See chapter 5 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 ✓
```

A typical set of Julia packages to include in notebooks.

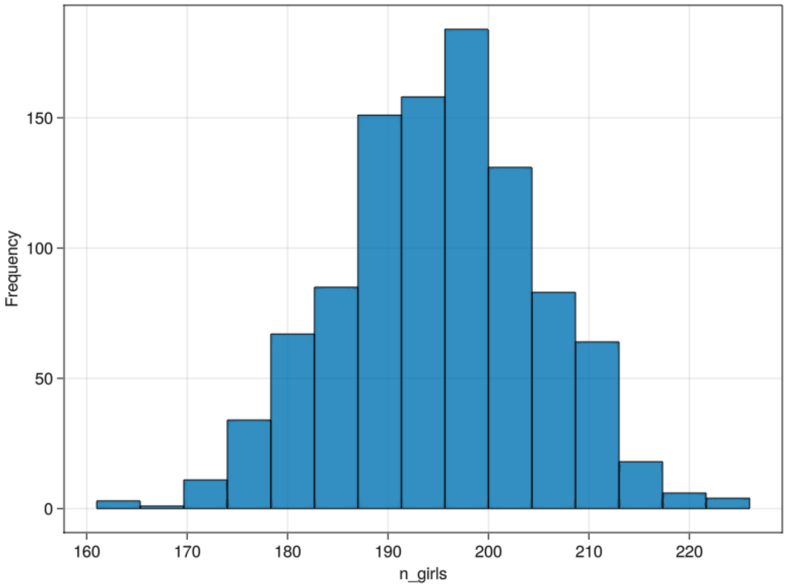
```
begin
    # Specific to this notebook
    using GLM ✓
    using PlutoUI ✓

    # Specific to ROSStanPluto
    using StanSample ✓

    # Graphics related
    using GLMakie ✓

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

5.1 Simulations of discrete events.

```
@bind nsim PlutoUI.Slider(2:5, default=3)
```

3

```
nsim
```



```
let
    f = Figure()
    ax = Axis(f[1, 1]; xlabel="n_girls",
    ylabel="Frequency")
    n_girls = rand(Binomial(400, 0.488),
    10^nsim)
    hist!(n_girls; strokewidth = 1,
    strokecolor = :black)
    f
end
```

```
prob_girls (generic function with 1 method)
```

```julia
function prob_girls(bt)
    res = if bt == :single_birth
        rand(Binomial(1, 0.488), 1)
    elseif bt == :fraternal_twin
        2rand(Binomial(1, 0.495), 1)
    else
        rand(Binomial(2, 0.495), 1)
    end
    return res[1]
end
```

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

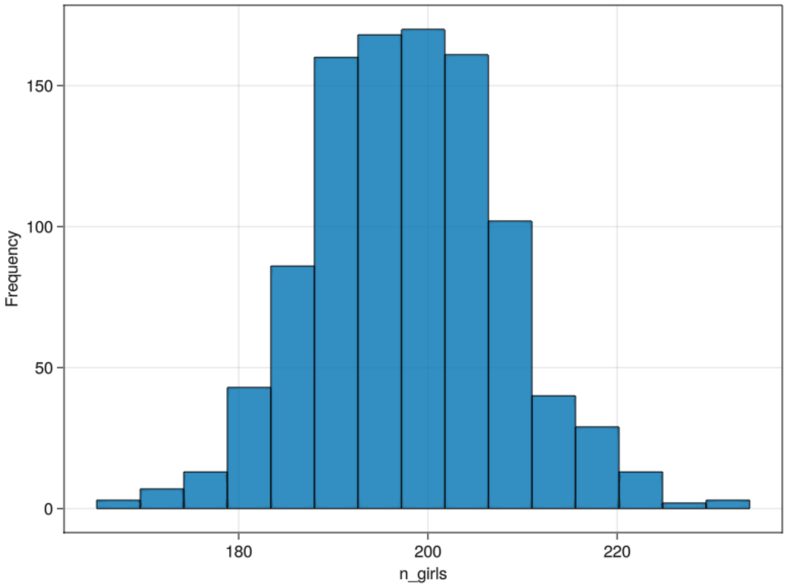
```julia
function girls(no_of_births = 400;
        birth_types = [:fraternal_twin,
        :identical_twin, :single_birth],
        probabilities = [1/125, 1/300, 1 -
        1/125 - 1/300])

    return prob_girls.(sample(birth_types,
    Weights(probabilities), no_of_births))
end
```

```
▶ [0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1
```

```julia
girls()
```

