Chap 15: Missing Data and Other Opportunities

```julia
md"# Chap 15: Missing Data and Other Opportunities"
```

```julia
versioninfo()
```

```
Julia Version 1.11.1
Commit 8f5b7ca12ad (2024-10-16 10:53 UTC)
Build Info:
  Official https://julialang.org/ release
Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 32 × Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
  WORD_SIZE: 64
  LLVM: libLLVM-16.0.6 (ORCJIT, haswell)
Threads: 16 default, 0 interactive, 8 GC (on 32 virtual cores)
Environment:
  JULIA_PKG_SERVER = https://mirrors.tuna.tsinghua.edu.cn/julia
  JULIA_REVISE_WORKER_ONLY = 1
```

```julia
html"""
<style>
    main {
        margin: 0 auto;
        max-width: max(1600px, 75%);
        padding-left: max(5px, 1%);
        padding-right: max(350px, 10%);
    }
</style>
"""
```

Table of Contents


```
begin
  using Pkg, DrWatson
  using PlutoUI
  TableOfContents()
end
```

```
begin
    using Turing
    using Turing
    using DataFrames
    using CSV
    using Random
    using Dagitty
    using Distributions
    using StatisticalRethinking
    #using StatisticalRethinking: link
    using StatisticalRethinkingPlots
    using StatsPlots
    using StatsBase
    using Logging
    using LinearAlgebra
    using LogExpFunctions # for logistic()
end
```

Code 15.1

```
md"## Code 15.1"
```

0.6617857711284418

```
begin
    Random.seed!(2)

    function sim_pancake()
        pancake = [[1, 1], [1, 0], [0, 0]]
        sides = sample(pancake)
        sample([sides, reverse(sides)])
    end

    @time pancakes = vcat([sim_pancake() for _ in 1:100_000]'...)
    up = pancakes[:,1]
    down = pancakes[:,2]

    num_11_10 = sum(up .== 1)
    num_11 = sum((up .== 1) .& (down .== 1))
    num_11 / num_11_10
end
```

```
  0.114458 seconds (1.65 M allocations: 6
4.906 MiB, 54.89% compilation time)
```

pancake = [[1, 1], [1, 0], [0, 0]]

```
pancake = [[1, 1], [1, 0], [0, 0]]
```

sides = [1, 1]

```
sides = sample(pancake)
```

[1, 1]

```
sample([sides, reverse(sides)])
```

[[1, 1], [1, 1]]

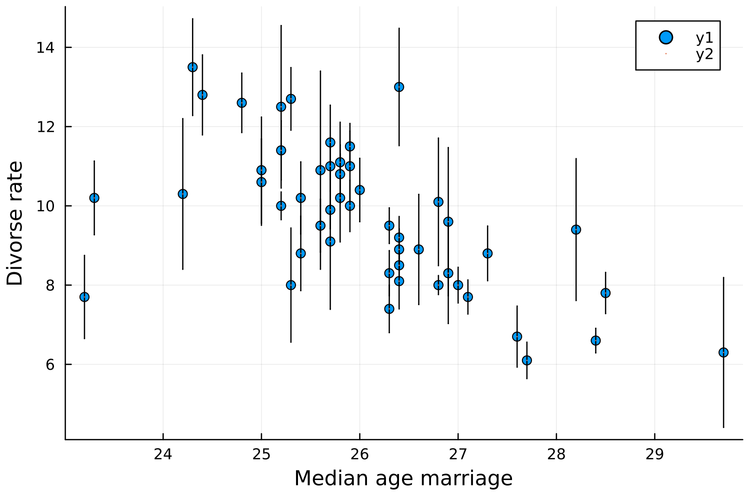
```
[sides, reverse(sides)]
```

15.1 Measurement error

```
md" # 15.1 Measurement error"
```

Code 15.2

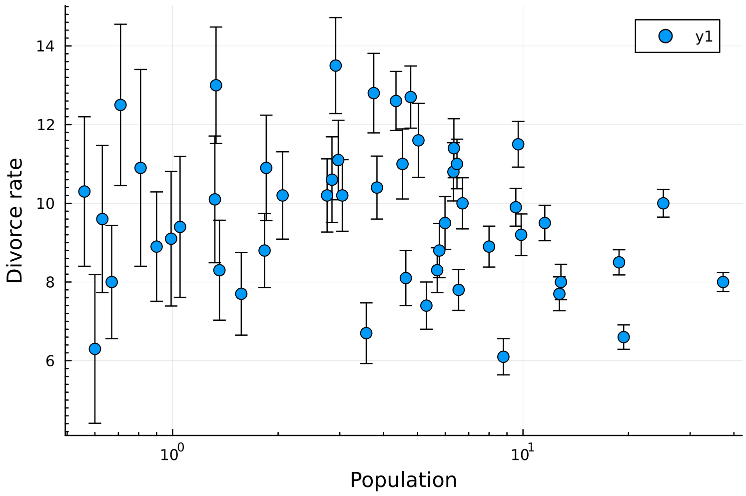
```
md"## Code 15.2"
```




```
begin
    d_divorce =
    DataFrame(CSV.File("data/WaffleDivorce.csv"))

    scatter(d_divorce.MedianAgeMarriage,
    d_divorce.Divorce,
        xlab="Median age marriage", ylab="Divorse
        rate")
    scatter!(d_divorce.MedianAgeMarriage,
    d_divorce.Divorce, yerror=d_divorce."Divorce
    SE", ms=0)
end
```

	Location	Loc	Population	MedianAgeMarriage
1	"Alabama"	"AL"	4.78	25.3
2	"Alaska"	"AK"	0.71	25.2
3	"Arizona"	"AZ"	6.33	25.8

```
first(d_divorce,3)
```




```
begin
    scatter(d_divorce.Population,
    d_divorce.Divorce, xaxis=:log10,
        xlab="Population", ylab="Divorce rate",
        xminorticks=9, yminorticks=10,
        yerror=d_divorce."Divorce SE", ms=5)
    #scatter!(d_divorce.Population,
    d_divorce.Divorce, yerror=d_divorce."Divorce
    SE", ms=0)
end
```

Code 15.3 model `m15_1`

```
md"## Code 15.3 model `m15_1`"
```

	D_true[10]	D_true[11]	D_true[12]	D_true[13]	D
1	-0.784182	0.591328	-0.412152	0.12795	-0
2	-0.418201	1.43575	-0.850778	0.108591	-0
3	-0.638492	0.392377	-0.361222	1.00863	-0
4	-0.509696	1.10383	-0.518881	0.27218	-0
5	-0.691987	0.769628	-0.666241	0.826797	-0
6	-0.521323	0.589482	-0.991947	-0.00510027	-1
7	-0.586819	0.689822	-0.252866	0.751031	-0
8	-0.582998	0.455282	-0.310074	0.56882	-0
9	-0.630576	0.790008	-0.888668	1.30604	-0
10	-0.630576	0.790008	-0.888668	1.30604	-0
more					
1000	-1.02493	0.883494	-0.144567	-0.00200293	-0

```
begin
    d_divorce_ls = (
        D_obs = standardize(ZScoreTransform, d_divorce.Divorce),
        D_sd = d_divorce."Divorce SE" ./ std(d_divorce.Divorce),
        M = standardize(ZScoreTransform, d_divorce.Marriage),
        A = standardize(ZScoreTransform, d_divorce.MedianAgeMarriage),
        N = nrow(d_divorce),
    )

    @model function m15_1(D_obs, D_sd, M, A, N)
        a ~ Normal(0, 0.2)
        bA ~ Normal(0, 0.5)
        bM ~ Normal(0, 0.5)
        μ = @. a + bA * A + bM * M
        σ ~ Exponential()
        D_true ~ MvNormal(μ, σ)
        @. D_obs ~ Normal(D_true, D_sd)
    end

    Random.seed!(1)
    @time m15_1_ch = sample(m15_1(d_divorce_ls...), NUTS(), 1000)
    m15_1_df = DataFrame(m15_1_ch);
end
```

Sampling 100%

Found initial step size
ϵ: 0.2

11.457821 seconds (16.59 M allocations: 6.154 GiB, 10.11% gc time, 55.48% compilation time)

