CSE 564 Visualization and Visual Analytics - Lab2 Report

Student: Bo (Bryan) Cao SBU ID: 112130213
Email: bo.cao.1@stonybrook.edu or boccao@cs.stonybrook.edu

Demo
Video: https://youtu.be/QDyNf82vi28

Environment
Python: 3.7.0, D3: v5, jQuery: 3.3.1

Running the Code
In the terminal, locate to the directory **112130213_BryanBoCao_lab02_CSE564**, run **python3 app.py**, and enter http://127.0.0.1:5000/ in the address field of a browser. I use Chrome Version 72.0.3626.121 (Official Build) (64-bit).

File Structure
All files include **1) app.py, 2) templates/index.html,** and **3) College.csv,** where **1) app.py** is the server code to run in the backend and **2) index.html** is the file in the frontend to visualize data sent from backend; **3) College.csv** is the dataset downloaded from the college dataset from https://vincentarelbundock.github.io/Rdatasets/datasets.html, the original College dataset includes 777 data points, 18 dimensions.

Code
Backend
Python is used as the language for backend. The structure of **app.py** is depicted as follows:
@app.route("/", methods = ['POST', 'GET'])
index(), random_sample(), stratified_sample(), my_PCA(), top2_PCA_Vectors(), my_MDS() and compute_scatterplot_matrix().

When we enter http://127.0.0.1:5000/ on a browser, a GET request is sent to app.py and it goes into index(), where all the data for visualization is computed, including random_sample(), stratified_sample(), my_PCA(), top2_PCA_Vectors(), my_MDS() and compute_scatterplot_matrix(). random_sample() samples half of the population from all data. In stratified_sample(), clusters are computed by KMeans, the k which locates in the elbow is computed simply by comparing the speed of decrease of distortion - we iterate k from 0 to n, and determine the elbow_k as soon as the difference is less than the threshold 0.05. This elbow_k will also be sent to the frontend to highlight it, which will be shown in the next section. my_PCA() will take data matrix as input and return PCA object that includes PCA-related information and top 3 attributes with highest PCA loadings. The explained variance ratio is in the PCA object. Top 3 attributes are computed by summation square of all the components. top2_PCA_Vectors() simply takes the data as input and returns the matrix with respect to top 2 PCA vectors. my_MDS() takes data as input and returns the embedding MDS matrix, depending on the parameter “dissimilarity” to be Euclidean or Correlation. compute_scatterplot_matrix() is mainly to compute the scatterplot matrix for the convenience of the frontend to visualize. These information is displayed in the backend, such as:

```
Random Sampling...
Number of instance in df_sampled_data: 388
Number of dimension of df_sampled_data: 17
Stratified Sampling...
KMeans optimizing k using elbow
k: 1, distortion: 0.5801, dec: -inf
k: 2, distortion: 0.4929, dec: -0.0872
k: 3, distortion: 0.4515, dec: -0.0415
k: 4, distortion: 0.4225, dec: -0.0289
k: 5, distortion: 0.4050, dec: -0.0175
k: 6, distortion: 0.3924, dec: -0.0126
k: 7, distortion: 0.3829, dec: -0.0095
k: 8, distortion: 0.3756, dec: -0.0074
k: 9, distortion: 0.3691, dec: -0.0064
elbow_k: 2
cluster_ratio:  [0.6241956241956242, 0.3758043758043758]
Population of df_all_data: 777
Number of cluster 0: 242
Number of cluster 1: 145
Number of instance in df_ss_data: 387
Number of dimension of df_ss_data: 18
```

```
================================
All Data
pca_data_explained_variance_ratio_: [0.42997329 0.20454876 0.08219963 0.0665
0751 0.05507498 0.03714582
 0.02648041 0.02320302 0.01742888 0.01433827 0.01200863 0.00991943
 0.00776715 0.00636906 0.00474101 0.00170213]
pca_data_explained_variance_: [0.16638095 0.07915147 0.03180768 0.02573551 0
.02131162 0.01437382
 0.01024677 0.00897856 0.00674422 0.00554829 0.00464682 0.00383839
 0.00300555 0.00246455 0.00183457 0.00065865]
pca_data_singular_values_: [11.36272924  7.83718948  4.96817473  4.46886535
 4.06667147  3.33977279
  2.81983894  2.63957596  2.2876875   2.07496276  1.8989287   1.7258589
  1.52718914  1.38292851  1.19315653  0.71492189]
attribute_loadings:  [0.5837535105653665, 0.48042707064231044, 0.95358541990
66716, 0.9937575378689513, 0.9993011340561643, 0.9946394544973539, 0.9997089
28484454, 0.9986646043007749, 0.999970773033256, 0.9999979247172859, 0.99996
75333989514, 0.999876861650951, 0.9999991512516299, 0.9998846102735741, 0.99
99649860994082, 0.9965736815473321, 0.9999268177055698]
attribute_loadings_sorted:  [0.9999991512516299, 0.9999979247172859, 0.99997
0773033256, 0.9999675333989514, 0.9999649860994082, 0.9999268177055698, 0.99
98846102735741, 0.999876861650951, 0.999708928484454, 0.9993011340561643, 0.
9986646043007749, 0.9965736815473321, 0.9946394544973539, 0.9937575378689513
, 0.9535854199066716, 0.5837535105653665, 0.48042707064231044]

top3_attributes_i_ls:  [12, 9, 8]
top 3 attributes with highest PCA loadings:
     Terminal
     Books
     Room.Board
================================
```

