Lab 5: Sea-Level Rise

Patricia Hashimoto (pch3)>

Fri., Feb. 16

1 Setup

1.1 The usual

As always:

1. Clone the lab repository to your computer
2. Open the lab repository in VS Code
3. Open the Julia REPL and activate, then instantiate, the lab environment
4. Make sure you can render: `quarto render template.qmd` in the terminal.
    - If you run into issues, try running `] build IJulia` in the Julia REPL (`]` enters the package manager).
    - If you still have issues, try opening up `blankfile.py`. That should trigger VS Code to give you the option to install the Python extension, which you should do. Then you should be able to open a menu in the bottom right of your screen to select which Python installation you want VS Code to use.

1.2 Load packages

```
using CSV
using DataFrames
using DataFramesMeta
using Distributions
using Plots
using StatsPlots
using Unitful

Plots.default(; margin=5Plots.mm)
```

1.3 Local package

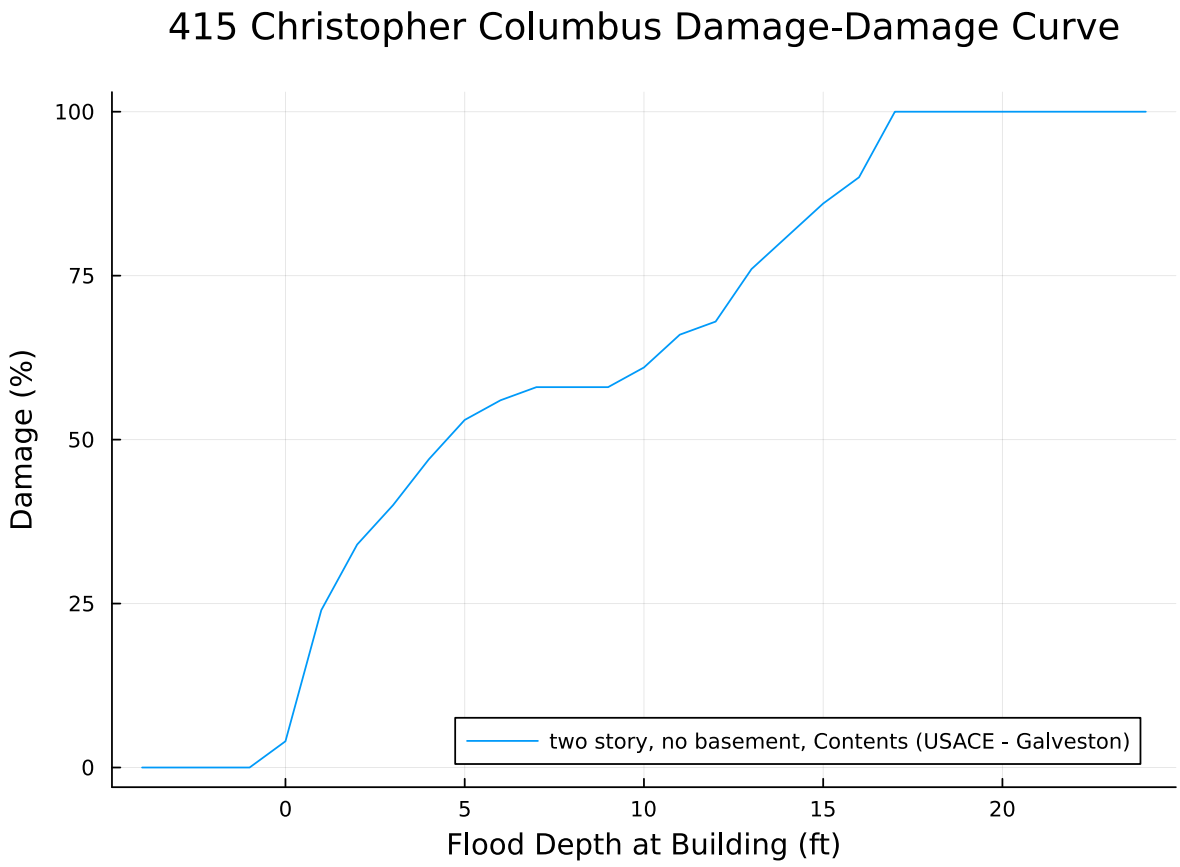
```
using Revise
using HouseElevation
```

The house investigated in this model is at 415 Christopher Columbus Blvd, Galveston, TX 77550. According to Zillow, the interior of this house is 1,726 sqft and the value of the house is about $331,000. Google Maps estimates that this house is 3,243 ft from Galveston Pier 21, TX Station

