Chap 11 God Spiked the Integers

```
md"# Chap 11 God Spiked the Integers"
```

```
#    margin-left: 1%;
#    margin-right: 5%;
html"""<style>
main {
    margin: 0 auto;
    max-width: 90%;
    padding-left: max(50px, 1%);
    padding-right: max(253px, 10%);
    # 253px to accomodate TableOfContents(aside=true)
}
"""

```

```
using Pkg, DrWatson
```

Table of Contents

11.3 Multinomial and categorical models


```
begin
  using PlutoUI
  TableOfContents()
end
```

```
begin
    using Optim
    using Turing
    using DataFrames
    using CSV
    using Random
    using StatisticalRethinking
    using StatisticalRethinking: link
    using StatisticalRethinkingPlots
    using ParetoSmooth
    using StatsPlots
    using StatsBase
    using FreqTables
    using Logging
end
```

Error requiring `Turing` from `StatisticalRethinking`
exception:

LoadError: UndefVarError: `ModeResult` not defined

in expression starting at /y/home/huangyu/.julia/packages/StatisticalRethinking/RYYWV/src/require/turing/turing_optim_sample.jl:5

in expression starting at /y/home/huangyu/.julia/packages/StatisticalRethinking/RYYWV/src/require/turing/turing.jl:7

Stack trace

Here is what happened, the most recent locations are first:

1. *turing_optim_sample.jl:5*
2. **include**(mod::Module, _path::String)
   @ *Base.jl:495*
3. **include**(x::String)
   @ *StatisticalRethinking.jl:1*
4. *turing.jl:7*
5. **include**(mod::Module, _path::String)
   @ *Base.jl:495*
6. **include**(x::String)
   @ *StatisticalRethinking.jl:1*
7. *Requires.jl:40*
8. eval @ boot.jl:385

```
begin
    Plots.default(label=false)
    #Logging.disable_logging(Logging.Warn);
end;
```

11.1 Binomial regression.

Code 11.1: Load chimpanzees prosocial data

```
md" ## Code 11.1: Load chimpanzees prosocial data"
```

```
chimpanzees = CSV.read(sr_datadir("chimpanzees.csv"),
DataFrame; delim=';');
```

```
(504, 11)
```

```
size(chimpanzees)
```

	variable	mean	min	median	max	nmissing	e
1	:actor	4.0	1	4.0	7	0	Int
2	:recipient	nothing	"2"	nothing	"NA"	0	Str
3	:condition	0.5	0	0.5	1	0	Int
4	:block	3.5	1	3.5	6	0	Int
5	:trial	36.375	1	36.0	72	0	Int
6	:prosoc_left	0.5	0	0.5	1	0	Int
7	:chose_prosoc	0.56746	0	1.0	1	0	Int
8	:pulled_left	0.579365	0	1.0	1	0	Int
9	:treatment	2.5	1	2.5	4	0	Int
10	:side	1.5	1	1.5	2	0	Int
11	:cond	1.5	1	1.5	2	0	Int

```
describe(chimpanzees)
```

Code 11.2: `treatment` encoding: based on prosoc_left and condition of partner presence

```
md" ## Code 11.2: `treatment` encoding: based on
prosoc_left and condition of partner presence"
```

```
chimpanzees[!,:treatment] = 1 .+ chimpanzees.prosoc_left .+
2*chimpanzees.condition;
```

Code 11.3: contrast table

```
md" ## Code 11.3: contrast table"
```

```
2×2 Named Matrix{Int64}
treatment \ prosoc_left │   0    1
────────────────────────┼──────────
1                       │ 126    0
2                       │   0  126
```

```
freqtable(chimpanzees, :treatment, :prosoc_left,
subset=chimpanzees.condition .== 0)
```

```
2×2 Named Matrix{Int64}
treatment \ prosoc_left │   0    1
────────────────────────┼──────────
3                       │ 126    0
4                       │   0  126
```

```
freqtable(chimpanzees, :treatment, :prosoc_left,
subset=chimpanzees.condition .== 1)
```

Code 11.4: ppl11_1, try a Normal(0,1) prior

```
md" ## Code 11.4: ppl11_1, try a Normal(0,1) prior"
```

```
ppl11_1 (generic function with 2 methods)
```

```
@model function ppl11_1(pulled_left)
    a ~ Normal(0, 10)
    p = logistic(a)     # inverse of the `logit` function
    pulled_left ~ Binomial(1, p)
end
```

Code 11.5: Sample data from ppl11_1 prior.

```
md" ### Code 11.5: Sample data from ppl11_1 prior."
```

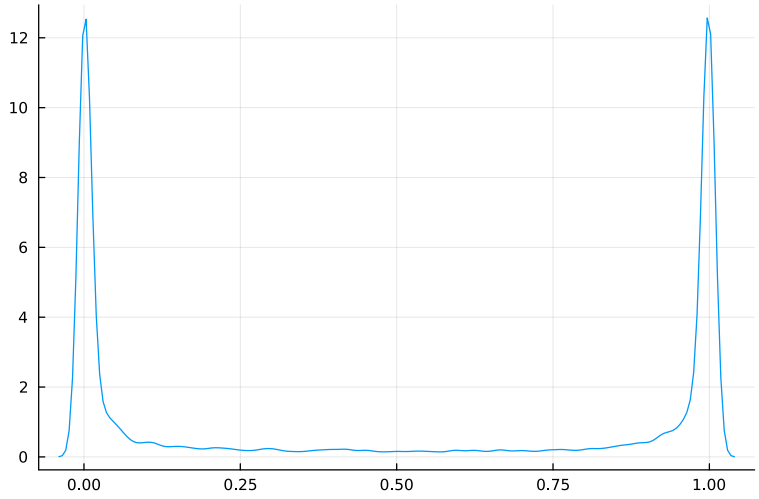
	a
1	14.85
2	13.422
3	-12.1449
4	-13.1042
5	8.01591
6	3.17127
7	-12.5028
8	1.78021
9	7.70168
10	8.77692
	more
10000	-5.07009

```
begin
    Random.seed!(1999)
    prior11_1 = sample(ppl11_1(chimpanzees.pulled_left),
    Prior(), 10000)
    prior11_1_df = DataFrame(prior11_1);
end
```

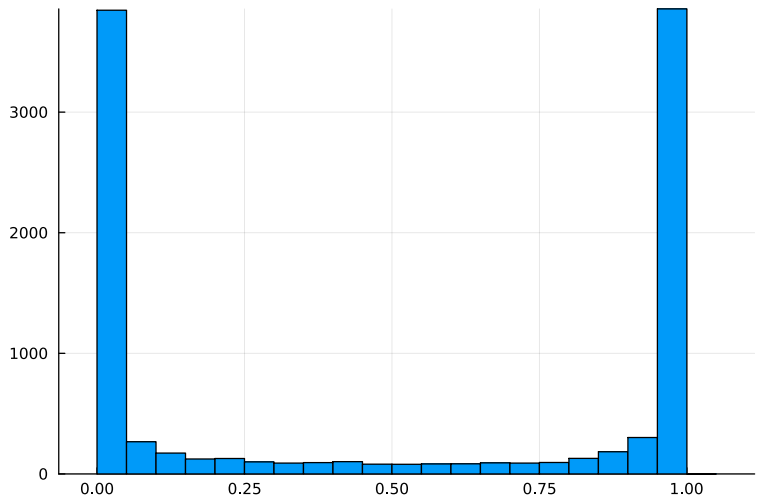
100%

Code 11.6: Histogram of p's prior from `ppl11_1`

```
md" ### Code 11.6: Histogram of p's prior from `ppl11_1`"
```




```
let
    p = logistic.(prior11_1_df.a)
    density(p, bandwidth=0.01)
end
```




```
let
        p = logistic.(prior11_1_df.a)
        histogram(p, bins=30)
end
```

Code 11.7: ppl11_2, Normal(0, 1.5) for a, Normal(0, 10) for b and sample from this prior

```
md" ## Code 11.7: ppl11_2, Normal(0, 1.5) for a, Normal(0, 10) for b and sample from this prior"
```

```
ppl11_2 (generic function with 2 methods)
```

```
@model function ppl11_2(pulled_left, treatment)
    a ~ Normal(0, 1.5)
    treat_levels = length(levels(treatment))
    b ~ MvNormal(zeros(treat_levels), 10)

    p = @. logistic(a + b[treatment])
    for i ∈ eachindex(pulled_left)
        pulled_left[i] ~ Binomial(1, p[i])
    end
end
```

	variable	mean	min	median	max
1	:a	0.00462767	-6.21253	0.0118894	5.656
2	Symbol("b[1]")	-0.126626	-36.2479	-0.189042	38.2774
3	Symbol("b[2]")	-0.235768	-38.3193	-0.206774	40.103
4	Symbol("b[3]")	-0.0800449	-44.1269	-0.0231773	37.6808
5	Symbol("b[4]")	0.00810773	-39.0784	-0.0298612	37.9247

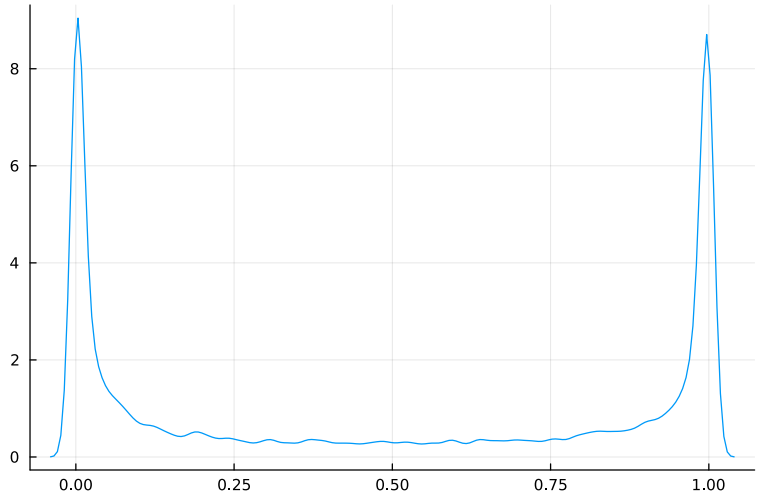
```
begin
    Random.seed!(1999)
    prior_chain11_2t =
    sample(ppl11_2(chimpanzees.pulled_left,
    chimpanzees.treatment), Prior(), 10000)
    prior11_2t_df = DataFrame(prior_chain11_2t)
    describe(prior11_2t_df)
end
```

```
100%
```

Code 11.8: Histogram of Δp between the first two treatments implied by the prior.

