Solution to Lab 9

Univariate Statistics with R

A tiny bit more R!

ifelse

Today's lab is primarily focused on extending our knowledge/experience of conducting and interpreting linear models. But before that, a brief note on if-else statements as you may have noticed examples of them in the lecture slides. Take a look at the following code example taken from slide 16:

```
apples$redness <- ifelse(apples$colour == "red", 1, 0)
```

What is happening here? In this command, a new variable is being created in the *apples* dataframe called 'redness'. To decide what value to give each case (row) for this variable, an **ifelse** statement is used. For each row in the data, the value of the 'redness' variable is based on the value of the 'colour' variable. When 'colour' is red, the value for 'redness' is 1, for any other value of 'colour' - 'redness; will be 0. The reason for this is due to how **ifelse()** works. Take a look at the following general formulation:

```
outcome <- ifelse(SOME_EXPRESSION, VALUE_IF_TRUE, VALUE_IF_FALSE)
```

`SOME_EXPRESSION` can take the form of any expression that evaluates to a **logical vector** (this means TRUE/FALSE values) - we have worked with some examples of this over the last few weeks (*e.g.,* `mat > mean(mat) + 2.5 * sd(mat)` produced a logical vector where a TRUE value represented values of mat greater than 2.5 standard deviations from the mean, and FALSE for all other values). The second argument `VALUE\_IF\_TRUE` does exactly what it claims and puts this value into 'outcome' at each position a TRUE appears as a result of `SOME\_EXPRESSION`. Hopefully it naturally follows that `VALUE\_IF\_FALSE` does.\

To consolidate this, we will finish off the example from the lecture. Initially, this is what the **apples** dataframe looks like:

```
head(apples)
```

```
##   colour taste
## 1  green   4.3
## 2  green   3.5
## 3    red   3.9
## 4    red   2.7
## 5  green   6.4
```

We then run:

```
apples$redness <- ifelse(apples$colour == "red", 1, 0)
```

And now **apples** looks like so:

```
head(apples)
```

```
##   colour taste redness
## 1  green   4.3       0
## 2  green   3.5       0
## 3    red   3.9       1
## 4    red   2.7       1
## 5  green   6.4       0
```

And this is because the value of `SOME\_EXPRESSION` in our example evaluated to:

```
apples$colour == "red"
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

The first two elements evaluate to FALSE because the first two values in the 'colour' variable are not 'red'. Therefore, the first two elements of our outcome (the new variable 'redness') are set to `0` which was our `VALUE\_IF\_FALSE` argument. The next two values in the 'colour' variable were 'red' so these elements evaluated to TRUE so the next two values of 'redness' were set to our `VALUE\_IF\_TRUE` argument which was `1`.

Model Specification

In the lecture, you were introduced to interactions in linear models. Therefore we need to know how to tell R to also fit interaction terms in our calls to `lm()`.

```
lm(y ~ x1)
lm(y ~ x1 + x2)
```

So far, these are the types of linear models we have been asking R to fit. In the first example, the command tells R we want to predict `y` using the explanatory variable `x1`. The second example extends this by also using `x2` as an explanatory variable. To also ask for R to fit the interaction between `x1` and `x2`, we can execute:

```
lm(y ~ x1 + x2 + x1:x2)
```

We use a colon to seperate two terms to indicate we want to fit the interaction between these terms. An alternative method is:

```
lm(y ~ x1 * x2)
```

Using the * tells R to fit the full model using these variables: each of the variables individually and the interaction term.

Moving On...

Today's exercise is a little different: We're going to build a dataset and then analyse it. The purpose is to make clear the relationship between the regression formula (of the form $y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + ... + b_n x_{ni} + \epsilon_i$) and the formula that creates the numbers.

Task 1: Create a data frame to hold your data with participant labels for 100 people.

To do this, you can use the `gl()` function: `gl' stands for`generate levels'. The general format of 'gl' is `gl(n, k)` where n is the number of levels you want to create and k is how many of each level you want. See `?gl` for further information.

