Using R to generate data for questions

Craig Alexander & Eilidh Jack

Contents

Chapter 1

Overview

In this session, we will take a look at how we can use R to generate data to create questions. The first example will work through question 1 of Paper 1 from the 2021 exam. You can access the paper and solutions by clicking on the links. The second example will introduce how to generate data for a simple linear model.

1.1 Libraries

Throughout this tutorial, we will use some libraries within R. If you are logged into a University computer, please load the following libraries:

```
library(tidyverse)
library(truncnorm)
library(gridExtra)
library(BSDA)
library(ggfortify)
```

If you would prefer to work through the examples on your own device, you will need to install the following libraries:

- `tidyverse`
- `truncnorm`
- `gridExtra`
- `BSDA`
- `ggfortify`

Chapter 2

2021 Paper 1 Example

2.1 Question 1

In this example, we will take a look at question 1 from Paper 1 in 2021. This question is a report style question based on Google AI data of times taken to draw a cat or a dog. The example contains a stem and leaf diagram with the combined data and some summary statistics for both sets of drawings. Following this, a Mann-Whitney Test is carried out to test whether both samples have different average drawing times.

We will now look at how we can simulate a sample of the data from the question, by randomly sampling data for both groups using properties from their summary statistics

2.1.1 Generating a random sample of data

We can generate a random sample of data for both the categories using the summary statistics provided. As we can see from the stem and leaf diagram, the data have a lower bound, where we cannot observe any data below zero, as the data recorded are based on time elapsed.

In order to sample data of this form, we can use a variation of the Normal distribution called the truncated normal distribution, which allows us to bound a Normal distribution through a given range.

To randomly sample data from this distribution, we can use the `rtruncnorm` function from the `truncnorm` package in R as follows:

```
sample_cat <- rtruncnorm(n=121, a=0, b=14, mean=5.4, sd=2.31)
sample_dog <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)
```

The parameters required are defined as follows:

- `n` - the number of samples we wish to draw
- `a` - the lower bound of the distribution (here, we will set this to 0)
- `b` - the upper bound of the distribution (here, we have set this to be the ceiling of the max value for each group)
- `mean` - the mean of each group
- `sd` - the standard deviation of each group

Running the code above will produce a sample for both groups based on their relative summary statistics. We can check the summary statistics of our data using `summary()`

```
summary(sample_cat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3641  3.4919  4.7518  5.0838  6.3575 11.2882
```

```
summary(sample_dog)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2323  6.4623  8.1252  7.9339  9.3019 15.5964
```

We can now compare these summary statistics to those of the real data provided in the paper and notice that our simulated data produces fairly similar summary statistics.

