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 ✓

    # Graphics related
    using GLMakie ✓

    # Specific to ROSStanPluto
    using StanSample ✓

    # Common data files and functions
    using RegressionAndOtherStories ✓
end
```

3.1 - Weighted averages

pop =

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

```julia
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

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

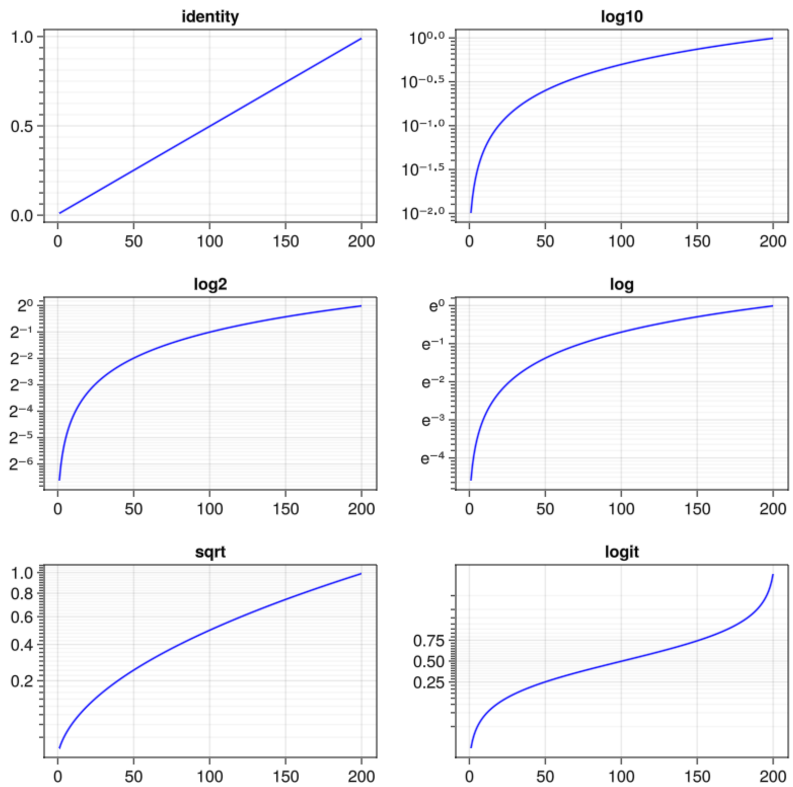
▸ [0.679825, 0.245614, 0.0745614]

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

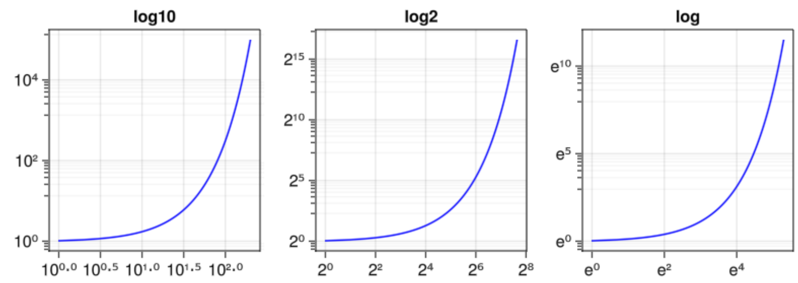
	variable	mean	min	median
1	:stratum	2.0	1	2.0
2	:country	nothing	"Canada"	nothing
3	:population	1.52e8	34000000	1.12e8
4	:average_age	34.7333	26.7	36.8

```julia
describe(pop)
```

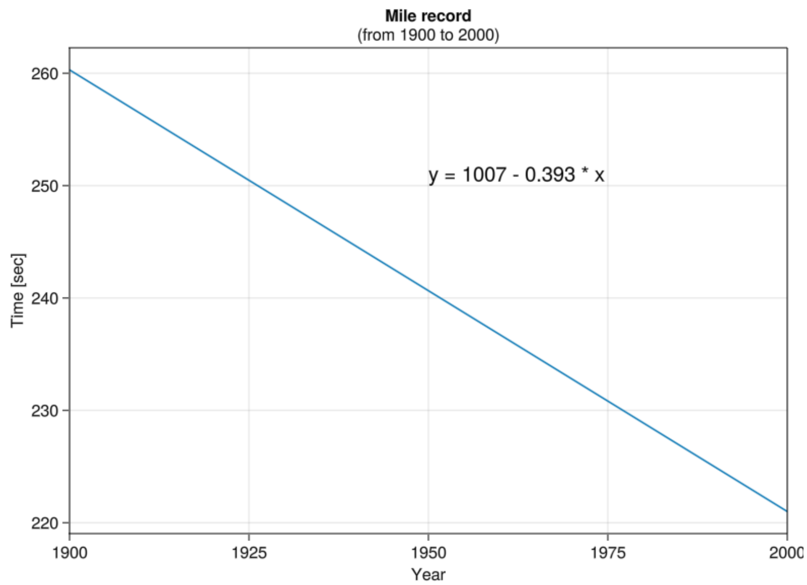
3.3 - Graphing a line




```
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
```




```julia
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	2.98408	3.76147
2	4.08312	4.44374
3	4.16122	4.48304
4	4.47076	4.77568
5	4.60702	4.89275
6	4.72867	4.89577
7	5.41953	5.48691
8	5.85765	5.76041
9	5.93956	5.77697
10	5.9717	5.88113
⋮ more		
200	9.1991	8.22749

