Final Project Report

Data Visualization (CSCI 627)

SUNKU BHANU KEDHAAR NATH - Z1974769

DataSet

The dataset I worked on Louisiana Home Rebuilding Grants.

Link : https://www.propublica.org/datastore/dataset/road-home-rebuilding-grants

About the data : This data is about the individual-level records for all property owners who received grants from the Road Home program, which was set up after hurricanes Katrina and Rita to cover repair and rebuilding costs that exceeded insurance payouts or FEMA aid for owner-occupied properties in Louisiana.

Data Dictionary :

Total_Closing_Amount: The total amount of funds disbursed to property owners, which is a key indicator of the grant amounts.

Compensation_Grants: The total amount received from the compensation grant program, which is a significant part of the grants.
s
Additional_Compensation_Grants: The total amount received from additional compensation grants, as this is another major component of the grants.

Elevation_Grant_Amount: The total amount received from elevation grants, which may be important for flood prevention efforts.

Individual_Mitigation_Measure_Amount: The total amount received from individual mitigation measure grants, which may be used for other flood prevention measures.

GIS_City: The city where the property is located, which is essential for geographic visualizations.

GIS_Parish: The parish (equivalent to a county) where the property is located, another important geographic indicator.

Closing_Option: The subprogram the property owner selected (e.g., repair, sell, or rent).

STFID : The census block containing the property, based on 2000 vintage geography. This column is important as this is used to collect latitude and longitude values from the us census bureau.

Example :
220,710,017,325,019 this is one of the value from this column.
22 represents Louisiana state, 071 represents Parish and rest represents census tract code.

NOLA_Planning_District_Name and NOLA_Neighborhood_Name: Geographic information specific to New Orleans.

Current_Damage_Assessment: The most recent damage estimate, which might be relevant for visualizing the condition of properties.

Current_PSV: The most recent estimate of the property's pre-storm market value, which could be significant for grant calculations.

Applicant_With_Current_Insurance: Whether the property owner had insurance.

Total_DOB_Amount: The total duplication of benefits received, which can impact grant calculations.

INTPTLAT00 : Latitude of home.

INTPTLON00 : Longitude of the home.

Combination of Latitude and Longitude are not unique for all the homes because homes that are near to each other have same tract code.

The above mentioned variables are the important ones but I have not done any dimensionality reduction at any step of the process because they might be needed in any of the analysis and for decision making. The propublica has very detailed explanation for each variable.I have found a little detail that the Data dictionary from the propublica did not give any explanation for these two variables.

1) Closing Damage Assessment
2) PSV at Closing

There are few variables with words Closing and Current in them My understanding is that closing means on day the contractor saw the home(just after the hurricanes) and gave an estimate for that home (PSV, Repair cost , Rebuild cost etc..) current means at present what is the present number or the updated numbers.

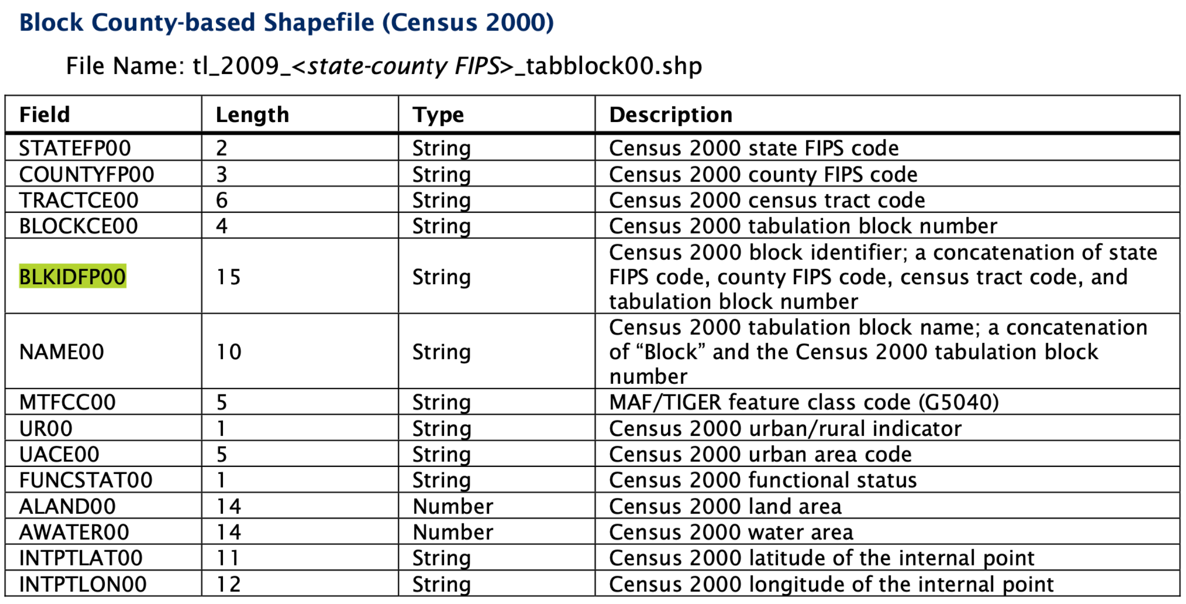
I have got the lat and log details from the following link

https://www2.census.gov/geo/tiger/TIGER2010/TABBLOCK/2000/

The above link has shapefile(.shp) for all of the counties in the United States
And the files are named in the state and countyFIPS format as shown In the figure below. The Road home data has is covered only 37 parishes. I have took all the unique STFID from the road home data and converted to string and split them based on state and countyFIPS for example
STFID : 220710017325019
Split the code like this => 22071
Now I have 37 unique splitted codes(state and county FIPS) and used these uniques codes to download the shape files.

Example file name : tl_2010_22071_tabblock00.shp, 2010 is the year here although the readme file of road home data says the census is from year 2000 but censes bureau has 2010 as the file name in the 2000/ directory see the link above.

Below is the data dictionary for those shape files The common attribute for road home dataset and shape file is BLKIDFP00 and STFID.