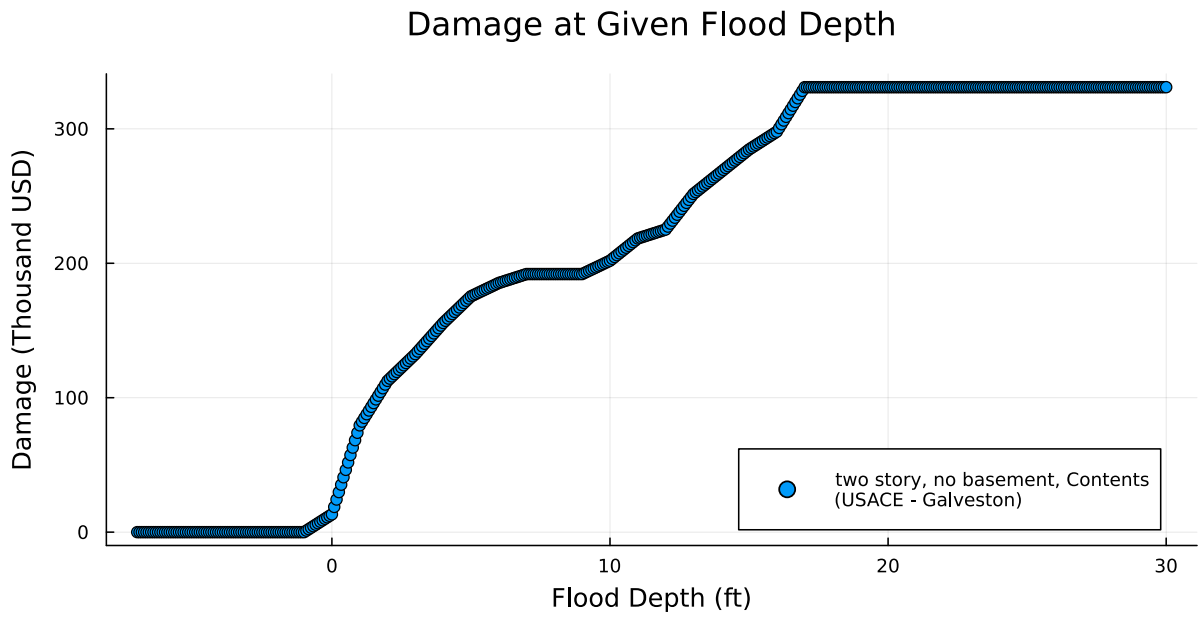
8771450, which is located at Galveston Pier 21, TX - Station ID: 8771450. The USGS National Map Viewer estimates that the elevation of the house is about 5.86 ft, which I round to 6 ft. The depth-damage curve will be estimated using a HAZUS model. The house is two story and has no basement, according to Zillow, so the curve used will be the Galveston district curve for a two-story residential building with no basement.

```
house = let
    haz_fl_dept = CSV.read("data/haz_fl_dept.csv", DataFrame) # read in the file
    desc = "two story, no basement, Contents"
    source = "USACE - Galveston"
    row = @rsubset(haz_fl_dept, :Description == desc, :Source == source)[1, :] # select the rou
    area = 1726u"ft^2"
    height_above_gauge = 6u"ft"
    House(
        row;
        area=area,
        height_above_gauge=height_above_gauge,
        value_usd=331_000,
    )
end
```

