**Data Cleaning**

The data that we received initially was not formatted in a way that worked well with our visualization. The data had groups, sub-groups, and the individual language rows with the totals and sub-totals as their own row. This would lead to problems with aggregating the data (since it would duplicate number values) and so we took the totals and sub-total rows and created new columns for them. We did the main portion of the data manipulation manually in excel since we decided it would take too long to complete all of it using a programming language and we were eager to start using d3 on the final dataset. We validated the new totals against the old sub-totals and totals to make sure we were not duplicating or deleting values and to help reduce the human-error in manipulating the data. There were also some letters being used for null values, and so we replaced all of the letters with NA’s so we had a common way to filter out null values in the data.

**Styling**

Our aim for the styling was to make the interface clean and user-friendly. We also wanted to incorporate continuity among all the views by using the same color scheme for the same categories, legends, similar tooltip styling and fonts for all views. We aimed for a clean feel without a cluttered look and interactivity on all of the views.

**Scrolling**

Figuring out how to create a scrolling visualization seemed complicated at first, but became easier to understand once I got through reading the articles and going through the code myself. The initial inspiration for building a scroller came from this medium post: <https://towardsdatascience.com/how-i-created-an-interactive-scrolling-visualisation-with-d3-js-and-how-you-can-too-e116372e2c73>. The visualization that was created can be found here: <https://cuthchow.github.io/college-majors-visualisation/>. We read the medium article first and then consulted Cuthbert’s reference for building a scroller: <https://vallandingham.me/scroller.html>.

The basic architecture used the same svg for all of the five different visualizations. Essentially they were all rendered on top of each other. The opacity for each is adjusted based on where the user is in the scrolling. There were several difficulties with rendering tooltips when five different visualizations are on top of each other. Several techniques we incorporated included the d3.raise() function to move the current visualization to be rendered last in the svg and to turn event listeners on and off for each visualization based on where the user was currently scrolling.

The challenges with scrolling was difficult to get right initially since we were trying to coordinate five different views. An initial struggle was getting simulations to start in the middle of the page instead of being created in the top left and then floating to the middle. Another struggle was getting the visualizations to disappear by changing the opacity, so we would have errors like the following:

A picture containing map

Description automatically generated

Figure ? Example of scrolling errors

**Transitions**

Originally, we wanted transitions between all our visualizations to create a sense of continuity and a creative aesthetic. We accomplished this for two of the view changes. The first view change from the cluster to the map shows the clustered circles coming from the first view into the top of the map view. These same circles are used in the interactivity in the map view. The second transition involved the change from the horizontal bar graph to the vertical bar view. We created a custom transition by moving the rectangles/bars from the horizontal view and showing them consolidating into the vertical bars of the same color. We used timed transitions and d3 rendering functions to create this transition.

We struggled to find transitions between the remaining views since the elements were represented in different forms among these views (i.e. circles vs rectangles and rectangles vs. areas). With more time we would have tried to create innovative transitions between these views.

**Bubble Chart View**

To build the clustered bubble chart view, we had to learn about how to use force simulations. The first step was getting the circles to show up at all. We had always passed ‘cx’ and ‘cy’ values explicitly and never derived them from a simulation, so it was a new technique for us. We learned that in a simulation, if the nodes positions aren’t explicitly coded, then they we automatically be created. This was what the first example of creating the cluster looked like:

