See chapter 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 ✓
```

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

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

    # Graphics related
    using CairoMakie ✓
    using AlgebraOfGraphics ✓

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

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

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

1.1 The three challenges of statistics.

Note

It is not common for me to copy from the book but this particular section deserves an exception!

The three challenges of statistical inference are:

1. Generalizing from sample to population, a problem that is associated with survey sampling but actually arises in nearly every application of statistical inference;
2. Generalizing from treatment to control group, a problem that is associated with causal inference, which is implicitly or explicitly part of the interpretation of most regressions we have seen; and
3. Generalizing from observed measurements to the underlying constructs of interest, as most of the time our data do not record exactly what we would ideally like to study.

All three of these challenges can be framed as problems of prediction (for new people or new items that are not in the sample, future outcomes under different potentially assigned treatments, and underlying constructs of interest, if they could be measured exactly).

1.2 Why learn regression?

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-
growth        3.06053     0.696274   4.40    0.00
─────────────────────────────────────────────────
```

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

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

```
residuals(hibbs_lm)
```

```
2.2744434224582912
```

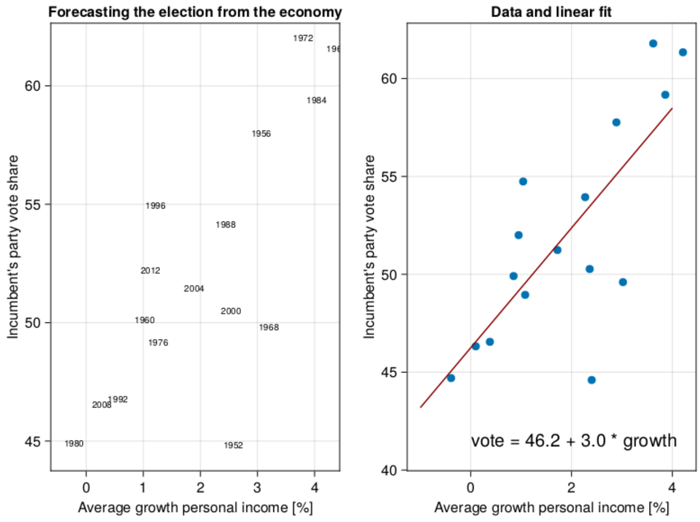
```
mad(residuals(hibbs_lm))
```

```
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
```

```
ppl7_1 (generic function with 2 methods)
```

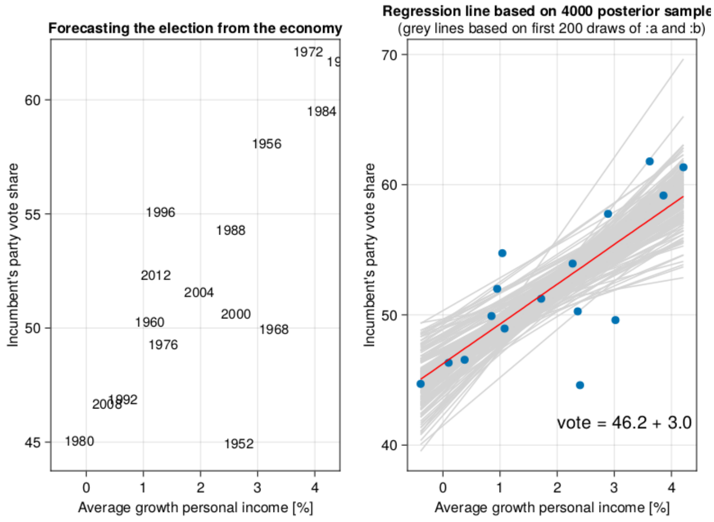
```
@model function ppl7_1(growth, vote)
    a ~ Normal(50, 20)
    b ~ Normal(2, 10)
    σ ~ Exponential(1)
    μ = a .+ b .* growth
    for i in eachindex(vote)
        vote[i] ~ Normal(μ[i], σ)
    end
end
```

	parameters	mean	std	naive_se
1	:a	46.2658	1.57137	0.0248455
2	:b	3.04382	0.673282	0.0106455
3	:σ	3.59767	0.625671	0.0098927

```
begin
    m7_1t = ppl7_1(hibbs.growth, hibbs.vote)
    chns7_1t = sample(m7_1t, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns7_1t)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	46.242	1.537	46.266	1.57
2	"b"	3.053	0.642	3.044	0.67
3	"σ"	3.526	0.589	3.598	0.62

```
begin
    post7_1t = DataFrame(chns7_1t)[:, 3:5]
    ms7_1t = model_summary(post7_1t,
    names(post7_1t))
end
```




```
let
    growth_range =
    LinRange(minimum(hibbs.growth),
    maximum(hibbs.growth), 200)
    votes = median.(link(post7_1t, (r,x) ->
    r.a + x * r.b, growth_range))

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

    fig = Figure()
    let
        title = "Forecasting the election
from the economy"
        plt = data(hibbs) *
            mapping(:label => verbatim,
(:growth, :vote) => Point) *
            visual(Annotations, textsize=15)
        axis = (; title, xlabel, ylabel)
        draw!(fig[1, 1], plt; axis)
    end

    ax = Axis(fig[1, 2]; title="Regression
    line based on 4000 posterior samples",
        subtitle = "(grey lines based on
        first 200 draws of :a and :b)",
        xlabel, ylabel)
    for i in 1:200
        lines!(growth_range, post7_1t.a[i]
        .+ post7_1t.b[i] .* growth_range,
        color = :lightgrey)
    end
    scatter!(hibbs.growth, hibbs.vote)
```

```
    lines!(growth_range, votes, color =
    :red)
    annotations!("vote = 46.2 + 3.0 *
growth"; position=(2, 41))
    fig
end
```

1.3 Some examples of regression.

Electric company

	post_test	pre_test	grade	t
1	48.9	13.8	1	1
2	70.5	16.5	1	1
3	89.7	18.5	1	1
4	44.2	8.8	1	1
5	77.5	15.3	1	1
6	84.7	15.0	1	1
7	78.9	19.4	1	1
8	86.8	15.0	1	1
9	60.8	11.8	1	1
10	75.7	16.4	1	1
⋮ more				
192	110.0	102.6	4	0