```
### ANSWER ###

my.data <- data.frame(ppt = gl(100, 1))
```

Task 2: Now let's add in some observations about these participants. Give some values for their `age`, their `musicality` (some measure of how interested the participant is in music generally), and their `gender`.

You shouldn't need much guidance here, except to make up some values that vary plausibly for each column. So for example if you thought that **age** was best expressed in years and our participants' ages should vary uniformly from 20 to 60, you might type

```
my.data$age <- runif(100, 20, 60)
```

but this is only an example. You might assume that the age of participants is normally distributed; you might want to express it in months; and so on.

Similarly, you can define **musicality** however you like: You could use `rnorm()` or some other random function to assign values to it; or you might want to `sample()` (with replacement) from a strict set of numbers such as `1:7`, for example.

For **gender**, the `gl()` function you've just encountered should do the trick: look at `?gl`, and consider using the `labels =` argument to make it clear which gender is which.

```
### ANSWER ###

## Here are some simple versions based on the hints above:
my.data$musicality <- sample(c(1:7), 100, replace = TRUE)
my.data$gender <- gl(2, 50, labels = c('F', 'M'))
```

Task 3: Have a look at the dataset you have created: `my.data`. Have a look at a summary of the variables you have created: `summary(my.data)`. Do they look sensible? *most likely yes, but make sure nothing stands out as strange*

```
### ANSWER ###

## here's the summary based on the commands above:
summary(my.data)
```

```
##       ppt          age          musicality   gender
##  1      : 1   Min.   :21.62   Min.   :1.00   F:50
##  2      : 1   1st Qu.:33.72   1st Qu.:2.00   M:50
##  3      : 1   Median :44.08   Median :4.00
##  4      : 1   Mean   :42.47   Mean   :3.74
##  5      : 1   3rd Qu.:51.88   3rd Qu.:5.25
##  6      : 1   Max.   :59.47   Max.   :7.00
##  (Other):94
```

Task 4: OK, now comes the fun part. We want to create a variable called `OneDLove` in our data frame, which is a *function* of `musicality` and `age`. In other words, you're being asked: 'What is the (fictional) way in which age and musicality predict the amount that people love One Direction?'

To do this, you'll need to think about regression equations in general, and in particular, about interaction terms. How do age and musicality *interact*? Do old musical people like One Direction more than young musical people? And so on. The exact formula is *up to you* but, it's a linear equation, so it should be something like:

```
my.data$OneDLove <- ... + (... * my.data$age) + (... * my.data$musicality) +
  (... * my.data$age * my.data$musicality)
```

```
### ANSWER ###

# could be anything really, but how about:
my.data$OneDLove <- 100 - 0.1 * my.data$age -
  2 * my.data$musicality -
  0.1 * my.data$age * my.data$musicality
```

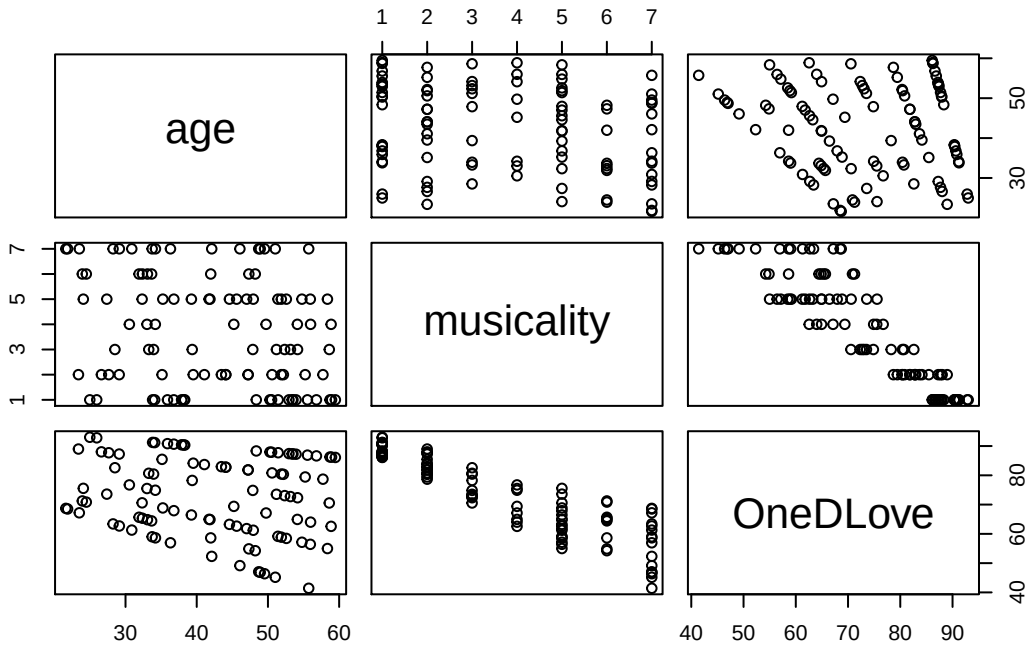
In this equation, the ...'s represent spaces where b_0, b_1, b_2, and so on *(see slides 36-37 from the lecture)*. Which means that what this equation is actually calculating is $\widehat{\text{OneDLove}}$, the *estimated* (or fitted) {value} of `OneDLove` *(based on your choices of b_0, b_1, etc.)*. Your job is to chose bs that give a plausible estimate, for whatever you think the relationship might be.

Task 5: Have a look at your data. There are various ways of doing this! For a fancy 3D-plot you can install the `rgl` library and try the code below, and then move the resulting graph around with your mouse.