Code 15.4

```
md"## Code 15.4"
```

	variable	mean	min	media
1	Symbol("D_true[10]")	-0.622426	-1.17513	-0.6214
2	Symbol("D_true[11]")	0.752743	-0.167793	0.76452
3	Symbol("D_true[12]")	-0.54162	-2.09472	-0.5389
4	Symbol("D_true[13]")	0.191023	-1.80048	0.19718
5	Symbol("D_true[14]")	-0.86873	-1.59464	-0.8784
6	Symbol("D_true[15]")	0.563774	-0.450136	0.55976
7	Symbol("D_true[16]")	0.269308	-0.855484	0.28287
8	Symbol("D_true[17]")	0.505615	-0.78145	0.50451
9	Symbol("D_true[18]")	1.25328	0.14058	1.25724
10	Symbol("D_true[19]")	0.428978	-0.812482	0.44128
more				
54	:σ	0.579131	0.30084	0.57578

```
describe(m15_1_df)
```

Code 15.5 model `m15_2`

```
md"## Code 15.5 model `m15_2`"
```

(D_obs = [1.65421, 1.54436, 0.610716, 2.09357, -0.927058

```
begin
    dlist2 = (
        D_obs = standardize(ZScoreTransform, d_divorce.Divorce),
        D_sd = d_divorce."Divorce SE" ./ std(d_divorce.Divorce),
        M_obs = standardize(ZScoreTransform, d_divorce.Marriage),
        M_sd = d_divorce."Marriage SE" ./ std(d_divorce.Marriage),
        A = standardize(ZScoreTransform, d_divorce.MedianAgeMarriage),
        N = nrow(d_divorce),
    )
end
```

```
[0.083057, 1.01903, 0.0594721, 1.41732, -0.266635, 0.830
```

```
begin

    @model function m15_2(D_obs, D_sd, M_obs, M_sd, A, N)
        a ~ Normal(0, 0.2)
        bA ~ Normal(0, 0.5)
        bM ~ Normal(0, 0.5)
        M_true ~ filldist(Normal(), N)

        μ = @. a + bA * A + bM * M_true
        σ ~ Exponential()
        D_true ~ MvNormal(μ, σ)
        @. D_obs ~ Normal(D_true, D_sd)
        @. M_obs ~ Normal(M_true, M_sd)
    end

    Random.seed!(1)
    @time m15_2_ch = sample(m15_2(dlist2...), NUTS(), 1000)
    m15_2_df = DataFrame(m15_2_ch);
    D_true = [mean(m15_2_df[!, "D_true[$i]"]) for i ∈ 1:dlist2.N]
    M_true = [mean(m15_2_df[!, "M_true[$i]"]) for i ∈ 1:dlist2.N]
end
```

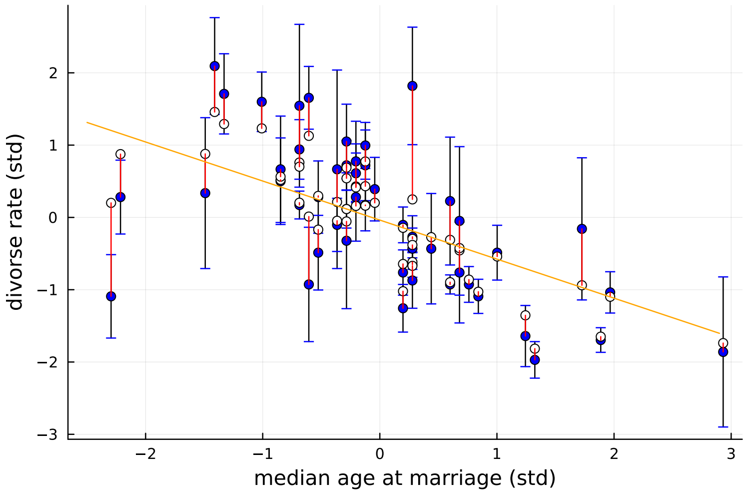
```
Sampling 100%

Found initial step size
ϵ: 0.4

37.370198 seconds (53.81 M allocations: 4
2.068 GiB, 22.55% gc time, 20.53% compilation
time)
```

	variable	mean	min	medi
1	Symbol("D_true[10]")	-0.616598	-1.09836	-0.616
2	Symbol("D_true[11]")	0.773391	-0.153289	0.7721
3	Symbol("D_true[12]")	-0.455932	-1.96422	-0.469
4	Symbol("D_true[13]")	0.201203	-1.44406	0.2043
5	Symbol("D_true[14]")	-0.860255	-1.57298	-0.859
6	Symbol("D_true[15]")	0.540992	-0.540644	0.5437
7	Symbol("D_true[16]")	0.297736	-0.943139	0.2935
8	Symbol("D_true[17]")	0.519618	-1.31079	0.5227
9	Symbol("D_true[18]")	1.23177	0.22005	1.2234
10	Symbol("D_true[19]")	0.431547	-0.906202	0.4161
	more			
104	:σ	0.563163	0.242072	0.5583

```
describe(m15_2_df)
```

Figure 15.2

```
md"## Figure 15.2"
```

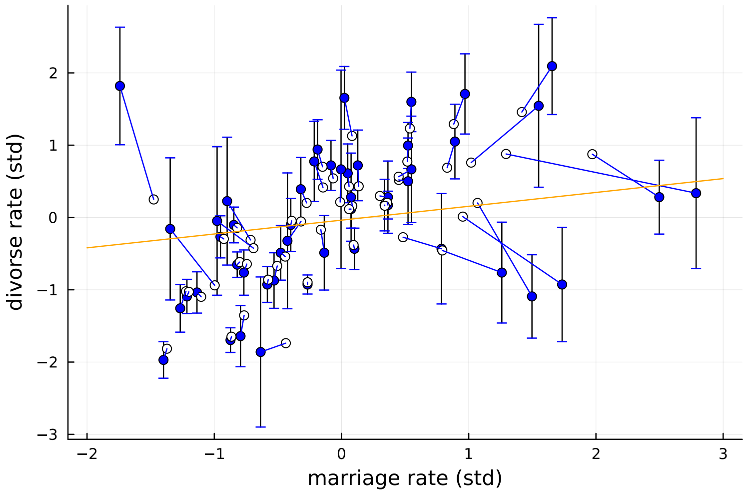



```
begin
    p15_1_1 = scatter(dlist2.A, dlist2.D_obs,
    mc=:blue, yerror=dlist2.D_sd,
        label="observed", xlab="median age at
    marriage (std)", ylab="divorse rate (std)")
    scatter!(dlist2.A, D_true, mc=:white,
    label="true")

    for i ∈ 1:dlist2.N
        plot!([dlist2.A[i], dlist2.A[i]],
    [dlist2.D_obs[i], D_true[i]], c=:red,
    legend=false)
    end
    x = -2.5:0.2:3
    y = -0.0368595 .+  -0.540089 .* x
    plot!(x,y, c=:orange, label="m15_2 estimate")
    p15_1_1
end
```

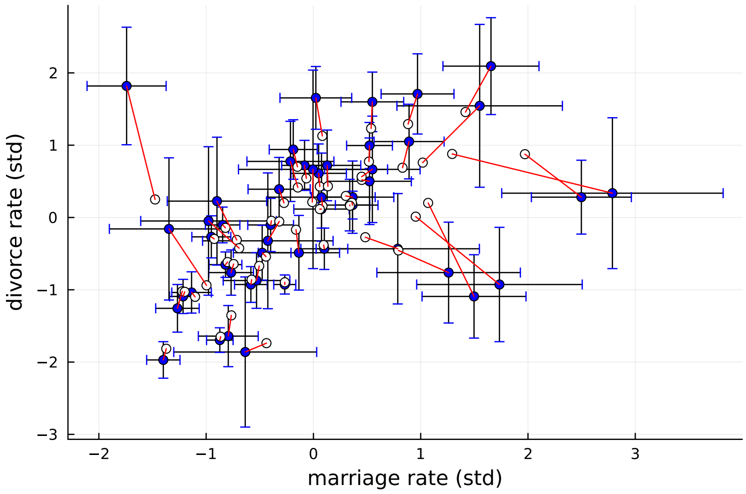
```
Enter cell code...
```

Code 15.6 Figure 15.3

```
md"## Code 15.6 Figure 15.3"
```




```
begin
    p15_1_2 = scatter(dlist2.M_obs, dlist2.D_obs, mc=:blue, yerror=dlist2.D_sd,
        label="observed", xlab="marriage rate (std)", ylab="divorse rate (std)",
        legend=true)
    scatter!(M_true, D_true, mc=:white, label="true", legend=true)

    for i ∈ 1:dlist2.N
        plot!([dlist2.M_obs[i], M_true[i]], [dlist2.D_obs[i], D_true[i]], c=:blue, legend=false)
    end
    x2 = -2:0.2:3
    y2 = -0.0368595 .+  0.1915 .* x2
    plot!(x2,y2, c=:orange, label="m15_2 estimate")
    p15_1_2
end
```




```
begin
    p3 = scatter(dlist2.M_obs, dlist2.D_obs,
    mc=:blue, xerror=dlist2.M_sd,
    yerror=dlist2.D_sd,
        label="observed", xlab="marriage rate
    (std)", ylab="divorce rate (std)")
    scatter!(M_true, D_true, mc=:white,
    label="true")

    for i ∈ 1:dlist2.N
        plot!([dlist2.M_obs[i], M_true[i]],
    [dlist2.D_obs[i], D_true[i]], c=:red,
    legend=false)
    end
    p3
end
```

