Lab 4 – Netflix Data Information

Data Sources:

- custom.geo.json – geojson website
- netflix2019.csv – Kaggle

Data Understanding:

Netflix Content:

As Netflix has soared in popularity, a partial reason for that success was the contracts it gained for bringing on several TV shows and Movies and Standup Specials altogether in one place, making it easy for the general audience to access at a low rate rather than spending a lot more money and paying several TV channels to access the same content, as well as it was all free of Ad's. Thus with the general popularity of Netflix rising, the amount of the content added in the different formats also accordingly rose along with it.

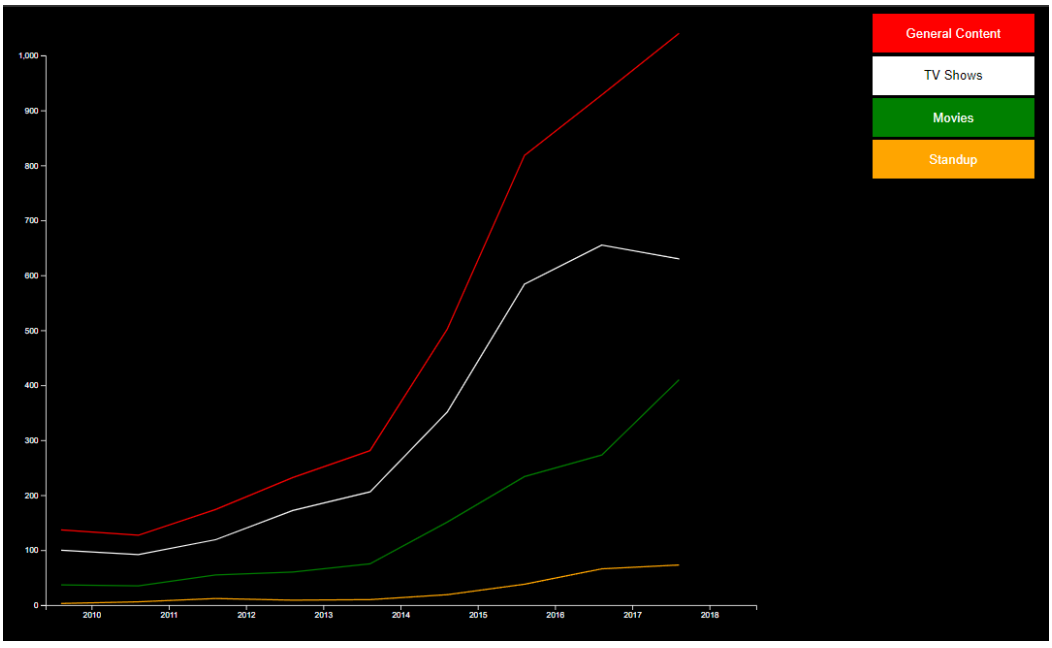

Fig 1. Line chart of all the content

Above we can see that the amount of general content added to the platform had dramatically risen and in correlation with that was the amount of TV shows it had on its platform. A reason for this occurring was because of TV Shows made by Netflix itself, when the popularity rose for the platform, it had opened it itself up to a very large audience thirsty for more content, and the way it could get around contracting and giving away portions of its profits to its contractors was by creating its own shows. This trend started to increase further around 2013-2014 and ever since then Netflix has been

creating shows quite regularly, now being that it has diversified the amount of content that it has on its platform to gain the attention of the several types of customers that are looking for different types of content based on their own unique tastes.

Another interesting thing that has started to occur is the amount of stand-up content that Netflix is providing their customer base, not only are they giving comedians a chance to make their break on stage but also creating the opportunity to bring their audience something that they generally would have not seen because it would be usually done in the setting of a bar or something similar. Netflix gave the stand-up scene a new life and had encouraged several other comedians to come on their platform. It has become so popular, that now getting a Netflix special is considered to be making a break in the Stand-up scene itself amongst comedians.

The general quantity of movies that have been added in the past few years of Netflix has not seen a dramatic increase such as that of TV Shows. A reason for this occurring is because of the time commitment to provide a movie holds a larger cost than that of a episode or a couple episodes of a TV show. A few of the most popular TV shows on the platform such as Friends and The Office have relatively lower time commitment needs (at on average 20-30 minutes per episode) than the average movie. Because of this lower time commitment, it does two things, firstly being it makes it easier for the user to choose between the two types, with the TV Shows often winning. Secondly, it psychologically fools the person into watching far more episodes (binge watching), often exceeding the amount of time a movie would have taken, due to the addition of cliff-hangers and other directive choices when designing the plot of the episode which hooks the watcher on, causing them to spend more time on the TV Shows. Consequently, TV Shows are far more profitable than Movies, thus TV Shows saw a dramatic increase in the amount of content added to the platform in the place of movies.

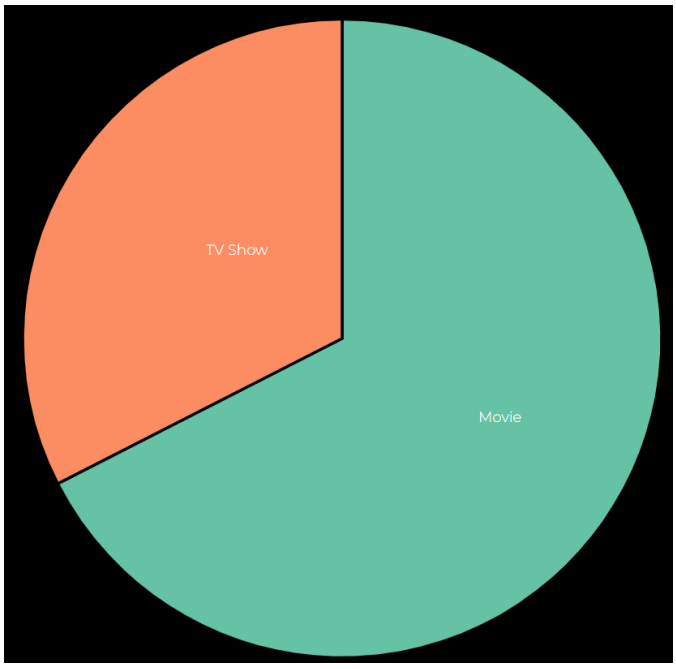

Fig 2. Pie chart of ratio between tv shows to movies on netlfix

But although the type of content that Netflix has been recently putting onto their platform has largely been TV Shows instead of Movies, the general amount of content that exists on the platform is largely Movies in comparison to the quantity of TV shows that are present on Netflix. With the rate with which Netflix has been adding tv shows to the platform, the amount of tv shows will soon overtake the number of movies that are present on the platform.

From where?

General Content

As the amount of Netflix user increased along with rise of popularity of the platform itself, the variety of the customer base had also diversified considerably. Initially being an American company, the company originally was intended to provide content for an American audience, consequently a lot of the of the content present on the platform was of American origin, but as the customer base diversified and several other customers from other countries could gain access to the platform, the variety of languages and other such requirements were also required to be met as well. Down below we see the Top 10 number of countries which have Netflix content largely related with that country's audience. The highest of them being America, India, and the UK. This is due either the large population residing within the country as well as the popularity of their film industry and the language in which they speak, leading to a larger part of the customer base being able to enjoy that same content as well.