```
library(rgl)
with(my.data, plot3d(age, musicality, OneDLove, size = 3, col = 'red'))
```

If you can't install `rgl` or want to stay within 2-dimensions, try a flat plot, like this:

```
plot(my.data[, c('age', 'musicality', 'OneDLove')])
```



Task 6: Does everything look relatively plausible? If not, you could start again at Task 4 (or earlier), changing values or the formula to create `OneDLove` until you're happy with the result.

Task 7: The `OneDLove` data you've created above is of *fitted values*, but real data is *noisy* - even when the relationship is extremely strong the real values will not lie perfectly on the 'line of best fit'. So, add some noise to `OneDLove`!

Remember that the noise will have to be normally distributed to match the assumptions of a linear model. Refer back to the formalisations of the linear models in the lecture slides. The noise you are adding corresponds to the error term in the model formalisation - ϵ_i. You might want to experiment with the size (sd) of the noise a little...

Tip: R makes it really easy to add a value to every element of a vector. To add 5 to every element of vector `x` you can execute `x + 5`. Combine this with the fact that the noise you are adding needs to be normally distributed.

```
### ANSWER ###

# it should be fairly easy to do this, although obviously the sd of the noise depends on
# the scale of the outcome variable
my.data$OneDLove <- my.data$OneDLove + rnorm(100, 0, 5)
```

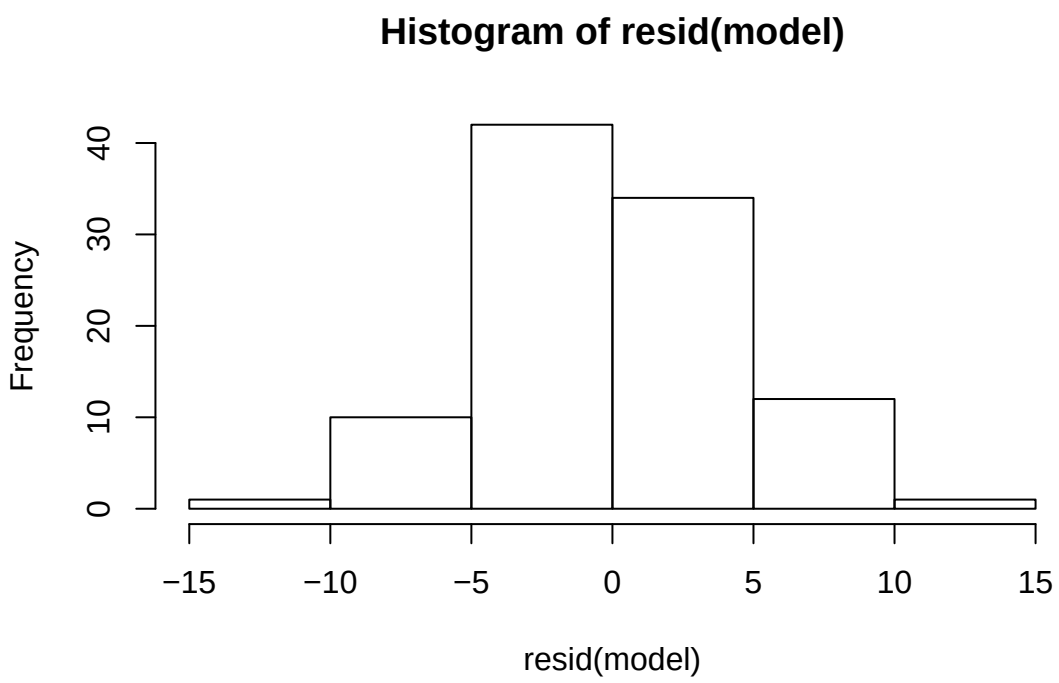
Task 8: Run a linear regression to test how `age`, `musicality`, and their interaction predict `OneDLove`. Examine your model. How close are the coefficients to the b values that you initially entered into the equation?

