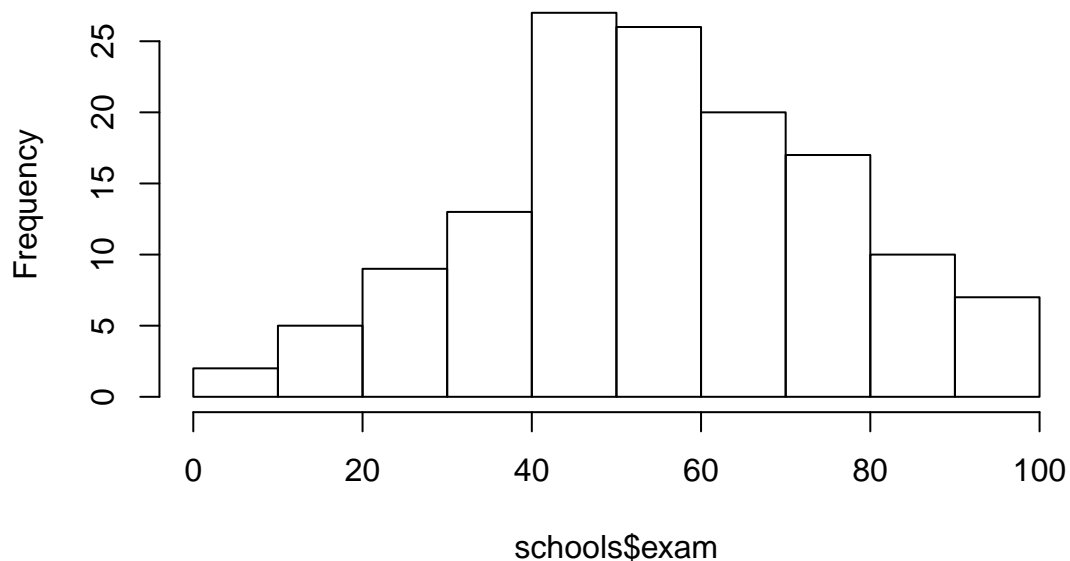
```
## [1] 5
```

```
# there's a very low score. But there was for SchoolA (not an outlier)
# so I'll wait until the scores are merged.
# However, there's an impossible exam mark of 104!
schoolC$exam[schoolC$exam > 100] <- NA
```

```
# now merge again
schools <- merge(schools, schoolC, all = T)
```

```
## check exam one more time for outliers
hist(schools$exam)
```

## Histogram of schools\$exam



```
which(outliers(schools$exam))
```

```
## [1] 42
```

```
schools$exam[42]
```

```
## [1] 3
```

```
### OK, that 3 *is* low according to the 2.5sd default
```