```
begin
    electric =
    CSV.read(ros_datadir("ElectricCompany",
    "electric.csv"), DataFrame)
    electric = electric[:, [:post_test,
    :pre_test, :grade, :treatment]]
    electric.grade =
    categorical(electric.grade)
    electric.treatment =
    categorical(electric.treatment)
    electric
end
```

A quick look at the overall values of `pre_test` and `post_test`.

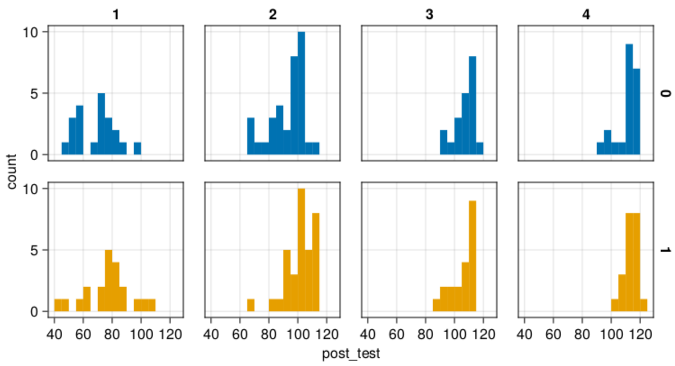
	variable	mean	min	median	max
1	:post_test	97.1495	44.2	102.3	122.0
2	:pre_test	72.2245	8.8	80.75	119.8
3	:grade	nothing	1	nothing	4
4	:treatment	nothing	0	nothing	1

```
describe(electric)
```

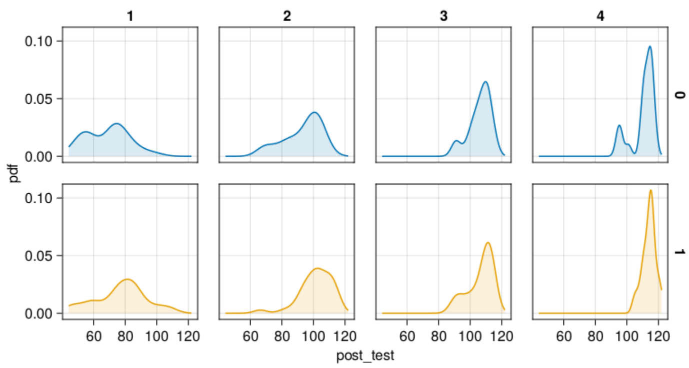
```
true
```

```
all(completecases(electric)) == true
```

Post-test density for each grade conditioned on treatment.



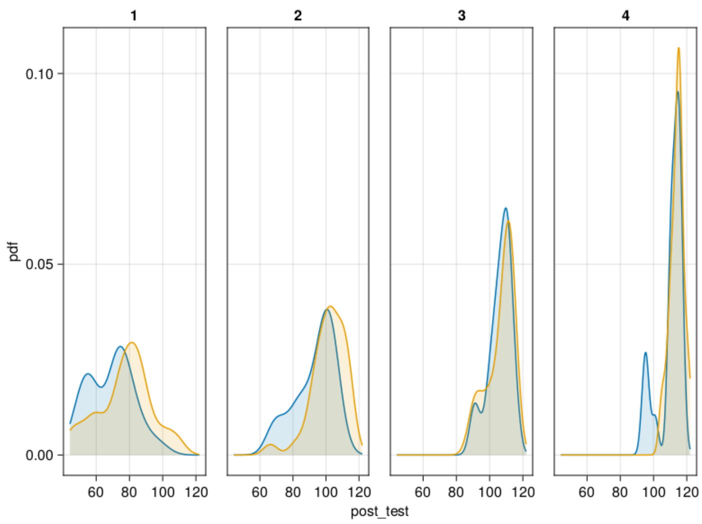

```
let
    f = Figure()
    axis = (; width = 150, height = 150)
    el = data(electric) *
    mapping(:post_test, col=:grade,
    color=:treatment)
    plt = el *
    AlgebraOfGraphics.histogram(;bins=20) *
    mapping(row=:treatment)
    draw!(f[1, 1], plt; axis)
    f
end
```



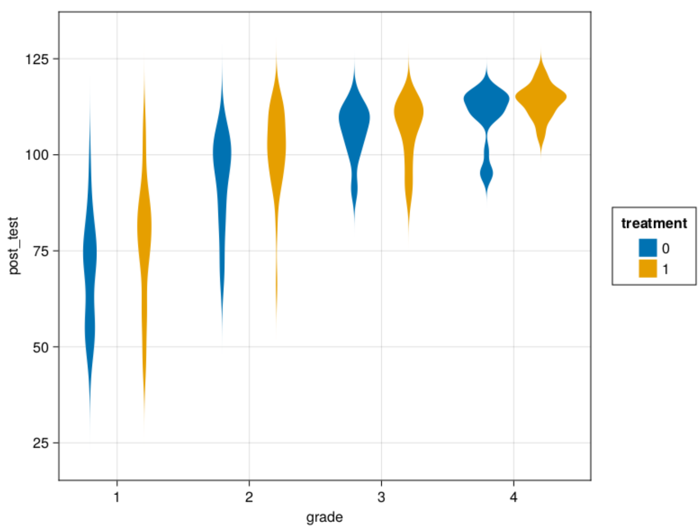

```
let
    f = Figure()
    axis = (; width = 150, height = 150)
    el = data(electric) *
    mapping(:post_test, col=:grade,
    color=:treatment)
    plt = el * AlgebraOfGraphics.density()
    * mapping(row=:treatment)
    draw!(f[1, 1], plt; axis)
    f
end
```

Note

In above cell, as density() is exported by both GLMakie and AlgebraOfGraphics, it needs to be qualified.




