See chapter 3 in Regression and Other Stories.

Widen the notebook

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

```
using Pkg ✓ , DrWatson ✓
```

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

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

    # Graphics related
    using GLMakie ✓

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

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

3.1 - Weighted averages

pop =

	stratum	country	population	avera
1	1	"United States"	310000000	36.8
2	2	"Mexico"	112000000	26.7
3	3	"Canada"	34000000	40.7

```
pop = DataFrame(stratum=1:3, country=
["United States", "Mexico", "Canada"],
    population=Int[310e6, 112e6, 34e6],
    average_age=[36.8, 26.7, 40.7])
```

34.61008771929824

```
mean(pop.average_age,
weights(pop.population))
```

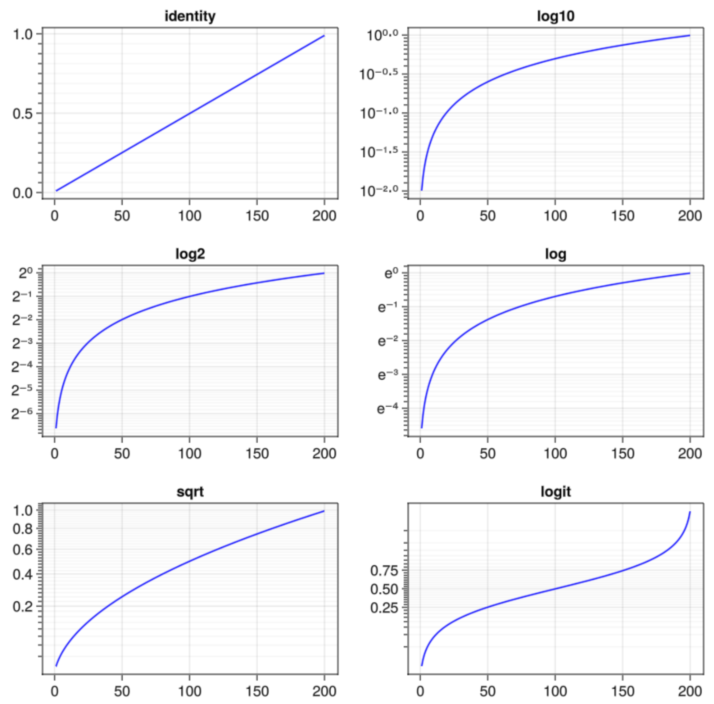
[0.679825, 0.245614, 0.0745614]

```
weights(pop.population)/sum(pop.population)
```

	variable	mean	min	median	
1	:stratum	2.0	1	2.0	3
2	:country	nothing	"Canada"	nothing	"
3	:population	1.52e8	34000000	1.12e8	3
4	:average_age	34.7333	26.7	36.8	4

```
describe(pop)
```

3.3 - Graphing a line



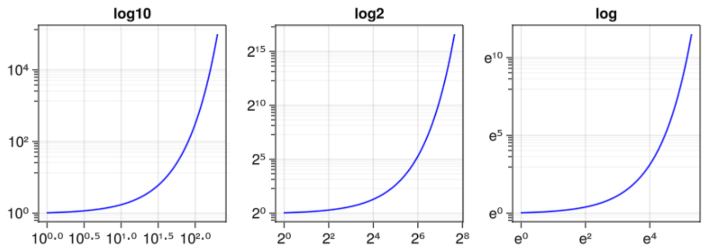

```
let
    data = LinRange(0.01, 0.99, 200)
    f = Figure(resolution = (800, 800))

    for (i, scale) in enumerate([identity, log10, log2, log, sqrt, Makie.logit])

        row, col = fldmod1(i, 2)
        Axis(f[row, col], yscale = scale, title = string(scale),
            yminorticksvisible = true, yminorgridvisible = true,
            yminorticks = IntervalsBetween(8))

        lines!(data, color = :blue)
    end

    f
end
```