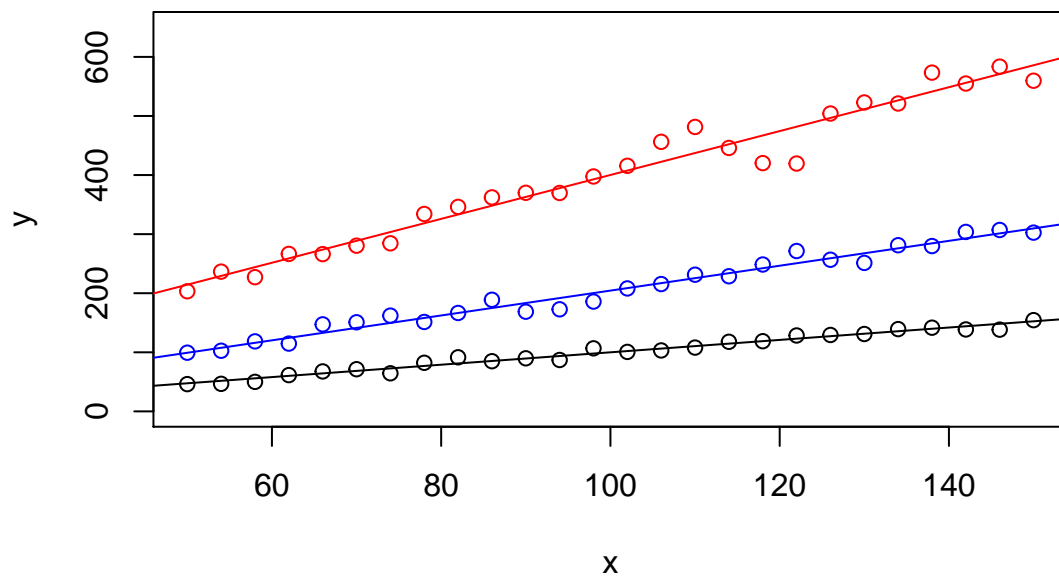
```
schools$exam[42] <- NA
```

## Side Dish: A note on correlation

```
x = seq(50, 150, 4)
y = x + rnorm(n = length(x), m = 0, sd = 5) # add some random noise to x and store it in y
y2 = 2 * x + rnorm(n = length(x), m = 0, sd = 10)
y3 = 4 * x + rnorm(n = length(x), m = 0, sd = 20)
```

Take a look at the plot generated below where the black, blue, and red values (and their ‘line of best fit’) represent different variables and their relationship with the variable  $x$ . Take a second to think about what you expect the correlations between  $x$  and the three different variables representing  $Y$  (black, blue, and red) will be.

```
plot(x, y, ylim = c(0, 650))
points(x, y2, col = "blue")
points(x, y3, col = "red")
abline(lm(y ~ x))
abline(lm(y2 ~ x), col = "blue")
abline(lm(y3 ~ x), col = "red")
```



Some of you may have been tempted to suggest that the blue line indicates a higher correlation than the black line, and similarly that the red line indicates a higher correlation than both alternatives. This is a



common mistake when thinking about correlations. All these correlations are actually identical (almost 1):

```
cor(x, y) # Black
```

```
## [1] 0.9881674
```

```
cor(x, y2) # Blue
```

```
## [1] 0.9874537
```

```
cor(x, y3) # Red
```

```
## [1] 0.9802778
```

Remember that correlation tells you how well the line fits the data (how close the points are to it) and not how steep it is. Looking at all the lines again - you can see that in each case knowing the value of **X** for any one participant allows you to say with almost certainty what their **Y** value would be. The steepness of the gradient is determined by the increase in **Y** units for each one unit increase in **X** - this is the coefficient from the linear model.

## Dessert: Some statistics!

Using `cor.test()` and `lm()`, run a correlation, and then a linear model, to examine the relationship between IQ and exam performance in the `schools` dataset you've just created. What can you conclude from your analyses?

**Tip:** `cor.test()`, and other R functions ending in `.test`, are simple functions, designed to do a test and print out information immediately. `lm()` is a bit more complex: The simplest way to start with it is to assign its output to a variable (`var <- lm(...)`) and then use `summary()` to tell you the most useful information about it (`summary(var)`).

```
load(url("http://is.gd/tsktsk"))
```

```
# NB., results may differ slightly depending on merging decisions
```

```
with(schools, cor.test(iq, exam))
```

```
##
```

```
## Pearson's product-moment correlation
```

```
##
```

```
## data: iq and exam
```

```
## t = 3.015, df = 132, p-value = 0.003083
```

```
## alternative hypothesis: true correlation is not equal to 0
```

```
## 95 percent confidence interval:
```

```
## 0.08802564 0.40593935
```

```
## sample estimates:
```

```
## cor
```

```
## 0.2538249
```

```
# there's a significant positive correlation between iq and exam;
```

```
# higher iq is associated with better exam score
```

```
model <- lm(exam~iq, data = schools)
```

```
summary(model)
```

```
##
```

```
## Call:
## lm(formula = exam ~ iq, data = schools)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -52.742 -14.319  -1.098  12.805  49.294
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  21.7961    11.4757   1.899  0.05970 .
## iq           0.3423     0.1135   3.015  0.00308 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19.82 on 132 degrees of freedom
## (17 observations deleted due to missingness)
## Multiple R-squared:  0.06443,    Adjusted R-squared:  0.05734
## F-statistic:  9.09 on 1 and 132 DF,  p-value: 0.003083
# the model is better than the null model (F-test; p < .05)
# for each point increase in iq, exam score goes up by 0.34 (coefficient)
```