```
let
    f = Figure()
    el = data(electric) *
    mapping(:post_test, col=:grade)
    plt = el * AlgebraOfGraphics.density()
    * mapping(color=:treatment)
    draw!(f[1, 1], plt)
    f
end
```




```
let
    plt = data(electric) * visual(Violin) *
    mapping(:grade, :post_test,
    dodge=:treatment, color=:treatment)
    draw(plt)
end
```

Peacekeeping

peace =

	war	cfdate	faildate
1	"Afghanistan-Mujahideen"	8150	8257
2	"Afghanistan-Taliban"	8466	8505
3	"Algeria-FIS/AIS"	10149	12783
4	"Angola"	7820	8319
5	"Angola"	9089	10564
6	"Azerbaijan-N.K."	8643	8678
7	"Azerbaijan-N.K."	8901	12783
8	"Bangladesh-CHT"	8248	12783
9	"Myanmar-Karen"	8153	9282
10	"Myanmar-Karen"	9296	9907
⋮ more			
96	"Yugoslavia-Kosovo"	10751	12783

```
peace =
CSV.read(ros_datadir("PeaceKeeping",
"peacekeeping.csv"), missingstring="NA",
DataFrame)
```

	variable	mean	min
1	:war	nothing	"Afghanistan-Mujah:
2	:cfdate	8925.1	6985
3	:faildate	10795.8	7074
4	:peacekeepers	0.354167	0
5	:badness	-8.15228	-12.26
6	:delay	5.12177	0.04
7	:censored	0.416667	0

```
describe(peace)
```

A quick look at this Dates stuff!

```
8150
```

```julia
peace.cfdate[1]
```

```
1992-04-25T00:00:00
```

```julia
DateTime(1992, 4, 25)
```

```
107 days
```

```julia
Date(1992, 8, 10) - Date(1992, 4, 25)
```

```
1970-01-01
```

```julia
Date(1970,1,1)
```

```
1992-04-25
```

```julia
Date(1970,1,1) + Dates.Day(8150)
```

```
8150 days
```

```julia
Date(1992, 4, 25) - Date(1970, 1, 1)
```

```
107
```

```julia
peace.faildate[1] - peace.cfdate[1]
```

```julia
begin
    pks_df = peace[peace.peacekeepers .==
    1, [:cfdate, :faildate]]
    nopks_df = peace[peace.peacekeepers .==
    0, [:cfdate, :faildate]]
end;
```

```
0.4166666666666667
```

```julia
mean(peace.censored)
```

```
64
```

```julia
length(unique(peace.war))
```

```
0.5588235294117647
```

```julia
mean(peace[peace.peacekeepers .== 1,
:censored])
```

```
0.3387096774193548
```

```julia
mean(peace[peace.peacekeepers .== 0,
:censored])
```

1.382

```
mean(peace[peace.peacekeepers .== 1 .&&
peace.censored .== 0, :delay])
```

1.5153658536585364

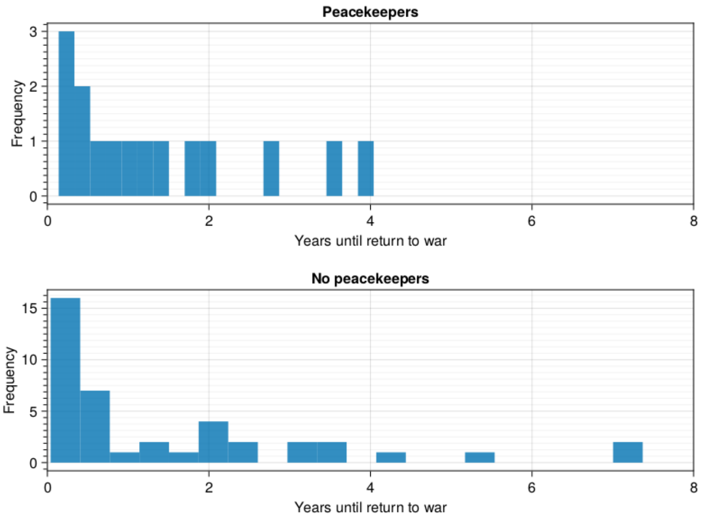
```
mean(peace[peace.peacekeepers .== 0 .&&
peace.censored .== 0, :delay])
```

1.05

```
median(peace[peace.peacekeepers .== 1 .&&
peace.censored .== 0, :delay])
```

0.59

```
median(peace[peace.peacekeepers .== 0 .&&
peace.censored .== 0, :delay])
```




```
let
    f = Figure()
    pks = peace[peace.peacekeepers .== 1
    .&& peace.censored .== 0, :]
    nopks = peace[peace.peacekeepers .== 0
    .&& peace.censored .== 0,:]

    for i in 1:2
        title = i == 1 ? "Peacekeepers" :
        "No peacekeepers"

        ax = Axis(f[i, 1]; title,
        xlabel="Years until return to war",
        ylabel = "Frequency",
    yminorticksvisible = true,
        yminorgridvisible = true,
        yminorticks = IntervalsBetween(8))

        xlims!(ax, [0, 8])
        hist!(i == 1 ? pks.delay :
        nopks.delay; bins=20)
    end
    f
end
```

Note

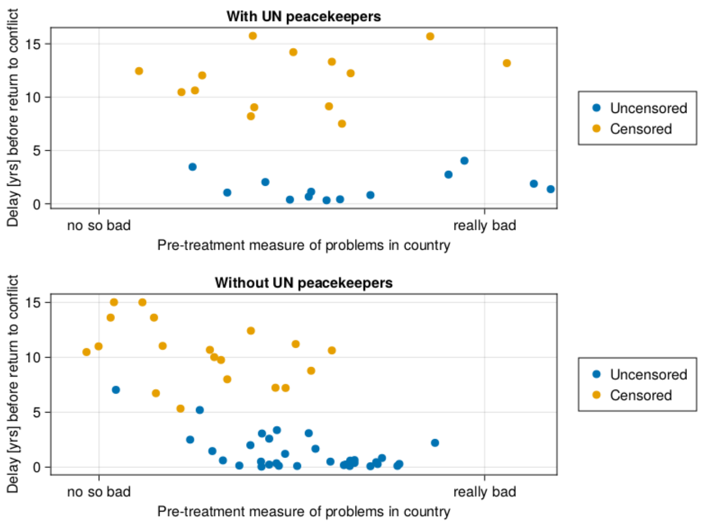
Censored means conflict had not returned until end of observation period (2004).