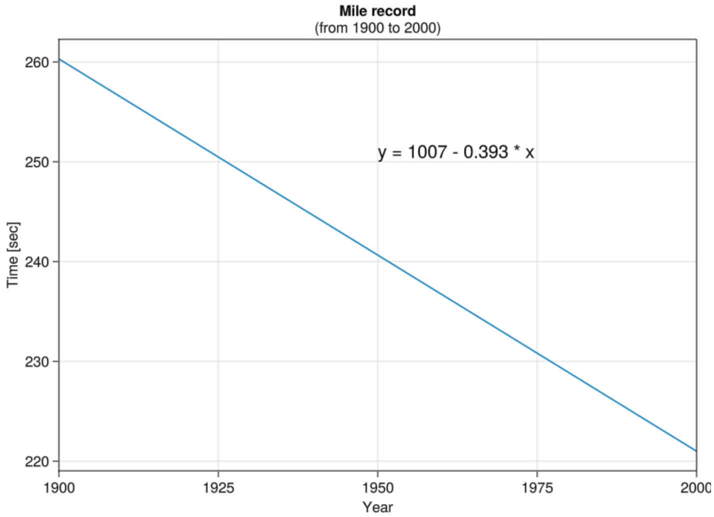

```
let
    data = 10 .^ LinRange(0.01, 5.0, 200)
    f = Figure(resolution = (800, 300))

    for (i, scale) in enumerate([log10,
    log2, log])

        row, col = fldmod1(i, 2)
        Axis(f[1, i], yscale = scale, xscale
     = scale, title = string(scale),
            yminorticksvisible = true,
    yminorgridvisible = true,
            yminorticks =
    IntervalsBetween(8))

        lines!(data, color = :blue)
    end

    f
end
```




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title = "Mile
    record", subtitle = "(from 1900 to
    2000)", xlabel = "Year", ylabel = "Time
    [sec]")
    xlims!(ax, 1900, 2000)
    ax.xticks = 1900:25:2000
    x = 1900:2000
    y = 1007 .- 0.393 .* x
    lines!(x, y)
    annotations!("y = 1007 - 0.393 * x",
    position=(1950, 250))
    current_figure()
end
```

3.4 - Log and exponential scales

Simulated data for metabolic.

	body_mass	rate
1	3.95141	4.30636
2	4.80828	4.84668
3	5.15223	5.2515
4	5.7138	5.69544
5	5.96146	5.89688
6	5.97363	5.85267
7	6.04036	5.8827
8	6.23519	6.06916
9	6.25148	6.05207
10	6.34917	6.06364
⋮ more		
200	9.19981	8.21907

```
begin
    x = sort(rand(Uniform(0.01, 10000), 200))
    y = 4.1 * x.^0.74 .+ [rand.(Normal.(0, sqrt(x[i])), 1)[1] for i in 1:length(x)]
    metabolic = DataFrame(:body_mass => log.(x), :rate => log.(y))
end
```

```
ppl3_1 (generic function with 2 methods)
```

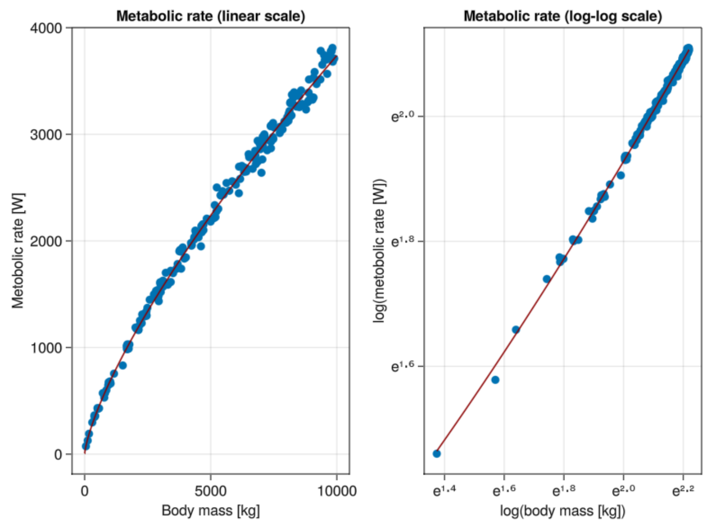
```
@model function ppl3_1(m, r)
    a ~ Normal(0, 0.3)
    b ~ Normal(0, 0.3)
    σ ~ Exponential(1)
    μ = a .+ b .* m
    for i in eachindex(y)
        r[i] ~ Normal(μ[i], σ)
    end
end
```

	parameters	mean	std	naiv
1	:a	1.39357	0.0208158	0.0003
2	:b	0.742259	0.00248751	3.9331
3	:σ	0.0312047	0.00164332	2.5983

