Stitistical Rethinking Chapter 3

```
begin
    using StatsBase
    using Colors
    using PlutoUI
    using Distributions
    using CairoMakie
    using StatisticalRethinking
    using KernelDensity
end
```

code 3.1

0.08683729433272395

```
let
    Pr_Positive_Vampire = 0.95
    Pr_Positive_Mortal = 0.01
    Pr_Vampire = 0.001
    Pr_Positive = Pr_Positive_Vampire * Pr_Vampire + Pr_Positive_Mortal * (1 - Pr_Vampire)
    Pr_Vampire_Positive = Pr_Positive_Vampire * Pr_Vampire / Pr_Positive
end
```

1000

```
@bind slider_1 PlutoUI.Slider(collect(range(3, 1000)), show_value = true)
```

code 3.2

[0.0, 8.43366e-19, 5.38133e-17, 6.11125e-16, 3.42337e-15, 1.30198e-14, 3.87596e-14, 9.7442

```
begin
    size = slider_1

    # define grid
    p_grid = range(0, 1, size)

    # define prior
    prior = ones(size)

    # compute probability of data at each value in grid
    prob_data = pdf.(Binomial.(9, p_grid), 6)

    #compute product of prob_data and prior
    posterior = prob_data .* prior

    #standardize the posterior so it sums to 1
    posterior = posterior / sum(posterior)
end
```

code 3.3

samples =

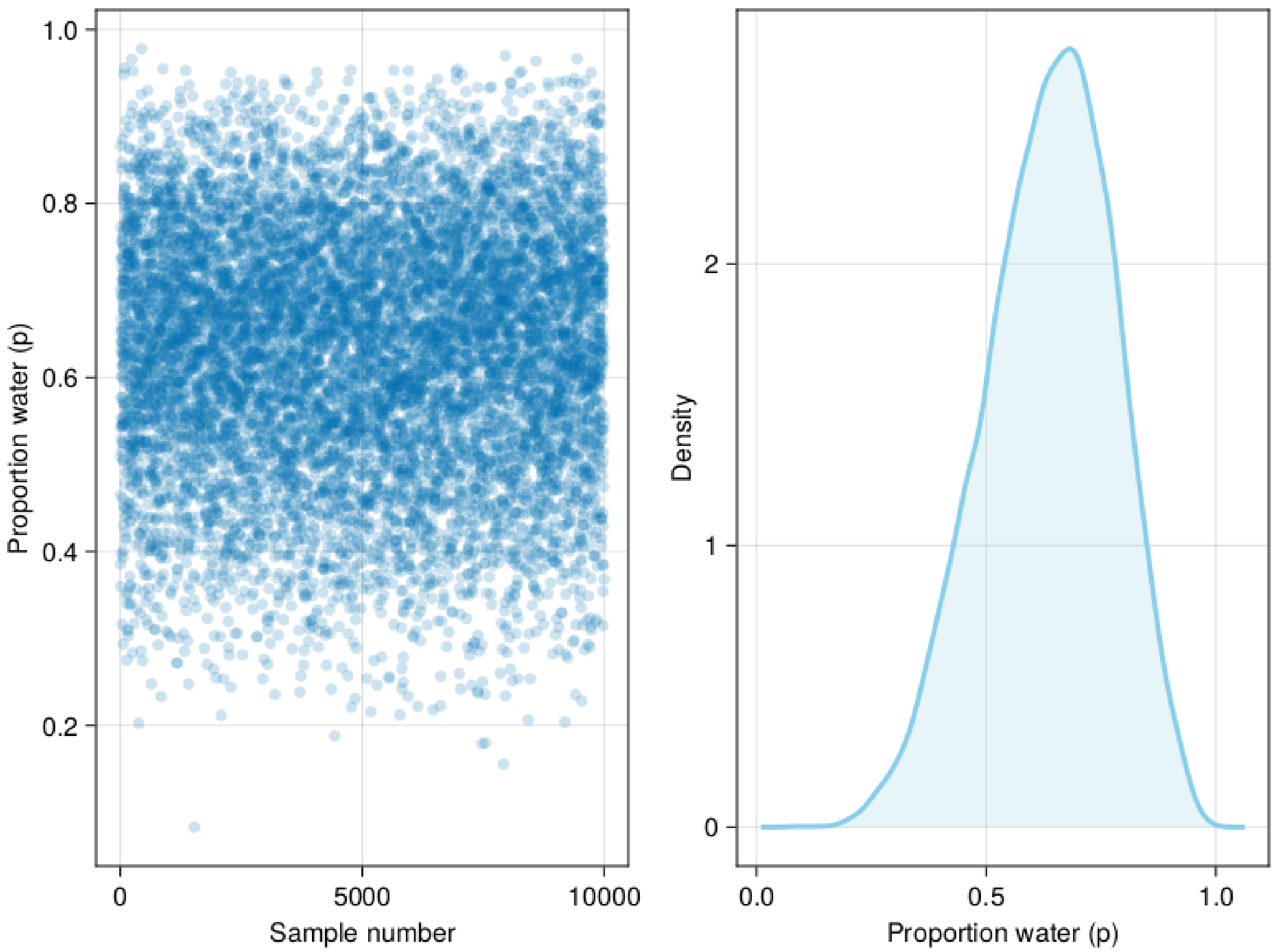
[0.474474, 0.376376, 0.6997, 0.532533, 0.361361, 0.85986, 0.77978, 0.507508, 0.661662, 0.5

```
samples = sample(p_grid, Weights(posterior), 10_000; replace=true)
```

