See chapter 8 in Regression and Other Stories.

Widen the notebook.

```
html"""
<style>
    main {
        margin: 0 auto;
        max-width: 2000px;
        padding-left: max(160px, 10%);
        padding-right: max(160px, 10%);
    }
</style>
"""
```

```
using Pkg ✓ , DrWatson ✓
```

A typical set of Julia packages to include in notebooks.

```
begin
    # Specific to this notebook
    using GLM ✓

    # Specific to ROSTuringPluto
    using Optim ✓
    using Logging ✓
    using Turing ✓

    # Graphics related]
    using GLMakie ✓

    # Common data files and functions
    using RegressionAndOtherStories ✓
    import RegressionAndOtherStories: link

    Logging.disable_logging(Logging.Warn)
end;
```

8.1 Least squares, maximum likelihood, and Bayesian inference.

	x	y	ϵ	error
1	0.0	45.9177	-0.282333	-0.282333
2	0.0251256	48.4013	2.12591	2.12591
3	0.0502513	43.1233	-3.22741	-3.22741
4	0.0753769	56.2541	9.82797	9.82797
5	0.100503	51.161	4.6595	4.6595
6	0.125628	47.6471	1.07026	1.07026
7	0.150754	53.652	6.99973	6.99973
8	0.175879	43.4236	-3.30408	-3.30408
9	0.201005	42.632	-4.17101	-4.17101
10	0.226131	45.5619	-1.31654	-1.31654
⋮ more				
200	5.0	57.0887	-4.11126	-4.11126

```
let
    Random.seed!(1)
    a = 46.2
    b = 3.0
    sigma = 4.0
    x = LinRange(0, 5, 200)
    ϵ = rand(Normal(0, sigma), length(x))
    y = a .+ b .* x .+ ϵ

    # DataFrame used to collect estimates,
    shown later on.

    global estimate_comparison = DataFrame()
    estimate_comparison.parameters = [:a,
    :b, :sigma]

    global sim = DataFrame(x = x, y = y, ϵ
    = ϵ, error = y .- (a .+ b .* x))
end
```

ppl8_1 (generic function with 2 methods)

```
@model function ppl8_1(x, y)
    a ~ Normal(1, 5)
    b ~ Normal(1, 5)
    σ ~ Exponential(1)
    μ = a .+ b .* x
    for i in eachindex(y)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

	parameters	mean	std	naive_se
1	:a	45.6072	0.609098	0.0096306
2	:b	3.25363	0.215495	0.0034072
3	:σ	4.36749	0.221494	0.0035021

```
begin
    m8_1t = ppl8_1(sim.x, sim.y)
    chns8_1t = sample(m8_1t, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns8_1t)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	45.612	0.618	45.607	0.60
2	"b"	3.257	0.219	3.254	0.21
3	"σ"	4.359	0.22	4.367	0.22

```
begin
    post8_1t = DataFrame(chns8_1t)[:, 3:5]
    ms8_1t = model_summary(post8_1t, Symbol.
    (names(post8_1t)))
end
```

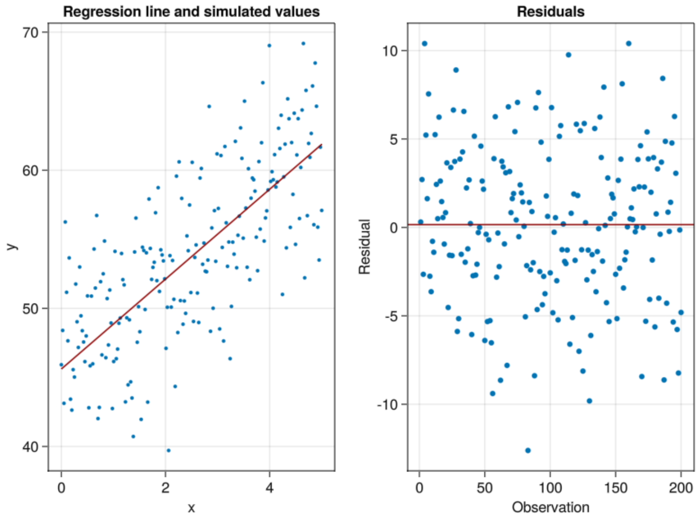
```
begin
    estimate_comparison[!, :Turing] =
    [[ms8_1t[p, :mean], ms8_1t[p, :std]] for
     p in [:a, :b, :σ]]
end;
```