- Bad!

```
md" ### Code 11.8: Histogram of Δp between the first two
treatments implied by the prior.
- Bad!"
```




```
let
    f = i -> @. logistic(prior11_2t_df.a +
    prior11_2t_df[!,"b[$i]"])
    p1 = map(f, 1:4)
    density(abs.(p1[1] .- p1[2]), bandwidth=0.01)
end
```

Code 11.9: `m11_3`, Normal(0, 1.5) for a, Normal(0, 0.5) for b.

```
md" ## Code 11.9: `m11_3`, Normal(0, 1.5) for a, Normal(0,
0.5) for b."
```

```
m11_3 (generic function with 2 methods)
```

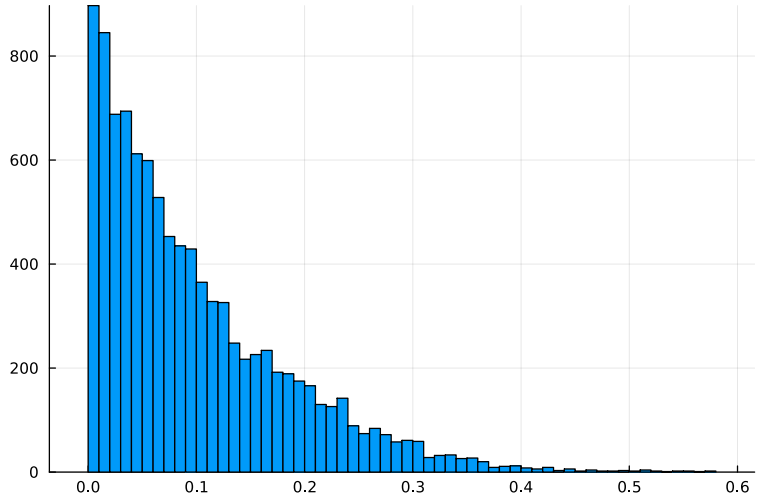
```
@model function m11_3(pulled_left, treatment)
    a ~ Normal(0, 1.5)
    treat_levels = length(levels(treatment))
    b ~ MvNormal(zeros(treat_levels), 0.5)

    p = @. logistic(a + b[treatment])
    for i ∈ eachindex(pulled_left)
        pulled_left[i] ~ Binomial(1, p[i])
    end
end
```

	a	b[1]	b[2]	b[3]	b[4]
1	2.22751	0.671101	-0.607244	-0.655211	0.400796
2	0.47569	-0.625141	0.0890104	0.385084	0.438846
3	0.560083	0.458511	-0.717306	0.0886391	0.477489
4	0.428818	-0.0285005	0.0657721	0.360067	-0.66726
5	-2.40598	0.159513	-0.366338	-0.452723	-0.41840
6	1.8485	-0.423045	1.28846	0.656191	0.720912
7	-0.761127	0.301917	-0.0420804	-0.943026	-0.20489
8	1.04003	-0.212795	-0.138058	0.832358	0.070654
9	2.91907	-0.0348572	-0.598412	0.278982	-0.34629
10	-0.0636526	0.173388	-0.448934	-0.41949	0.15899
more					
10000	-0.599323	-0.210639	1.41076	0.389495	0.364247

```
begin
    Random.seed!(1999)
    prior_chain3 = sample(m11_3(chimpanzees.pulled_left,
    chimpanzees.treatment), Prior(), 10000)
    prior3 = DataFrame(prior_chain3);
end
```

100%




```
let
    f = i -> @. logistic(prior3.a + prior3[!,"b[$i]"])
    p2 = map(f, 1:4)
    @show mean(abs.(p2[1] .- p2[2]))
    histogram(abs.(p2[1] .- p2[2]))
end
```

```
mean(abs.(p2[1] .- p2[2])) = 0.09781807904443517
```

Now the prior-implied Δp is much smaller

```
md"Now the prior-implied Δp is much smaller"
```

Code 11.10: Trim data columns

```
md" ## Code 11.10: Trim data columns"
```

```
dat_list = chimpanzees[!,[:pulled_left, :actor,
:treatment]];
```

Code 11.11: `m11_4`, pull_left ~ actor + treatment with sensible prior.

```
md" ## Code 11.11: `m11_4`, pull_left ~ actor + treatment
with sensible prior."
```

```
m11_4 (generic function with 2 methods)
```

```
@model function m11_4(actor, treatment, pulled_left)
    act_levels = length(levels(actor))
    a ~ MvNormal(zeros(act_levels), 1.5)
    treat_levels = length(levels(treatment))
    b ~ MvNormal(zeros(treat_levels), 0.5)

    p = @. logistic(a[actor] + b[treatment])
    for i ∈ eachindex(pulled_left)
        pulled_left[i] ~ Binomial(1, p[i])
    end
end
```

	variable	mean	min	median	max
1	Symbol("a[1]")	-0.438127	-1.64746	-0.436987	0.627686
2	Symbol("a[2]")	3.91721	1.83393	3.85155	6.65498
3	Symbol("a[3]")	-0.734151	-1.70495	-0.731365	0.221815
4	Symbol("a[4]")	-0.738928	-1.99161	-0.73496	0.368401
5	Symbol("a[5]")	-0.445311	-1.73442	-0.462772	0.678659
6	Symbol("a[6]")	0.485383	-0.490516	0.498496	1.495
7	Symbol("a[7]")	1.96677	0.502998	1.96425	3.27718
8	Symbol("b[1]")	-0.0495266	-1.08009	-0.0493664	0.863124
9	Symbol("b[2]")	0.47593	-0.414017	0.479686	1.38662
10	Symbol("b[3]")	-0.388545	-1.32583	-0.378317	0.523468
11	Symbol("b[4]")	0.362129	-0.590455	0.363076	1.17797

```
begin
    @time m11_4_chain = sample(m11_4(dat_list.actor,
    dat_list.treatment, dat_list.pulled_left), NUTS(), 1000)
    m11_4_df = DataFrame(m11_4_chain)
    describe(m11_4_df)
end
```

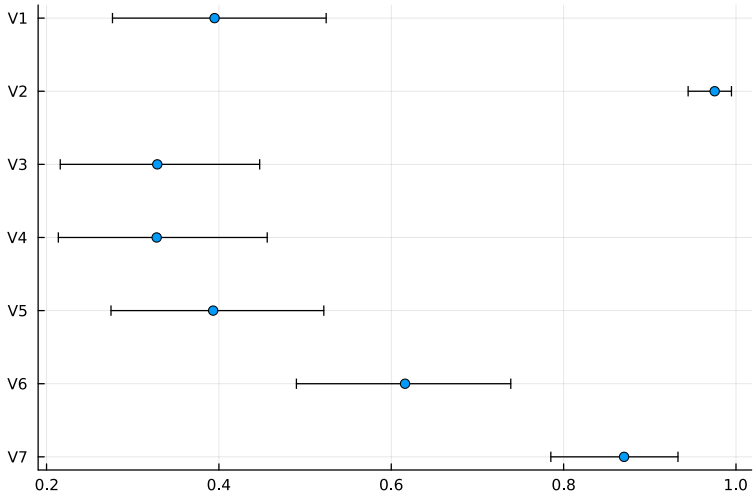
```
100%
```

```
Found initial step size
ϵ: 0.4
```

```
  4.035758 seconds (1.79 M allocations: 2.154 GiB,
4.76% gc time)
```

Code 11.12: Compare prob of pull_left among 7 actors/chimps

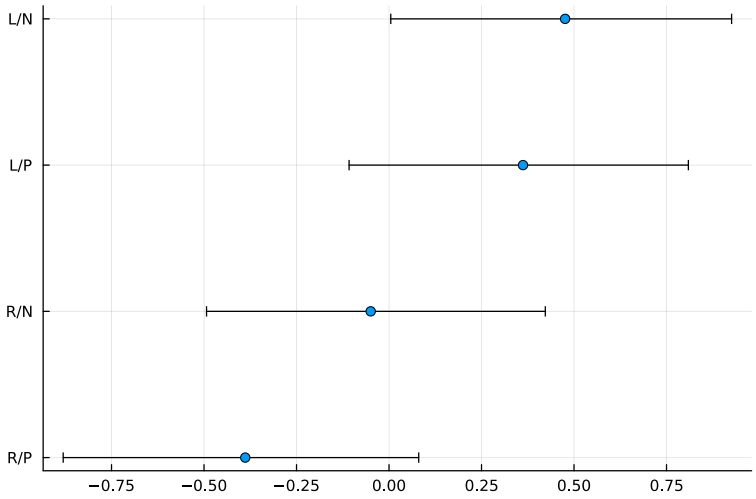
```
md" ### Code 11.12: Compare prob of pull_left among 7
actors/chimps"
```




```
let
    p_left = DataFrame(map(i -> "V$i" => logistic.
    (m11_4_df[:,"a[$i]"]), 1:7)...);
    coeftab_plot(p_left)
end
```

Code 11.13: Compare β among 4 treatments.

```
md" ### Code 11.13: Compare β among 4 treatments."
```

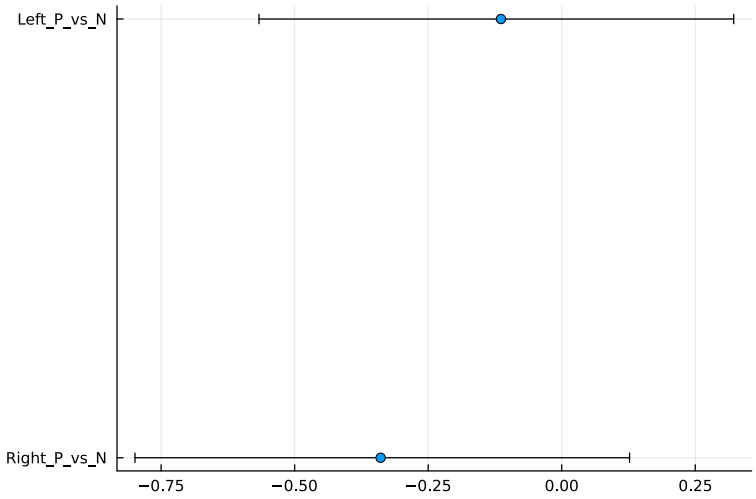



```
let
    names = ["R/N", "L/N", "R/P", "L/P"]
    labs = DataFrame(map(i -> names[i] =>
    m11_4_df[:,"b[$i]"], 1:4)...)
    @show size(labs)
    @show first(labs,3)
    # the order of coef plot is not same as names.
    coeftab_plot(labs)
end
```

```
size(labs) = (1000, 4)
first(labs, 3) = 3×4 DataFrame
 Row │ R/N        L/N         R/P        L/P
     │ Float64    Float64     Float64    Float64
─────┼──────────────────────────────────────────────
   1 │ -0.437684  -0.136985   -0.778798  0.0958664
   2 │ -0.398359  -0.0134336  -0.901249  0.0624103
   3 │ -0.365186   0.216916   -0.497192  0.205603
```

Code 11.14: Compare prob of pull_left between partner and no-partner presence.

```
md" ### Code 11.14: Compare prob of pull_left between partner and no-partner presence."
```




```
let
    diffs = DataFrame(
        Right_P_vs_N=m11_4_df."b[3]" .- m11_4_df."b[1]",
        Left_P_vs_N=m11_4_df."b[4]" .- m11_4_df."b[2]",
    )
    coeftab_plot(diffs)
end
```

- When food is on the left, almost no difference in probability of pulling left with or without partner.
- When food is on the right, slightly less likely to pull left (more likely to pull right) with a partner than without. Slight prosocial evidence.

```
md"- When food is on the left, almost no difference in probability of pulling left with or without partner.
-  When food is on the right, slightly less likely to pull left (more likely to pull right) with a partner than without. Slight prosocial evidence."
```

Code 11.15: Proportions of left pulls for each actor in each treatment

```
md" ## Code 11.15: Proportions of left pulls for each actor in each treatment"
```

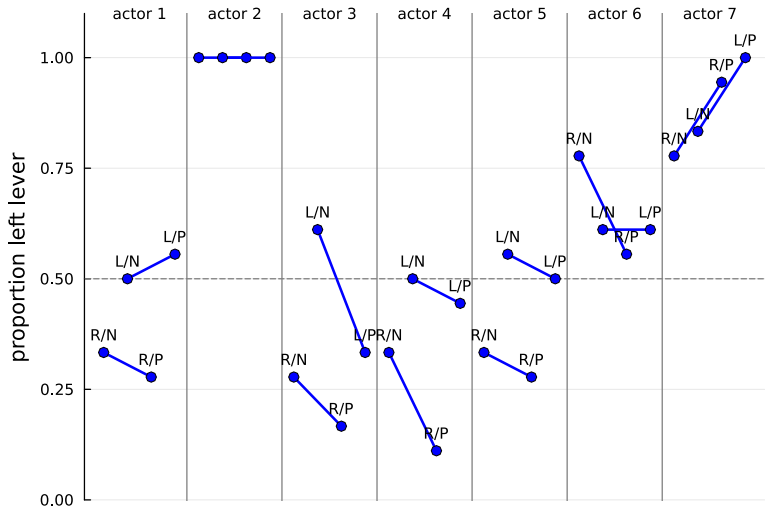
DataFrameRow (5 columns)

Row	actor	1	2	3	4
	Int64	Float64?	Float64?	Float64?	Float64?
1	1	0.333333	0.5	0.277778	0.555556

```
let
    gd = groupby(chimpanzees, [:actor, :treatment])
    c = combine(gd, :pulled_left => mean => :val)
    global pl = unstack(c, :actor, :treatment, :val)
    pl[1,:]
end
```