```julia
begin
    # Filter out missing badness rows.
    pb = peace[peace.badness .!== missing,
    :];

    # Delays until return to war for
    uncensored, peacekeeper cases
    pks_uc = pb[pb.peacekeepers .== 1 .&&
    pb.censored .== 0, :delay]
    # Delays until return to war for
    censored, peacekeeper cases
    pks_c = pb[pb.peacekeepers .== 1 .&&
    pb.censored .== 1, :delay]

    # No peacekeepr cases.
    nopks_uc = pb[pb.peacekeepers .== 0 .&&
    pb.censored .== 0, :delay]
    nopks_c = pb[pb.peacekeepers .== 0 .&&
    pb.censored .== 1, :delay]

    # Crude measure (:badness) used for
    assessing situation
    badness_pks_uc = pb[pb.peacekeepers .==
    1 .&& pb.censored .== 0,
        :badness]
    badness_pks_c = pb[pb.peacekeepers .== 1
     .&& pb.censored .== 1,
        :badness]
    badness_nopks_uc = pb[pb.peacekeepers
    .== 0 .&& pb.censored .== 0,
        :badness]
    badness_nopks_c = pb[pb.peacekeepers
    .== 0 .&& pb.censored .== 1,
        :badness]
end;
```




```julia
begin
    local f = Figure()
    ax = Axis(f[1, 1], title = "With UN peacekeepers",
        xlabel = "Pre-treatment measure of problems in country",
        ylabel = "Delay [yrs] before return to conflict")
    sca1 = scatter!(badness_pks_uc, pks_uc)
    sca2 = scatter!(badness_pks_c, pks_c)
    xlims!(ax, [-13, -2.5])
    Legend(f[1, 2], [sca1, sca2], ["Uncensored", "Censored"])
    ax.xticks = ([-12, -4], ["no so bad", "really bad"])


    ax = Axis(f[2, 1], title = "Without UN peacekeepers",
        xlabel = "Pre-treatment measure of problems in country",
        ylabel = "Delay [yrs] before return to conflict")
    sca1 = scatter!(badness_nopks_uc, nopks_uc)
    sca2 = scatter!(badness_nopks_c, nopks_c)
    xlims!(ax, [-13, -2.5])
    Legend(f[2, 2], [sca1, sca2], ["Uncensored", "Censored"])
    ax.xticks = ([-12, -4], ["no so bad", "really bad"])

    f
end
```

1.4 Challenges in building, understanding, and interpreting regression.

Simple causal

```
ppl1_2a (generic function with 2 methods)
```

```
@model function ppl1_2a(x, y)
    a ~ Normal(10, 10)
    b ~ Normal(10, 10)
    σ ~ Exponential(1)
    μ = a .+ b .* x
    for i in eachindex(x)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

```
ppl1_2b (generic function with 2 methods)
```

```
@model function ppl1_2b(x_binary, y)
    a ~ Normal(10, 10)
    b ~ Normal(10, 10)
    σ ~ Exponential(1)
    μ = a .+ b .* x_binary
    for i in eachindex(x_binary)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

Note

Aki Vehtari did not include a seed number in his code.

```
▸ [24.3056, 18.7671, 24.9769, 19.2313, 16.5586, 11
```

```
begin
    Random.seed!(123)
    n = 50
    x = rand(Uniform(1, 5), n)
    x_binary = [x[i] < 3 ? 0 : 1 for i in
    1:n]
    y = [rand(Normal(10 + 3x[i], 3), 1)[1]
    for i in 1:n]
end
```

	iteration	chain	a	b	σ
1	501	1	11.1788	2.73829	3.64277
2	502	1	9.42879	3.32674	2.8129
3	503	1	9.42879	3.32674	2.8129
4	504	1	7.77652	3.82608	3.868
5	505	1	8.24453	3.47313	3.92143
6	506	1	9.43826	3.0795	3.1656
7	507	1	10.0971	3.22799	3.44101
8	508	1	9.68236	2.95636	3.06576
9	509	1	11.2297	2.59135	3.5051
10	510	1	10.7354	2.75952	3.20894

⋮ more

```
begin
    m1_2at = ppl1_2a(x, y)
    chns1_2at = sample(m1_2at, NUTS(),
    MCMCThreads(), 1000, 4)
end
```

▶ [

	parameters	mean	std	naive_se
1	:a	9.45302	1.41061	0.0223037
2	:b	3.22644	0.4397	0.0069522
3	:σ	3.46059	0.354418	0.0056038

```
describe(chns1_2at)
```

	parameters	median	mad_sd	mean	st
1	"a"	9.429	1.343	9.453	1.4
2	"b"	3.235	0.41	3.226	0.4
3	"σ"	3.435	0.345	3.461	0.3

```
begin
    post1_2at = DataFrame(chns1_2at)[:, 3:5]
    ms1_2at = model_summary(post1_2at,
    names(post1_2at))
end
```

	iteration	chain	a	b	σ
1	501	1	15.4531	7.85783	3.4064
2	502	1	16.8764	6.2749	3.99948
3	503	1	16.4703	6.19308	3.17825
4	504	1	15.6367	6.99355	3.61191
5	505	1	16.6607	6.61947	3.19945
6	506	1	15.6082	7.19212	2.89343
7	507	1	16.8871	6.0466	3.80746
8	508	1	17.3657	6.1923	3.73728
9	509	1	15.9063	6.6282	3.77437
10	510	1	15.8045	8.20579	3.08753

⋮ more

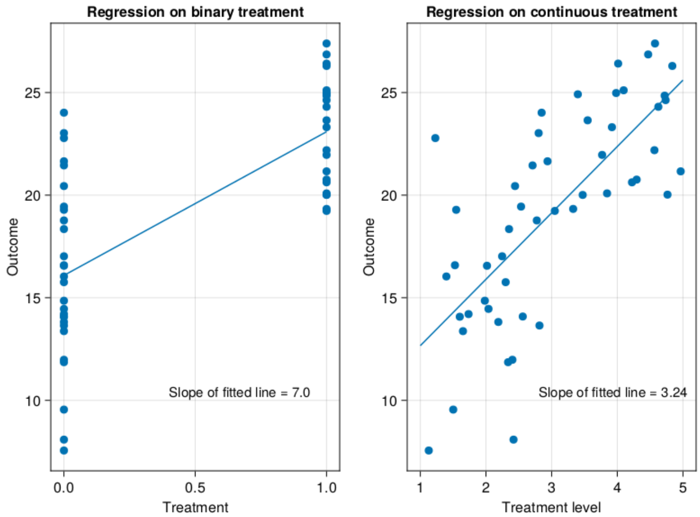
```
begin
    m1_2bt = ppl1_2b(x_binary, y)
    chns1_2bt = sample(m1_2bt, NUTS(),
    MCMCThreads(), 1000, 4)
end
```

	parameters	mean	std	naive_se
1	:a	16.1111	0.70719	0.0111817
2	:b	6.99196	1.06435	0.0168288
3	:σ	3.6793	0.352004	0.0055656