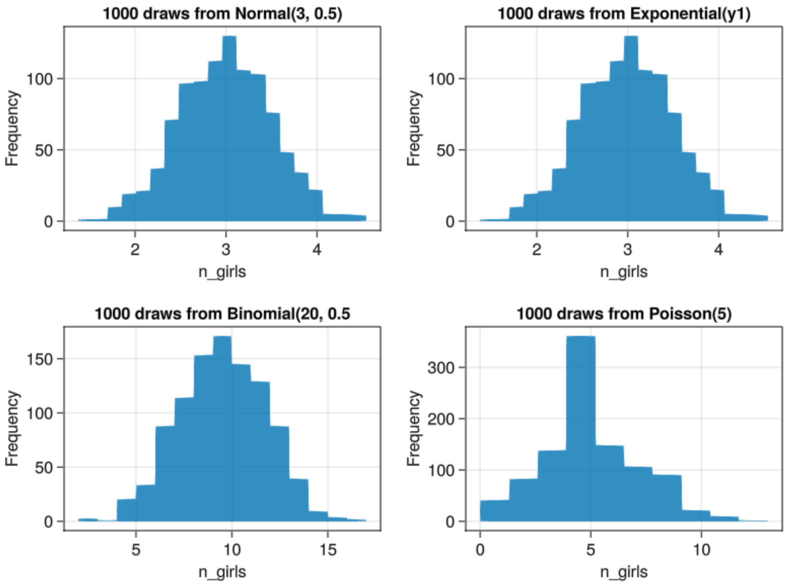
```
210
```

```julia
sum(girls())
```



```julia
let
    #Random.seed!(1)
    f = Figure()
    ax = Axis(f[1, 1]; xlabel="n_girls",
    ylabel="Frequency")
    girls_sim = [sum(girls()) for i in
    1:1000]
    hist!(f[1, 1], girls_sim; strokewidth
    = 1, strokecolor = :black,
    xlabel="Girls")
    f
end
```

5.2 Simulation of continuous and mixed/continuous models.



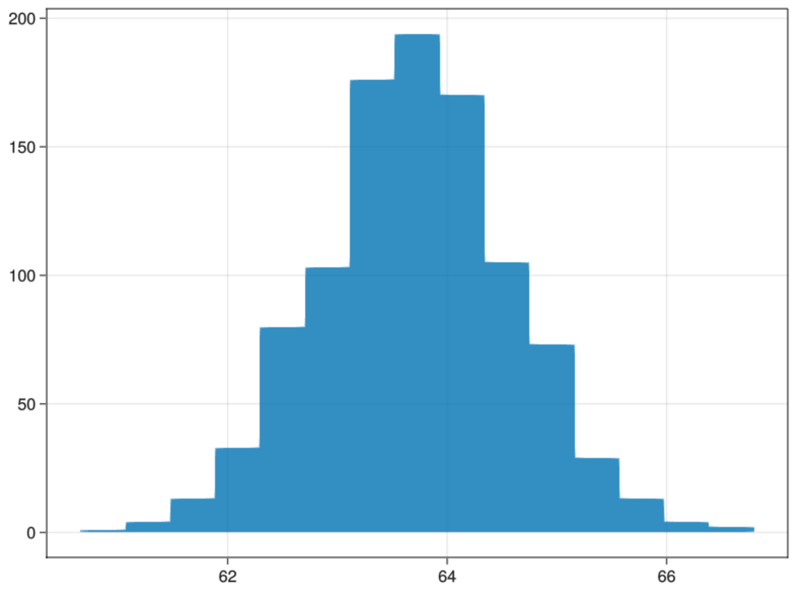
```
let
    n_sims = 1000
    y1 = rand(Normal(3, 0.5), n_sims)
    y2 = [Exponential(y1[i]).θ for i in
    1:length(y1)]
    y3 = rand(Binomial(20, 0.5), n_sims)
    y4 = rand(Poisson(5), n_sims)

    f = Figure()
    ax = Axis(f[1, 1]; title="1000 draws
    from Normal(3, 0.5)",
    xlabel="n_girls", ylabel="Frequency")
    hist!(y1; bins=20)
    ax = Axis(f[1, 2]; title="1000 draws
    from Exponential(y1)",
    xlabel="n_girls", ylabel="Frequency")
    hist!(y2; bins=20)

    ax = Axis(f[2, 1]; title="1000 draws
    from Binomial(20, 0.5",
    xlabel="n_girls", ylabel="Frequency")
    hist!(y3; bins=15)

    ax = Axis(f[2, 2]; title="1000 draws
    from Poisson(5)", xlabel="n_girls",
    ylabel="Frequency")
    hist!(y4; bins=10)
    f
end
```

```
sim (generic function with 1 method)
```

```
function sim()
    N = 10
    male = rand(Binomial(1, 0.48), N)
    height = male == 1 ? rand(Normal(69.1,
    2.9), N) : rand(Normal(63.7, 2.7), N)
    avg_height = mean(height)
end
```

```
64.19577422922887
```

```
sim()
```



```
let
    n_sim = 1000
    avg_height = Float64[]
    for i in 1:n_sim
        append!(avg_height, [sim()])
    end
    hist(avg_height)
end
```