Code 11.16: Proportions of left pulls in the data

```
md" ## Code 11.16: Proportions of left pulls in the data"
```

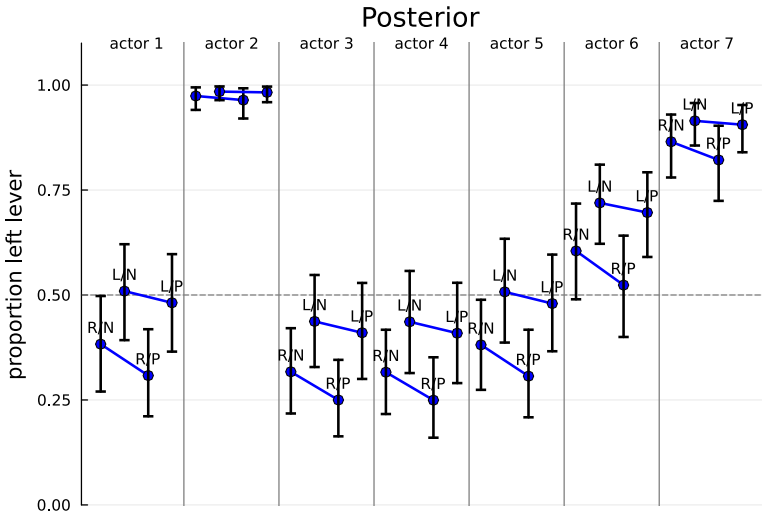



```
let
    names = ["R/N", "L/N", "R/P", "L/P"]
    p = plot(ylims=(0, 1.1), ylab="proportion left lever",
    showaxis=:y, xticks=false)
    hline!([0.5], c=:gray, s=:dash)
    for actor in 1:7
        ofs = (actor-1)*4
        actor > 1 && vline!([ofs+0.5], c=:gray)
        plot!([ofs+1,ofs+3], collect(pl[actor,["1","3"]]),
    lw=2, m=:o, c=:blue)
        plot!([ofs+2,ofs+4], collect(pl[actor,["2","4"]]),
    lw=2, m=:o, c=:blue)
        anns = [
            (ofs+idx, pl[actor,string(idx)]+.04, (name, 8))
            for (idx,name) ∈ enumerate(names)
        ]
        actor != 2 && annotate!(anns)
    end

    annotate!([
        (2.5 + (idx-1)*4, 1.1, ("actor $idx", 8))
        for idx ∈ 1:7
    ])
    p
end
```

Code 11.17: `m11_4` posterior predictions.

```
md" ## Code 11.17: `m11_4` posterior predictions."
```




```
let
    l_fun = (r, (ai, bi)) -> logistic(get(r, "a[$ai]",
    missing) + get(r, "b[$bi]", missing))
    p_post = link(m11_4_df, l_fun, Iterators.product(1:7,
    1:4))
    p_μ = map(mean, p_post)
    p_ci = map(PI, p_post);

    # visualize mean and cred intervals of posterior
    distribution

    # compute relative intervals
    rel_ci = map(idx -> (p_μ[idx]-p_ci[idx][1], p_ci[idx]
    [2]-p_μ[idx]), CartesianIndices(p_ci))

    names = ["R/N", "L/N", "R/P", "L/P"]
    p = plot(ylims=(0, 1.1), ylab="proportion left lever",
    title="Posterior", showaxis=:y, xticks=false)
    hline!([0.5], c=:gray, s=:dash)
    for actor in 1:7
        ofs = (actor-1)*4
        actor > 1 && vline!([ofs+0.5], c=:gray)
        err = [rel_ci[actor,1], rel_ci[actor,3]]
        plot!([ofs+1,ofs+3], collect(p_μ[actor,[1,3]]),
    err=err, lw=2, m=:o, c=:blue)
        err = [rel_ci[actor,2], rel_ci[actor,4]]
        plot!([ofs+2,ofs+4], collect(p_μ[actor,[2,4]]),
    err=err, lw=2, m=:o, c=:blue)
        anns = [
            (ofs+idx, p_μ[actor,idx]+.04, (name, 8))
            for (idx,name) ∈ enumerate(names)
        ]
        actor != 2 && annotate!(anns)
    end

    annotate!([
        (2.5 + (idx-1)*4, 1.1, ("actor $idx", 8))
        for idx ∈ 1:7
    ])
    p
end
```

Code 11.18-19: `m11_5`, split treatment into two variables.

```
md" ## Code 11.18-19: `m11_5`, split treatment into two variables."
```

```
begin
    chimpanzees.side = chimpanzees.prosoc_left .+ 1
    chimpanzees.cond = chimpanzees.condition .+ 1;
end;
```

```
m11_5 (generic function with 2 methods)
```

```
@model function m11_5(actor, side, cond, pulled_left)
    act_levels = length(levels(actor))
    a ~ MvNormal(zeros(act_levels), 1.5)
    side_levels = length(levels(side))
    bs ~ MvNormal(zeros(side_levels), 0.5)
    cond_levels = length(levels(cond))
    bc ~ MvNormal(zeros(cond_levels), 0.5)

    p = @. logistic(a[actor] + bs[side] + bc[cond])
    for i ∈ eachindex(pulled_left)
        pulled_left[i] ~ Binomial(1, p[i])
    end
end
```

	variable	mean	min	median	max
1	Symbol("a[1]")	-0.615305	-1.98699	-0.59705	0.784308
2	Symbol("a[2]")	3.82935	1.75427	3.80516	7.10686
3	Symbol("a[3]")	-0.917844	-2.32418	-0.922928	0.282024
4	Symbol("a[4]")	-0.917937	-2.63657	-0.92453	0.315609
5	Symbol("a[5]")	-0.613839	-2.14824	-0.615079	0.787937
6	Symbol("a[6]")	0.31551	-1.2284	0.313658	1.58189
7	Symbol("a[7]")	1.79773	0.06224	1.77555	3.84113
8	Symbol("bc[1]")	0.252171	-0.862571	0.247378	1.34487
9	Symbol("bc[2]")	0.00761282	-1.42254	0.0108904	1.03439
10	Symbol("bs[1]")	-0.19632	-1.32937	-0.209468	0.955004
11	Symbol("bs[2]")	0.497677	-0.669889	0.514969	1.49326

```
begin
    @time m11_5_chain = sample(m11_5(chimpanzees.actor, chimpanzees.side, chimpanzees.cond, chimpanzees.pulled_left), NUTS(), 1000)
    m11_5_df = DataFrame(m11_5_chain)
    describe(m11_5_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.4
```

```
  6.047116 seconds (3.22 M allocations: 4.539 GiB, 5.87% gc time)
```

Code 11.20: Compare `m11_4` vs `m11_5` via PSIS

```
md" ### Code 11.20: Compare `m11_4` vs `m11_5` via PSIS"
```

```
l_fun1 = #27 (generic function with 1 method)
```

```
l_fun1 = (r, (ai, bi, pull_left)) -> begin
    p = logistic(get(r, "a[$ai]", 0) + get(r, "b[$bi]", 0))
    binomlogpdf(1, p, pull_left)
end
```

```
m11_4_ll_1 =
  [[-0.353985, -0.478014, -0.575215, -0.484109, -0.55734, -0.355,
```

```
m11_4_ll_1 = link(m11_4_df, l_fun1, zip(chimpanzees.actor, chimpanzees.treatment, chimpanzees.pulled_left))
```

```
m11_4_ll_2 = hcat(m11_4_ll_1...);
```

```
l_fun2 = #29 (generic function with 1 method)
```

```
l_fun2 = (r, (ai, si, ci, pull_left)) -> begin
    p = logistic(get(r, "a[$ai]", 0) + get(r, "bs[$si]", 0) + get(r, "bc[$ci]", 0))
    binomlogpdf(1, p, pull_left)
end
```

	models	PSIS	lppd	SE	dPSIS	dSE	pPSI
1	"m11_4"	531.8	515.24	18.89	0.0	0.0	8.85
2	"m11_5"	23505.3	10016.2	1150.03	22973.5	1159.96	16952

```
begin
    m11_5_ll_3 = link(m11_5_df, l_fun, zip(chimpanzees.actor, chimpanzees.side, chimpanzees.cond, chimpanzees.pulled_left))
    m11_5_ll_4 = hcat(m11_5_ll_3...);

    compare([m11_5_ll_4, m11_4_ll_2], :psis, mnames=["m11_5", "m11_4"])
end
```

- m11_4 is better and has fewer effective number of parameters

```
md"- m11_4 is better and has fewer effective number of parameters"
```

Code 11.21 & 11.22 were omitted, as they are stan-specific

```
md"## Code 11.21 & 11.22 were omitted, as they are stan-specific"
```

Code 11.23: `m11_4`, relative odds (p/1-p) of pulling left of treatment 2 vs 4 (adding a parter), when food is on the left.

```
md" ## Code 11.23: `m11_4`, relative odds (p/1-p) of pulling left of treatment 2 vs 4 (adding a parter), when food is on the left."
```

```
0.9271746455591969
```

```
mean(@. exp(m11_4_df."b[4]" - m11_4_df."b[2]"))
```

Code 11.24: Aggregated data.

```
md" ## Code 11.24: Aggregated data."
```

	treatment	actor	side	cond	left_pulls
1	1	1	1	1	6
2	1	2	1	1	18
3	1	3	1	1	5
4	1	4	1	1	6
5	1	5	1	1	6
6	1	6	1	1	14
7	1	7	1	1	14
8	2	1	2	1	9

```
let
    gb = groupby(chimpanzees, [:treatment, :actor, :side, :cond])
    global d_aggregated = combine(gb, :pulled_left => sum => :left_pulls)
    first(d_aggregated, 8)
end
```

Code 11.25: `m11_6`, Aggregated #left-pull ~ actor + treatment

```
md" ### Code 11.25: `m11_6`, Aggregated #left-pull ~ actor + treatment"
```

```
m11_6 (generic function with 2 methods)
```

```
@model function m11_6(actor, treatment, left_pulls)
    act_levels = length(levels(actor))
    a ~ MvNormal(zeros(act_levels), 1.5)
    treat_levels = length(levels(treatment))
    b ~ MvNormal(zeros(treat_levels), 0.5)

    p = @. logistic(a[actor] + b[treatment])
    for i ∈ eachindex(left_pulls)
        left_pulls[i] ~ Binomial(18, p[i])
    end
end
```

	a[1]	a[2]	a[3]	a[4]	a[5]	
1	-0.432671	2.81658	-0.625481	-0.588585	-0.5328	0.
2	-0.974294	3.2472	-1.04284	-1.2618	-0.630291	0.
3	0.0725099	3.85358	-0.48418	-0.226174	-0.127853	0.
4	-0.138014	4.51484	-0.252708	-0.483543	0.167865	0.
5	-0.162932	3.55526	-0.439181	-0.519729	0.228057	0.
6	-0.132688	3.99287	-0.688315	-0.0672385	-0.100338	0.
7	-0.444038	3.71139	-0.581698	-1.21662	-0.450134	0.
8	-0.234228	4.01124	-0.606364	-1.07806	-0.19428	0.
9	-0.885567	4.72198	-0.472631	-0.420528	-0.25121	0.
10	-0.889908	4.931	-0.499748	-0.536377	-0.287978	0.
	more					
1000	-0.660098	2.90624	-0.64813	-0.501873	-0.382198	0.

```
begin
    m11_6_chain = sample(m11_6(d_aggregated.actor,
    d_aggregated.treatment, d_aggregated.left_pulls),
    NUTS(), 1000)
    m11_6_df = DataFrame(m11_6_chain)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.4
```

Code 11.26: Compare `m11_4` vs `m11_6` via PSIS

```
md" ### Code 11.26: Compare `m11_4` vs `m11_6` via PSIS"
```

```
l_fun = #31 (generic function with 1 method)
```

```
l_fun = (r, (ai, bi, left_pulls)) -> begin
    p = logistic(get(r, "a[$ai]", 0) + get(r, "b[$bi]", 0))
    binomlogpdf(18, p, left_pulls)
end
```

```
begin
    m11_6_ll = link(m11_6_df, l_fun,
    zip(d_aggregated.actor, d_aggregated.treatment,
    d_aggregated.left_pulls))
    m11_6_ll = hcat(m11_6_ll...)

    try
        compare([m11_6_ll, m11_4_ll_2], :psis, mnames=
    ["m11_6", "m11_4"])
    catch e
        println(e)
    end
end
```

```
DimensionMismatch("arrays could not be broadcast to
a common size; got a dimension with lengths 28 and 50
4")
```

- Two models with different number of data points. Can't directly compare.
- But can compare separately.

	models	PSIS	lppd	SE	dPSIS	dSE	pPSI
1	"m11_6"	111.3	97.37	7.96	0.0	0.0	7.64

```
compare([m11_6_ll], :psis, mnames=["m11_6"])
```

	models	PSIS	lppd	SE	dPSIS	dSE	pPSI
1	"m11_4"	531.8	515.24	18.89	0.0	0.0	8.85

```
compare([m11_4_ll_2], :psis, mnames=["m11_4"])
```

Code 11.27: aggregated probabilities larger.

