Lab Directions

Complete Homework 6 - D3 Movies Line Demo**:**

1. Accept the GitHub assignment here: https://classroom.github.com/a/m5E8MEHP
2. Watch **0:12:46-1:01:03** of this recording
3. Follow my demo to the best of your ability
4. If you get stuck:
    a. Ask a friend for help
    b. Ask Jasper for help
    c. Write a comment in your Canvas submission about where you got stuck and how you tried to solve it
5. Submit PDF to Canvas (link)

Submitting to Canvas

Submit a **PDF** to Canvas containing the following:

- A **link** to your github repo
  (contains code)
- A **link** to your github page
  (contains viz)
- A **screenshot** of your viz
  (fail-safe if above links don't work)

Upcoming...

Homework:

- Read:
  - 2. Strengthen Your Spreadsheet Skills | Hands-On Data Visualization
  - 3. Find and Question Your Data | Hands-On Data Visualization
- Finish Homework 6 - D3 Movies Line Demo

Tomorrow: Make rough draft of Dynamic Trends dash w/Tableau or Sheets

Next week: Start coding Dynamic Trends

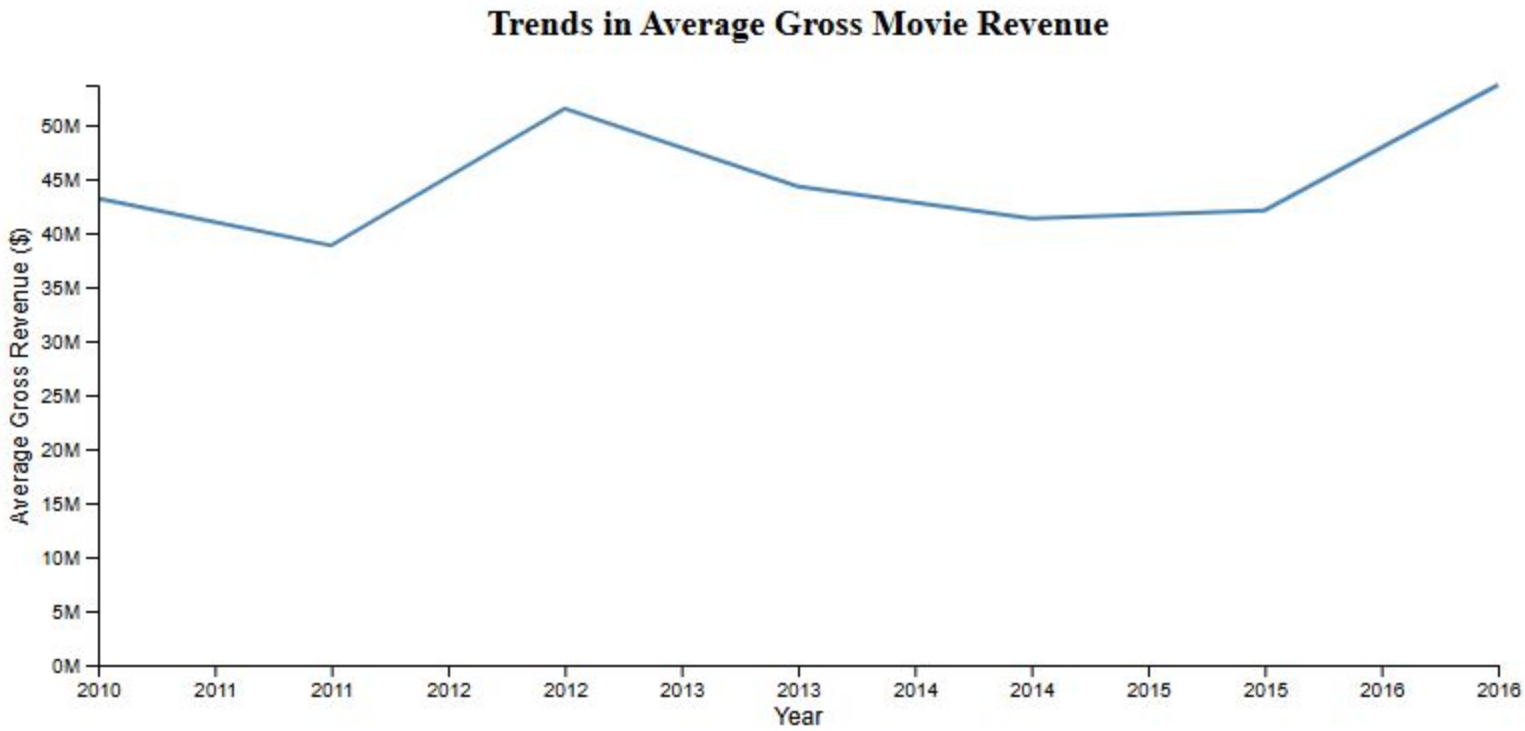
D3 Movies Line Graph

Lecture 6.2 (2/11)

High-level steps

1. Create blank SVG
2. Load and reformat data
3. Prepare data
4. Set axis scales
5. Plot data
6. Add axes
7. Add title and labels
8. Push changes and publish

