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 ✓ , DrWatson ✓
```

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

    # Specific to ROSStanPluto
    using StanSample ✓

    # Graphics related
    using CairoMakie ✓
    using AlgebraOfGraphics ✓

    # Include basic packages
    using RegressionAndOtherStories ✓
end
```

```
Replacing docs for `RegressionAndOtherStories.
rames.DataFrame, AbstractString}` in module `R
`
```

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_candidate
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"

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

hibbs_lm =

```
StatsModels.TableRegressionModel{LinearModel{Gl

vote ~ 1 + growth

Coefficients:
─────────────────────────────────────────────
                 Coef.  Std. Error      t  Pr(>|
─────────────────────────────────────────────
(Intercept)  46.2476     1.62193    28.51    <1e
growth        3.06053    0.696274    4.40    0.0
─────────────────────────────────────────────
```

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

```
▶ [-8.99292, 2.66743, 1.0609, 2.20753, -5.89044,
```

```julia
residuals(hibbs_lm)
```

2.2744434224582912

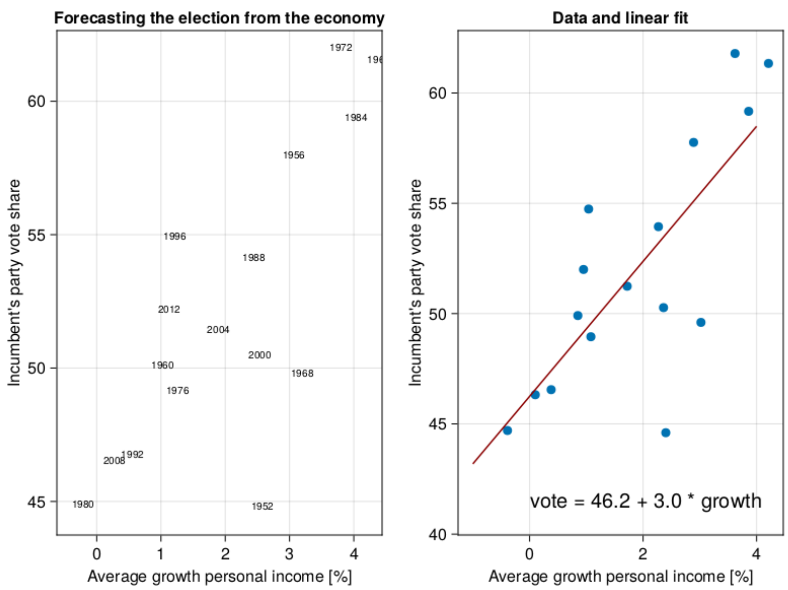
```
mad(residuals(hibbs_lm))
```

3.635681268522063

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

[46.2476, 3.06053]

```
coef(hibbs_lm)
```



```
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
```

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`.

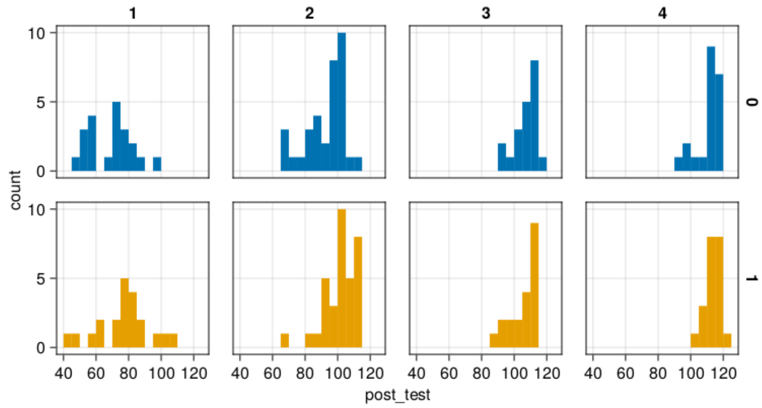
	variable	mean	min	median	max
1	:post_test	97.1495	44.2	102.3	122.0
2	:pre_test	72.2245	8.8	80.75	119.8
3	:grade	nothing	1	nothing	4
4	:treatment	nothing	0	nothing	1

```
describe(electric)
```

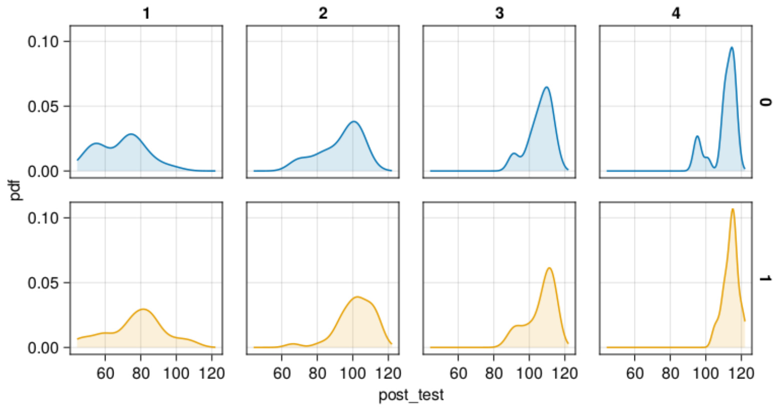
```
true
```

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

Post-test density for each grade conditioned on treatment.