	parameters	Turing
1	:a	[45.607, 0.609]
2	:b	[3.254, 0.215]
3	:sigma	[4.367, 0.221]

```
estimate_comparison
```

	x	y	ϵ	error
1	0.0	45.9177	-0.282333	-0.282333
2	0.0251256	48.4013	2.12591	2.12591
3	0.0502513	43.1233	-3.22741	-3.22741
4	0.0753769	56.2541	9.82797	9.82797
5	0.100503	51.161	4.6595	4.6595
6	0.125628	47.6471	1.07026	1.07026
7	0.150754	53.652	6.99973	6.99973
8	0.175879	43.4236	-3.30408	-3.30408
9	0.201005	42.632	-4.17101	-4.17101
10	0.226131	45.5619	-1.31654	-1.31654
⋮ more				
200	5.0	57.0887	-4.11126	-4.11126

```
let
    â = ms8_1t[:a, :median]
    b̂ = ms8_1t[:b, :median]
    sim.residual = sim.y .- (â .+ b̂ .*
    sim.x)
    sim
end
```

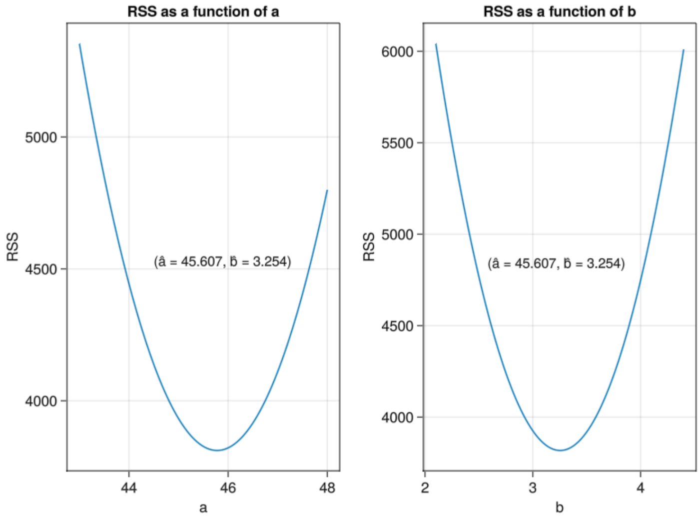



```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="Regression line and simulated values", xlabel="x", ylabel="y")
    x_range = LinRange(minimum(sim.x), maximum(sim.x), 200)
    y_res = mean.(link(post8_1t, (r,x) -> r.a + x * r.b, x_range))
    scatter!(sim.x, sim.y; markersize=4)
    lines!(x_range, y_res; color=:darkred)

    ax = Axis(f[1, 2]; title="Residuals", xlabel="Observation", ylabel="Residual")
    scatter!(sim.residual; markersize=6)
    hlines!(ax, mean(sim.residual); color=:darkred)
    f
end
```

RSS = 3817.0638864618904

```
RSS = sum(sim.residual .^ 2)
```




```
let
    â = ms8_1t[:a, :mean]
    b̂ = ms8_1t[:b, :mean]

    f = Figure()
    ax = Axis(f[1, 1]; title="RSS as a
    function of a", xlabel="a",
    ylabel="RSS")
    a_range = LinRange(43, 48, 100)
    r = [sum((sim.y .- (k .+ b̂ .* sim.x))
    .^ 2) for k in a_range]
    lines!(a_range, r)
    annotations!("$((â = â, b̂ = b̂))",
    position=(44.5, 4500), textsize=15)
    ax = Axis(f[1, 2]; title="RSS as a
    function of b", xlabel="b",
    ylabel="RSS")
    b_range = LinRange(2.1, 4.4, 100)
    r = [sum((sim.y .- (â .+ k .* sim.x))
    .^ 2) for k in b_range]
    lines!(b_range, r)
    annotations!("$((â = â, b̂ = b̂))",
    position=(2.58, 4800), textsize=15)
    f
end
```

