Lauren Courtney

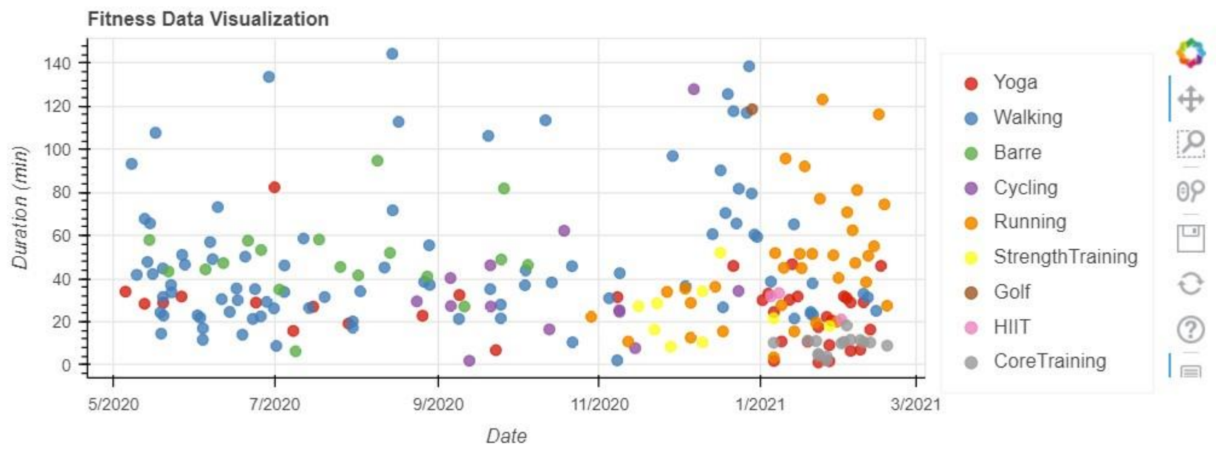
Data Visualization

March, 2021

Project 4 – Interactive Visualizations with JavaScript, R, or Python

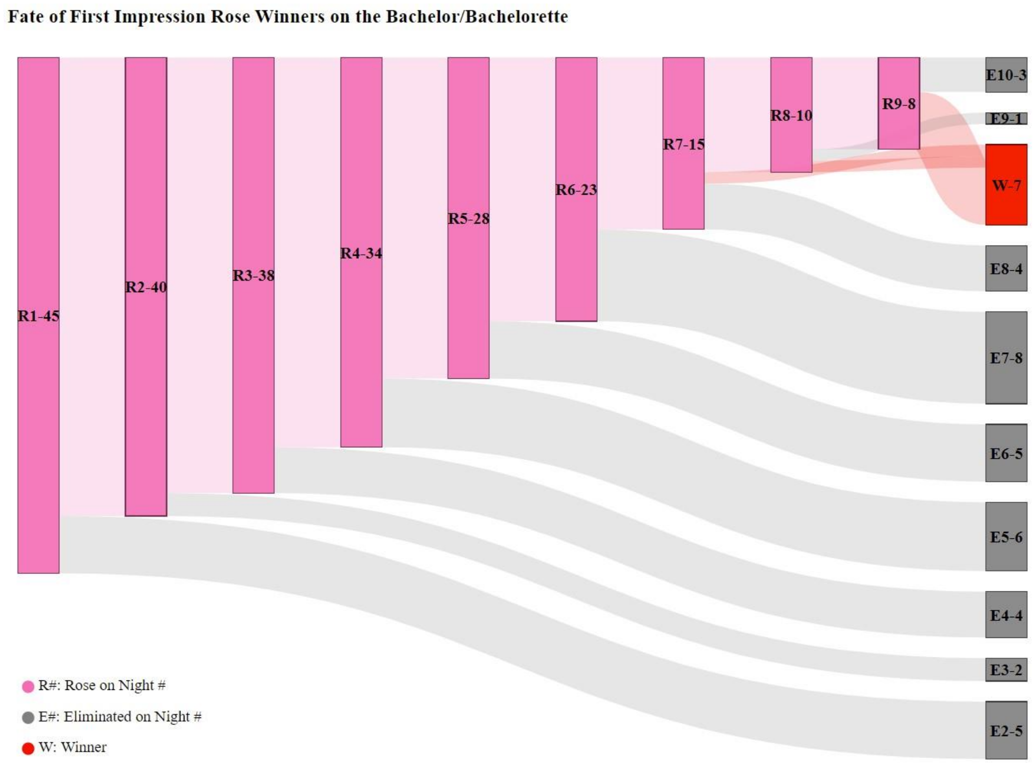
The purpose of this project was to familiarize ourselves with writing our own visualizations without the use of visualization aids such as Tableau. I used both JavaScript and Python to create my visualizations, in order to explore the many different libraries available. The datasets I explored were quite interesting. I chose them explicitly to give myself the opportunity to create a few different types of visualizations, and the process was quite instructive. I used Jupyter notebooks and Python to load in and analyze my data as a first step for all of the visualizations. The first visualization I started using matplotlib, then switched to the Python library Bokeh for more customization. For the second visualization I started with Bokeh, but ended up switching to JavaScript and using D3 for full customization. For the third visualization I used Python's Folium map library. Each of these libraries had their own nuances, and it was a great adventure in visualization techniques.

