# MA26620: Practical 3

Scatterplots, Regression, and the package 'ggplot2'

Good morning! In today's practical we:

- learn how to generate random data from various distributions;
- learn how to make a Q-Q plot like the ones we discussed in yesterday's lecture;
- meet the package `ggplot2` which lets us make a wide array of professional-looking plots;
- think about linear regression, which includes some remarkable (real) data from the 1960s on how taking LSD affects one's mathematical ability (yes, really).

There are some exercises at the end on regression. If you don't finish all of this within two hours (which you likely won't – some work outside timetabled sessions is to be expected, especially in a 20 credit module!), do make sure you give everything a go before the next practical in a fortnight.

## 1 Investigating Normality

We've seen in the lectures that the Normal distribution is incredibly important and frequently occurring, in part due to the Central Limit Theorem. Often in our models, we'll make the assumption that something is Normally distributed. But how do we know whether this is a sensible assumption or not?

Well, we've said in the lectures that the best way to do this is a quantile-quantile (or Q-Q) plot, which is more sophisticated than looking at a histogram. At the start of this practical, we're going to look at generating random samples in R from different distributions, and then we're going to see what the resulting histograms and Q-Q plots look like.

### 1.1 Generating some random data

There are many commands in R for generating random samples (strictly speaking pseudo-random) from different distributions.

The following produces a random sample of size 100 from a Normal distribution, $N(\mu, \sigma^2)$:

```
x <- rnorm(100,3,2)
```

This creates a sample of 100 pseudo-random numbers drawn from a $N(3, 2^2)$ distribution. Note that `rnorm` takes three inputs: the sample size, the mean $\mu$ and the **standard deviation** $\sigma$, NOT $\sigma^2$. You can type `x` to see all the numbers you've generated and `summary(x)` will show you the mean (which is hopefully not a million miles away from 3) and the various quartiles.

It's hard to see how our generated data is distributed from this list of numbers. Let's make a quick histogram:

```
hist(x, probability=TRUE, main="N(3,4)")
```

Does this look Normally distributed to you? Each of you will have generated a different sample so some people's data might look nice and symmetrical and not dissimilar to a bell-shaped Normal probability curve. Others' may look less so.

This is why a Q-Q plot is a more refined tool. See the module webpages (specifically Normal random samples > Is my data Normally distributed?) or your lecture notes for a reminder of what a Q-Q plot does.

We can make a simple Normal Q-Q plot by typing

```
qqnorm(x)
qqline(x)
```

The points will not stray far from the line, and any deviations don't have too strong a pattern to them. We'd naturally expect a bit more variation in smaller samples, so try

```
x1 <- rnorm(30);qqnorm(x1);qqline(x1)
```

which will run 3 commands from one line (that's what the semi-colons do) which generates 30 values from a $N(0,1)$ distribution), then makes the Q-Q plot.

You can easily repeat this a few times (up arrow, enter), generating a fresh sample `x1` each time, in order to get a feel for the variation you might expect. The straight line added by `qqline()` passes through the upper and lower quartiles (ie 25th and 75th quantiles) and it indicates the expected line if the distribution is truly Normal. Departures from the line show whether the data have more extreme values than expected (eg Exponential) or are more constrained (eg Uniform).

You can easily modify the above commands to explore some different distributions comparing with the Normal in each case. For example, generate 30 values from an Exponential distribution and make a Normal Q-Q plot:

```
x2 <- rexp(30,0.25);qqnorm(x2)
qqline(x2)
```

You'll see from your plot that any assumption of Normality here would be a foolish one.

For a more extreme example of non-normality, let's use a data set called `faithful` that's supplied with R. (Remember `?faithful` to find out more; `head()` to see what the data looks like.)

```
attach(faithful)
hist(eruptions,breaks=25,xlim=c(1,6), probability=TRUE, col=gray(.75),
           main="Eruption durations(min) of the Old Faithful geyser")
```

As you can see the distribution is clearly bimodal; there are quite distinct 'subpopulations' of short and long eruptions. How does this appear in the Q-Q plot?

```
qqnorm(eruptions,main="Eruption durations(min) of the Old Faithful geyser")
qqline(eruptions)
detach(faithful) # it's good to tidy up!
```

## 1.2 Related functions in R

Functions for generating random numbers include `runif()` for Uniform, `rbinom()` for Binomial, `rexp()` for Exponential, `rpois()` for Poisson. Similarly, for the Uniform distribution, `dunif()` returns the density, `punif()` the cumulative distribution function (cdf), `qunif()` the quantiles. See Help for details. You can change to other distributions in the obvious way.