Least squares

```
▸ (46.2831, 3.05172)
```

```
let
    global lsq = [0.0 missing; 0.0 missing;
    0.0 missing]
    df = DataFrame(ones = ones(nrow(sim)), x
     = sim.x)
    X = Array(df)
    Xt = transpose(X)
    â, b̂ = (Xt * X)^-1 * Xt * sim.y
    lsq[1, 1] = â
    lsq[2, 1] = b̂
    â, b̂
end
```

```
▸ (â = 46.2831, b̂ = 3.05172)
```

```
let
    b̂ = sum((sim.x .- mean(sim.x)) .*
    sim.y) / sum(((sim.x .- mean(sim.x)) .^
    2))
    â = mean(sim.y) - b̂ * mean(sim.x)
    (â = â, b̂ = b̂)
end
```

```
4.390683367845627
```

```
let
    σ̂ = sqrt(sum(sim.residual .^
    2)/(nrow(sim) - 2))
    lsq[3, 1] = σ̂
    estimate_comparison[!, :least_squares]
    = [Vector(i) for i in eachrow(lsq)]
    σ̂
end
```

Maximum likelihood

```
loglik (generic function with 1 method)
```

```
function loglik(x)
    ll = 0.0
    ll += log(pdf(Normal(50, 20), x[1]))
    ll += log(pdf(Normal(2, 10), x[2]))
    ll += log(pdf(Exponential(1), x[3]))
    for i in 1:nrow(sim)
        ll += sum(logpdf.(Normal(x[1] .+
        x[2] .* sim.x[i], x[3]), sim.y[i]))
    end
    -ll
end
```

```
0.1353352832366127
```

```julia
pdf(Exponential(1), 2.0)
```

```
[170.0, 10.0, 2.0]
```

```julia
begin
    lower = [0.0, 0.0, 0.0]
    upper = [250.0, 50.0, 10.0]
    x0 = [170.0, 10.0, 2.0]
end
```

```
res =
 * Status: success (objective increased between :

 * Candidate solution
    Final objective value:     5.895739e+02

 * Found with
    Algorithm:     Fminbox with L-BFGS

 * Convergence measures
    |x - x'|               = 2.27e-08 ≰ 0.0e+00
    |x - x'|/|x'|          = 4.88e-10 ≰ 0.0e+00
    |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
    |g(x)|                 = 6.16e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   1  (vs limit Inf)
    Iterations:    5
    f(x) calls:    120
    ∇f(x) calls:   120
```

```julia
res = optimize(loglik, lower, upper, x0)
```

```
[46.2877, 3.05023, 4.30947]
```

```julia
let
    mle = Optim.minimizer(res)
    lsq[:, 1] = mle
    estimate_comparison[!, :mle] =
    [Vector(i) for i in eachrow(lsq)]
    mle
end
```

MLE and MAP estimates.

```
mle_estimate =
ModeResult with maximized lp of -578.08
3-element Named Vector{Float64}
A   │
────┼──────────
:a  │  46.2831
:b  │  3.05172
:σ  │  4.35565
```

```julia
mle_estimate = optimize(m8_1t, MLE())
```

```
▸ [46.2831, 3.05172, 4.35565]
```

```julia
Vector(coef(mle_estimate))
```

```
map_estimate =
ModeResult with maximized lp of -627.99
3-element Named Vector{Float64}
A   │
────┼──────────
:a  │  45.6313
:b  │  3.24578
:σ  │  4.32148
```

```julia
map_estimate = optimize(m8_1t, MAP())
```

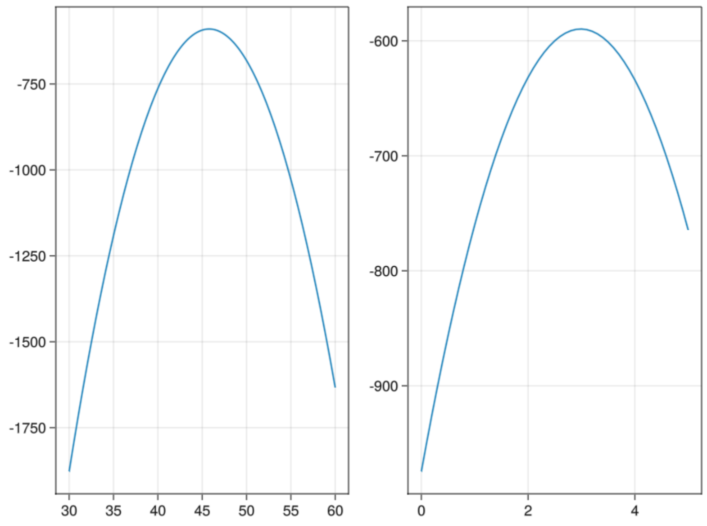
Compare the four results.