```
describe(chns1_2bt)
```

	parameters	median	mad_sd	mean	st
1	"a"	16.089	0.697	16.111	0.70
2	"b"	7.002	1.063	6.992	1.06
3	"σ"	3.656	0.34	3.679	0.35

```
begin
    post1_2bt = DataFrame(chns1_2bt)[:, 3:5]
    ms1_2bt = model_summary(post1_2bt,
    names(post1_2bt))
end
```




```
let
    x1 = 1.0:0.01:5.0
    f = Figure()
    medians = [ms1_2at[p, "median"] for p in
     [:a, :b, :σ]]
    ax = Axis(f[1, 2], title = "Regression
    on continuous treatment",
        xlabel = "Treatment level", ylabel
        = "Outcome")
    sca1 = scatter!(x, y)
    annotations!("Slope of fitted line =
    $(round(medians[2], digits=2))",
        position = (2.8, 10), textsize=15)
    lin1 = lines!(x1, medians[1] .+
    medians[2] * x1)

    x2 = 0.0:0.01:1.0
    medians = [ms1_2bt[p, "median"] for p in
     [:a, :b, :σ]]
    ax = Axis(f[1, 1], title="Regression on
    binary treatment",
        xlabel = "Treatment", ylabel =
        "Outcome")
    sca1 = scatter!(x_binary, y)
    lin1 = lines!(x2, medians[1] .+
    medians[2] * x2)
    annotations!("Slope of fitted line =
    $(round(medians[2], digits=2))",
        position = (0.4, 10), textsize=15)
    f
end
```

ppl1_3a (generic function with 2 methods)

```
@model function ppl1_3a(x, y)
    a ~ Normal(10, 5)
    b ~ Normal(0, 5)
    σ ~ Exponential(1)
    μ = a .+ b .* x
    for i in eachindex(x)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

ppl1_3b (generic function with 2 methods)

```
@model function ppl1_3b(x, y)
    a ~ Normal(10, 5)
    b_exp ~ Normal(5, 5)
    σ ~ Exponential(1)
    μ = a .+ b_exp .* exp.(-x)
    for i in eachindex(x)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

```
begin
    #Random.seed!(1533)
    n1 = 50
    x1 = LinRange(1, 6, 50)
    y1 = [rand(Normal(5 + 30exp(-x1[i]),
    2), 1)[1] for i in 1:length(x1)]
end;
```

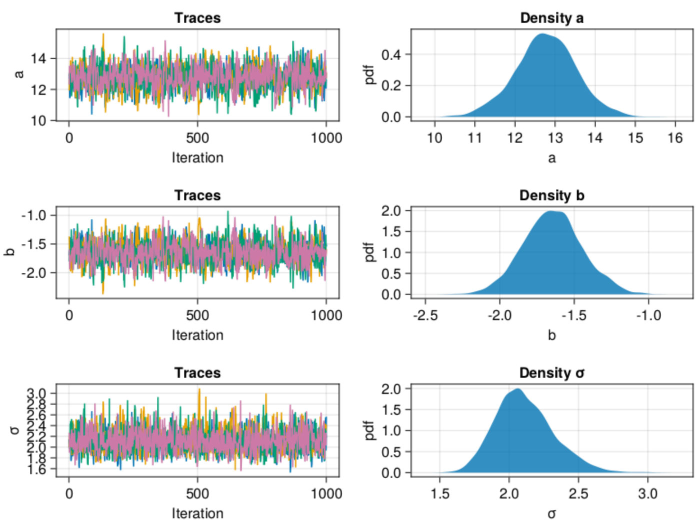
▸ [

	parameters	mean	std	naive_se
1	:a	12.771	0.753352	0.0119115
2	:b	-1.64431	0.198672	0.0031412
3	:σ	2.11171	0.210912	0.0033348

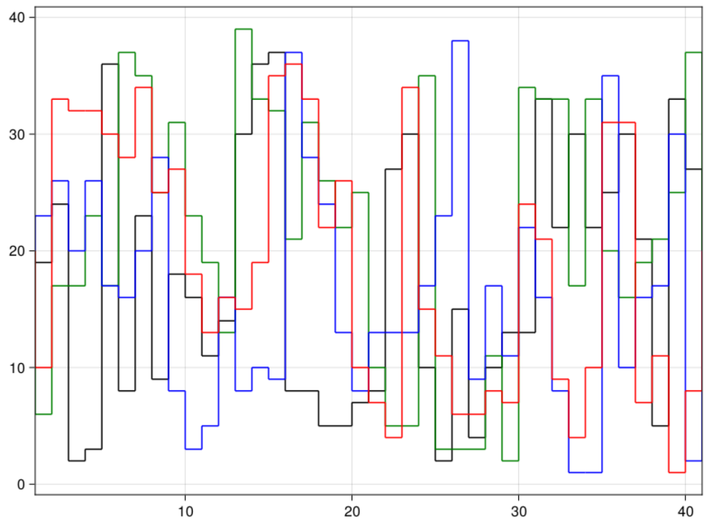
```
begin
    m1_3at = ppl1_3a(x1, y1)
    chns1_3at = sample(m1_3at, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns1_3at)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	12.778	0.726	12.771	0.75
2	"b"	-1.646	0.193	-1.644	0.19
3	"σ"	2.09	0.202	2.112	0.21

```julia
begin
    post1_3at = DataFrame(chns1_3at[[:a,
    :b, :σ]])
    ms1_3at = model_summary(post1_3at, [:a,
    :b, :σ])
end
```




```julia
plot_chains(post1_3at, [:a, :b, :σ])
```