# 2 The package "ggplot2"

We saw in the last practical that R's basic plotting commands have their limitations. Making complicated plots beyond the `plot(x,y)` type that we saw last week can get rather tricky, and we saw that even basic things like adding a legend in a sensible place can be a little on the cumbersome side.

To get more sophisticated plots, we can use the package **ggplot2**. Packages are add-ons to R and very many exist due in part to the open-source nature of the package. If there's anything that a statistician wants to do in R, then chances are that someone has made a package that makes it easier.

Before we can use one of these add-on packages, we have to **install and enable** it. University machines may well already have some common packages pre-installed, although this can vary from room to room. I'll include instructions below for how to install packages, which will be useful if you want to use them on your own machine. The *installation* step only needs to be done once, but you'll likely have to *enable* it each time you open RStudio and want to use ggplot2.

## 2.1 Install ggplot2

First, let's check with ggplot2 is already installed. Look for ggplot2 in the Packages tab of the bottom right hand pane of RStudio — if it's there, then skip to section 2.2. If it's not, run the command `install.packages("ggplot2")` and then wait a few seconds (minutes?) while it downloads it and installs it. On some machines you may get a warning about ggplot2 having been built under a different version of R; you can ignore this warning should it appear.

## 2.2 Enable ggplot2

To do this, look for ggplot2 in the Packages tab of the bottom right hand pane of RStudio, and simply click the tickbox next to it. Nice and simple. Alternatively you can type `library("ggplot2")` – this does exactly the same as ticking the box.

## 2.3 Great, now you're good to go!

So, what is ggplot2? Well, the 'gg' stands for *grammar of graphics*. In spoken and written languages, grammar allows you to combine lots of little things to make one big, more meaningful thing – a sentence. The package ggplot2 does a similar thing conceptually: it takes several small commands which when combined make professional looking graphs. I like to think of it as building a graph up in several layers which can be built up step-by-step and modified.

## 2.4 Sampling a 'big data' set

You've probably heard of 'big data', which concerns the analysis of very large datasets (e.g. analysing people's Facebook activity). Visualising or analysing massive data sets can be difficult, for instance a scatterplot may have so many points on it that it becomes hard to see what's going on. One way to deal with this problem is to make a more manageable random subset of the large dataset. That's exactly what we're going to do, using a very large data set concerning diamonds. This is a built-in dataset in R, so we don't need to grab any data off the module webpages.

Type `?diamonds` to see the help file that describes the data. By the way, you can always type a `?` immediately followed by a command's name to see its help file, or if you want to search all the help files for a particular term, you can type `??` followed immediately (no space) by the term.

Since 54,000 is far too many diamonds for any non-Kardashian, let's take a random sample of just 1,000 diamonds. We can do this by using the command

```
diamonds1000<-diamonds[sample(nrow(diamonds),1000), ]
```

Here, `sample( )` generates a random selection, without replacement, of 1000 row numbers. These rows are then copied into a new dataframe, `diamonds1000`. Each row gives information about one diamond and, because of the blank after the final comma, ie the "`, ]`" bit, all columns are copied by default. Note that each of you will get a different random subset of (virtual!) diamonds. Have a quick look at the first few rows by running `head(diamonds1000)` and some summary statistics by running `summary(diamonds1000)`.

## 2.5 Scatterplots

To illustrate what ggplot2 can do we'll build a graph showing how the relationship between the petal length and width varies for different diamonds. We'll build this layer-by-layer, which is the natural approach with this package, saving versions that might be useful as we go. The layers typically consist of visual objects called **geoms**, for points, lines, boxes, text etc. Each has associated properties, like colour, size, shape, position etc. These are specified in aesthetic properties called **aes( )**.

Our starting point specifies a dataframe where R should look for the data, and chooses the variables to be plotted:

```
ggplot(diamonds1000, aes(x=carat, y=price))
```

No plot appears; this occurs because we haven't yet said how the data should be plotted (R doesn't know whether it should draw us a scatterplot or a boxplot or something else). We need to add a layer (a *geom*) to tell ggplot what to put in it. To tell ggplot that we'd like to see a scatterplot, we add the command `geom_point()`:

```
ggplot(diamonds1000, aes(x=carat, y=price)) + geom_point()
```

A scatterplot will appear. However, it's committed the sin of not having a title. We can fix that by adding on a `ggtitle` command: add `+ ggtitle("Price and Quality of Diamonds")` to the previous command.

By now you'll have figured out that we build up plots with lots of short commands, adding a little to an old command and checking that what it does is as desired. This is probably a good time to point out that if you're just adding bits to previous commands, you can press the up-arrow on the keyboard when the cursor is in the console to get old commands back as a starting point rather than having to type the same thing in several times. This is quite a time-saver!

