In `Regression and Other Stories`, mcmc is *just* a tool. Hence whether one uses Stan or Turing is not the main focus of the book. This notebook uses `ElectionsEconomy: hibbs.csv` to illustrate how Stan and other tools are used in the Julia *project* ROSStanPluto.jl.

Over time I will expand below the list of topics:

1. Stan (StanSample.jl, ...)
2. Using median and mad to summarize a posterior distribution.
3. ...
4. Model comparison (TBD)
5. DAGs (TBD)
6. Graphs (TBD)
7. ...

See Chapter 1.2, Figure 1.1 in Regression and Other Stories.

Widen the cells.

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

A typical set of Julia packages to include in notebooks.

```
using Pkg ✓ , DrWatson ✓
```

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

    # Specific to ROSStanPluto
    using StanSample ✓

    # Graphics related
    using GLMakie ✓

    # Include basic packages
    using RegressionAndOtherStories ✓
end
```

```
Replacing docs for `RegressionAndOtherStories.tr
DataFrame, AbstractString}` in module `Regressic
```

Note

All data files are available (as .csv files) in the data subdirectory of package RegressionAndOtherStories.jl.

```
"/Users/rob/.julia/packages/RegressionAndOtherSto
ros_datadir()
```

hibbs =

	year	growth	vote	inc_party_candidat
1	1952	2.4	44.6	"Stevenson"
2	1956	2.89	57.76	"Eisenhower"
3	1960	0.85	49.91	"Nixon"
4	1964	4.21	61.34	"Johnson"
5	1968	3.02	49.6	"Humphrey"
6	1972	3.62	61.79	"Nixon"
7	1976	1.08	48.95	"Ford"
8	1980	-0.39	44.7	"Carter"
9	1984	3.86	59.17	"Reagan"
10	1988	2.27	53.94	"Bush, Sr."
⋮	more			
16	2012	0.95	52.0	"Obama"

```
hibbs =
CSV.read(ros_datadir("ElectionsEconomy",
"hibbs.csv"), DataFrame)
```

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

vote ~ 1 + growth

Coefficients:
─────────────────────────────────────────────
                 Coef.  Std. Error      t  Pr(>|t
─────────────────────────────────────────────
(Intercept)  46.2476      1.62193   28.51    <1e-1
growth        3.06053     0.696274   4.40    0.000
─────────────────────────────────────────────
```

```
hibbs_lm = lm(@formula(vote ~ growth),
hibbs)
```

```
▸ [-8.99292, 2.66743, 1.0609, 2.20753, -5.89044, 4
```

```
residuals(hibbs_lm)
```

```
2.2744434224582912
```

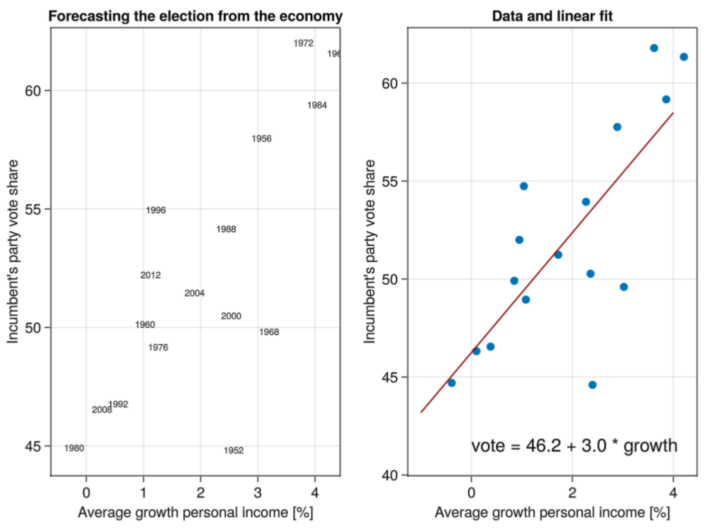
```
mad(residuals(hibbs_lm))
```

```
3.635681268522063
```

```julia
std(residuals(hibbs_lm))
```

```
[46.2476, 3.06053]
```

```julia
coef(hibbs_lm)
```




```julia
let
    fig = Figure()
    hibbs.label = string.(hibbs.year)
    xlabel = "Average growth personal income [%]"
    ylabel = "Incumbent's party vote share"
    let
        title = "Forecasting the election from the economy"
        ax = Axis(fig[1, 1]; title, xlabel, ylabel)
        for (ind, yr) in enumerate(hibbs.year)
            annotations!("$(yr)"; position=(hibbs.growth[ind], hibbs.vote[ind]), textsize=10)
        end
    end
    let
        x = LinRange(-1, 4, 100)
        title = "Data and linear fit"
        ax = Axis(fig[1, 2]; title, xlabel, ylabel)
        scatter!(hibbs.growth, hibbs.vote)
        lines!(x, coef(hibbs_lm)[1] .+ coef(hibbs_lm)[2] .* x; color=:darkred)
        annotations!("vote = 46.2 + 3.0 * growth"; position=(0, 41))
    end
    fig
end
```