Block County-based Shapefile (Census 2000)

File Name: tl_2009_<*state-county FIPS*>_tabblock00.shp

Field	Length	Type	Description
STATEFP00	2	String	Census 2000 state FIPS code
COUNTYFP00	3	String	Census 2000 county FIPS code
TRACTCE00	6	String	Census 2000 census tract code
BLOCKCE00	4	String	Census 2000 tabulation block number
BLKIDFP00	15	String	Census 2000 block identifier; a concatenation of state FIPS code, county FIPS code, census tract code, and tabulation block number
NAME00	10	String	Census 2000 tabulation block name; a concatenation of "Block" and the Census 2000 tabulation block number
MTFCC00	5	String	MAF/TIGER feature class code (G5040)
UR00	1	String	Census 2000 urban/rural indicator
UACE00	5	String	Census 2000 urban area code
FUNCSTAT00	1	String	Census 2000 functional status
ALAND00	14	Number	Census 2000 land area
AWATER00	14	Number	Census 2000 water area
INTPTLAT00	11	String	Census 2000 latitude of the internal point
INTPTLON00	12	String	Census 2000 longitude of the internal point

I have took all of the shape files and concatenated into single geo pandas dataframe and then merged road home data with geo pandas dataframe and kept only latitude, longitude and BLKIDFP00 variables with the all of the columns listed above.

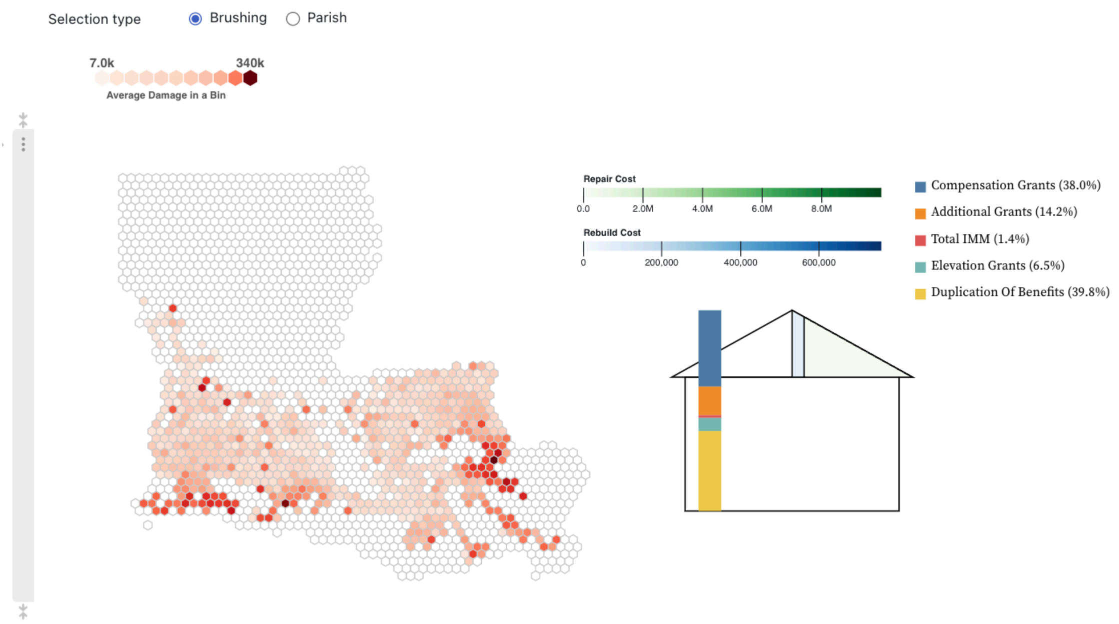
All of the data wrangling and integration has been done in python and the file has also been submitted.

Questions :

The question that I am trying to answer via visualization is :

out of current damage assessment how much amount did each parish get from road home program and how much from other sources such as insurance, FEMA out of these how much money is used for repairing the homes and rebuilding the homes.

Design:



I have updated my design. The new design not only shows the damage assessment per County but also user defined regions by selecting the region(brushing) the analysis on the right is also updated.

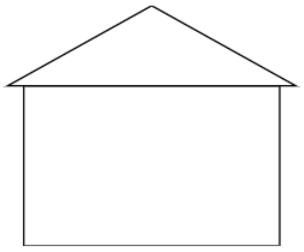
To crate Louisiana state I have used D3 HexGrid. How it works is that based on the svg width and height the d3 hex grid creates a tessellated grid with dots and creates hexagon with user defined radius. The difference between d3Hexbin and d3HexGrid is that hexbin creates bins only at where there is data in the map or chart.

Detailed Explanation for the d3 hex bin and Hexgrid is clearly explained by lars here:

https://observablehq.com/@larsvers/hexgrid-maps-with-d3-hexgrid?query=d3+hexgrid&onlyOwner=false

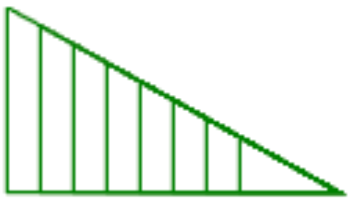
The Louisiana Hexgrid map shows the average damage in that particular bin. If not by average damage the darkness of the red will mostly be in the New Orleans and Jefferson region. In order to have even spread of the damages I have used Average damage per bin.

Now coming to home visualization



Now Lets see how the home works This is the base home everything except the chimney, triangles and the blue shaded part is constant and acts as the basic outline as shown above .

Now lets see triangles.

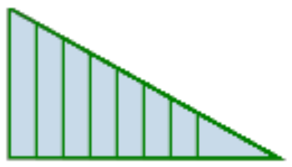


There are total of 8 triangles in the above picture each has different A and B point considering AC as diagonal line. When does each triangle be picked?? That depends on how much money is decided as Repair cost.

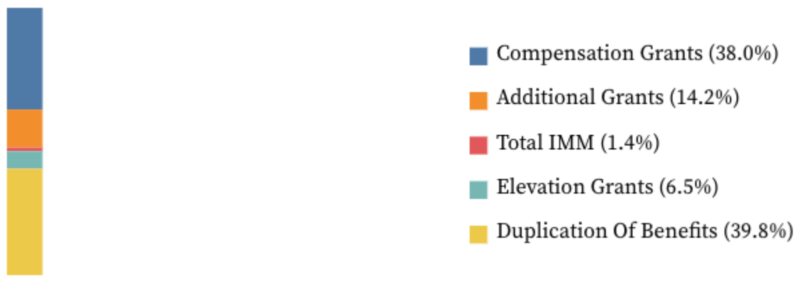
There are two variables

1) Current_Damage_Assessment
2) Damage Type 1 or 2

Current Damage Assessment is total decided losses a home has faced. Is the damage type repair or rebuild is listed in Damage Type 1 or 2 I have calculated all of the repair costs and rebuild costs separately and Current_Damage_Assessment the triangle is decided based on the percentage of repair amount if the repair amount is 80 percent out of total Current_Damage_Assessment then 2nd biggest triangle is selected. I have used ScaleLinear to to be selected according to the percentage. Now after selecting the triangle the fill color is based on the average cost of the selected region that's why the color scale changes every time brush is changed.