Line Graph



Trends in Total Gross Revenue

Learning Objectives

- Making a line graph with **1 numeric** and **1 date** field
- Filtering out null and unwanted values
- Aggregating using .rollup()
- Sorting and using datum()

Set-up

1. Click here to get GitHub assignment
2. Git clone to your computer
3. Open the repo in VS Code
4. Create a new terminal in VS Code
5. Make sure you're in the right folder (use cd to change directory)
6. Start python server: **`python -m http.server 8080`**
7. Copy-paste this site in search bar: **`http://localhost:8080/`**

Familiarize yourself with your project files

1. **colleges.csv** - your data
2. **index.html** - web page structure code
3. **main.js** - data viz code; where we'll be working primarily
4. **style.css** - style code

gross	title_year
59774	2010
95001343	2013
127437	2011
	2013
	2015
71897215	2016
14616	2015
56667870	2013
18329466	2010
52822418	2016

Examine data: movies.csv

```
JS main.js > ...
 8  // Create the SVG container for the bar chart
 9  const svgBar = d3.select("#barChart")
10      .append("g")
11      .attr("transform", `translate(${margin.left},${margin.top})`);
12
13  // Create the SVG container for the line chart
14  const svgLine = d3.select("#lineChart")
15  .append("g")
16  .attr("transform", `translate(${margin.left},${margin.top})`);
```

Note: Don't worry too much about understanding this code for now; I will provide it as starter code for DC2.

```
index.html > html > body
 2  <html lang="en">
10  <body>
12      <svg id="barChart" width="800" height="350"></svg> <!-- Bar chart for directors -->
13      <svg id="lineChart" width="800" height="350"></svg> <!-- Line chart for IMDb scores -->
14      <script src="main.js"></script>
15  </body>
16  </html>
```

Step 1: Create blank "canvas" in main.js (referenced in index.html)

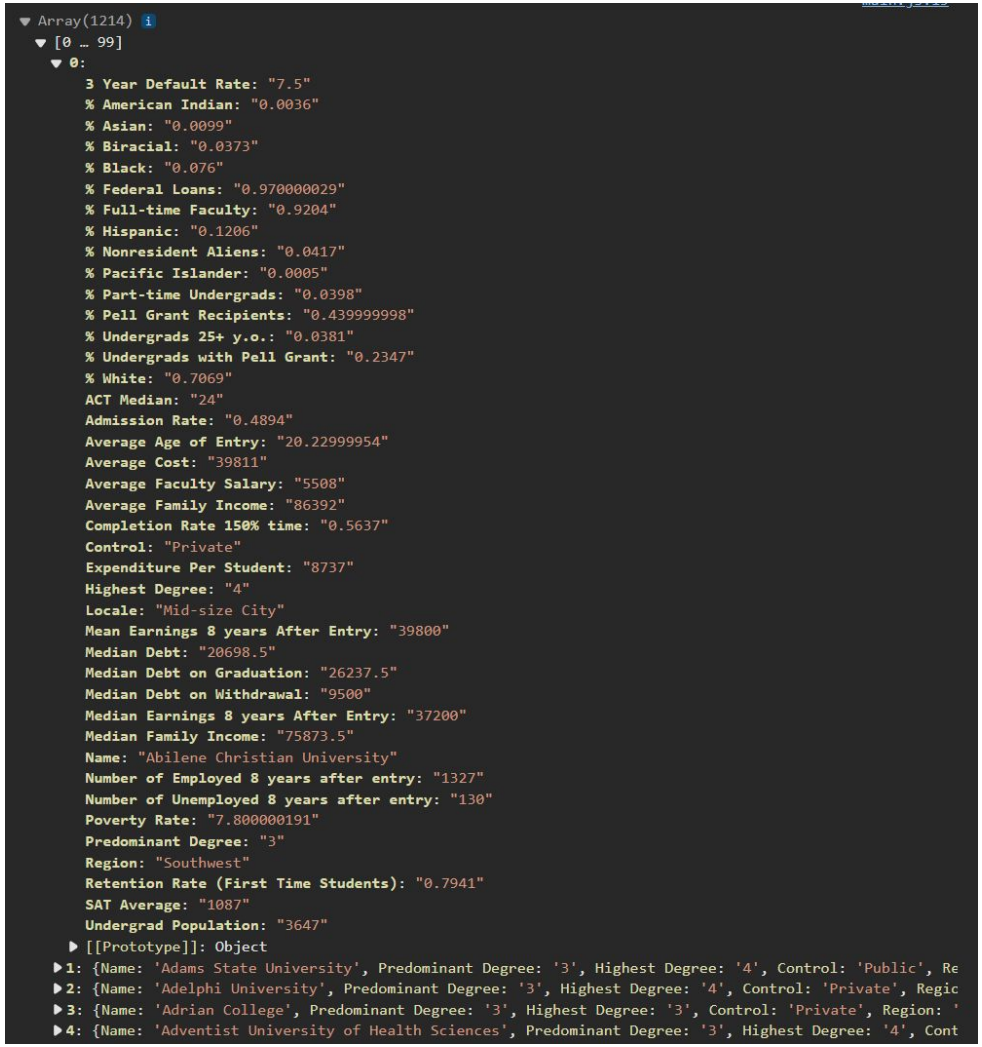
Step 2: Load...