Priors used in the Stan model.

```
stan1_0 = "
parameters {
    real b;               // Coefficient
    independent variable
    real a;               // Intercept
    real<lower=0> sigma; // dispersion
    parameter
}
model {
    // priors including constants
    a ~ normal(50, 20);
    b ~ normal(2, 10);
    sigma ~ exponential(1);
}";
```

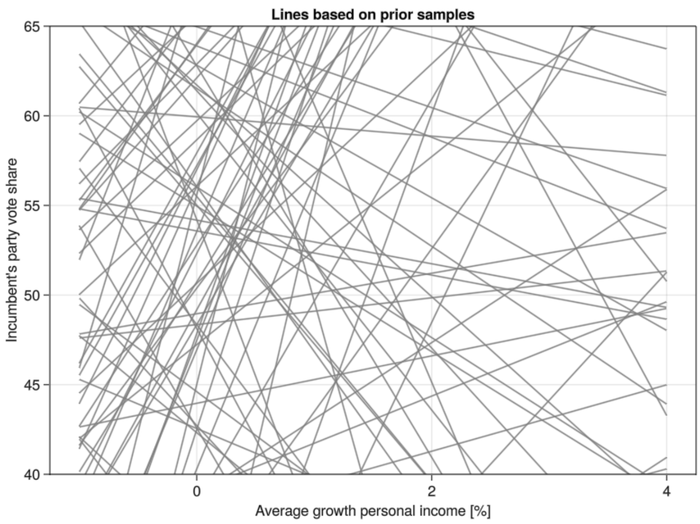
	parameters	mean	mcse	std	
1	"b"	1.65098	0.177134	10.0993	-1
2	"a"	49.9506	0.356125	20.1822	16
3	"sigma"	1.01953	0.0163088	1.01051	0.

```
begin
    m1_0s = SampleModel("hibbs", stan1_0)
    rc1_0s = stan_sample(m1_0s)
    success(rc1_0s) && describe(m1_0s)
end
```

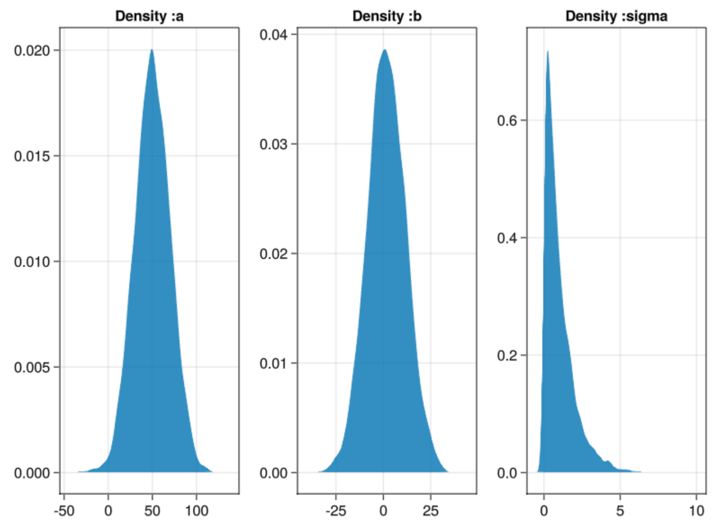
```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gn/T
d.
```