```
begin
    m3_1t = ppl3_1(metabolic.body_mass,
    metabolic.rate)
    chns3_1t = sample(m3_1t, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns3_1t)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	1.393	0.02	1.394	0.02
2	"b"	0.742	0.002	0.742	0.00
3	"σ"	0.031	0.002	0.031	0.00

```
begin
    post3_1t = DataFrame(chns3_1t)[:, 3:5]
    ms3_1t = model_summary(post3_1t,
    names(post3_1t))
end
```




```
let
    x = LinRange(1, 10000, 1000)
    y = 4.1 * x.^0.74
    f = Figure()
    ax = Axis(f[1, 1]; title="Metabolic rate (linear scale)", xlabel="Body mass [kg]", ylabel="Metobolic rate [W]")
    scatter!(exp.(metabolic.body_mass), exp.(metabolic.rate))
    lines!(x, y; color=:darkred)

    ax = Axis(f[1 , 2]; title="Metabolic rate (log-log scale)", xscale=log, yscale=log,
        xlabel="log(body mass [kg])",
        ylabel="log(metobolic rate [W])")
    x = LinRange(minimum(metabolic.body_mass), maximum(metabolic.body_mass), 100)
    scatter!(metabolic.body_mass, metabolic.rate)
    lines!(x, ms3_1t["a", "median"] .+ ms3_1t["b", "median"] * x; color=:darkred)
    current_figure()
end
```

```
100-element LinRange{Float64, Int64}:
 3.95141, 4.00443, 4.05744, 4.11045, 4.16347, …,
```

```
LinRange(minimum(metabolic.body_mass), maximum(metabolic.body_mass), 100)
```

```
57.69669926958961
```

```
exp(exp(1.4))
```

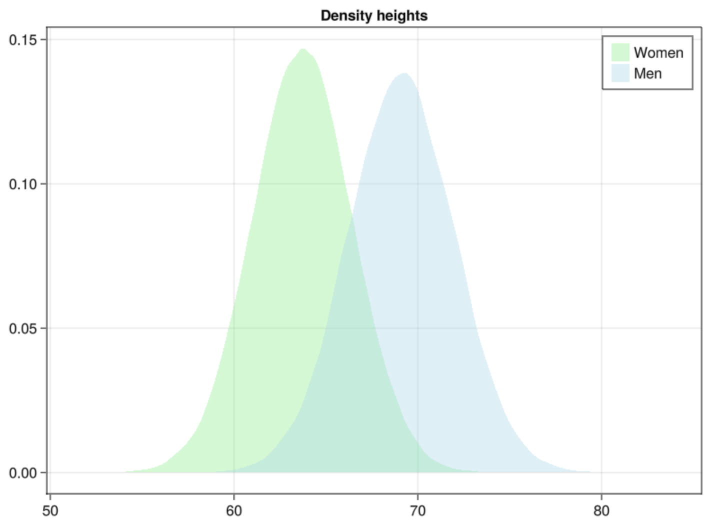
3.5 - Probability distributions

	height	sex
1	67.7877	"female"
2	63.9182	"female"
3	66.0975	"female"
4	66.752	"female"
5	61.0871	"female"
6	68.1143	"female"
7	61.1092	"female"
8	62.9689	"female"
9	64.4691	"female"
10	61.2035	"female"
⋮ more		
200000	67.0396	"male"

```
begin
    N = 100000
    heights = DataFrame()
    height = vcat(rand(Normal(63.7, 2.7),
    N),
        rand(Normal(69.1, 2.9), N))
    sex = repeat(["female", "male"],
    inner=N)
    heights.height = height
    heights.sex = sex
    heights
end
```