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

Code 15.7

```
md"## Code 15.7"
```

[-0.860429, 0.151987, 2.67642, 0.24338, -1.82141, -1.683

```
let
    N = 500
    A = rand(Normal(), N)
    M = rand.(Normal.(-A))
    D = rand.(Normal.(A))
    A_obs = rand.(Normal.(A));
end
```

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

15.2 Missing data

```
md"# 15.2 Missing data"
```

`m15_3`

- UndefVarError: `logistic` not defined in Main.var
- Suggestion: check for spelling errors or missing imports.
- Hint: a global variable of this name may be made accessible by importing LogExpFunctions in the current active module Main
- Hint: a global variable of this name may be made accessible by importing StatsFuns in the current active module Main

```
md"## `m15_3`

- UndefVarError: `logistic` not defined in Main.var
- Suggestion: check for spelling errors or missing imports.
- Hint: a global variable of this name may be made accessible by importing LogExpFunctions in the current active module Main
- Hint: a global variable of this name may be made accessible by importing StatsFuns in the current active module Main
"
```

```
m15_3 (generic function with 2 methods)
```

```
begin
    @model function m15_3(H, S)
        a ~ Normal()
        bS ~ Normal(0, 0.5)
        p = @. LogExpFunctions.logistic(a + bS*S)
        @. H ~ Binomial(10, p)
    end
end
```

Code 15.8 Vanilla simulation: a=0, b=1

```
md"## Code 15.8 Vanilla simulation: a=0, b=1"
```

```
[6, 4, 4, 5, 5, 4, 2, 7, 3, 8, 7, 5, 5, 6, 4, 6, 9, 9, 7, 4,
```

```
begin
    N0 = 100
    S0 = rand(Normal(), N0)
    a0 = 0
    bS0 = 1
    H0 = rand.([BinomialLogit(10, a0+bS0*l) for l in S0]);
end
```

(100)

```
size(H0)
```

	variable	mean	min	median	max
1	:a	-0.0324142	-0.240182	-0.0326535	0.16273
2	:bS	0.905949	0.680666	0.903763	1.16795

```
begin
    Random.seed!(1)
    @time m15_3_ch0 = sample(m15_3(H0, S0),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df0 = DataFrame(m15_3_ch0)
    describe(m15_3_df0)
end
```

Sampling 100%

0.480070 seconds (894.10 k allocations: 100.208 MiB, 10.03% gc time)

- Estimates of a and b are close to the truth.

```
md"
- Estimates of a and b are close to the truth."
```

Code 15.9 Simulate a: H* randomly missing (H randomly eaten by the dog)

```
md"## Code 15.9 Simulate a: H* randomly missing (H
randomly eaten by the dog)"
```

view(::Vector{Union{Missing, Int64}}, [1, 3, 4, 6, 9, 11,

```
begin
    Da = rand(Bernoulli(), N0)
    Hma = Vector{Union{Missing,Int}}(H0)
    Hma[Da .== 1] .= missing;
end
```

[missing, 4, missing, missing, 5, missing, 2, 7, missing,

```
Hma
```

(0.5)

```
params(Bernoulli())
```

```
BitVector: [false, true, false, false, true, false, true,
```

```
.!ismissing.(Hma)
```

15.9.1 Complete data fitting `m15_3`

```
md"### 15.9.1 Complete data fitting `m15_3`"
```

	variable	mean	min	median	max
1	:a	-0.0549241	-0.410573	-0.0556711	0.25872
2	:bS	1.04709	0.73206	1.04013	1.39939

```
begin
    Random.seed!(1)
    index_vec = .!ismissing.(Hma)
    @time m15_3_ch_a =
    sample(m15_3(Hma[index_vec], S0[index_vec]),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_a = DataFrame(m15_3_ch_a)
    describe(m15_3_df_a)
end
```

```
Sampling 100%
0.549874 seconds (1.48 M allocations: 10
0.117 MiB)
```

Code 15.10 Simulate b: Dog only eats Homework of students who study hard (spend less time playing with the dog)

```
md"## Code 15.10 Simulate b: Dog only eats
Homework of students who study hard (spend less
time playing with the dog)"
```

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

```
view(::Vector{Union{Missing, Int64}}, [8, 10, 11, 13, 14,
```

```
begin
    Db = S0 .> 0
    Hmb = Vector{Union{Missing,Int}}(H0)
    Hmb[Db .== 1] .= missing;
end
```

15.10.1 Complete data fitting `m15_3`

- Results are reasonably OK.

```
md"### 15.10.1 Complete data fitting `m15_3`
- Results are reasonably OK."
```

	variable	mean	min	median	max
1	:a	-0.120064	-0.580817	-0.117291	0.410149
2	:bS	0.865742	0.267384	0.866562	1.41655

```
begin
    Random.seed!(1)
    index_vecb = .!ismissing.(Hmb)
    @time m15_3_ch_b =
    sample(m15_3(Hmb[index_vecb], S0[index_vecb]),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_b = DataFrame(m15_3_ch_b)
    describe(m15_3_df_b)
end
```

Sampling 100%

0.654514 seconds (1.67 M allocations: 11
2.330 MiB, 6.12% gc time)

Code 15.11 Simulate c: X (noisy house) impacts Homework quality and Dog homework-eating behavior

```
md"## Code 15.11 Simulate c: X (noisy house)
impacts Homework quality and Dog homework-eating
behavior"
```

view(::Vector{Union{Missing, Int64}}, [5, 12, 29, 31, 51,

```
begin
    Random.seed!(501)
    N2 = 1000
    X = rand(Normal(), N2)
    Sc = rand(Normal(), N2)
    Hc = rand.([BinomialLogit(10, l) for l in 2 .+
    Sc .- 2X])
    Dc = X .> 1
    Hmc = Vector{Union{Missing,Int}}(Hc)
    Hmc[Dc .== 1] .= missing;
end
```

Code 15.12 Use true H to fit `m15_3`

- Estimates are off.

```
md"### Code 15.12 Use true H to fit `m15_3`
- Estimates are off."
```

	variable	mean	min	median	max	nm
1	:a	1.19348	1.12922	1.19236	1.26304	0
2	:bS	0.577602	0.485119	0.57752	0.664904	0

```
begin
    Random.seed!(1)
    @time m15_3_ch_c_use_H = sample(m15_3(Hc, Sc),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_c_use_H = DataFrame(m15_3_ch_c_use_H)
    describe(m15_3_df_c_use_H)
end
```

Sampling 100%

```
4.468720 seconds (3.80 M allocations: 69
9.392 MiB, 2.73% gc time, 69.74% compilation t
ime)
```

15.12.1 Use Hm but complete-data fitting `m15_3`

- Estimates are off too. Esp. estimate a.
- But estimate b improves a bit.

```
md"### 15.12.1 Use Hm but complete-data fitting
`m15_3`
- Estimates are off too. Esp. estimate a.
- But estimate b improves a bit."
```

	variable	mean	min	median	max	nn
1	:a	1.87375	1.76504	1.87376	1.98606	0
2	:bS	0.822737	0.726672	0.823292	0.924049	0

```
begin
    Random.seed!(1)
    index_vecc = .!ismissing.(Hmc)
    @time m15_3_ch_c =
    sample(m15_3(Hmc[index_vecc], Sc[index_vecc]),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_c = DataFrame(m15_3_ch_c)
    describe(m15_3_df_c)
end
```

Sampling 100%

```
5.519539 seconds (17.96 M allocations:
1.078 GiB, 2.67% gc time, 60.06% compilation t
ime)
```