	parameters	median	mad_sd	mean	st
1	"a"	49.87	20.042	49.951	20.1
2	"b"	1.477	10.128	1.651	10.(
3	"sigma"	0.71	0.726	1.02	1.01

```
begin
    post1_0s = read_samples(m1_0s,
    :dataframe)
    ms1_0s = model_summary(post1_0s, [:a,
    :b, :sigma])
end
```




```
let
    fig = Figure()
    xlabel = "Average growth personal
    income [%]"
    ylabel="Incumbent's party vote share"
    ax = Axis(fig[1, 1]; title="Lines based
    on prior samples",
        xlabel, ylabel)
    ylims!(ax, 40, 65)
    xrange = LinRange(-1, 4, 200)
    for i = 1:100
        lines!(xrange, post1_0s.a[i] .+
        post1_0s.b[i] .* xrange, color =
        :grey)
    end
    fig
end
```




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="Density :a")
    density!(f[1, 1], post1_0s.a)
    ax = Axis(f[1, 2]; title="Density :b")
    density!(f[1, 2], post1_0s.b)
    ax = Axis(f[1, 3]; title="Density
    :sigma")
    density!(f[1, 3], post1_0s.sigma)
    f
end
```

Conditioning based on the available data.

```
stan1_1 = "
functions {
}
data {
    int<lower=1> N;       // total number of
    observations
    vector[N] growth;     // Independent
    variable: growth
    vector[N] vote;       // Dependent
    variable: votes
}
parameters {
    real b;               // Coefficient
    independent variable
    real a;               // Intercept
    real<lower=0> sigma;  // dispersion
    parameter
}
model {
    vector[N] mu;
    mu = a + b * growth;

    // priors including constants
    a ~ normal(50, 20);
    b ~ normal(2, 10);
    sigma ~ exponential(1);

    // likelihood including constants
    vote ~ normal(mu, sigma);
}";
```

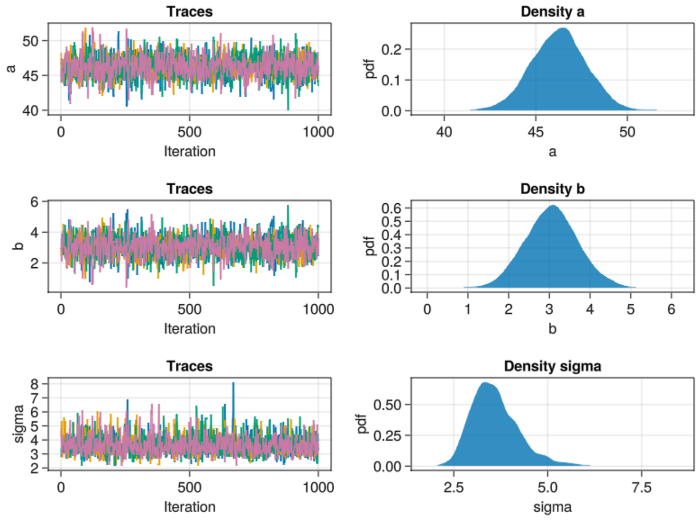
	parameters	mean	mcse	std	5[illegible]
1	"a"	46.27	0.04	1.54	43.[illegible]
2	"b"	3.06	0.02	0.67	1.9[illegible]
3	"sigma"	3.6	0.02	0.66	2.7