▸ (mean = 63.7107, var = 7.34652, std = 2.71045,

```
begin
    menHeights = heights[heights.sex .==
    "male", :height]
    womenHeights = heights[heights.sex .==
    "female", :height]
    (mean=mean(womenHeights),
    var=var(womenHeights),
        std=std(womenHeights),
        median=median(womenHeights),
        mad_sd=mad(womenHeights))
end
```




```
let
        f = Figure()
        ax = Axis(f[1, 1]; title="Density
        heights")
        density!(womenHeights; color=color
        = (:lightgreen, 0.4), label="Women")
        density!(menHeights; color=color =
        (:lightblue, 0.4), label="Men")
        axislegend()
        f
    end
```

0.49714220980937984

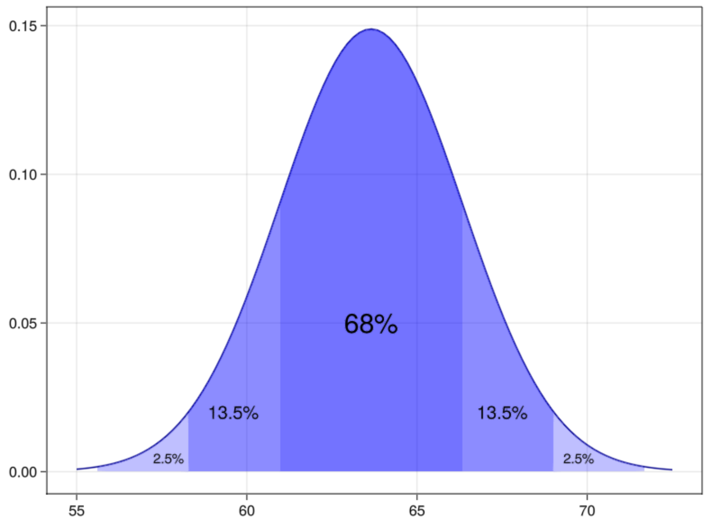
```
begin
    wdf = Normal(63.65, 2.68)
    cdf(wdf, 63.65 + 0.67 * 2.68) -
    cdf(wdf, 63.65 - 0.67 * 2.68)
end
```

0.6826894921370859

```
cdf(wdf, 63.65 + 2.68) - cdf(wdf, 63.65 -
2.68)
```

0.9544997361036417

```
cdf(wdf, 63.65+2*2.68) - cdf(wdf, 63.65-
2*2.68)
```




```
let
    wdf = Normal(63.65, 2.68)
    x = range(55.0, 72.5 ; length=100)
    lines(x, pdf.(wdf, x); color=:darkblue)

    x1 = range(63.65 - 3 * 2.68, 63.65 - 2
    * 2.68; length=20)
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.25), label =
        "Label")

    x1 = range(63.65 + 2 * 2.68, 63.65 + 3
    * 2.68; length=20)
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.25), label =
        "Label")

    x1 = range(63.65 - 2 * 2.68, 63.65 - 1
    * 2.68; length=20)
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.45), label =
        "Label")

    x1 = range(63.65 + 1 * 2.68, 63.65 + 2
    * 2.68; length=20)
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.45), label =
        "Label")

    x1 = range(63.65 - 1 * 2.68, 63.65;
    length=20)