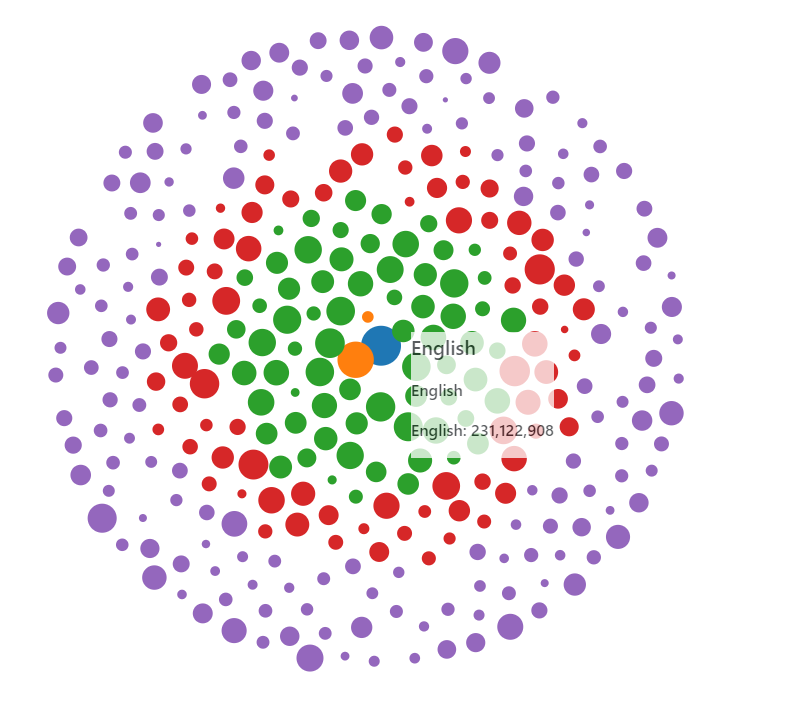


Figure 1: The default phyllotaxis arrangement

The guide that we used to help make this first visual was a combination of the tutorial for the course (<http://dataviscourse.net/tutorials/lectures/lecture-d3-layouts/>) and the d3 documentation on force layouts (<https://github.com/d3/d3-force/blob/master/README.md>). The problem with this first view was the circles overlapped. In addition to removing overlap, we wanted to cluster the circles instead of having them in rings. So we first got the collision detection working using this blocks page as reference: <https://bl.ocks.org/d3indepth/9d9f03a0016bc9df0f13b0d52978c02f>. See the next figure for a picture of what it looked like.

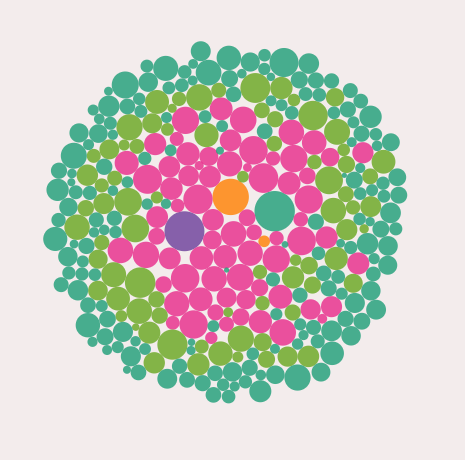


Figure 2: Collision detection was now working

Lastly, we wanted to get the clusters working. Before we got the clusters working, we changed from a log scale to an exponential scale since it was easier to control the sizing between the nodes. Linear scaling didn’t work since the smallest nodes were too small compared to the largest nodes, which is why we eventually decided to do exponential sizing as a happy medium between linear and log scales.

Part of getting the clusters working included using the force testing ground found here to fine-tune the simulation: <https://bl.ocks.org/steveharoz/8c3e2524079a8c440df60c1ab72b5d03>. The reference we used for clustering was found here: <https://bl.ocks.org/pbogden/854425acb57b4e5a4fdf4242c068a127>. Once we realized that we needed to have centroids that the circles clustered around and understood the code more clearly, we created this view:

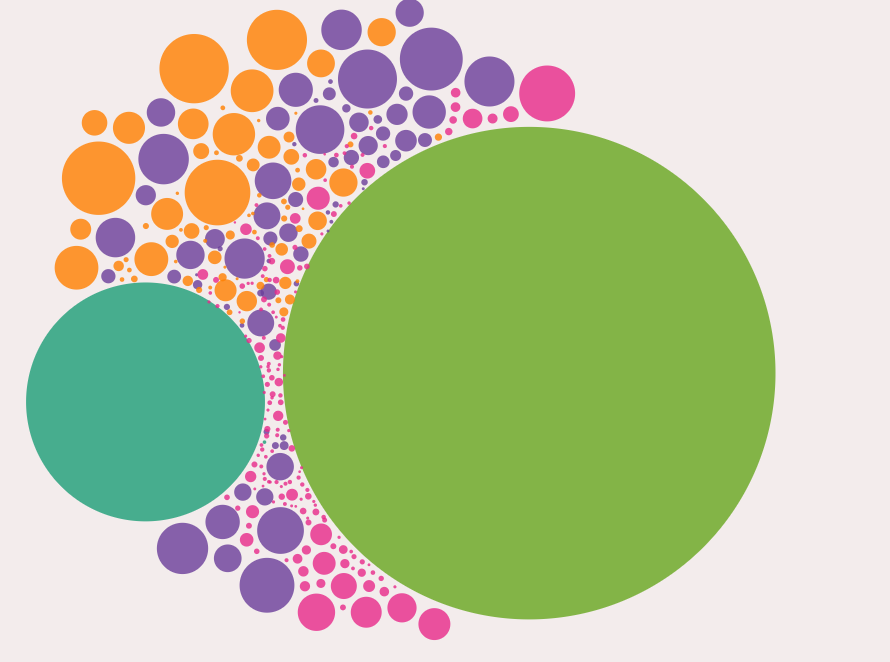


Figure 3: Initial clustering attempt

Finally, after fine-tuning some more, we ended up with the final result for our first visualization in our scrolling visualization.

Eventually we were able to control the opacity of the visualizations. The next challenge was sharing one tooltip div between the visualizations. The issue was the order that the SVG’s were drawn was causing the tooltip to only work for the first visual and not the rest. In order to fix this, we learned to use the d3.raise() function.

**Map View**

RACHEL’S STUFF GOES HERE….

**Horizontal Bar Graph View**

The horizontal bar graph was designed to show the breakout of foreign speakers for each state. The horizontal bars were meant to give the user a relatively accessible comparison of the foreign language diversity in each state. This visualization used the datasets for all of the states. The languages for each state were summed by category using the d3 rollup methods. The categories are presented as the different colored bars in the graph. The first implementation was done in an html table layout, with the state names as one column and an svg element containing the bars/rects as the other column. Here is the first version of our bar chart:

Background pattern

Description automatically generated

Figure ?: Initial view of the horizontal bar graph

The next functionality we implemented was clicking the first bar and sorting by that bar. The same was done with the names of the states. The sorting was done by sorting the data bound to the either the name column or the first rect in the svg.

Background pattern

Description automatically generated

Figure ? First sort implementation of the horizontal bar graph

Sorting the first bar (i.e. the yellow category) was easy. Sorting by the other categories was more difficult since we wanted the sorted bar to appear first. By showing the sorted bar first, the user could make an easier comparison. Otherwise, the sorting didn’t appear as intuitive or obvious. Incorporating this feature meant that the x attribute for each rectangle would change with each sort. We approached this the hard way at first before realizing that the d3 library incorporated a stack layout that could have made this process much easier! However, we only realized this once the implementation was complete. To compute the x attribute for each sort, we created an array that contained each possible x value depending upon the order of the bars, which in turn was determined by the group being sorted. When the sorting was finished, a tooltip was added that displays the information for each bar when the user hovers over that bar.

Chart, diagram, bar chart

Description automatically generated

Figure ? : Sorting implementation completed and tooltip added to horizontal bar graph

An added complication came when trying to incorporate this view into the overall visualization. Our visualization was designed so that every view used the same svg, eseentially rendering them on top of each other. Because of this, the table element that his graphic was originally rendered in did not work. Therefore, it had to be redone using all svg elements rather than the html table element.

The final version includes storytelling features on the left side in the scrolling section. We wanted to highlight some of the interesting facts about this data while also showcasing some of the interactive tools for the graph. When the user clicks the highlighted links/words in the storytelling section, the graph will highlight the information. We also included a legend so the different language categories could easily be determined, using the same color scheme and categories that were used in the previous views.

Graphical user interface

Description automatically generated

Figure ?: Final version of the horiztonal bar graph

**Vertical Bar Chart View**

The purpose of this visualization is to explore which foreign language speakers are the most fluent in English. This visualization used the national dataset, so the languages were totaled for all of the states. The first step in the implementation was deciding how to represent the data, since we wanted to show the data at all three levels: group, subgroup and language. A tree data structure seemed to be the best fit. We created a Tree Class to store our data. Our data tree had four levels: the root node, the group level, the subgroup level and the language level as the tree leaves. The Tree Class incorporated several recursive methods to traverse the tree/data that would facilitate rendering different views of our graph.