```
### ANSWER ###

model <- lm(OneDLove ~ age * musicality, data = my.data)
summary(model)
```

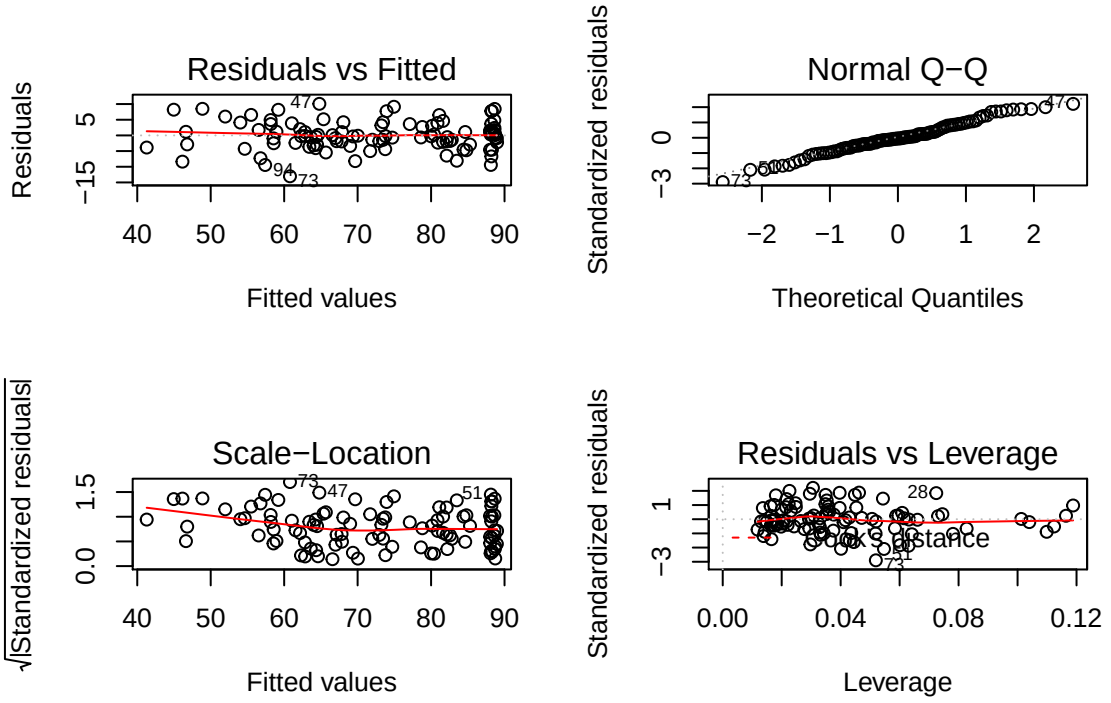
```
##
## Call:
## lm(formula = OneDLove ~ age * musicality, data = my.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.0455  -2.5948  -0.2201   3.0906  10.0716
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    90.44938    3.89147  23.243  < 2e-16 ***
## age             0.09855    0.08586   1.148    0.254
## musicality     -0.77326    0.83874  -0.922    0.359
## age:musicality -0.12624    0.01959  -6.443 4.66e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.623 on 96 degrees of freedom
## Multiple R-squared:  0.8904, Adjusted R-squared:  0.8869
## F-statistic: 259.9 on 3 and 96 DF,  p-value: < 2.2e-16
```

Task 9: Check the model assumptions. Are the residuals normal? Do the other diagnostics look sensible?