```
let
    data = (N=16, vote=hibbs.vote,
    growth=hibbs.growth)
    global m1_1s = SampleModel("hibbs",
    stan1_1)
    global rc1_1s = stan_sample(m1_1s; data)
    success(rc1_1s) && describe(m1_1s, [:a,
    :b, :sigma])
end
```

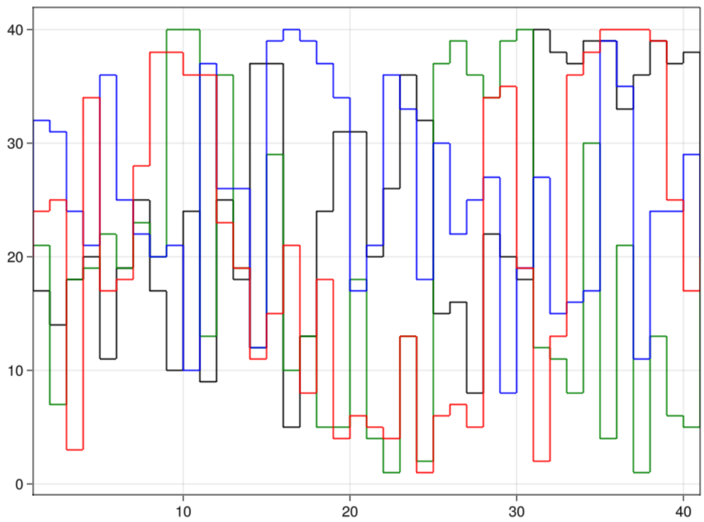
```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gn/T
d.
```

	parameters	median	mad_sd	mean	st[illegible]
1	"a"	46.279	1.488	46.266	1.5[illegible]
2	"b"	3.061	0.642	3.055	0.6[illegible]
3	"sigma"	3.516	0.588	3.604	0.6[illegible]

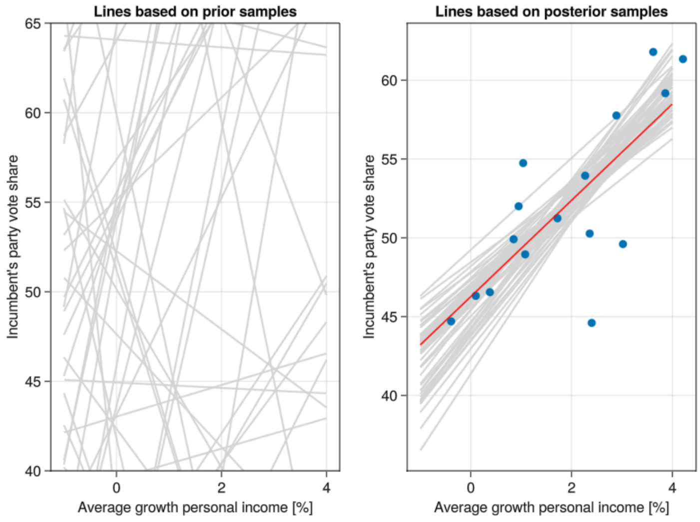
```
if success(rc1_1s)
    post1_1s = read_samples(m1_1s,
    :dataframe)
    ms1_1s = model_summary(post1_1s, [:a,
    :b, :sigma])
end
```




```
plot_chains(post1_1s, [:a, :b, :sigma])
```



```
trankplot(post1_1s, "b")
```



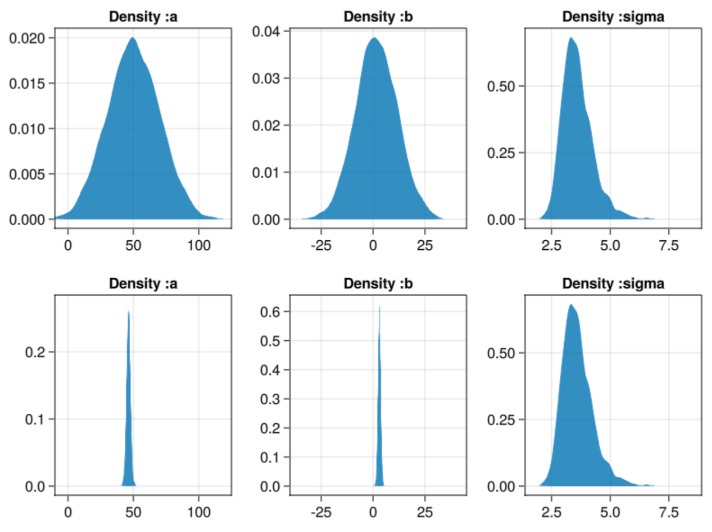

```
let
    N = 100
    x = LinRange(-1, 4, N)
    a = rand(Normal(50, 20), N)
    b = rand(Normal(2, 10), N)
    mat1 = zeros(50, 100)
    for i in 1:50
        mat1[i, :] = a[i] .+ b[i] .* x
    end
    ā = ms1_1s[:a, :mean]
    b̄ = ms1_1s[:b, :mean]