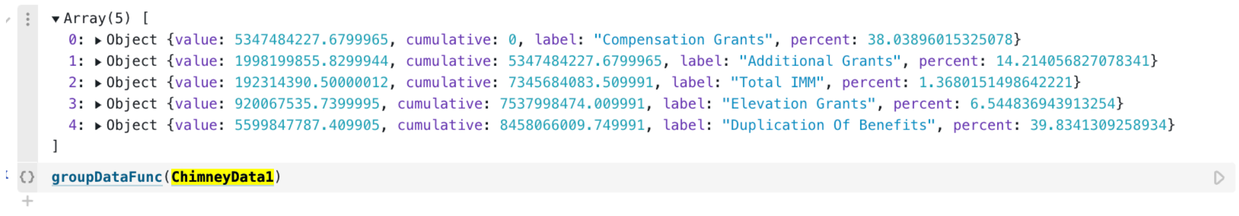


Now how are Rebuild damages are shown in the blue color back side of the repair cost the repair cost covers unto 80% (how much ever the repair amount is for selected region) of the base triangle which leaves 20% of the space that means 20 percent of the total current assessment damage is of Rebuilding the home and density of the color is based on the average Rebuild damage of the selected region.



The chimney is a single staked bar with height fixed. The Chimney represents how grants are distributed how much do they contribute for the selected region.

I have created cumulative addition like below image. This helps keeps the small values in the middle always so that they won't be on the top or bottom (ensuring the small parts are also visible).



```
▾Array(5) [
  0: ▸Object {value: 5347484227.6799965, cumulative: 0, label: "Compensation Grants", percent: 38.03896015325078}
  1: ▸Object {value: 1998199855.8299944, cumulative: 5347484227.6799965, label: "Additional Grants", percent: 14.214056827078341}
  2: ▸Object {value: 192314390.50000012, cumulative: 7345684083.509991, label: "Total IMM", percent: 1.3680151498642221}
  3: ▸Object {value: 920067535.7399995, cumulative: 7537998474.009991, label: "Elevation Grants", percent: 6.544836943913254}
  4: ▸Object {value: 5599847787.409905, cumulative: 8458066009.749991, label: "Duplication Of Benefits", percent: 39.8341309258934}
]
groupDataFunc(ChimneyData1)
```

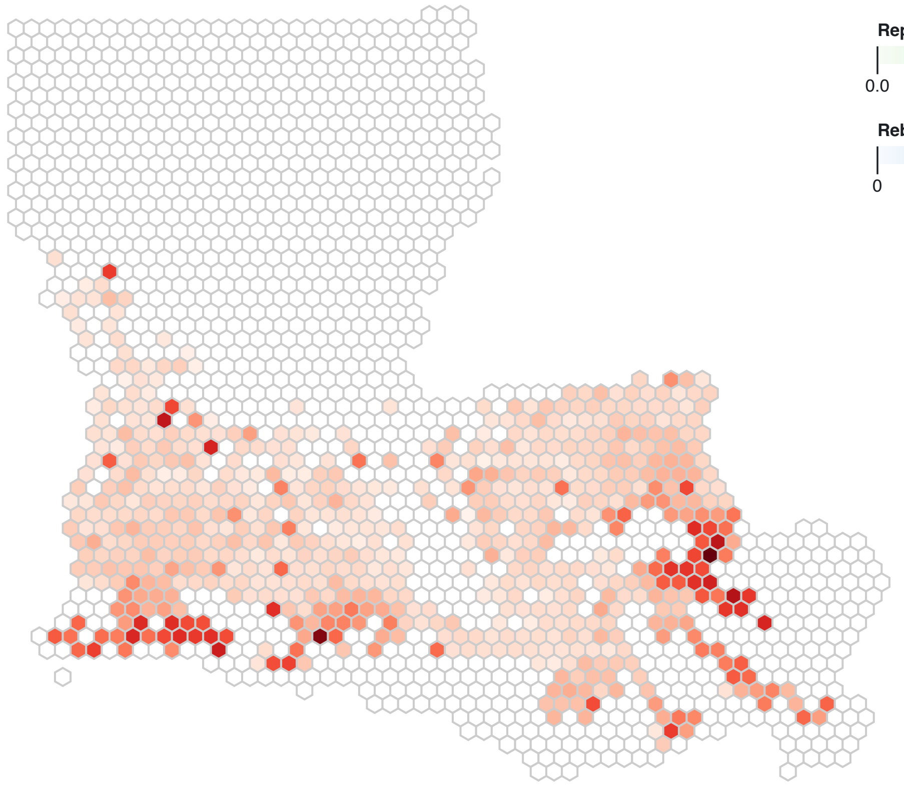
My visualization correctly provide insights for my question how are grants distributed for a region and how much is spent on repairing the home and rebuilding the home.