```
### ANSWER ###

hist(resid(model))
```




```
par(mfrow = c(2, 2)) # to plot all of the below in one window
plot(model)
```




```
par(mfrow = c(1, 1)) # revert back to one plot per window
```

Task 10: Re-run the equation you created at Task 4 (this resets the `OneDLove` variable to the exact fitted values without the noise (ϵ)). Now add a different type of residual that isn't sampled from the normal distribution like before - use a different `r-` (*i.e.*, not `rnorm`) function, or generate the residuals in some different way. Re-run the regression and check the model assumptions. What has changed?

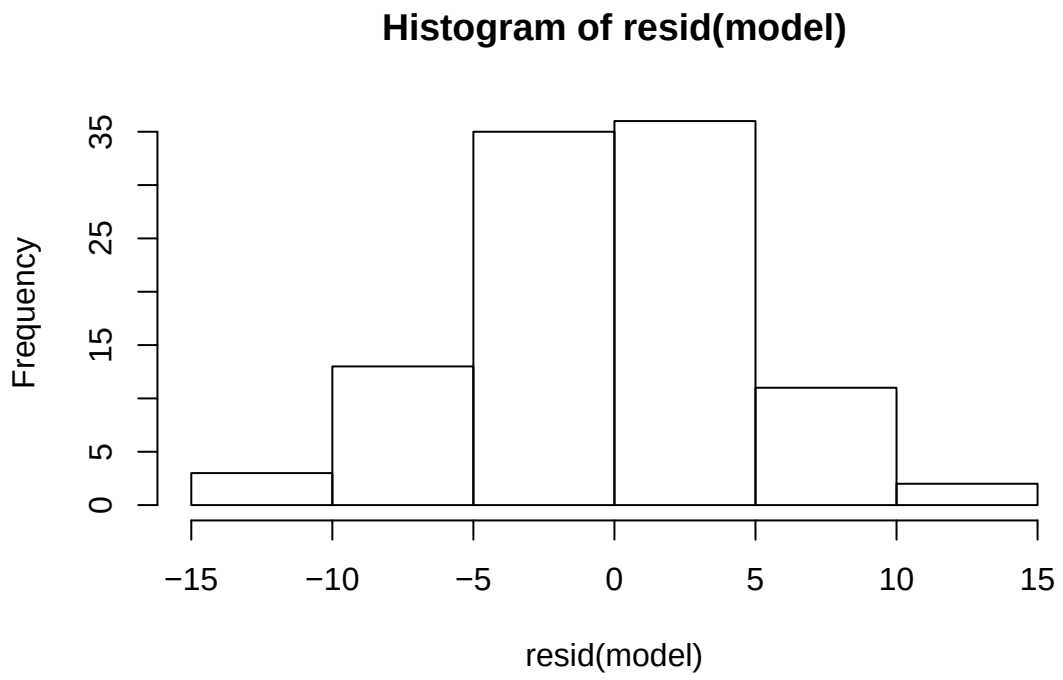
Tip: Before, we added noise to `OneDLove` by randomly selecting values using `rnorm()` which should have ensured the added noise was more-or-less normally distributed. Here, we want to make sure the added noise is **not** normally distributed. Think about the continuous probablity distributions you have come across in lectures and/or the Navarro textbook that are not normal. One of many possibilities might be the chi-square distribution with low degrees of freedom.