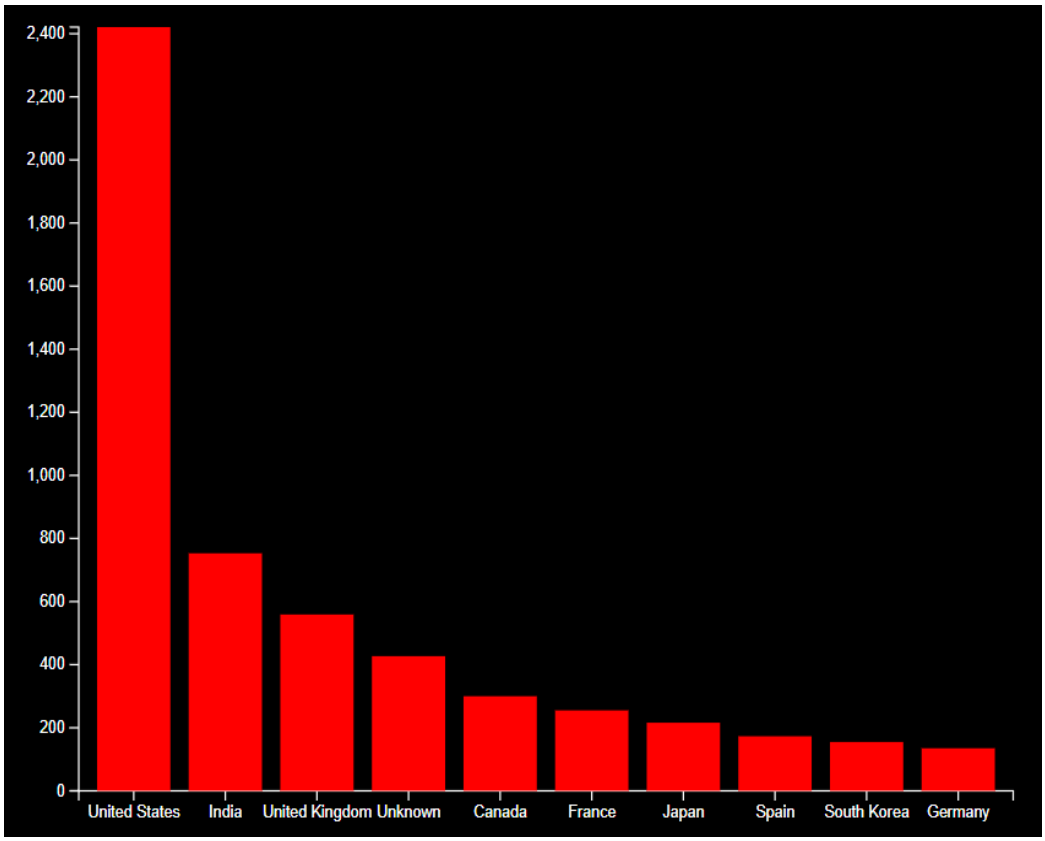

Fig 3. Bar chart of the countries with the most Netflix content

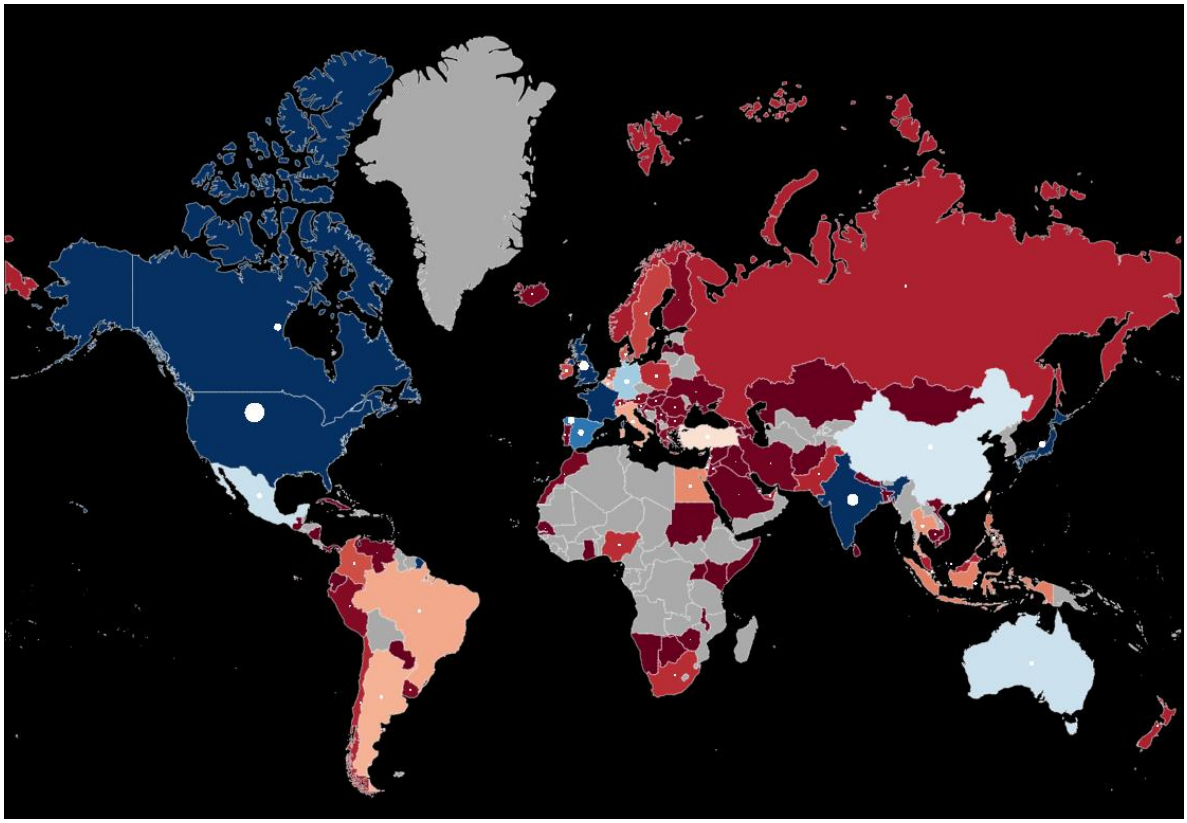

Fig 4. World map shows countries with the most content in blue, and the least in red

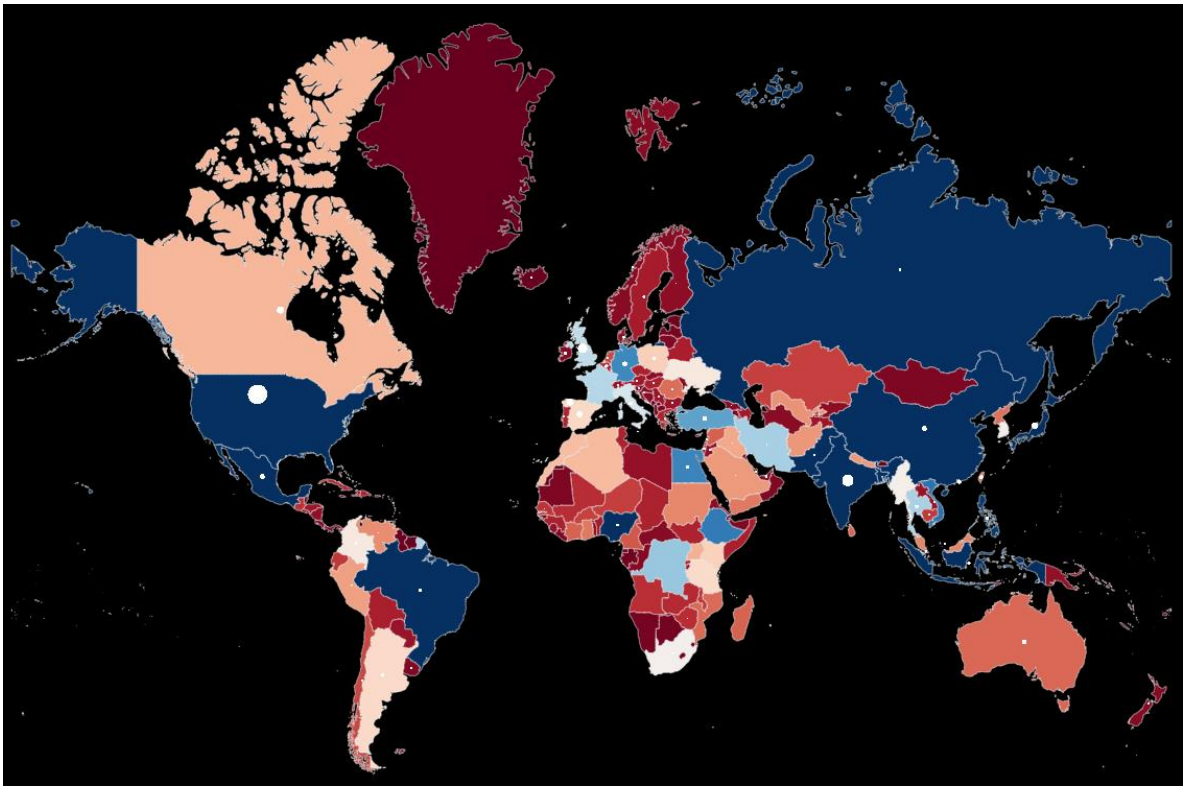

Fig 5. World map showing the population of the different countries, Blue being highest, red being the lowest. The size of the dots shows the amount of content produced that linked with the country.

Carrying over from the trend of the origin of the content in the previous paragraph, there is a similar trend but more a pattern that can be observed for the Stand-up content on Netflix is one of being more closely related to the language the content is presented in. As we can see with most of the content that is presented is from the United States, the other countries that have also presented their stand-up content have a population that speaks a very popular language within the current modern world. Most of the countries on the list presented below are either of a Spanish speaking origin or have an English-speaking population, with the exception of Italy and South Korea. Even then Italy and France have languages that are quite widely spoken through the world with only the most realistic outlier being South Korea.