5.3 Summarizing a set of simulations using median and median absolute deviation.

```
▶ (mean = 4.99, median = 4.97, std = 2.02, mad
```

```julia
let
    N = 10000
    z = rand(Normal(5, 2), N)
    vals = round.([mean(z), median(z),
    std(z), mad(z), 1.483 .* median(abs.(z
    .- median(z))), std(z)/sqrt(N)];
    digits=2)
    (mean=vals[1], median=vals[2],
    std=vals[3], mad=vals[4],
    mad_sd=vals[5], std_mean = vals[6])
end
```

Standard deviation of the mean:

```
▶ [3.64036, 6.32076]
```

```julia
quantile(rand(Normal(5, 2), 10000), [0.25,
0.75])
```

5.4 Bootstrapping to simulate a sampling distribution.

earnings =

	height	weight	male	earn	earnk
1	74	"210"	1	50000.0	50.0
2	66	"125"	0	60000.0	60.0
3	64	"126"	0	30000.0	30.0
4	65	"200"	0	25000.0	25.0
5	63	"110"	0	50000.0	50.0
6	68	"165"	0	62000.0	62.0
7	63	"190"	0	51000.0	51.0
8	64	"125"	0	9000.0	9.0
9	62	"200"	0	29000.0	29.0
10	73	"230"	1	32000.0	32.0
⋮ more					
1816	68	"150"	1	6000.0	6.0

```
earnings =
CSV.read(ros_datadir("Earnings",
"earnings.csv"), DataFrame)
```

ratio = 0.6

```
ratio = median(earnings[earnings.male .==
0, :earn]) /
median(earnings[earnings.male .== 1,
:earn])
```

take_df_sample (generic function with 1 method)

```
function take_df_sample(df, size; replace
= true, ordered = true)
    df[sample(axes(df, 1), size; replace,
ordered), :]
end
```

	height	weight	male	earn	earnk
1	62	"140"	0	10000.0	10.0
2	66	"160"	0	136500.0	136.5
3	63	"112"	0	33000.0	33.0

```
take_df_sample(earnings, 3)
```

```
boot_ratio (generic function with 1 method)
```

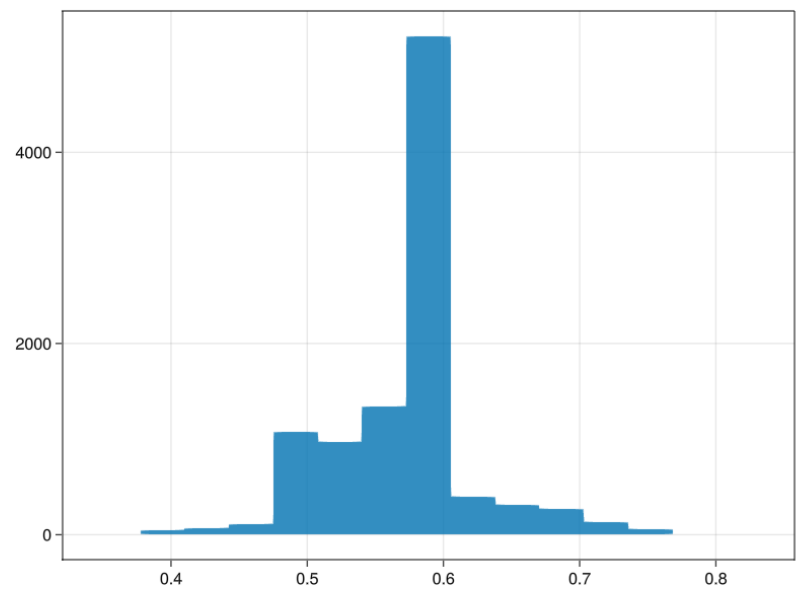
```
function boot_ratio(df::DataFrame,
sym::Symbol; draws=1000, replace=true)
    df = take_df_sample(df, draws; replace)
    ratio = median(df[df.male .== 0, sym])
    /  median(df[df.male .== 1, sym])
end
```

	height	weight	male	earn	earnk
1	65	"105"	0	18700.0	18.7
2	73	"130"	0	5000.0	5.0
3	67	"140"	0	15000.0	15.0
4	71	"170"	1	100000.0	100.0
5	71	"160"	1	25000.0	25.0
6	61	"235"	0	0.0	0.0
7	63	"126"	0	15000.0	15.0
8	66	"130"	0	1500.0	1.5
9	64	"180"	0	0.0	0.0
10	62	"140"	0	20000.0	20.0

```
take_df_sample(earnings, 10)
```

```
1.069767441860465
```

```
boot_ratio(earnings, :earn; draws=5)
```



```
let
    n_sims = 10000
    global boot_output =
    [boot_ratio(earnings, :earn;
    draws=500) for _ in 1:n_sims]
    hist(boot_output)
end
```

```
▶ [0.6, 0.681818, 0.6, 0.481481, 0.625, 0.681818
```

```
boot_output
```

```
0.05451447262602836
```

```
std(boot_output)
```

5.5 Fake-data simulations as a way of life.