- there are more ways to see the data. So the PSIS/WAIC scores end up being smaller.
- difference is entirely meaningless. It is just a side effect of how we organized the

data. The posterior distribution for the probability of success on each trial will end up the same, either way.

```
md" ### Code 11.27: aggregated probabilities larger.
- there are more ways to see the data. So the PSIS/WAIC
scores end up being smaller.
-  difference is entirely meaningless. It is just a side
effect of how we organized the
data. The posterior distribution for the probability of
success on each trial will end up the same, either way."
```

```
(11.7905, 20.6521)
```

```
(
    # deviance of aggregated 6-in-9
    -2*binomlogpdf(9, 0.2, 6),
    # deviance of dis-aggregated
    -2*sum(binomlogpdf.(1, 0.2, [1,1,1,1,1,1,0,0,0]))
)
```

Code 11.28: UC Berkeley Male vs Female admission rate

```
md" ## Code 11.28: UC Berkeley Male vs Female admission
rate"
```

ucbadmit =

	dept	gender	admit	reject	applications
1	"A"	"male"	512	313	825
2	"A"	"female"	89	19	108
3	"B"	"male"	353	207	560
4	"B"	"female"	17	8	25
5	"C"	"male"	120	205	325
6	"C"	"female"	202	391	593
7	"D"	"male"	138	279	417
8	"D"	"female"	131	244	375
9	"E"	"male"	53	138	191
10	"E"	"female"	94	299	393
11	"F"	"male"	22	351	373
12	"F"	"female"	24	317	341

```
ucbadmit = CSV.read(sr_datadir("UCBadmit.csv"), DataFrame)
```

Code 11.29: `m11_7`, a Binomial whose p only depends on Male/Female

```
md" ## Code 11.29: `m11_7`, a Binomial whose p only depends on Male/Female"
```

	admit	applications	gid
1	512	825	1
2	89	108	2
3	353	560	1
4	17	25	2
5	120	325	1
6	202	593	2
7	138	417	1
8	131	375	2
9	53	191	1
10	94	393	2
11	22	373	1
12	24	341	2

```
begin
    dat = ucbadmit[!, [:admit, :applications]]
    dat.gid = @. ifelse(ucbadmit.gender == "male", 1, 2)
    dat
end
```

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

```
m11_7 (generic function with 2 methods)
```

```
@model function m11_7(admit, applications, gid)
    a ~ MvNormal([0, 0], 1.5)
    p = @. logistic(a[gid])
    for i ∈ eachindex(applications)
        admit[i] ~ Binomial(applications[i], p[i])
    end
end
```

	variable	mean	min	median	max
1	Symbol("a[1]")	-0.22222	-0.352231	-0.223466	-0.0892277
2	Symbol("a[2]")	-0.830155	-1.01898	-0.829242	-0.684664

```
begin
    m11_7_chain = sample(m11_7(dat.admit, dat.applications,
    dat.gid), NUTS(), 1000)
    m11_7_df = DataFrame(m11_7_chain)
    describe(m11_7_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.05
```

Code 11.30: Check the difference in a & p(admission probability) between male and female

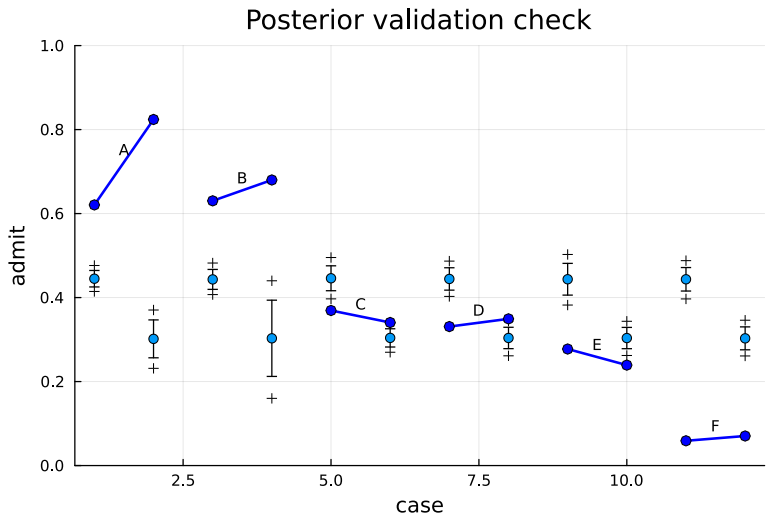
```
md" ### Code 11.30: Check the difference in a & p(admission
probability) between male and female"
```

	variable	mean	min	median	max	nmissing
1	:diff_a	0.607935	0.388663	0.608644	0.849225	0
2	:diff_p	0.140981	0.0910707	0.141111	0.192437	0

```
let
    diff_a = m11_7_df."a[1]" .- m11_7_df."a[2]"
    diff_p = @. logistic(m11_7_df."a[1]") -
    logistic(m11_7_df."a[2]")
    describe(DataFrame(diff_a=diff_a, diff_p=diff_p))
end
```

Code 11.31: `m11_7` fits poorly.

```
md" ### Code 11.31: `m11_7` fits poorly."
```




```
let
    Random.seed!(1)

    fun = (r, (apps, gid)) -> begin
        p = logistic(get(r, "a[$gid]", 0))
        rand(Binomial(apps, p))
    end

    admit_pred = link(m11_7_df, fun, zip(dat.applications,
    dat.gid))
    admit_rate = @. admit_pred ./ dat.applications

    # plot
    xrange = 1:12
    p = plot(yrange=(0, 1), title="Posterior validation
    check", xlab="case", ylab="admit")
    scatter!(xrange, mean.(admit_rate), yerr=std.
    (admit_rate))
    pi_rate = PI.(admit_rate)
    scatter!(xrange, first.(pi_rate), shape=:cross,
    c=:black)
    scatter!(xrange, last.(pi_rate), shape=:cross, c=:black)

    dat_rates = dat.admit ./ dat.applications

    for idx in 1:2:12
        r = [dat_rates[idx], dat_rates[idx+1]]
        plot!([idx, idx+1], r, mark=:o, lw=2, c=:blue)
        annotate!([(idx+0.5, mean(r)+0.03,
    (ucbadmit.dept[idx], 8))])
    end
    p
end
```

Code 11.32: `m11_8` consider each department separately.

```
md" ## Code 11.32: `m11_8` consider each department
separately."
```

DataFrame

```
begin
    @show size(dat)
    @show typeof(dat)
end
```

```
size(dat) = (12, 4)
typeof(dat) = DataFrames.DataFrame
```

```
reshape(adjoint(::Matrix{Int64}), 12): [1, 1, 2, 2, 3, 3, 4, 4, 5
```

```
dat.dept_id = reshape(hcat(1:6, 1:6)', 12)
```

	admit	applications	gid	dept_id
1	512	825	1	1
2	89	108	2	1
3	353	560	1	2
4	17	25	2	2
5	120	325	1	3
6	202	593	2	3
7	138	417	1	4
8	131	375	2	4
9	53	191	1	5
10	94	393	2	5
11	22	373	1	6
12	24	341	2	6

```
dat
```

```
model_m11_8 (generic function with 2 methods)
```

```
@model function model_m11_8(admit, applications, gid,
dept_id)
    a ~ MvNormal([0, 0], 1.5)
    delta ~ MvNormal(zeros(6), 1.5)
    p = @. logistic(a[gid] + delta[dept_id])
    for i ∈ eachindex(applications)
        admit[i] ~ Binomial(applications[i], p[i])
    end
end
```

	variable	mean	min	median	ma
1	Symbol("a[1]")	-0.524277	-1.71867	-0.586839	0.684
2	Symbol("a[2]")	-0.427836	-1.55949	-0.481259	0.754
3	Symbol("delta[1]")	1.10492	-0.168497	1.17146	2.362
4	Symbol("delta[2]")	1.06151	-0.192133	1.11204	2.270
5	Symbol("delta[3]")	-0.15511	-1.39352	-0.0971642	1.006
6	Symbol("delta[4]")	-0.186793	-1.39906	-0.137429	1.012
7	Symbol("delta[5]")	-0.636415	-1.87819	-0.586936	0.571
8	Symbol("delta[6]")	-2.18372	-3.43125	-2.12868	-0.93

```
begin
    m11_8n = sample(model_m11_8(dat.admit,
    dat.applications, dat.gid, dat.dept_id), NUTS(), 1000)
    m11_8_df = DataFrame(m11_8n)
    describe(m11_8_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.2
```

Code 11.33: difference in a & p(admission probability) between male and female

```
md" ### Code 11.33: difference in a & p(admission probability) between male and female"
```

	variable	mean	min	median	max	nm
1	:diff_a	-0.0964411	-0.387561	-0.0946801	0.203117	0
2	:diff_p	-0.0218944	-0.0925247	-0.0211127	0.0464518	0

```
let
    diff_a = m11_8_df."a[1]" .- m11_8_df."a[2]"
    diff_p = logistic.(m11_8_df."a[1]") .- logistic.
    (m11_8_df."a[2]")
    describe(DataFrame(diff_a=diff_a, diff_p=diff_p))
end
```

Code 11.34: applications ratio for each department by sex

```
md" ## Code 11.34: applications ratio for each department by sex"
```

	gender	A	B	C	D	E
1	"male"	0.884244	0.957265	0.354031	0.526515	0.327055
2	"female"	0.115756	0.042735	0.645969	0.473485	0.672945

```
let
    gb1 = groupby(ucbadmit, :dept)
    fun = (as,gs) -> [(gender=g, a_ratio=a/sum(as)) for
    (a,g) in zip(as,gs)]
    c = combine(gb1, [:applications, :gender] => fun =>
    AsTable)
    unstack(c, :gender, :dept, :a_ratio)
end
```

Code 11.33: not sure why this appears again.

```
md" ## Code 11.33: not sure why this appears again."
```

	variable	mean	min	median	max	nm
1	:diff_a	-0.0964411	-0.387561	-0.0946801	0.203117	0
2	:diff_p	-0.0218944	-0.0925247	-0.0211127	0.0464518	0

```
begin
    diff_a = m11_8_df."a[1]" .- m11_8_df."a[2]"
    diff_p = logistic.(m11_8_df."a[1]") .- logistic.
    (m11_8_df."a[2]")
    describe(DataFrame(diff_a=diff_a, diff_p=diff_p))
end
```

gb =
GroupedDataFrame with 6 groups based on key: dept

First Group (2 rows): dept = "A"

Row	dept	gender	admit	reject	applications
	String1	String7	Int64	Int64	Int64
1	A	male	512	313	825
2	A	female	89	19	108

⋮

Last Group (2 rows): dept = "F"

Row	dept	gender	admit	reject	applications
	String1	String7	Int64	Int64	Int64
1	F	male	22	351	373
2	F	female	24	317	341

```
# Code 11.34

gb = groupby(ucbadmit, :dept)
```

fun = #39 (generic function with 1 method)

```
fun = (as,gs) -> [(gender=g, a_ratio=a/sum(as)) for (a,g)
in zip(as,gs)]
```

c =

	dept	gender	a_ratio
1	"A"	"male"	0.884244
2	"A"	"female"	0.115756
3	"B"	"male"	0.957265
4	"B"	"female"	0.042735
5	"C"	"male"	0.354031
6	"C"	"female"	0.645969
7	"D"	"male"	0.526515
8	"D"	"female"	0.473485
9	"E"	"male"	0.327055
10	"E"	"female"	0.672945
11	"F"	"male"	0.522409
12	"F"	"female"	0.477591

```
c = combine(gb, [:applications, :gender] => fun => AsTable)
```

	gender	A	B	C	D	E
1	"male"	0.884244	0.957265	0.354031	0.526515	0.327055
2	"female"	0.115756	0.042735	0.645969	0.473485	0.672945

```
unstack(c, :gender, :dept, :a_ratio)
```

11.2 Poisson regression.

```
md" # 11.2 Poisson regression."
```

Code 11.35 Poisson is limit of Binomial(N, p), N->∞, p -> 0.