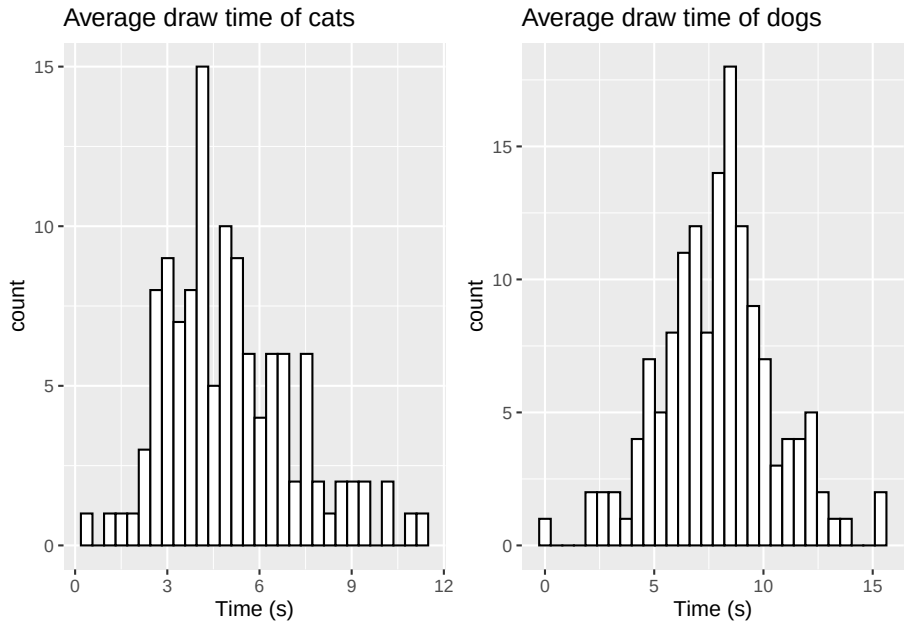
2.1.2 Visualising the data

We can also check the distribution of our sampled data for comparison by visualising it using a histogram. The `ggplot2` library found in the `tidyverse` library provides several functions for data visualisation and has become more popular than base R graphics. We will use the `geom_histogram()` function from the library in this example as follows:

```
dog_hist <- ggplot(data.frame(sample_dog),aes(x=sample_dog))+
  geom_histogram(color="black",fill="white") +
            labs(title="Average draw time of dogs",x="Time (s)")
cat_hist <- ggplot(data.frame(sample_cat),aes(x=sample_cat))+
  geom_histogram(color="black",fill="white") +
  labs(title="Average draw time of cats",x="Time (s)")

grid.arrange(cat_hist, dog_hist,ncol=2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To create these histograms, the code above works in the following fashion:

- We first specify our data using `ggplot(data)`, where our data here is either group of samples.
- To specify which variables we would like to select, we use the `aes()` argument. As we only have one sample of data in each case, we specify this using `x=data`.
- We then generate the histogram using `geom_histogram()`, where we can define the line colour using `color` and the filled colour of the bars using `fill`.
- We can label our plot using the `labs` argument, where we can include a `title` and an `x` axis label

We can also alter the number of bins we use (`ggplot` will set a standard number of bins by default) using the `bins` argument. Let's alter the number of bins for the dog data to be 10

```
dog_hist <- ggplot(data.frame(sample_dog),aes(x=sample_dog))+
  geom_histogram(color="black",fill="white",bins=10) +
  labs(title="Average draw time of dogs",x="Time (s)")
```

2.1.3 Setting random seeds for reproducibility

When we randomly sample data each time in R, we will obtain a different sample than before. Let's run our previous code twice to see if there is any differences:

```
sample_dog1 <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)
sample_dog2 <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)

summary(sample_dog1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4464  5.7736  7.3936  7.5150  9.2837 13.4899
```

```
summary(sample_dog2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2391  5.3456  7.5616  7.4537  9.2504 13.4568
```

We see that both samples produce different summary statistics. This can cause difficulty when you are working on a specific problem and want to design questions around the specific characteristics of the data you have sampled the first time.

We can force R to use the same random number generation by using the `set.seed()` function. Here, we specify the seed from the random number generator we want to use each time we generate samples. This number can be any number you wish to choose! The example below highlights how this works:

```
set.seed(2023)
sample_dog1 <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)
set.seed(2023)
sample_dog2 <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)

summary(sample_dog1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.006   6.316   7.545   7.798   9.365  14.776
```