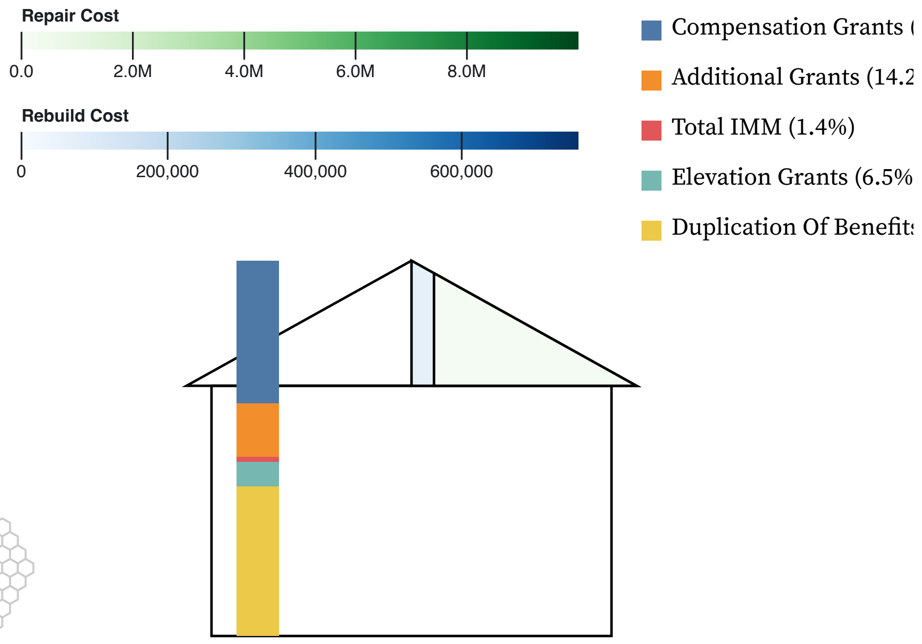
Project

Selection type

[✓ Brushing] Parish

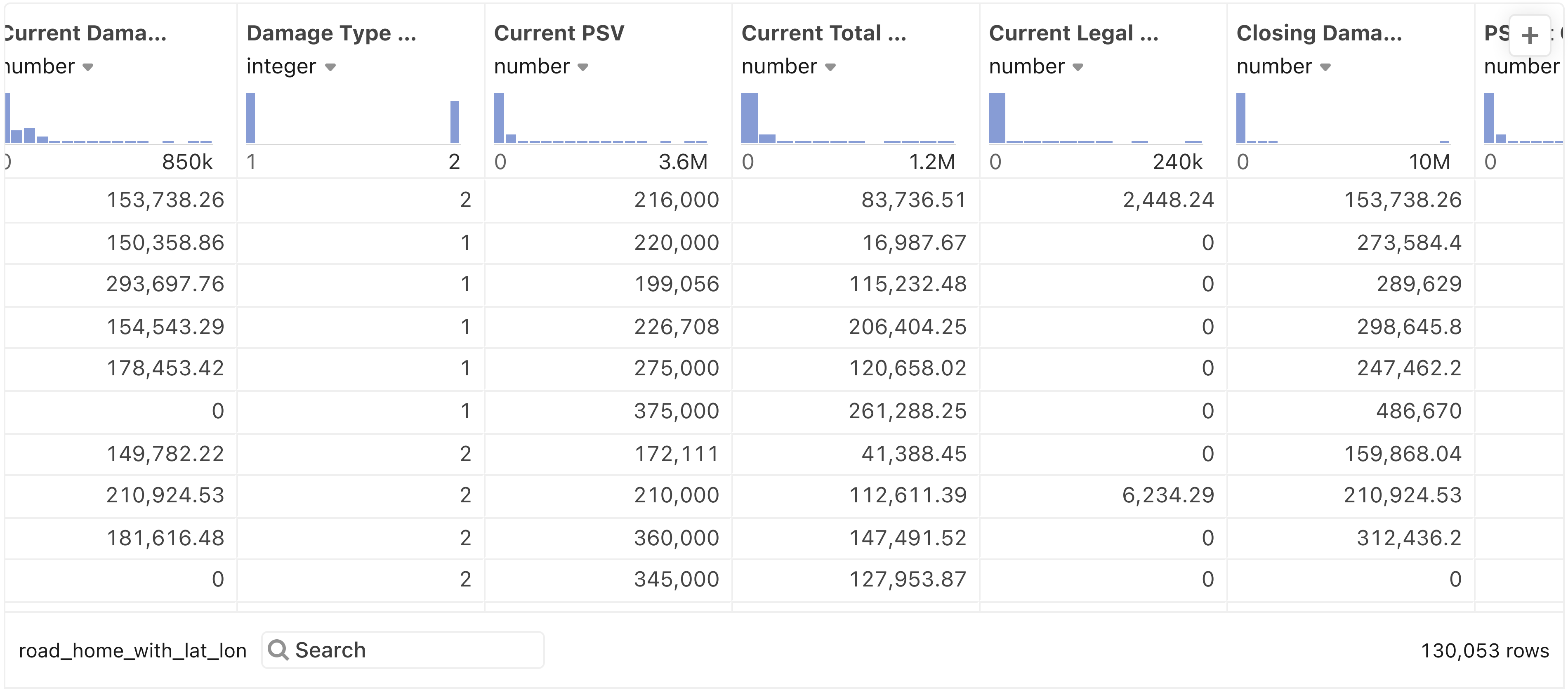
7.0k
340k
Average Damage in a Bin





Data

```
// googleUsCountyBoundaries = FileAttachment("google://us-county-boundaries.json").json()
```



Current Dama...	Damage Type ...	Current PSV	Current Total ...	Current Legal ...	Closing Dama...	PS
umber	integer	number	number	number	number	number
0 850k	1 2	0 3.6M	0 1.2M	0 240k	0 10M	0
153,738.26	2	216,000	83,736.51	2,448.24	153,738.26	
150,358.86	1	220,000	16,987.67	0	273,584.4	
293,697.76	1	199,056	115,232.48	0	289,629	
154,543.29	1	226,708	206,404.25	0	298,645.8	
178,453.42	1	275,000	120,658.02	0	247,462.2	
0	1	375,000	261,288.25	0	486,670	
149,782.22	2	172,111	41,388.45	0	159,868.04	
210,924.53	2	210,000	112,611.39	6,234.29	210,924.53	
181,616.48	2	360,000	147,491.52	0	312,436.2	
0	2	345,000	127,953.87	0	0	

road_home_with_lat_lon Search 130,053 rows

road_home_with_lat_l... road_ho...