Full function syntax:

```
d3.csv("fileName.csv").then(function(data) {
      console.log(data) // Check data
      data.forEach(function(d) {
            // Reformatting code…
            return ...;
      })
      // Data viz code…
});
```

Concise function syntax:

```
d3.csv("fileName.csv").then(data => {
      console.log(data) // Check data
      data.forEach(d => {
            // Reformatting code…
      })
      // Data viz code…
});
```



Step 2: ...and format

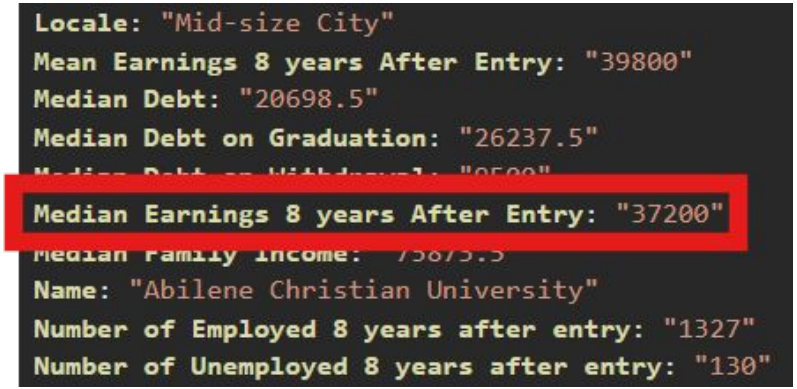
For each point of data...

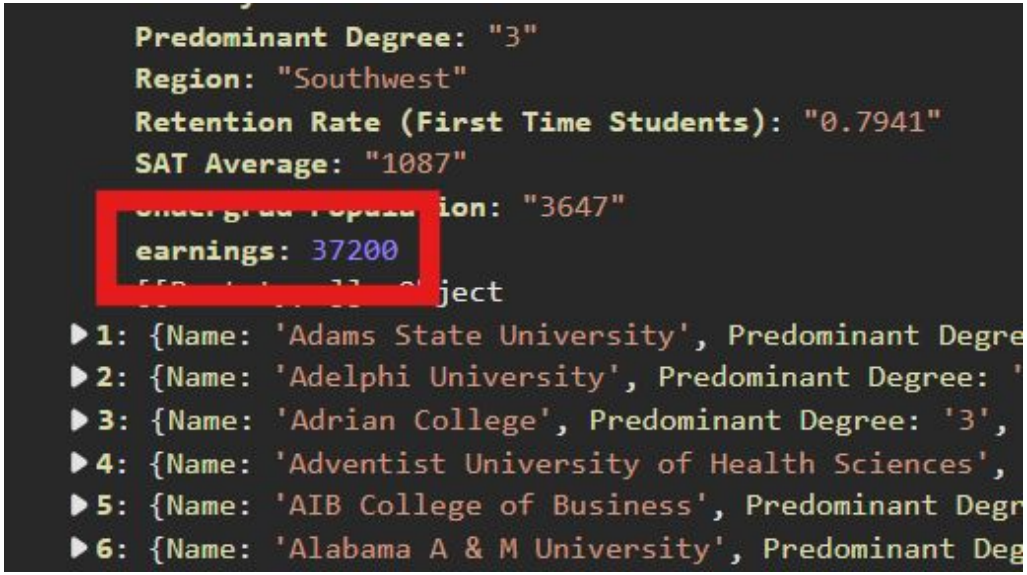
1. **Rename** the column (ideally with NO spaces)
2. **Convert** value from string to number (add `+`)
3. (Optional) **Test** data type using typeof() function:

```
console.log(
    typeof(d[“new_column_name”])
)
```

Format:

```
data.forEach(d => {
    d["new_column_name"] = +d["Old Column Name"];
})
```





Before vs. after formatting

Step 3: Prepare data

We need to get our data from this (raw)...

gross	title_year
59774	2010
95001343	2013
127437	2011
	2013
	2015
71897215	2016
14616	2015
56667870	2013
18329466	2010
52822418	2016

To this (grouped, sorted, filtered)...

Row Labels	Sum of gross
2010	9931756876
2011	8745094799
2012	11380108510
2013	10501612904
2014	10422826870
2015	9513928206
2016	5692378656

3.a: Remove nulls and “box” range

Syntax:

```
data.filter(d => d.column1 != null
        && d.column2 != null
        && d.column3 >= some_#
        ...
);
```

Things to do:

- Remove null from gross and year
- “Box” year’s range to be >= 2010 (when most of the data begins)

3.b: group and aggregate with rollup()

- What are you grouping by? (For each **[some category var]**...)
  - For each **YEAR**
- How are you aggregating this group? (...get the [some aggregation] **[some number var]**...)
  - Get the SUM OF **GROSS**

Similar to the group_by() -> summarize() step in R.