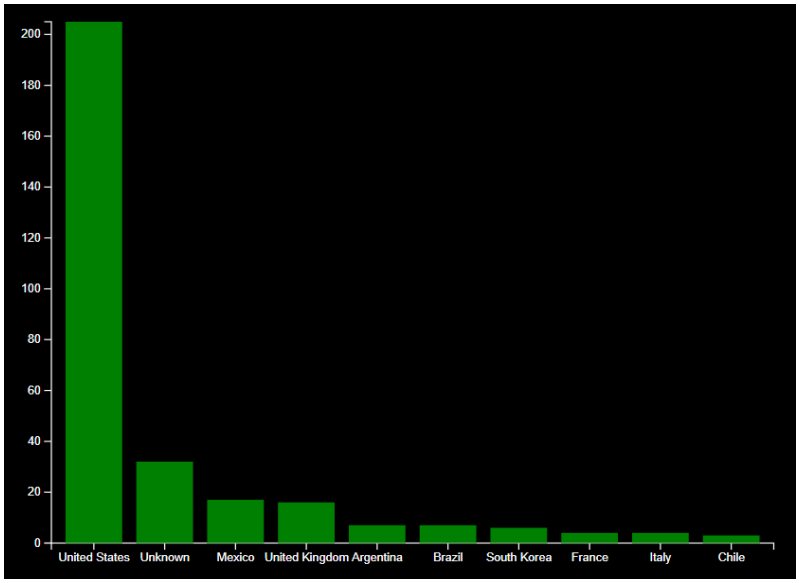

Fig 6. Bar chart showing the top 10 countries with the most stand-up content

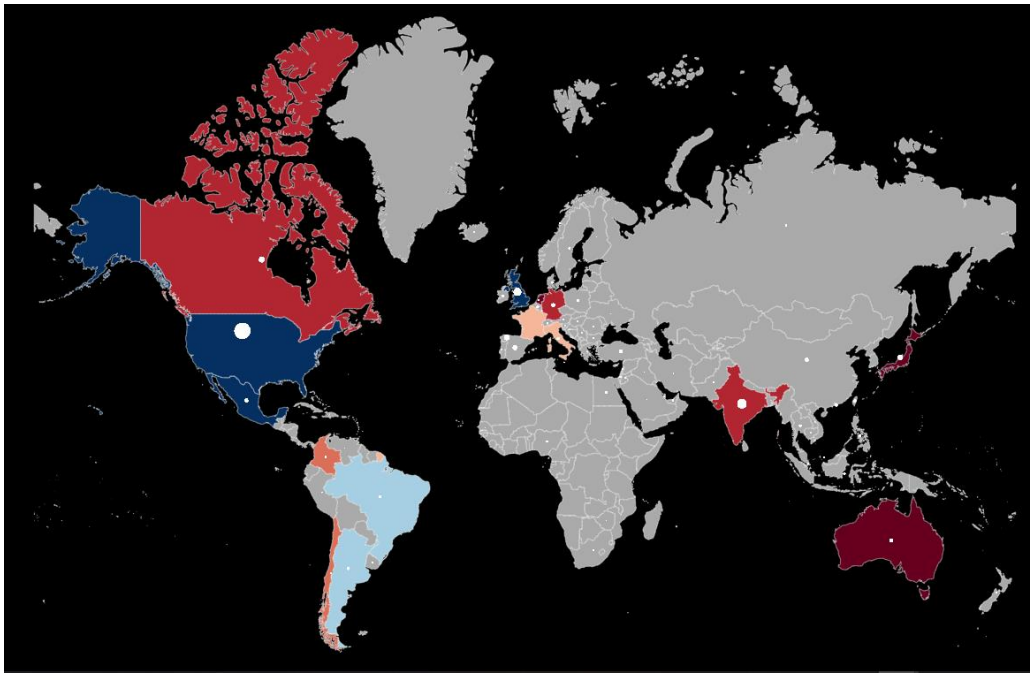

Fig 7. World map showing all the countries with stand-up content, blue being highest and red being the lowest. The size of the dots shows the amount of content produced that linked with the country.

Economic Status:

In correlation to the sort of content provided by each country, all the countries here are colour coded in the group of the respective economic status. Highly affluent countries, such as the countries in Western Europe, the Americas and Australia have content which can be produced at mass and can also be highly influential due to their potential of reach to higher audience count. Languages such as English, French, and Spanish have the most potential to reach a larger audience than for example languages such as Hindi, Russian or Japanese. Thus, a lot of the time, popular content made in English, French or Spanish are far more profitable and can overturn the population countries such as India have.

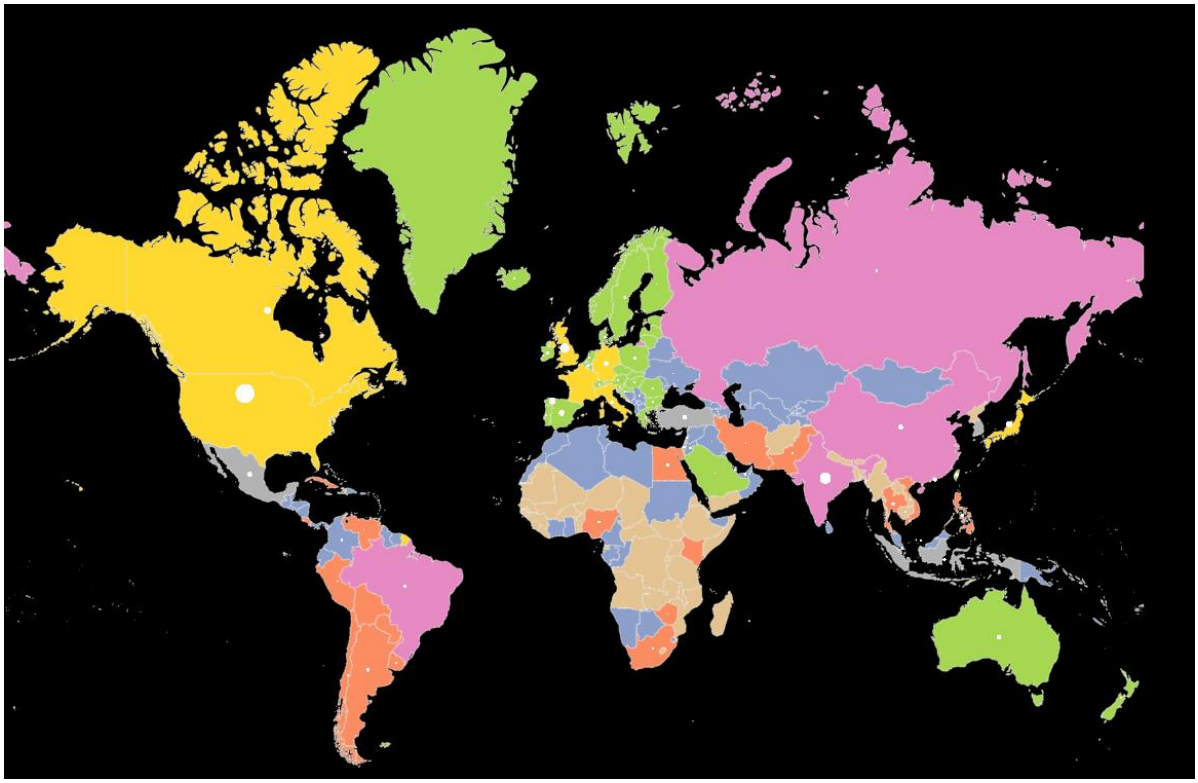

Fig 8. World Map showing countries categorised into different economic brackets. The size of the dots shows the amount of content produced that linked with the country.

Ratings

The amount of content based on the rating of the content that is published on the platform is very reflective of the audience watching the content itself. The categorisations are provided below. Most content is meant for a Mature audience, along with which another large amount of content that is directed towards kids and teens which would take up a larger portion of the Netflix content space. A lot of content would be directed towards that demographic as that demographic in general tend to have for more time to spare and watch, thus they would be more likely to engage with content on Netflix. The mature content provided on the platform can also be attested to the variety of topics and subjects such as Crime, Relationships, etc, that can be explored compared a bigger limit as well the type of emphasis placed on content provided to younger audience.