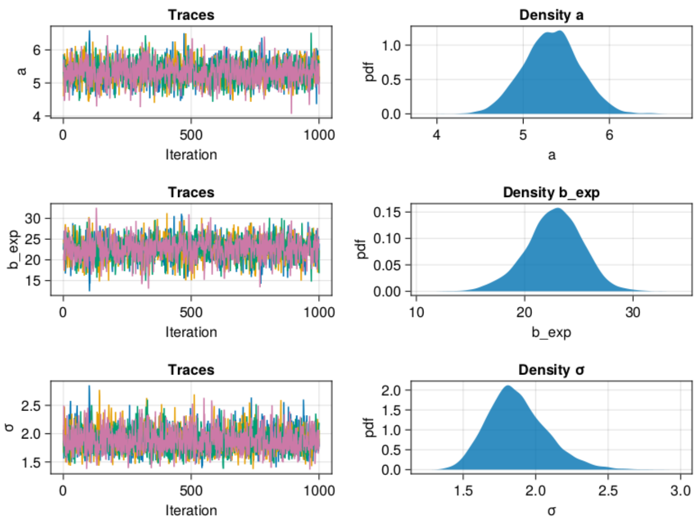

```
trankplot(post1_3at, "a")
```

	parameters	mean	std	naive_se
1	:a	5.32923	0.324659	0.0051333
2	:b_exp	22.7231	2.59568	0.0410412
3	:σ	1.87305	0.205268	0.0032455

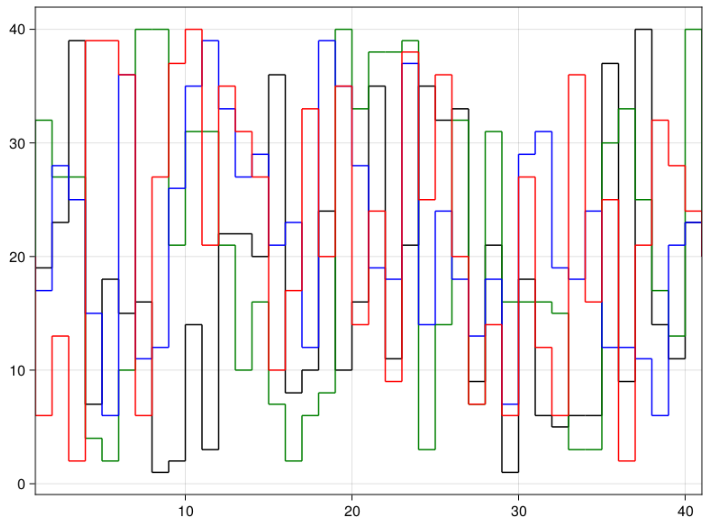
```
begin
    m1_3bt = ppl1_3b(x1, y1)
    chns1_3bt = sample(m1_3bt, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns1_3bt)
end
```

	parameters	median	mad_sd	mean	st
1	"a"	5.331	0.319	5.329	0.32
2	"b_exp"	22.828	2.488	22.723	2.59
3	"σ"	1.851	0.194	1.873	0.20

```
begin
    post1_3bt = DataFrame(chns1_3bt[[:a,
    :b_exp, :σ]])
    ms1_3bt = model_summary(post1_3bt, [:a,
    :b_exp, :σ])
end
```




```
plot_chains(post1_3bt, [:a, :b_exp, :σ])
```



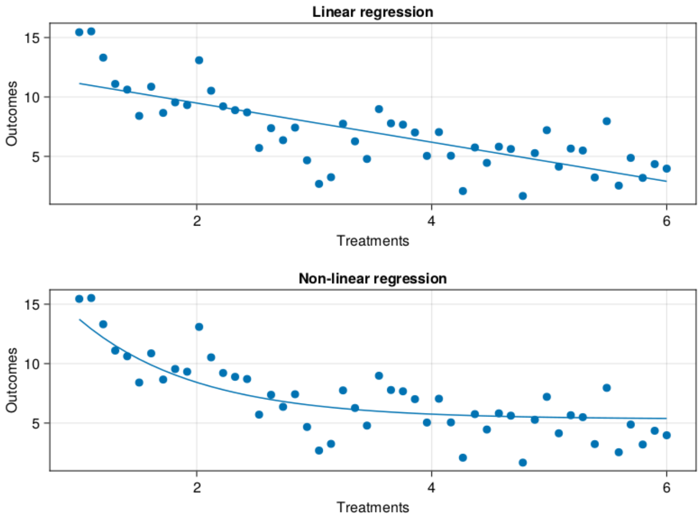

```
trankplot(post1_3bt, "b_exp")
```

```
▸ [12.778, -1.646, 2.09]
```

```
â₁, b̂, σ̂₁ = ms1_3at[:, :median]
```

```
▸ [5.331, 22.828, 1.851]
```

```
â₂, b̂ₑₓₚ, σ̂₂ = ms1_3bt[:, :median]
```




```
let
    f = Figure()
    ax = Axis(f[1, 1], title = "Linear
    regression",
        xlabel = "Treatments", ylabel =
        "Outcomes")
    scatter!(x1, y1)
    lines!(x1, â₁ .+ b̂ .* x1)

    ax = Axis(f[2, 1], title = "Non-linear
    regression",
        xlabel = "Treatments", ylabel =
        "Outcomes")
    scatter!(x1, y1)
    lines!(x1, â₂ .+ b̂ₑₓₚ .* exp.(-x1))
    f
end
```

