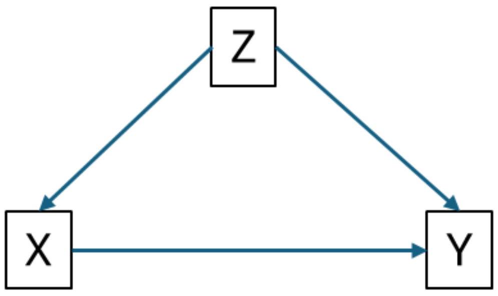
Demonstration 1: Confounding.

We aim, here, to demonstrate the effect that an unobserved confounder might have on regression estimates. With regression we might be interested in the effect of X on Y. But there might be an unobserved variable, Z, which affects both X and Y. This can be visualised with this path diagram:



For instance, if X is education and Y is earnings, Z might be a measure of intelligence – it might be that smarter people are more likely to do well in school, and independent of that be more likely to earn more. In that case, a correlation would be generated between education and earnings, even if additional schooling had no impact on earnings at all.

We start by clearing our data (to make sure nothing is hidden that we don't know about), and then setting the size of the dataset – we'll do 1000 observations for now

```
obs <- 1000
```

We are going to be storing our simulation inside of a tibble object (a type of data frame used by the tidyverse) called `sim`, so we're going to start by using the tibble function

```
sim <- tibble(

  ...

)
```

In order to simulate data which matches this, we need to start by generating the variable(s) that (in this simplified version of the world at least) isn't caused by anything else – that is, Z

For now we will do this with Z being drawn from a random normal distribution, with a mean of zero and a variance of 2. You might want to adapt this depending on what you imagine the variable being. For instance, if you wanted to generate a variable with a uniform distribution, you would use `runif()`

```
sim <- tibble(
  z = rnorm(obs, 0, 2)
  ...
)
```

That is, Z is 1000 observations drawn from a normal distribution with a mean of 0 and a standard deviation of 2. Now we have Z, we are able to generate the other variables. We can generate X as a function of Z

```
sim <- tibble(
  z = rnorm(obs, 0, 2),
  x = 0.5*z + rnorm(obs, 0, 2)
  ...
)
```

And Y as a function of both X and Z. We also want to add some random noise to Y to represent measurement error/sampling error/unexplained variance. We can add e to our tibble at the same time. Now our simulation is ready to be run.

```
sim <- tibble(
  e = rnorm(obs, 0, 4),
  z = rnorm(obs, 0, 2),
  x = 0.5*z + rnorm(obs, 0, 2),
  y = 1 + 2*x + 2*z + e
)
```

Again, we might choose to vary the parameters we are using here, in order to mimic a particular real-life scenario.

Now that we have all our variables created, we can run our model. Remember, we are envisioning that Z is unmeasured (even though here we have measured it!) so we wouldn’t be able to include that in our model

```
mod <- lm(data = sim, y ~ x)
summary(mod)
```

We know that the true value here should be 2 (as that is what we put in our code when defining Y). In this case, we would expect there to be bias in the estimate of the effect of X because of the confounding that we have simulated in our data.

To fix this, we could control for Z (of course, this might not be possible if Z is unmeasured, but in simulation world, we can still do this!)

```
mod <- lm(data = sim, y ~ x + z)
```

```
summary(mod)
```

We would expect this model to be unbiased in the estimate of Z – that is it should be approximately 2 – although because we are simulating our variables from random distributions, it won't be exactly 2.

Check: do your results match your expectations?

It is likely that we might want to test different levels of confounding – that is, what happens when there is no confounding, a small amount of confounding, lots of confounding, etc. We could do this by repeating the above code, but changing the size of the effect of Z on Y: e.g. if we replaced Y as followed:

```
y = 1 + 2*x + 0*z + e
```

We would expect our estimate of the effect of X to be relatively unbiased – that is fairly close to 2. However if we generated Y with a really large Z effect

```
y = 1 + 2*x + 4*z + e
```

We'd expect significant bias in our estimates of the effect of Z.

It's a bit time consuming to do loads of different repetitions (particularly if we are testing lots of combinations of estimates). So we might utilise loops to automate the running of multiple models for us. In this case, that might look like this:

```
set.seed(12345)
for (b2 in seq(0, 3, 1)) {
  sim <- tibble(
    e = rnorm(obs, 0, 4),
    z = rnorm(obs, 0, 2),
    x = 0.5*z + rnorm(obs, 0, 2),
    y = 1 + 2*x + b2*z + e
  )
  mod <- lm(data = sim, y ~ x)
  print(coefficients(mod))
  mod <- lm(data = sim, y ~ x + z)
  print(coefficients(mod))
}
```

So, what is happening here? The first line introduces a for loop – it says that for values of B2 between 0 and 3 (that is: 0, 1, 2, and 3) do the things that are within the {} parentheses. What is within those parentheses is the exact code that we went through above. The only difference is that in defining Y, the coefficient associated with Z is replaced with `` `b2' ``. It effectively takes the value of B2 for that round of the loop. As such, this code will repeat the simulation 4 times, for each of the four specified values of B2. It will print out the coefficients for two regression models each time, one where Z is excluded from the model and one where it is included.