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

Code 15.13. Use H and complete-data fitting `m15_3`

- Estimates almost identical to the ones above

```
md"### Code 15.13. Use H and complete-data fitting
`m15_3`

- Estimates almost identical to the ones above"
```

	variable	mean	min	median	max	nn
1	:a	1.87375	1.76504	1.87376	1.98606	0
2	:bS	0.822737	0.726672	0.823292	0.924049	0

```
begin
    Random.seed!(1)
    @time m15_4_ch_c_use_H_complete =
    sample(m15_3(Hc[Dc .== 0], Sc[Dc .== 0]),
    NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_4_df_c_use_H_complete =
    DataFrame(m15_4_ch_c_use_H_complete)
    describe(m15_4_df_c_use_H_complete)
end
```

Sampling 100%

```
1.316840 seconds (950.11 k allocations:
511.360 MiB, 8.74% gc time, 0.34% compilation
time)
```

Code 15.14 Change simulation c: reverse the missingness

```
md"### Code 15.14 Change simulation c: reverse the
missingness
"
```

```
view(::Vector{Union{Missing, Int64}}, [1, 2, 3, 4, 6, 7, 8
```

```
begin
    Dc2 = abs.(X) .< 1;
    Hmc2 = Vector{Union{Missing,Int}}(Hc)
    Hmc2[Dc2 .== 1] .= missing;
end
```

15.14.1 Use Hmc2 but complete-data fitting `m15_3`

- Removing missing data reduces the estimate of b.

```
md"### 15.14.1 Use Hmc2 but complete-data fitting
`m15_3`
- Removing missing data reduces the estimate of b."
```

	variable	mean	min	median	max	nn
1	:a	0.584132	0.478793	0.58252	0.724042	0
2	:bS	0.384676	0.280504	0.382544	0.500593	0

```
begin
    Random.seed!(1)
    index_vec_c2 = .!ismissing.(Hmc2)
    @time m15_3_ch_c_reverse_missing =
    sample(m15_3(Hmc2[index_vec_c2],
    Sc[index_vec_c2]),
        NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_c_reverse_missing =
    DataFrame(m15_3_ch_c_reverse_missing)
    describe(m15_3_df_c_reverse_missing)
end
```

Sampling 100%

```
  1.077021 seconds (4.28 M allocations: 32
5.958 MiB, 4.76% gc time)
```

Code 15.15 Simulate d: Homework affects dog. Bad homework more likely gets eaten.

```
md"## Code 15.15 Simulate d: Homework affects dog.
Bad homework more likely gets eaten."
```

```
view(::Vector{Union{Missing, Int64}}, [2, 3, 5, 6, 8, 9, 1
```

```
begin
    Sd = rand(Normal(), N0)
    Hd = rand.([BinomialLogit(10, l) for l in Sd])
    Dd = Hd .< 5
    Hmd = Vector{Union{Missing,Int}}(Hd)
    Hmd[Dd .== 1] .= missing;
end
```

15.15.1 Complete-data fitting `m15_3`

```
md"### 15.15.1 Complete-data fitting `m15_3`"
```

	variable	mean	min	median	max	nn
1	:a	0.398245	0.026878	0.396347	0.848112	0
2	:bS	0.782425	0.36436	0.780993	1.19886	0

```
begin
    Random.seed!(1)
    index_vec_d = .!ismissing.(Hmd)
    @time m15_3_ch_d =
    sample(m15_3(Hmd[index_vec_d],
    Sd[index_vec_d]),
        NUTS(100, 0.65, init_ϵ=0.25), 1000)
    m15_3_df_d = DataFrame(m15_3_ch_d)
    describe(m15_3_df_d)
end
```

Sampling 100%

```
  0.590207 seconds (1.68 M allocations: 11
3.420 MiB)
```

Code 15.16 Milk calories ~ Mass + Brain size. Load data and standardize

```
md"## Code 15.16 Milk calories ~ Mass + Brain
size.  Load data and standardize"
```

```
(K = [-0.940041, -0.816126, -1.12591, -1.002, -0.258511,
```

```
begin
    d_milk = DataFrame(CSV.File("data/milk.csv",
    missingstring="NA"))

    # get rid of dots in column names
    rename!(n -> replace(n, "." => "_"), d_milk)

    d_milk.neocortex_prop = d_milk.neocortex_perc
    ./ 100
    d_milk.logmass = log.(d_milk.mass)

    t = Vector{Union{Missing, Float64}}(missing,
    nrow(d_milk))
    present_mask = completecases(d_milk,
    :neocortex_prop)
    t[present_mask] .=
    standardize(ZScoreTransform,
        Vector{Float64}
        (d_milk.neocortex_prop[present_mask]))

    dat_list = (
        K = standardize(ZScoreTransform,
d_milk.kcal_per_g),
        B = t,
        M = standardize(ZScoreTransform,
d_milk.logmass),
    );
end
```

	clade	species	k
1	"Strepsirrhine"	"Eulemur fulvus"	0
2	"Strepsirrhine"	"E macaco"	0
3	"Strepsirrhine"	"E mongoz"	0
4	"Strepsirrhine"	"E rubriventer"	0
5	"Strepsirrhine"	"Lemur catta"	0
6	"New World Monkey"	"Alouatta seniculus"	0
7	"New World Monkey"	"A palliata"	0
8	"New World Monkey"	"Cebus apella"	0
9	"New World Monkey"	"Saimiri boliviensis"	0
10	"New World Monkey"	"S sciureus"	0
11	"New World Monkey"	"Cebuella pygmaea"	0
12	"New World Monkey"	"Callimico goeldii"	0
13	"New World Monkey"	"Callithrix jacchus"	0
14	"New World Monkey"	"Leontopithecus rosalia"	0
15	"Old World Monkey"	"Chlorocebus pygerythrus"	0

```
d_milk
```

17

```
sum(present_mask)
```

```
[-2.0802, missing, missing, missing, missing, -0.508641,
```

```
t
```

Code 15.17 `m15_5` Model imputation and fitting

```
md"### Code 15.17 `m15_5` Model imputation and fitting"
```

	B_impute[10]	B_impute[11]	B_impute[12]	B_impu
1	-1.55351	0.0844085	1.08743	-2.165
2	-2.5169	0.278236	1.1723	-2.229
3	-2.34466	-0.189826	0.420911	-1.094
4	-2.12903	-0.405498	0.328087	-0.924
5	1.06764	-0.880436	1.31149	-0.216
6	-1.31796	3.33127	-0.49526	1.4558
7	-0.224652	2.50136	0.35028	0.3448
8	-0.922883	-2.37955	-0.0131398	-1.029
9	0.0159517	1.75816	0.0486348	-0.2671
10	-1.00052	-1.31987	-1.01234	0.1258
more				
1000	0.1668	-0.840739	0.57879	-1.270

```
begin
    @model function m15_5(K, B, M)
        σ ~ Exponential()
        σ_B ~ Exponential()
        a ~ Normal(0, 0.5)
        ν ~ Normal(0, 0.5)
        bB ~ Normal(0, 0.5)
        bM ~ Normal(0, 0.5)

        N_missing = sum(ismissing.(B))
        B_impute ~ filldist(Normal(ν, σ_B), N_missing)

        i_missing = 1
        for i in eachindex(B)
            if ismissing(B[i])
                #B_impute[i_missing] ~ Normal(ν, σ_B) # this line is bug!
                b = B_impute[i_missing]
                i_missing += 1
            else
                B[i] ~ Normal(ν, σ_B)
                b = B[i]
            end
            μ = a + bB * b + bM * M[i]
            K[i] ~ Normal(μ, σ)
        end
    end

    Random.seed!(1)
    @time m15_5_ch = sample(m15_5(dat_list...), NUTS(), 1000);
    m15_5_df = DataFrame(m15_5_ch);
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.05
```

```
8.173490 seconds (9.36 M allocations: 959.330 MiB, 2.12% gc time, 76.74% compilation time)
```

	variable	mean	min	
1	Symbol("B_impute[10]")	-0.421178	-3.19289	-0
2	Symbol("B_impute[11]")	-0.297335	-3.66384	-0
3	Symbol("B_impute[12]")	0.158509	-3.03178	0.
4	Symbol("B_impute[1]")	-0.574773	-4.84528	-0
5	Symbol("B_impute[2]")	-0.666931	-3.83844	-0
6	Symbol("B_impute[3]")	-0.706487	-4.51215	-0
7	Symbol("B_impute[4]")	-0.275485	-3.07226	-0
8	Symbol("B_impute[5]")	0.522288	-2.87903	0.
9	Symbol("B_impute[6]")	-0.14819	-3.99323	-0
10	Symbol("B_impute[7]")	0.148524	-4.43891	0.
11	Symbol("B_impute[8]")	0.28102	-2.47574	0.
12	Symbol("B_impute[9]")	0.486673	-2.94571	0.
13	:a	0.0213586	-0.688616	0.
14	:bB	0.492542	-0.376103	0.
15	:bM	-0.544161	-1.12879	-0

```
describe(m15_5_df)
```

