

Simulating Complex Data: Start with a DAG

MATH 4939

January 2026

Last update: January 18 2026 21:39

Contents

1	Setting up a DAG	3
2	Write a program to generate data	8
3	Graphic presentation	36
4	Exercises	41

List of Figures

1	A DAG	4
2	A DAG with parent-less variables	5
3	A DAG with unique independent "error" variances for each variable	6
4	A DAG with causal coefficients	7
5	$\hat{\beta}_x \pm SE(\hat{\beta}_x)$ with $\beta_x = 1.6$	42
6	$\hat{\beta}_x \pm SE(\hat{\beta}_x)$ with $\beta_x = 1.6$, focusing on 1.6	43

List of Tables

1	Estimates of $\hat{\beta}_x$	34
2	Estimates of $\hat{\beta}_x$	35

1 Setting up a DAG

Start with a DAG

```
include_graphics('simdag1.png')
```

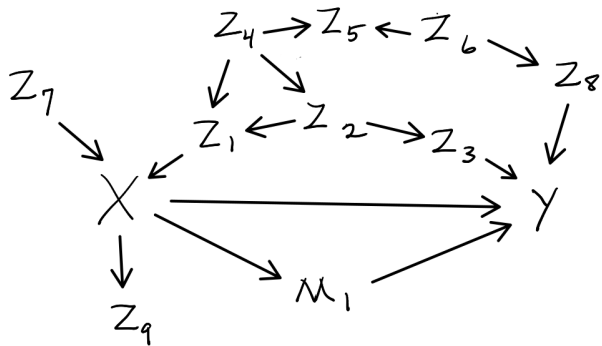


Figure 1: A DAG

Identify variables (nodes) without parents???

```
include_graphics('simdag3.png')
```

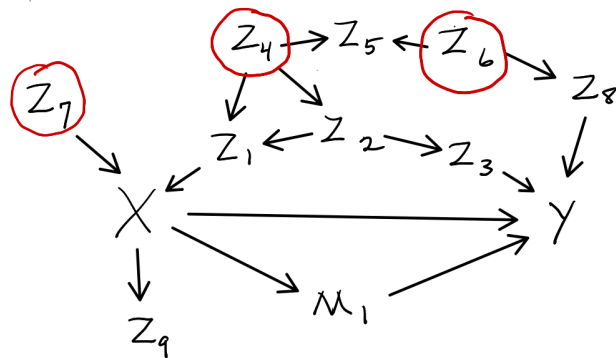


Figure 2: A DAG with parent-less variables

Decide on unique variance for each variable

```
include_graphics('simdag4.png')
```