- Mean and std of a Binomial(N, p), N=1K, p=0.001 are almost identical.

```
md" ## Code 11.35 Poisson is limit of Binomial(N, p), N->$\infty$, p -> 0.
- Mean and std of a Binomial(N, p), N=1K, p=0.001 are almost identical. "
```

```
(1.00067, 1.00412)
```

```
let
    Random.seed!(1)
    y_bin = rand(Binomial(1000, 1/1000), 10^5)
    mean(y_bin), var(y_bin)
end
```

Code 11.36 Load the societies & tools Kline data

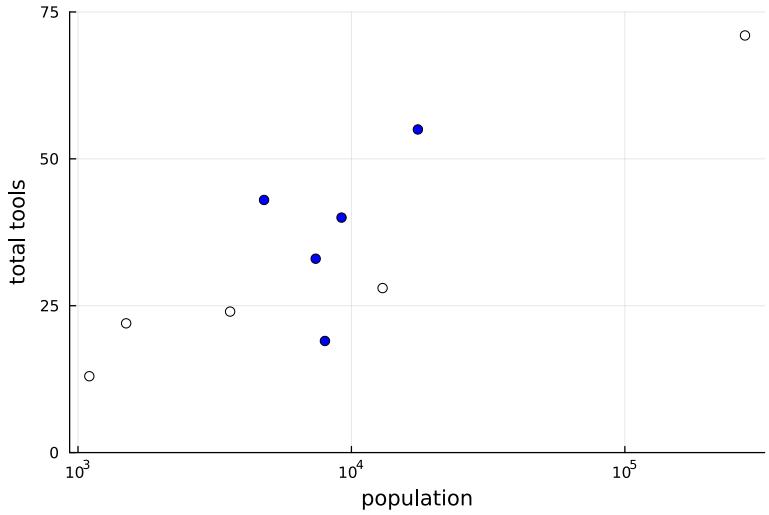
```
md" ## Code 11.36 Load the societies & tools Kline data"
```

```
[1, 1, 1, 2, 2, 2, 2, 1, 2, 1]
```

```
begin
    kline = CSV.read(sr_datadir("Kline.csv"), DataFrame)
    kline.P = standardize(ZScoreTransform, log.(kline.population))
    kline.contact_id = ifelse.(kline.contact .== "high", 2, 1)
end
```

	variable	mean	min	median	max	n
1	:culture	nothing	"Chuuk"	nothing	"Yap"	0
2	:population	34109.1	1100	7700.0	275000	0
3	:contact	nothing	"high"	nothing	"low"	0
4	:total_tools	34.8	13	30.5	71	0
5	:mean_TU	4.83	3.2	4.85	6.6	0
6	:P	1.02141e-15	-1.29147	-0.0188353	2.32101	0
7	:contact_id	1.5	1	1.5	2	0

```
describe(kline)
```

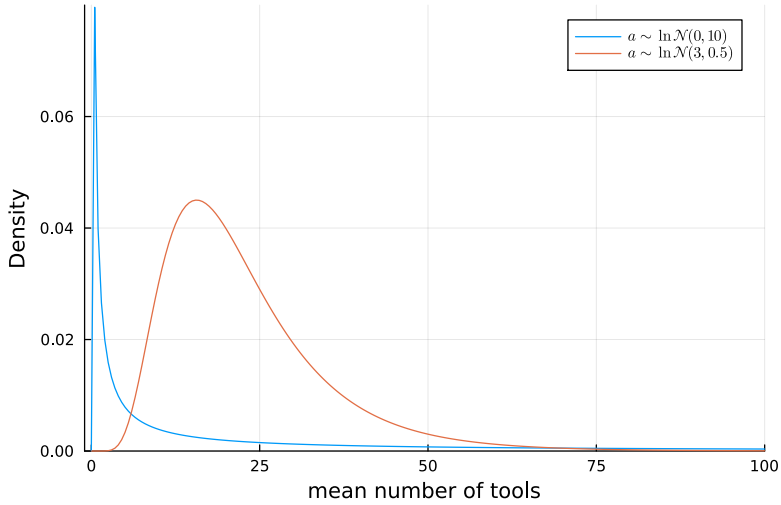



```
let
    mark_color = ifelse.(kline.contact_id .== 1, :white,
    :blue)
    scatter(kline.population, kline.total_tools,
    xlab="population", ylab="total tools",
        ylim=(0, 75), mc=mark_color, xaxis=:log10)
end
```

Code 11.38 Prior distribution of λ with different log-normal parameters

```

md" ## Code 11.38 Prior distribution of $\lambda$ with
different log-normal parameters"
```




```
let
    p = plot(xlims=(-1, 100), ylims=(0, 0.08),
        xlab="mean number of tools", ylab="Density")
    r = 0:0.5:100
    plot!(r, LogNormal(0, 10), label=L"a \sim \ln
    \mathcal{N}(0, 10)")
    plot!(r, LogNormal(3, 0.5), label=L"a \sim \ln
    \mathcal{N}(3, 0.5)")
end
```

Code 11.39: Very large Poisson rate with a Normal(0,10) prior

```
md" ## Code 11.39: Very large Poisson rate with a Normal(0,10) prior"
```

3.744194573171826e15

```
let
    a = rand(Normal(0, 10), 10^4)
    λ = exp.(a)
    @show λ̄ = mean(λ)
end
```

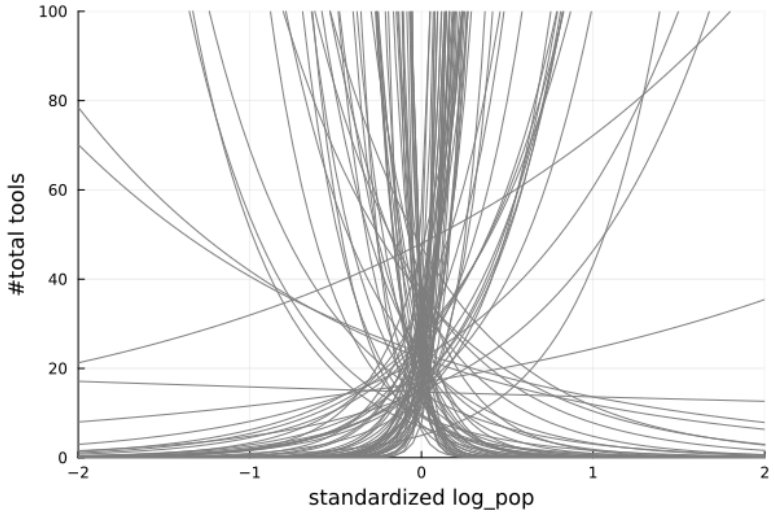
```
λ̄ = mean(λ) = 3.744194573171826e15
```

Code 11.40: joined with 11.38

```
md" ## Code 11.40: joined with 11.38"
```

Code 11.41: Prior trends of #total tools given standardized log population

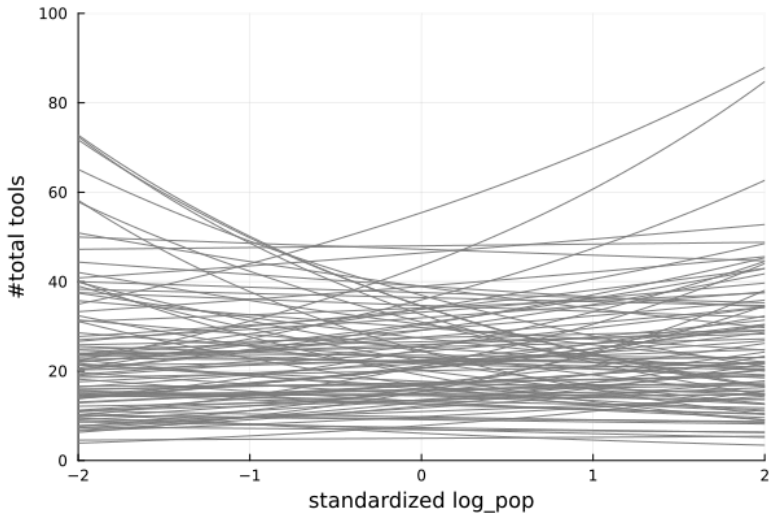
```
md" ## Code 11.41: Prior trends of #total tools given standardized log population"
```




```
let
    Random.seed!(1)
    N = 100
    α = rand(Normal(3, 0.5), N)
    β = rand(Normal(0, 10), N)
    p = plot(xlims=(-2, 2), ylims=(0, 100),
    xlab="standardized log_pop", ylab="#total tools")
    r = -2:0.05:2
    for (aᵢ, bᵢ) ∈ zip(α, β)
        plot!(r, x -> exp(aᵢ + bᵢ * x), c=:gray)
    end
    p
end
```

Code 11.42: Narrow the Normal σ to obtain a flatter prior

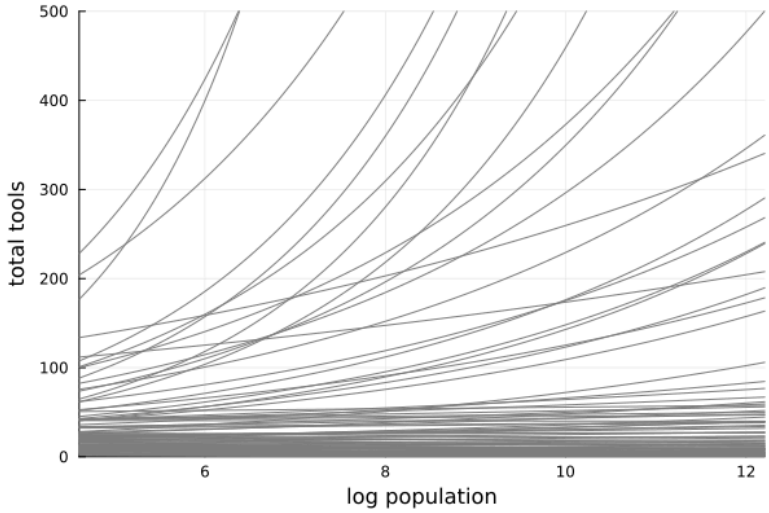
```
md" ## Code 11.42: Narrow the Normal σ to obtain a flatter prior"
```




```
let
    Random.seed!(1)
    N = 100
    α = rand(Normal(3, 0.5), N)
    β = rand(Normal(0, .2), N)
    p = plot(xlims=(-2, 2), ylims=(0, 100),
    xlab="standardized log_pop", ylab="#total tools")
    x_range = -2:0.05:2
    for (αᵢ, βᵢ) ∈ zip(α, β)
        plot!(x_range, x -> exp(αᵢ + βᵢ * x), c=:gray)
    end
    p
end
```

Code 11.43: X-axis on unstandardized scale.

```
md" ## Code 11.43: X-axis on unstandardized scale."
```