Code 15.19 `m15_6` Model fitting using only the non-missing values

```
md"### Code 15.19 `m15_6` Model fitting using only the non-missing values"
```

	a	bB	bM	ν	
1	0.37319	0.658656	-0.607679	-0.53481	0.
2	0.16674	0.828162	-0.931949	0.0587369	0.
3	0.0823521	0.541567	-0.638686	-0.0399501	0.
4	-0.0467399	0.697646	-0.713644	-0.115263	0.
5	0.0919513	0.836835	-0.955096	0.00590266	0.
6	0.0864378	0.462202	-0.395563	-0.0157	0.
7	0.31745	0.397549	-0.567001	0.14617	0.
8	-0.147122	0.705595	-0.791945	-0.073959	0.
9	0.372846	0.846506	-0.644569	-0.206733	0.
10	0.252837	0.0665011	-0.434392	0.296508	1.
more					
1000	0.157658	0.848862	-0.982046	0.260027	0.

```julia
begin
    dat_list_obs = (
    K = dat_list.K[present_mask],
    B = Vector{Float64}(dat_list.B[present_mask]),
    M = dat_list.M[present_mask]
)

@model function m15_6(K, B, M)
    σ ~ Exponential()
    σ_B ~ Exponential()
    a ~ Normal(0, 0.5)
    ν ~ Normal(0, 0.5)
    bB ~ Normal(0, 0.5)
    bM ~ Normal(0, 0.5)

    @. B ~ Normal(ν, σ_B)
    μ = @. a + bB * B + bM * M
    @. K ~ Normal(μ, σ)
end

Random.seed!(1)
@time m15_6_ch = sample(m15_6(dat_list_obs...), NUTS(), 1000)
m15_6_df = DataFrame(m15_6_ch);
end
```

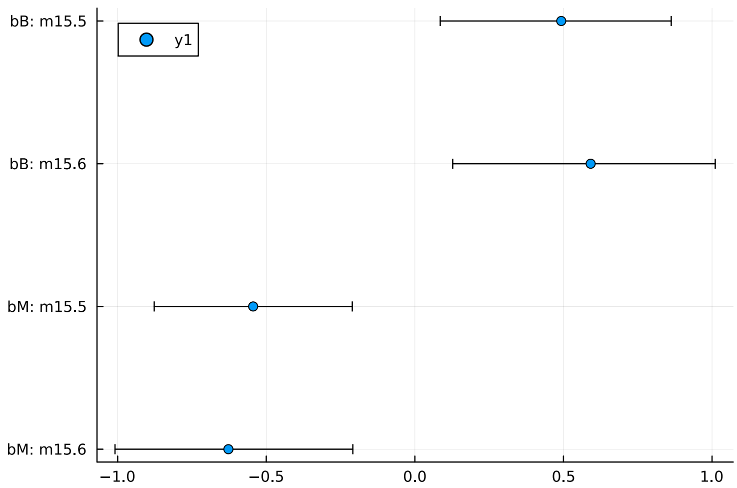
```
Sampling 100%

Found initial step size
ϵ: 0.4

6.575922 seconds (6.97 M allocations: 531.774 MiB, 1.85% gc time, 82.97% compilation time)
```

	variable	mean	min	median	ma
1	:a	0.090448	-0.546451	0.0942606	0.652
2	:bB	0.591536	-0.578625	0.599326	1.351
3	:bM	-0.627407	-1.30115	-0.634604	0.323
4	:ν	-9.05504e-5	-0.921007	-0.00634828	0.719
5	:σ	0.880438	0.486684	0.849687	1.922
6	:σ_B	1.02796	0.620253	1.00107	2.118

```
describe(m15_6_df)
```

Code 15.20 Compare parameter estimates and CI between `m15_5` and `m15_6`

```
md"### Code 15.20 Compare parameter estimates and CI between `m15_5` and `m15_6`"
```

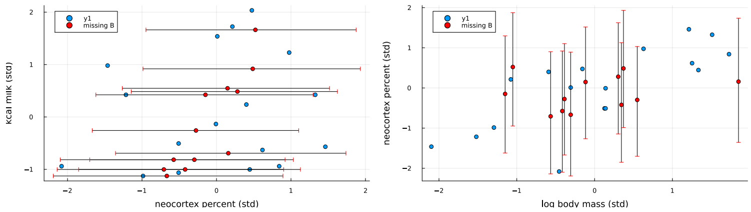



```
coeftab_plot(m15_5_df, m15_6_df; pars=(:bB, :bM), names=("m15.5", "m15.6"))
```

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

Code 15.21 Fig 15.5 Plot the imputed values and its confidence

```
md"### Code 15.21 Fig 15.5 Plot the imputed values and its confidence"
```




```julia
let
    N_missing = sum(ismissing.(dat_list.B))
    miss_mask = ismissing.(dat_list.B)

    B_impute_μ = [
        mean(m15_5_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    B_impute_pi = [
        PI(m15_5_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    err = (
        B_impute_μ .- first.(B_impute_pi),
        last.(B_impute_pi) .- B_impute_μ
    )

    p1 = scatter(dat_list.B, dat_list.K, xlab="neocortex percent (std)", ylab="kcal milk (std)")
    Ki = dat_list.K[miss_mask]
    scatter!(B_impute_μ, Ki, xerr=err, mc=:red, label="missing B")
    #scatter!(B_impute_μ, Ki, xerr=err, ms=0)

    p2 = scatter(dat_list.M, dat_list.B, ylab="neocortex percent (std)", xlab="log body mass (std)")
    Mi = dat_list.M[miss_mask]
    scatter!(Mi, B_impute_μ, yerr=err, mc=:red, label="missing B")
    #scatter!(Mi, B_impute_μ, yerr=err, ms=0)

    plot(p1, p2, size=(1400, 400), margin=5*Plots.mm)
end
```

Code 15.22 `m15_7_1` : add a bivariate normal between two predictors.

```julia
md"### Code 15.22 `m15_7_1`: add a bivariate normal between two predictors."
```

```
m15_7_1 (generic function with 2 methods)
```

```julia
@model function m15_7_1(K, MB, M_missingB)
    σ ~ Exponential()
    σ_BM ~ Exponential()
    a ~ Normal(0, 0.5)
    μB ~ Normal(0, 0.5)
    μM ~ Normal(0, 0.5)
    bB ~ Normal(0, 0.5)
    bM ~ Normal(0, 0.5)
    Rho_BM ~ LKJ(2, 2)

    Σ = (σ_BM .* σ_BM') .* Rho_BM

    # process complete cases
    for i ∈ eachindex(MB)
        MB[i] ~ MvNormal([μM, μB], Σ)
    end

    # impute and process incomplete cases
    N_missing = length(M_missingB)
    #B_impute = Array{Float64}(undef, N_missing) # Note =, not ~. Note Float64, not Real. Vector{..} also works.
    B_impute ~ filldist(Normal(0, 3), N_missing)  # this would cause all estimates to be from the prior.
    #B_impute ~ filldist(Normal(μB, σ_BM), N_missing)  # this would fail to sample.
    MB_missingB = [
        [m, b]
        for (m, b) ∈ zip(M_missingB, B_impute)
    ]

    for i ∈ eachindex(MB_missingB)
        MB_missingB[i] ~ MvNormal([μM, μB], Σ)
    end

    # from both sets, build mean vector for K
    μ = [
        a + bB * b + bM * m
        for (m, b) ∈ Iterators.flatten((MB, MB_missingB))
    ]

    @. K ~ Normal(μ, σ)
end
```

```
[-0.940041, -1.06396, -0.50634, 1.53825, 1.72412, 0.9806
```

```
begin
    # prepare data for sampling

    # to improve stability and performance, need to separate full samples and samples need to be imputed
    pres_mask =  @. !ismissing(dat_list.B)
    _miss_mask = ismissing.(dat_list.B)
    MB = [
        [m, b]
        for (m, b) ∈ zip(dat_list.M[pres_mask], Vector{Float64}(dat_list.B[pres_mask]))
    ]
    M_missingB = dat_list.M[_miss_mask]

    # very important to reorder K values to match order of samples
    KK = vcat(dat_list.K[pres_mask], dat_list.K[_miss_mask])
end
```

```
[-0.415002, -0.307158, -0.565025, -0.387477, -1.05084, -
```

```
M_missingB
```