Filter Columns Sort Slice Save

```
["Structure Type", "PARISH", "TOTAL_CLOSING_AMOUNT", "Total CG Amount", "Total ACG Amunt", "Total IMM Amount",
"Total Elevation Amount", "Current Damage Assessment", "Current Damage Assessment - Type 1", "Current Damage
Assessment - Type 2", "Damage Type 1 or 2", "Current PSV", "Current Total DOB Amount (no Legal Fees removed)",
"Closing Total DOB Amount"]
```

```
LousianaCounties = ▸Array(64) [Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, O
  LousianaCounties = FileAttachment("us-county-boundaries (3).json").json()
```

```
// LousianaCounties = (googleUsCountyBoundaries.filter((d) => d["stusab"] == "LA"))
```

```
lousianaCounitiesGeoData = LousianaCounties.map((obj, i) => {
  const type = obj["geo_shape"]["type"]
  const geometry = obj["geo_shape"]["geometry"]
  const properties = obj["geo_shape"]["properties"]
  const statefp = obj["statefp"]
  const countyfp = obj["countyfp"]
  const name = obj["name"]
  const stusab = obj["stusab"]
  return {type, geometry, properties, statefp, countyfp, name, stusab}
})
```

```
geoData = LousianaCounties.map((obj) => obj["geo_shape"]);
```

```
lousianaWithCounties = ({ type: "FeatureCollection", features : geoData });
```

```
width = 1000
```

```
height = 384;
```

```
projection = d3
    .geoAlbers()
    .fitSize([width, height], lousianaWithCounties);
```

```
▸ Array(2) [14059.667177159443, -3995.235327449189]
  projection([30.035, -89.956])
  // first home's lat lng
```

```
path = ƒ(t)
  path = d3.geoPath().projection(projection)
```

D3 Hexbin and Hexgrid

```
d3_hexbin = ▸ Object {hexbin: ƒ()}
  d3_hexbin = require("d3-hexbin@0.2")
```

```
d3_hexgrid = ▸ Object {hexgrid: ƒ(), geoPolygon: ƒ(t, n), polygonPoints: ƒ(r, t)}
  d3_hexgrid = require("d3-hexgrid")
```

```
hexbin = ƒ(points)
  hexbin = d3_hexbin.hexbin()
    .extent([[0,0],[width, height]])
    .radius(8);
```

```
hexgrid = ƒ(t, n)
  hexgrid = d3_hexgrid.hexgrid()
      .extent([width, height])
      .geography(lousianaWithCounties)
      .projection(projection)
      .pathGenerator(path)
      .hexRadius(4.5)
      .gridExtend(0);
```

```
dataForHexGrid = ▸ Array(121128) [Object, Object, Object, Object, Object, Object, Object, Object, Object, Object,
  dataForHexGrid = (road_home_with_lat_lon.filter(d => d["ARS File (Yes/No)"] === "N")).map((d, i) => {
      return {
        x: projection([d["INTPTLON00"], d["INTPTLAT00"]])[0],
        y: projection([d["INTPTLON00"], d["INTPTLAT00"]])[1],
        currentDamageAssesment: d["Current Damage Assessment"],
        lat: d["INTPTLAT00"],
        lng: d["INTPTLON00"],
        ...d
      };
    })
```

```
hex = ƒ(t)
```

```
  hex = hexgrid(dataForHexGrid, ["currentDamageAssesment","PARISH", "Structure Type", "TOTAL_CLOSING_AMOUNT",
  "Total CG Amount", "Total ACG Amunt", "Total IMM Amount", "Total Elevation Amount", "Current Damage Assessment",
  "Current Damage Assessment - Type 1", "Current Damage Assessment - Type 2", "Damage Type 1 or 2", "Current PSV",
  "Current Total DOB Amount (no Legal Fees removed)", "Closing Total DOB Amount"])
```

```
lousianaDamageColours = ƒ(n)
```

```
  lousianaDamageColours = d3.scaleSequential(t => {
    var tNew = Math.pow(t, 10);
    return d3.interpolateViridis(tNew);
  }).domain(hex.grid.extentPointDensity.reverse())
```

```
summedUpData = ▸ Array(1887) [Array(0), Array(0), Array(0), Array(0), Array(0), Array(0), Array(0), Array(0), Arr
```

```
   summedUpData = hex.grid.layout.map(d => {
    if (d.length > 0) {
      const totalDamage = d.reduce((acc, dd) => acc + dd.currentDamageAssesment, 0);
      d["AverageBinDamage"] = totalDamage/ d.length;
      d["TotalBinDamage"] = totalDamage

    } else {
      d["AverageBinDamage"] = 0;
      d["TotalBinDamage"] = 0;
    }
    return d;
```

```
 });
```

```
color = ƒ(n)
  color = d3.scaleSequential(d3.interpolateReds).domain(d3.extent(summedUpData.map(d => d["AverageBinDamage"])));
```

```
▸ Array(2) [0, 337488.1318018016]
  d3.extent(summedUpData.map(d => d["AverageBinDamage"]))
```

```
filteredData = ▸ Array(675) [Array(1), Array(1), Array(4), Array(6), Array(1), Array(1), Array(1), Array(3), Arra
  filteredData = (hex.grid.layout).filter(d => d.length > 0)
```