3.b: Grouping and aggregating with rollup()

- **rollup()** returns a **map** where...
  - The keys are group identifiers (like **YEAR**)
  - The values are the aggregated results (like SUM OF **GROSS**)
- Example:
  - [{2010: 990000000000 },
    { 2011: 870000000000 },
    { 2012: 11300000000000]

3.b: Grouping and aggregating with rollup()

Full syntax:

```
const yourMapName = d3.rollup(data,
    function(v) {
        return d3.yourAggreg(v, function(d) {
            return d.yourNumVar;
        });
    },
    function(d) {
        return d.yourCategoryVar;
    }
);
```

Concise syntax:

```
const yourMapName = d3.rollup(data,
    v => d3.yourAggreg(v, d => d.yourNumVar),
    d => d.yourCategoryVar
);
```

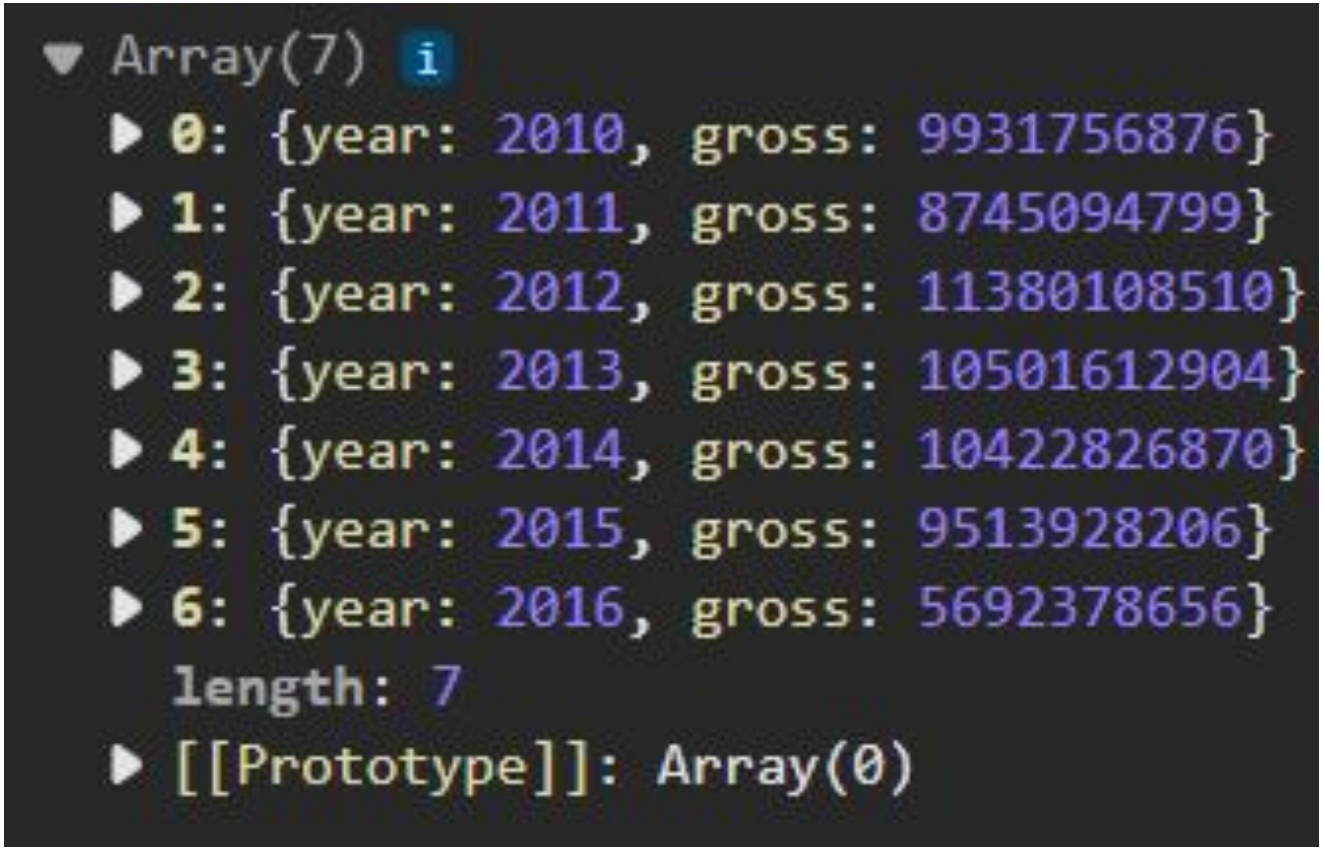
3.c: Converting to array and sorting

1. Turn **map** into **array** using **Array.from()**

```
const yourFinalArray = Array.from(yourMapName,
    ([category, num]) => ({ category, num })
)
```

1. **Sort** array using **sort()** (**a - b for ascending**, b - a for descending)

```
    .sort((a, b) => a.x_axis_var - b.x_axis_var)
```



Step 3: Final result (use `console.log(`**`yourFinalArray`**`)` to check)

Step 4: Scale & Determine Axes

Purpose: map data values to pixel values to position elems correctly based on data.

Methods:

- **scaleLinear()** - Creates a linear scale for axis
- **domain([min, max])** - Specifies range of **input data values**
- **range([min, max])** - Specifies range of **output pixel values**, where width/height = axis end
- **max(data, value)** - Finds the maximum VALUE in DATA

More on scales in D3 textbook Ch. 7.

Step 4: Scale & Determine Axes

Do this twice, once for x variable and once for y variable.