```
summary(sample_dog2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.006   6.316   7.545   7.798   9.365  14.776
```

Here, we see we can produce the same data as the first sample by setting the seed prior to sampling.

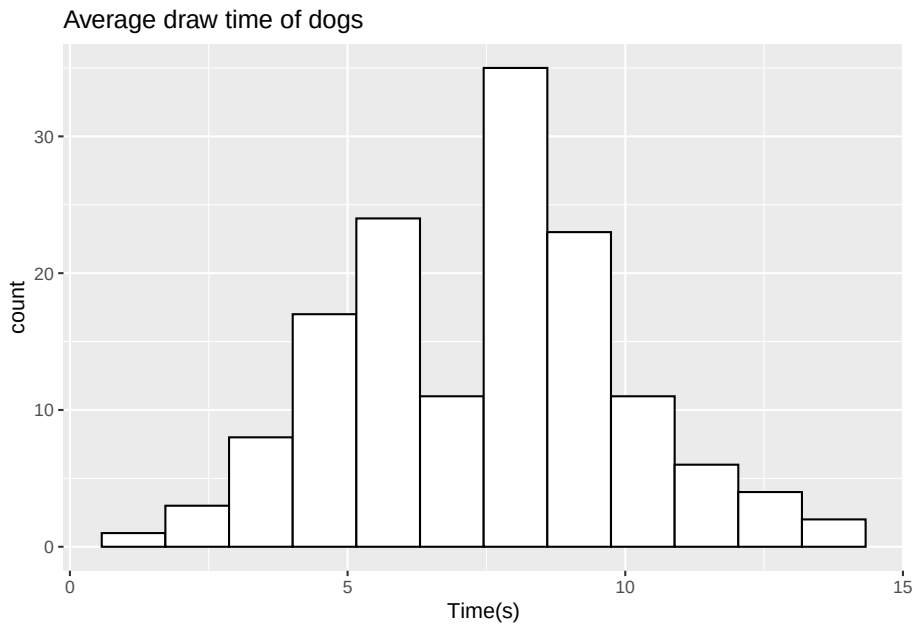
2.1.4 Manually editing data

If we take a sample of data and perhaps wish to add some additional variables to mimic the original data closer, this can easily be done in R. Let's take a sample for the dog data but lower our boundary to 14 and visualise.

```
sample_dog <- rtruncnorm(n=145, a=0, b=14, mean=7.5, sd=2.66)

ggplot(data.frame(sample_dog),aes(x=sample_dog)) +
```

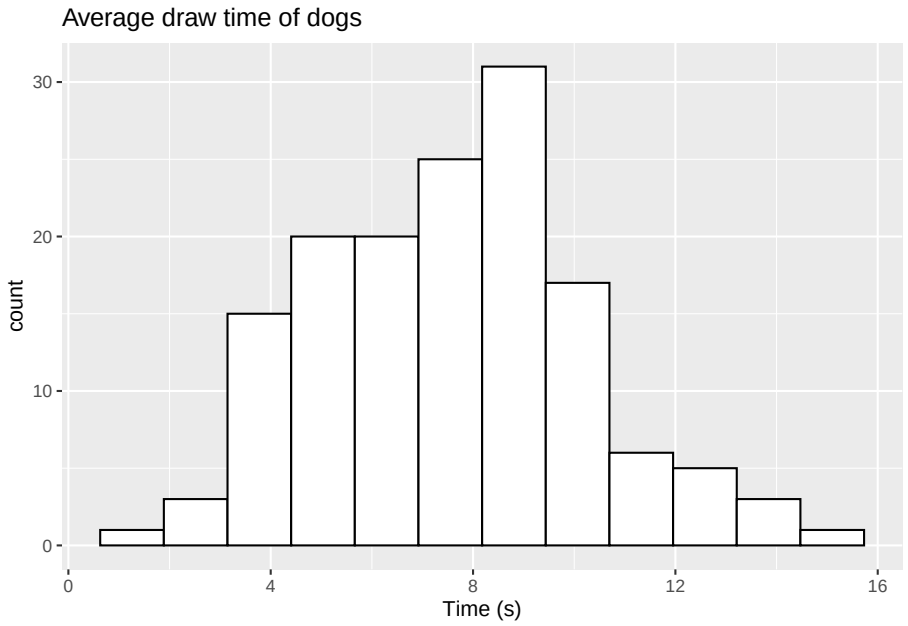
```
  geom_histogram(color="black",fill="white",bins=12) +
            labs(title="Average draw time of dogs",x="Time(s)")
```



When comparing this to the original data from the paper, we see that we do not observe the two outliers at 14.3 seconds and 15.2 seconds. We can manually add these values as follows:

```
sample_dog <- c(sample_dog, c(14.3,15.2))

ggplot(data.frame(sample_dog),aes(x=sample_dog)) +
  geom_histogram(color="black",fill="white",bins=12) +
            labs(title="Average draw time of dogs",x="Time (s)")
```



We now observe the two additional values in the histogram. For a vector of data, we can easily add new values using the `c()` command. This can be done by specifying our original data first, and then including our additional variables in a new `c()` object. If we only wish to add one value, we do not need to use `c()`.

2.1.5 Sampling from different distributions

In this case, the data shown roughly takes the form of a Normal distribution. There are several cases where this may not be an appropriate distribution. R contains a full range of different distributions we can sample from. Some common choices are shown below:

- `rbinom()` - Sample from a Binomial distribution
- `rchisq()` - Sample from a Chi-squared distribution
- `rgamma()` - Sample from a Gamma distribution
- `runif()` - Sample from a Uniform distribution
- `rexp()` - Sample from an Exponential distribution

This list is not exhaustive, and R will contain libraries that will allow sampling from almost any known distribution.

2.1.6 Performing a Mann-Whitney Test

We can carry out a Mann-Whitney test in R as follows

```
sample_dog <- rtruncnorm(n=145, a=0, b=16, mean=7.5, sd=2.66)
sample_cat <- rtruncnorm(n=121, a=0, b=14, mean=5.4, sd=2.31)

mann_whitney <- wilcox.test(sample_dog,sample_cat)
mann_whitney
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  sample_dog and sample_cat
## W = 12823, p-value = 9.048e-11
## alternative hypothesis: true location shift is not equal to 0
```

The output from this test gives us the rank sum `W` and a p-value which can be used for question creation

2.2 Performing a z-test

We can perform a two-sample z test in R using the `z.test` function from the `BSDA` package. We can carry out this test as follows:

```
z.test(sample_cat,sample_dog,sigma.x=2.307,sigma.y=2.655)
```

```
##
##  Two-sample z-Test
##
## data:  sample_cat and sample_dog
## z = -6.7694, p-value = 1.293e-11
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.656354 -1.463513
## sample estimates:
## mean of x mean of y
##  5.308262  7.368196
```

Here, we specify the sample standard deviation for cats and dogs respectively. The output from the test gives us the z-statistic, p-value and 95% confidence interval.

Chapter 3

Linear model example

In this section we will introduce how to generate data for a linear model example using the linear model equation.

This example is based on the following question.

3.1 Question

A global ice cream company is interested in using linear regression to predict its ice cream waste (kg) based on the average temperatures of its store locations (°C).

The estimated linear model is:

$$\hat{waste} = 597 - 10.91 \times temperature.$$

1. What is the predicted waste for a store location whose average temperature is 18°C? *Provide your answer to two decimal places.*

2. The standard error for the slope coefficient is 0.92, which is associated with $df = 32$. Calculate a 95% confidence interval for the slope parameter. *Provide your answer to two decimal places.*

Lower: Upper:

3. Suppose you performed a hypothesis test to test if average temperature is a significant predictor of ice cream waste. Working at a significance level of 5%, would you expect the p-value of the hypothesis test to be:

Cannot tell with the information providedgreater than 0.05less than 0.05

3.2 Generating data

In order to fit this linear model we need to generate two variables:

- Response variable (y): *waste* and
- Explanatory variable (x): *temperature*.

Crucially, these variables need to be associated with each other as shown in the linear model equation.

3.2.1 Explanatory variable

Let's start by generating our x-variable, *temperature*. We will use a Normal distribution to do this since our variable is continuous.

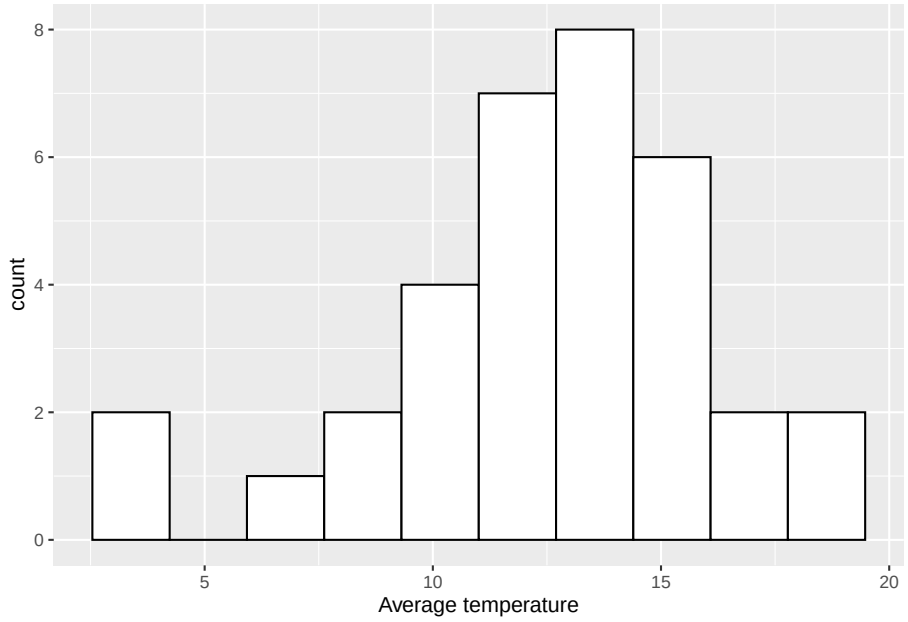
```
temp<-rnorm(n=34, mean=12, sd=4)
```

The parameters required are defined as follows:

- `n` - the number of samples we wish to draw. In our question $df = 32$, so $n = 34$ (since for a simple linear model $df = n - 2$).
- `mean` - the mean temperature which our data will be centered on. Pick something sensible here, we have gone for 12°C.
- `sd` - the standard deviation for *temperature*. Again go for something that seems sensible (high enough to show some variability, but not too high that the values at the extreme are no longer sensible). This can take a bit of trial and error.

Let's take a look at our simulated data to check that is looks sensible. To do this we will the `geom_histogram()` function that we used in our previous example.