- Little Kids: G, TV-Y, TV-G
- Older Kids: PG, TV-Y7, TV-Y7-FV, TV-PG
- Teens: PG-13, TV-14
- Mature: R, NC-17, TV-MA

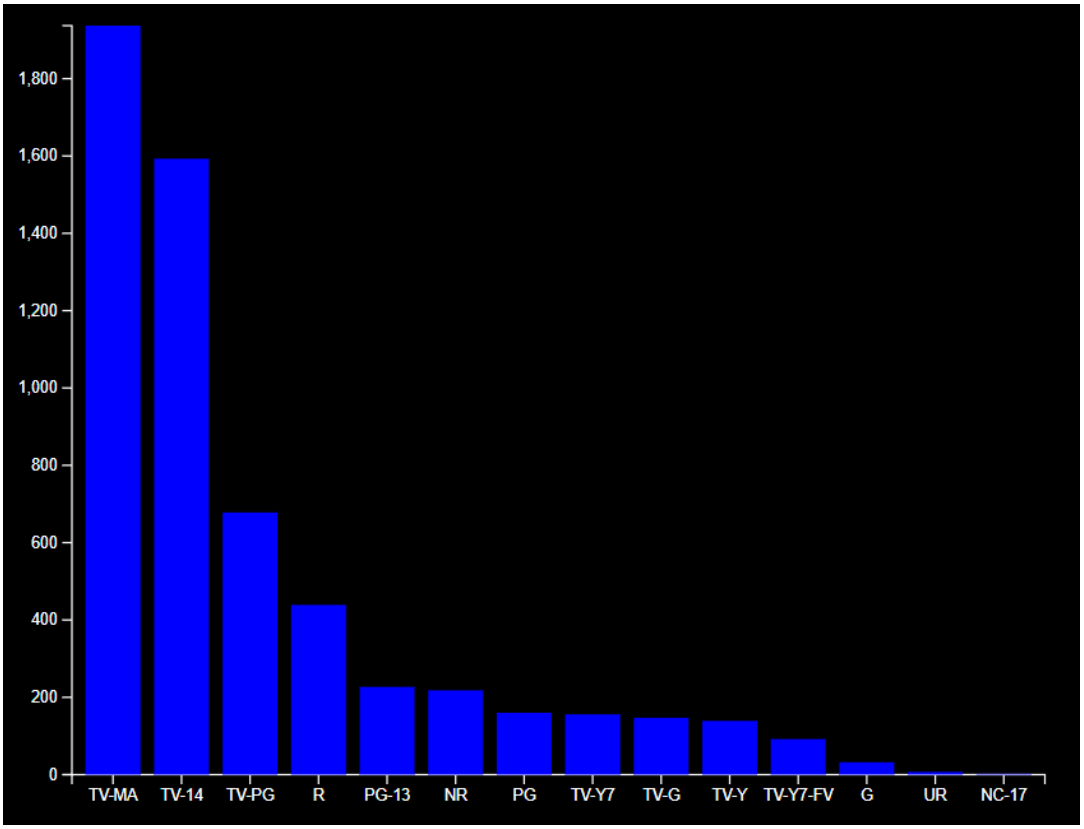

Fig 9. Bar chart showing the amount of the content per rating tag

Durations

TV Series'

As there has been a upsurge in the demand of TV Shows recently in the past few years, Netflix has continually added a large amount of shows, most of which were experimental in the hopes that they would reach some form of long term popularity with the audience on their platform, thus most shows on the platform only have one season. Shows which did well one their first season, often get a few other seasons as well, but based on budgeting and also the writing for the show, if it stands the test of time through the second season, it often gets a few more seasons until its end does come. Very few shows can get past 3 seasons as the popularity generally fades quick unless the show is somehow represented in mainstream culture on the internet.

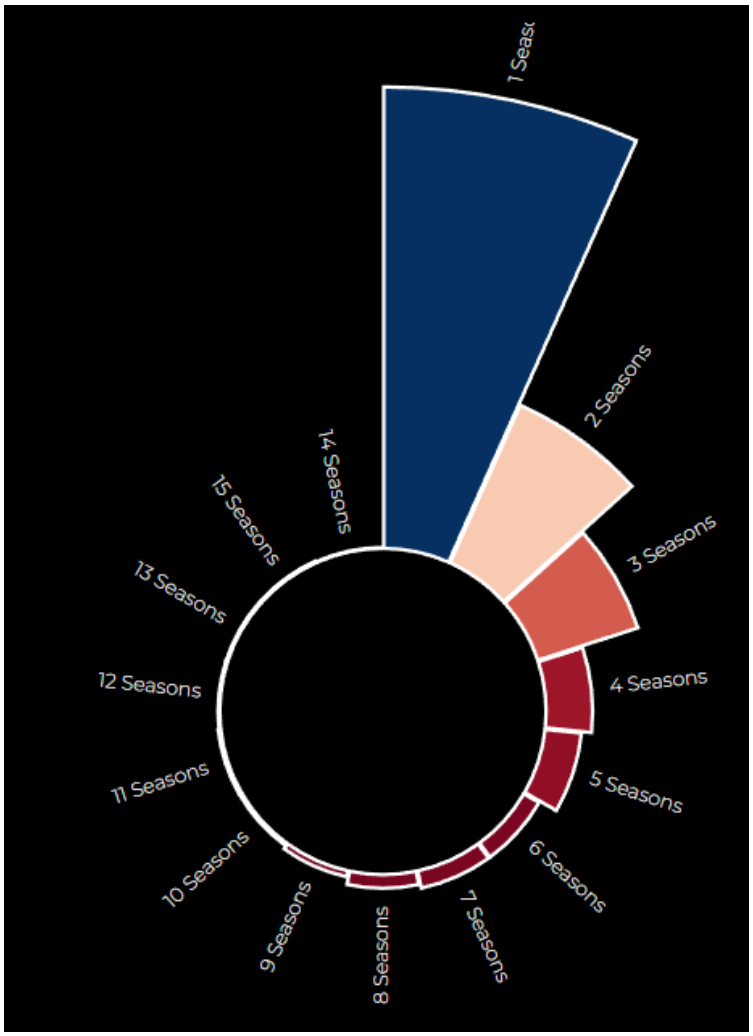

Fig 10. Circular bar chart showing the amount of shows according to their respective amount of seasons.

Generally, the most well to do shows tend to get high amounts of seasons under their belt, the shows below are the shows that have stood the test of time over a long period and have been able to create several seasons as viewer demand increases. Shows such as The Office (US) are shows which are an example of long running fan favourite shows. The show although finished a few years back still carries a legacy of it being a very high-quality comedy show and is still loved by several people across the globe. Shows such as this are also the reason why people want to own a Netflix membership due to the ease of access to such long running, binge worthy shows all accessible in one place without illegal means. This is beneficial to Netflix as they get to keep a loyal and content customer base.

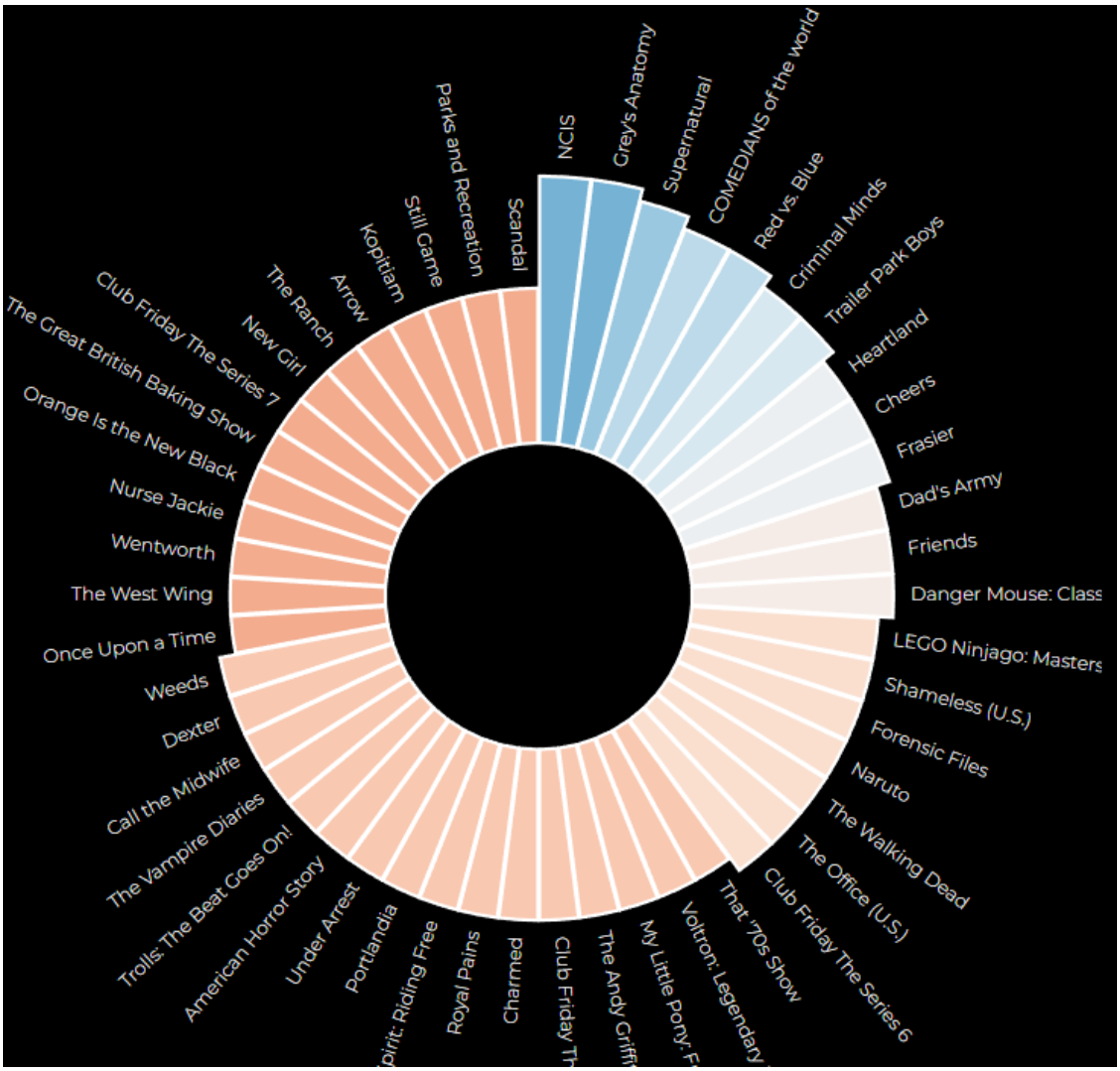

Fig 11. Top 50 shows in a circular bar chart, each with size corresponding to the amount of seasons it has.

