

# Final Project

March 8, 2025

```
[210]: using DataFrames #for data wrangling
using StatsPlots #for plotting
using Turing #for MCMC
using CSV #CSV import
using Random #data generation (if required)
using Distributions
```

Import Data

```
[211]: #this is a local document, but i've also saved a slightly cleaned up version in
↳github with the same "valveTestRaw" title
valveData = CSV.read("./Results of New Valve Tests.csv", DataFrame)
```

	Vendor	Population	Rejected	High Pop	% Over Set	% Over Limit	Set Pressure	
	String1	Int64	Int64	Int64	String7	String7	Int64?	
1	a	9	2	1	10.00%	4.30%	35	...
2	b	13	0	0	0.00%	0.00%	missing	...
3	c	85	1	1	13.00%	9.70%	175	...
4	d	108	21	4	4.50%	1.50%	350	...
5	e	1	0	0	0.00%	0.00%	missing	...
6	f	55	9	2	5.50%	2.50%	400	...
7	g	41	5	2	6.70%	3.30%	165	...
8	h	2	0	0	0.00%	0.00%	missing	...
9	z	40	6	1	24.00%	20.00%	165	...
10	k	1	0	0	0.00%	0.00%	missing	...
11	y	66	4	4	6.00%	3.00%	100	...
12	m	54	6	1	27.00%	24.00%	75	...
13	m	14	7	3	10.00%	3.00%	80	...
14	n	1	0	0	0.00%	0.00%	missing	...
15	o	3	0	0	0.00%	0.00%	missing	...
16	p	3	0	0	0.00%	0.00%	missing	...
17	r	38	7	1	6.80%	4.00%	250	...
18	s	8	0	0	0.00%	0.00%	missing	...
19	x	18	2	1	6.30%	1.20%	175	...

Collect Number of Trials & Separate into Service Mediums

```
[212]: # vendors = (collect(valveData[:,1]))
# n=0
```

```

# for i in 1:length(vendors)
#     n = valveData[i, 2] + n
# end
# n

steamDF = []
airDF = []
liquidDF = []
airLiqDF = []

steamDF = filter(:Service == "steam", valveData)
airDF = filter(:Service == "air", valveData)
liquidDF = filter(:Service == "liquid", valveData)
airLiqDF = filter(:Service == "air/liquid", valveData)
naDF = filter(:Service == "na", valveData)

n_steam = 0
n_air = 0
n_liquid = 0
n_airLiq = 0
n_na = 0

steamVend = (collect(steamDF[:,1]))
for i in 1:length(steamVend)
    n_steam = steamDF[i,2] + n_steam
end

airVend = (collect(airDF[:,1]))
for i in 1:length(airVend)
    n_air = airDF[i,2] + n_air
end

liquidVend = (collect(liquidDF[:,1]))
for i in 1:length(liquidVend)
    n_liquid = liquidDF[i,2] + n_liquid
end

airLiqVend = (collect(airLiqDF[:,1]))
for i in 1:length(airLiqVend)
    n_airLiq = airLiqDF[i,2] + n_airLiq
end

naVend = (collect(naDF[:,1]))
for i in 1:length(naVend)
    n_na = naDF[i,2] + n_na
end

```

```
steamDF
```

	Vendor	Population	Rejected	High Pop	% Over Set	% Over Limit	Set Pressure	
	String1	Int64	Int64	Int64	String7	String7	Int64?	
1	a	9	2	1	10.00%	4.30%	35	...
2	c	85	1	1	13.00%	9.70%	175	...
3	g	41	5	2	6.70%	3.30%	165	...
4	z	40	6	1	24.00%	20.00%	165	...
5	k	1	0	0	0.00%	0.00%	missing	...
6	m	14	7	3	10.00%	3.00%	80	...