```
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
```

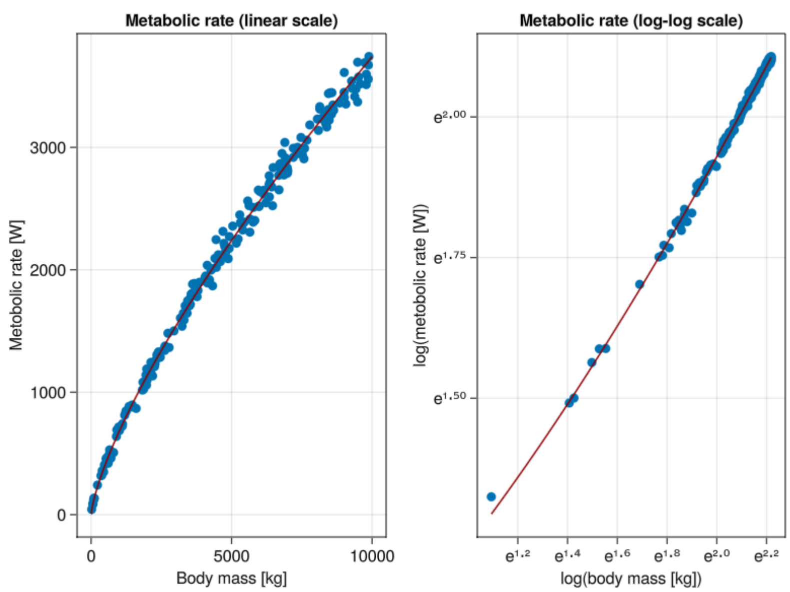
```
stan3_1 = "
data {
    int N;
    vector[N] m;
    vector[N] r;
}
parameters {
    real a;
    real b;
    real sigma;
}
model {
    vector[N] mu;
    a ~ normal(0, 0.3);
    b ~ normal(0, 0.3);
    sigma ~ exponential(1);
    mu = a + b * m;
    r ~ normal(mu, sigma);
}";
```

	parameters	mean	mcse	std
1	"a"	1.45414	0.000701633	0.02043
2	"b"	0.734723	8.56162e-5	0.00249
3	"sigma"	0.0384015	4.3462e-5	0.00189

```
let
    data = (N =
    length(metabolic.body_mass), m =
    metabolic.body_mass, r =
    metabolic.rate)
    global m3_1s = SampleModel("m3.1s",
    stan3_1)
    global rc3_1s = stan_sample(m3_1s;
    data)
    success(rc3_1s) && describe(m3_1s)
end
```

	parameters	median	mad_sd	mean	std
1	"a"	1.455	0.02	1.454	0.02
2	"b"	0.735	0.002	0.735	0.00
3	"sigma"	0.038	0.002	0.038	0.00

```
if success(rc3_1s)
    post3_1s = read_samples(m3_1s,
    :dataframe)
    ms3_1s = model_summary(post3_1s, [:a,
    :b, :sigma])
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_1s[:a, :mean] .+
    ms3_1s[:b, :mean] * x; color=:darkred)
    current_figure()
end
```

```
100-element LinRange{Float64, Int64}:
 2.98408, 3.04686, 3.10964, 3.17241, 3.23519, …
```

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