```
### ANSWER ###

## obviously you could choose almost anything here; I've chosen the f distribution
## because I know it's skewed
my.data$OneDLove <- my.data$OneDLove + rf(100, 1, 50)
model <- lm(OneDLove ~ age * musicality, data = my.data)
summary(model)
```

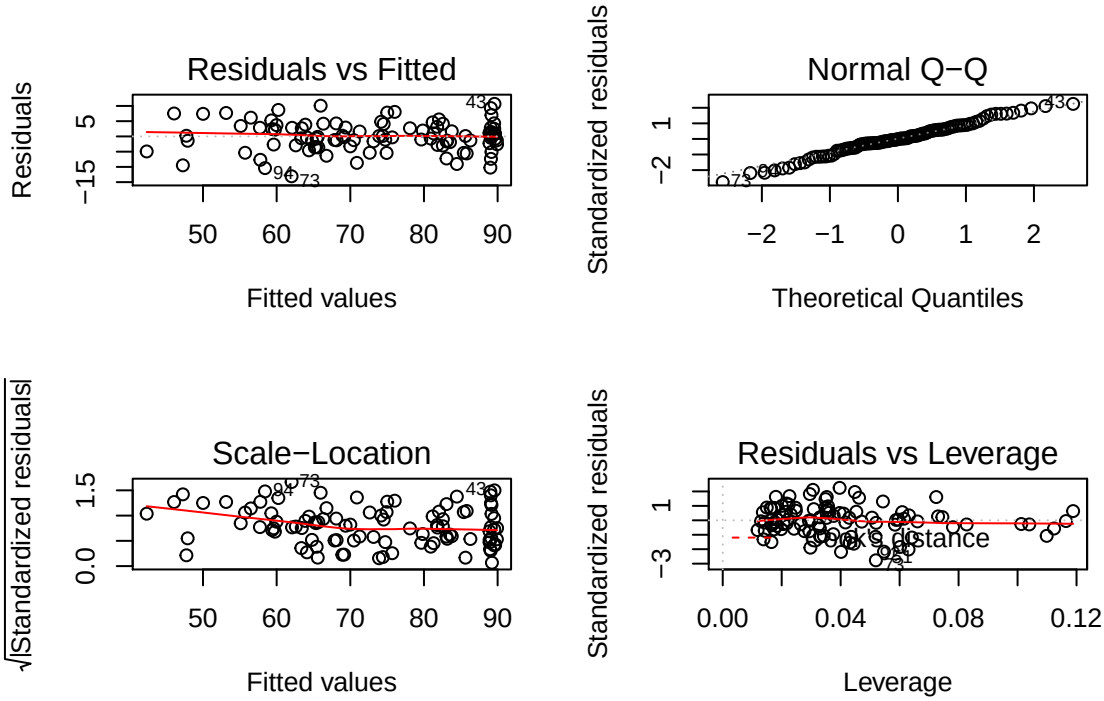
```
##
## Call:
## lm(formula = OneDLove ~ age * musicality, data = my.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.0826  -2.7037  -0.0784   2.8574  10.6909
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    91.37039    4.08396  22.373  < 2e-16 ***
## age             0.09722    0.09011   1.079    0.283
## musicality     -0.69871    0.88023  -0.794    0.429
## age:musicality -0.12700    0.02056  -6.177 1.57e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.851 on 96 degrees of freedom
## Multiple R-squared:  0.8793, Adjusted R-squared:  0.8755
## F-statistic: 233.1 on 3 and 96 DF,  p-value: < 2.2e-16
```

```
hist(resid(model))
```




```
par(mfrow = c(2, 2))
plot(model)
```

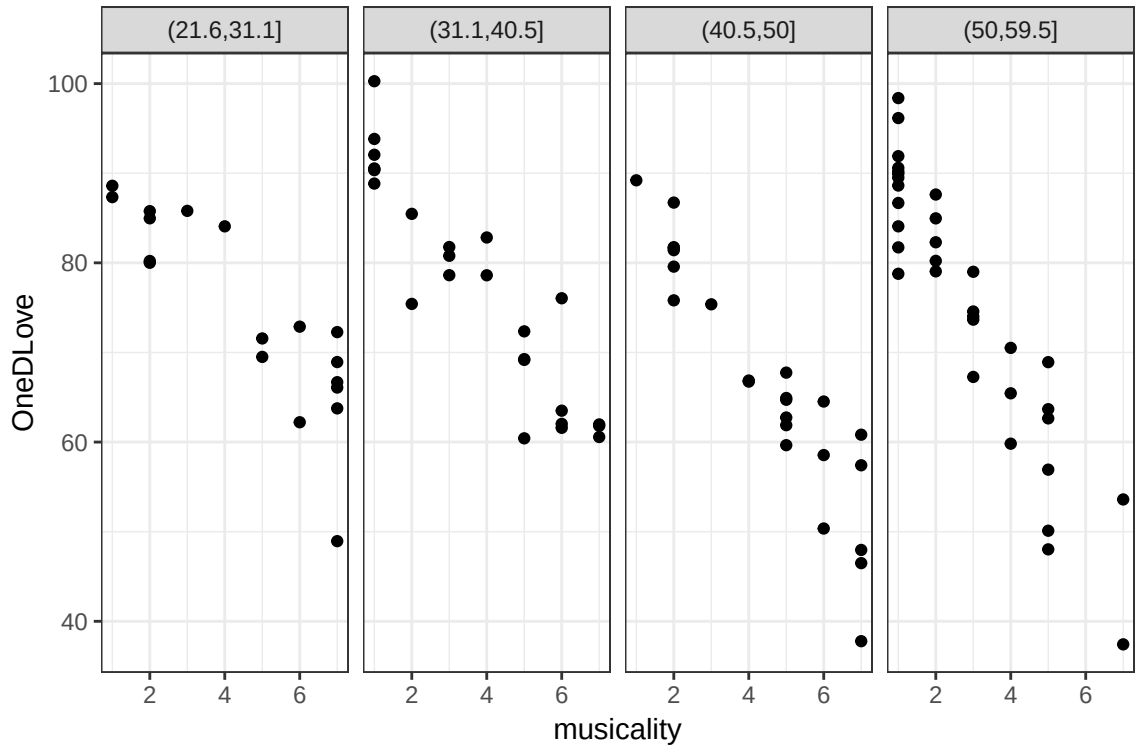