The first dataset was my own Apple Watch workout data. I was able to export the activity off the Apple Watch in order to visualize my exercise routine, which was fascinating for me. This data is located in the folder courtney_project04/src/Sample01-Fitness/Data1-fitness.csv, but is not publicly available as it is my own personal data. This dataset has 221 rows and 16 columns. The rows each represent a separate workout I recorded on my watch. The columns used in the visualization are ActivityType, StartDate, Duration (minutes), and TotalEnergyBurned (calories). I thought a scatter plot would be a reasonable way to start looking at this data, and I started with matplotlib because I was familiar with its features. I plotted the Duration of activities on the y-axis and StartDate on the x-axis. I immediately noticed a large outlier, my watch had recorded that I had done a 24 hour run in December. I've certainly never done something like that, so it was interesting to see the bug, but I dropped the outlier for visualization purposes. I wanted to color the points by activity type, so I added this grouping to the scatterplot. However, the legend was right on top of the visualization and I wasn't able to customize as much as I would have liked. I switched to using the Bokeh library instead, as it has really great interactive properties. A screenshot is below.



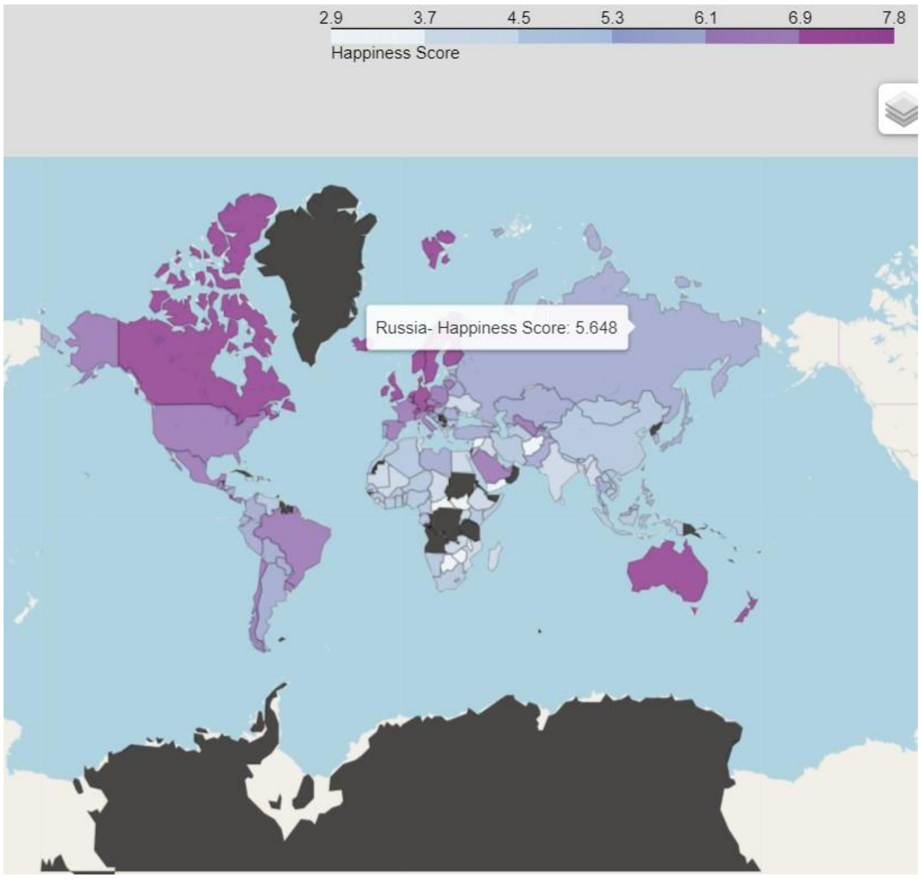
I changed the color scheme from the default to Set1 from ColorBrewer, as I found it provided the best contrast between the different activity types. I created a mapping between these colors and my activities. I also added in the ability for a user to "mute" an activity when they clicked on it in the legend, which required updating the click policy of the legend. This changes the intensity of the color of that activity's points from .8 to .2, giving the impression of "turning off" certain activities in order to emphasize others. This gives users freedom to explore the data. In addition, I added a Hover Tool which will tell you the calories burned during each workout if you hover over the point. These modifications were most interesting to help me explore my own activity data, and I was able to look at patterns in my activity. For instance, you can see that my new years resolution was to run more, as there are far more running dots after 1/2021! Bokeh was the perfect library for me use when exploring this data.