	parameters	Turing	least_squa
1	:a	▸ [45.607, 0.609]	▸ [46.2831, m
2	:b	▸ [3.254, 0.215]	▸ [3.05172, m
3	:sigma	▸ [4.367, 0.221]	▸ [4.39068, m

```julia
let
    estimate_comparison[!, :mle_turing] =
    Vector(coef(mle_estimate))
    estimate_comparison[!, :map_turing] =
    Vector(coef(map_estimate))
    estimate_comparison
end
```

```
590.2799283100538
```

```julia
loglik([45.6, 3.25, 4.4])
```




```julia
let
    f = Figure()
    ax = Axis(f[1, 1])
    lines!(30:0.1:60, [-loglik([a, 3.25,
    4.4]) for a in 30:0.1:60])
    ax = Axis(f[1, 2])
    lines!(0:0.1:5, [-loglik([46.5, b,
    4.4]) for b in 0:0.1:5])
    f
end
```

```
600.0086334504888
```

```julia
loglik([45, 3, 4.4])
```

```
2×200 Matrix{Float64}:
  1.0       1.0201    1.0402    1.0603    1.0804   …
 46.6171   43.4073   42.8262   48.5102   51.4102
```

```julia
let
    using StatsAPI ✓
    Random.seed!(123)
    a = 46.2
    b = 3.0
    sigma = 4.0
    x = LinRange(1, 5, 200)
    ϵ = rand(Normal(0, sigma), length(x))
    y = a .+ b .* x .+ ϵ
    global obs = Matrix(hcat(x, y)')
end
```

```
distr8_1 =
FullNormal(
dim: 2
μ: [2.9999999999999996, 55.07524018977074]
Σ: [1.3467336683417086 3.9331584800491735; 3.933:
)
```

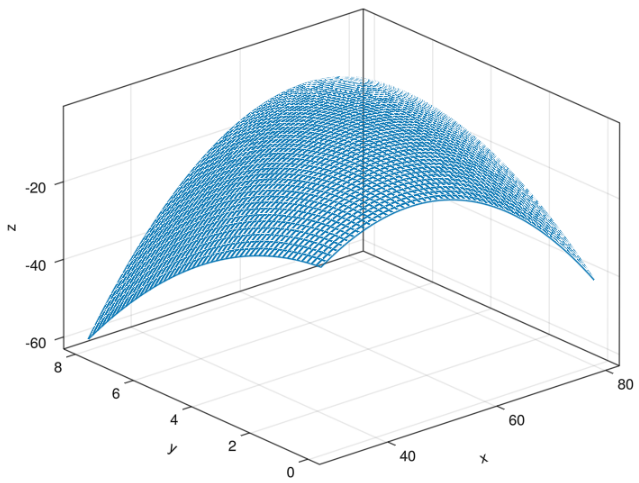
```
distr8_1 = fit_mle(MvNormal, obs)
```

```
2×1 Matrix{Float64}:
  2.9611268137526
 55.05684781136959
```

```
mean(rand(distr8_1, 1000); dims=2)
```

```
-3.389758334022121
```

```
loglikelihood(distr8_1, [3, 55])
```