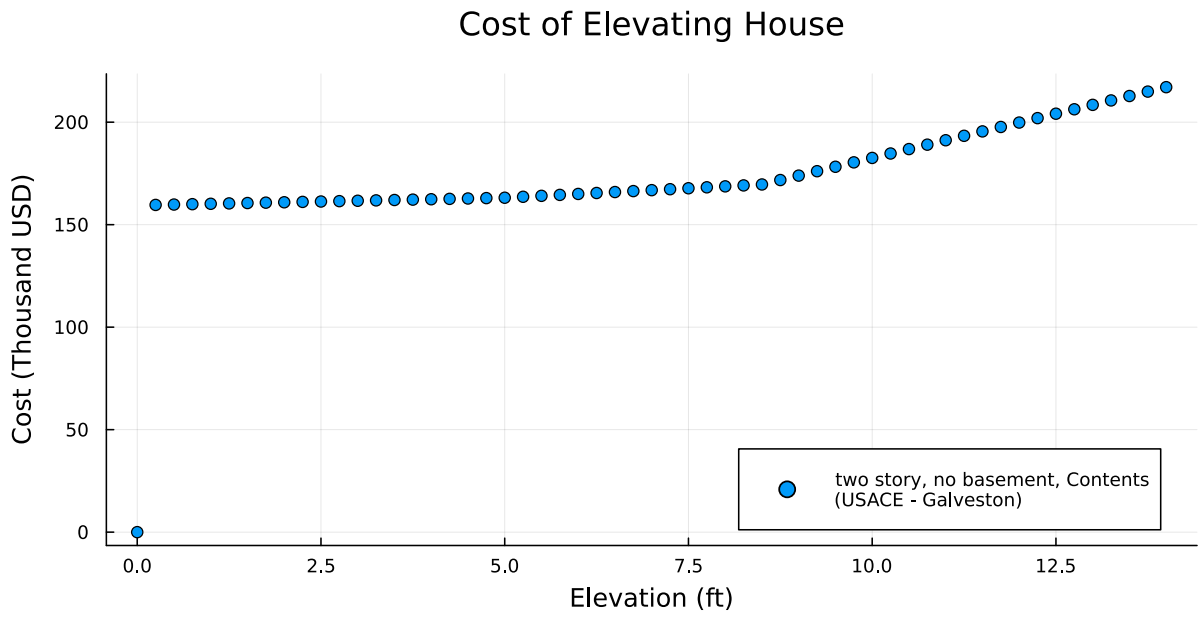
```
include("depthdamage.jl")

let
    haz_fl_dept = CSV.read("data/haz_fl_dept.csv", DataFrame) # read in the file
    desc = "two story, no basement, Contents"
    source = "USACE - Galveston"
    row = @rsubset(haz_fl_dept, :Description == desc, :Source == source)[1, :]
    house_dd = DepthDamageData(row)
    plot(
        house_dd.depths,
        house_dd.damages;
        xlabel="Flood Depth at Building",
        ylabel="Damage (%)",
        label="$(house_dd.description) ($(house_dd.source))",
        legend=:bottomright,
        size=(700, 500),
        title = "415 Christopher Columbus Damage-Damage Curve"
    )
end
```




```
let
    depths = uconvert.(u"ft", (-7.0u"ft"):(1.0u"inch"):(30.0u"ft"))
    damages = house.ddf.(depths) ./ 100
    damages_1000_usd = damages .* house.value_usd ./ 1000
    scatter(
        depths,
        damages_1000_usd;
        xlabel="Flood Depth",
        ylabel="Damage (Thousand USD)",
        label="$(house.description)\n($(house.source))",
        legend=:bottomright,
        size=(800, 400),
        yformatter=:plain,
        title = "Damage at Given Flood Depth"
    )
end
```




```
let
    elevations = 0u"ft":0.25u"ft":14u"ft"
    costs = [elevation_cost(house, e ) for e  in elevations]
    scatter(
        elevations,
        costs ./ 1_000;
        xlabel="Elevation",
        ylabel="Cost (Thousand USD)",
        label="$(house.description)\n($(house.source))",
        legend=:bottomright,
        size=(800, 400),
        yformatter=:plain,
        title = "Cost of Elevating House"
    )
end
```



2. Read in the sea-level rise data

```
slr_scenarios = let
    df = CSV.read("data/slr_oddo.csv", DataFrame)
    [Oddo17SLR(a, b, c, tstar, cstar) for (a, b, c, tstar, cstar) in eachrow(df)]
end
```

3. Modify my code to create a function to draw samples of storm surge and the discount rate. Explain your modeling choices!

I next define the distributions from which storm surges and discount rates will be sampled. The storm surge distribution is a hypothetical generalized extreme value distribution that is not rooted in actul storm data from the area.

```
function draw_surge_distribution()
      = rand(Normal(5, 1))
      = rand(Exponential(1.5))
      = rand(Normal(0.1, 0.05))
    GeneralizedExtremeValue( ,  ,  )
end
```

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

The water level at the house is the water level at the gauge minus the elevation of the house above the gauge. The elevation of the gauge is 6.24 ft above MSL [1], and the house is at 5.86 ft, so the water level at the house will be 0.38 ft above that at the gauge.

The discount rate is treated as a random variable drawn from a normal distribution. A more accurate way to estimate the discount rate could be by modeling discount rate with autoregressive techniques, such as in Zarekarizi (2020).

[1] https://tidesandcurrents.noaa.gov/stationhome.html?id=8771450

```julia
function draw_discount_rate()
    return rand(Normal(0.04, 0.02))
end
```

4. Define an illustrative action, SOW, and model parameters, and run a simulation.

I will first run a simulation in which the house is elevated 7 ft.

```julia
a = Action(7.0u"ft")
```

```julia
p = ModelParams(
    house=house,
    years=2024:2050
)
```

```julia
sow = SOW(
    rand(slr_scenarios),
    draw_surge_distribution() + .38,
    draw_discount_rate()
)
```

```julia
res = run_sim(a, sow, p)
```

```
-219070.48575942055
```

In this simulation, the net present value of raising the house 7 ft is `res`.