    # Maybe could use a `link` function here
    mat2 = zeros(50, 100)
    for i in 1:50
        mat2[i, :] = post1_1s.a[i] .+
        post1_1s.b[i] .* x
    end

    fig = Figure()
    xlabel = "Average growth personal
    income [%]"
    ylabel="Incumbent's party vote share"
    ax = Axis(fig[1, 1]; title="Lines based
    on prior samples",
        xlabel, ylabel)
    ylims!(ax, 40, 65)
    series!(fig[1, 1], x, mat1,
    solid_color=:lightgrey)
    ax = Axis(fig[1, 2]; title="Lines based
    on posterior samples",
        xlabel, ylabel)
    series!(fig[1, 2], x, mat2,
    solid_color=:lightgrey)
    scatter!(hibbs.growth, hibbs.vote)
```

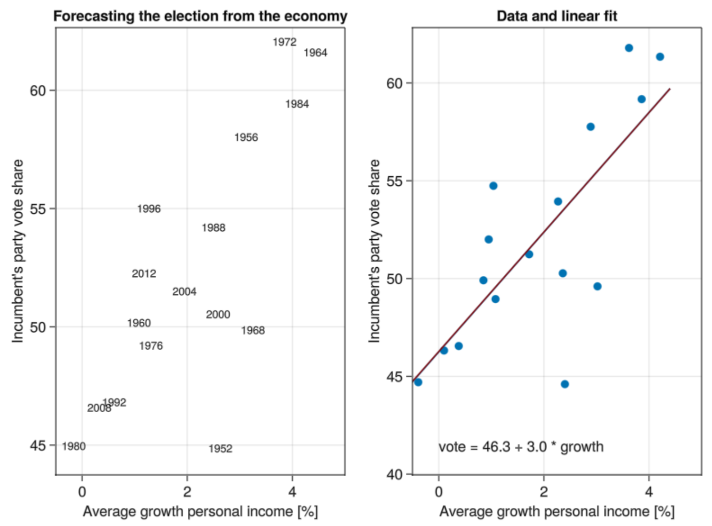
```
    lines!(fig[1, 2], x, ā .+ b̄ * x, color
    = :red)

    fig
end
```




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="Density :a")
    xlims!(ax, -10, 125)
    density!(post1_0s.a)
    ax = Axis(f[1, 2]; title="Density :b")
    xlims!(ax, -40, 45)
    density!(post1_0s.b)
    ax = Axis(f[1, 3]; title="Density
    :sigma")
    density!(post1_1s.sigma)

    ax = Axis(f[2, 1]; title="Density :a")
    density!(post1_1s.a)
    xlims!(ax, -10, 125)
    ax = Axis(f[2, 2]; title="Density :b")
    xlims!(ax, -40, 45)
    density!(post1_1s.b)
    ax = Axis(f[2, 3]; title="Density
    :sigma")
    density!(post1_1s.sigma)
    f
end
```