Our axis labels aren't too insightful; their names are all in lowercase and we don't have units, so it's not very professional looking. Press the up-arrow in the console to get your previous command on the screen, and then add the following to the end: `+ labs(x="Weight of diamond (carat)", y="Price ($)")`

Now we have a graph that we can be rightly proud of, let's give it a name:

```
plot1<-ggplot(diamonds1000, aes(x=carat, y=price)) + geom_point()
                            + ggtitle("Price and Quality of Diamonds")
                             + labs(x="Weight of diamond (carat)", y="Price ($)")
```

We saw earlier that we also had data for the cut of each diamond (see `?diamonds` for a reminder). Surely that affects the price? We can colour the points to differentiate them by specifying an "aesthetic" for cut.

```
plot2 <- plot1 + geom_point(aes(colour=cut))
plot2
```

This has worked a treat (hopefully!), giving us different colours for different qualities of cut. It's worth noting for future situations that if, instead of words, the cut had been represented by numerical codes (e.g. 1,2,3,4,5 instead of Fair,...,Ideal) then R may have struggled to realise that cut was a categorical rather than a measurement variable. Consequently R would have given us a sliding colour scale rather than all these different colours that are represented as points in the legend. If you ever come across such a case where R erroneously interprets a categorical variable as a continuous one, you can put the variable concerned in a `factor( )` command. That is, if the cut classifications had been numbered 1,2,3,4,5, we would have used `...+ geom_point(aes(colour=factor(cut)))`.

Perhaps it would be better to have separate plots in different panels for different levels of cut?

```
plot2 + facet_wrap(~cut) # Using '~' to split up using the diamonds' cuts
```

Note that you can navigate between all the plots you've made in the plot pane (bottom right) by using the back/forward arrows.

The variant `facet_grid( )` instead of `facet_wrap( )` is also worth noting though perhaps not with 5 plots. Try it!

Alternatively we could use the size of the point to represent cut. How do you think we could do that? Try it and call it plot3! Arguably, a categorical variable like cut is better represented by colour while size is better for representing magnitude.

Now try `plot4<- plot1 + geom_point(aes(colour=cut, size=depth))` and have a look at `plot4`. This plot is now giving us lots of information!

## 2.6 Thinking about regression

So, we've now seen the technicalities of how to get a really nice looking scatterplot with lots of information on. Now let's think about whether we can use the regression models (line fitting) that we've met in the lectures. The first question must always be 'is the relationship linear?'. What do you think from your plot? The relationship between price and carat is similar for different cuts but clearly non-linear. Perhaps taking logs would yield a simpler pattern?

```
plotlog1 <- ggplot(diamonds1000, aes(log(carat), log(price)))
                                + geom_point(aes(colour=cut))
                                + ggtitle("Price and Quality of Diamonds")
```

Notice that we included `colour=cut` inside `aes( )`. Also note that R understood which variable was `x` and which was `y` from their order. Now modify `plotlog1` to make five graphs, one for each cut, in a single plot.

## 2.7 Adding fitted lines

To add fitted lines we need another layer called geom_smooth( ) (so called because lines are smooth whereas points are not!) To fit the least squares regression line we specify `method = "lm"` (lm stands for linear model).

We can choose either to fit a line through the entire data set:

```
plotlog1+geom_smooth(method="lm")
```

or separately to each subset:

```
plotlog1+geom_smooth(method="lm")+facet_wrap(~cut)
```

You may notice in the Fair subplot that there's a shaded region either side of the regression line. This shaded band is a pointwise 95% confidence interval on the fitted values (the line). Were R

to repeat the procedure it used to generate the shaded area (which we'll cover later in the module) numerous times for different samples, the true (population) regression line would lie within the shaded region for 95% of samples. If we don't want to plot the shaded region, we can use `geom_smooth(method="lm", fill=NA)` instead.

As an aside it's interesting to see the effect of moving `colour=cut` into the `geom_smooth( )` in the first command in this subsection (the one for the the whole dataset). This imposes different colours on everything cut-related in particular the fitted lines which, as a by-product, you can see that the lines appear to be roughly parallel.

## 2.8 Least squares estimators, equation of regression line and correlation coeffient

Recall from the lectures the linear regression model: for a given $x_i$ we assume that

$$Y_i = \beta_0 + \beta_1 x_i + E_i,$$

where $E_i$ are independent $N(0, \sigma^2)$ errors for $i = 1, 2, \ldots n$. Since $E_i$ all have expected value 0, we have that $\mathbb{E}[Y_i] = \beta_0 + \beta_1 x_i$, i.e. a straight line. We derived least squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ for the least squares regression line's intercept and slope respectively, finding that:

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where

$$S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2, \quad S_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}), \quad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i. \quad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

These quantities, computed from the data, are tedious to evaluate by hand, although algebraic identities exist that enable them to be computed more easily (see the 'Calculating $S_{xx}$ and $S_{yy}$' section at the bottom of the Regression page on the module webpages).