Charts

```
<style>
```

```
<detached>
```

```
  cc = {
    const margin = { top: 40, bottom: 40, left: -215, right: 50 };
    let filteredData2 = newFilteredData;

    function formatNumber(number, decimalPlaces = 2) {
      return number.toLocaleString("en-US", {
        maximumFractionDigits: decimalPlaces
      });
    }
    // const divElt = htl.html`<div id = "charts"></div>`;
    const fullSvg = d3
      .create("svg")
      .attr("width", 1000)
      .attr("height", height + 50);
    const svg = fullSvg
      .append("g")
      .attr("transform", `translate(${margin.left}, ${margin.top})`);
    const chart2svg = fullSvg
      .append("g")
      .attr("transform", `translate(${[1000 / 2, margin.top]})`);
    Chart2(newFilteredData);

    // Creating hexagons
    const hexs = svg
      .append("g")
      .selectAll(".hex")
      .data(hex.grid.layout)
      .enter()
      .append("path")
      .attr("class", "hex")
      .attr("d", hex.hexagon())
      .attr("transform", (d) => `translate(${d.x}, ${d.y})`)
      .attr("fill", (d) => (!d.datapoints ? "#fff" : color(d.AverageBinDamage)))
      .style("stroke", "#ccc");
    // .append("title")
    // .text(d => `Total Damage Cost : $${formatNumber(d.TotalBinDamage)} \nAverage Damage Cost :
  $${formatNumber(d.AverageBinDamage)} \nNumber of Houses : ${d.length}`)

    // defining brush
    const brush = d3
      .brush()
      .on("start", brushstart)
      // .on("brush", brushmove)
      // .on("end", brushend)
      .on("brush end", brushmov)
      .extent([
        [0, 0],
        [width, height]
      ]);

    let brushCell;

    // brush functions.
```

```
function brushstart(p) {
  if (brushCell !== this) {
    d3.select(brushCell).call(brush.move, null);
    brushCell = this;
  }
}

function brushmov({ selection }) {
  if (selection) {
    const [[x0, y0], [x1, y1]] = selection;
    const filteredHex = svg
      .selectAll(".hex")
      .style("fill-opacity", 0.3)
      .filter((d) => {
        const [hx, hy] = [d.x, d.y];
        return hx >= x0 && hx <= x1 && hy >= y0 && hy <= y1;
      })
      .style("fill", (d) =>
        !d.datapoints ? "#fff" : color(d.AverageBinDamage)
      )
      .style("fill-opacity", 1);
    const filteredData1 = hex.grid.layout.filter((d, i) => {
      const [hx, hy] = [d.x, d.y];
      return hx >= x0 && hx <= x1 && hy >= y0 && hy <= y1;
    });
    filteredData2 = [].concat(
      ...filteredData1
        .filter((d) => d.length > 0)
        .map((d) => d.slice(0, d.length))
    );
  } else {
    filteredData2 = newFilteredData;
    d3.selectAll(".hex").style("fill-opacity", 1);
  }
  Chart2(filteredData2);
}
```

```
// function brushmove(p) {
//   var e = d3.brushSelection(this);

//   if (!p) {
//     d3.selectAll(".hex").style("fill-opacity", 0.3);
//     filteredData2 = newFilteredData;
//   }
//   else {
//     const [[x0, y0], [x1, y1]] = e;
//     const filteredHex = svg
//       .selectAll(".hex")
//       .style("fill-opacity", 0.3)
//       .filter((d) => {
//         const [hx, hy] = [d.x, d.y];
//         return hx >= x0 && hx <= x1 && hy >= y0 && hy <= y1;
//       })
//       .style("fill", (d) =>
//         !d.datapoints ? "#fff" : color(d.AverageBinDamage)
//       )
//       .style("fill-opacity", 1);

//     const filteredData1 = hex.grid.layout.filter((d, i) => {
//       const [hx, hy] = [d.x, d.y];
//       return hx >= x0 && hx <= x1 && hy >= y0 && hy <= y1;
//     });
//     console.log(filteredData1);
//     filteredData2 = [].concat(
//       ...filteredData1
//         .filter((d) => d.length > 0)
//         .map((d) => d.slice(0, d.length))
//     );
//   }
//   Chart2(filteredData2);
// }

// function brushend() {
//   var e = d3.brushSelection(this);
```

```
  //   if (e === null) {
  //     d3.selectAll(".hex")
  //       .style("fill", (d) =>
  //         !d.datapoints ? "#fff" : color(d.AverageBinDamage)
  //       )
  //       .style("fill-opacity", 1);
  //   }
  // }

  // interaction
  if (interaction == "Parish") {
    // appending parish boundries.
    svg
      .append("g")
      .attr("id", "LousianaMap")
      .selectAll("path")
      .data(lousianaCounitiesGeoData)
      .join("path")
      .attr("d", path)
      .attr("stroke", "grey")
      .attr("stroke-width", 1)
      .attr("fill", "white")
      .attr("fill-opacity", 0)
      // .append("title")
      // .text((obj) => `Parish : ${obj["name"]}`)
      .on("click", (event, d) => {
        Chart2(newFilteredData.filter((dd) => dd["PARISH"] == d.name));
      });
  } else {
    svg.call(brush);
  }

  // Function to create glyph.
  function Chart2(data, psvValue = null) {
    chart2svg.selectAll("g").remove();
    const margin = { top: 40, bottom: 40, left: -215, right: 50 };
    const newFilteredData = data;
```

```
      const filteredDataTotalRebuildDamage = d3.sum(
        newFilteredData.map((d) => d["Current Damage Assessment - Type 2"])
      );
      const filteredDataAvgPSV = d3.mean(
        newFilteredData.map((d) => d["Current PSV"])
      );
      const sc = d3
        .scaleLinear()
        .domain(d3.extent(newFilteredData.map((d) => d["Current PSV"])))
        .range([0.95, 1.5]);
      const filteredDataTotalRepairDamage = d3.sum(
        newFilteredData.map((d) => d["Current Damage Assessment - Type 1"])
      );
      const filteredDataTotalDecidedDamage = d3.sum(
        newFilteredData.map((d) => d["Current Damage Assessment"])
      );
      const RepairDamageInDecidedDamage = d3.sum(
        newFilteredData
          .filter((d) => d["Damage Type 1 or 2"] == 1)
          .map((d) => d["Current Damage Assessment"])
      );
      const RebuildDamageInDecidedDamage = d3.sum(
        newFilteredData
          .filter((d) => d["Damage Type 1 or 2"] == 2)
          .map((d) => d["Current Damage Assessment"])
      );
      const repairDamageScale = d3.scaleLinear().domain([0, 100]).range([0, 7]);
      const filteredDataTotalGrants = d3.sum(
        newFilteredData.map((d) => d["TOTAL_CLOSING_AMOUNT"])
      );
      const filteredDataTotalCG = d3.sum(
        newFilteredData.map((d) => d["Total CG Amount"])
      );
      const filteredDataTotalACG = d3.sum(
        newFilteredData.map((d) => d["Total ACG Amunt"])
      );