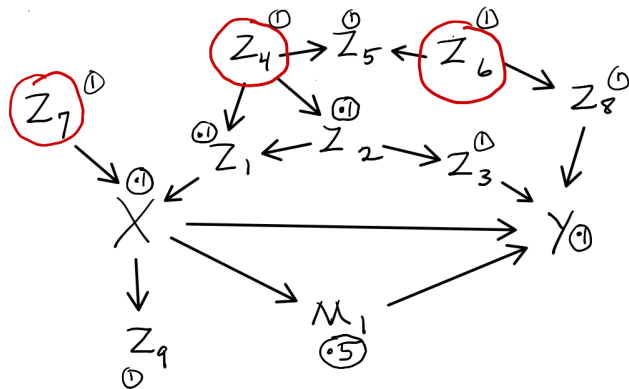


Figure 3: A DAG with unique independent "error" variances for each variable

Decide on causal coefficients

```
include_graphics('simdag5.png')
```

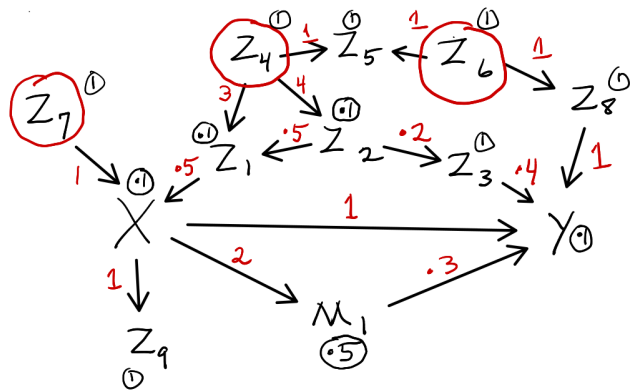


Figure 4: A DAG with causal coefficients

2 Write a program to generate data

If you plan to vary some parameters, include them among parameters of your function. You should probably always make the sample size a parameter.

```
gen <- function(N, seed = NULL) {  
  
  # N:      sample size  
  # seed: optional random seed for reproducibility, default: current seed  
  
  if(!is.null(seed)) set.seed(seed)  
  
  # create a data frame with one of the parent-less variables  
  
  dd <- data.frame(z4 = rnorm(N))  
  
  # generate remaining variables making sure, of course, to generate  
  # all of a variable's parents before generating each variable itself.  
  # You can use 'within' in base R or 'mutate' in 'dplyr'.  
}
```

```
dd <- within(  
  dd,  
  {  
    z2 <- 4 * z4 + 0.1 * rnorm(N)  
    z1 <- 3 * z4 + .5 * z2 + 0.1 * rnorm(N)  
    z6 <- rnorm(N)  
    z5 <- z4 + z6 + rnorm(N)  
    z8 <- z6 + rnorm(N)  
  
    z7 <- rnorm(N)  
    x <- 1 * z7 + .5 * z1 + 0.1 * rnorm(N)  
    z9 <- x + rnorm(N)  
    m1 <- 2 * x + 0.5 * rnorm(N)  
  
    z3 <- .2 * z2 + rnorm(N)  
  
    y <- z8 + .4 * z3 + x + .3 * m1 + 0.1 * rnorm(N)
```

```
    }  
  )  
  return(dd) # not essential to use 'return' but good style  
             # according to the Google style sheet.  
}
```

Test it

gen(2)

	z4	y	z3	m1	z9	x	z7
1	0.4567741	1.411973	1.4096053	3.055266	0.07205968	1.308341	0.1591697
2	-1.2899697	-4.396043	-0.2442343	-2.874327	-0.42334624	-1.840181	1.1997312

	z8	z5	z6	z1	z2
1	-1.398159	0.7989613	-0.8393987	2.225992	1.815585
2	-1.483633	-2.1669816	-1.6710456	-6.372441	-5.117562

gen(3)

	z4	y	z3	m1	z9	x	z7
1	-1.9944765	-8.3561449	-2.2047212	-10.2333492	-5.9984083	-5.4122466	-0.5156033
2	-1.0967523	-3.7430409	-1.4528013	-4.1921542	-0.5048275	-1.9559680	0.7151142
3	0.2573146	0.6538282	-0.8348407	0.7714436	-0.2252675	0.4768678	-0.1232570

	z8	z5	z6	z1	z2
1	0.93870747	-3.3476935	0.2764679	-9.973325	-7.906066
2	0.04750681	-1.5308173	-0.2857255	-5.376385	-4.321405

```
3 0.23879426 -0.4557481 -0.5460521 1.311060 1.159410
```

```
gen(2, 874432)
```

	z4	y	z3	m1	z9	x	z7
1	-0.6425993	-2.176644	-0.93561809	-0.8272788	-0.0813327	-0.4272518	1.1036912
2	-0.4352907	-1.942288	-0.05057259	-3.9698085	-2.5692993	-1.8564261	-0.8091094

	z8	z5	z6	z1	z2
1	-1.053950	-1.287208	-0.6007308	-3.064878	-2.53848
2	1.088388	1.300774	0.4431584	-2.168651	-1.77171

```
gen(4, 874432) # first 2 same as previous
```

	z4	y	z3	m1	z9	x	z7
1	-0.6425993	-4.3552290	0.28988441	-7.204441	-4.6634468	-3.0976540	-1.3842679
2	-0.4352907	-1.3743454	0.79165036	-1.281635	-0.3594277	-0.7310431	0.2451981
3	0.3191732	0.9032587	-0.07237435	2.478024	2.0315718	0.9498847	0.1565729
4	-0.3054719	-0.9870533	-0.99254302	-1.994990	0.1453753	-0.7941729	-0.1892907

	z8	z5	z6	z1	z2
1	0.6757691	0.8070111	1.10369125	-3.151304	-2.438237

2	-0.5053400	-1.9572734	-0.80910941	-2.035625	-1.718086
3	-0.7151477	0.3885813	0.01495838	1.520508	1.216620
4	0.7392971	-0.4492975	0.37008714	-1.440678	-1.177572

With a huge data set, your estimated parameters will be close to the population parameter, i.e. the coefficients you used in your simulation.