	xx	z	yy
1	3.1425	0	37.5294
2	0.0335661	1	30.0628
3	1.60408	0	28.1095
4	0.478735	1	31.3638
5	2.65874	0	35.7382
6	1.02705	1	36.0009
7	1.28799	0	24.3213
8	0.052966	1	29.5242
9	0.543994	0	25.4181
10	0.0304007	1	25.1863
⋮ more			
100	0.00156342	1	28.8751

```
begin
    Random.seed!(12573)
    n2 = 100
    z = repeat([0, 1]; outer=50)
    df1_8 = DataFrame()
    df1_8.xx = [(z[i] == 0 ? rand(Normal(0, 1.2), 1).^2 : rand(Normal(0, 0.8), 1).^2)[1] for i in 1:n2]
    df1_8.z = z
    df1_8.yy = [rand(Normal(20 .+ 5df1_8.xx[i] .+ 10df1_8.z[i], 3), 1)[1] for i in 1:n2]
    df1_8
end
```

```
lm1_8 =
StatsModels.TableRegressionModel{LinearModel{GLM…

yy ~ 1 + xx + z

Coefficients:
─────────────────────────────────────────────
                  Coef.  Std. Error      t  Pr(>|t
─────────────────────────────────────────────
(Intercept)  20.1093      0.529823  37.95     <1e-5
xx            4.97503     0.213492  23.30     <1e-4
z             9.625       0.604978  15.91     <1e-2
─────────────────────────────────────────────
```

```
lm1_8 = lm(@formula(yy ~ xx + z), df1_8)
```

```
lm1_8_0 =
StatsModels.TableRegressionModel{LinearModel{GLM…

yy ~ 1 + xx

Coefficients:
─────────────────────────────────────────────
                  Coef.  Std. Error      t  Pr(>|t
─────────────────────────────────────────────
(Intercept)  20.0337      0.544062  36.82     <1e-3
xx            5.01957     0.226965  22.12     <1e-2
─────────────────────────────────────────────
```

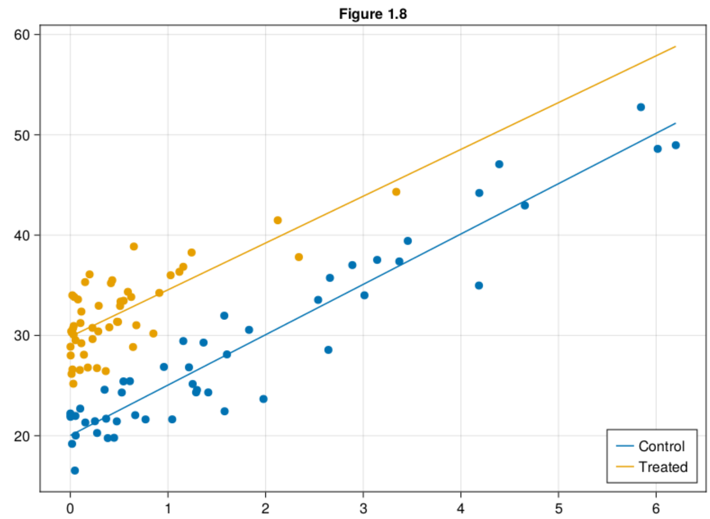
```
lm1_8_0 = lm(@formula(yy ~ xx),
df1_8[df1_8.z .== 0, :])
```

```
lm1_8_1 =
StatsModels.TableRegressionModel{LinearModel{GLM…

yy ~ 1 + xx

Coefficients:
─────────────────────────────────────────────
                  Coef.  Std. Error      t  Pr(>|t
─────────────────────────────────────────────
(Intercept)  29.8841      0.49051   60.92     <1e-4
xx            4.66553     0.609796   7.65     <1e-0
─────────────────────────────────────────────
```

```
lm1_8_1 = lm(@formula(yy ~ xx),
df1_8[df1_8.z .== 1, :])
```




```
let
    â₁, b̂₁ = coef(lm1_8_0)
    â₂, b̂₂ = coef(lm1_8_1)
    x = range(0, maximum(df1_8.xx),
    length=40)

    f = Figure()
    ax = Axis(f[1, 1]; title="Figure 1.8")
    scatter!(df1_8.xx[df1_8.z .== 0],
    df1_8.yy[df1_8.z .== 0])
    scatter!(df1_8.xx[df1_8.z .== 1],
    df1_8.yy[df1_8.z .== 1])
    lines!(x, â₁ .+ b̂₁ * x, label =
    "Control")
    lines!(x, â₂ .+ b̂₂ * x, label =
    "Treated")
    axislegend(; position=(:right, :bottom))
    current_figure()
end
```

1.5 Classical and Bayesian inference.

No code.

1.6 Computing least-squares and Bayesian regression.

No code.

1.8 Exercises.

Helicopters

helicopters =

	Helicopter_ID	width_cm	length_cm	time_s
1	1	4.6	8.2	1.64
2	1	4.6	8.2	1.74
3	1	4.6	8.2	1.68
4	1	4.6	8.2	1.62
5	1	4.6	8.2	1.68
6	1	4.6	8.2	1.7
7	1	4.6	8.2	1.62
8	1	4.6	8.2	1.66
9	1	4.6	8.2	1.69
10	1	4.6	8.2	1.62
⋮ more				
20	2	4.6	8.2	1.61

```julia
helicopters =
CSV.read(ros_datadir("Helicopters",
"helicopters.csv"), DataFrame)
```

Simulate 40 helicopters.

	width_cm	length_cm	time_sec
1	7.13607	15.7472	1.8204
2	6.10818	7.01771	1.33316
3	5.31182	12.3115	1.85854
4	4.76825	2.28141	0.874946
5	5.06939	7.22618	1.39635
6	5.86893	3.15562	1.10142
7	3.81515	10.34	1.41662
8	7.35128	4.75261	1.20684
9	3.17521	15.8526	2.02466
10	5.73403	16.011	1.87681
⋮ more			
40	9.53975	2.3966	1.03466

```
begin
    helis = DataFrame(width_cm =
    rand(Normal(5, 2), 40), length_cm =
    rand(Normal(10, 4), 40))
    helis.time_sec = 0.5 .+ 0.04 .*
    helis.width_cm .+ 0.08 .*
    helis.length_cm .+ 0.1 .*
    rand(Normal(0, 1), 40)
    helis
end
```

Simulate 40 helicopters.

```
ppl1_4 (generic function with 2 methods)
```

```
@model function ppl1_4(w, l, y)
    a ~ Normal(10, 5)
    b ~ Normal(0, 5)
    c ~ Normal(0, 5)
    σ ~ Exponential(1)
    μ = a .+ b .* w .+ c .* l
    for i in eachindex(y)
        y[i] ~ Normal(μ[i], σ)
    end
end
```