Movie durations

A lot of the movies shown below are either of Hollywood Origin or Bollywood Origin, which suggests the major cinematic industries represented within Netflix are from America and India. This well corresponds as we had earlier discussed the amount of content present in both countries is spectacularly high, one is due to the reach of their language as a large portion of the world is capable of understanding English, the other is due to the high population within the country, so although the movies are Indian, there is a large enough market within India that speaks Hindi and can enjoy the movies as well. It also seems that in general, a large portion of Indian movies tend to be of a longer duration than movies in English.

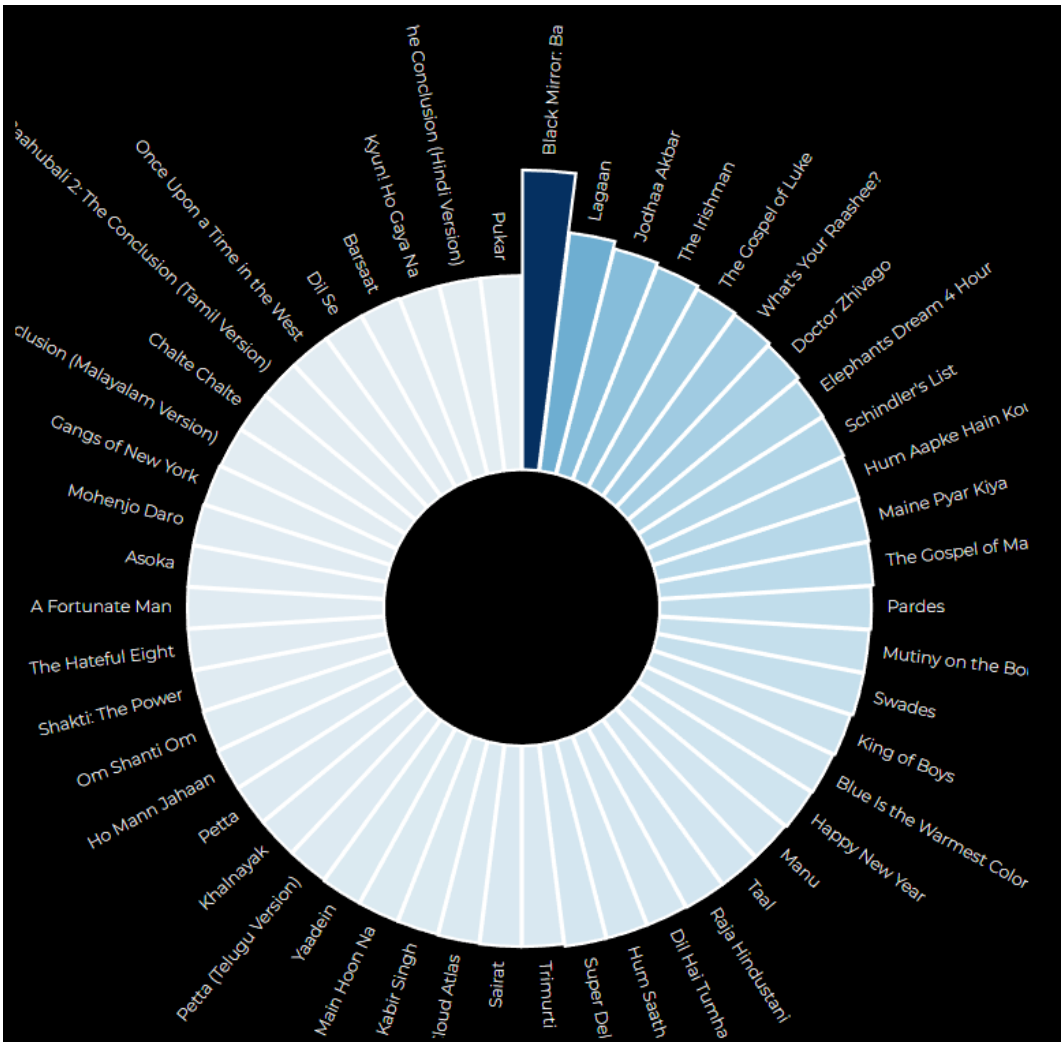

Fig 12. Circular bar chart showing the duration times for the top 50 longest movies

Another interesting this that was discovered is that a large portion of movies were under 100 minutes that were existent on Netflix. Below we have presented a similar ratio of the content where its longest movies to the shortest movies. Movies that tend to be shorter tend to be English based movies whereas Indian movies tend to be far longer, a reason to explain this phenomenon would be that Indian movies tend to be like musicals where a lot of songs are a part of the movie and accompany the movies mood as well. Thus with a combination of songs and the plot, the movie may often take longer to complete than several English movies.

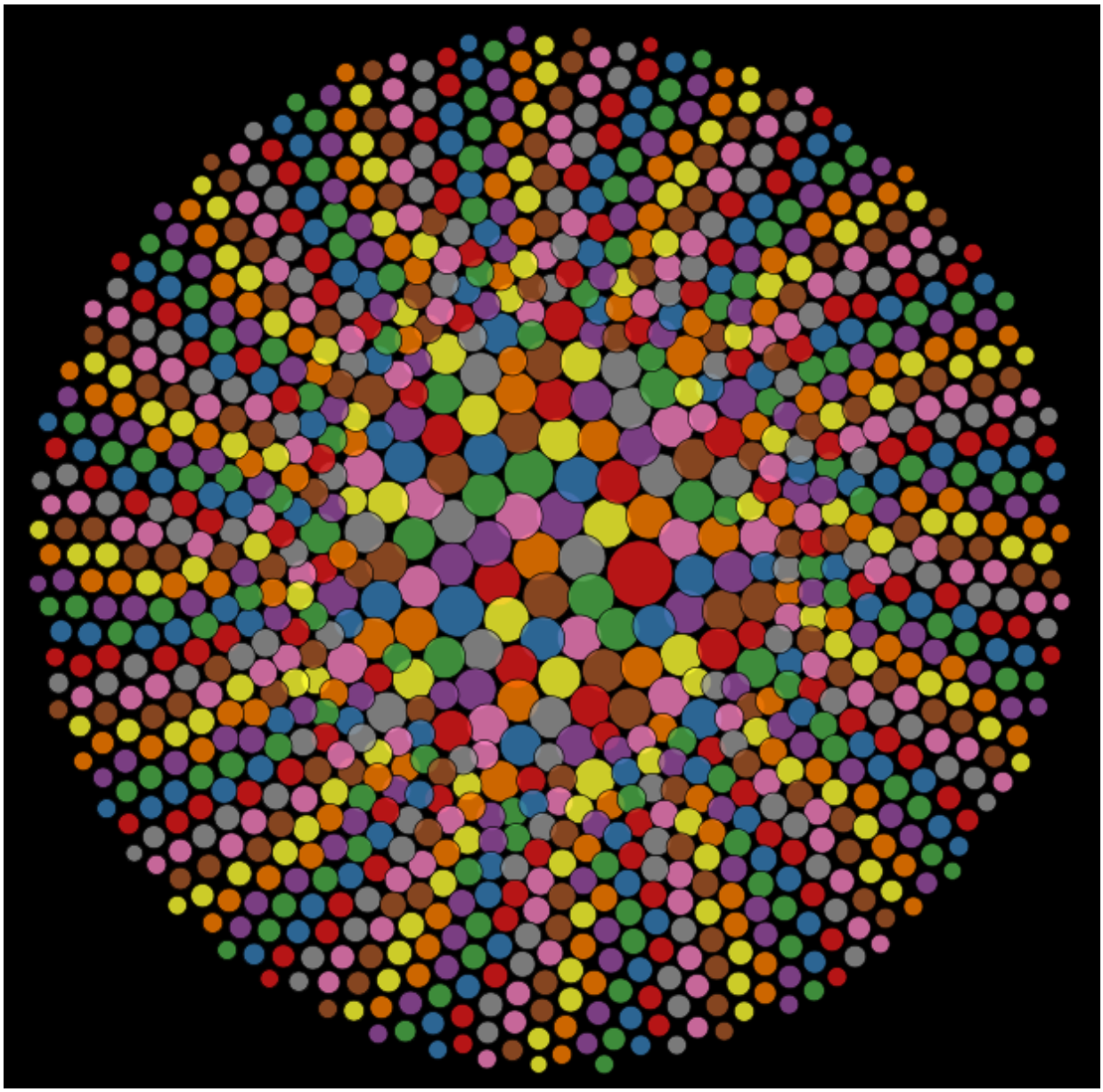

Fig 13. Nodes forced to the middle showing the 600 different movies, top 100 longest movies and the top 500 shortest movies.