	B_impute[10]	B_impute[11]	B_impute[12]	B_impu
1	-0.470577	0.757419	3.46814	2.28901
2	0.813646	0.395205	-4.13456	-2.8953
3	5.28155	0.282224	-2.28238	-0.7551
4	-6.22276	-0.358102	2.47944	0.77917
5	-4.3162	-1.77658	-3.10947	2.86125
6	2.58361	-0.557708	0.901645	1.31044
7	-3.03621	0.398519	-0.223246	-1.3126
8	2.87532	-0.180627	0.333789	0.65251
9	-3.77811	0.162254	-0.498088	-0.6210
10	1.05255	-0.121459	1.35053	1.81235
more				
1000	2.47606	-5.37574	1.83822	1.74409

```julia
begin
    Random.seed!(1)
    @time m15_7_1_ch = sample(m15_7_1(KK, MB,
    M_missingB), NUTS(), 1000)
    m15_7_1_df = DataFrame(m15_7_1_ch);
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.2

29.675125 seconds (117.78 M allocations:
17.016 GiB, 9.27% gc time, 31.96% compilation
time)
```

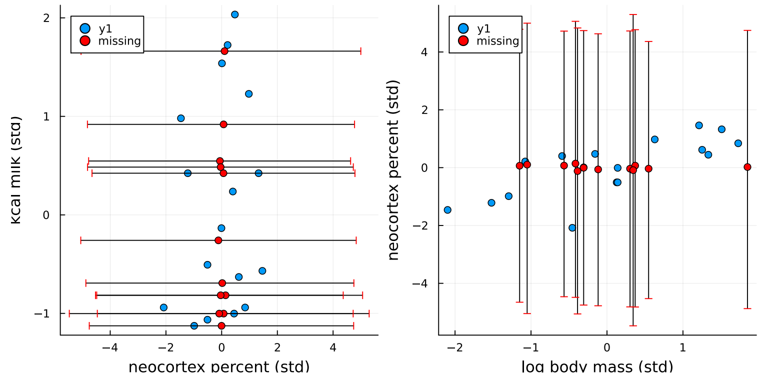
	variable	mean	min
1	Symbol("B_impute[10]")	-0.086222	-10.78
2	Symbol("B_impute[11]")	-0.0347327	-10.04
3	Symbol("B_impute[12]")	0.0215038	-9.089
4	Symbol("B_impute[1]")	0.140579	-9.849
5	Symbol("B_impute[2]")	-0.0038949	-9.790
6	Symbol("B_impute[3]")	0.0713118	-9.462
7	Symbol("B_impute[4]")	-0.117313	-10.41
8	Symbol("B_impute[5]")	0.104244	-10.99
9	Symbol("B_impute[6]")	0.0656626	-9.575
10	Symbol("B_impute[7]")	-0.0615376	-9.230
11	Symbol("B_impute[8]")	-0.0358849	-8.321
12	Symbol("B_impute[9]")	0.0693097	-8.798
13	Symbol("MB_missingB[10][1]")	0.293279	-3.397
14	Symbol("MB_missingB[10][2]")	-0.2171	-3.389
15	Symbol("MB_missingB[11][1]")	0.255512	-4.224

```
describe(m15_7_1_df)
```

Plot `m15_7_1` estimates

- Sampling not working very well. Estimates of missing B hovers around 0.

```
md"### Plot `m15_7_1` estimates
- Sampling not working very well. Estimates of
missing B hovers around 0."
```




```
let
    N_missing = sum(ismissing.(dat_list.B))
    miss_mask = ismissing.(dat_list.B)

    B_impute_μ = [
        #mean(m15_7_2_df[!,"MB_missingB[$i][2]"])
        mean(m15_7_1_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    B_impute_pi = [
        #PI(m15_7_2_df[!,"MB_missingB[$i][2]"])
        PI(m15_7_1_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    err = (
        B_impute_μ .- first.(B_impute_pi),
        last.(B_impute_pi) .- B_impute_μ
    )

    p1 = scatter(dat_list.B, dat_list.K, xlab="neocortex percent (std)", ylab="kcal milk (std)")
    Ki = dat_list.K[miss_mask]
    scatter!(B_impute_μ, Ki, mc=:red, label="missing", xerr=err)


    #scatter!(B_impute_μ, Ki, xerr=err, ms=0)

    p2 = scatter(dat_list.M, dat_list.B, ylab="neocortex percent (std)", xlab="log body mass (std)")
    Mi = dat_list.M[miss_mask]
    scatter!(Mi, B_impute_μ, mc=:red, label="missing", yerr=err)
    #scatter!(Mi, B_impute_μ, yerr=err, ms=0)

    plot(p1, p2, size=(800, 400))
end
```

model `m15_7_2`: B_impute is undef Float64.

- Still buggy. The observed M is regarded as missing as well in this model.
- A better option might be employing two conditional distributions. M|B, and B|M.

```
md"### model `m15_7_2`: B_impute is undef Float64.
- Still buggy. The observed M is regarded as missing as well in this model.
- A better option might be employing two conditional distributions. M|B, and B|M."
```

```
m15_7_2 (generic function with 2 methods)
```

```julia
@model function m15_7_2(K, MB, M_missingB)
    σ ~ Exponential()
    σ_BM ~ Exponential()
    a ~ Normal(0, 0.5)
    μB ~ Normal(0, 0.5)
    μM ~ Normal(0, 0.5)
    bB ~ Normal(0, 0.5)
    bM ~ Normal(0, 0.5)
    Rho_BM ~ LKJ(2, 2)

    Σ = (σ_BM .* σ_BM') .* Rho_BM

    # process complete cases
    for i ∈ eachindex(MB)
        MB[i] ~ MvNormal([μM, μB], Σ)
    end

    # impute and process incomplete cases
    N_missing = length(M_missingB)
    B_impute = Array{Float64}(undef, N_missing)  # Note =, not ~. Note Float64, not Real. Vector{..} also works.
    #B_impute ~ filldist(Normal(), N_missing)   # this would cause all estimates to be from the prior.
    #B_impute ~ filldist(Normal(μB, σ_BM), N_missing)  # this would fail to sample.
    MB_missingB = [
        [m, b]
        for (m, b) ∈ zip(M_missingB, B_impute)
    ]

    for i ∈ eachindex(MB_missingB)
        MB_missingB[i] ~ MvNormal([μM, μB], Σ)
        MB_missingB[i][1] = M_missingB[i] # this would pull the estimated B values closer to the main trend, but not as much as the book.

        #MB_missingB[i] ~ MvNormal([M_missingB[i], μB], Σ) # this didn't improve.

        #MB_missingB[i][1] ~ Normal(μM, σ_BM) # this is wrong. same random variable defined twice.

    end

    # from both sets, build mean vector for K
    μ = [
        a + bB * b + bM * m
        for (m, b) ∈ Iterators.flatten((MB, MB_missingB))
    ]

    @. K ~ Normal(μ, σ)
end
```

	MB_missingB[10][1]	MB_missingB[10][2]	MB_missingB[[1]
1	0.334615	-0.379059	-0.70782
2	-1.32381	-0.846944	0.649774
3	-0.0387845	0.206419	-0.438227
4	-0.527676	0.282668	-0.948367
5	-0.325789	0.951219	1.60756
6	-2.5738	-1.19229	-2.82478
7	-0.450555	0.537884	-0.0785012
8	-1.64276	-1.82066	-0.297684
9	1.36486	1.17069	0.540476
10	-0.7454	-0.262098	0.741061
more			
1000	-1.093	-0.534637	-0.166601

```julia
begin
    Random.seed!(1)

    @time m15_7_2_ch = sample(m15_7_2(KK, MB,
    M_missingB), NUTS(), 1000)
    m15_7_2_df = DataFrame(m15_7_2_ch);
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.05

19.886576 seconds (66.76 M allocations:
8.762 GiB, 8.28% gc time, 44.88% compilation time)
```