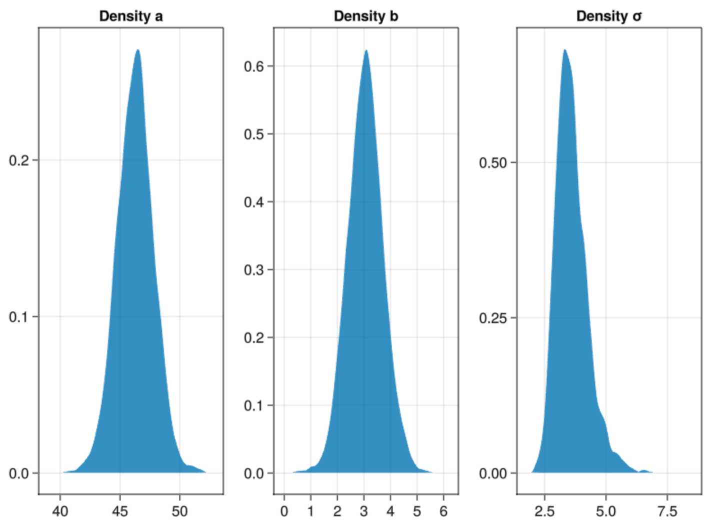

```
begin
    fig = Figure()
    hibbs.label = string.(hibbs.year)
    xlabel = "Average growth personal
    income [%]"
    ylabel="Incumbent's party vote share"

    # Same figure as above
    let
        title = "Forecasting the election
        from the economy"
        ax = Axis(fig[1, 1]; title, xlabel,
        ylabel)
        xlims!(ax, -0.5, 5)
        for (ind, yr) in
        enumerate(hibbs.year)
            annotations!("$(yr)"; position=
            (hibbs.growth[ind],
            hibbs.vote[ind]), textsize=12)
        end
    end

    # Superimpose Stan fit
    let
        ā = ms1_1s[:a, :mean]
        b̄ = ms1_1s[:b, :mean]
        title = "Compare GLM and Stan
        fitted lines"
        axis = (; title, xlabel, ylabel)

        x = LinRange(-1, 4.4, 100)
        title = "Data and linear fit"
        ax = Axis(fig[1, 2]; title, xlabel,
        ylabel)
        xlims!(ax, -0.5, 5)
        scatter!(hibbs.growth, hibbs.vote)
```

```
            lines!(x, coef(hibbs_lm)[1] .+
            coef(hibbs_lm)[2] .* x)
            lines!(x, ā .+  b̄ .* x;
            color=:darkred)
            annotations!("vote = $(round(ā,
            digits=1)) + $(round(b̄, digits=0)) \
                * growth"; position=(0, 41),
                textsize=16)
        end
        fig
    end
```




```
let
    N = 10000
    nt = (
        a = post1_1s.a,
        b = post1_1s.b,
        σ = post1_1s.sigma,
    )

    fig = Figure()
    for (i, k) in enumerate(keys(nt))
        ax = Axis(fig[1, i]; title =
        "Density $k")
        den = density!(nt[k])
    end
    fig
end
```

Compute median and mad.

Alternative computation of mad().

[1.48804, 0.642436, 0.587705]

```
let
    1.483 .* [
        median(abs.(post1_1s.a .-
        median(post1_1s.a))),
        median(abs.(post1_1s.b .-
        median(post1_1s.b))),
        median(abs.(post1_1s.sigma .-
        median(post1_1s.sigma)))]
end
```

ms1_1 =

	parameters	median	mad_sd	mean	st
1	"a"	46.279	1.488	46.266	1.53
2	"b"	3.061	0.642	3.055	0.66
3	"sigma"	3.516	0.588	3.604	0.65

```
ms1_1 = model_summary(post1_1s, ["a", "b",
"sigma"])
```

0.642

```
ms1_1[:b, :mad_sd]
```

ss1_1 =

	parameters	mean	mcse	std	5
1	"a"	46.27	0.04	1.54	43.7
2	"b"	3.06	0.02	0.67	1.98
3	"sigma"	3.6	0.02	0.66	2.7

```
ss1_1 = describe(m1_1s, ["a", "b",
"sigma"]; digits=2)
```

1490.84

```
ss1_1["a", "ess"]
```

Quick simulation with median, mad, mean and std of Normal observations.

```
nt =
▸ (x = [3.99721, 6.45366, 4.15654, 4.08856, 0.108
```

```julia
nt = (x=rand(Normal(5, 2), 10000),)
```

```
▸ [4.98422, 2.00871, 4.9748, 1.99798]
```

```julia
[median(nt.x), mad(nt.x), mean(nt.x),
std(nt.x)]
```

```
sd_mean = 0.02
```