jitter (generic function with 1 method)

```
# define the jitter function as is done in R programing language
# this funtion is simply adding noise to data so the points don't overlap when plotted
function jitter(x)
    z = findmax(collect(skipmissing(x)))[1] - findmin(collect(skipmissing(x)))[1]
    a = z/50
    if a == 0
        x = x .+ rand(length(x))
        return x
    else
        x = x .+ rand(Uniform(-a, a), length(x))
        return x
    end
end
```

code 3.4 and 3.5




```
let
    f = Figure()
    ax = Axis(f[1, 1]; xlabel="Sample number", ylabel="Proportion water (p)")
    scatter!(jitter(samples); alpha=0.2, markersize=10)
    ax = Axis(f[1, 2]; xlabel="Proportion water (p)", ylabel="Density")
    density!(jitter(samples); color=(:lightblue, 0.3), strokecolor=:skyblue,
    strokewidth = 3, strokearound = false)
    f

end
```

code 3.6

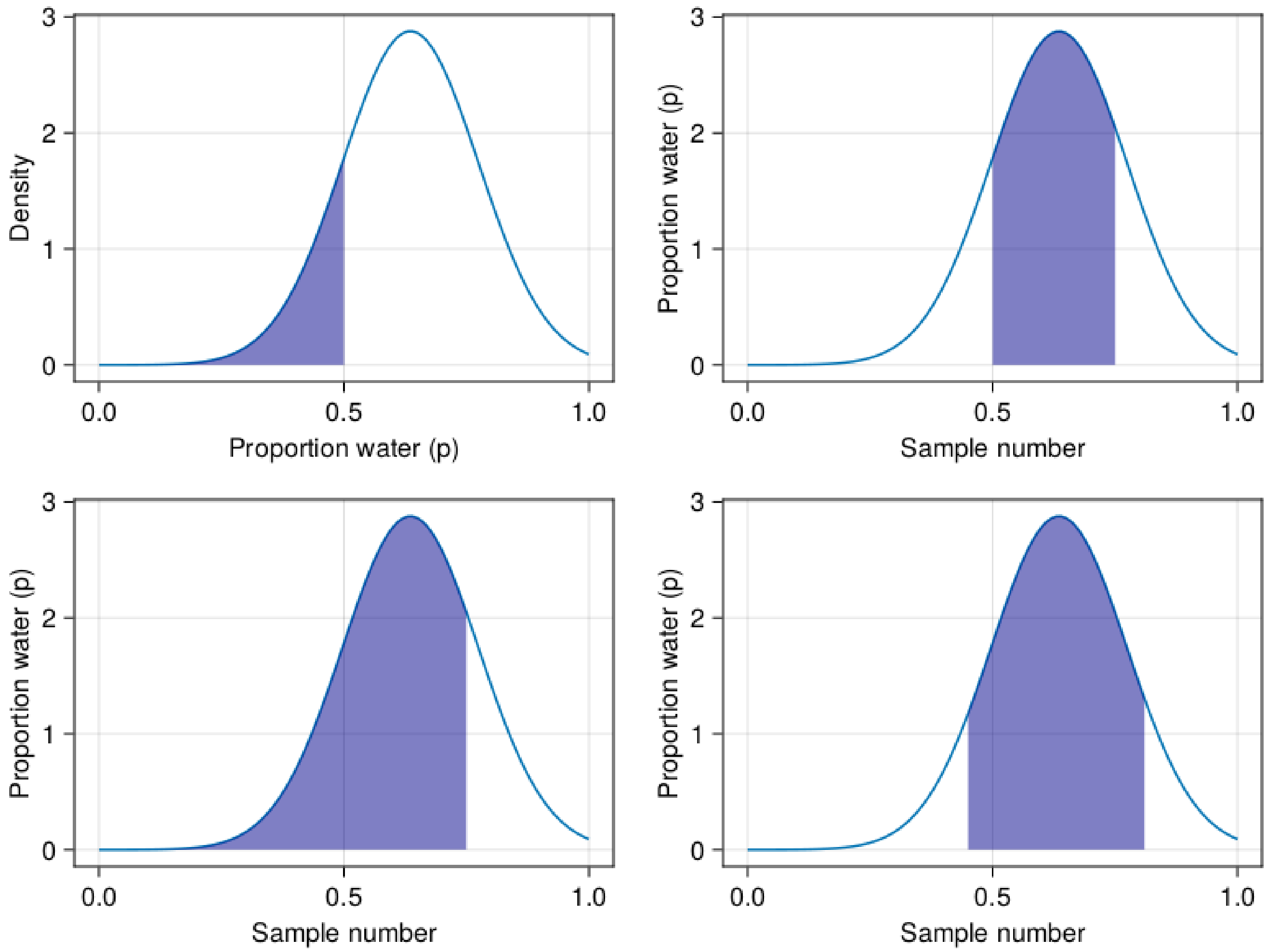
0.17187458902022873

```
# add up posterior probability where p < 0.5
sum(posterior[p_grid .< 0.5])
```

code 3.7

0.1718

```
# proportion of water less than 0.5 using samples
sum(samples .< 0.5) / length(samples)