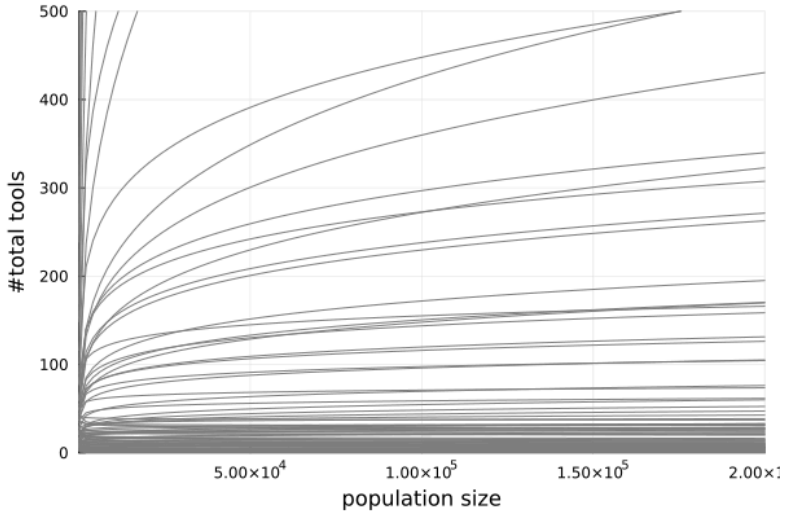



```
let
    N = 100
    x_seq = range(log(100), log(200_000), length=100)
    a = rand(Normal(3, 0.5), N)
    b = rand(Normal(0, .2), N)
    λ = map(x -> (@. exp(a + b * x)), x_seq)
    @show size(λ)
    λ = hcat(λ...)
    @show size(λ)
    p = plot(xlims=extrema(x_seq), ylims=(0, 500),
    xlab="log population", ylab="total tools")
    for λᵢ ∈ eachrow(λ)
        plot!(x_seq, λᵢ, c=:gray)
    end
    p
end
```

```
size(λ) = (100,)
size(λ) = (100, 100)
```

Code 11.44. X-axis (population size) on natural scale.

```
md" ## Code 11.44. X-axis (population size) on natural
scale."
```




```
let
    N = 100
    x_seq = range(100, 200_000, length=100)
    a = rand(Normal(3, 0.5), N)
    b = rand(Normal(0, .2), N)
    λ = map(x -> (@. exp(a + b * log(x))), x_seq)
    @show extrema(x_seq)
    @show size(λ)
    λ = hcat(λ...)
    @show size(λ)
    p = plot(xlims=extrema(x_seq), ylims=(0, 500),
        xlab="population size", ylab="#total tools")
    for λᵢ ∈ eachrow(λ)
        plot!(x_seq, λᵢ, c=:gray)
    end
    p
end
```

```
extrema(x_seq) = (100.0, 200000.0)
size(λ) = (100,)
size(λ) = (100, 100)
```

Code 11.45 `m11_9` & `m11_10`.

- m11_9: intercept only model. Every island has the same rate.
- m11_10: interaction between contact & population size.

```
md" ## Code 11.45 `m11_9` & `m11_10`.
- m11_9: intercept only model. Every island has the same
rate.
  - m11_10: interaction between contact & population size."
```

```
m11_9 (generic function with 2 methods)
```

```
# intercept only
@model function m11_9(T)
    a ~ Normal(3, 0.5)
    λ = exp(a)
    T ~ Poisson(λ)
end
```

	a
1	3.46553
2	3.57456
3	3.57566
4	3.51728
5	3.63992
6	3.6264
7	3.58334
8	3.40385
9	3.65227
10	3.52106
	more
1000	3.53975

```
begin
    m11_9_ch = sample(m11_9(kline.total_tools), NUTS(),
    1000)
    m11_9_df = DataFrame(m11_9_ch)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.2
```

```
m11_10 (generic function with 2 methods)
```

```
# interaction model
@model function m11_10(T, cid, P)
    a ~ MvNormal([3, 3], 0.5)
    b ~ MvNormal([0, 0], 0.2)
    λ = @. exp(a[cid] + b[cid]*P)
    for i ∈ eachindex(T)
        T[i] ~ Poisson(λ[i])
    end
end
```

@time m11_10_ch =

	iteration	chain	a[1]	a[2]	b[1]	b[2]	
1	501	1	3.30373	3.62535	0.473277	0.0725445	-4(
2	502	1	3.34336	3.59945	0.283818	0.322327	-4(
3	503	1	3.2849	3.6118	0.319857	-0.104533	-4:
4	504	1	3.37969	3.76079	0.35086	0.33172	-4:
5	505	1	3.35772	3.45825	0.361852	0.0510922	-4:
6	506	1	3.3545	3.62793	0.407491	0.400259	-3:
7	507	1	3.25575	3.58257	0.331323	0.0383401	-4(
8	508	1	3.25575	3.58257	0.331323	0.0383401	-4(
9	509	1	3.30911	3.6364	0.467695	0.360489	-4(
10	510	1	3.16249	3.69397	0.401286	0.119838	-4:

more

```
1 @time m11_10_ch = sample(m11_10(kline.total_tools,
  kline.contact_id, kline.P), NUTS(), 1000)
```

100%

Found initial step size
ϵ: 0.2

```
  0.586028 seconds (1.02 M allocations: 101.450 Mi
B, 4.97% gc time)
```

```
1 m11_10_df = DataFrame(m11_10_ch);
```

	variable	mean	min	median	max	nn
1	Symbol("a[1]")	3.32478	3.04435	3.32648	3.57487	0
2	Symbol("a[2]")	3.61055	3.36253	3.61082	3.85607	0
3	Symbol("b[1]")	0.377476	0.224731	0.376304	0.532294	0
4	Symbol("b[2]")	0.198729	-0.326732	0.201975	0.756263	0

```
1 describe(m11_10_df)
```

Code 11.46 Compare `m11_9` vs `m11_10`

```
1 md" ### Code 11.46 Compare `m11_9` vs `m11_10`"
```

	models	PSIS	lppd	SE	dPSIS	dSE	pPS
1	"m11_10"	82.4	71.3	11.96	0.0	0.0	5.78
2	"m11_9"	139.8	125.06	31.39	57.4	31.36	7.93

```
let
    f = (r, T) -> poislogpdf(exp(r.a), T)
    ll_m11_9 = link(m11_9_df, f, kline.total_tools)
    ll_m11_9 = hcat(ll_m11_9...)
    f = (r, (T, cid, P)) -> poislogpdf(exp(get(r,
    "a[$cid]", 0) + get(r, "b[$cid]", 0)*P), T)
    likeL_m11_10 = link(m11_10_df, f,
    zip(kline.total_tools, kline.contact_id, kline.P))
    likeL_m11_10 = hcat(likeL_m11_10...)
    global m11_9_vs_m11_10_by_waic =
    StatisticalRethinking.compare([ll_m11_9, likeL_m11_10],
    :waic, mnames=["m11_9", "m11_10"])
    global m11_9_vs_m11_10_by_psis =
    StatisticalRethinking.compare([ll_m11_9, likeL_m11_10],
    :psis, mnames=["m11_9", "m11_10"])
end
```

	models	WAIC	lppd	SE	dWAIC	dSE	pWA
1	"m11_10"	82.8	71.3	12.13	0.0	0.0	5.78
2	"m11_9"	140.9	125.06	31.84	58.1	33.79	7.93

```
m11_9_vs_m11_10_by_waic
```

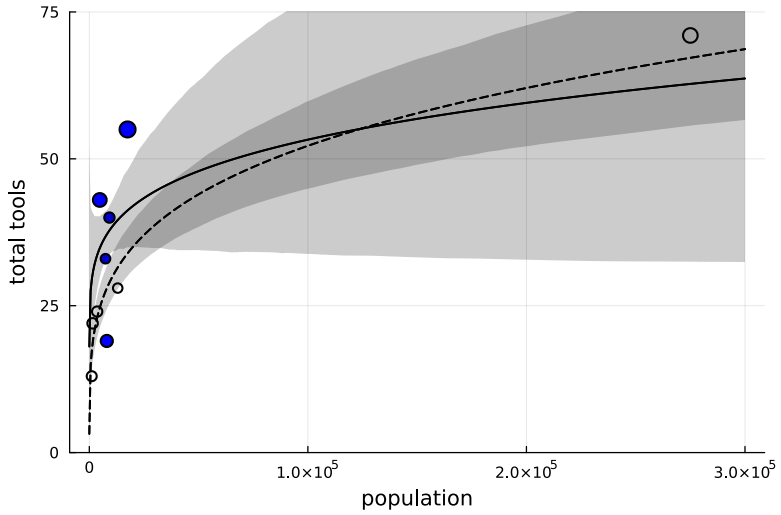
```
3×10 view(::Matrix{Float64}, 1:3, :) with eltype Float64:
 -3.93019  -2.46852  -2.58788  -3.5986   …  -2.80514  -2.96834  -
 -3.81295  -2.46827  -2.65585  -4.33616     -2.82499  -3.44175  -
 -2.2164   -4.60206  -3.59107  -4.1843      -3.02325  -2.74629  -
```

```
let
    # cell above uses likeL_m11_10
    # because ll_m11_10 is an existing matrix, maybe from
    some old cell.
    @show typeof(ll_m11_10)
    @show size(ll_m11_10)
    @view ll_m11_10[1:3, :]
end
```

```
typeof(ll_m11_10) = Matrix{Float64}
size(ll_m11_10) = (1000, 10)
```

Code 11.47-48: Label the highly influential points

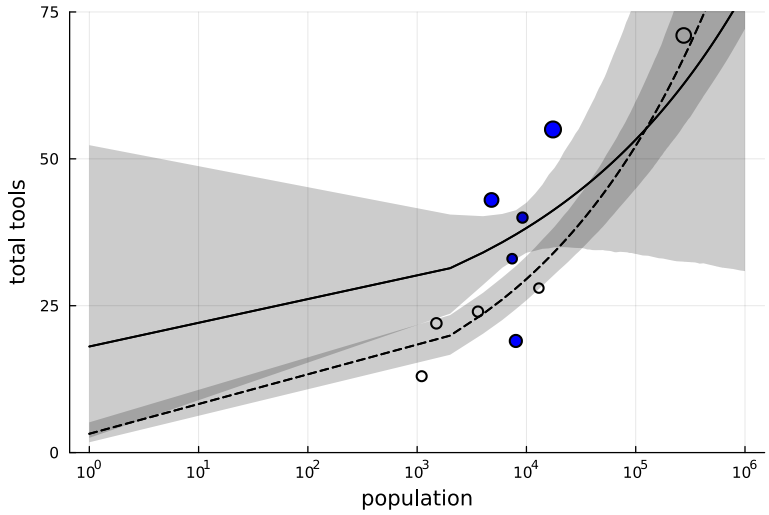
```
md" ### Code 11.47-48: Label the highly influential points"
```




```
let
    # get psis K values
    t = ll_m10'
    m10_t = collect(reshape(t, size(t)..., 1))
    PSIS_m10 = psis_loo(m10_t)
    k = PSIS_m10.pointwise(:pareto_k)
    mark_size = standardize(ZScoreTransform, k).+5
    mark_color = ifelse.(kline.contact_id .== 1, :white,
    :blue)
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=mark_size, mc=mark_color, msw=2)
    ns = 500
    logpop_m, logpop_std = mean(log.(kline.population)),
    std(log.(kline.population))
    P_seq = range(1, 300_000, length=ns)
    # cid=1 predictions
    f = (r,p) -> exp(r."a[1]" + r."b[1]"*(log(p)-
    logpop_m)/logpop_std )
    λ = link(m11_10_df, f, P_seq)
    λ = hcat(λ...)
    λ1_mean = mean.(eachcol(λ))
    λ1_ci = PI.(eachcol(λ))
    λ1_ci = vcat(λ1_ci'...)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2,
        ls=:dash, c=:black, lw=1.5)
    f = (r,p) -> exp(r."a[2]" + r."b[2]"*(log(p)-
    logpop_m)/logpop_std )
    # cid=2 predictions
    λ = link(m11_10_df, f, P_seq)
    λ = hcat(λ...)
    λ2_mean = mean.(eachcol(λ))
    λ2_ci = PI.(eachcol(λ))
    λ2_ci = vcat(λ2_ci'...)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
        fillalpha=0.2, c=:black, lw=1.5)
    # Code 11.48
    #=
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=mark_size, mc=mark_color, msw=2, xaxis=:log10)
    #P_seq = range(-5, 3, length=ns)
    # 1.53 is sd of log(population)
    # 9 is mean of log(population)
    #pop_seq = @. exp(P_seq * 1.53 + 9)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2, ls=:dash, c=:black, lw=1.5, xaxis=:log10)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
    fillalpha=0.2, c=:black, lw=1.5, xaxis=:log10)
    =#
```

```
end
```

```
No source provided for samples; variables are assumed t
o be from a Markov Chain. If the samples are independen
t, specify this with keyword argument `source=:other`.
```




```
let
    # get psis K values
    t = ll_m10'
    m10_t = collect(reshape(t, size(t)..., 1))
    PSIS_m10 = psis_loo(m10_t)
    k = PSIS_m10.pointwise(:pareto_k)
    mark_size = standardize(ZScoreTransform, k).+5
    mark_color = ifelse.(kline.contact_id .== 1, :white,
    :blue)
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=mark_size, mc=mark_color, msw=2, xaxis=:log10)
    ns = 500
    logpop_m, logpop_std = mean(log.(kline.population)),
    std(log.(kline.population))
    P_seq = range(1, 1000_000, length=ns)
    # cid=1 predictions
    f = (r,p) -> exp(r."a[1]" + r."b[1]"*(log(p)-
    logpop_m)/logpop_std )
    λ = link(m11_10_df, f, P_seq)
    λ = hcat(λ...)
    λ1_mean = mean.(eachcol(λ))
    λ1_ci = PI.(eachcol(λ))
    λ1_ci = vcat(λ1_ci'...)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2,
        ls=:dash, c=:black, lw=1.5)
    f = (r,p) -> exp(r."a[2]" + r."b[2]"*(log(p)-
    logpop_m)/logpop_std )
    # cid=2 predictions
    λ = link(m11_10_df, f, P_seq)
    λ = hcat(λ...)
    λ2_mean = mean.(eachcol(λ))
    λ2_ci = PI.(eachcol(λ))
    λ2_ci = vcat(λ2_ci'...)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
        fillalpha=0.2, c=:black, lw=1.5)
    # Code 11.48
    #=
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=mark_size, mc=mark_color, msw=2, xaxis=:log10)
    #P_seq = range(-5, 3, length=ns)
    # 1.53 is sd of log(population)
    # 9 is mean of log(population)
    #pop_seq = @. exp(P_seq * 1.53 + 9)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2, ls=:dash, c=:black, lw=1.5, xaxis=:log10)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
    fillalpha=0.2, c=:black, lw=1.5, xaxis=:log10)
    =#
```

```
end
```

```
No source provided for samples; variables are assumed t
o be from a Markov Chain. If the samples are independen
t, specify this with keyword argument `source=:other`.
```