If we regress y on its parents (immediate ancestors) we should get something close to the parameters we used to generate y .

```
simdata <- gen(10000)
```

```
head(simdata)
```

	z4	y	z3	m1	z9	x	z7
1	1.1588973	3.562070	0.5194690	2.389926	1.4896115	1.6483907	-1.35719800
2	-0.3263696	1.014703	-1.0974942	0.796184	-0.6920269	0.2160176	1.16785840
3	-0.3646530	-4.022105	-1.2333630	-3.270555	-2.8889790	-1.2007559	-0.18999605
4	1.3368885	7.682182	1.3931109	9.538604	5.8454109	4.5580784	1.28575761
5	0.5337667	4.402521	2.2245167	2.505767	1.7771921	1.1955074	-0.05291829
6	0.2237682	0.182740	0.1655138	2.366024	1.9807087	1.1367916	0.72753107

	z8	z5	z6	z1	z2
1	1.1378368	1.8055321	-0.35901555	5.8020844	4.5728273
2	0.9639717	0.6496645	0.80746882	-1.6625297	-1.2573378

```

3 -1.2659796 -0.3200180  0.23099433 -1.8452179 -1.6393384
4 -0.3374098  1.0976104 -0.55940269  6.6406905  5.3361792
5  1.7134026  1.4053368  0.00874473  2.6686537  2.2462490
6 -1.7177119 -0.8474606 -0.90736316  0.9358874  0.9261273

```

```
dim(simdata)
```

```
[1] 10000    12
```

```

system.time(
  summary(lm(y ~ z8 + z3 + x + m1, data = simdata)) %>% print
)

```

Call:

```
lm(formula = y ~ z8 + z3 + x + m1, data = simdata)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-0.39005	-0.06718	-0.00094	0.06855	0.35360

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.0005340	0.0009954	-0.536	0.592
z8	1.0013042	0.0006970	1436.538	<2e-16 ***
z3	0.3994370	0.0009565	417.584	<2e-16 ***
x	1.0043195	0.0040171	250.012	<2e-16 ***
m1	0.2979849	0.0019941	149.433	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.09952 on 9995 degrees of freedom

Multiple R-squared: 0.9996, Adjusted R-squared: 0.9996

F-statistic: 5.757e+06 on 4 and 9995 DF, p-value: < 2.2e-16

user	system	elapsed
0.082	0.000	0.007

What do you think would happen to the SEs if you were to use a sample size of 1,000,000 instead of 10,000?

What models should give an unbiased estimate of the causal effect of X?

Here are 4 of them. Can you list the others?

```
fit1 <- lm(y ~ x + z1, simdata)
fit2 <- lm(y ~ x + z3, simdata)
fit3 <- lm(y ~ x + z3 + z8, simdata)
fit4 <- lm(y ~ x + z3 + z7, simdata)
```

Here are two that might not. Why?

```
fit5 <- lm(y ~ x, simdata)
fit6 <- lm(y ~ x + z3 + m1, simdata)
```

```
summary(fit1)
```

Call:

```
lm(formula = y ~ x + z1, data = simdata)
```

Residuals:

Min	1Q	Median	3Q	Max
-6.0674	-0.9835	-0.0008	1.0159	5.4581

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.002499	0.014970	-0.167	0.867
x	1.594684	0.014865	107.280	< 2e-16 ***
z1	0.061357	0.007997	7.673	1.84e-14 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.497 on 9997 degrees of freedom

Multiple R-squared: 0.9018, Adjusted R-squared: 0.9018

F-statistic: 4.591e+04 on 2 and 9997 DF, p-value: < 2.2e-16

```
summary(fit2)
```

Call:

```
lm(formula = y ~ x + z3, data = simdata)
```

Residuals:

Min	1Q	Median	3Q	Max
-6.3885	-0.9764	0.0184	0.9732	5.1786

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.006430	0.014443	-0.445	0.656
x	1.589376	0.006677	238.054	<2e-16 ***
z3	0.394257	0.013881	28.402	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.444 on 9997 degrees of freedom

Multiple R-squared: 0.9086, Adjusted R-squared: 0.9086

F-statistic: 4.97e+04 on 2 and 9997 DF, p-value: < 2.2e-16

```
summary(fit3)
```

Call:

```
lm(formula = y ~ x + z3 + z8, data = simdata)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-0.6728	-0.1193	0.0003	0.1199	0.6772

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.0027225	0.0017898	-1.521	0.128
x	1.6006552	0.0008275	1934.411	<2e-16 ***
z3	0.3992140	0.0017201	232.083	<2e-16 ***
z8	1.0033729	0.0012532	800.646	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.179 on 9996 degrees of freedom
Multiple R-squared: 0.9986, Adjusted R-squared: 0.9986
F-statistic: 2.371e+06 on 3 and 9996 DF, p-value: < 2.2e-16

```
summary(fit4)
```

Call:

```
lm(formula = y ~ x + z3 + z7, data = simdata)
```

Residuals:

Min	1Q	Median	3Q	Max
-6.3907	-0.9748	0.0178	0.9730	5.1803

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.006452	0.014444	-0.447	0.655
x	1.588816	0.007481	212.387	<2e-16 ***

```
z3          0.394934    0.014469   27.296    <2e-16 ***
z7          0.002683    0.016157    0.166     0.868
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 1.444 on 9996 degrees of freedom

Multiple R-squared: 0.9086, Adjusted R-squared: 0.9086

F-statistic: 3.313e+04 on 3 and 9996 DF, p-value: < 2.2e-16

```
summary(fit5)
```

Call:

```
lm(formula = y ~ x, data = simdata)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-6.1299	-0.9887	0.0019	1.0078	5.4023

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.003256	0.015013	-0.217	0.828
x	1.700295	0.005629	302.047	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.501 on 9998 degrees of freedom

Multiple R-squared: 0.9012, Adjusted R-squared: 0.9012

F-statistic: 9.123e+04 on 1 and 9998 DF, p-value: < 2.2e-16

`summary(fit6)`

Call:

`lm(formula = y ~ x + z3 + m1, data = simdata)`

Residuals:

Min	1Q	Median	3Q	Max
-----	----	--------	----	-----

-6.2737 -0.9645 0.0142 0.9685 5.0721

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.003815	0.014336	-0.266	0.79
x	0.879207	0.057844	15.199	<2e-16 ***
z3	0.394535	0.013777	28.637	<2e-16 ***
m1	0.354880	0.028715	12.359	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.433 on 9996 degrees of freedom

Multiple R-squared: 0.91, Adjusted R-squared: 0.91

F-statistic: 3.368e+04 on 3 and 9996 DF, p-value: < 2.2e-16

Create a table with estimated values and SEs of $\hat{\beta}_x$

```
library(spida2)
waldf(fit1)
```

	coef	se	U2	L2	p-value
(Intercept)	-0.002499468	0.014970323	0.02744118	-0.03244011	8.674037e-01
x	1.594683775	0.014864756	1.62441329	1.56495426	0.000000e+00
z1	0.061356540	0.007996589	0.07734972	0.04536336	1.839353e-14

	t-value	DF	L.1	L.2	L.3
(Intercept)	-0.1669615	9997	1	0	0
x	107.2795094	9997	0	1	0
z1	7.6728392	9997	0	0	1

The `lapply` function applies its second argument (a function) to each element of the first argument, which should be a **vector** or a **list**. Note that a **data.frame** is a list whose elements are the variables.

```
fitlist <- list(fit1, fit2, fit3, fit4, fit5, fit6)
coeflist <-
  lapply(fitlist,
    function(fit) {
      ret <- wald(fit)['x', c('coef', 'se', 'p-value')]
      ret$rse <- sqrt(mean(resid(fit)^2))
      rownames(ret) <- as.character(formula(fit))[3]
      ret
    }
  )
```