Set Up Model

```
[213]: @model function valveTesting(Vendors, Population, highPops) # rejects
    ↪serviceMedium)

    #additional functionality can be added to this, especially when considering
    ↪that the probability of failing (p-1) is the sum of the probability of
    ↪failing on a low pop (less dangerous) & a high pop (more dangerous)

    #hyper prior
    ~ Beta(4, 2)
    ~ Exponential(0.1)

    p = Vector{Real}(undef, length(Vendors))
    n = Vector{Real}(undef, length(Vendors))
    valvePass = Vector{Real}(undef, length(Vendors))
    for i in 1:length(Vendors)

        #prior
        n[i] = Population[i]
        valvePass[i] ~ Normal( , )
        #distribution of valves passing given pass probability p and n trials
        #valvePass[i] ~ Binomial(n[i], p[i])
        #valvePass[i] ~ Normal( , )

    end
end
```

valveTesting (generic function with 6 methods)

```
[214]: steam_model = valveTesting(steamDF[:,1], steamDF[:,2], steamDF[:,3])

steam_posterior = sample(steam_model, NUTS(), 1000)
```

```
Sampling 0% | ETA: N/A
Info: Found initial step size
      = 0.2
@ Turing.Inference
/Users/user/.julia/packages/Turing/NQDYt/src/mcmc/hmc.jl:207
```

```

Number of chains = 1
Samples per chain = 1000
Wall duration = 0.82 seconds
Compute duration = 0.82 seconds
parameters = , , valvePass[1], valvePass[2], valvePass[3], valvePass[4],
↳valvePass[5], valvePass[6]
internals = lp, n_steps, is_accept, acceptance_rate, log_density,
↳hamiltonian_energy, hamiltonian_energy_error, max_hamiltonian_energy_error,
↳tree_depth, numerical_error, step_size, nom_step_size

```

#### Summary Statistics

parameters	mean	std	mcse	ess_bulk	ess_tail	rhat
Symbol	Float64	Float64	Float64	Float64	Float64	Float64
	0.6261	0.1734	0.0204	72.4505	211.2309	1.0308
	0.1465	0.0947	0.0139	27.0439	20.8341	1.0076
valvePass[1]	0.6231	0.2538	0.0233	92.4972	299.8958	1.0054
valvePass[2]	0.6332	0.2526	0.0241	92.7321	294.9325	1.0327
valvePass[3]	0.6325	0.2277	0.0182	142.0857	340.5463	1.0065
valvePass[4]	0.6192	0.2393	0.0210	127.0519	236.7808	1.0239
valvePass[5]	0.6172	0.2344	0.0177	143.4016	386.8376	1.0189
valvePass[6]	0.6140	0.2536	0.0237	101.1608	437.3298	1.0277