```
ggplot(data.frame(temp),aes(x=temp)) +
  geom_histogram(color="black",fill="white", bins=10) +
            labs(x="Average temperature")
```



Exercise Change the values of the `mean` and `sd` in the code above and plot a histogram of the newly simulated x value to see how the data changes.

3.2.2 Using a linear model to generate the response

In order to generate the response variable y we need to choose a sensible linear model which will relate the explantory variable x to the response variable y.

To specify a simple linear model we need to choose two values, the intercept, α, and the slope, β. In this example we want to estimate ice cream waste from average temperature, we are therefore going to assume there is a *negative* relationship between these variables, i.e. the warmer it is, the more ice cream you will sell! Therefore, the slope parameter (β) should be negative. The size of this will depend on what you think is a sensible interpretation. Here let's go for -10.65, so for every unit increase in average temperature, ice cream waste decreases by 10.65kg. For the intercept, again think about a sensible value based on the context. Remember this value is where the regression line will cut the y-axis. Let's choose 591. The choice for α and β will be subjective, and might take some trial and error before deciding on your final values. Just make sure that the interpretation of the final model is sensible.

We can now generate the response variable, y as follows:

```
waste<-591 - 10.65*temp
```

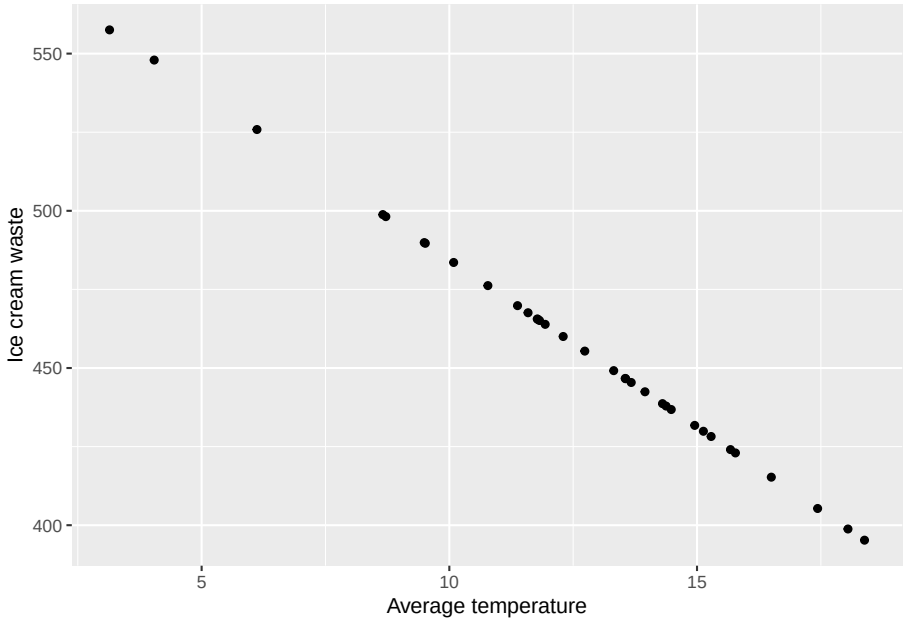
where

- 591 is our chosen value for α,
- -10.65 is our chosen value for β,
- `temp` is the x-variable we have already simulated.

Let's now produce a scatterplot of our data using `ggplot2`. We will use a similar structure to the plots we have produced previously but we will use the `geom_point()` function to produce a scatterplot as follows:

```
data<-data.frame(waste=waste, temp=temp)

ggplot(data, aes(x=temp, y=waste)) +
  geom_point() +
  labs(x="Average temperature", y="Ice cream waste")
```



To create this scatterplot we first had to store our two variables in a *dataframe* which we called `data`. Here we are creating a dataframe with two columns which we have given the same names as the two data vectors we have already created, namely `waste` and `temp`. In these columns we are simply storing the data vectors we have simulated above. We then create the scatterplot which works as follows:

- First we specify our data using `ggplot(data)`.
- We then specify the variables we want to plot using the `aes()` argument, where `x=temp` and `y=waste`.
- We then generate the scatterplot using `geom_point()`. Here we have used the default values so we don't need to specify anything within this function.
- Finally, we label our x and y axis using the `labs` function.

Now, you may have noticed that our scatterplot has identified an issue with the data we have generated. The code we have used has produced a perfect linear relationship between *temperature* and *waste*. This does not seem realistic so we need to add some random noise to y. We can do this by adding some randomly generated values from a normal distribution with mean 0. Think of this as adding residuals to our model.

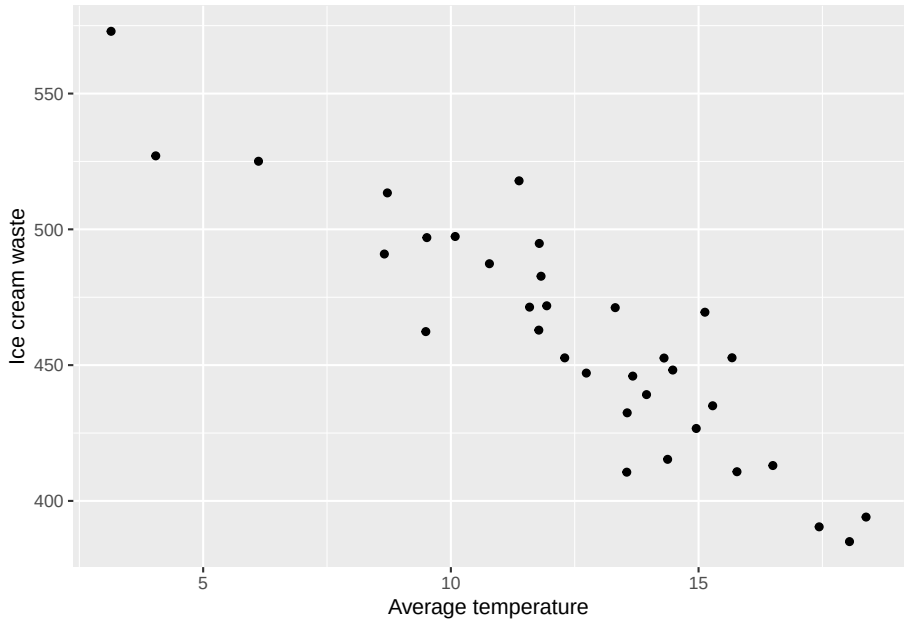
```
waste<-591 - 10.65*temp + rnorm(n=34, mean=0, sd=20)
```

We always want the model residuals to be centered on 0, so the mean within `rnorm()` should be 0. The level of variability within y can be controlled via the standard deviation (`sd`). Again, this might take some trial and error to get something you are happy with.

Let's take a look at the scatterplot with our newly generated response.