This created a list whose elements are one-row data frames.

```
coeflist
```

```
[[1]]
```

	coef	se	p-value	rse
x + z1	1.594684	0.01486476	0	1.496758

```
[[2]]
```

	coef	se	p-value	rse
x + z3	1.589376	0.006676544	0	1.444027

```
[[3]]
```

	coef	se	p-value	rse
x + z3 + z8	1.600655	0.0008274638	0	0.1789319

```
[[4]]
```

	coef	se	p-value	rse
x + z3 + z7	1.588816	0.007480757	0	1.444025

```
[[5]]
      coef      se p-value      rse
x 1.700295 0.005629242      0 1.501158
```

```
[[6]]
      coef      se      p-value      rse
x + z3 + m1 0.8792073 0.05784449 1.341973e-51 1.433119
```

We'd like to make them into rows of one data frame.

The `rbind` function might do this. “rbind” ‘binds’ its arguments into rows of the same data.frame or matrix.

The `cbind` function does the same thing to columns.

We could do this with

```
rbind(coeflist[[1]], coeflist[[2]], coeflist[[3]], coeflist[[4]]) # etc.
```

```
      coef      se p-value      rse
x + z1 1.594684 0.0148647564      0 1.4967576
x + z3 1.589376 0.0066765441      0 1.4440265
```

x + z3 + z8	1.600655	0.0008274638	0	0.1789319
x + z3 + z7	1.588816	0.0074807568	0	1.4440245

but that wouldn't be efficient if there are many arguments. Also, you wouldn't be able to use such code within a function in which the number of arguments can vary.

The solution is the `do.call` function, which takes two arguments:

- 1) a function
- 2) a list whose elements are used as individual arguments to the function

```
do.call(rbind, coeflist)
```

	coef	se	p-value	rse
x + z1	1.5946838	0.0148647564	0.000000e+00	1.4967576
x + z3	1.5893760	0.0066765441	0.000000e+00	1.4440265
x + z3 + z8	1.6006552	0.0008274638	0.000000e+00	0.1789319
x + z3 + z7	1.5888159	0.0074807568	0.000000e+00	1.4440245
x	1.7002948	0.0056292423	0.000000e+00	1.5011583
x + z3 + m1	0.8792073	0.0578444913	1.341973e-51	1.4331194

Let's fit a few more models that provide an estimate of the causal effect and compare SEs of $\hat{\beta}_x$.

```

fit7 <- lm(y ~ x + z1 + z7, simdata)
fit8 <- lm(y ~ x + z2, simdata)
fit9 <- lm(y ~ x + z1 + z2 + z3, simdata)
fit10 <- lm(y ~ x + z1 + z2 + z3 + z8, simdata)

fitlist2 <- c(fitlist, list(fit7, fit8, fit9, fit10))

summ_table <-
  lapply(fitlist2,
    function(fit) {
      ret <- waldF(fit)['x', c('coef','se','p-value')]
      ret$sy <- sqrt(mean(resid(fit)^2))
      rownames(ret) <- as.character(formula(fit))[3]
      ret
    }
  ) %>%
  do.call(rbind,.)
summ_table

```

	coef	se	p-value	sy
x + z1	1.5946838	0.0148647564	0.000000e+00	1.4967576
x + z3	1.5893760	0.0066765441	0.000000e+00	1.4440265
x + z3 + z8	1.6006552	0.0008274638	0.000000e+00	0.1789319
x + z3 + z7	1.5888159	0.0074807568	0.000000e+00	1.4440245
x	1.7002948	0.0056292423	0.000000e+00	1.5011583
x + z3 + m1	0.8792073	0.0578444913	1.341973e-51	1.4331194
x + z1 + z7	1.5123298	0.1505263401	1.226671e-23	1.4967349
x + z2	1.5946830	0.0148384615	0.000000e+00	1.4967394
x + z1 + z2 + z3	1.5905371	0.0143433200	0.000000e+00	1.4440257
x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	0.000000e+00	0.1789029