Of course, doing lots of tedious computations very quickly is easy work for a computer. So how do we get R to compute useful quantities related to regression? Let's load some suitable data. `mathslsd.csv` on the module webpages (`https://stats.vellender.com`) contains some data on the effect of drugs on mathematical ability from a 1968 study by Wagner, Agahajanian, and Bing. A group of volunteers was given LSD, their mean scores on a maths exam (called `maths` in the data) and tissue concentrations of LSD (`lsd` in the data) were obtained at 7 time points. Load it into R and attach the dataframe.

As we've discussed in the lectures, it's always a good idea to do a quick scatterplot first to make sure that a linear relationship seems a sensible fit for the data. Do a quick `plot(maths,lsd)` to see if it looks like a straight line relationship is appropriate (the `plot` command is probably quicker than doing a full nicer-looking `ggplot`).

Now that we've determined that a linear relationship seems sensible, we can get R to fit the linear model. We do this by using the `lm` command. We must decide which of our two variables (`maths` and `lsd`) is the predictor and which is the response. This can depend on context. Using the weight/height example that I'm so fond of in the lectures: if we had some scales but no ruler, we would want to measure weight to predict height (weight is the predictor, height the response), whereas if we had a ruler but no scales, it would be the other way around.

With our LSD data, let's suppose we want to predict someone's LSD muscle concentration by giving them a maths test (which may be cheaper and quicker than conducting medical tests). Then `maths` is the predictor and `lsd` is the response. To fit a linear model, we use a command in the form `lm(response~predictor)`, so in this case `lm(lsd~maths)`. Take note of the order here... to plot we use `plot(x,y)` whereas to fit a regression model of y on x we use `lm(y~x)`.

R outputs the values of the predicted intercept ($\hat{\beta}_0$) and slope, which it calls `maths` here because the slope $\hat{\beta}_1$ is the coefficient that gets multiplied by the $x_i$'s, i.e. the maths scores.

We can get the correlation coefficient, usually denoted $R$, by `cor(maths,lsd)`. The value is negative and not too far from -1; what do you deduce from this? Would it be a good idea to take LSD before your exams? (Perhaps not the most difficult question to answer, that.)

Anyhow, as we've said in the lectures, $R^2$ is often quoted as a measure of the strength of the linear relationship; in particular, $100R^2$ is often quoted as it gives the percentage of the variation in $Y$ accounted for by its linear relationship with $X$. We can calculate this with `100*cor(maths,lsd)^2`

## 2.9 Histograms in ggplot2

So far our data's been on exciting things like diamonds, volcanoes, and making people do maths after taking hallucinogenic drugs (disclaimer: that last one is not advisable and might not be fun – i.e. don't sue me). Let's now calm ourselves down by using data on (the perhaps less sexy) malaria and dogs' breath for this section and the next.

A medical researcher took blood samples from 31 children who were infected with malaria and determined the number of malaria parasites in 1ml of blood for each child. Get the `malaria.csv` data from the module webpages and load it into R as a dataframe called `malaria`. To produce a histogram, we use the same kind of command structure as we have been doing all the time, but with `geom_histogram()`:

```
malariaHist<-ggplot(malaria,aes(parasites))+ geom_histogram( )
```

Add suitable axis labels and a title. How would you describe the shape of this distribution?

Redo the histogram replacing `parasites` with `log(parasites)` in `aes( )` . You can change the width of the bars with `geom_histogram(binwidth= )`. Experiment with different values eg 2, 1, 1.5, 2.5.

### 2.9.1 Histograms of discrete data

When faced with a discrete random variable, such as `clarity` in the diamonds dataframe, `geom_bar` produces a barchart of the counts (`geom_histogram( )` is more suitable for continuous data).

```
ggplot(diamonds1000,aes(clarity))+geom_bar()
```

Compare this with the same barchart for the original dataframe, `diamonds`. Does your sample faithfully represent the distribution in the original population? The following example constructs a barchart for two variables

```
ggplot(diamonds1000, aes(clarity, fill=cut))+geom_bar(position="stack")
```

The default is a stacked barchart. To get side by side use `position="dodge"`.