```

```
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.55), label =
        "Label")

    x1 = range(63.65, 63.65 + 2.68;
    length=20)
    band!(x1, fill(0, length(x1)), pdf.
    (wdf, x1);
        color = (:blue, 0.55), label =
        "Label")

    text!("68%", position = (63.65, 0.05),
    align = (:center,  :center),
        textsize = 30)
    text!("13.5%", position = (67.5, 0.02),
    align = (:center,  :center),
        textsize = 20)
    text!("13.5%", position = (59.6, 0.02),
    align = (:center,  :center),
        textsize = 20)
    text!("2.5%", position = (69.75,
    0.0045), align = (:center,  :center),
        textsize = 15)
    text!("2.5%", position = (57.7,
    0.0045), align = (:center,  :center),
        textsize = 15)
    current_figure()
end
```

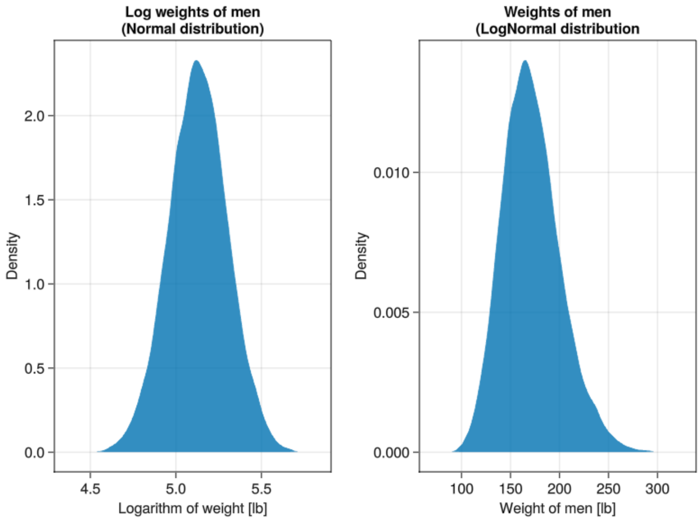
```
▶ (m̂ = 6.0014, m = 6.0, σ̂ = 2.03902, σ = 2.04939)
```

```
let
    n = 20; p = 0.3
    y = rand(Binomial(n, p), 10000)
    (m̂ = mean(y), m = 20 * 0.3, σ̂ = std(y),
    σ = sqrt(n * p * (1 - p)))
end
```

```
▶ (m̂ = 0.2993, m = 0.3)
```

```
let
    n = 20; p = 0.3
    y = rand(Bernoulli(p), 10000)
    (m̂ = mean(y), m = 0.3)
end
```

LogNormal




```
let
    menw = rand(LogNormal(5.13, 0.17),
    10000)
    menwl = log.(menw)
    f = Figure()
    ax = Axis(f[1, 1]; title="Log weights
    of men\n(Normal distribution)",
    xlabel="Logarithm of weight [lb]",
    ylabel="Density")
    density!(menwl)
    ax = Axis(f[1, 2]; title="Weights of
    men\n(LogNormal distribution",
    xlabel="Weight of men [lb]",
    ylabel="Density")
    density!(menw)
    current_figure()
end
```

Binomial

	parameters	median	mad_sd	mean	st
1	"bv"	6.0	1.483	5.934	1.96

```
let
    df = DataFrame(bv = rand(Binomial(20,
    0.3), 1000))
    model_summary(df, [:bv])
end
```

```
(mean = 6.0, std = 2.04939)
```

```julia
begin
    n = 20
    p = 0.3
    (mean = n * p, std = √(n * p * (1 - p)))
end
```

Poisson

```
[4, 4, 5, 3, 6, 4, 5, 6, 4, 8]
```

```julia
rand(Poisson(4.52), 10)
```

3.6 - Probability modeling

```
20689.577579211084
```

```julia
1 / (pdf(Normal(0.49, 0.04), 0.5) / 200000)
```

```
20.689577579211083
```

```julia
1 / (1000pdf(Normal(0.49, 0.04), 0.5) /
200000)
```