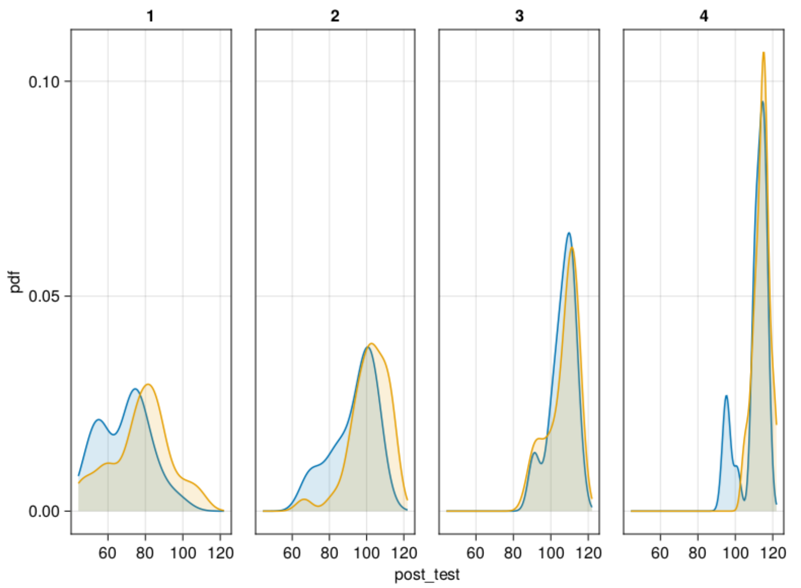

```
let
    f = Figure()
    axis = (; width = 150, height = 150)
    el = data(electric) *
    mapping(:post_test, col=:grade,
    color=:treatment)
    plt = el *
    AlgebraOfGraphics.histogram(;bins=20)
    * mapping(row=:treatment)
    draw!(f[1, 1], plt; axis)
    f
end
```



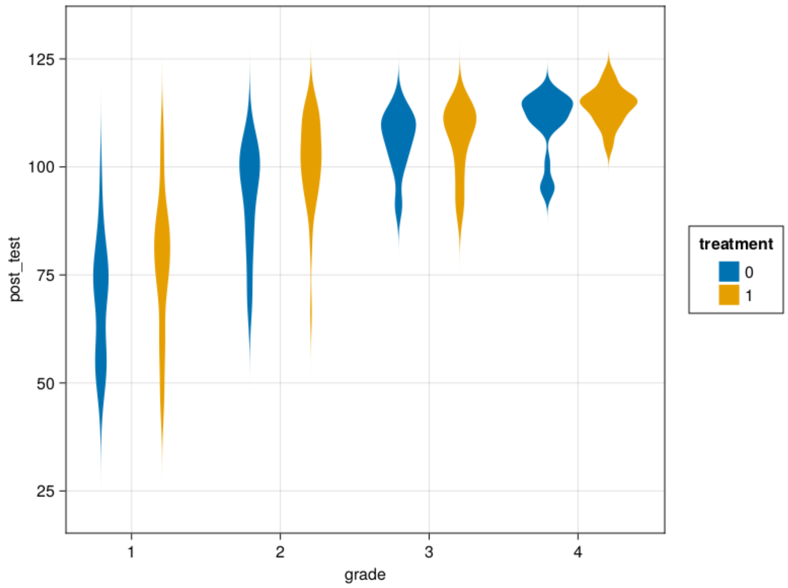

```
let
    f = Figure()
    axis = (; width = 150, height = 150)
    el = data(electric) *
    mapping(:post_test, col=:grade,
    color=:treatment)
    plt = el * AlgebraOfGraphics.density()
    * mapping(row=:treatment)
    draw!(f[1, 1], plt; axis)
    f
end
```

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-Mujal
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