```




```
let
    f = Figure()
    x = 0:0.01:1
    d = Normal(mean(samples), std(samples))

    ax = Axis(f[1, 1]; xlabel="Proportion water (p)", ylabel="Density")
    lines!(x, pdf.(d, x))
    x1 = range(0, 0.5; length=100)
    band!(x1, fill(0, length(x1)), pdf.(d, x1); color = (:darkblue, 0.5), label =
    "Label")

    ax = Axis(f[1, 2]; xlabel="Sample number", ylabel="Proportion water (p)")
    lines!(x, pdf.(d, x))
    x1 = range(0.5, 0.75; length=100)
    band!(x1, fill(0, length(x1)), pdf.(d, x1); color = (:darkblue, 0.5), label =
    "Label")

    ax = Axis(f[2, 1]; xlabel="Sample number", ylabel="Proportion water (p)")
    lines!(x, pdf.(d, x))
    x1 = range(0, 0.75; length=100)
    band!(x1, fill(0, length(x1)), pdf.(d, x1); color = (:darkblue, 0.5), label =
    "Label")

    ax = Axis(f[2, 2]; xlabel="Sample number", ylabel="Proportion water (p)")
    lines!(x, pdf.(d, x))
```

```
    x1 = range(0.45, 0.81; length=100)
    band!(x1, fill(0, length(x1)), pdf.(d, x1); color = (:darkblue, 0.5), label =
    "Label")
    f
```