```
par(mfrow = c(1, 1))
### etc.
```

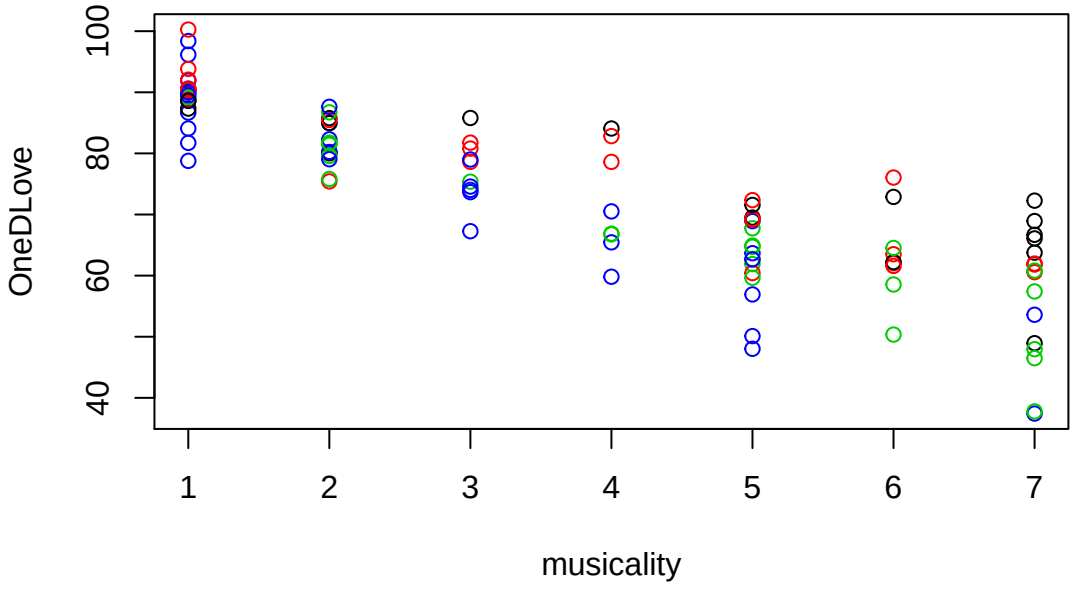
If you selected a continuous measure of musicality you were working with a *continuous x continuous* interaction. A good way to visualise this is to 'bin' the data based on one of the explanatory variables. In the example above we might create a grouped age variable that split age into 4 or 5 equal sized categories (we used `cut()` in an earlier lab to ahcieve this). Then create plots of musicality by age for each of these groups seperately (or you could put them in one plot and use a method to identify which points belong to which group, it's up to you). A significant interaction in our models suggests that the pattern across each group should not be identical, there should be some deviation.

Task 11: *If you have time*: Create these plots for your data to visualise any significant interaction.