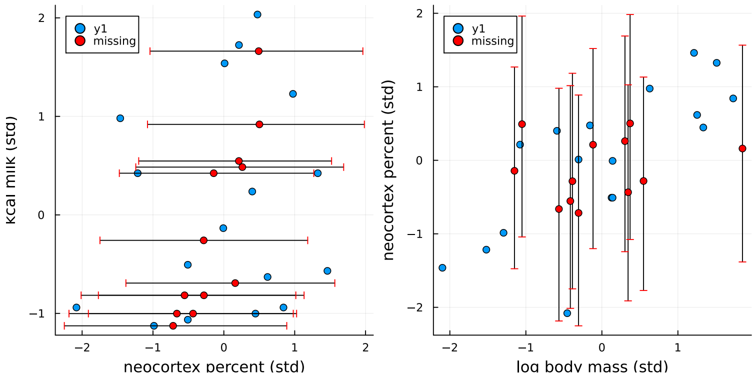
	variable	mean	min
1	Symbol("MB_missingB[10][1]")	-0.238076	-3.5941
2	Symbol("MB_missingB[10][2]")	-0.434491	-3.632(
3	Symbol("MB_missingB[11][1]")	-0.141753	-3.3424
4	Symbol("MB_missingB[11][2]")	-0.28145	-3.1912
5	Symbol("MB_missingB[12][1]")	0.123383	-3.0267
6	Symbol("MB_missingB[12][2]")	0.160404	-3.138(
7	Symbol("MB_missingB[1][1]")	-0.328377	-4.0497
8	Symbol("MB_missingB[1][2]")	-0.554389	-3.4102
9	Symbol("MB_missingB[2][1]")	-0.407122	-3.8943
10	Symbol("MB_missingB[2][2]")	-0.716034	-3.7084
11	Symbol("MB_missingB[3][1]")	-0.381969	-3.7112
12	Symbol("MB_missingB[3][2]")	-0.662566	-3.5352
13	Symbol("MB_missingB[4][1]")	-0.132596	-3.9637
14	Symbol("MB_missingB[4][2]")	-0.284366	-3.6752
15	Symbol("MB_missingB[5][1]")	0.353378	-3.419(

```
describe(m15_7_2_df)
```

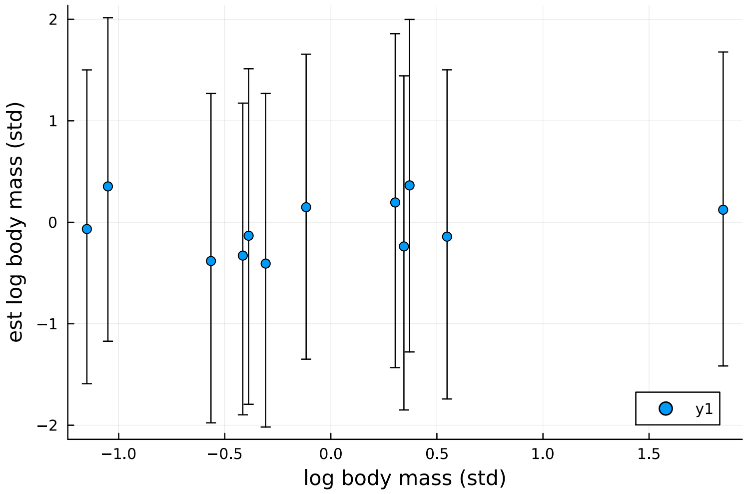
Plot `m15_7_2` estimates

- The Julia model didn't use the observed values for the M variable and instead sampled M as well.
- That results in imputation not working very well. Both estimated M and estimated B hover around 0.

```
md"### Plot `m15_7_2` estimates
- The Julia model didn't use the observed values for the M variable and instead sampled M as well.
- That results in imputation not working very well. Both estimated M and estimated B hover around 0."
```




```julia
let
    N_missing = sum(ismissing.(dat_list.B))
    miss_mask = ismissing.(dat_list.B)

    B_impute_μ = [
        mean(m15_7_2_df[!,"MB_missingB[$i][2]"])
        #mean(m15_7_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    B_impute_pi = [
        PI(m15_7_2_df[!,"MB_missingB[$i][2]"])
        #PI(m15_7_df[!,"B_impute[$i]"])
        for i ∈ 1:N_missing
    ]

    err = (
        B_impute_μ .- first.(B_impute_pi),
        last.(B_impute_pi) .- B_impute_μ
    )

    p1 = scatter(dat_list.B, dat_list.K,
    xlab="neocortex percent (std)", ylab="kcal
    milk (std)")
    Ki = dat_list.K[miss_mask]
    scatter!(B_impute_μ, Ki, xerr=err, mc=:red,
    label="missing")


    #scatter!(B_impute_μ, Ki, xerr=err, ms=0)

    p2 = scatter(dat_list.M, dat_list.B,
    ylab="neocortex percent (std)", xlab="log body
    mass (std)")
    Mi = dat_list.M[miss_mask]
    scatter!(Mi, B_impute_μ, yerr=err, mc=:red,
    label="missing")
    #scatter!(Mi, B_impute_μ, yerr=err, ms=0)

    plot(p1, p2, size=(800, 400))
end
```




```
let
    N_missing = sum(ismissing.(dat_list.B))
    miss_mask = ismissing.(dat_list.B)

    M_impute_μ = [
        mean(m15_7_2_df[!,"MB_missingB[$i][1]"])
        for i ∈ 1:N_missing
    ]

    M_impute_pi = [
        PI(m15_7_2_df[!,"MB_missingB[$i][1]"])
        for i ∈ 1:N_missing
    ]


    err = (
        M_impute_μ .- first.(M_impute_pi),
        last.(M_impute_pi) .- M_impute_μ
    )

    Mi = dat_list.M[miss_mask]
    p2 = scatter(Mi, M_impute_μ, yerr=err,
    ylab="est log body mass (std)", xlab="log body
    mass (std)")
    #scatter!(Mi, M_impute_μ, yerr=err, ms=0)

    p2
end
```

ToDo: split the bivariate normal into two univariate conditional normal

```
md"### ToDo: split the bivariate normal into two
univariate conditional normal"
```

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

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

Code 15.23 Obtain index of data with missing B (Brain/Neocortex size)

```
md"## Code 15.23 Obtain index of data with missing B (Brain/Neocortex size)"
```

```
BitVector: [false, true, true, true, true, false, false,
```

```
ismissing.(dat_list.B)
```

Code 15.24 Load the Gods dataset

```
md"## Code 15.24 Load the Gods dataset"
```

	variable	mean	min	me
1	:polity	nothing	"Big Island Hawaii"	not
2	:year	-1339.35	-9600	-60
3	:population	4.86246	1.40832	4.7
4	:moralizing_gods	0.949405	0	1.0
5	:writing	0.459491	0	0.0

```
begin
    d_gods = DataFrame(CSV.File("data/Moralizing_gods.csv", missingstring="NA"))
    describe(d_gods)
end
```

	polity	year	population	moralizing_g
1	"Big Island Hawaii"	1000	3.72964	missing
2	"Big Island Hawaii"	1100	3.72964	missing
3	"Big Island Hawaii"	1200	3.59834	missing
4	"Big Island Hawaii"	1300	4.02624	missing

```
first(d_gods, 4)
```

Code 15.25 Count rows with different moralizing_gods

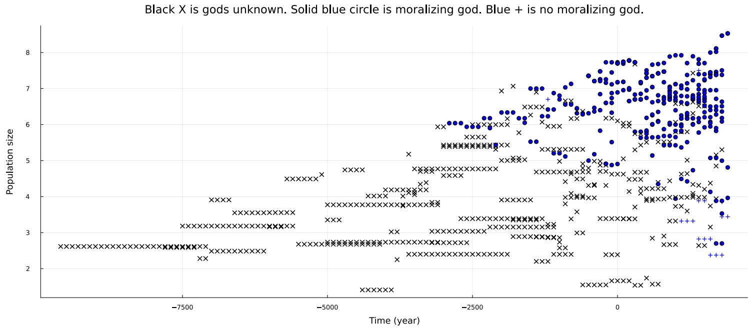
```
md"## Code 15.25 Count rows with different moralizing_gods"
```

```
Dict(0 ⇒ 17, missing ⇒ 528, 1 ⇒ 319)
```

```
countmap(d_gods.moralizing_gods)
```