code 3.8

0.6084

```
# asking how much posterior probability lies between 0.5 and 0.75 using samples
sum((samples .> 0.5) .& (samples .< 0.75)) / length(samples)
```

code 3.9

0.7587587587587588

```
quantile(samples, 0.8)
```

code 3.10

[0.446446, 0.808809]

```
quantile(samples, [0.1, 0.9])
```

code 3.11

[0.652653, 0.827828, 0.730731, 0.756757, 0.906907, 0.872873, 0.436436, 0.296296, 0.672673,

```
let
    size = 1000

    # define grid
    global p_grid2 = range(0, 1, size)

    # define prior
    prior = ones(size)

    # compute probability of data at each value in grid
    prob_data = pdf.(Binomial.(3, p_grid2), 3)

    #compute product of prob_data and prior
    global posterior2 = prob_data .* prior

    #standardize the posterior so it sums to 1
    posterior2 = posterior2 / sum(posterior2)

    global samples2 = sample(p_grid2, Weights(posterior2), 10_000; replace=true)

end
```

code 3.12

```
PI = [0.707708, 0.930931]
```

```julia
PI = percentile(samples2, [25, 75])
```

code 3.13

```
HPDI = [0.840841, 0.998999]
```

```julia
HPDI = hpdi(samples2, alpha=0.5)
```

code 3.14

```
1.0
```

```julia
p_grid[argmax(posterior2)]
```

code 3.15

```
0.9782665909681054
```

```julia
let
    k = kde(samples2, bandwidth=0.01)
    k.x[argmax(k.density)]
end
```

code 3.16

```
(0.799752, 0.841842)
```

```julia
mean(samples2), median(samples2)
```

code 3.17

```
(0.312875, 0.312875)
```

```julia
sum(posterior2 .* abs.( 0.5 .- p_grid)), sum(@. posterior2 * abs( 0.5 - p_grid))
```

code 3.18

```
loss =
[0.8004, 0.799399, 0.798398, 0.797397, 0.796396, 0.795395, 0.794394, 0.793393, 0.792392, 0
```

```julia
loss = map(d -> sum(@. posterior2 * abs(d - p_grid2)), p_grid2)
```

code 3.19

```
(0.840841, 0.841842)
```

```julia
p_grid2[argmin(loss)], median(samples2)
```