I next run a large ensemble of simulations. Due to limitations in computing power, I will run the simulation 100 times. This time I include a range of options for action in which the building could be elevated from 1 to 14 ft, with the action randomly sampled from a uniform distribution. It is unlikely that a private homeowner would elevate a two story house more than one story.

```julia
sows = [SOW(rand(slr_scenarios), draw_surge_distribution(), draw_discount_rate()) for _ in 1:10
actions = [Action(rand(1:14)*u"ft") for _ in 1:100] # these are all the same
results = [run_sim(a, s, p) for (a, s) in zip(actions, sows)]
```

```
100-element Vector{Float64}:
 -191617.31471690987
 -173950.82425692282
 -252109.71460326796
      -1.0990388322712937e6
 -193922.18090807015
 -217077.5
 -484199.43608610466
 -208447.5
 -204655.36866938596
 -163027.86263663662
 -547873.9153529087
 -203348.50882812194
 -186676.0858318912

 -162573.4708309751
```

```
-456924.33970857225
-585974.0842455172
-173988.9796852044
-162170.01083126685
-213755.60236399443
     -1.1632070075911297e6
-398601.5619307411
-189443.7690646654
-166838.57142857142
-204861.3682981762
-166838.57142857142
```

```
df = DataFrame(
    npv=results,
    Δh_ft=[a.Δh_ft for a in actions],
    slr_a=[s.slr.a for s in sows],
    slr_b=[s.slr.b for s in sows],
    slr_c=[s.slr.c for s in sows],
    slr_tstar=[s.slr.tstar for s in sows],
    slr_cstar=[s.slr.cstar for s in sows],
    surge_ =[s.surge_dist.  for s in sows],
    surge_ =[s.surge_dist.  for s in sows],
    surge_ =[s.surge_dist.  for s in sows],
    discount_rate=[s.discount_rate for s in sows],
)
```

	npv	Δh_ft	slr_a	slr_b	slr_c	slr_tstar	slr_cstar	surge_	
	Float64	Int64	Float64	Float64	Float64	Float64	Float64	Float64	
1	-1.91617e5	10	31.9053	1.8527	-0.000778514	2017.27	12.0608	5.85939	...
2	-1.73951e5	9	52.4114	2.93091	0.0073943	2026.93	11.1873	4.42214	...
3	-2.5211e5	13	42.9205	1.93323	-0.00147052	2070.8	31.163	4.27033	...
4	-1.09904e6	1	43.0704	2.58838	0.00499265	2043.43	27.1087	5.46678	...
5	-1.93922e5	10	46.5993	2.61474	0.0056419	2027.43	17.9874	6.12563	...
6	-2.17078e5	14	14.2282	1.75763	0.000161225	2053.2	17.5081	5.12993	...
7	-4.84199e5	14	32.5799	2.21218	0.0025509	2087.36	33.2821	5.33165	...
8	-2.08448e5	13	36.1867	2.44328	0.00525424	2065.28	15.7395	4.28376	...
9	-2.04655e5	9	33.2416	2.1129	0.00147229	2050.31	10.4833	4.46765	...
10	-1.63028e5	3	32.8849	1.98019	0.000616495	2068.11	15.9544	5.94887	...
11	-5.47874e5	9	66.651	3.18053	0.00853902	2022.49	14.0528	4.09349	...
12	-2.03349e5	9	38.1954	2.35331	0.00402575	2052.45	20.388	3.74249	...
13	-1.86676e5	8	40.3025	2.62536	0.00647678	2055.55	18.2324	6.44166	...
14	-1.91647e5	5	30.1338	2.16898	0.00257393	2048.68	13.1868	6.09545	...
15	-2.62803e5	13	26.8753	1.94751	0.000565848	2062.7	32.7959	4.34148	...
16	-1.66456e5	4	19.0593	2.09157	0.00154357	2030.61	22.0503	4.92236	...
17	-2.5228e5	3	42.5047	1.86087	-0.00124537	2035.29	15.8443	4.68151	...
18	-4.42214e5	6	46.3466	2.64401	0.00513911	2063.12	30.9556	5.18183	...
19	-1.61663e5	3	36.2446	2.25013	0.00274248	2023.0	4.4149	3.35971	...
20	-1.98731e5	7	46.5174	2.72088	0.00542627	2058.65	30.1148	3.64888	...
21	-1.73936e5	9	28.0754	2.07802	0.00170968	2074.22	20.1098	3.84819	...
22	-2.06829e5	12	33.3497	1.97691	-0.000145179	2086.58	8.25334	6.2527	...
23	-1.64449e5	5	51.2566	2.95228	0.00756198	2028.17	3.62694	4.80348	...
24	-1.73928e5	9	29.5975	2.1745	0.00233227	2030.68	25.5011	4.80496	...
25	-2.72631e5	7	42.5377	2.61353	0.00517856	2053.51	10.8473	4.7059	...
26	-1.65049e5	6	49.4878	2.36833	0.00307873	2043.49	7.8545	4.69339	...
27	-2.85657e5	10	45.0717	2.52686	0.00381226	2053.99	24.5622	4.26101	...
28	-1.10574e6	4	26.5314	2.2957	0.00430496	2044.57	24.5513	5.68269	...
29	-4.10386e5	3	39.4128	2.67184	0.00669738	2064.06	27.4198	5.23149	...
30	-4.58033e5	8	20.8965	1.29747	-0.00546871	2020.18	24.5287	4.31957	...
...	...	...	...	...	...	...	...	...	