Let's keep track of the models that are causal

```
summ_table$causal <- c('yes','no')[c(1,1,1,1,2,2,1,1,1,1)]
# when adding only one variable I don't bother using `within`
summ_table <- sortdf(summ_table, ~ se) # use idempotent programming
# when modifying an object
```

sortdf is in **spida2**. The official way to order a data.frame in R:

```
summ_table <- summ_table[order(summ_table$se),]
```

```
summ_table
```

	coef	se	p-value	sy	causal
x + z3 + z8	1.6006552	0.0008274638	0.000000e+00	0.1789319	yes
x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	0.000000e+00	0.1789029	yes
x	1.7002948	0.0056292423	0.000000e+00	1.5011583	no
x + z3	1.5893760	0.0066765441	0.000000e+00	1.4440265	yes
x + z3 + z7	1.5888159	0.0074807568	0.000000e+00	1.4440245	yes
x + z1 + z2 + z3	1.5905371	0.0143433200	0.000000e+00	1.4440257	yes
x + z2	1.5946830	0.0148384615	0.000000e+00	1.4967394	yes
x + z1	1.5946838	0.0148647564	0.000000e+00	1.4967576	yes
x + z3 + m1	0.8792073	0.0578444913	1.341973e-51	1.4331194	no
x + z1 + z7	1.5123298	0.1505263401	1.226671e-23	1.4967349	yes

```
summ_table$`p-value` <- pfmt(summ_table$`p-value`)
```

```
summ_table
```

coef	se	p-value	sy	causal
------	----	---------	----	--------

x + z3 + z8	1.6006552	0.0008274638	<.00001	0.1789319	yes
x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	<.00001	0.1789029	yes
x	1.7002948	0.0056292423	<.00001	1.5011583	no
x + z3	1.5893760	0.0066765441	<.00001	1.4440265	yes
x + z3 + z7	1.5888159	0.0074807568	<.00001	1.4440245	yes
x + z1 + z2 + z3	1.5905371	0.0143433200	<.00001	1.4440257	yes
x + z2	1.5946830	0.0148384615	<.00001	1.4967394	yes
x + z1	1.5946838	0.0148647564	<.00001	1.4967576	yes
x + z3 + m1	0.8792073	0.0578444913	<.00001	1.4331194	no
x + z1 + z7	1.5123298	0.1505263401	<.00001	1.4967349	yes

```
kable(summ_table, caption = r"(Estimates of  $\hat{\beta}_x$ )")
```

```
library(kableExtra)
```

```
summ_table2 <- cbind(formula = rownames(summ_table), summ_table)
rownames(summ_table2) <- NULL
kable(summ_table2, caption = r"(Estimates of  $\hat{\beta}_x$ )",
      booktabs = TRUE) %>% kable_styling
```

Table 1: Estimates of $\hat{\beta}_x$

	coef	se	p-value	sy	causal
x + z3 + z8	1.6006552	0.0008275	<.00001	0.1789319	yes
x + z1 + z2 + z3 + z8	1.6033071	0.0017772	<.00001	0.1789029	yes
x	1.7002948	0.0056292	<.00001	1.5011583	no
x + z3	1.5893760	0.0066765	<.00001	1.4440265	yes
x + z3 + z7	1.5888159	0.0074808	<.00001	1.4440245	yes
x + z1 + z2 + z3	1.5905371	0.0143433	<.00001	1.4440257	yes
x + z2	1.5946830	0.0148385	<.00001	1.4967394	yes
x + z1	1.5946838	0.0148648	<.00001	1.4967576	yes
x + z3 + m1	0.8792073	0.0578445	<.00001	1.4331194	no
x + z1 + z7	1.5123298	0.1505263	<.00001	1.4967349	yes

Table 2: Estimates of $\hat{\beta}_x$

formula	coef	se	p-value	sy	causal
x + z3 + z8	1.6006552	0.0008275	<.00001	0.1789319	yes
x + z1 + z2 + z3 + z8	1.6033071	0.0017772	<.00001	0.1789029	yes
x	1.7002948	0.0056292	<.00001	1.5011583	no
x + z3	1.5893760	0.0066765	<.00001	1.4440265	yes
x + z3 + z7	1.5888159	0.0074808	<.00001	1.4440245	yes
x + z1 + z2 + z3	1.5905371	0.0143433	<.00001	1.4440257	yes
x + z2	1.5946830	0.0148385	<.00001	1.4967394	yes
x + z1	1.5946838	0.0148648	<.00001	1.4967576	yes
x + z3 + m1	0.8792073	0.0578445	<.00001	1.4331194	no
x + z1 + z7	1.5123298	0.1505263	<.00001	1.4967349	yes