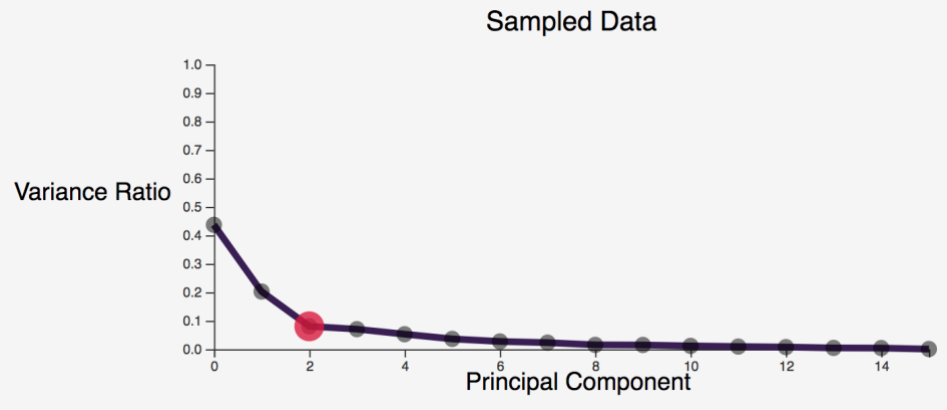
Note that when some data is visualized, users can update the data in the frontend by sending a POST request to the backend, the app.py will do all the computation mentioned above and jsonify the data to be sent to the frontend.

Frontend

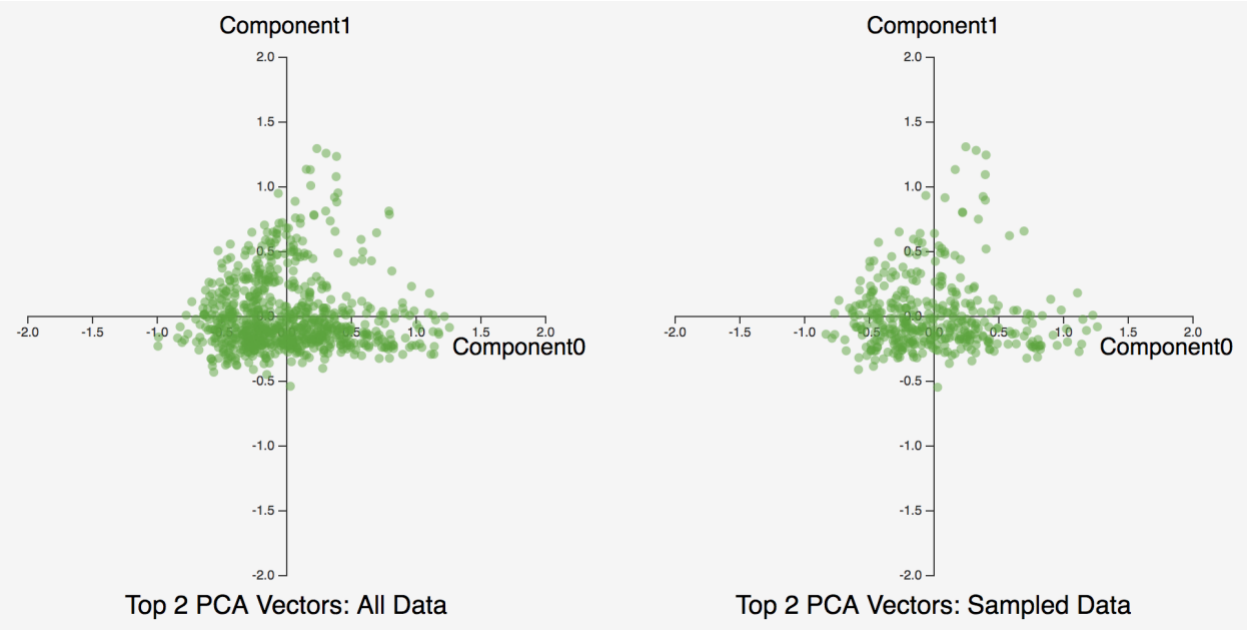
D3 is used for visualization, Bootstrap is used as the base css and jQuery is used to send GET/POST request to backend to retrieve new data.