1 column omitted

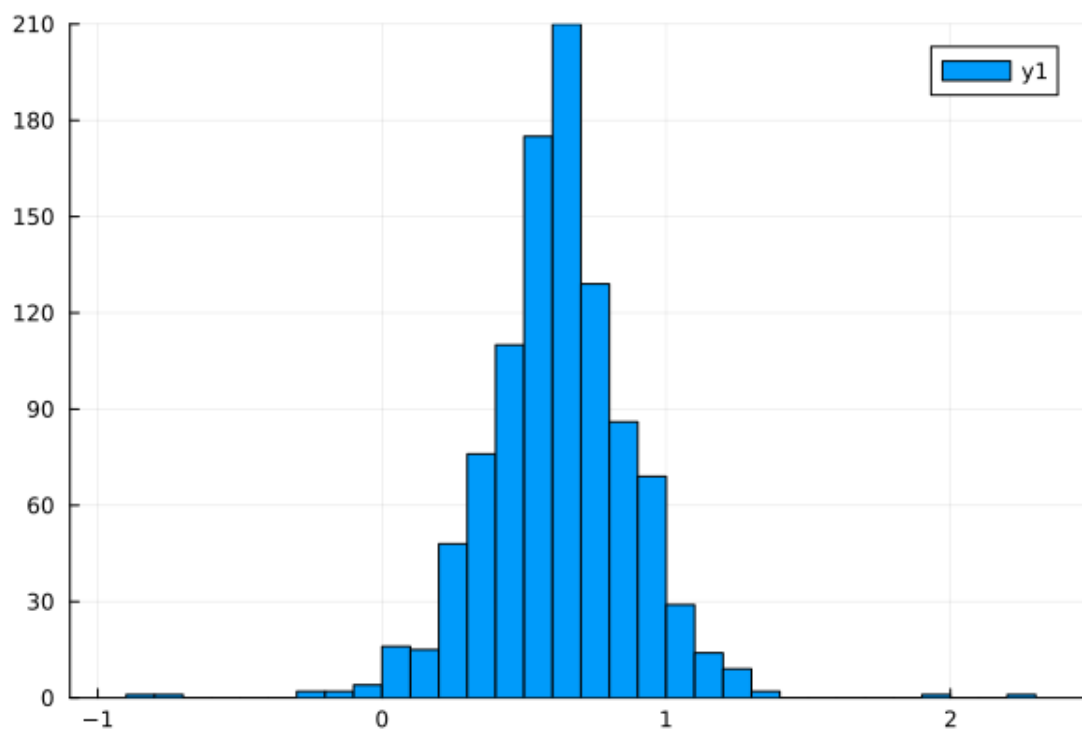
#### Quantiles

parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
	0.2821	0.5029	0.6416	0.7477	0.9312
	0.0450	0.0769	0.1195	0.1934	0.3898
valvePass[1]	0.0958	0.4840	0.6240	0.7656	1.1022
valvePass[2]	0.1325	0.4747	0.6419	0.7739	1.1489
valvePass[3]	0.1837	0.4939	0.6320	0.7523	1.1059
valvePass[4]	0.1396	0.4555	0.6274	0.7705	1.0622
valvePass[5]	0.1448	0.4896	0.6439	0.7500	1.0654
valvePass[6]	0.1153	0.4474	0.6180	0.7697	1.1252

```
[215]: postDF = DataFrame(steam_posterior)
```

	iteration	chain			valvePass[1]	valvePass[2]	valvePass[3]	
	Int64	Int64	Float64	Float64	Float64	Float64	Float64	
1	501	1	0.624363	0.101732	0.621401	0.676699	0.682054	...
2	502	1	0.638855	0.0943729	0.69544	0.636279	0.608389	...
3	503	1	0.631277	0.162388	0.805922	0.572874	0.499749	...
4	504	1	0.725277	0.17159	0.745444	0.902084	0.631404	...
5	505	1	0.768456	0.123373	0.746789	0.740318	0.923238	...
6	506	1	0.844632	0.143884	1.02434	0.79383	0.994382	...
7	507	1	0.876712	0.136271	0.646383	0.847353	0.821888	...
8	508	1	0.743402	0.123898	0.928348	0.650311	0.81534	...
9	509	1	0.746138	0.0714451	0.662507	0.851967	0.786254	...
10	510	1	0.953225	0.0889042	1.13765	0.936168	1.02415	...
11	511	1	0.919838	0.162016	0.703202	0.921017	0.699027	...
12	512	1	0.831178	0.131615	0.856382	0.89783	0.87989	...
13	513	1	0.839883	0.125637	0.932323	0.922863	0.707186	...
14	514	1	0.593754	0.140075	0.480825	0.50159	0.762223	...
15	515	1	0.765232	0.0961476	0.705133	0.791697	0.570825	...
16	516	1	0.724821	0.108031	0.771887	0.763367	0.647144	...
17	517	1	0.724821	0.108031	0.771887	0.763367	0.647144	...
18	518	1	0.724821	0.108031	0.771887	0.763367	0.647144	...
19	519	1	0.790564	0.101784	0.631645	0.93898	0.848137	...
20	520	1	0.817293	0.0784538	0.99265	0.833883	0.819455	...
21	521	1	0.833972	0.0912988	0.854417	0.773157	0.864878	...
22	522	1	0.833972	0.0912988	0.854417	0.773157	0.864878	...
23	523	1	0.842019	0.0916136	0.978067	0.838782	0.737009	...
24	524	1	0.627328	0.135182	0.515388	0.65372	0.704575	...
...	...	...	...	...	...	...	...	...

```
[216]: histogram(postDF[:, 5])
```



```
[217]: histogram(postDF[:, 6])
```