3 Graphic presentation

```
gtab <- summ_table2[rep(1:nrow(summ_table2), each = 3),]  
gtab <- within(  
  gtab,  
  {  
    factor <- c(-1,0,1) # replicates to number of rows in gtab  
    val <- coef + factor * se  
    formula <- as.factor(formula)  
    formula <- reorder(formula, se)  
    pch <- c(40, 16, 41)[factor + 2]  
    col <- dplyr::if_else(causal == 'yes', 'blue', 'red')  
  }  
)  
gtab
```

	formula	coef	se	p-value	sy	causal	col
1	x + z3 + z8	1.6006552	0.0008274638	<.00001	0.1789319	yes	blue

1.1	x + z3 + z8	1.6006552	0.0008274638	<.00001	0.1789319	yes	blue
1.2	x + z3 + z8	1.6006552	0.0008274638	<.00001	0.1789319	yes	blue
2	x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	<.00001	0.1789029	yes	blue
2.1	x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	<.00001	0.1789029	yes	blue
2.2	x + z1 + z2 + z3 + z8	1.6033071	0.0017771802	<.00001	0.1789029	yes	blue
3	x	1.7002948	0.0056292423	<.00001	1.5011583	no	red
3.1	x	1.7002948	0.0056292423	<.00001	1.5011583	no	red
3.2	x	1.7002948	0.0056292423	<.00001	1.5011583	no	red
4	x + z3	1.5893760	0.0066765441	<.00001	1.4440265	yes	blue
4.1	x + z3	1.5893760	0.0066765441	<.00001	1.4440265	yes	blue
4.2	x + z3	1.5893760	0.0066765441	<.00001	1.4440265	yes	blue
5	x + z3 + z7	1.5888159	0.0074807568	<.00001	1.4440245	yes	blue
5.1	x + z3 + z7	1.5888159	0.0074807568	<.00001	1.4440245	yes	blue
5.2	x + z3 + z7	1.5888159	0.0074807568	<.00001	1.4440245	yes	blue
6	x + z1 + z2 + z3	1.5905371	0.0143433200	<.00001	1.4440257	yes	blue
6.1	x + z1 + z2 + z3	1.5905371	0.0143433200	<.00001	1.4440257	yes	blue
6.2	x + z1 + z2 + z3	1.5905371	0.0143433200	<.00001	1.4440257	yes	blue
7	x + z2	1.5946830	0.0148384615	<.00001	1.4967394	yes	blue
7.1	x + z2	1.5946830	0.0148384615	<.00001	1.4967394	yes	blue

7.2	x + z2	1.5946830	0.0148384615	<.00001	1.4967394	yes	blue
8	x + z1	1.5946838	0.0148647564	<.00001	1.4967576	yes	blue
8.1	x + z1	1.5946838	0.0148647564	<.00001	1.4967576	yes	blue
8.2	x + z1	1.5946838	0.0148647564	<.00001	1.4967576	yes	blue
9	x + z3 + m1	0.8792073	0.0578444913	<.00001	1.4331194	no	red
9.1	x + z3 + m1	0.8792073	0.0578444913	<.00001	1.4331194	no	red
9.2	x + z3 + m1	0.8792073	0.0578444913	<.00001	1.4331194	no	red
10	x + z1 + z7	1.5123298	0.1505263401	<.00001	1.4967349	yes	blue
10.1	x + z1 + z7	1.5123298	0.1505263401	<.00001	1.4967349	yes	blue
10.2	x + z1 + z7	1.5123298	0.1505263401	<.00001	1.4967349	yes	blue

	pch	val	factor
1	40	1.5998277	-1
1.1	16	1.6006552	0
1.2	41	1.6014827	1
2	40	1.6015299	-1
2.1	16	1.6033071	0
2.2	41	1.6050843	1
3	40	1.6946655	-1
3.1	16	1.7002948	0