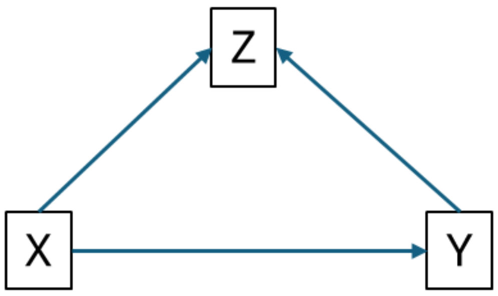
Have a look at the results from these regressions.

Do the results match what you would have expected? That is, does there appear to be more bias for bigger values of B2?

There estimate when B2 = 0 is not exactly 2 – that is it isn't getting the truth perfectly. Can you explain why?

Exercise 1: Colliders

We now want you to repeat this exercise, but with a different DGP. That is instead of an unmeasured confounder, we now want to introduce a collider. A collider is a variable that is caused by two other variables. In the below case, Z is caused by both X and Y.



For instance, if X is education and Y is earnings, Z might be a measure of occupational prestige.

The process for this is the same as before:

- Start with the variable that doesn't have any arrows going into it, and create a simulated version of that
- Then generate those that are only produced by that variable, and so on until all the variables are generated
- You can then attempt to run the regression model of interest, to see how including Z in your model (or not) affect the estimation of X.

Demonstration 2: a full simulation study using confounding

In this session, we're going to extend the code we used before, and extend it to create a full simulation study. That is, rather than just running a single simulation, we're going to do multiple simulations (in this case, 100), collate the results, and then find things out about those simulations (bias, RMSE, optimism, etc).

First, we're going to create an empty tibble (data frame) where we'll be storing the results of our simulations as well as some other useful information and various statistics required for calculating our quantities of interest. In the model we're interested in there will be a total of 4 things that we will be estimating – the intercept, the slope, and the standard errors of each of those. We will also want to keep a record of the different values of B2 that we are predicting for for different simulations. We'll also set our number of observations and our "true" value for b1 as objects so they can easily be changed if we want to

```
obs <- 1000

b1true <- 2

results <- tibble(

  iteration = numeric(),

  sample_size = numeric(),

  b2true = numeric(),

  b1 = numeric(),

  b0 = numeric(),

  b1se = numeric(),

  b0se = numeric()

)
```

Next, we're going to set the seed. The problem with simulation studies is that it's surprisingly difficult for computers to generate genuinely random data. The data that is generated is made to appear random, and that's mostly fine for our purposes. But in order to do this, it takes an initial value (or seed) and does some complex maths to that as the basis of the subsequent variable generation. This is actually helpful for us – it means if we run the analysis with the same seed twice, we'll get the same answer, meaning our results can be replicated.

We can set the seed to anything that we want – here I arbitrarily choose 2345

```
set.seed(2345)
```

We’re then ready to run some code that runs our models as before. This is exactly the same as the code we ran in the previous session, but we’ve now added an additional loop within the loop (in red) – that is, for each of the four values of B2, we’re going to generate the data and run the model 300 times.

```
for (b2 in seq(0, 3, 1)) {
  for (iteration in 1:300) {
    # simulate the data
    sim <- tibble(
      e = rnorm(obs, 0, 4),
      z = rnorm(obs, 0, 2),
      x = 0.5*z + rnorm(obs, 0, 2),
      y = 1 + b1true*x + b2*z + e
    )
    # create a regression model
    mod <- lm(data = sim, y ~ x)
  }
}
```

All good – but we forgot to store the estimates! So we need to add some extra code in (in red below)

```
for (b2 in seq(0, 3, 1)) {
  for (iteration in 1:300) {
    # simulate the data
    sim <- tibble(
      e = rnorm(obs, 0, 4),
      z = rnorm(obs, 0, 2),
      x = 0.5*z + rnorm(obs, 0, 2),
      y = 1 + b1true*x + b2*z + e
    )
    # create a regression model
    mod <- lm(data = sim, y ~ x)

    res <- c(
      iteration = iteration,
      sample_size = obs,
      b2true = b2,
      b1 = coefficients(mod)["x"][[1]],
      b0 = coefficients(mod)["(Intercept)"][[1]],
      b1se = sqrt(diag(vcov(mod)))["x"][[1]],
      b0se = sqrt(diag(vcov(mod)))["(Intercept)"][[1]]
      )

    # append our results to our results dataframe
    results <- results %>% bind_rows(res)

  }
}
```

This additional code does the following:

- Creates a vector with named values using the same names as the tibble we initialised earlier.
- Adds the iteration number, the sample size, and the true value of b2 (in case we want multiple versions with different things varying)
- Extracts the beta estimates from the model
- Extracts the uncertainty measures — these come from a variance covariance matrix in the model object so need to be square rooted in order for them to be standard errors. There are other ways to get the standard error you could use, like taking them from a summary(model) object.
- Appends the results to our results tibble that we made earlier using bind_rows.