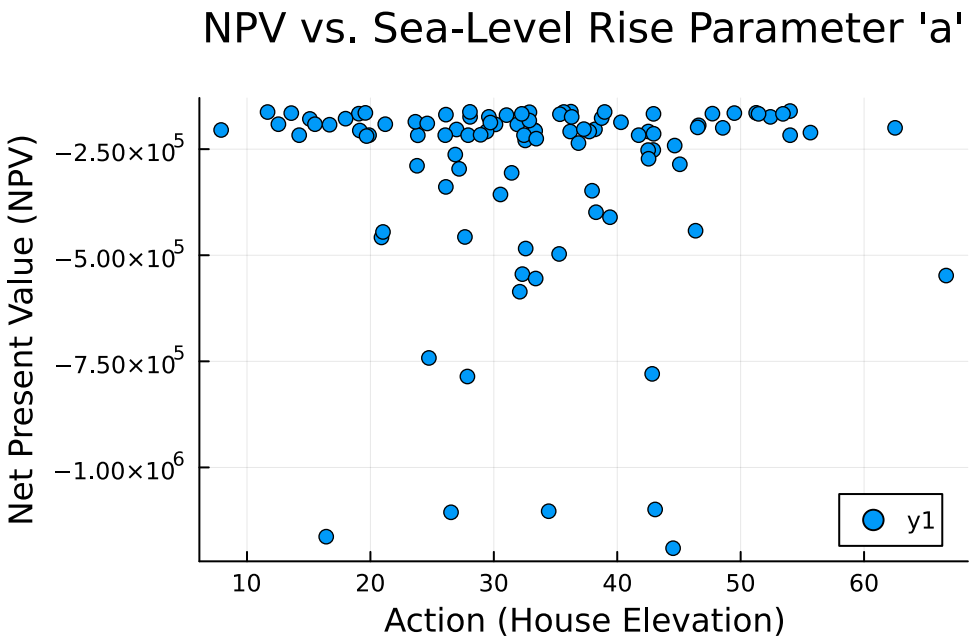
Now, analyze your results. You can use scatterplots and other visualizations, or any other statistical analyses that you think may be helpful. Remember that the goal is to understand how different parameter values affect the success or failure of different actions.

Some questions to consider:

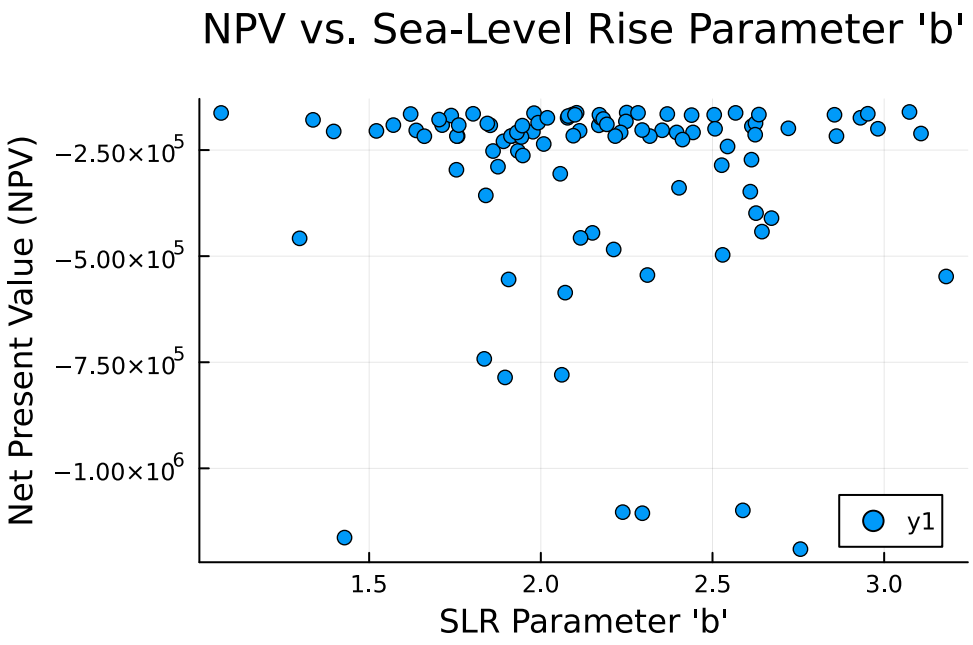
- When do you get the best results?
- When do you get the worst results?
- What are the most important parameters?
- If you had unlimited computing power, would you run more simulations? How many?
- What are the implications of your results for decision-making?

```
using DataFrames
using StatsPlots
```