Code 11.49: `m11_11`, Dynamic model with rate=0 if population size=0

```
md" ## Code 11.49: `m11_11`, Dynamic model with rate=0 if
population size=0"
```

```
m11_11 (generic function with 2 methods)
```

```
@model function m11_11(T, P, cid)
    a ~ MvNormal([1,1], 1)
    b₁ ~ Exponential(1)
    b₂ ~ Exponential(1)
    b = [b₁, b₂]
    g ~ Exponential(1)
    λ = @. exp(a[cid]) * P ^ b[cid] / g
    for i ∈ eachindex(T)
        T[i] ~ Poisson(λ[i])
    end
end
```

```
@time m11_11_ch =
```

	iteration	chain	a[1]	a[2]	b₁	b₂	
1	501	1	-0.391892	1.21383	0.252012	0.099245	(
2	502	1	0.936484	2.83103	0.269741	0.080837	(
3	503	1	1.00024	2.05055	0.224131	0.133116	(
4	504	1	1.99223	2.24032	0.230988	0.216428	2
5	505	1	0.733262	1.07032	0.257919	0.234114	(
6	506	1	1.35087	0.551475	0.244434	0.3529	1
7	507	1	0.239392	2.24236	0.329369	0.144032	(
8	508	1	0.361064	2.07118	0.310107	0.170788	(
9	509	1	0.581997	2.01903	0.30093	0.179399	(
10	510	1	1.05662	1.934	0.22965	0.154713	(

more

```
@time m11_11_ch = sample(m11_11(kline.total_tools,
kline.population, kline.contact_id), NUTS(), 1000)
```

```
100%
```

```
Found initial step size
ϵ: 6.103515625e-6
```

```
  2.197323 seconds (7.53 M allocations: 933.778 Mi
B, 7.03% gc time)
```

m11_11_df =

	a[1]	a[2]	b_1	b_2	g
1	-0.391892	1.21383	0.252012	0.099245	0.230722
2	0.936484	2.83103	0.269741	0.080837	0.999209
3	1.00024	2.05055	0.224131	0.133116	0.678028
4	1.99223	2.24032	0.230988	0.216428	2.01474
5	0.733262	1.07032	0.257919	0.234114	0.749316
6	1.35087	0.551475	0.244434	0.3529	1.21731
7	0.239392	2.24236	0.329369	0.144032	0.875121
8	0.361064	2.07118	0.310107	0.170788	0.897907
9	0.581997	2.01903	0.30093	0.179399	0.93587
10	1.05662	1.934	0.22965	0.154713	0.782179
more					
1000	0.0359832	-0.256444	0.216408	0.28457	0.279476

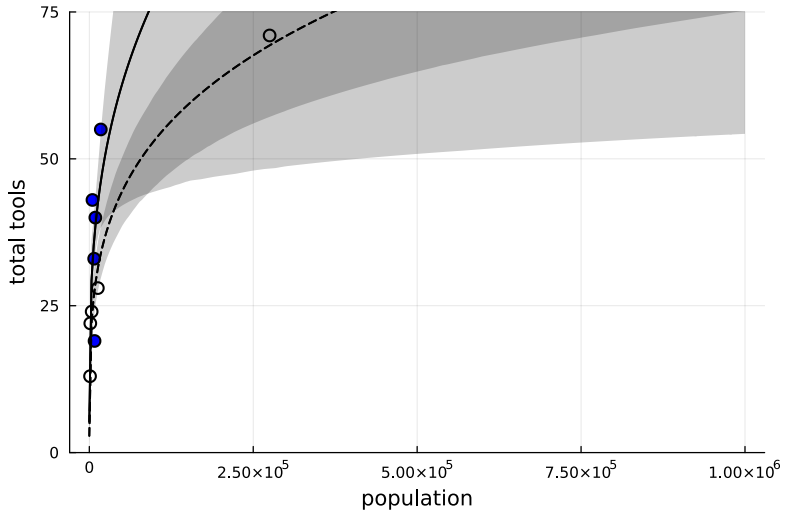
```
m11_11_df = DataFrame(m11_11_ch)
```

	variable	mean	min	median	max	n
1	Symbol("a[1]")	0.846537	-1.38151	0.874638	3.02551	0
2	Symbol("a[2]")	0.982461	-1.67794	1.00742	3.80374	0
3	:b_1	0.258828	0.144949	0.260166	0.371622	0
4	:b_2	0.276251	0.00440361	0.27432	0.598927	0
5	:g	1.10284	0.081964	0.916338	4.76032	0

```
describe(m11_11_df)
```

- High contact society/islands have slightly higher α (coefficient in front of pop size) and β (exponent of pop size).

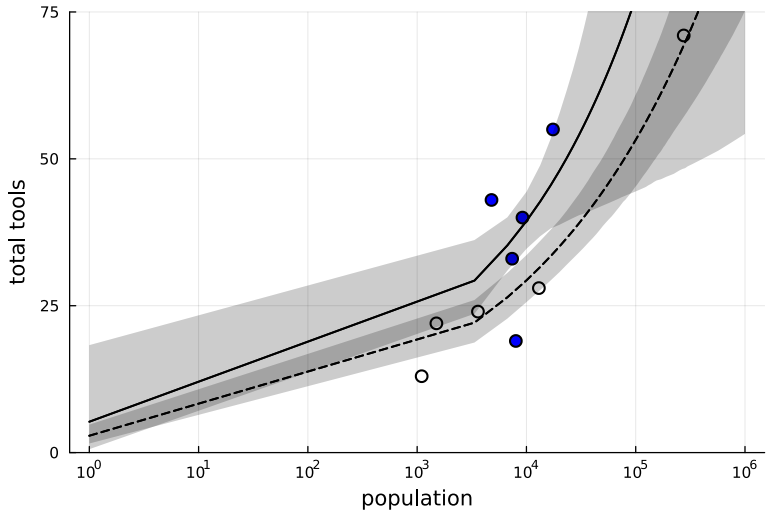
```
md"
- High contact society/islands have slightly higher α
(coefficient in front of pop size) and β (exponent of pop
size)."
```




```
let
    mark_color = ifelse.(kline.contact_id .== 1, :white,
    :blue)
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=5,
        mc=mark_color, msw=2)

    ns = 300
    P_seq = range(1, 10^6, length=ns)
    # cid=1 predictions
    f = (r,p) -> exp(r."a[1]") * p ^ r.b₁ / r.g
    λ = link(m11_11_df, f, P_seq)
    λ = hcat(λ...)
    λ1_mean = mean.(eachcol(λ))
    λ1_ci = PI.(eachcol(λ))
    λ1_ci = vcat(λ1_ci'...)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2,
        ls=:dash, c=:black, lw=1.5)
    f = (r,p) -> exp(r."a[2]") * p ^ r.b₂ / r.g
    # cid=2 predictions
    λ = link(m11_11_df, f, P_seq)
    λ = hcat(λ...)
    λ2_mean = mean.(eachcol(λ))
    λ2_ci = PI.(eachcol(λ))
    λ2_ci = vcat(λ2_ci'...)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
    fillalpha=0.2, c=:black,
        lw=1.5)

end
```




```
let
    mark_color = ifelse.(kline.contact_id .== 1, :white,
    :blue)
    scatter(kline.population, kline.total_tools,
        xlab="population", ylab="total tools",
        ylim=(0, 75),
        ms=5,
        mc=mark_color, msw=2, xaxis=:log10)
    ns = 300
    P_seq = range(1, 10^6, length=ns)
    # cid=1 predictions
    f = (r,p) -> exp(r."a[1]") * p ^ r.b₁ / r.g
    λ = link(m11_11_df, f, P_seq)
    λ = hcat(λ...)
    λ1_mean = mean.(eachcol(λ))
    λ1_ci = PI.(eachcol(λ))
    λ1_ci = vcat(λ1_ci'...)
    plot!(P_seq, [λ1_mean λ1_mean], fillrange=λ1_ci,
    fillalpha=0.2,
        ls=:dash, c=:black, lw=1.5)
    f = (r,p) -> exp(r."a[2]") * p ^ r.b₂ / r.g
    # cid=2 predictions
    λ = link(m11_11_df, f, P_seq)
    λ = hcat(λ...)
    λ2_mean = mean.(eachcol(λ))
    λ2_ci = PI.(eachcol(λ))
    λ2_ci = vcat(λ2_ci'...)
    plot!(P_seq, [λ2_mean λ2_mean], fillrange=λ2_ci,
    fillalpha=0.2, c=:black,
        lw=1.5)

end
```

Code 11.50: Simulate a month of daily manuscript production at a monastery.

```
md" ## Code 11.50: Simulate a month of daily manuscript
production at a monastery."
```

```
[1, 1, 0, 1, 2, 1, 2, 3, 1, 2, 2, 0, 1, 2, 0, 3, 2, 1, 2, 2, 1, 2,
```

```
begin
    Random.seed!(1)
    num_days = 30
    y = rand(Poisson(1.5), num_days)
end
```

Code 11.51: Simulate 4 weeks of weekly manuscript production at another monastery.

```
md" ## Code 11.51: Simulate 4 weeks of weekly manuscript production at another monastery."
```

```
[1, 1, 0, 1, 2, 1, 2, 3, 1, 2, 2, 0, 1, 2, 0, 3, 2, 1, 2, 2, 1, 2,
```

```
begin
    Random.seed!(2)
    num_weeks = 4
    y_new = rand(Poisson(0.5*7), num_weeks)
    y_all = [y; y_new]
end
```