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

1992-04-25T00:00:00

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

107 days

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

1970-01-01

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

1992-04-25

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

8150 days

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

107

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

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

0.4166666666666667

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

64

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

0.5588235294117647

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

0.3387096774193548

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

1.382

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

1.5153658536585364

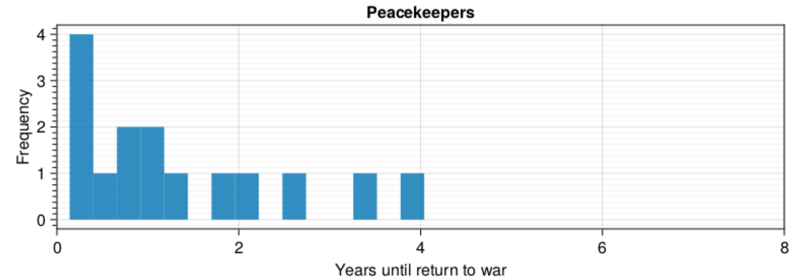
```
mean(peace[peace.peacekeepers .== 0 .&&
peace.censored .== 0, :delay])
```

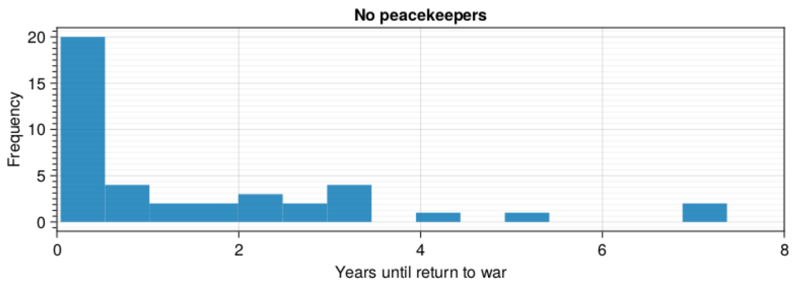
1.05

```
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)
    end
    f
end
```

Note

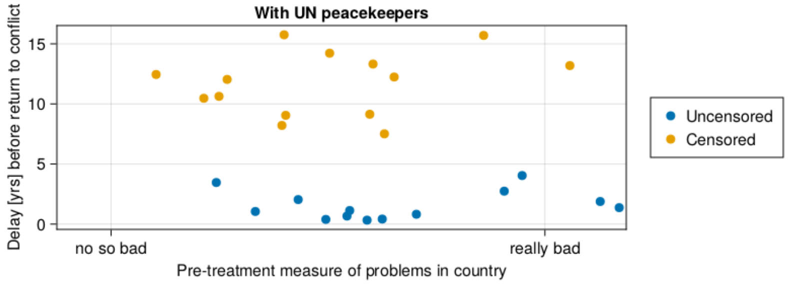
Censored means conflict had not returned until end of observation period (2004).

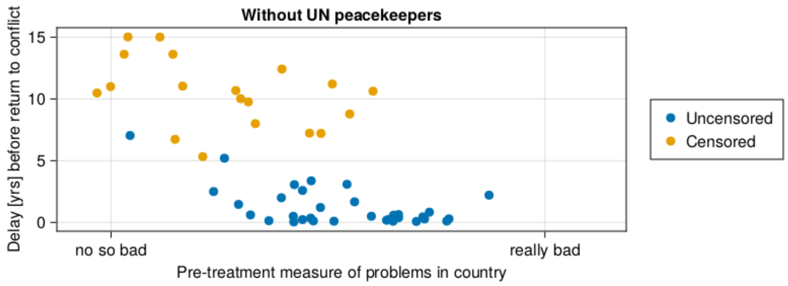
```
begin
    # Filter out missing badness rows.
    pb = peace[peace.badness .!== missing,
    :];

    # 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;
```


With UN peacekeepers
Delay [yrs] before return to conflict
15
10
5
0
no so bad
really bad
Pre-treatment measure of problems in country
Uncensored
Censored


Without UN peacekeepers
Delay [yrs] before return to conflict
15
10
5
0
no so bad
really bad
Pre-treatment measure of problems in country
Uncensored
Censored

```
begin
    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

Note

In models like below I usually prefer to create 2 separate Stan Language models, one for the continuous case and another for the binary case. But they can be combined in a single model as shown below. I'm using this example to show one way to handle vectors returned from Stan's cmdstan.