```
### ANSWER ###

# A few different examples of illustrating this:

my.data$cutAge <- cut(my.data$age, 4)

#a ggplot
library(ggplot2)
inter.plot <- ggplot(data = my.data,
       aes(x = musicality, y = OneDLove))
inter.plot + geom_point() + theme_bw() + facet_grid(. ~ cutAge)
```

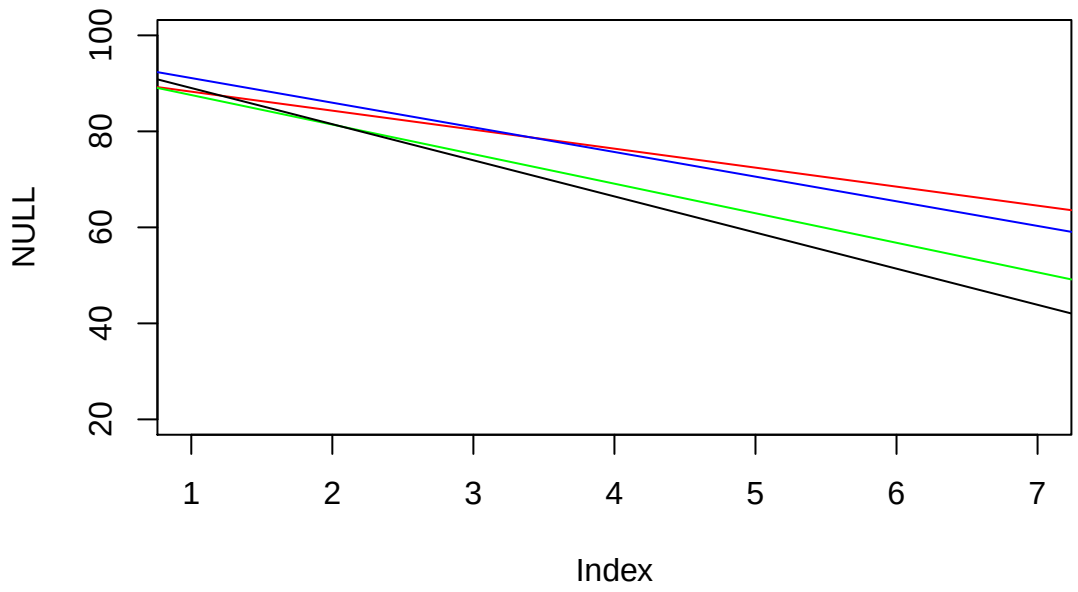



```
#plot() colouring by age bin
with(my.data, plot(musicality, OneDLove, col = cutAge))
```




```
#alternatively, plot each fitted line using abline()
#Note: repeat command, changing the subset variable and col on each
```

```r
plot(NULL, xlim = c(1, 7), ylim = c(20, 100))
abline(lm(OneDLove ~ musicality,
          data = subset(my.data, cutAge == levels(cutAge)[1])), col = "red")
abline(lm(OneDLove ~ musicality,
          data = subset(my.data, cutAge == levels(cutAge)[2])), col = "blue")
abline(lm(OneDLove ~ musicality,
          data = subset(my.data, cutAge == levels(cutAge)[3])), col = "green")
abline(lm(OneDLove ~ musicality,
          data = subset(my.data, cutAge == levels(cutAge)[4])), col = "black")
```



Task 12: ***If you have time*****: Start again at Task 4. This time create a variable called `PFLove` (for 'Pink Floyd Love') which is based on `age` and `gender`. (*N.B.*, you may need to make a column called `is.male`, like similar columns in the lecture, to help with this (use the `ifelse()`) function.) Run through the steps above again, checking that you can retrieve your b values from the regression you finally run.**

```r
### ANSWER ###

### something like:
my.data$is.male <- ifelse(my.data$gender == 'M', 1, 0)
my.data$PFLove <- with(my.data, 30 + 5 * is.male + .4 * age + .3 * age * is.male)
my.data$PFLove <- my.data$PFLove + rnorm(100, 0, 5)
# look at data
with(my.data, plot3d(age, is.male, PFLove))

# we can take advantage of automatic contrast coding in R
model <- lm(PFLove ~ age * gender, data = my.data)
```

```r
summary(model)
```

```
##
## Call:
## lm(formula = PFLove ~ age * gender, data = my.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.9094  -3.0603  -0.4266   3.2777  13.1830
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 27.67892    2.61769  10.574  < 2e-16 ***
## age          0.45377    0.06050   7.500 3.19e-11 ***
## genderM     12.01206    4.00593   2.999  0.00345 **
## age:genderM  0.14781    0.09127   1.620  0.10862
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.883 on 96 degrees of freedom
## Multiple R-squared:  0.8429, Adjusted R-squared:  0.838
## F-statistic: 171.7 on 3 and 96 DF,  p-value: < 2.2e-16
```