code 3.20

```
[0.09, 0.42, 0.49]
```

```julia
pdf.(Binomial.(2, 0.7), 0:2)
```

code 3.21

```
1
```

```julia
rand(Binomial(2, 0.7))
```

code 3.22

```
[2, 1, 1, 1, 2, 2, 1, 1, 0, 0]
```

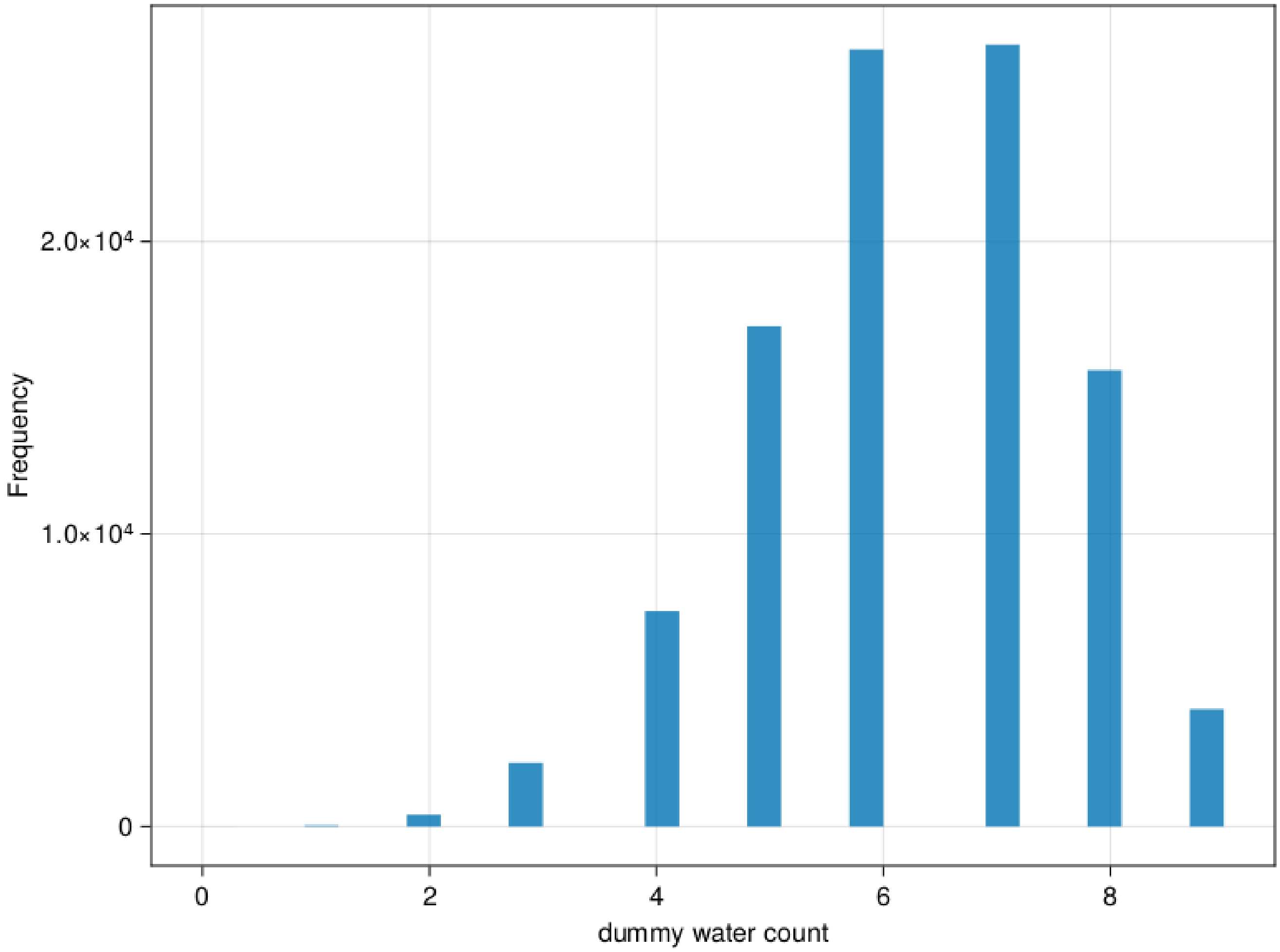
```julia
rand(Binomial(2, 0.7), 10)
```

code 3.23

```
[0.09213, 0.41957, 0.4883]
```

```julia
let
    dummy_w = rand(Binomial(2, 0.7), 100_000)
    proportions(dummy_w)  # or counts(dummy_w)/100000
end
```

code 3.24




```julia
let
    dummy_w = rand(Binomial(9, 0.7), 100_000)
    f = Figure()
    ax = Axis(f[1, 1]; xlabel="dummy water count", ylabel="Frequency")
    hist!(dummy_w; bins=30)
    f
end
```

code 3.25

```
[4, 3, 6, 5, 2, 6, 7, 8, 7, 5, 4, 7, 7, 4, 7, 4, 7, 5, 7, 6,    more ,6, 5, 3, 7, 5, 5, 7, 5,
```

```julia
let
    w = rand(Binomial(9, 0.6), 10_000)
end
```

code 3.26

```
w =
[5, 9, 8, 9, 7, 8, 3, 5, 6, 7, 5, 9, 8, 7, 7, 9, 9, 8, 7, 9,    more ,6, 9, 8, 9, 8, 8, 9, 6,
```

```julia
w = @. rand(Binomial(9, samples2))
```