```
stan1_4_1 = "
data {
    int N;
    vector[N] x;
    vector[N] x_binary;
    vector[N] y;
}
parameters {
    vector[2] a;
    vector[2] b;
    vector<lower=0>[2] sigma;
}
model {
    // Priors
    a ~ normal(10, 10);
    b ~ normal(10, 10);
    sigma ~ exponential(1);
    // Likelihood
    y ~ normal(a[1] + b[1] * x, sigma[1]);
    y ~ normal(a[2] + b[2] * x_binary,
    sigma[2]);
}
";
```

Note

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

```
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;
```

	parameters	mean	mcse	std	5'
1	"a[1]"	9.4	0.027	1.4	7.1
2	"a[2]"	16.0	0.013	0.69	15.(
3	"b[1]"	3.3	0.0084	0.43	2.5
4	"b[2]"	7.0	0.019	1.0	5.3
5	"sigma[1]"	3.5	0.0058	0.35	2.9
6	"sigma[2]"	3.7	0.0064	0.37	3.1

```
let
    data = (N = n, x = x, x_binary =
    x_binary, y = y)
    global m1_4_1s = SampleModel("m1_4_1s",
    stan1_4_1);
    global rc1_4_1s = stan_sample(m1_4_1s;
    data)
    success(rc1_4_1s) && describe(m1_4_1s)
end
```

```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gr
n updated.
```

Note

This is a good point to take a quick look at Pluto cell metadata: the top left `eye` symbol and the top right `3-dots in a circle` glyph (both only visible when the curser is in the input cell). Both are used quite often in these notebooks. Try them out!

The output of above method of the function `model_summary(::SampleModel)`, called directly on a SampleModel, is different from method `model_summary(::DataFrame)`, typically used later on. Above table shows important mcmc diagnostic columns like `n_eff` and `r_hat`.

If Stan parameters are vectors (as in this example), cmdstan returns those using '.' notation, e.g. a.1, a.2, ...

	parameters	median	mad_sd	mean	st
1	"a.1"	9.387	1.396	9.368	1.39
2	"a.2"	16.15	0.652	16.133	0.69
3	"b.1"	3.255	0.438	3.252	0.43
4	"b.2"	6.969	1.04	6.972	1.04
5	"sigma.1"	3.43	0.336	3.461	0.34
6	"sigma.2"	3.658	0.366	3.676	0.37

```
if success(rc1_4_1s)
    post1_4_1s = read_samples(m1_4_1s,
    :dataframe)
    model_summary(post1_4_1s,
    names(post1_4_1s))
end
```

With vector parameters `read_samples()` can create a nested DataFrame:

nd1_4_1s =

	a	b
1	[10.3282, 14.1843]	[2.75128, 8.3895
2	[10.7021, 16.067]	[2.61631, 8.9163
3	[6.66245, 17.2466]	[4.09877, 7.9034
4	[8.92833, 16.0098]	[3.59349, 7.9635
5	[8.26656, 16.2145]	[3.5647, 7.7423]
6	[7.95563, 16.0481]	[3.72812, 7.0411
7	[8.45144, 16.6981]	[3.47334, 6.9575
8	[11.4261, 14.7333]	[2.83714, 7.5298
9	[10.6172, 15.1755]	[2.957, 7.7749]
10	[8.84695, 15.9684]	[3.54249, 7.9980
⋮ more		
4000	[10.6575, 17.1363]	[2.87471, 6.0266

```julia
nd1_4_1s = read_samples(m1_4_1s,
:nesteddataframe)
```

ms1_4_1s =

	parameters	median	mad_sd	mean	st
1	"a.1"	9.387	1.396	9.368	1.39
2	"a.2"	16.15	0.652	16.133	0.69
3	"b.1"	3.255	0.438	3.252	0.43
4	"b.2"	6.969	1.04	6.972	1.04
5	"sigma.1"	3.43	0.336	3.461	0.34
6	"sigma.2"	3.658	0.366	3.676	0.37

```
ms1_4_1s = success(rc1_4_1s) &&
model_summary(post1_4_1s,
names(post1_4_1s))
```

1.04

```
ms1_4_1s["b.2", "mad_sd"]
```

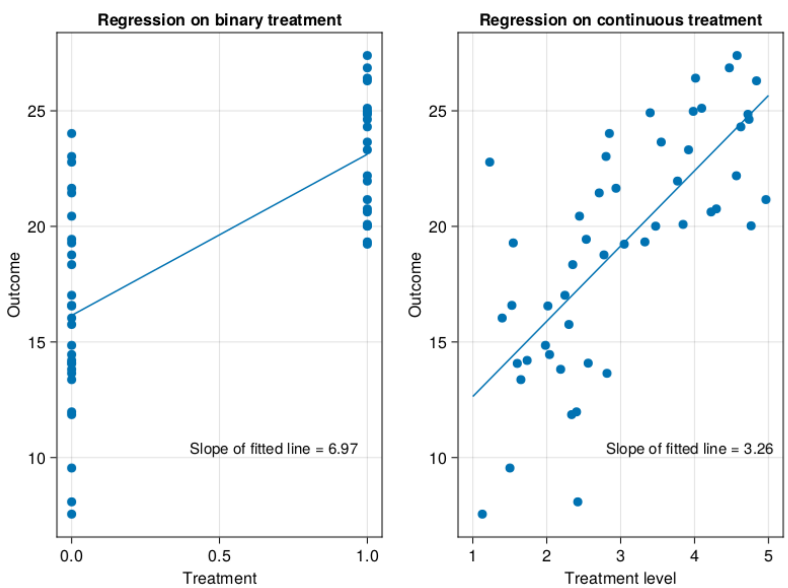
Nested dataframes are handy to obtain a matrix of say the b values:

```
4000×2 Matrix{Float64}:
 2.75128  8.38958
 2.61631  8.9163
 4.09877  7.90345
 3.59349  7.96354
 3.5647   7.7423
 3.72812  7.04112
 3.47334  6.95756
 ⋮
 2.79654  4.78503
 2.75535  9.23017
 3.33143  7.67702
 2.83813  6.82789
 2.86538  4.61493
 2.87471  6.0266
```

```
array(nd1_4_1s, :b)
```

```
4000×2 Matrix{Float64}:
 2.75128  8.38958
 2.61631  8.9163
 4.09877  7.90345
 3.59349  7.96354
 3.5647   7.7423
 3.72812  7.04112
 3.47334  6.95756
 ⋮
 2.79654  4.78503
 2.75535  9.23017
 3.33143  7.67702
 2.83813  6.82789
 2.86538  4.61493
 2.87471  6.0266
```

```
Array(post1_4_1s[:, ["b.1", "b.2"]])
```



```
let
    x1 = 1.0:0.01:5.0
    f = Figure()
    medians = ms1_4_1s[:, "median"]
    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[3], digits=2))",
        position = (2.8, 10), textsize=15)
    lin1 = lines!(x1, medians[1] .+
    medians[3] * x1)

    x2 = 0.0:0.01:1.0
    ax = Axis(f[1, 1], title="Regression
    on binary treatment",
        xlabel = "Treatment", ylabel =
        "Outcome")
    sca1 = scatter!(x_binary, y)
    lin1 = lines!(x2, medians[2] .+
    medians[4] * x2)
    annotations!("Slope of fitted line =
    $(round(medians[4], digits=2))",
        position = (0.4, 10), textsize=15)
    f
end
```

```
stan1_4_2 = "
data {
    int N;
    vector[N] x;
    vector[N] y;
}
parameters {
    vector[2] a;
    real b;
    real b_exp;
    vector<lower=0>[2] sigma;
}
model {
    // Priors
    a ~ normal(10, 5);
    b ~ normal(0, 5);
    b_exp ~ normal(5, 5);
    sigma ~ exponential(1);
    // Likelihood
    vector[N] mu;
    for ( i in 1:N )
        mu[i] = a[2] + b_exp * exp(-x[i]);
    y ~ normal(mu, sigma[2]);
    y ~ normal(a[1] + b * x, sigma[1]);
}
";
```

	parameters	mean	mcse	std	5(
1	"a[1]"	15.0	0.02	0.96	14.(
2	"a[2]"	5.7	0.0078	0.41	5.1
3	"b"	-2.5	0.0	0.3	-3.(
4	"b_exp"	24.2	0.1	3.1	19.:
5	"sigma[1]"	2.4	0.0041	0.24	2.0
6	"sigma[2]"	2.2	0.0047	0.26	1.8

```
let
    #Random.seed!(1533)
    n1 = 50
    x1 = rand(Uniform(1, 5), n1)
    y1 = [rand(Normal(5 + 30exp(-x1[i]),
    2), 1)[1] for i in 1:n]
    data = (N = n1, x = x1, y = y1)
    global m1_4_2s = SampleModel("m1.4_2s",
    stan1_4_2);
    global rc1_4_2s = stan_sample(m1_4_2s;
    data)
    success(rc1_4_2s) && describe(m1_4_2s)
end
```

```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gr
n updated.
```

	parameters	median	mad_sd	mean	st
1	"a.1"	15.488	0.942	15.48	0.96
2	"a.2"	5.703	0.407	5.722	0.40
3	"b"	-2.492	0.285	-2.489	0.29
4	"b_exp"	24.354	3.014	24.248	3.05
5	"sigma.1"	2.379	0.231	2.399	0.24
6	"sigma.2"	2.209	0.253	2.231	0.25

```
if success(rc1_4_2s)
    post1_4_2s = read_samples(m1_4_2s,
    :dataframe)
    ms1_4_2s = model_summary(post1_4_2s,
    ["a.1", "a.2", "b", "b_exp",
    "sigma.1", "sigma.2"])
end
```

nd1_4_2s =

	b	b_exp	a
1	-2.48576	20.0849	[15.9861, 6.25252]
2	-2.48382	27.7827	[14.8627, 5.18205]
3	-2.51677	27.8339	[16.1262, 5.18631]
4	-2.63137	27.8789	[15.4167, 5.25809]
5	-2.52726	28.7973	[16.002, 4.68289]
6	-2.63422	29.3018	[15.6185, 5.16142]
7	-2.31383	22.5431	[14.7362, 5.84994]
8	-2.10616	29.9681	[14.4546, 5.23276]
9	-2.63249	29.1633	[15.8549, 5.28732]
10	-2.73308	27.838	[16.2086, 5.07141]
⋮ more			
4000	-2.31223	25.1546	[14.6061, 5.48579]