Our original plan was to incorporate a diverging bar chart, with percentages of bilingualism on one side and total number of speakers on the other side. However, due to special constraints we decided to use a vertical bar chart instead. We chose to have the total number of speakers represented by the height of the bar. The percentage of speakers who are bilingual would be represented as a saturated line on the bar, with the height of the line proportional to the percentage of bilingualism. Our first version is shown here, with the bars representing the top group level:

Chart, bar chart

Description automatically generated

Figure :? First version of the vertical bar graph

To navigate this graph, we wanted the user to click on a category. The category bar would then break out into subcategories, showing information for each subcategory. The user could continue down the tree by clicking again on a subcategory and that bar would break out into language data. Navigating the tree was one of the first decision choices. We wanted to make the navigation as simple and intuitive as possible. We decided upon having the user click once to drill down the tree and double clicking to go back up a level. If the user was at the bottom (hence the language level) and clicked nothing would happen. The same would be for the top (group level) if the user double clicked there. The first challenge encountered was that a double click also fired the click event. We needed to distinguish between the two events. We found an answer online at <http://bl.ocks.org/couchand/6394506> that we based our implementation on.

A picture containing chart

Description automatically generatedChart

Description automatically generated

Figure ?: Examples of scaling issues with vertical bar graph

A few views of the graph shown at subgroup and language level are shown above. As can be seen, one of the biggest challenges was the large difference in the scale of the data. Some languages have as little as 100 or less total speakers nationwide. On the other hand, Spanish has over 37 million speakers! Trying to capture this data in a meaningful way on the same graph was challenging. Some of the data was so small that it was meaningless in the graph. Ultimately, we decided that showing the total number of speakers for each language was distracting and complicating the visualization and also duplicating the same information presented in our first view. We decided to only show the percentages for each language within the bar graph. This eliminated the scaling problem and made the graph much easier to see.

Here are some views of the group, subgroup and language levels using only the percentages for the bars. The scaling looks much better! A tooltip was added for additional information when the user hovers over a bar.