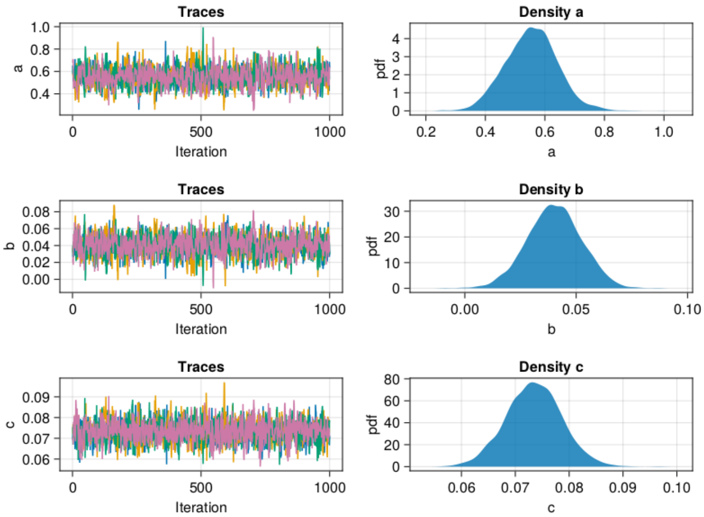
	parameters	mean	std	naiv
1	:a	0.554994	0.0868273	0.0013
2	:b	0.0401721	0.0122261	0.0001
3	:c	0.0734549	0.00514703	8.1381
4	:σ	0.132477	0.0159901	0.0002

```
begin
    m1_4t = ppl1_4(helis.width_cm,
    helis.length_cm, helis.time_sec)
    chns1_4t = sample(m1_4t, NUTS(),
    MCMCThreads(), 1000, 4)
    describe(chns1_4t)
end
```

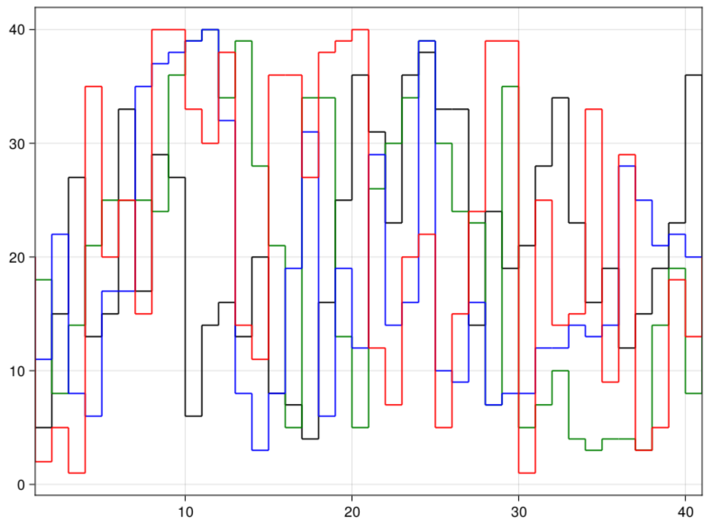
	parameters	median	mad_sd	mean	st
1	"a"	0.556	0.084	0.555	0.08
2	"b"	0.04	0.012	0.04	0.01
3	"c"	0.073	0.005	0.073	0.00
4	"σ"	0.131	0.015	0.132	0.01

```
begin
    post1_4t = DataFrame(chns1_4t[[:a, :b,
    :c, :σ]])
    ms1_4t = model_summary(post1_4t, [:a,
    :b, :c, :σ])
end
```

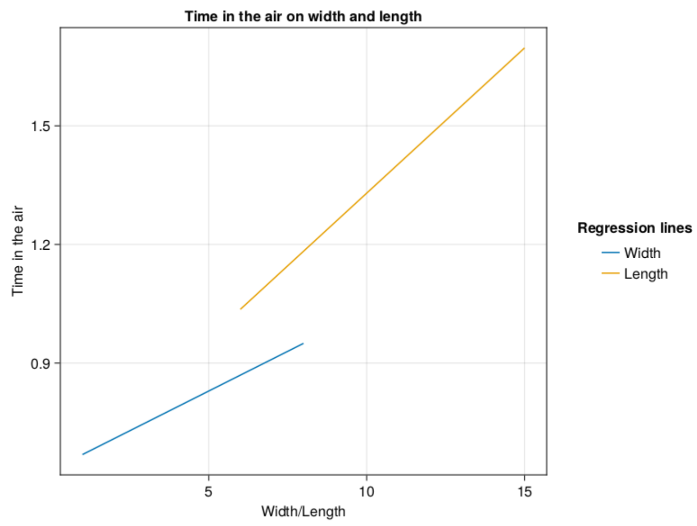
```
ms1_4t[:b, :media]
```




```
plot_chains(post1_4t, [:a, :b, :c])
```



```
trankplot(post1_4t, "b")
```




```
let
    w_range = LinRange(1.0, 8.0, 100)
    w_times = mean.(link(post1_4t, (r, w) -
    > r.a + r.c + r.b * w, w_range))

    l_range = LinRange(6.0, 15.0, 100)
    l_times = mean.(link(post1_4t, (r, l) -
    > r.a + r.b + r.c * l, l_range))

    f = Figure()
    ax = Axis(f[1, 1], title = "Time in the
    air on width and length",
        xlabel = "Width/Length", ylabel =
        "Time in the air")

    lines!(w_range, w_times; label="Width")
    lines!(l_range, l_times; label="Length")

    f[1, 2] = Legend(f, ax, "Regression
    lines", framevisible = false)

    current_figure()
end
```

Note

Note that the `link` function is defined in both RegeressionAndOtherStories (ROS) and Turing. In this case I added the `import` statement at the top of this notebook but I could also have qualified the call to link (`ROS.link`).

lnk1_4t =

▶ [[1.02944, 0.977409, 0.922584, 1.00679, 1.00253

```
lnk1_4t = link(post1_4t, (r, l) -> r.a + r.b
  + r.c * l, [5, 10,12])
```

▶ [0.963863, 1.32978, 1.4764]

```
median.(lnk1_4t)
```

▶ [0.0603975, 0.0537825, 0.0562974]

```
mad.(lnk1_4t)
```

▶ [0.962441, 1.32972, 1.47663]

```
mean.(lnk1_4t)
```