      const filteredDataTotalIMM = d3.sum(
        newFilteredData.map((d) => d["Total IMM Amount"])
      );
      const filteredDataTotalClosingDOB = d3.sum(
        newFilteredData.map((d) => d["Closing Total DOB Amount"])
      );
      const svg2 = d3.create("svg").attr("width", 1300).attr("height", 500);
      const repairMean = d3.mean(
        newFilteredData
          .filter((d) => d["Damage Type 1 or 2"] == 1)
          .map((d) => d["Current Damage Assessment"])
      );
      const repairColor = d3
        .scaleSequential(d3.interpolateGreens)
        .domain(
          d3.extent(
            newFilteredData
              .filter((d) => d["Damage Type 1 or 2"] == 1)
              .map((d) => d["Current Damage Assessment"])
          )
        );

      // const repairColor = d3
      //   .scaleSequential(d3.interpolateReds)
      //   .domain([
      //     0,
      //     d3.sum(
      //       newFilteredData1
      //         .filter((d) => d["Damage Type 1 or 2"] == 1)
      //         .map((d) => d["Current Damage Assessment"])
      //     )
      //   ]);

      const rebuildMean = d3.mean(
        newFilteredData
          .filter((d) => d["Damage Type 1 or 2"] == 2)
          .map((d) => d["Current Damage Assessment"])
      );
```

```
const rebuildColor = d3
  .scaleSequential(d3.interpolateBlues)
  .domain(
    d3.extent(
      newFilteredData
        .filter((d) => d["Damage Type 1 or 2"] == 2)
        .map((d) => d["Current Damage Assessment"])
    )
  );
const filteredDataCG = d3.sum(
  newFilteredData.map((d) => d["Total CG Amount"])
);
const filteredDataACG = d3.sum(
  newFilteredData.map((d) => d["Total ACG Amunt"])
);

const filteredDataEGA = d3.sum(
  newFilteredData.map((d) => d["Total Elevation Amount"])
);
const ChimneyData1 = [
  { label: "Compensation Grants", value: filteredDataCG },
  { label: "Additional Grants", value: filteredDataACG },
  { label: "Total IMM", value: filteredDataTotalIMM },
  { label: "Elevation Grants", value: filteredDataEGA },
  { label: "Duplication Of Benefits", value: filteredDataTotalClosingDOB }
];

const xScale = d3
  .scaleLinear()
  .domain([0, d3.sum(ChimneyData1, (d) => d.value)])
  .range([0, 150]);

const colors = [
  "#4e79a7",
  "#f28e2c",
  "#e15759",
  "#76b7b2",
    // "#59a14f",
    "#edc949",
    "#af7aa1",
    "#ff9da7",
    "#9c755f",
    "#bab0ab"
  ];
  const spacing = 15;
  // creating home structure.
  const g = chart2svg
    .append("g")
    .attr(
      "transform",
      `translate(${70}, ${margin.top + margin.top + 50})scale(1.25,1.25)`
    );
  g.append("rect")
    .attr("x", 20)
    .attr("y", 50)
    .attr("width", 160)
    .attr("height", 100)
    .attr("fill", "none")
    .attr("stroke", "black");
  // Upper left half triangle
  g.append("polygon")
    .attr("points", "10,50 100,0 190,50")
    .attr("fill", "none")
    .attr("stroke", "black");
  // upper right half triangle (for rebuild homes)
  g.append("polygon")
    .attr("points", "100,0 100,50 190,50")
    .attr("fill", rebuildColor(rebuildMean))
    .attr("stroke", "black")
    .append("title")
    .text(
      (d, i) =>
        `Total Rebuild Damage: ${d3.format(".2s")(
          RebuildDamageInDecidedDamage.toFixed(2)
          )}\nRebuild % in whole damage: ${(
            (RebuildDamageInDecidedDamage / filteredDataTotalDecidedDamage) *
            100
          ).toFixed(2)}%\nAvg Damage:${d3.format(".2s")(
            d3.mean(
              newFilteredData
                .filter((d) => d["Damage Type 1 or 2"] == 2)
                .map((d) => d["Current Damage Assessment"])
            )
          )}`
      );
    // upper right half triangel (for repair homes).
    g.append("path")
      .attr(
        "d",
        paths[
          Math.round(
            repairDamageScale(
              (RepairDamageInDecidedDamage / filteredDataTotalDecidedDamage) *
                100
            )
          )
        ]
      )
      .attr(
        "fill",
        repairColor(
          d3.mean(
            newFilteredData
              .filter((d) => d["Damage Type 1 or 2"] == 1)
              .map((d) => d["Current Damage Assessment"])
          )
        )
      )
      .attr("stroke", "black")
      .append("title")
      .text(
        (d, i) =>
          `Total repair Damage: ${d3.format(".2s")(
            RepairDamageInDecidedDamage.toFixed(2)
          )}\nRepair % in whole damages: ${(
            (RepairDamageInDecidedDamage / filteredDataTotalDecidedDamage) *
            100
          ).toFixed(2)}%\nAvg Damage: ${d3.format(".2s")(
            d3.mean(
              newFilteredData
                .filter((d) => d["Damage Type 1 or 2"] == 1)
                .map((d) => d["Current Damage Assessment"])
            )
          )}\n`
      );

    //  creating Chimney
    g.selectAll("g")
      .data(groupDataFunc(ChimneyData1))
      .join("g")
      .append("rect")
      .attr("x", 30)
      .attr("y", (d) => xScale(d.cumulative))
      .attr("width", 17)
      .attr("height", (d) => xScale(d.value))
      .style("fill", (d, i) => colors[i])
      .append("title")
      .text((d) => `${d.label} : $${d3.format(".2s")(d.value.toFixed(2))}`);

    // Chimney Legend
    chart2svg
      .append("g")
      .selectAll("g")
      .data(groupDataFunc(ChimneyData1))
      .join("rect")
      .attr("x", 310)
      .attr("y", (d, i) => 10 * (i + 1) + spacing * i)
      .attr("width", 10)
      .attr("height", 10)
      .attr("fill", (d, i) => colors[i]);

    chart2svg
      .append("g")
      .selectAll("text")
      .data(groupDataFunc(ChimneyData1))
      .join("text")
      .attr("x", 325)
      .attr("y", (d, i) => 10 * (i + 1) + spacing * i + 7)
      .style("fill", "black")
      .style("font-size", "12px")
      .text((d) => `${d.label} (${d.percent.toFixed(1)}%)`);

    chart2svg
      .append("g")
      .attr("transform", "translate(0,0)scale(0.87, 0.87)")
      .append(() =>
        Legend(repairColor, {
          title: "Repair Cost",
          tickFormat: (d) => d3.format(".2s")(d)
        })
      );

    chart2svg
      .append("g")
      .attr("transform", "translate(0,50)scale(0.87,0.87)")
      .append(() => Legend(rebuildColor, { title: "Rebuild Cost" }));
  }

  return fullSvg.node();
}
```