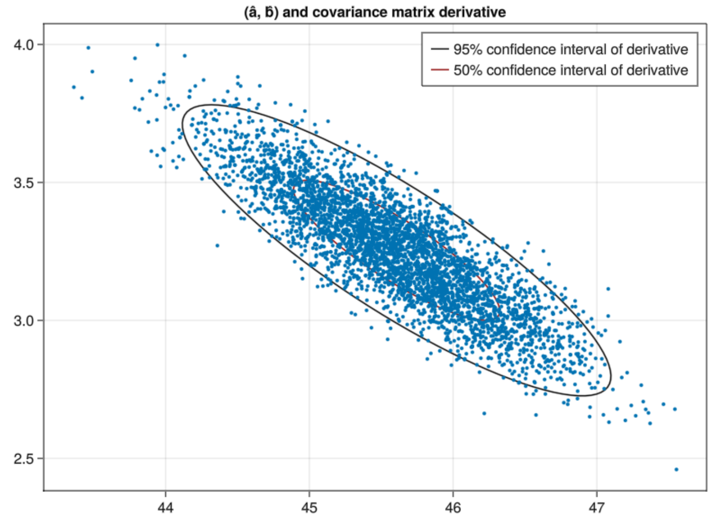

```
let
    a = collect(LinRange(30, 80, 50))
    b = collect(LinRange(0, 8, 50))
    global z = [loglikelihood(distr8_1, [b,
    a]) for a in a, b in b]
    m, i = findmax(z)
    maxz = [a[i[1]], b[i[1]], z[i]]
    println(maxz)
    wireframe(a, b, z, axis=(type=Axis3,))
end
```

```
my_μ = ▸[45.607, 3.254]
```

```
my_μ = [ms8_1t["a", "mean"], ms8_1t["b",
"mean"]]
```

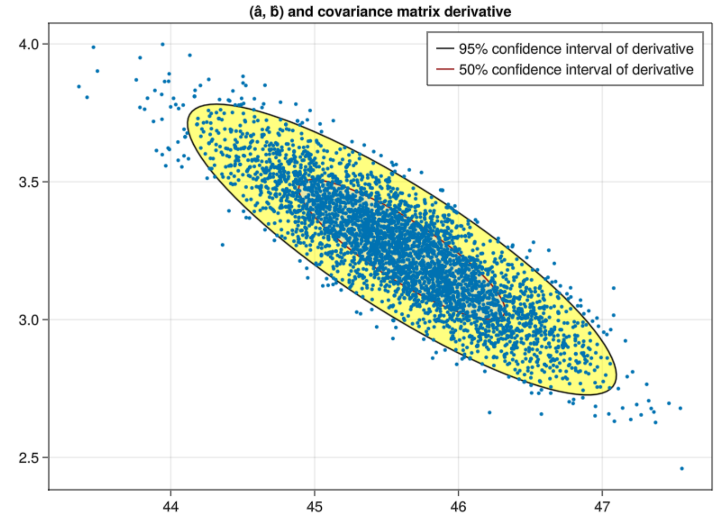
```
my_Σ = 2×2 Matrix{Float64}:
  0.371001  -0.113603
 -0.113603   0.046438
```

```julia
my_Σ = cov([post8_1t.a post8_1t.b])
```



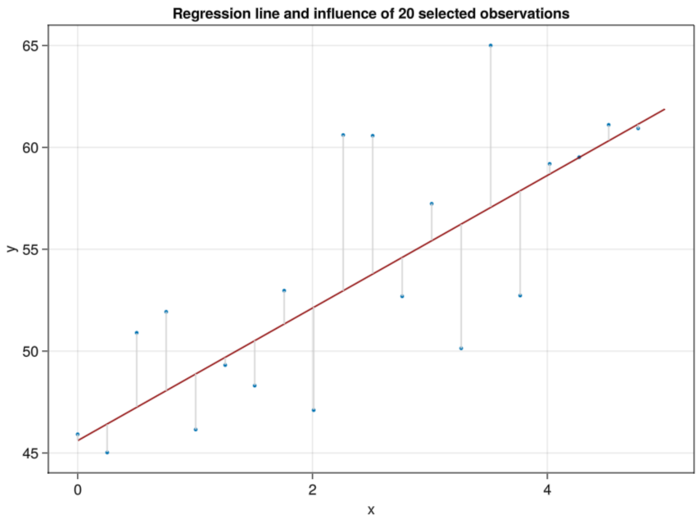

```julia
let
    f = Figure()
    ax = Axis(f[1, 1]; title="(â, b̂) and covariance matrix derivative")
    lines!(getellipsepoints(my_μ, my_Σ)..., label="95% confidence interval of derivative", color=:black)
    lines!(getellipsepoints(my_μ, my_Σ, 0.5)..., label="50% confidence interval of derivative", color=:darkred)
    scatter!(post8_1t.a, post8_1t.b; markersize=4)
    axislegend(position=:rt)
    f
end
```

```
Enter cell code...
```