```
data<-data.frame(waste=waste, temp=temp)

ggplot(data, aes(x=temp, y=waste)) +
  geom_point() +
  labs(x="Average temperature", y="Ice cream waste")
```



That looks better!

Exercise 1. Change the values of α and β in the code above to see how the relationship between x and y changes. 2. Alter the value of `sd` in the code above to control the level of variation in the y.

3.3 Fitting a linear model

Now that we have simulated values for *temperature* and *waste* we can now fit a linear model and generate output that can be used to create questions.

```
model<-lm(waste~temp)
summary(model)
```

```
##
## Call:
## lm(formula = waste ~ temp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -38.514 -11.412  -2.084  10.421  45.052
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 596.9391    11.8702   50.29  < 2e-16 ***
## temp        -10.9101     0.9153  -11.92 2.62e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.85 on 32 degrees of freedom
## Multiple R-squared:  0.8162, Adjusted R-squared:  0.8104
## F-statistic: 142.1 on 1 and 32 DF,  p-value: 2.62e-13
```

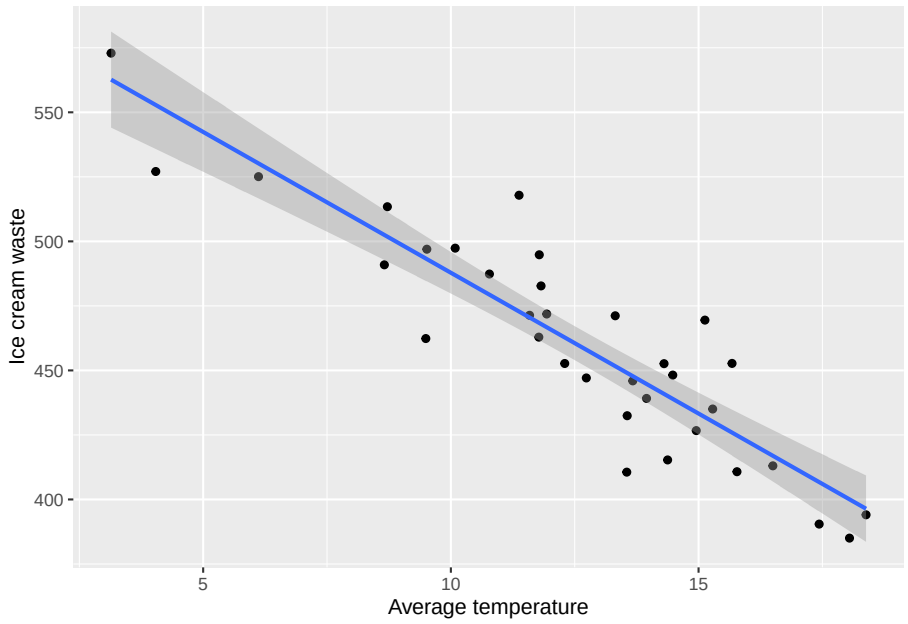
Note that the estimates for α and β are slightly different to the values we used to generate the data. This is because of the random noise we added when generating our response variable. The values are fairly similar though. Notice how these values correspond with the linear model used in the question at the top on the page, and how we have used the standard error from the output table when asking for a 95% CI for the slope parameter (although this may be beyond the scope for the AH stats cirriculum).

3.4 Adding the linear model line to a scatterplot

We can now add our fitted model and 95% confidence interval to the scatterplot of our data using the following code.