Genres

The most genres on the platform are a reflection of the audiences tastes and which shows generally do better than others.

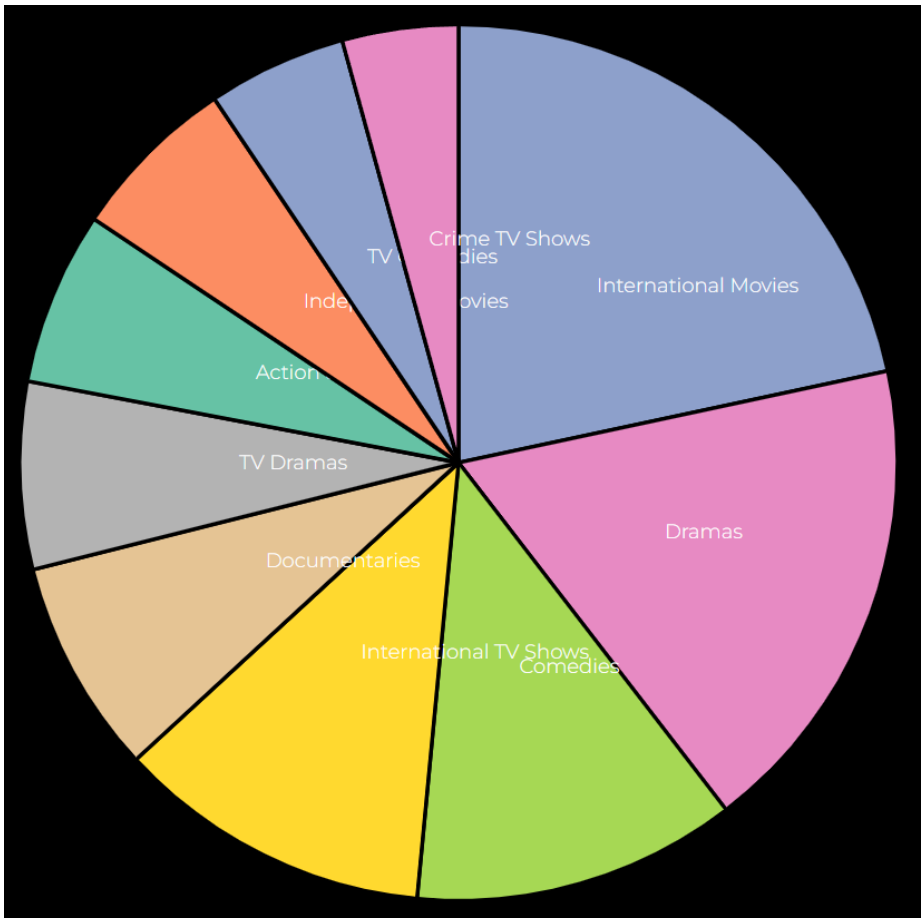

Fig 14. Genres pie chart

The most popular genres that generally do well are the International TV Shows and Movies as there is a large international audience outside of America that are generally accessing content produced on their countries on Netflix, this means that nearly 33 percent of the content is catered towards an international crowd. Furthermore, Dramas tend to be a very popular. They may have a high popularity rate due to their emotional highs and lows and the extremes they portray in society which would fascinate people because that is something you would not actually see in the real world yet the concepts, they present are very real parts of our own lives. Other genres tend to be smaller as they are targeted towards specific audiences whilst the bigger genres are highly flexible and malleable and thus can serve as an underlying genre for several other genres.

Code Explanation

Bar chart

By using Promise.all, I loaded in several different csv's, changed their count value into integers and pushed the csv file into a array

```
        //load data
Promise.all([
    d3.csv('countries.csv'),
    d3.csv('standup_by_country.csv'),
    d3.csv('ratingstypes.csv'),
]).then(([csv,standupcsv,ratingcsv]) => {
    csv.forEach(function(d) {
        d.count = parseInt(d.count)
        content.push(d)
    })
    standupcsv.forEach(function(d) {
        d.count = parseInt(d.count)
        standup.push(d)
    })
    ratingcsv.forEach(function(d) {
        d.count = parseInt(d.count)
        rating.push(d)
    })
```

Fig 15. Loading data

```
top10content = content.slice(0,10)
top10standup = standup.slice(0,10)

var x = setX(top10content, "country")
var y = setY(top10content, "country")
```

Fig 16. Getting top 10 countries and setting the scales

I made functions to set up scales dynamically

```
function setX(data, group){
    var x = d3.scaleBand()
        .range([ 0, barwidth ])
        .domain(data.map(function(d) { return d[group]; }))
        .padding(0.2);
        return x;
}

function setY(data){
    var y = d3.scaleLinear()
    .domain([0, d3.max(data, function(d) { return d.count; })])
    .range([ barheight, 0]);
    return y;
}
```

Fig 16. Scaling functions

I created an update function so I would be able to show several types of information, I reset the axis and their scale according to the incoming data

```
function update(data, colour, group) {
    s = setX(data, group);
    z = setY(data);

    // console.log(data)

    svg.select(".bottom")
            .attr("transform", "translate(0," + barheight + ")")
            .attr("color", "white")
            .call(d3.axisBottom(s));

    svg.select(".left")
    .attr("color", "white")
            .call(d3.axisLeft(z));
```

Fig 17. Update for bar graph and resetting the axis

```
var u = svg.selectAll("rect")
    .data(data);
u
    .enter()
    .append("rect")
    .merge(u)
    .transition()
    .duration(1000)
    .attr("x", function(d) {
        console.log("drawn")
        return s(d[group]);
    })
    .attr("y", function(d) {
        return z(d.count);
    })
    .attr("width", s.bandwidth())
    .attr("height", function(d) { return barheight - z(d.count); })
    .attr("fill", colour);

    u.exit().remove();
```

Fig 18. Drawing bars to represent the information and removing the previous bars

Pie charts

Here I set the type of colour scheme I would like to use on my pie chart as well as the radius that I would be using when creating my pie chart. As well as I set up the pie chart variables that I would be using when drawing my pie chart.

```
const radius = Math.min(xSize, ySize) / 2;
var color = d3.scaleOrdinal().range(d3.schemeSet2);

// Generate the pie
var pie = d3.pie();
// Generate the arcs
var arc = d3.arc()
    .innerRadius(0)
    .outerRadius(radius);
```

Fig 19. Radius and Colour and setting up the pie chart

Once I set up everything, I loaded up all the content that I required

```
Promise.all([
    d3.csv("amount_types.csv"),
    d3.csv('top10genres.csv'),
]).then(([data, genres]) => {
    for (i in data) {
        type.push(data[i].type)
        count.push(data[i].count)
    }
    type.pop()
    count.pop()
    console.log(type)
    console.log(count)
    for (i in genres) {
        genre.push(genres[i].genre)
        genrecount.push(genres[i].count)
    }
    genre.pop()
    genrecount.pop()
    console.log(genre)
    console.log(genrecount)

    update(type, count)
})
```

Fig 20. Loading data

Once I accomplished that, I drew the entire pie chart inside a update function so I could use it several times.

```
function update(type, count){
    //Generate groups
    var arcs = piechart.selectAll("arc")
        .data(pie(count))
        .enter()
        .append("g")
        .attr("class", "arc")
        .text(function(d, i){
            console.log(i)
            return d.data;
        });
    //Draw arc paths
    arcs.append("path")
        .attr("fill", function(d, i) {
        return color(d.data);
        })
        .attr("d", arc)
        .attr("stroke", "black")
        .attr("stroke-width", "3px")
        .transition()
        .duration(1000)
        .attrTween("d", function (d) {
            var i = d3.interpolate(d.endAngle, d.startAngle);
            return function (t) {
                d.startAngle = i(t);
                return arc(d);
            }
        });

    arcs.append("text")
        .attr("transform", function(d) {
            return "translate("+ arc.centroid(d)+")";
        }) // arc.centroid function used in order to compute the center of the arc})
        .attr("fill", "white")
        .text(function(d, i){
            console.log(type)
            return type[i];
        });
```