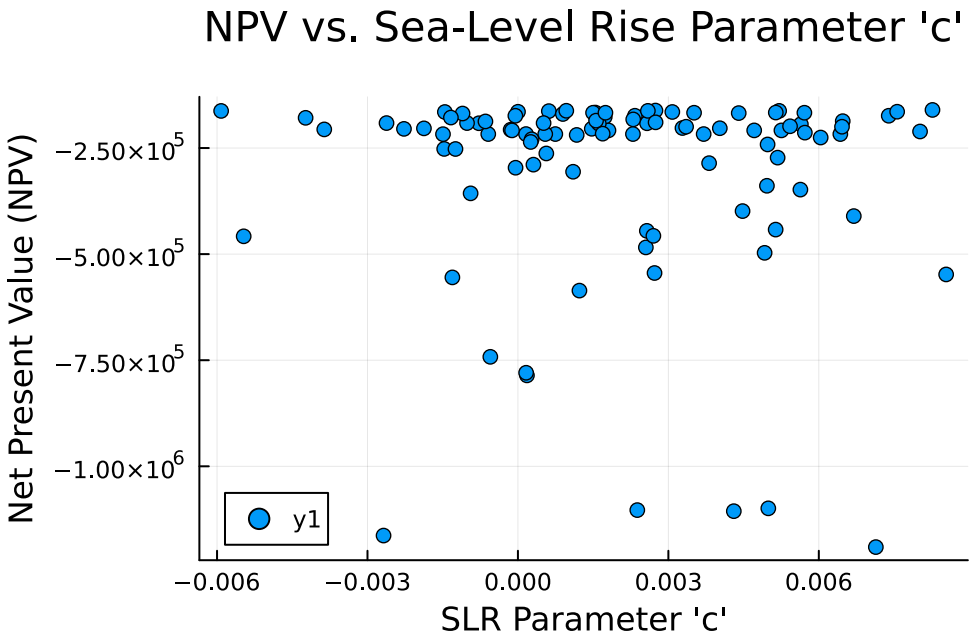
```
# Scatterplot of NPV vs. sea-level rise parameters
scatter(df.slr_a, df.npv,
    xlabel = "Action (House Elevation)",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Sea-Level Rise Parameter 'a'"
)
```




```
scatter(df.slr_b, df.npv,
    xlabel = "SLR Parameter 'b'",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Sea-Level Rise Parameter 'b'"
)
```

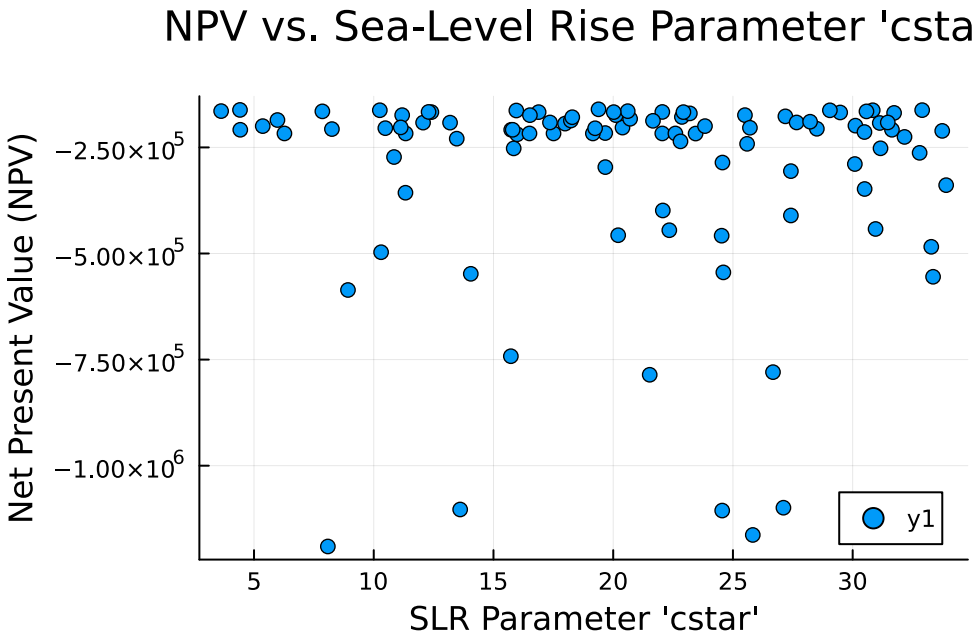



```
scatter(df.slr_c, df.npv,
    xlabel = "SLR Parameter 'c'",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Sea-Level Rise Parameter 'c'"
)
```