Code 11.52: Combine two monasteries into a dataframe.

```
md" ## Code 11.52: Combine two monasteries into a dataframe."
```

	y	days	monastery
1	1	1	0
2	1	1	0
3	0	1	0
4	1	1	0
5	2	1	0
6	1	1	0
7	2	1	0
8	3	1	0
9	1	1	0
10	2	1	0
	more		
34	6	7	1

```
begin
    exposure = [repeat([1], 30); repeat([7], 4)]
    monastery = [repeat([0], 30); repeat([1], 4)]
    expmon = DataFrame(y=y_all, days=exposure,
    monastery=monastery)
end
```

```
m11_12 (generic function with 2 methods)
```

```
@model function m11_12(y, log_days, monastery)
    a ~ Normal()
    b ~ Normal()
    λ = @. exp(log_days + a + b*monastery)
    @. y ~ Poisson(λ)
end
```

Code 11.53: `m11_12` Fit data of two monasteries with different offsets.

```
md" ## Code 11.53: `m11_12` Fit data of two monasteries with different offsets."
```

	variable	mean	min	median	max	nmissi
1	:a	0.29415	-0.118089	0.301367	0.689338	0
2	:b	-0.968469	-1.98439	-0.955207	-0.160673	0

```
begin
    Random.seed!(1)
    expmon.log_days = log.(expmon.days)
    @time m11_12_chain = sample(m11_12(expmon.y,
    expmon.log_days, expmon.monastery), NUTS(), 1000)
    m11_12_df = DataFrame(m11_12_chain)
    describe(m11_12_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.4
```

```
  0.999724 seconds (753.04 k allocations: 79.366 Mi
B, 14.25% gc time)
```

Code 11.54: Compare production rates of two monasteries

```
md" ## Code 11.54: Compare production rates of two monasteries"
```

Note

Values are slightly different from the book. This is due to non-Normal distributions (you can check this yourself with `optimize`)

	variable	mean	min	median	max	nmissing	
1	:λ_old	1.35692	0.888617	1.35171	1.9924	0	F
2	:λ_new	0.524821	0.230096	0.515848	1.00162	0	F

```
let
    λ_old = exp.(m11_12_df.a)
    λ_new = @. exp(m11_12_df.a + m11_12_df.b)
    describe(DataFrame(λ_old=λ_old, λ_new=λ_new))
end
```

11.3 Multinomial and categorical models

```
md" # 11.3 Multinomial and categorical models"
```

Code 11.55. Simulate data. Included in 11.57

```
md" ## Code 11.55. Simulate data. Included in 11.57"
```

Code 11.56: `m11_13`, different LMs with the same set of predictors.

- Career choice ~ expected income.

```
md" #### Code 11.56: `m11_13`, different LMs with the same
set of predictors.
- Career choice ~ expected income."
```

```
m11_13 (generic function with 2 methods)
```

```
@model function m11_13(career, income)
    a ~ MvNormal([0, 0], 1)
    #b ~ TruncatedNormal(0, 0.5, 0, Inf) no longer exist.
    b ~ Truncated(Normal(0, 0.5), 0, Inf)
    # the 3rd career's probability is held as constant.
    same beta, but different alpha.
    p = softmax([a[1] + b*income[1], a[2] + b*income[2], 0])
    career ~ Categorical(p)
end
```

Code 11.57: Simulate and fit `m11_13`

```
md" ## Code 11.57: Simulate and fit `m11_13`"
```

	variable	mean	min	median	max
1	Symbol("a[1]")	-2.34287	-2.90733	-2.33399	-1.72228
2	Symbol("a[2]")	-1.99189	-2.7949	-1.95606	-1.37037
3	:b	0.121509	2.07798e-5	0.0892325	0.517194

```
let
    # expected income of 3 careers
    global income = [1,2,5]
    c_score = 0.5*income
    @show p = softmax(c_score)
    @show w = Weights(p)
    @show typeof(w)

    # simulate choice
    Random.seed!(34302)
    # simulate career choice among 500 individuals
    N = 500
    career = [sample(w) for _ in 1:N]
    Random.seed!(121)
    @time m11_13_chain = sample(m11_13(career, income),
    NUTS(), 5000, n_chains=4)
    global m11_13_df = DataFrame(m11_13_chain)
    describe(m11_13_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.2
```

```
p = softmax(c_score) = [0.09962364806231834, 0.1642
5162762508783, 0.7361247243125939]
w = Weights(p) = [0.09962364806231834, 0.1642516276250
8783, 0.7361247243125939]
typeof(w) = StatsBase.Weights{Float64, Float64, Vector
{Float64}}
  1.808311 seconds (3.77 M allocations: 316.651 MiB,
3.88% gc time)
```

Code 11.58. Double the expected income of the 2nd career and check how the probabilities change.

```
md" ## Code 11.58. Double the expected income of the 2nd
career and check how the probabilities change."
```

	variable	mean	min	median	max	nmissin
1	:p_diff	0.0332507	5.94844e-6	0.0221128	0.17602	0

```
let
    # logit scores
    s1 = m11_13_df."a[1]" + m11_13_df.b * income[1]
    s2_orig = m11_13_df."a[2]" + m11_13_df.b * income[2]
    s2_new = m11_13_df."a[2]" + m11_13_df.b * income[2] * 2
    # compute probabilities for original and counterfactual
    p_orig = softmax.(eachrow(hcat(s1, s2_orig,
    zeros(length(s1)))))
    p_orig = hcat(p_orig...)'
    p_new = softmax.(eachrow(hcat(s1, s2_new,
    zeros(length(s1)))))
    p_new = hcat(p_new...)'
    p_diff = p_new[:, 2] - p_orig[:, 2]
    describe(DataFrame(p_diff=p_diff))
end
```

- 2.2% probability increase in choosing the 2nd career if its income is doubled.

```
md"- 2.2% probability increase in choosing the 2nd career
if its income is doubled."
```

Code 11.59: `m11_14`, use family income instead. Different β for different carreer choices.

```
md" ## Code 11.59: `m11_14`, use family income instead.
Different $\beta$ for different carreer choices."
```

```
m11_14 (generic function with 2 methods)
```

```
@model function m11_14(career, family_income)
    a ~ MvNormal([0, 0], 1.5)
    b ~ MvNormal([0, 0], 1)

    for i ∈ eachindex(career)
        income = family_income[i]
        s = [a[1] + b[1] * income, a[2] + b[2] * income, 0]
        p = softmax(s)
        career[i] ~ Categorical(p)
    end
end
```

	variable	mean	min	median	max	nn
1	Symbol("a[1]")	-2.77846	-4.06288	-2.76885	-1.6696	0
2	Symbol("a[2]")	-1.04814	-1.71833	-1.04717	-0.410016	0
3	Symbol("b[1]")	-1.79132	-4.64158	-1.7493	0.0703574	0
4	Symbol("b[2]")	-1.8986	-3.28525	-1.88477	-0.557297	0

```
let
    Random.seed!(1)
    N = 500
    family_income = rand(Uniform(), N)
    #assume different β for 3 different careers.
    b = [-2, 0, 2]
    career = [
        begin
            score = (0.5 * 1:3) + b * inc
            p = softmax(score)
            sample(Weights(p))
            #@show Weights(p)
        end
        for inc in family_income
    ]
    @time m11_14_chain = sample(m11_14(career,
    family_income), NUTS(), 1000)
    m11_14_df = DataFrame(m11_14_chain)
    describe(m11_14_df)
end
```

```
100%
```

```
Found initial step size
ϵ: 0.2
```

```
  6.185097 seconds (22.37 M allocations: 3.566 GiB,
9.41% gc time)
```

- β estimates are quite off from the true values. Something wrong?

```
md"- β estimates are quite off from the true values.
Something wrong?"
```

Code 11.61: `m_binom` vs `m_poisson`

```
md" ## Code 11.61: `m_binom` vs `m_poisson`"
```

```
m_binom (generic function with 2 methods)
```

```
@model function m_binom(admit, applications)
    a ~ Normal(0, 1.5)
    p = logistic(a)
    @. admit ~ Binomial(applications, p)
end
```

@time m_binom_chain =

	iteration	chain	a	lp	n_steps	is_accept	a
1	501	1	-0.44658	-474.422	3.0	1.0	0
2	502	1	-0.454338	-474.371	3.0	1.0	1
3	503	1	-0.454338	-474.371	3.0	1.0	0
4	504	1	-0.45712	-474.369	3.0	1.0	0
5	505	1	-0.482546	-474.731	3.0	1.0	0
6	506	1	-0.487868	-474.895	1.0	1.0	0
7	507	1	-0.447239	-474.415	3.0	1.0	0
8	508	1	-0.451441	-474.383	3.0	1.0	1
9	509	1	-0.445711	-474.432	1.0	1.0	0
10	510	1	-0.479947	-474.662	3.0	1.0	0

more

```
@time m_binom_chain = sample(m_binom(ucbadmit.admit, ucbadmit.applications), NUTS(), 1000)
```

100%

Found initial step size
ϵ: 0.05

```
  0.519601 seconds (554.60 k allocations: 39.397 MiB)
```

```
m_binom_df = DataFrame(m_binom_chain);
```

	variable	mean	min	median	max	nmissi
1	:a	-0.458424	-0.551845	-0.459951	-0.354057	0

```
describe(m_binom_df)
```

```
describe(logistic.(m_binom_df[!, :a]))
```

```
Summary Stats:
Length:         1000
Missing Count:  0
Mean:           0.387383
Minimum:        0.365436
1st Quartile:   0.382720
Median:         0.386997
3rd Quartile:   0.392023
Maximum:        0.412399
Type:           Float64
```

m_pois (generic function with 2 methods)

```
@model function m_pois(admit, reject)
    a ~ MvNormal([0, 0], 1.5)
    λ = exp.(a)
    admit ~ Poisson(λ[1])
    reject ~ Poisson(λ[2])
end
```

m_pois_chain =

	iteration	chain	a[1]	a[2]	lp	n_steps	is_ac
1	501	1	5.0158	5.43363	-1305.29	3.0	1.0
2	502	1	4.9466	5.41377	-1306.56	3.0	1.0
3	503	1	4.98509	5.38948	-1307.95	3.0	1.0
4	504	1	4.994	5.47236	-1305.76	3.0	1.0
5	505	1	4.97855	5.42118	-1304.89	3.0	1.0
6	506	1	4.972	5.42535	-1304.79	3.0	1.0
7	507	1	4.97944	5.44568	-1304.36	3.0	1.0
8	508	1	5.032	5.42009	-1306.97	3.0	1.0
9	509	1	5.032	5.42009	-1306.97	1.0	1.0
10	510	1	4.97823	5.46044	-1304.86	3.0	1.0

more

```
m_pois_chain = sample(m_pois(ucbadmit.admit, ucbadmit.reject), NUTS(), 1000)
```

100%

Found initial step size
ϵ: 0.05

```
m_pois_df = DataFrame(m_pois_chain);
```

	variable	mean	min	median	max	nmissin
1	Symbol("a[1]")	4.98495	4.90547	4.98471	5.08733	0
2	Symbol("a[2]")	5.44098	5.38695	5.44184	5.50449	0

```
describe(m_pois_df)
```

Code 11.62: mean p (admission prob) in `m_binom`

```
md" ## Code 11.62: mean p (admission prob) in `m_binom`"
```

0.38738321480703797

```
m_binom_df.a .|> logistic |> mean
```

Code 11.63. Softmax over two Poisson rates can obtain the same average admission probability as m_binom.

```
md" ## Code 11.63. Softmax over two Poisson rates can
obtain the same average admission probability as m_binom."
```

a1 = 4.984954265506834

```
a1 = m_pois_df."a[1]" |> mean
```

a2 = 5.440981706692681

```
a2 = m_pois_df."a[2]" |> mean
```

0.387928647278407

```
exp(a1)/(exp(a1)+exp(a2))
```