Format:

```
let xVarName = d3.scaleLinear()
    .domain([0*, d3.max(data, d => d.COLUMN_NAME)])
    .range([0, width]); // START low, INCREASE
```

*In this case, we want domain's min to be 2010 since that's our lowest year

```
// Think backwards for y!
let yVarName = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.COLUMN_NAME)])
    .range([height,0]); // START high, DECREASE
```

Step 4: Create line generator (line graphs only)

Syntax:

```
const line = d3.line()
    .x(d => yourXScale(d.x_column))
    .y(d => yourYScale(d.y_column));
```

Creates the “path” between points on your graph.

Step 5: Plot data with datum()

Instead of selectAll > data > enter > append workflow, simply:

```
yourSvg.append("shape")
    .datum(yourFinalArray)
```

Use datum() when array of data represents ONE element on graph (line graph).

Step 5: plot data with **path** shape

For line charts, use the **“path”** shape, which has the following attributes:

1. **d** - the line connecting points (should be based on line generator you made)

```
.attr("d", yourLine)
```

1. **stroke** - line color

```
.attr("stroke", “some color”)
```

1. **stroke-width** - line width

```
.attr(“stroke-width”, #)
```

1. **fill** - color between line gaps (set this to “none” in most cases)

```
.attr(“fill”, “none”)
```

4: Plotted line

Step 6: Add Axes

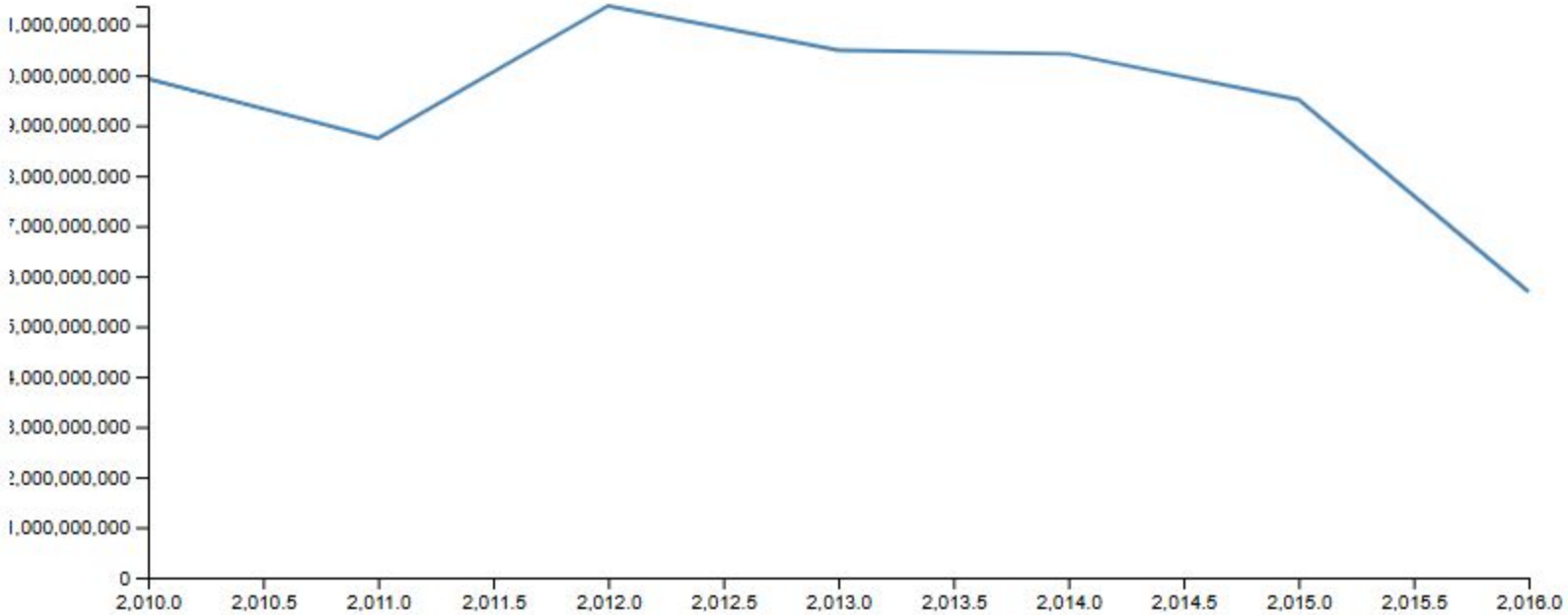
X-axis syntax:

```
svgName.append("g")
    .attr("transform", `translate(0,${height})`)
    .call(d3.axisBottom(yourXScale));
```