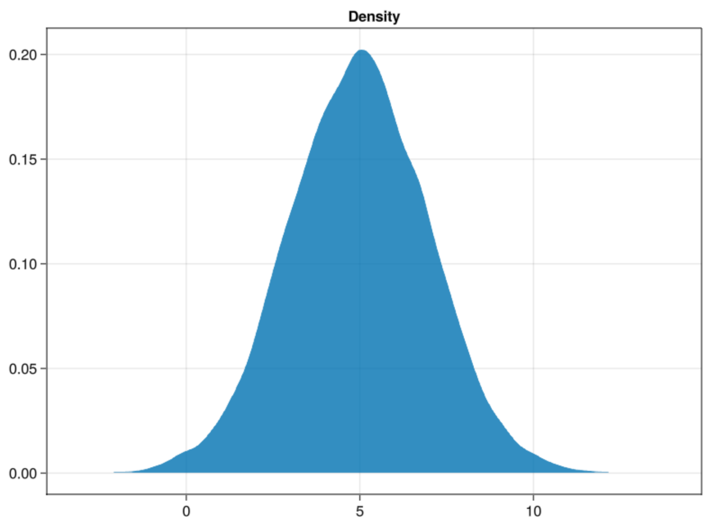
```julia
sd_mean = round(mad(nt.x)/√10000; digits=2)
```

```
1.3548569833911626
```

```julia
median(abs.(nt.x .- median(nt.x)))
```

```
2.009252906369094
```

```julia
1.483 * median(abs.(nt.x .- median(nt.x)))
```




```julia
let
    fig = Figure()
    ax = Axis(fig[1, 1]; title = "Density")
    den = density!(nt.x)
    fig
end
```

```
▸ [1.08543, 8.91972]
```

```julia
quantile(nt.x, [0.025, 0.975])
```

[3.61705, 6.32982]

```
quantile(nt.x, [0.25, 0.75])
```

A closer look at Stan's summary. Below the full version:

	parameters	mean	mcse	
1	"lp__"	-31.9627	0.037871	1.3
2	"accept_stat__"	0.917142	0.00184506	0.1
3	"stepsize__"	0.41254	0.0281173	0.0
4	"treedepth__"	2.647	0.0165177	0.6
5	"n_leapfrog__"	7.766	0.0928592	4.1
6	"divergent__"	0.0	NaN	0.0
7	"energy__"	33.4793	0.049884	1.8
8	"b"	3.05516	0.0169224	0.6
9	"a"	46.2662	0.039804	1.5
10	"sigma"	3.6036	0.015569	0.6

```
success(rc1_1s) && describe(m1_1s;
showall=true)
```

Usually I use the abbreviated version:

ss1_1s =

	parameters	mean	mcse	std	5
1	"b"	3.06	0.02	0.67	1.9
2	"a"	46.27	0.04	1.54	43.
3	"sigma"	3.6	0.02	0.66	2.7

```
ss1_1s = success(rc1_1s) && describe(m1_1s,
names(post1_1s))
```

	parameters	median	mad_sd	mean	st
1	"a"	46.279	1.488	46.266	1.53
2	"b"	3.061	0.642	3.055	0.66
3	"sigma"	3.516	0.588	3.604	0.65

```
ms1_1s
```

1490.84

```
ss1_1s[:a, :ess]
```

1.488

```
ms1_1s[:a, :mad_sd]
```

Experimental use of BridgeStan.

```
bernoulli_model = "
data {
  int<lower=1> N;
  int<lower=0,upper=1> y[N];
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  theta ~ beta(1,1);
  y ~ bernoulli(theta);
}
";
```

	parameters	mean	mcse	st
1	:lp__	-8.18169	0.0173597	0.752
2	:accept_stat__	0.898038	0.00225164	0.151
3	:stepsize__	1.10995	0.0378972	0.053
4	:treedepth__	1.38625	0.00828256	0.486
5	:n_leapfrog__	2.27	0.0174635	0.972
6	:divergent__	0.0	NaN	0.0
7	:energy__	8.6847	0.0243358	1.041
8	:theta	0.329246	0.0035045	0.131

```
begin
    data = Dict("N" => 10, "y" => [0, 1, 0,
    1, 0, 0, 0, 0, 0, 1])
    sm = SampleModel("bernoulli",
    bernoulli_model)
    rc = stan_sample(sm; data)
    success(rc) && read_summary(sm)
end
```

```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gn/T
dated.
```

```
st =
StanSample.StanTable{Matrix{Float64}} with 4000 r
 :theta   Float64
```

```
st = success(rc) && read_samples(sm)
```

```
bernoulli_lib =
"/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gn/T
```

```
bernoulli_lib = joinpath(sm.tmpdir,
"bernoulli_model.so")
```