3.2	41	1.7059240	1
4	40	1.5826995	-1
4.1	16	1.5893760	0
4.2	41	1.5960526	1
5	40	1.5813352	-1
5.1	16	1.5888159	0
5.2	41	1.5962967	1
6	40	1.5761938	-1
6.1	16	1.5905371	0
6.2	41	1.6048804	1
7	40	1.5798445	-1
7.1	16	1.5946830	0
7.2	41	1.6095215	1
8	40	1.5798190	-1
8.1	16	1.5946838	0
8.2	41	1.6095485	1
9	40	0.8213628	-1
9.1	16	0.8792073	0
9.2	41	0.9370518	1

10	40	1.3618035	-1
10.1	16	1.5123298	0
10.2	41	1.6628561	1

```
library(lattice)
library(latticeExtra)
library(latex2exp)
```

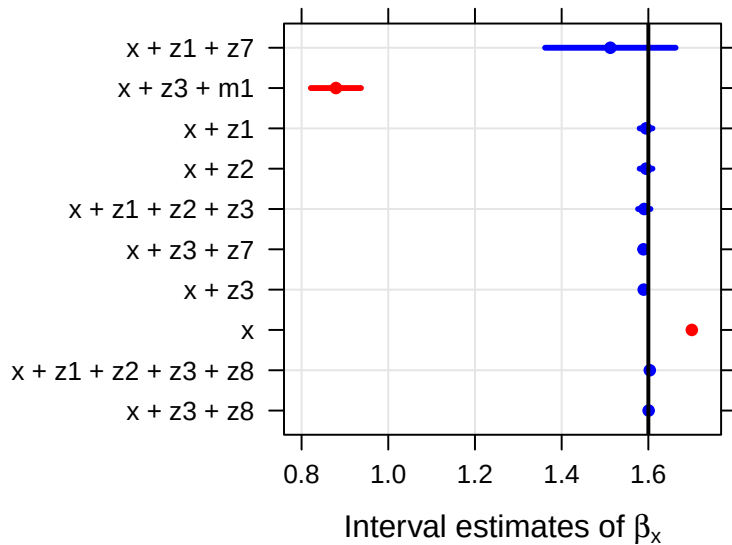



Figure 5: $\hat{\beta}_x \pm SE(\hat{\beta}_x)$ with $\beta_x = 1.6$

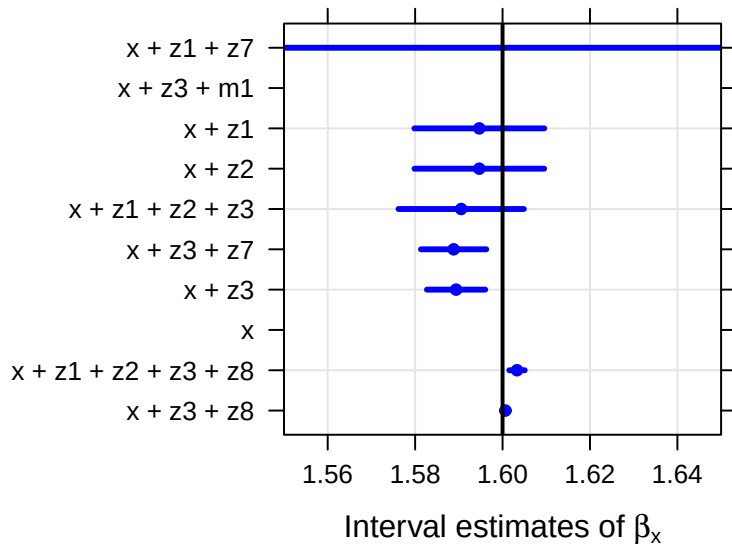


Figure 6: $\hat{\beta}_x \pm SE(\hat{\beta}_x)$ with $\beta_x = 1.6$, focusing on 1.6

- 2) Think of an example in everyday life with at least one backdoor path and at least one causal path in addition to the direct path from X to Y. Generate plausible data. Note that the means of your variables don't have to be zero, nor the standard deviations one. Play with different models, classify whether they are causal or not, and explore the differences in estimates and standard errors.
- 3) If you are familiar with **ggplot2**, use it to reproduce graphics like those in this scripts.

```
knitr::knit_exit()
```