When index.html gets the data from backend, it visualizes them in scree plot, scatter plot and scatterplot matrix. All data, sampled data and stratified sampled data are shown in scree plot while the latter two visualize all data and sampled data.

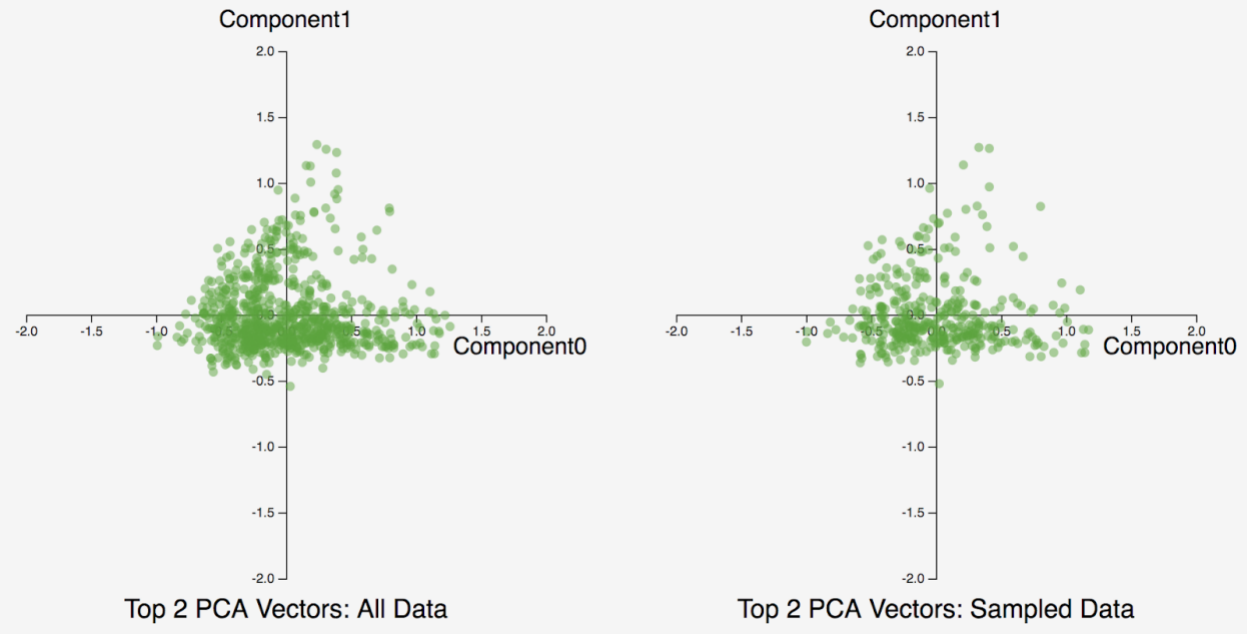
The picture below shows a scree plot for sampled data. Note that the elbow (k=2) is marked and highlighted in a larger red dot. This is done according to elbow_k computed from backend.