	x	q	log_density	gradient
1	0.883837	2.02928	-15.4398	-5.83837

```
if isfile(bernoulli_lib)
    blib = Libc.Libdl.dlopen(bernoulli_lib)

    bernoulli_data = joinpath(sm.tmpdir,
"bernoulli_data_1.json")
    smb = StanModel(blib, bernoulli_data)
    x = rand(smb.dims)
    q = @. log(x / (1 - x))        #
unconstrained scale
    log_density_gradient!(smb, q, jacobian
= 0)
    DataFrame(x=x, q=q,
log_density=smb.log_density,
gradient=smb.gradient)
else
    @info "Shared library
`bernoulli_model.so` has not been created."
    @info "Maybe BridgeStan has not been
installed in $(ENV["CMDSTAN"])?"
end
```

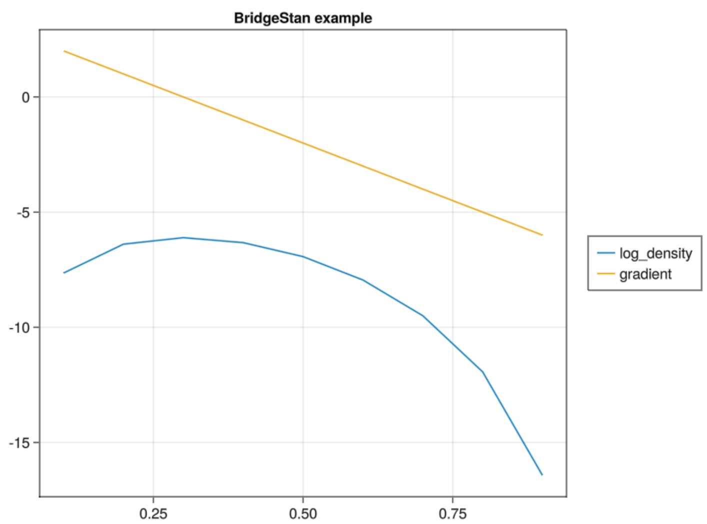
```
sim (generic function with 2 methods)
```

```
function sim(smb::StanModel, x=0.1:0.1:0.9)
    y = zeros(length(x))
    q = zeros(length(x))
    ld = zeros(length(x))
    g = zeros(length(x))
    for (i, p) in enumerate(x)
        y[i] = p
        q[i] = @. log(p / (1 - p))        #
        unconstrained scale
        log_density_gradient!(smb, q[i],
        jacobian = 0)
        ld[i] = smb.log_density[1]
        g[i] = smb.gradient[1]
    end
    return DataFrame(x=x, q=q,
    log_density=ld, gradient=g)
end
```

```
sim_df =
```

	x	q	log_density	gradient
1	0.1	-2.19722	-7.64528	2.0
2	0.2	-1.38629	-6.39032	1.0
3	0.3	-0.847298	-6.10864	0.0
4	0.4	-0.405465	-6.32465	-1.0
5	0.5	0.0	-6.93147	-2.0
6	0.6	0.405465	-7.94651	-3.0
7	0.7	0.847298	-9.49783	-4.0
8	0.8	1.38629	-11.9355	-5.0
9	0.9	2.19722	-16.4342	-6.0

```
sim_df = sim(smb)
```




```
let
    f = Figure()
    ax = Axis(f[1, 1]; title="BridgeStan example")
    dens = lines!(sim_df.x, sim_df.log_density)
    gra = lines!(sim_df.x, sim_df.gradient)
    Legend(f[1, 2], [dens, gra], ["log_density", "gradient"])
    f
end
```