Fig 21. Update functioning for drawing the pie chart

Line Chart

I created a line chart function which would allow me to draw multiple lines at once on the same graph.

```
function lineGraph(data, data2, data3, data4) {

    yExtent = [1000, 0];
    // Append SVG Object to the Page
    const svg = d3.select(".linechart")
        .append("svg")
        .attr('width', xSize )
        .attr('height', ySize )
        .append("g")
        .attr("transform","translate(" + margin + "," + margin + ")");

    function setX(data, group){
        var x = d3.scaleBand()
            .range([ 0, xMax ])
            .domain(data.map(function(d) { return d[group]; }))
            .padding(0.2);
            return x;
    }

    function setY(data){
        var y = d3.scaleLinear()
        .domain([yExtent[1], yExtent[0]])
        .range([ yMax, 0]);
        return y;
    }

    // X Axis
    const x = setX(data, 'x');

    // bottom
    svg.append("g")
        .attr("transform", "translate(0," + yMax + ")")
        .call(d3.axisBottom(x))
        .attr('color', 'white'); // make bottom axis green
```

Fig 22. Including the two code snippets below, this is the line chart function

```
// Y Axis
const y = setY(data)

// left y axis
svg.append("g")
    .call(d3.axisLeft(y))
    .attr('color', 'white');


// Add the line for general releases
var u1 = svg.append("path")
    .datum(data)
    .attr("fill", "none")
    .attr("stroke", "red")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
    .x(function(d) {
        console.log("general: " + d.x)
        return x(d.x)
    })
    .y(function(d) {
        console.log(d.y)
        return y(d.y)
    }));

// Adding the line for standup
var u2 = svg.append("path")
    .datum(data2)
    .attr("fill", "none")
    .attr("stroke", "green")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
    .x(function(d) {
        console.log("standup: " + d.x)
        return x(d.x)
    })
    .y(function(d) {
        console.log(d.y)
        return y(d.y)
    }));
```

```
// Adding the line for shows
var u3 = svg.append("path")
    .datum(data3)
    .attr("fill", "none")
    .attr("stroke", "orange")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
    .x(function(d) {
        console.log("shows " + d.x)
        return x(d.x)
    })
    .y(function(d) {
        console.log(d.y)
        return y(d.y)
    }));

// Adding the line for movies
var u4 = svg.append("path")
    .datum(data4)
    .attr("fill", "none")
    .attr("stroke", "white")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
    .x(function(d) {
        console.log("movies: " + d.x)
        return x(d.x)
    })
    .y(function(d) {
        console.log(d.y)
        return y(d.y)
    }));
```

Using Promise.all I loaded in all of the data and drew the line chart.

```
release = []
standup = []
shows = []
movies = []
Promise.all([
    d3.csv('release_year.csv'),
    d3.csv('standup_by_year.csv'),
    d3.csv('shows_release.csv'),
    d3.csv('movies_release.csv'),
]).then(([releasecsv, standupcsv, showscsv, moviescsv]) => {
    releasecsv.forEach(function(d) {
        d.count = parseInt(d.count)
        release.push({x: d.year, y: d.count});
    })
    standupcsv.forEach(function(d) {
        d.count = parseInt(d.count)
        standup.push({x: d.year, y: d.count});
    })
    showscsv.forEach(function(d) {
        d.count = parseInt(d.count)
        shows.push({x: d.year, y: d.count});
    })
    moviescsv.forEach(function(d) {
        d.count = parseInt(d.count)
        movies.push({x: d.year, y: d.count});
    })

    lineGraph(release, shows, standup, movies);
});
```

Fig 23. Loading the data

Circular Bar Chart

After loading in the required data, I created a function which would allow me to draw a circular bar chart several times

```
var clr = d3.scaleSequential().domain([1, max]).interpolator(d3.interpolateRdBu);
// Scales
const x = d3.scaleBand()
    .range([0, 2 * Math.PI])    // X axis goes from 0 to 2pi = all around the circle. If I stop at 1Pi, it will be around a half circle
    .align(0)                  // This does nothing
    .domain(data.map(d => d.country)); // The domain of the X axis is the list of states.
const y = d3.scaleRadial()
    .range([innerRadius, outerRadius])   // Domain will be define later.
    .domain([0, max]); // Domain of Y is from 0 to the max seen in the data
```

Fig 25. Set up the colour and scaled the data as need.

I appended to the svg in question and created the bar charts based on the values I collected and also created a mouseover function which would highlight over the bar that the mouse is hovering upon.

```
// Add the bars
svgtype.append("g").selectAll("path")
    .data(data)
    .join("path")
    .attr("fill", d=>clr(d['count']))
    .attr("stroke", "white")
    .attr("stroke-width", "2px")
    .attr("d", d3.arc()     // imagine your doing a part of a donut plot
        .innerRadius(innerRadius)
        .outerRadius(d => y(d['count']))
        .startAngle(d => x(d.country))
        .endAngle(d => x(d.country) + x.bandwidth())
        .padAngle(0.01)
        .padRadius(innerRadius))
    .on("mouseover", function(d) {
        d3.select(this)
            .attr("fill", "white")
            .attr("stroke", "white")
            .attr("stroke-width", "2px")

        d3.select("#tooltip")
            .style("left", (d3.event.pageX) + "px")
            .style("top", (d3.event.pageY - 28) + "px")
            .style("display", "inline-block")
            .html((d.country) + "<br>" + (d['count']));
    })
    .on("mouseout", function(d) {
        d3.select(this)
            .attr("fill", d=>clr(d['count']))
            .attr("stroke", "white")
            .attr("stroke-width", "2px")
        d3.select("#tooltip").style("display", "none");
    });
```

Fig 26. Adding the bars

As the bars by themselves would not make much sense, I added some accompanying text to show what the bar was representing.

```
// Add the labels
svgtype.append("g")
    .selectAll("g")
    .data(data)
    .join("g")
        .attr("text-anchor", function(d) { return (x(d.country) + x.bandwidth() / 2 + Math.PI) % (2 * Math.PI) < Math.PI ? "end" : "start"; })
        .attr("transform", function(d) { return "rotate(" + ((x(d.country) + x.bandwidth() / 2) * 180 / Math.PI - 90) + ")"+"translate(" + (y(d['count'])+10) + ",0)"; })
    .append("text")
        .text(function(d){return(d.country)})
        .attr("transform", function(d) { return (x(d.country) + x.bandwidth() / 2 + Math.PI) % (2 * Math.PI) < Math.PI ? "rotate(180)" : "rotate(0)"; })
        .style("font-size", "11px")
        .attr("fill", "white")
        .attr("alignment-baseline", "middle")
```

Fig 27. Adding text

Nodes

After loading up all the data, I created a function where I first set up the colour and scaled the data according to what I needed, which after I created a tooltip which would allow me to get the information of a node which my mode hovered upon.

```
function update(data,variable) {

    // Color palette
    const color = d3.scaleOrdinal().range(d3.schemeSet1);
    // Size scale for countries
    function setSize(data){
        const size = d3.scaleLinear()
            .domain([0, d3.max(data, d => d.value)])
            .range([7,30])  // circle will be between 7 and 55 px wide

        return size
    }

    const size = setSize(data);

    // create a tooltip
    const Tooltip = d3.select("#my_dataviz")
        .append("div")
        .style("opacity", 0)
        .attr("class", "tooltip")
        // .style("background-color", "white")
        .style("border", "solid")
        .style("border-width", "2px")
        .style("border-radius", "5px")
        .style("padding", "2px")
        .style("margin-right", "400px")
        .style("margin-left", "400px")
        .style('color', 'white')
```

Fig 28. Creating a tooltip and scaling data and setting colour scheme

I also created a few mouse functions which would allow me to see over the data point node that I am hovering upon with the cursor.

```
// Three function that change the tooltip when user hover / move / leave a cell
const mouseover = function(event, d) {
    Tooltip
    .style("opacity", 1)
}

const mousemove = function(event, d) {
    Tooltip
    .html('<u>' + d[variable] + '</u>' + "<br>" + d.value)
    .style("left", (event.x/2+20) + "px")
    .style("top", (event.y/2-30) + "px")
}
var mouseleave = function(event, d) {
    Tooltip
    .style("opacity", 0)
}
```