```
nd1_4_2s = read_samples(m1_4_2s,
:nesteddataframe)
```

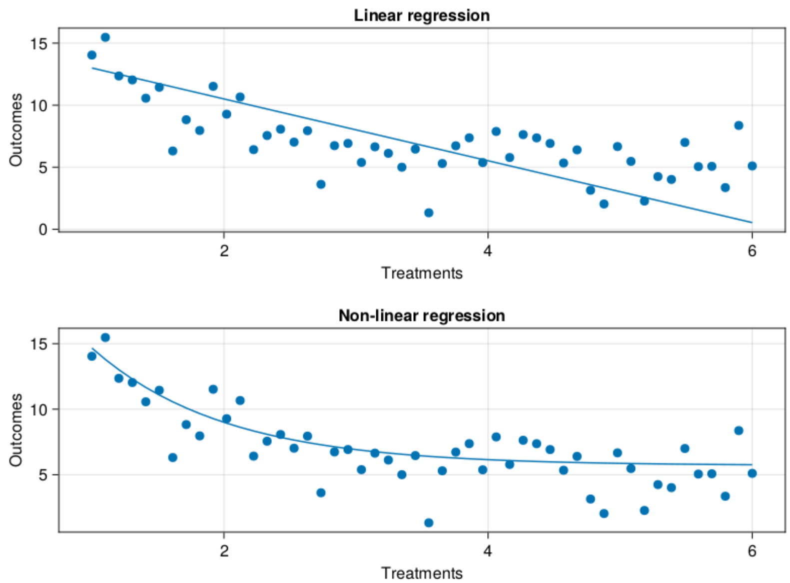
```
4000×2 Matrix{Float64}:
 15.9861  6.25252
 14.8627  5.18205
 16.1262  5.18631
 15.4167  5.25809
 16.002   4.68289
 15.6185  5.16142
 14.7362  5.84994
  ⋮
 14.423   6.11107
 16.5799  6.11235
 14.7391  5.71911
 14.2216  6.1607
 16.0556  5.70143
 14.6061  5.48579
```

```
array(nd1_4_2s, :a)
```

```
â₁, â₂, b̂, b̂ₑₓₚ, σ̂₁, σ̂₂ = [ms1_4_2s[p, "median"] for p in ["a.1", "a.2", "b", "b_exp", "sigma.1", "sigma.2"]];
```

5.703

```
â₂
```




```
let
    x1 = LinRange(1, 6, 50)
    y1 = [rand(Normal(5 + 30exp(-x1[i]), 2), 1)[1] for i in 1:length(x1)]

    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

```julia
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{Gl

yy ~ 1 + xx + z

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

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

```
lm1_8_0 =
StatsModels.TableRegressionModel{LinearModel{Gl

yy ~ 1 + xx

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

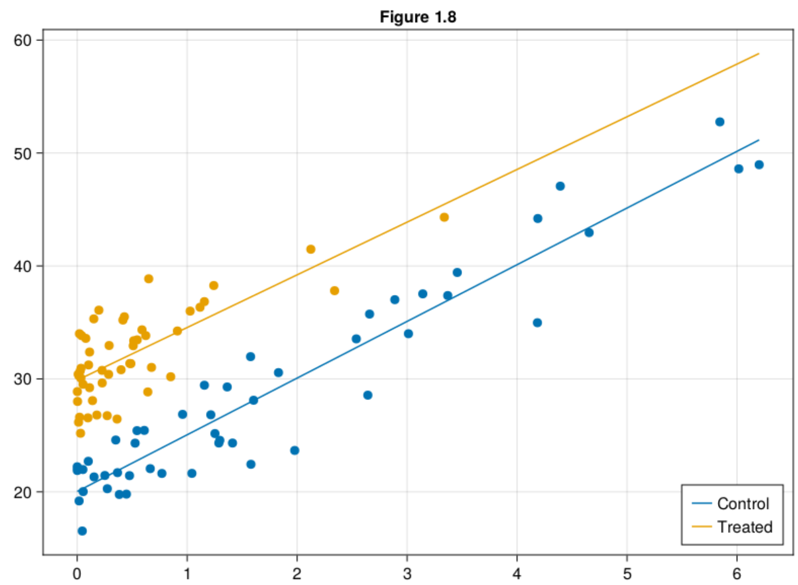
```
lm1_8_0 = lm(@formula(yy ~ xx),
df1_8[df1_8.z .== 0, :])
```

```
lm1_8_1 =
StatsModels.TableRegressionModel{LinearModel{Gl

yy ~ 1 + xx

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

```
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 = LinRange(0, maximum(df1_8.xx), 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.