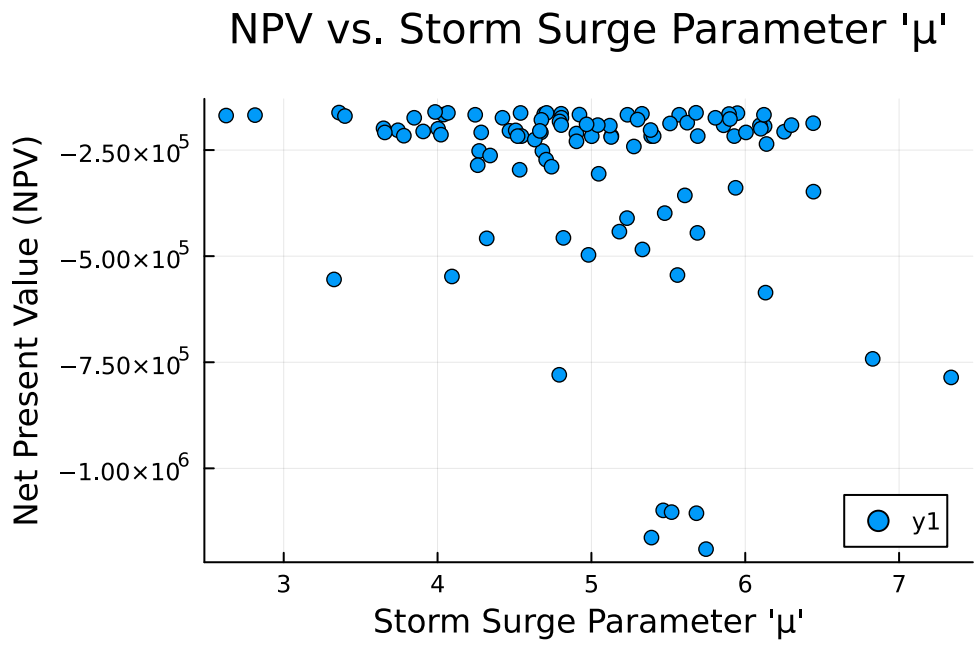

```
scatter(df.slr_tstar, df.npv,
    xlabel = "SLR Parameter 'tstar'",
    ylabel = "Net Present Value (NPV)",
```

```
    title = "NPV vs. Sea-Level Rise Parameter 'tstar'"
)

scatter(df.slr_cstar, df.npv,
    xlabel = "SLR Parameter 'cstar'",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Sea-Level Rise Parameter 'cstar'"
)
```