So this code will run 1200 models, 300 for each of the four values of B2. For each, it will generate new data, run the model, extract the results from that model, and store them as a line in the matrix A. And it does it all (on my computer at least) in under 3 seconds!

We could now look at the results table:

```
results
```

Take a look at the dataset that has been created. Is it the same length as you expect (it should be 1200)? Are there the right number of columns (should be 7)?

We can now calculate our measures of interest about these variables. For this, we want to calculate a value for each of the true values of B2, we can achieve this by using the `group_by` and `summarise` functions from the tidyverse (this will make each calculation be run per group of unique values of the variable inside `group_by()`. So we could use this code to calculate bias:

```
bias_measures <- results %>%
  group_by(b2true) %>%
  summarise(
    bias = mean(b1) / b1true,
    ...
  )
```

That is, we are taking the average estimate of B1 (for each value of B2) and then dividing it by the true value of B2 (2). In other words, we are calculating the ratio between the average estimate and the truth.

Next we could do the same with Root Mean Square Error (RMSE)

```
bias_measures <- results %>%
  group_by(b2true) %>%
  summarise(
    bias = mean(b1) / b1true,
    rmse = sqrt(mean((b1 - b1true)^2)),
    ...
  )
```

Here, we are calculating the deviation of each estimate of B1 from the truth of 2, squaring it (so they are all positive), again taking the mean for each value of B2, and finally square rooting that number to get back to the coefficients' original scale.

Lastly for this exercise, we calculate optimism. This is effectively a measure of bias for the SE rather than the beta estimate. However, the code is a little bit more involved, since the "true" value of the SE needs to be derived from the estimates, as well as the estimated values.

To calculate the true values, we want to find the mean deviation of the B1 estimate from the mean B1 estimate. To do this we calculate that deviation, square it to make it positive, take the mean for each value of B2, and then square root back to the original scale:

```
(sqrt(mean((b1-mean(b1))^2)))
```

To calcuate the estimated SEs, we do something similar: to get the scale right, we need to square, mean and then square root back again, to match the process for calculating the true SE

```
(sqrt(mean(b1se^2)))
```

We can then calculate the optimism as the ratio between these two values.

```
bias_measures <- results %>%
  group_by(b2true) %>%
  summarise(
    bias = mean(b1) / b1true,
    rmse = sqrt(mean((b1 - b1true)^2)),
    optimism = (sqrt(mean((b1-mean(b1))^2))) /
               (sqrt(mean(b1se^2)))
  )
```

Now we can print the summary dataset object we just created to have a look at the results”

```
bias_measures
```

So that’s an awful lot of work to get just a few numbers!

We can now think about how to present our results. It might be that we just want to present our results in a table. But we might also want to plot them. Some examples are below:

```
bias_measures %>%
  pivot_longer(-b2true) %>%
  mutate(
    target = case_when(name == "bias" ~ 1,
                       name == "rmse" ~ 0,
                       name == "optimism" ~ 1)
  ) %>%
  ggplot() +
  aes(x = b2true, y = value) +
  geom_point() +
  geom_line() +
  geom_hline(aes(yintercept = target)) +
  facet_wrap(~name)


results %>%
  ggplot() +
  geom_density(aes(x = b1)) +
  geom_vline(xintercept = b1true) +
  facet_grid(rows = vars(factor(b2true)))
```

Have a think about what you want graphs to look like, and create a graph that fits the brief.

Exercise 2: a full simulation study of collider bias

Your task, now, is to develop a full simulation study for the collider bias example. That is, what does *controlling for Z* do, with different levels of collider bias. The process will be the same as that undertaken above for the confounding example.

- Take the code you developed this morning with collider bias, with a model that includes both X and Z are independent variables and Y as a dependent variable
- Add an additional for loop to repeat it 100 times (it might be wise to start with just 10 iterations to ensure it works)
- Create an empty results tibble at the beginning of this code, to store your estimates. Note that, because the model stores more things (estimates associated with the intercept, X *and Z*) the matrix will need to be bigger.
- Add some code within the loop to ensure that the results are stored in the results tibble (make sure they are appended, rather than overwriting previous iterations!)
- Check that this code works and creates a results tibble that looks right (correct number of rows and columns)
- Based on the confounder example, write some code that extracts the results, summarises them into measures of bias, RMSE and optimism, and then plot those results.
- *Do the results match your expectations?*

If you have time, or want to play with this at home, you could think about the following

- What happens if you vary the sample size (ie "`obs <- 100`" or "`obs <- seq(10, 100, 10)`" where obs is the thing that changes instead of b2)?
- Could you simulate some data that looks a bit like data you are using? What things would you like to experiment with / vary?
- What are the plots/tables that you would like to create from your simulation? How would you produce those?