# 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 $SD = 15$.)

```
##   name        iq
## 1 Amos 102.03294
## 2 Bill  90.33829
## 3 Chas  96.96715
## 4 Dave  78.63744
## 5 Earl  97.16176
```

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

```
##   name   height
## 1 Bill 163.5755
## 2 Dave 145.3572
## 3 Earl 172.2430
## 4 Fred 187.4525
## 5 Greg 151.0741
```

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[1] 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 90.33829 163.5755
## 2 Dave 78.63744 145.3572
## 3 Earl 97.16176 172.2430
```

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

```
##   name        iq   height
## 1 Amos 102.03294       NA
## 2 Bill  90.33829 163.5755
## 3 Chas  96.96715       NA
## 4 Dave  78.63744 145.3572
## 5 Earl  97.16176 172.2430
## 6 Fred        NA 187.4525
## 7 Greg        NA 151.0741
```

[1] This would be a bit silly with 5 observations (and a bit silly anyway!) but it's just an example.

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 102.03294       NA
## 2 Bill  90.33829 163.5755
## 3 Chas  96.96715       NA
## 4 Dave  78.63744 145.3572
## 5 Earl  97.16176 172.2430
```

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

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

```
## [1] 93.02751
```

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

```
## [1] 93.02751
```

Now calculate the mean `height`:

### Missing Values

`mean(new.f$height)` doesn't give you a mean; instead it returns `NA`. If you completed last weeks worksheet you will have encountered `NA` values and perhaps learned to deal with them. The fact that `R` returns `NA` rather than ignoring any `NA`s 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] 160.3919
```

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
}
```

**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 `NA`s.

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]  81.62732 104.24717 105.90388  97.26439 250.00000  99.55823  81.57253
##  [8]  90.90572  87.28355 105.34602 100.41671 250.00000  93.19981  74.33235
## [15] 106.49141  83.50246  92.58841 101.70719 133.13769  90.79541
```

```
which(outliers(vec)) # check outliers function
```

```
## [1]  5 12
```

## 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?

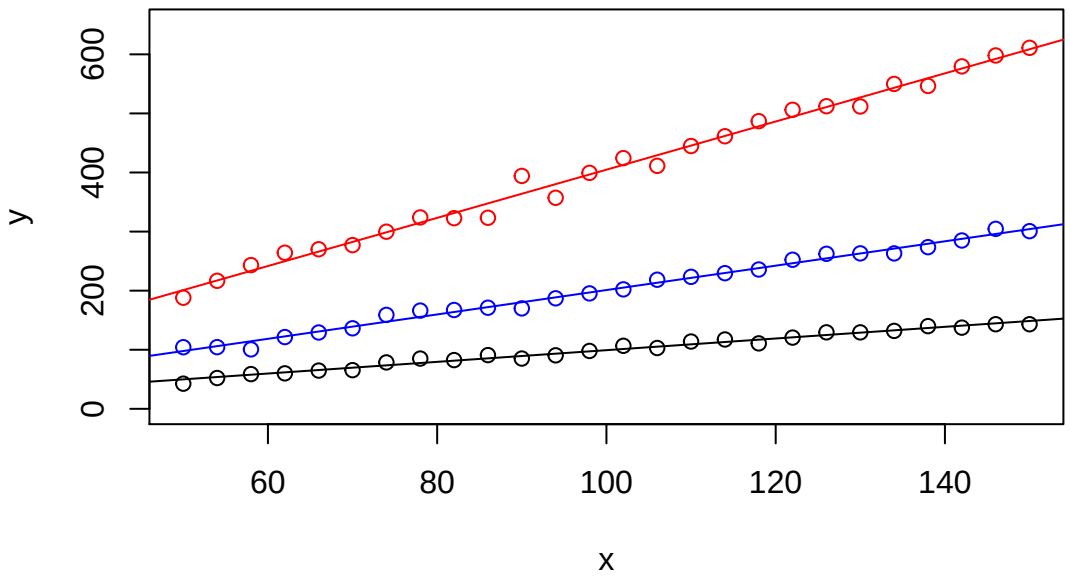
The general approach is probably to fix up (or merge) the data from each individual school, before merging the complete dataset together (NB., you can only merge two things at a time...)

## 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] 0.9912096
```

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

```
## [1] 0.9952614
```

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

```
## [1] 0.9950507
```

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)
```