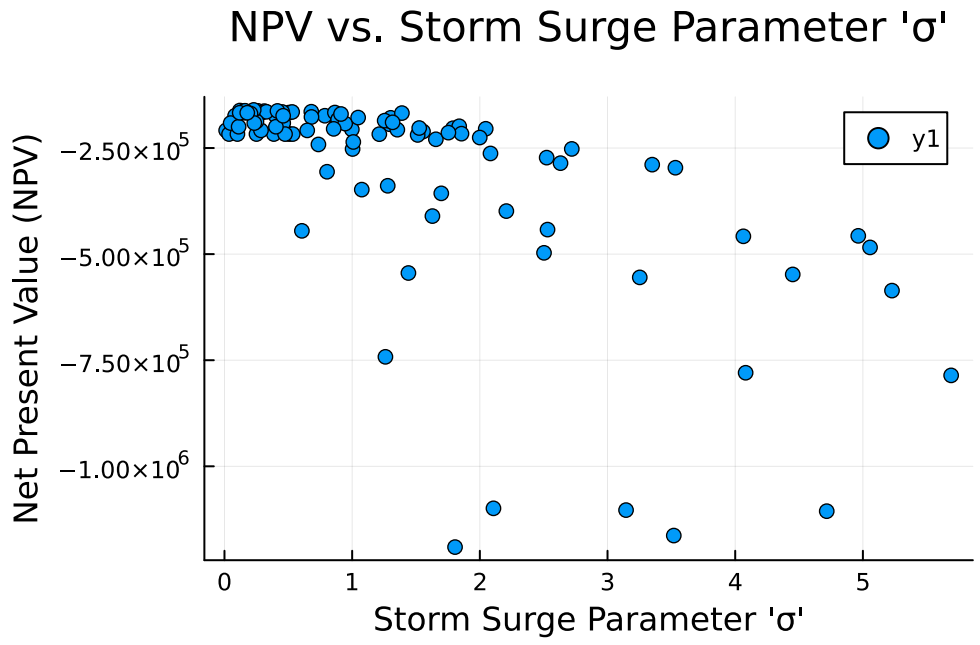


The most parameters with the most visually obvious relationship to NPV appear to be the storm surge parameters.

```
# Scatterplot of NPV vs. storm surge parameters
scatter(df.surge_ , df.npv,
    xlabel = "Storm Surge Parameter ' '",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Storm Surge Parameter ' '"
)
```

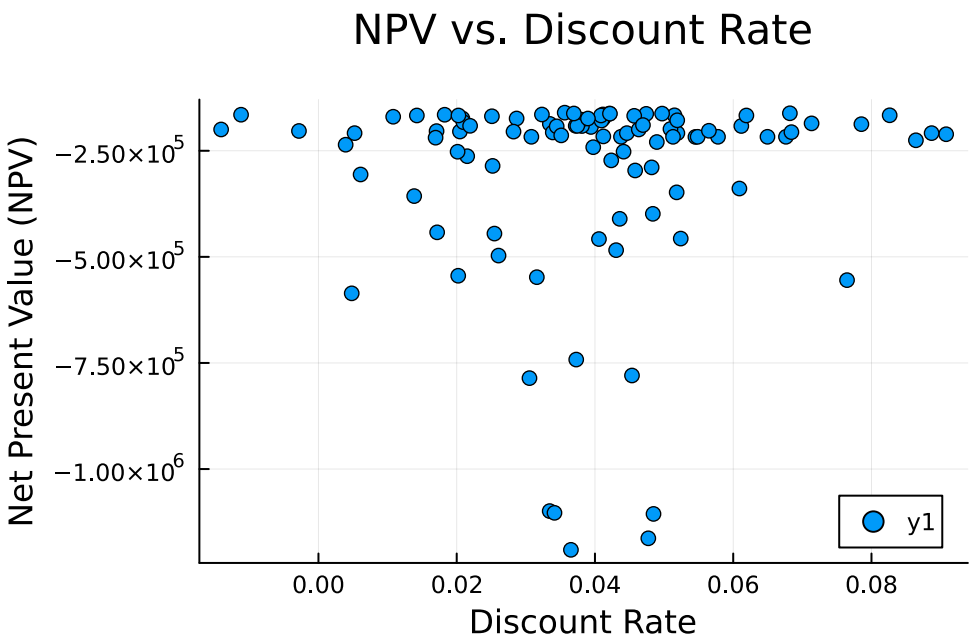



```
scatter(df.surge_ , df.npv,
    xlabel = "Storm Surge Parameter ' '",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Storm Surge Parameter ' '"
)
```




```
# Scatterplot of NPV vs. elevation action
scatter(df.Δh_ft, df.npv,
    xlabel = "Elevation Action (ft)",
```

```
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Elevation Action"
)

# Scatterplot of NPV vs. discount rate
scatter(df.discount_rate, df.npv,
    xlabel = "Discount Rate",
    ylabel = "Net Present Value (NPV)",
    title = "NPV vs. Discount Rate"
)
```




```
# Correlation matrix of parameters
correlation_matrix = cor(Matrix(df[:, [:npv, :Δh_ft, :slr_a, :slr_b, :slr_c, :slr_tstar, :slr_
```

```
correlation_matrix
```

```
11×11 Matrix{Float64}:
  1.0          0.29399     -0.00831476  …  -0.0152244    0.0342284
  0.29399      1.0          0.0332432       0.175131     0.0917991
 -0.00831476   0.0332432    1.0            -0.0716179   -0.0274141
 -0.0392597    0.00740046   0.839479        0.00997934   0.0041711
 -0.0551727    0.0279476    0.725475        0.0270607    0.0314428
  0.0396993   -0.00439222  -0.151384    …   0.0304893   -0.0273702
 -0.0318658   -0.0661202   -0.268626        0.227778    -0.0154055
 -0.254875    -0.031571    -0.082161        0.0115821   -0.0766578
 -0.634087     0.0878324    0.0238799       0.0176027    0.00656136
 -0.0152244    0.175131    -0.0716179       1.0          0.0203507
  0.0342284    0.0917991   -0.0274141   …   0.0203507    1.0
```