1.6 Computing least-squares and Bayesian regression.

1.8 Exercises.

Helicopters

helicopters =

	Helicopter_ID	width_cm	length_cm	time_sec
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

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

Simulate 40 helicopters.

	width_cm	length_cm	time_sec
1	10.0236	9.59097	1.78566
2	2.33684	7.81096	1.41044
3	7.68879	2.34587	0.794504
4	6.67829	13.6578	1.81387
5	8.79925	9.27474	1.57939
6	2.40055	7.55651	1.2363
7	5.9089	16.177	2.04483
8	5.12956	14.864	1.8629
9	3.6735	15.5807	1.7891
10	2.18695	7.53194	1.15288
⋮ more			
40	8.39115	5.10451	1.32113

```
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
```

```
stan1_5 = "
data {
    int N;
    vector[N] w;
    vector[N] l;
    vector[N] y;
}
parameters {
    real a;
    real b;
    real c;
    real<lower=0> sigma;
}
model {
    // Priors
    a ~ normal(10, 5);
    b ~ normal(0, 5);
    sigma ~ exponential(1);

    // Likelihood time on width
    vector[N] mu;
    for ( i in 1:N )
        mu[i] = a + b * w[i] + c * l[i];
    y ~ normal(mu, sigma);
}
";
```

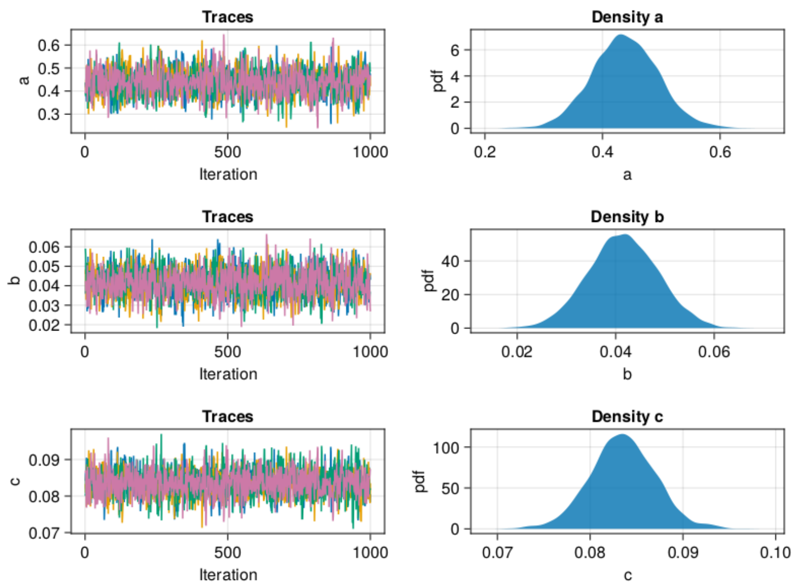
	parameters	mean	mcse	std
1	"a"	0.438333	0.00132571	0.05575
2	"b"	0.0415044	0.000159679	0.00701
3	"c"	0.0835149	7.1496e-5	0.00351
4	"sigma"	0.0946768	0.00026859	0.01124

```
let
    data = (N = nrow(helis), y =
    helis.time_sec, w = helis.width_cm, l
    = helis.length_cm)
    global m1_5s = SampleModel("m1.5s",
    stan1_5);
    global rc1_5s = stan_sample(m1_5s;
    data)
    success(rc1_5s) && describe(m1_5s)
end
```