Chart

Description automatically generatedChart

Description automatically generated

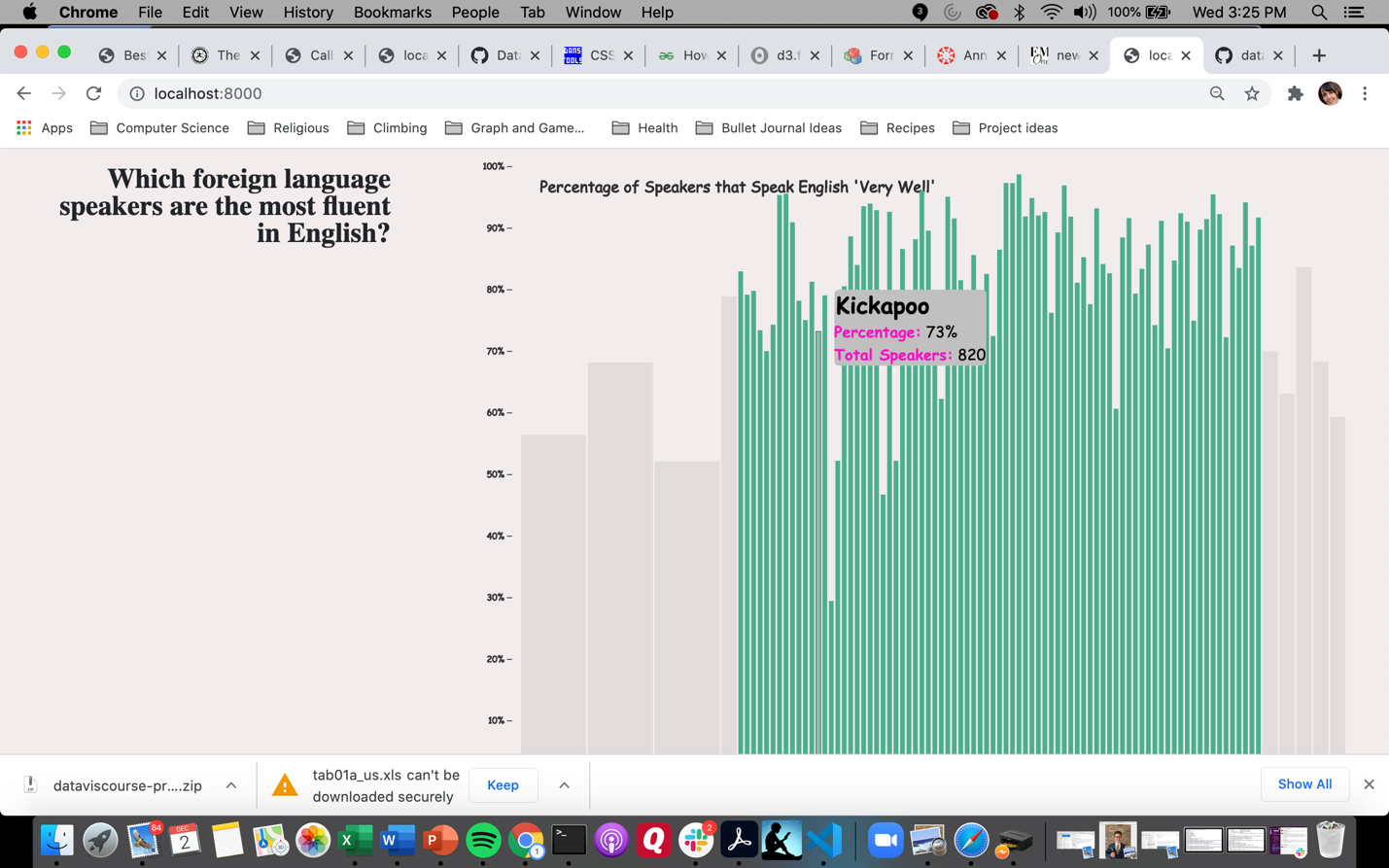


Figure ? : Examples of different levels selected within the vertical bar graph

The next functionality we wanted to incorporate was a sorting function, so that each group, subgroup and language could be compared to all of the other similar items. Below is a view of all the languages sorted.

A picture containing chart

Description automatically generated

Figure ?: Sorting function implemented into vertical bar graph

For the final version we added some storytelling links and a home button so the user could easily return to the original view. Again, we wanted to highlight some of the interesting facts about this data while also showcasing some of the interactive tools for the graph. Below is a final version of this graph.

Chart

Description automatically generated

Figure ?: Final version of the vertical bar graph

**Area Chart View**

This graph is meant to highlight the changes in the distribution of foreign languages within the U.S. over the past 40 years. We would have liked to incorporate a larger timeline into our graph, but only data from 1980 onwards was available. We considered using a line chart to dislay the data. We decided to implement an area chart because of its visual appeal and coloring and because we wanted the challenge of learning some new skills in d3!

This graph used a different dataset from the other graphs, although this dataset was fairly small and simple so there wasn’t much work to clean and prepare the data. To incorporate the area chart, we used the d3 stack layout feature and the d3 area function. The d3.stack() function calculated coordinates for all of the stacked areas. Then, these coordinates are used in the d3.area() function to creates svg paths to render the stacked areas. After spending some time reading the documentation and several tutorials, we had a working area chart.

Chart

Description automatically generated

Fig :? Numbers view of stacked area chart

The first view contained the total number of speakers for each year (1980 – 2010) in 10-year increments. We created a second view that showed the percentage of each particular group compared to the overall total foreign language speakers. This view can provide additional information as to how the overall distribution is changing for each group.

Graphical user interface

Description automatically generated

Figure :? Percentages view of stacked area chart

The next feature we added was a tooltip that shows the name of the language when the user hovers over an area and also greys out the other languages that are not highlighted.

Graphical user interface, application

Description automatically generated

Figure :? Highlighted area with tooltip in stacked area chart

Next we wanted the user to click an area and have only that area/language shown. To do this, the stacked coordinates had to be recalculated with only one stack. Then these new coordinates were again fed into the area generator to create the path for rendering. An additional feature with this view is a tooltip that shows information about each year as the user mouses close to that year. This involved creating a “snap” for the tooltip. A d3.bisect() function was used to calculate the snap for the mouse.

Graphical user interface

Description automatically generated

Fig ?: Single area shown with tooltip snap

To finish this view we added storytelling features and a category legend to the left of the view.



Figure :? Final view of the area chart