## 2.10 Boxplots (aka box and whisker plots) in ggplot2

The dataframe `tartar` on the module webpages contains the results of an experiment on treatments for tartar on the teeth of dogs[1]. Twenty-six dogs were used and allocated to one of three groups, a control group with a standard feed, $P_2O_7$ (pyrosulphate added to the feed) and HMP (hexametaphosphate added to the feed). After four weeks each dog was examined and the development of tartar was summarised by an index taking into account the spread and thickness of tartar on the teeth. Inspect the data.

[1] 'My dog has no nose.' 'How does he smell?' 'Terrible' *(but presumably not quite as terrible as that old joke)*

Boxplots are produced by `geom_boxplot( )`:

```
tartplot<- ggplot(tartar, aes(x=treat,y=index))+geom_boxplot()
```

One option is add a layer with the actual points on top of the boxplots.

```
tartplot1<-tartplot+geom_point(colour="red")
```

Note however that a plot of just the points on their own without the boxplot is often not all that useful as some points may overlap. Add suitable labels and title and comment on the plot.

One weakness of a basic box plot is that it does not tell you how the sample sizes differ. The `varwidth` option uses the square root of the sample size to define the width of the box as we see in the following example using the diamonds data.

```
boxofdiamonds<-ggplot(diamonds1000,aes(x=clarity,y=depth))+geom_boxplot(varwidth=T)
```

Experiment with the look of the points. What happens if you swap the roles of `x` and `y` here?

# 3 Exercises

**Question P3.Q1** The table below gives the annual disposable income and annual expenditure on food (both in multiples of £1000) of ten middle-income families of the same size.

| Disposable Income ($x$) | 21 | 18 | 25 | 30 | 16 | 26 | 35 | 22 | 33 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|
| Expenditure on food ($y$) | 4.1 | 3.9 | 4.2 | 5.1 | 3.8 | 4.4 | 5.8 | 3.9 | 4.9 | 4.9 |

Enter the data into a suitable dataframe in R. You can do this by first creating two variables `income` and `expenditure` (use the `c( )` function as in previous practicals) then `food <- data.frame(income, expenditure)`

(a) Create a well-labelled plot of the data using ggplot. Does the linear model seem suitable?

(b) The function `cor(income, expenditure)` calculates the correlation coefficient. Write down its value. What is the value of the 100 times the square of this, and what does it tell you about how effective a straight line regression model will be?

(c) What is the equation of the least squares regression line? If you didn't plot the line in (a), replace that plot with one that *does* include the least squares regression line.

(d) To help understand how the intercept and slope have been calculated, we'll replicate the calculation 'by hand' using formulae you'll find on the module webpages (under 'Linear regression', near the bottom). For this data:

$$\sum_{i=1}^{10} x_i = 255, \quad \sum_{i=1}^{10} y_i = 45.0, \quad \sum_{i=1}^{10} x_i^2 = 6861, \quad \sum_{i=1}^{10} y_i^2 = 206.34, \quad \sum_{i=1}^{10} x_i y_i = 1182.1.$$

Using your calculator, calculate $S_{xx}$ and $S_{xy}$ (write down their values) and hence find (i) the slope and (ii) the intercept of the least squares regression line of $y$ on $x$. Check that the equation of your line agrees with R's.

(e) Use the equation of the line to predict the expenditure of a family whose disposable income is (i) £29,000, (ii) £19,800. How about (iii) £666,000?

(f) A family decreases its disposable income by £2,500. Use your fitted model to estimate the impact this would have on its food expenditure. Is your estimate the same for a family whose disposable income is £29,000 as for one with £19,800? Or is it different? Briefly explain why.

**Question P3.Q2** The following table gives the observed values of Young's modulus ($y$), a measure of material stiffness, for sapphire rods at controlled temperatures ($x°C$)

| $x$ | $y$ | $x$ | $y$ | $x$ | $y$ |
|---|---|---|---|---|---|
| 100 | 4612 | 600 | 4389 | 1100 | 4140 |
| 200 | 4565 | 700 | 4347 | 1200 | 4100 |
| 300 | 4513 | 800 | 4303 | 1300 | 4073 |
| 400 | 4476 | 900 | 4251 | 1400 | 4024 |
| 500 | 4433 | 1000 | 4201 | 1500 | 3999 |

(a) Enter the data into a new dataframe (to create `Temperature` quickly, use `100*c(1:15)`)

(b) Calculate the correlation coefficent between $X$ and $Y$. What does this tell you?

(c) Use R to plot the data with the fitted regression line and write down the equation of the fitted straight line. Does it seem to be a suitable model? Give reasons.

(d) Use the model to estimate the decrease in Young's Modulus when the temperature is increased by 100°C