Fig 29. Mouse functions

Once I set up the things, I required for drawing my node, I then drew the one itself

```
// Initialize the circle: all located at the center of the svg area
var node = svg.append("g")
    .selectAll("circle")
    .data(data)
    .join("circle")
    .attr("class", "node")
    .attr("r", d => size(d.value))
    .attr("cx", width / 2)
    .attr("cy", height / 2)
    .style("fill", d => color(d[variable]))
    .style("fill-opacity", 0.8)
    .attr("stroke", "black")
    .style("stroke-width", 1)
    .on("mouseover", mouseover) // What to do when hovered
    .on("mousemove", mousemove)
    .on("mouseleave", mouseleave)
    .call(d3.drag() // call specific function when circle is dragged
        .on("start", dragstarted)
        .on("drag", dragged)
        .on("end", dragended));
```

Fig 30. Initializing nodes

After I have created the nodes, I create a force simulation which attracts all the nodes a specific centre when they are all first initalised. Once that has happened tho, the circles on the screen can be dragged around.

```
// Features of the forces applied to the nodes:
const simulation = d3.forceSimulation()
    .force("center", d3.forceCenter().x(width / 2).y(height / 2)) // Attraction to the center of the svg area
    .force("charge", d3.forceManyBody().strength(.1)) // Nodes are attracted one each other of value is > 0
    .force("collide", d3.forceCollide().strength(.2).radius(function(d){ return (size(d.value)+3) }).iterations(1)) // Force that avoids circle overlapping

// Apply these forces to the nodes and update their positions.
// Once the force algorithm is happy with positions ('alpha' value is low enough), simulations will stop.
simulation
    .nodes(data)
    .on("tick", function(d){
        node
            .attr("cx", d => d.x)
            .attr("cy", d => d.y)
    });

// What happens when a circle is dragged?
function dragstarted(event, d) {
    if (!event.active) simulation.alphaTarget(.03).restart();
    d.fx = d.x;
    d.fy = d.y;
}
function dragged(event, d) {
    d.fx = event.x;
    d.fy = event.y;
}
function dragended(event, d) {
    if (!event.active) simulation.alphaTarget(.03);
    d.fx = null;
    d.fy = null;
}
```

Fig 31. Dragged and Force simulation implementation

Map

Before loading the map in, I created different colours and created a function which would help me project and generate the map path that I wanted.

```
var svg = d3.select("#content g.map")
let projection = d3.geoMercator().scale(400).translate([200, 280]).center([0, 5]);
let geoGenerator = d3.geoPath().projection(projection);
const radiusScale = d3.scaleSqrt();

// colour schemes to use
var color = d3.scaleOrdinal().range(d3.schemeSet2);
var pop_clr = d3.scaleSequential().domain([1, 100000000]).interpolator(d3.interpolateRdBu);
var gdp_clr = d3.scaleSequential().domain([1, 15000]).interpolator(d3.interpolateRdBu);
var content_clr = d3.scaleSequential().domain([1, 200]).interpolator(d3.interpolateRdBu);
var standup_clr = d3.scaleSequential().domain([1, 10]).interpolator(d3.interpolateRdBu);
```

Fig 32. Colours and Map setting

I had also created a mouse over function which when hovering upon a country would give the information I wanted to provide. Also it would highlight country being hovered upon in red for further clarification about the country which is being hovered upon.

```
// Mouseover function
function handleMouseover(e, d) {
    d3.select('#content .info').text("Country: " + d.properties.name)
        .style('fontsize', '56px')
        .style('color', 'white');

    d3.select('#content .economy')
        .text("Economic Status: " + d.properties['economy'])
        .style('color', 'white');

    d3.select('#content .pop')
        .text("Estimated population of the region: " + d.properties['pop_est'])
        .style('color', 'white');

    d3.select('#content .content')
        .text("Amount of content: " + d.properties['content_count'])
        .style('color', 'white');

    d3.select('#content .standup')
        .text("Amount of standup content: " + d.properties['standup_count'])
        .style('color', 'white');


    d3.select(this)
    .style('fill', 'red');
}
```

Fig 33. Mouseover functions

Once I set up what I needed, I wanted to load in the data for several datasets, including the geojson dataset I acquired to draw the map but also contained information about each country as well. I manipulated the data according to which I was able to make features from the other datasets part of the feature set present in geojson files. I had also created a projection feature which would allow me to create different sized dots based on the information that I wanted to display.

```
//load data
Promise.all([
    d3.csv('countries.csv'),
    d3.json('custom.geo.json'),
    d3.csv('standup_by_country.csv'),
]).then(([csv, json, standupcsv]) => {
    //for the csv
    const rowbyLocation = {};
    csv.forEach(d => {
        d.content_count = parseInt(d.count);
        rowbyLocation[d.country] = d;
    })
    console.log(rowbyLocation)

    const srowbyLocation = {};
    standupcsv.forEach(d => {
        d.standup_count = parseInt(d.count);
        srowbyLocation[d.country] = d;
    })

    countries = json

    // //we are joining the data from the csv into countries
    countries.features.forEach(d => {
        Object.assign(d.properties, rowbyLocation[d.properties.brk_name]);
    });

    countries.features.forEach(d=> {
        Object.assign(d.properties, srowbyLocation[d.properties.brk_name]);
    })

    console.log(countries.features)

    countries.features = countries.features.map(d => {
        d.properties['projection'] = projection(d3.geoCentroid(d));
        return d;
    });

    console.log(countries.features)

    update('content_count');
});
```

Fig 34. Loading files in

I created an update function which would allow me to draw several different maps based on the information I desired to present. To the get colours of the countries correct, I used if statements to further differentiate what sort of colour configuration was needed to make the map. I also created a feature where it would allow me to create the custom sized dots that would represent the data that I needed to represent.

```
function update(property) {
    console.log(property)
    var clr = color;
    var strokeclr = 'white';

    if (property == 'economy') {
        var clr = color;
        strokeclr = 'black';
    } else if (property == 'standup_count') {
        var clr = standup_clr;
    } else if (property == 'pop_est') {
        var clr = pop_clr;
    } else if (property == 'content_count') {
        var clr = content_clr;
    }

    console.log(strokeclr)

    const radiusVal = d => d.properties[property];
    const radiusScale = d3.scaleSqrt()
        .domain(d3.extent(countries.features, radiusVal))
        .range([0, 20]);

    countries.features.forEach(d => {
        d.properties['radius'] = radiusScale(radiusVal(d));
    });
```

Fig 35. Configurations for the dots and colours for the map

I then drew the map and circles on the map with the data that I had now completely acquired.

```
var map = svg.selectAll('path').data(countries.features)

map
    .join(
        enter => {enter.append('path')
            .attr('class', 'country')
            .attr('d', geoGenerator)
            .style('fill', function(d) {
                console.log(d.properties[property])
                return clr(d.properties[property]);
            })
            .style("stroke", strokeclr)
            .on('mouseover', handleMouseover)
            .on('mouseout', function(d) {
                d3.select(this)
                    .style('fill', function(d) {
                        return clr(d.properties[property]);
                    });
            })
            .on('click', clickedFunction)
            .append('title')
            .text(d => d.properties.name);

            enter.selectAll('circle').data(countries.features)
                .enter()
                .append('circle')
                .style('fill', function(d) {return 'white';})
                .attr('class', 'country-circle')
                .attr('cx', function(d) {
                    return d.properties.projection[0];
                })
                .attr('cy', function(d) {
                    return d.properties.projection[1];
                })
                .attr('r', d => d.properties['radius']);
        },
```

```
    update => {
        update.attr('class', 'country')
        .attr('d', geoGenerator)
        .style('fill', function(d) {
            console.log(d.properties[property])
            return clr(d.properties[property]);
        })
        .on('mouseover', handleMouseover)
        .on('mouseout', function(d) {
            d3.select(this)
                .style('fill', function(d) {
                    return clr(d.properties[property]);
                });
        })
        .on('click', clickedFunction)
        .append('title')
        .text(d => d.properties.name);

        update.selectAll('circle').data(countries.features)
            .enter()
            .append('circle')
            .attr('class', 'country-circle')
            .style('fill', function(d) {return color(d.properties[property]);})
            .attr('cx', function(d) {
                // console.log("enter running.....")
                return d.properties.projection[0];
            })
            .attr('cy', function(d) {
                return d.properties.projection[1];
            })
            .attr('r',10);
    },
    exit => exit.remove()
)
```

Fig 36. Drawing the map

To zoom in and out of the map, I made a zoom feature which would allow me to perform that action.

```
var zoom = d3.zoom()
    .scaleExtent([-10, 20])
    .on('zoom', function (event) {
        svg.selectAll('path')
            .attr('transform', event.transform);
        svg.selectAll('circle')
            .attr('transform', event.transform);
    });
svg.call(zoom);
```

Fig 37. Zoom in and out