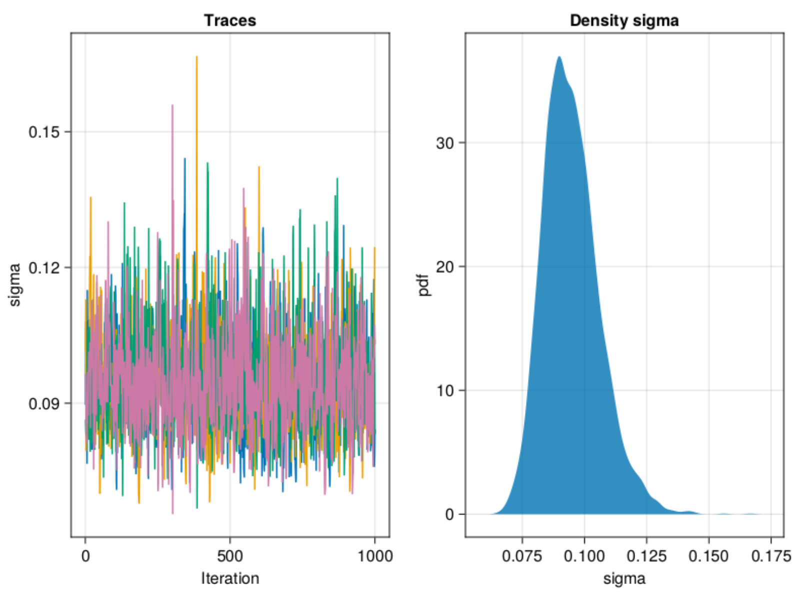
```
/var/folders/l7/pr04h0650q5dvqttnvs8s2c00000gr
updated.
```

	parameters	median	mad_sd	mean	st
1	"a"	0.4378	0.0554	0.4383	0.05
2	"b"	0.0415	0.007	0.0415	0.00
3	"c"	0.0835	0.0034	0.0835	0.00
4	"sigma"	0.0936	0.0108	0.0947	0.01

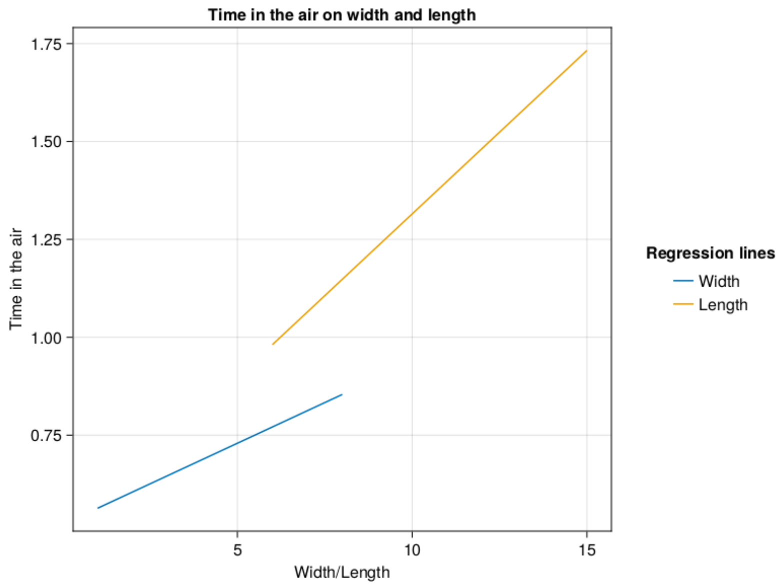
```
if success(rc1_5s)
    post1_5s = read_samples(m1_5s,
    :dataframe)
    model_summary(post1_5s, [:a, :b, :c,
    :sigma]; digits=4)
end
```




```
plot_chains(post1_5s, [:a, :b, :c])
```




```
plot_chains(post1_5s, [:sigma])
```




```
let
    w_range = LinRange(1.0, 8.0, 100)
    w_times = mean.(link(post1_5s, (r, w) -
    > r.a + r.c + r.b * w, w_range))
    l_range = LinRange(6.0, 15.0, 100)
    l_times = mean.(link(post1_5s, (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
```

```
lnk1_5s =
[[0.894902, 0.886343, 0.876715, 0.872754, 0.84
```

```
lnk1_5s = link(post1_5s, (r, l) -> r.a +
r.b + r.c * l, [5, 10,12])
```

[0.897544, 1.31378, 1.48082]

```
median.(lnk1_5s)
```

[0.0388104, 0.0333603, 0.0332957]

```
mad.(lnk1_5s)
```

[0.897412, 1.31499, 1.48202]

```
mean.(link(post1_5s, (r, l) -> r.a + r.b
+ r.c * l, [5, 10,12]))
```

	a	b	c	sigma
1	0.436716	0.0369489	0.0842475	0.08961
2	0.418862	0.0358816	0.0863198	0.10801
3	0.414789	0.051869	0.0820115	0.10764
4	0.390606	0.0437647	0.0876767	0.09686
5	0.35434	0.0420388	0.0905117	0.10267
6	0.404372	0.0515824	0.0810212	0.07660
7	0.512649	0.0268483	0.0845897	0.11502
8	0.46746	0.0448521	0.0787241	0.08613
9	0.448316	0.0290421	0.0883864	0.10002
10	0.397243	0.0549251	0.0808393	0.08290
⋮ more				
4000	0.465282	0.0268118	0.0873861	0.10298

```
read_samples(m1_5s, :nesteddataframe)
```

No nested columns found.