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="(â, b̂) and
    covariance matrix derivative")
    poly!(Point2f.
    (zip(getellipsepoints(my_μ, my_Σ)...));
    color=(:yellow, 0.5))
    poly!(Point2f.
    (zip(getellipsepoints(my_μ, my_Σ,
    0.50)...)); color=(:lightgrey, 0.5))
    lines!(getellipsepoints(my_μ, my_Σ)...,
    label="95% confidence interval of
    derivative", color=:black)
    lines!(getellipsepoints(my_μ, my_Σ,
    0.5)..., label="50% confidence interval
    of derivative", color=:darkred)
    scatter!(post8_1t.a, post8_1t.b;
    markersize=4)
    axislegend(position=:rt)
    f
end
```

8.2 Influence of individual points in a fitted regression.




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="Regression
    line and influence of 20 selected
    observations", xlabel="x", ylabel="y")
    x_range = LinRange(minimum(sim.x),
    maximum(sim.x), 200)
    y_res = mean.(link(post8_1t, (r,x) ->
    r.a + x * r.b, x_range))
    select_obs = 1:10:200
    scatter!(sim.x[select_obs],
    sim.y[select_obs]; markersize=4)
    lines!(x_range, y_res; color=:darkred)
    for ind in select_obs
        ymin = min(sim.y[ind], y_res[ind])
        ymax = max(sim.y[ind], y_res[ind])
        lines!([sim.x[ind], sim.x[ind]],
        [ymin, ymax]; color=:lightgrey)
    end
    f
end
```

8.3 Least squares slope as a weighted average of slopes of pairs.

▸ (weighted_slopes = 3.05172, least_squares = [3.

```
let
    s1 = sum([(sim.x[i]-sim.x[j]) *
    (sim.y[i]-sim.y[j]) for i in
    1:length(sim.x), j in 1:length(sim.y)])
    s2 = sum([(sim.x[i]-sim.x[j])^2 for i in
     1:length(sim.x), j in 1:length(sim.y)])
    (weighted_slopes = round(s1/s2;
    digits=5),
    least_squares=estimate_comparison[2,
    :least_squares])
end
```

8.4 Comparing two fitting functions: `glm` and `stan_sample`.

ppl8_2 (generic function with 2 methods)

```
@model function ppl8_2(x, y)
    a ~ Normal(0, 50)
    b ~ Normal(0, 50)
    σ ~ Exponential(1)
    μ = a .+ b .* x
    for i in eachindex(y)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

	parameters	mean	std	naive_se
1	:a	-13.6389	5.15404	0.081492
2	:b	5.08013	0.842866	0.013326
3	:σ	7.26264	1.29555	0.020484

```julia
let
    x = LinRange(1, 10, 10)
    y = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
    global fake = DataFrame(x = x, y = y)
    global m8_2t = ppl8_2(fake.x, fake.y)
    global chns8_2t = sample(m8_2t, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns8_2t)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	-13.596	4.941	-13.639	5.15
2	"b"	5.081	0.797	5.08	0.84
3	"σ"	7.105	1.182	7.263	1.29

```julia
begin
    post8_2t = DataFrame(chns8_2t)[:, 3:5]
    ms8_2t = model_summary(post8_2t, [:a,
    :b, :σ])
end
```

[3.42045, 6.75029]

```julia
quantile(post8_2t.b, [0.025, 0.975])
```

[3.70936, 6.42466]

```julia
quantile(post8_2t.b, [0.05, 0.95])
```

```
fake_lm =
StatsModels.TableRegressionModel{LinearModel{GLM.

y ~ 1 + x

Coefficients:
──────────────────────────────────────────────────
                Coef.  Std. Error      t  Pr(>|
──────────────────────────────────────────────────
(Intercept)  -13.8667     6.32766  -2.19    0.0
x              5.12121    1.01979   5.02    0.0
──────────────────────────────────────────────────
```

```
fake_lm = lm(@formula(y ~ x), fake)
```