Helpers and Testing

```
newFilteredData = [].concat(...(filteredData.filter(d => d.length > 0 )).map(d => d.slice(0,d.length)));
```

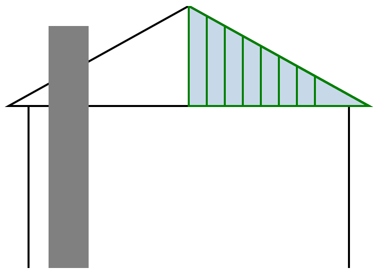
```
newFilteredData1 = [].concat(...(filteredData.filter(d => d.length > 0 )).map(d => d.slice(0,d.length)));
```

```
Chart2 = ƒ(…)
```

```
orleans = road_home_with_lat_lon.filter(d => d["PARISH"] == "Orleans")
```

```
p = "M128,208 H128 H208 V128 H224 L167,64 L112,128 H128 V208"
```

```
polp = "152,50 175,40 190,50"
// first third same, first is for the upper tip and third is for the straight lines lower tip
// fourth for the upper tip
```



```
<svg width="250" height="150">
    <rect x="20" y="50" width="160" height="100" fill="none" stroke="black" />
    <polygon points="10,50 100,0 190,50" fill="none" stroke="black" />
    <rect x="30" y="10" width="20" height="140" fill="gray" />
    <polygon id="fundsTriangle" points="100,0 100,50 190,50" fill="steelblue" fill-opacity = 0.3 stroke = "black"
/>
    <path d = "${paths[0]}" stroke = "green" fill = "none"/>
    <path d = "${paths[1]}" stroke = "green" fill = "none"/>
    <path d = "${paths[2]}" stroke = "green" fill = "none"/>
    <path d = "${paths[3]}" stroke = "green" fill = "none"/>
    <path d = "${paths[4]}" stroke = "green" fill = "none"/>
      <path d = "${paths[5]}" stroke = "green" fill = "none"/>
        <path d = "${paths[6]}" stroke = "green" fill = "none"/>
              <path d = "${paths[7]}" stroke = "green" fill = "none"/>
  </svg>

// paths = [
//   "M172,40 V50 H190 L172,40",
//   "M154,30 V50 H190 L154,30",
//   "M136,20 V50 H190 L136,20",
//   "M118,10 V50 H190 L118,10",
//   "M100,0 V50 H190 L100,0"
// ]
```

```
paths = [
  "M163,35 V50 H190 L163,35",
  "M154,30 V50 H190 L154,30",
  "M145,25 V50 H190 L145,25",
  "M136,20 V50 H190 L136,20",
  "M127,15 V50 H190 L127,15",
  "M118,10 V50 H190 L118,10",
  "M109,5 V50 H190 L109,5",
  "M100,0 V50 H190 L100,0"
]
```

```
filteredDataTotalIMM = d3.sum(
    newFilteredData.map((d) => d["Total IMM Amount"])
  );
```

```
filteredDataTotalGrants = d3.sum(
    newFilteredData.map((d) => d["TOTAL_CLOSING_AMOUNT"])
  );
```

```
filteredDataTotalClosingDOB = d3.sum(
    newFilteredData.map((d) => d["Closing Total DOB Amount"])
  );
```

```
filteredDataCG = d3.sum(
    newFilteredData.map((d) => d["Total CG Amount"])
  );
```

```
filteredDataACG = d3.sum(
    newFilteredData.map((d) => d["Total ACG Amunt"])
  );
```

```
filteredDataEGA = d3.sum(
    newFilteredData.map((d) => d["Total Elevation Amount"]))
```

```
xScale = d3.scaleLinear()
    .domain([0, d3.sum(ChimneyData1, d => d.value)])
    .range([0, 150]);
```

```
ChimneyData1 = [
      { label: "Compensation Grants", value: filteredDataCG},
      {label : "Additional Grants", value: filteredDataACG},
      { label: "Total IMM", value: filteredDataTotalIMM },
      { label : "Elevation Grants", value : filteredDataEGA},
      { label: "Duplication Of Benefits", value: filteredDataTotalClosingDOB },
    ];
```

```
▼ Array(5) [
  0: ▶ Object {value: 5347484227.6799965, cumulative: 0, label: "Compensation Grants", percent: 38.03896015325078
  1: ▶ Object {value: 1998199855.8299944, cumulative: 5347484227.6799965, label: "Additional Grants", percent: 14
```

```
    2: ▶ Object {value: 192314390.50000012, cumulative: 7345684083.509991, label: "Total IMM", percent: 1.368015149
    3: ▶ Object {value: 920067535.7399995, cumulative: 7537998474.009991, label: "Elevation Grants", percent: 6.544
    4: ▶ Object {value: 5599847787.409905, cumulative: 8458066009.749991, label: "Duplication Of Benefits", percent
  ]
```

```
groupDataFunc = f(data)
  groupDataFunc(ChimneyData1)
```

```
  function groupDataFunc(data) {
      const percent = d3.scaleLinear()
        .domain([0, d3.sum(data, d => d.value)])
        .range([0, 100])
      let cumulative = 0
      const _data = data.map(d => {
        cumulative += d.value
        return {
          value: d.value,
          cumulative: cumulative - d.value,
          label: d.label,
          percent: percent(d.value)
        }
      }).filter(d => d.value > 0)
      return _data
    };
```

```
  colors = [
    "#4e79a7",
    "#f28e2c",
    "#e15759",
    "#76b7b2",
    // "#59a14f",
    "#edc949",
    "#af7aa1",
    "#ff9da7",
    "#9c755f",
    "#bab0ab"
  ]
```

```
  {
    const width = 20;
    const height = 150;
    const svg = d3.create("svg").attr("width", 300).attr("height", 160);

   const g = svg.selectAll("g")
    .data(groupDataFunc(ChimneyData1))
    .join("g");

    g.append("rect")
      .attr("x", 30)
      .attr("y", d => xScale(d.cumulative))
      .attr("width", 20)
      .attr("height", d => xScale(d.value))
      .style("fill", (d, i) => colors[i])
      .append("title")
      .text(d => `${d.label} : ${d.value}`);

    return svg.node()
  }
```

```
import { Legend, Swatches } from "@d3/color-legend"
```