```
57.69669926958961
```

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

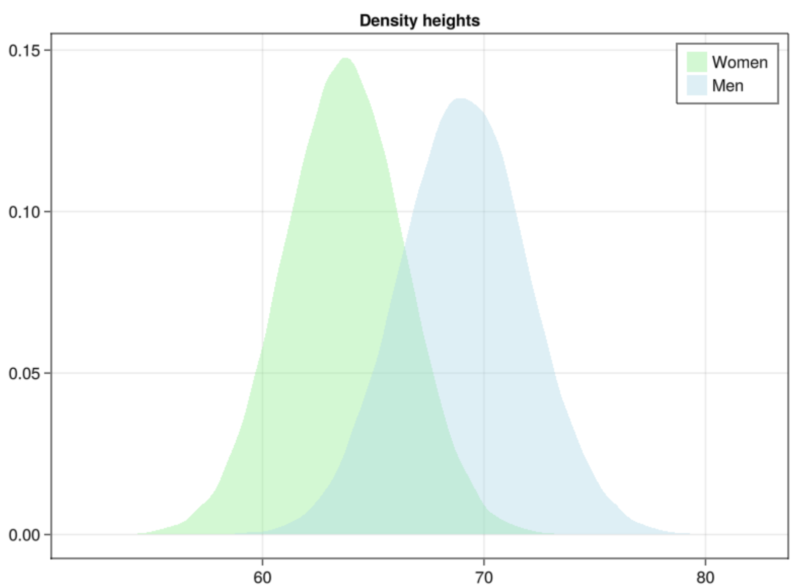
3.5 - Probability distributions

	height	sex
1	61.0116	"female"
2	63.1106	"female"
3	60.6683	"female"
4	62.4183	"female"
5	65.9579	"female"
6	64.6202	"female"
7	66.4275	"female"
8	61.6656	"female"
9	62.6349	"female"
10	66.2756	"female"
⋮ more		
200000	65.4376	"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 = 63.6951, var = 7.31388, std = 2.70442
```

```
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

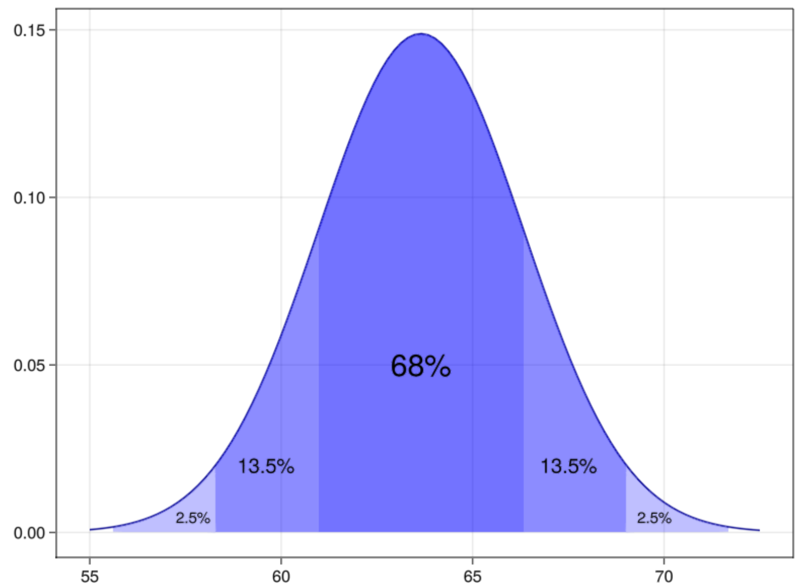
```
begin
    wdf = Normal(63.65, 2.68)
    cdf(wdf, 63.65 + 0.67 * 2.68) -
    cdf(wdf, 63.65 - 0.67 * 2.68)
end
```

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)
```

```
* 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
```

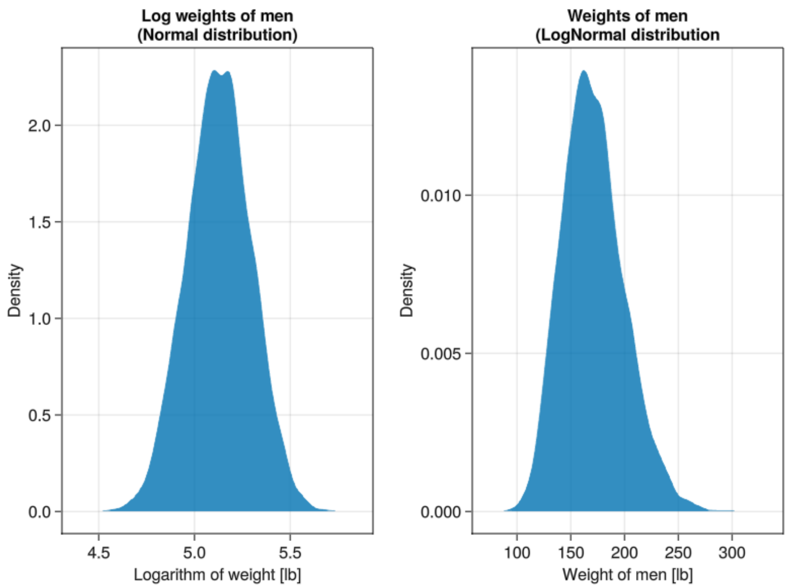
```
(m̂ = 6.0422, m = 6.0, σ̂ = 2.06428, σ = 2.0493
```

```
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 = 0.3053, m = 0.3)
```

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

LogNormal




```julia
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	std
1	"bv"	6.0	1.483	6.099	2.03

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

(mean = 6.0, std = 2.04939)

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

Poisson

[3, 3, 6, 2, 1, 3, 3, 4, 6, 7]

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

3.6 - Probability modeling

20689.577579211084

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

20.689577579211083

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