Code 15.26 Fig 15.7 Plot pop vs time

```
md"## Code 15.26 Fig 15.7 Plot pop vs time"
```




```
let
    symbol = map(g -> ismissing(g) ? :x : (g == 1 ? :circle : :+), d_gods.moralizing_gods)
    color = map(g -> ismissing(g) ? :black : :blue, d_gods.moralizing_gods)
    scatter(d_gods.year, d_gods.population, m=symbol, mc=color,
        xlab="Time (year)", ylab="Population size", title="Black X is gods unknown. Solid blue circle is moralizing god. Blue + is no moralizing god.", legend=false,
        size=(1400,600), margin=5*Plots.mm)
end
```

Code 15.27

- Check how god missingness is associated with writing/literacy.

```
md"## Code 15.27
- Check how god missingness is associated with writing/literacy."
```

```
Dict((0, 0) ⇒ 16, (1, 1) ⇒ 310, (missing, 1) ⇒ 86, (mis
```

```
countmap(zip(d_gods.moralizing_gods, d_gods.writing))
```

Code 15.28 Check how moralizing_gods varies over years for Hawaii

```
md"## Code 15.28 Check how moralizing_gods varies over years for Hawaii"
```

	year	writing	moralizing_gods
1	1000	0	missing
2	1100	0	missing
3	1200	0	missing
4	1300	0	missing
5	1400	0	missing
6	1500	0	missing
7	1600	0	missing
8	1700	0	missing
9	1800	0	1

```
d_gods[d_gods.polity .== "Big Island Hawaii", ["year", "writing", "moralizing_gods"]]
```

15.3 Categorical errors and discrete absences

```
md"# 15.3 Categorical errors and discrete absences"
```

Code 15.29 Simulate data

- parameter `r` (the rate of missing) has a large impact on accuracy of the β estimate (less so on the α estimate). The higher it is, the less accurate the β estimate is.

```
md"## Code 15.29 Simulate data
- parameter `r` (the rate of missing) has a large impact on accuracy of the β estimate (less so on the α estimate). The higher it is, the less accurate the β estimate is.

"
```

`simulate_missing_cat_data`

- cat_probability/k: probability that there is a cat in a house.
- missing*rate/r: The probability if the presence of cat in a house is unknown. The higher the missing*rate is, the less accurate the β estimate is.

```julia
"""
- cat_probability/k: probability that there is a cat in a house.
- missing_rate/r: The probability if the presence of cat in a house is unknown. The higher the missing_rate is, the less accurate the β estimate is.

"""
function simulate_missing_cat_data(N_houses::Int; α=5, β=-2, cat_probability=0.5, missing_rate=0.2)
    Random.seed!(9)


    cat = rand(Bernoulli(cat_probability), N_houses)
    # music_notes is the number of notes that the songbird in the house will sing.
    music_notes = rand.([Poisson(exp(α + β * c)) for c ∈ cat])   # wrongly omitted exp() before.

    R_C = rand(Bernoulli(missing_rate), N_houses)

    cat_obs = Vector{Int}(cat)
    cat_obs[R_C] .= -9 # -9 means unknown/missing .

    dat = (
        notes = music_notes,
        cat = cat_obs,
        RC = R_C,
        N = N_houses,
    )
end
```

Code 15.30 `m15_8`

```julia
md"## Code 15.30 `m15_8`"
```

```
m15_8 (generic function with 2 methods)
```

```
@model function m15_8(notes, cat, RC, N)
    α ~ Normal(0, 2)
    β ~ Normal(0, 2) #Uniform(-10, 10) does not help.
    k ~ Beta(2, 2)
    λ = @. exp(α + β * cat) # was logistic() in the original code.

    for i ∈ eachindex(cat)
        if !RC[i]  # Cat is not missing. RC[i]==0.
            cat[i] ~ Bernoulli(k)
            notes[i] ~ Poisson(λ[i])
            #Turing.@addlogprob! poislogpdf(λ[i], notes[i])   #equivalent to above ~.
        else
            Turing.@addlogprob! log(k) + poislogpdf(exp(α+β), notes[i])
            Turing.@addlogprob! log(1-k) + poislogpdf(exp(α), notes[i])
        end
    end
end
```

	variable	mean	min	median	max	nn
1	:k	0.499485	0.449953	0.499428	0.551044	0
2	:α	4.83313	4.82139	4.83313	4.84394	0
3	:β	-1.0567	-1.08047	-1.05679	-1.03105	0

```
begin
    dat1 = simulate_missing_cat_data(1000, cat_probability=0.5, missing_rate=0.2)
    @time m15_8_df1 = DataFrame(sample(m15_8(dat1...), NUTS(), 2000))
    describe(m15_8_df1)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.0015625

3.894634 seconds (3.15 M allocations: 818.502 MiB, 3.86% gc time)
```

	variable	mean	min	median	max	nn
1	:k	0.499189	0.449848	0.499347	0.544746	0
2	:α	4.99034	4.97391	4.99036	5.00251	0
3	:β	-1.89121	-1.92846	-1.89121	-1.85348	0

```
begin
    dat2 = simulate_missing_cat_data(1000, cat_probability=0.5, missing_rate=0.01)
    @time m15_8_df2 = DataFrame(sample(m15_8(dat2...), NUTS(), 2000))
    describe(m15_8_df2)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.0015625

  3.364698 seconds (3.08 M allocations: 789.633 MiB, 3.29% gc time)
```

	variable	mean	min	median	max	nn
1	:k	0.498575	0.443938	0.4985	0.548115	0
2	:α	4.99894	4.98674	4.99893	5.01077	0
3	:β	-1.96582	-2.00007	-1.96591	-1.93136	0

```
let
    dat_tmp = simulate_missing_cat_data(1000, cat_probability=0.5, missing_rate=0.001)
    @time m15_8_df_tmp = DataFrame(sample(m15_8(dat_tmp...), NUTS(), 2000))
    describe(m15_8_df_tmp)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.0015625

  2.990766 seconds (2.84 M allocations: 691.123 MiB, 4.70% gc time)
```

	variable	mean	min	median	ma
1	:k	0.49838	0.454194	0.498494	0.5360
2	:α	4.45526	4.44385	4.45516	4.4668
3	:β	-0.0393097	-0.0566165	-0.0393168	-0.022

```
let
    dat_tmp = simulate_missing_cat_data(1000,
    cat_probability=0.5, missing_rate=0.95)
    @time m15_8_df_tmp =
    DataFrame(sample(m15_8(dat_tmp...), NUTS(),
    2000))
    describe(m15_8_df_tmp)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.00078125

5.522623 seconds (3.31 M allocations: 88
6.095 MiB, 3.15% gc time)
```

	variable	mean	min	median	max
1	:k	0.504992	0.469798	0.504802	0.54660
2	:α	3.87201	3.85381	3.87204	3.88848
3	:β	-0.0546333	-0.074901	-0.0546454	-0.0349

```
let
    dat_tmp = simulate_missing_cat_data(1000, α=5,
    cat_probability=0.8, missing_rate=0.95)
    @time m15_8_df_tmp =
    DataFrame(sample(m15_8(dat_tmp...), NUTS(),
    2000))
    describe(m15_8_df_tmp)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.0015625

5.461961 seconds (3.30 M allocations: 88
1.118 MiB, 2.83% gc time)
```

	variable	mean	min	median	max	nn
1	:k	0.765602	0.724678	0.765707	0.810171	0
2	:α	4.86521	4.84156	4.86536	4.88382	0
3	:β	-1.74136	-1.77071	-1.74155	-1.70768	0

```
let
    dat_tmp = simulate_missing_cat_data(1000, α=5,
    cat_probability=0.8, missing_rate=0.05)
    @time m15_8_df_tmp =
    DataFrame(sample(m15_8(dat_tmp...), NUTS(),
    2000))
    describe(m15_8_df_tmp)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.003125

 3.534634 seconds (3.17 M allocations: 82
4.193 MiB, 4.28% gc time)
```

	variable	mean	min	median	max	nn
1	:k	0.782544	0.743269	0.783165	0.833083	0
2	:α	4.96041	4.93928	4.9605	4.98241	0
3	:β	-1.92514	-1.96068	-1.92529	-1.88983	0

```
let
    dat_tmp = simulate_missing_cat_data(1000, α=5,
    cat_probability=0.8, missing_rate=0.01)
    @time m15_8_df_tmp =
    DataFrame(sample(m15_8(dat_tmp...), NUTS(),
    2000))
    describe(m15_8_df_tmp)
end
```

```
Sampling 100%

Found initial step size
ϵ: 0.003125

 3.637285 seconds (3.29 M allocations: 87
3.586 MiB, 4.55% gc time)
```