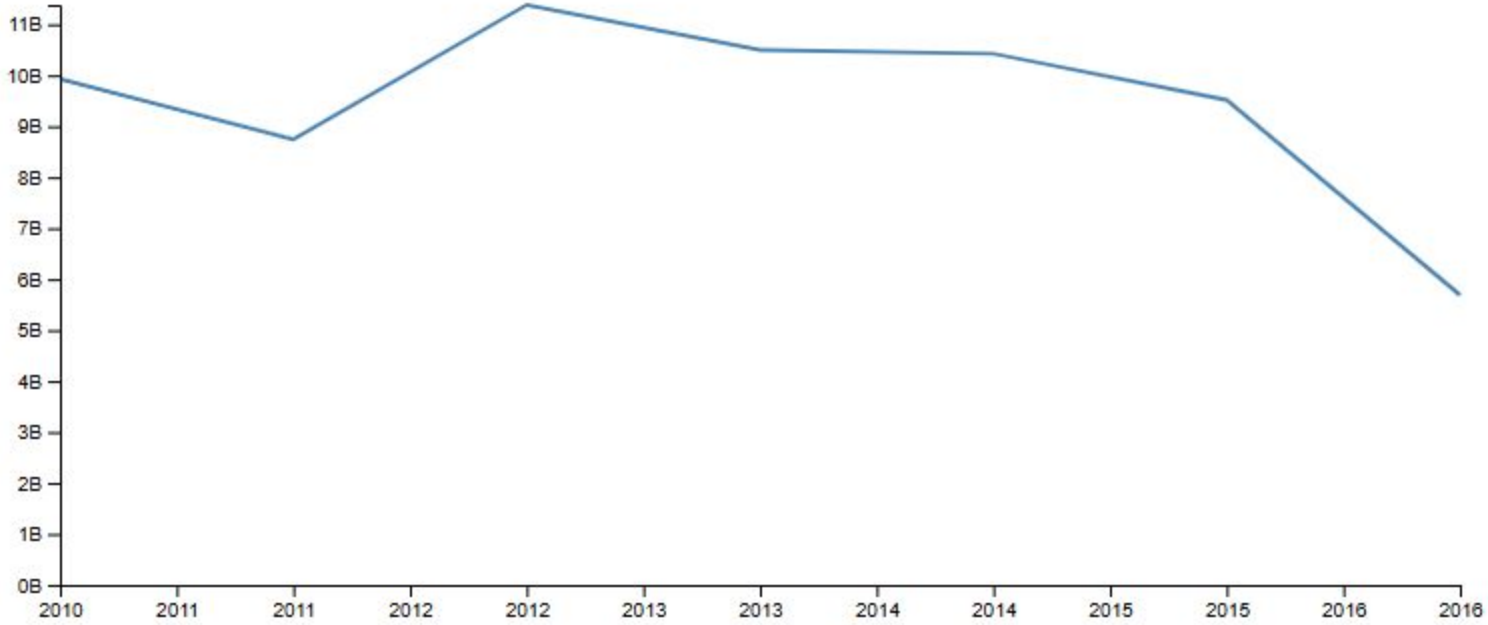
Y-axis syntax:

```
svgName.append("g")
    .call(d3.axisLeft(yourYScale));
```

<g> element: used to group SVG shapes and allows for easier transformations and styling



Step 5: Add axes, unformatted



5: Add axes, formatted

Step 7: Add labels

Steps:

1. Append text element to SVG
2. Add class defined in style.css
3. Adjust position (x and y attributes)
4. Set text

Syntax:

```
svgName.append("text")
    .attr("class", "CLASS NAME")
    .attr("x", #)
    .attr("y", #)
    .text("YOUR TEXT");
```

7.a: title

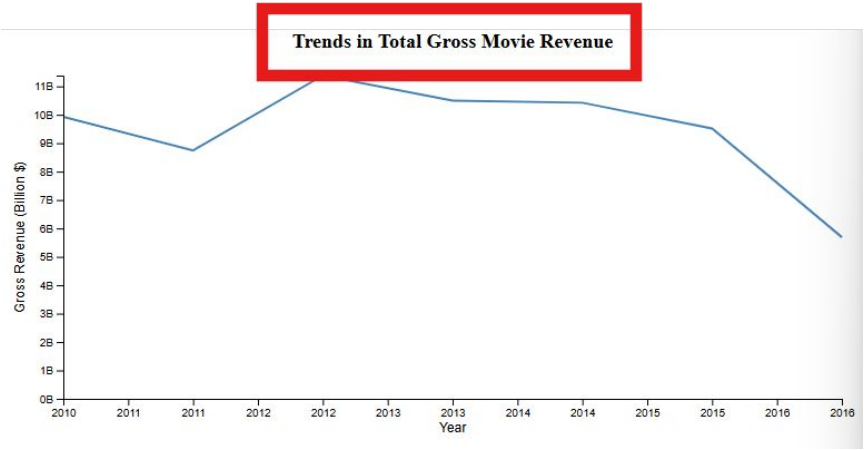
Class: `"title"`

X position:

- `width / 2`
- Centered horizontally

Y position:

- `-margin.top / 2`
- Centered vertically in top margin



7.b: x-axis (year)

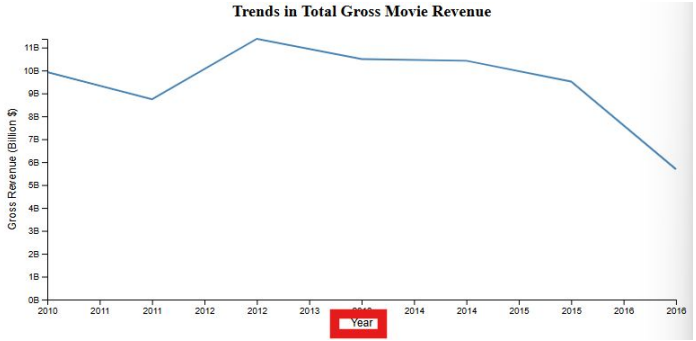
Class: `"axis-label"`

X position:

- `width / 2`
- Centered horizontally

Y position:

- `height + (margin.bottom / 2)`
- Centered vertically in bottom margin



Step 7.c: y-axis (gross) – THINK BACKWARDS

Class: `"axis-label"`

Transform:

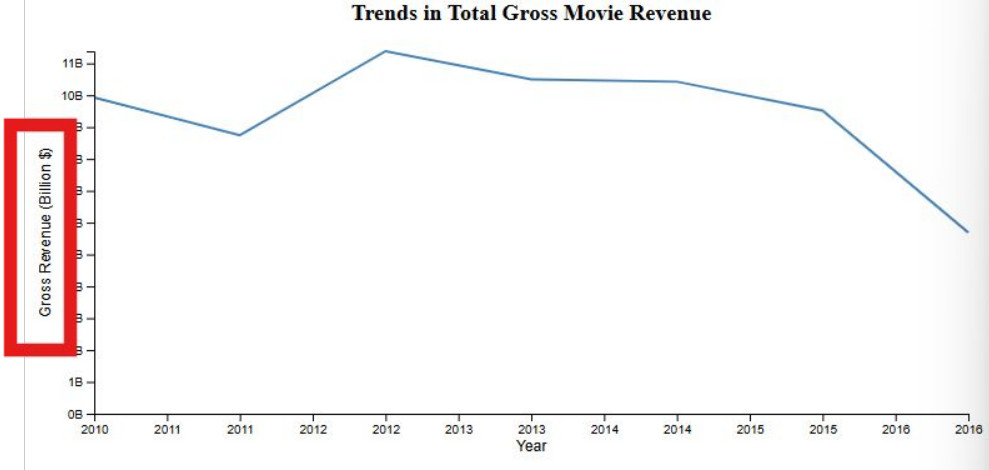
- `rotate(-90)`
- Rotated 90 degrees counterclockwise

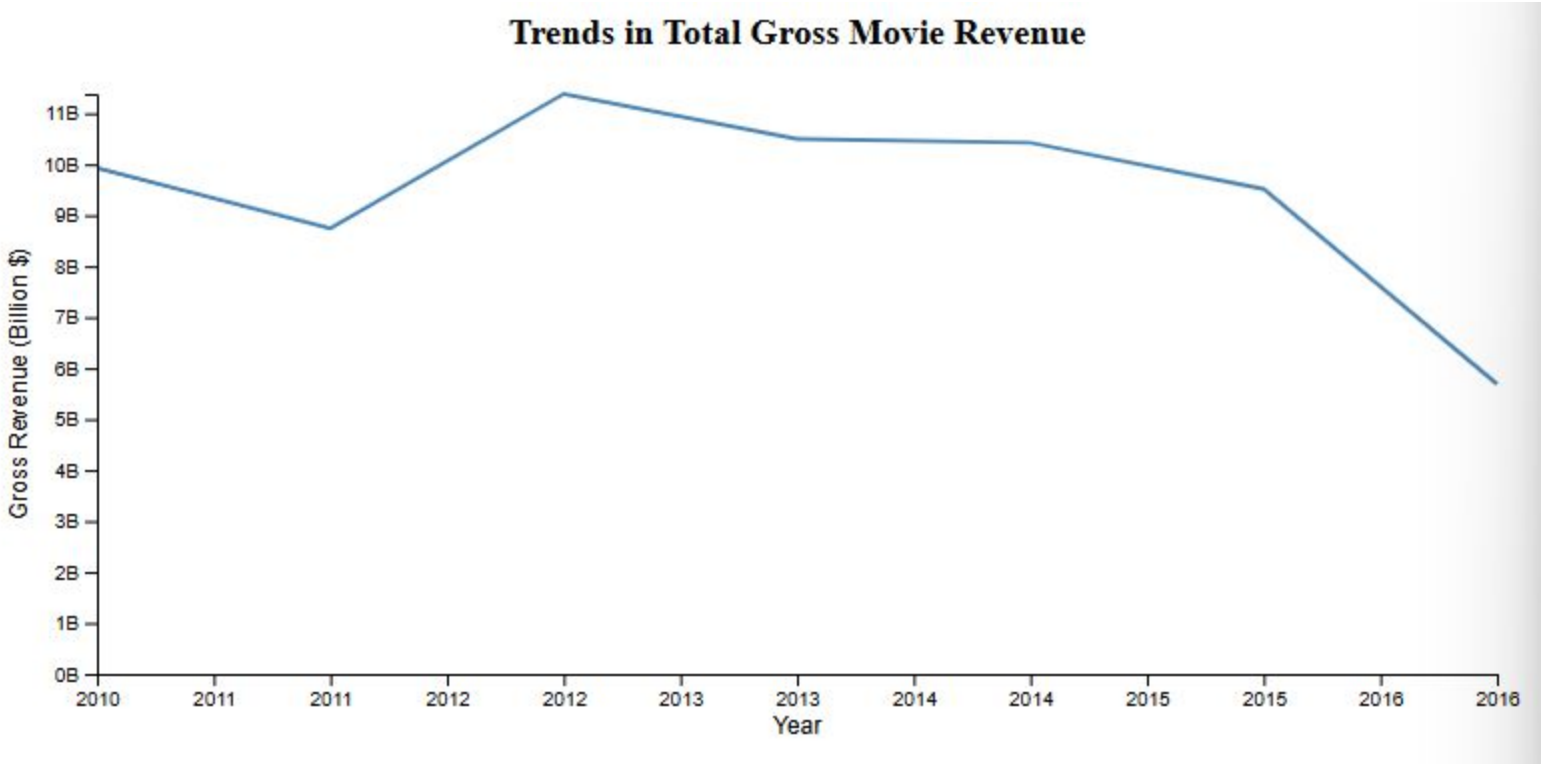
Y position:

- `-margin.left / 2`
- Centered in left margin

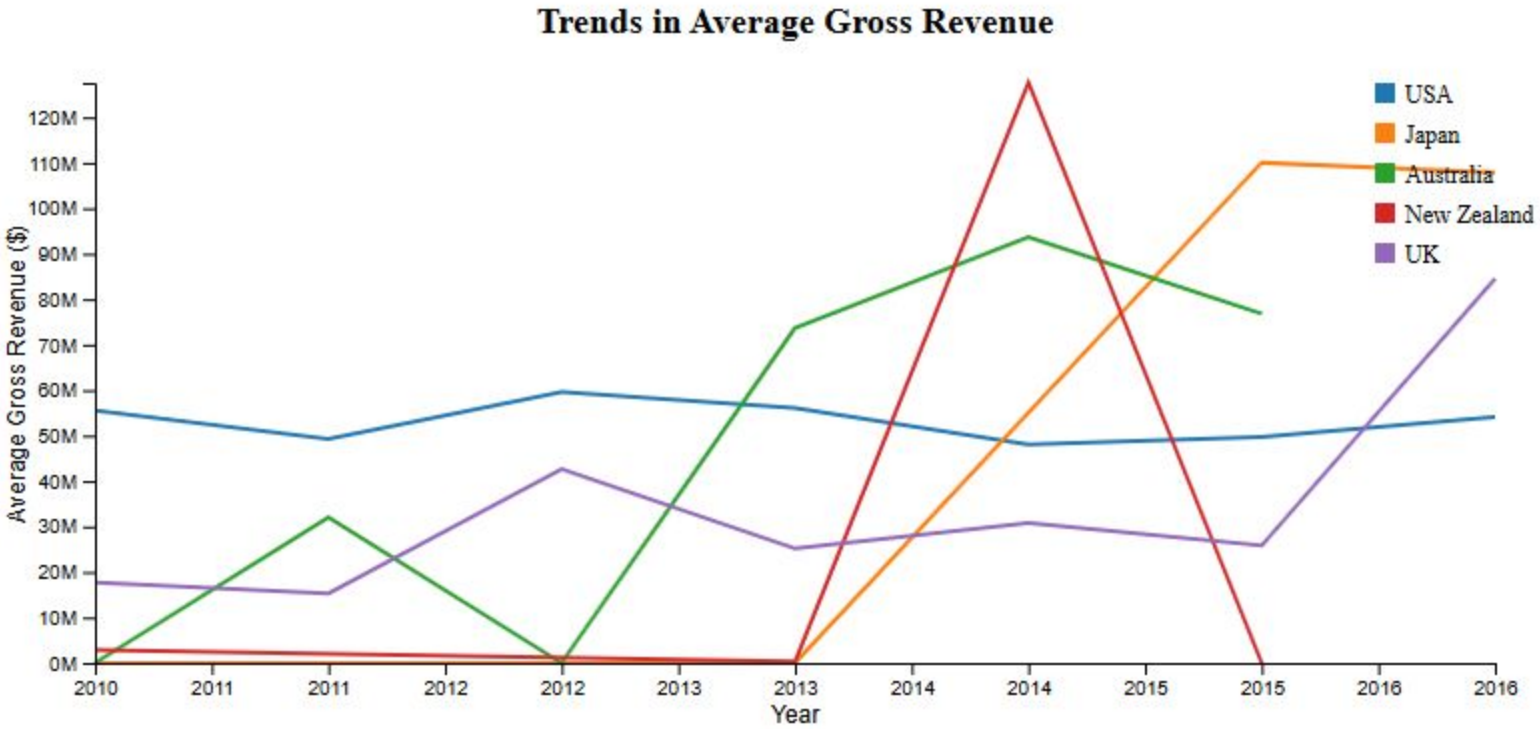
X position:

- `-height / 2`
- Centered horizontally





Final product



Multi-line graph (we'll learn later)

Final step: push changes & publish

1. Git add, commit, and push to GH
2. “Settings” > “Pages” > set branch to “main” > “Save”
3. Go back to main repo page (“Code” tab)
4. Scroll down to deployments in the bottom right (may take a few min to deploy)
5. “github-pages” > click github.io link