```
ggplot(data, aes(x=temp, y=waste)) +
  geom_point() +
  labs(x="Average temperature", y="Ice cream waste")+
  geom_smooth(method="lm", se=TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



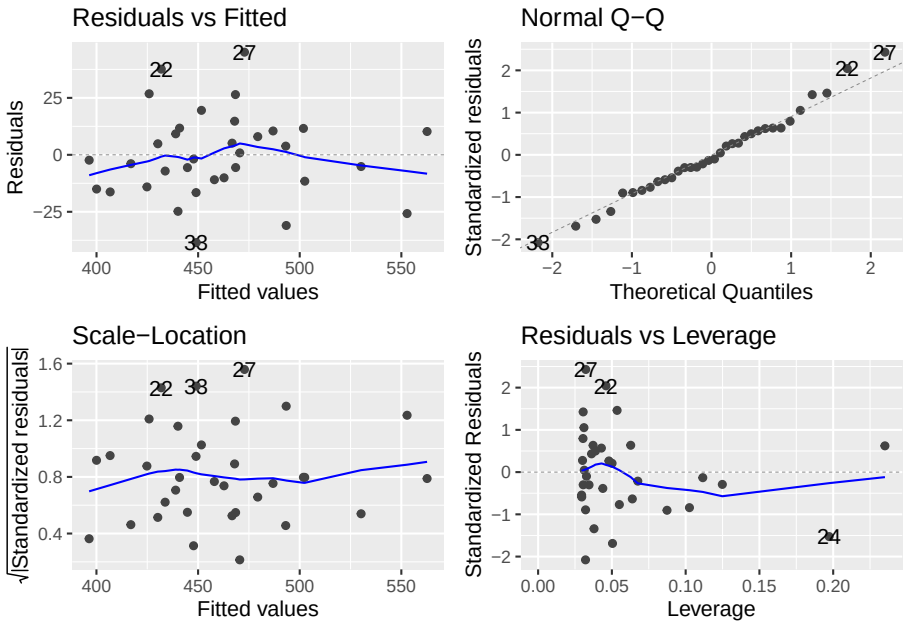
To generate this plot we have used the same code as previously and added the `geom_smooth()` with the following arguments:

- `method = lm` - fit a linear model line to the plot.
- `se = TRUE` - display a confidence interval around the line. This can be removed by setting `se=FALSE`.

3.5 Producing model diagnostics

We can use the `autoplot()` function in the `ggfortify` library to create diagnostic plots for our linear model using `ggplot2` as follows:

```
autoplot(model)
```



3.6 Calculating quantities of interest

You can also calculate quantities such as S_{xx}, S_{yy} and S_{xy} to be used in questions for calcuating estimates by hand.

```
S.xx <- sum((temp - mean(temp))^2)
S.yy <- sum((waste - mean(waste))^2)
S.xy <- sum((temp - mean(temp)) * (waste - mean(waste)))

r <- S.xy/(sqrt(S.xx*S.yy))
beta <- S.xy/S.xx
alpha <- mean(waste) - beta*mean(temp)

r
```

```
## [1] -0.9034156
```

```
beta
```

```
## [1] -10.91007
```

```
alpha
```

```
## [1] 596.9391
```

3.7 Summary

Using R to generate data can be a quick and simple way to create practice questions for students. It can also be helpful to demonstrate certain properties when introducing concepts in class. Another benefit is to allow the creation of multiple variants of the same question, all with slighty different data, and therefore slightly different answers. This can be particularly helpful for revision questions that students can attempt more than once, or in assessments so that all students have slightly different questions/answers.

However, in general we would advise using real data when introducing/demonstrating concepts in class as this allows students to appreciate the use of statistics in real-life scenarios and also allows for discussions around dealing with messy data.