REMIND ABOUT SURVEY

const loadScatter = async function() {

const yearExtent = d3.extent(data, d => d['year']);

const yearScale = d3.scaleLinear().domain(yearExtent).range([0, chartWidth]);

const grossExtent = d3.extent(data, d => d['worldwide\_gross']);

const grossScale = d3.scaleLog().domain(grossExtent).range([chartHeight, 0]);

const imdbExtent = d3.extent(data, d => d['imdb\_rating']);

const imdbScale = d3.scaleLinear().domain(imdbExtent).range([3,8]);

const genreScale = d3.scaleOrdinal(d3.schemeCategory10);

let leftAxis = d3.axisLeft(grossScale)

.tickFormat(d3.format("$.0s"));

let leftGridlines = d3.axisLeft(grossScale)

.tickSize(-chartWidth-10)

.tickFormat("")

annotations.append("g")

.attr("class", "y axis")

.attr("transform", `translate(${margin.left-10},${margin.top})`)

.call(leftAxis)

annotations.append("g")

.attr("class", "y gridlines")

.attr("transform", `translate(${margin.left-10},${margin.top})`)

.call(leftGridlines);

let bottomAxis = d3.axisBottom(yearScale)

.tickFormat(d3.format("~f")); // exact value

let bottomGridlines = d3.axisBottom(yearScale)

.tickSize(-chartHeight-10)

.tickFormat("")

.attr("transform",`translate(${margin.left},${chartHeight+margin.top+10})`)

REMIND ABOUT CSS

let circles = chartArea.selectAll("circle.point").data(data)

.join("circle")

.attr("class", "point")

.attr("label", d => d["title"])

.attr("genre", d => d["Main\_Genre"])

.attr("opacity", 0.8)

.attr("cx", d => yearScale(d['year']))

.attr("cy", d => grossScale(d['worldwide\_gross']+1))

.attr("r", d => imdbScale(d['imdb\_rating']))

.attr("fill", d => genreScale(d['Main\_Genre']);

// NOTE VERSION DIFFERENCES

circles.on("mouseover", function(event, d) {

d3.select(this)

.transition().duration(200)

.attr("stroke","black")

.attr("stroke-width", 4)

.attr("fill", lighten( genreScale(d['Main\_Genre']) )) ;

d3.select("#label").text(d['title']);

});

// Make sure to clean up effects on mouseout

circles.on("mouseout", function(event, d) {

d3.select(this)

.transition().duration(200)

.attr("stroke","")

.attr("stroke-width", 1)

.attr("fill", genreScale(d['Main\_Genre']) );

d3.select("#label").text("");

});

----------------------------------- MAP

var states = topojson.feature(us, us.objects.states);

var statesMesh = topojson.mesh(us, us.objects.states);

var projection = d3.geoAlbersUsa().fitSize([mapAreaWidth, mapAreaHeight],

states);

var path = d3.geoPath().projection(projection);

console.log(states);

console.log(statesMesh);

var graticule = d3.geoGraticule10();

map.append("path").attr("class","graticule").attr("d", path(graticule) )

map.selectAll("path.state").data(states.features)

.join("path")

.attr("class", "state")

.attr("note", d => d.id) // debugging

.attr("d", path);

map.append("path").datum(statesMesh)

.attr("class","outline")

.attr("d", path);

const minMax = d3.extent( Object.values(stateIdCounts) );

console.log(minMax);

const colorScale = d3.scaleQuantile()

.domain(Object.values(stateIdCounts))

.range(["#fff","#d1e8ed","#adc2da","#8879b3","#762b80"]);

map.selectAll(".state")

.style("fill", d => colorScale( stateIdCounts[d.id]) );

var sim = d3.forceSimulation()

.nodes(nodes)

.force("links", d3.forceLink()

.links(links)

.id( d => d['id'] ) )

.force("repulse", d3.forceManyBody().strength(-150) )

.force("center", d3.forceCenter(width/2.0, height/2.0))

.on("tick", render);

function render() {

let lines = layer.selectAll("line.link").data(links)

.join(

enter => enter.append("line")

.attr("class","link")

.attr("stroke","#333")

)

.attr("x1", d => d.source.x).attr("x2", d => d.target.x)

.attr("y1", d => d.source.y).attr("y2", d => d.target.y);

let circles = layer.selectAll("circle.node").data(nodes)

.join(

enter => enter.append("circle")

.attr("class","node")

.attr("stroke", "#333")

.attr("r", 8)

.attr("cx", 0)

.attr("cy", 0)

.attr("fill", d => {

if (d.id === 1) { return "#0C0"; }

else if (d.id === 34) { return "#C00"; }

else { return "steelblue" }

})

// NOTE VERSION DIFFERENCES

.call( d3.drag().on("start",dragstart)

.on("drag",dragging)

.on("end",dragend) )

)

.attr("transform", d => `translate(${d.x},${d.y})`);

}

render();

function dragstart(event, d) {

if (!event.active) {

label.text(d.id);

sim.alphaTarget(0.08).restart();

}

d.fx = event.x;

d.fy = event.y;

label.attr("x",event.x + 5).attr("y",event.y);

}

function dragging(event, d) {

d.fx = event.x;

d.fy = event.y;

label.attr("x",event.x + 5).attr("y",event.y);

}

function dragend(event, d) {

if (!event.active) {

sim.alphaTarget(0);

label.text("");

}

d.fx = null;

d.fy = null;

}