The second visualization uses a dataset of all contestants on the Bachelor and Bachelorette [1]. I had considered using this dataset for previous projects, but was glad that I didn't because it turned out to be harder to find an effective visualization than I had planned. The dataset has 885 rows and 23 columns. The rows represent each contestant on each season of the Bachelor and Bachelorette. The columns indicate the contestant's fate on the show. There are columns labeled E1-E10 that indicate the contestant's fate during each of 10 rose ceremonies. Contestants are either eliminated or given a rose to go into the next round. The question I set out to answer with this data was whether winning the coveted "First Impression Rose" is really indicative of how far a contestant will go in the competition. The First Impression rose is given out to the top contender(s) before the rose/elimination ceremony on night 1. I decided to model this with a Sankey diagram in order to show the flow of contestants from the First Impression Rose through all the episodes of the season. First, I needed to manipulate my data slightly to make a list of nodes, edges, and values to pass to the Sankey diagram libraries. I prepared this in a Jupyter notebook. Then I started by using Python and Bokeh's Sankey diagram. However, there wasn't an option for me to force the nodes into a certain order, and I didn't like the default order the algorithm had chosen. I decided to switch to using JavaScript and D3 so I could have full control over the algorithm. D3 also needed this list of nodes, edges, and values, so I saved it as Data2-nodes.csv which the HTML file implementing the Sankey module from D3 reads in. Unfortunately, CORS restrictions in most browsers will prevent pulling this file from a local directory. To get around this, I spun up a small server using python. In a command line, type "python -m http.server 8080", then you can access the visualization at https://localhost:8080/Index2.html. A screenshot is shown below.



In this visualization, I made quite a few modifications to the default D3 Sankey diagram [2]. The first, and the reason that I chose D3, is that the Sankey diagram by default tries to place each new node in the center vertically. This causes a lot of overlap of the link lines, and with my particular dataset, made it more difficult to see the flows. I made two modifications to the sankey.js file to fix this. First, I changed the ascendingDepth function, used to sort the edges, to go from b – a rather than a – b. This kept the ordering of elimination nights as seen above rather than reversed. Then, I changed the function center(node) to return 0 rather than the actual center of the visualization. This oriented all the nodes starting at the top of the viz. These two modifications combined to minimize overlap of the flows. Other modifications I made included changing the color of the nodes and links to pink for roses, gray for eliminations, and red for winners. I also added in a legend to explain this. I updated the node text to include the number of contestants left in each node, and I centered the text atop the nodes rather than having it float beneath. I also modified the spacing between nodes to make the graphic more compact. All of these modifications were really useful in personalizing the Sankey diagram to my dataset, and this diagram is interactive, so users can move the nodes around to see the data differently if they so choose.

The third dataset summarizes World Happiness Report [3] data from 2015. I wanted a dataset that would allow me to practice making a map visualization, and this was a good fit. This dataset lists countries and their happiness levels, as based on a survey conducted by Gallop. There are 158 rows and 12 columns in this dataset. The columns used in the visualization are country name and its happiness score, a value out of 10. I used the Folium library to display a map of each country. First, I used the geolocator library to pull down latitudes and longitudes for each of the countries, and I used these for a simple plot of the happiness score as a dot atop each country. What I really wanted was to color the countries on a scale according to their happiness score, and for this I used Folium's Choropleth map. To show a scale, a color gradient is always an effective choice. The final product can be seen below.



In order to effectively use Choropleth, I downloaded a geo JSON file (world_geo.json) [4] that Folium had already created with the coordinate data for the boundaries of countries. I changed the color scheme for the Happiness scale from Yellow-Green to Blue-Purple for a starker contrast. I also added a Hover tooltip over the different countries. This required updating the geo JSON file to include the text of the tooltip (world_geo_tooltip.json), and adding a layer to the map with this tip. The result is an interactive map that users can zoom, pan, and hover over to find out the happiness scores across the world. Folium was best for the task because of its wide range of geospatial graph options, and I appreciated that I could download the country coordinates for the choropleth without needing to generate it on my own.

In conclusion, this project allowed me to explore many different methods of creating my own visualizations. JavaScript and D3 are certainly better for making very personalized changes, and they give you the ability to access every little element of the visualization, so you have room to make something far more unique. The Python options, particularly Bokeh and Folium, are a perfect middle ground between D3 and software like Tableau. You're in control, and you can make some modifications, but they're also very powerful and beautiful visualizations that don't require you to know too many of the details about how the graphic is created. For speed and efficiency, I would choose the Python options, but for true customization, I'd choose D3. I'm certainly glad I experimented with both of them, it gave me great insight into what goes into a good visualization.

References

[1] https://www.kaggle.com/fivethirtyeight/fivethirtyeight-bachelorette-dataset

[2] https://bl.ocks.org/d3noob/06e72deea99e7b4859841f305f63ba85

[3] https://www.kaggle.com/unsdsn/world-happiness

[4] https://github.com/python-visualization/folium/tree/master/examples/data

[5] https://towardsdatascience.com/using-python-to-create-a-world-map-from-a-list-of-country-names-cd7480d03b10

[6] https://towardsdatascience.com/using-folium-to-generate-choropleth-map-with-customised-tooltips-12e4cec42af2

[7] https://python-visualization.github.io/folium/quickstart.html