Data for Top 2 PCA Vectors is visualized in 2D scatter plot with origin in the center as shown in the picture below.



Note that if we resample data, we can see that the chart on the right side is different from the previous one (shown in the picture above and below). However, we can see that the overall shape still remains the same since they were sampled from the same distribution.



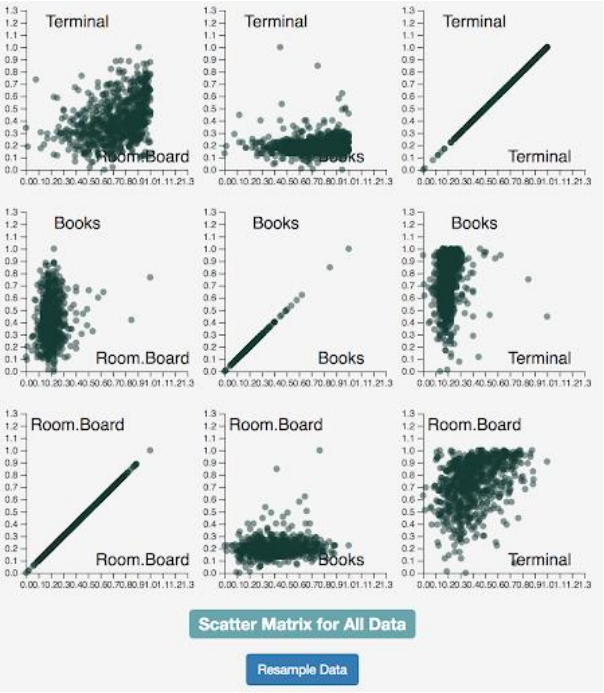
In terms of scatterplot matrix, I made a general scatter_plot() function that can specify scatterplot matrix. In the scatterplot_matrix() method, we iterate through the 3 by 3 matrix of the top 2 attributes computed from the backend and visualize each pair in the corresponding position. The snippet of the cod is shown below:

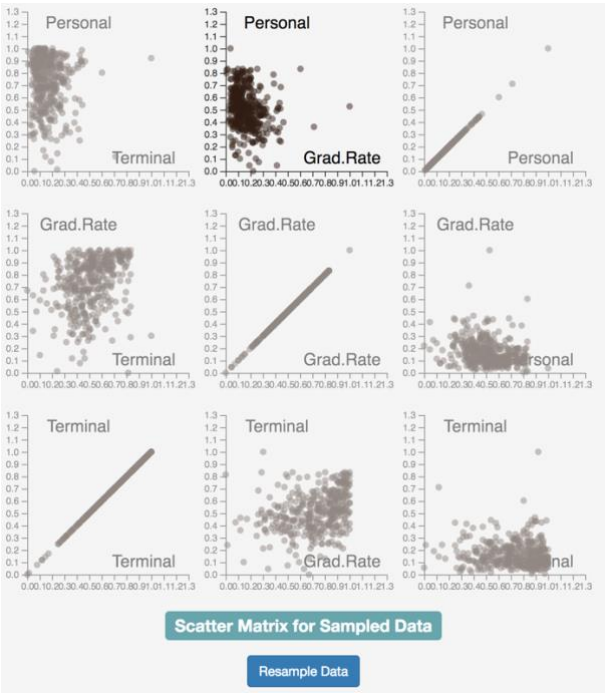
```
// ==================== scatterplot_matrix -- start ====================
function scatterplot_matrix(plot_type, data, top3_attributes_ls, title, x_axis_text, y_axis_text, color, domain, div_id){

    var pos = {i:0, n_i:0, j:0, n_j:0};
    // data should be in shape of (9, n_sample, 2)
    for (var ii = 0; ii < 3; ++ii) {
        for (var jj = 0; jj < 3; ++jj) {
            y_axis_text = top3_attributes_ls[ii];
            x_axis_text = top3_attributes_ls[2 - jj];
            scatter_plot(plot_type, data[ii * 3 + jj], title, x_axis_text, y_axis_text, color, domain, pos, div_id)
        }
    }

}
// ==================== scatterplot_matrix -- end ====================
```

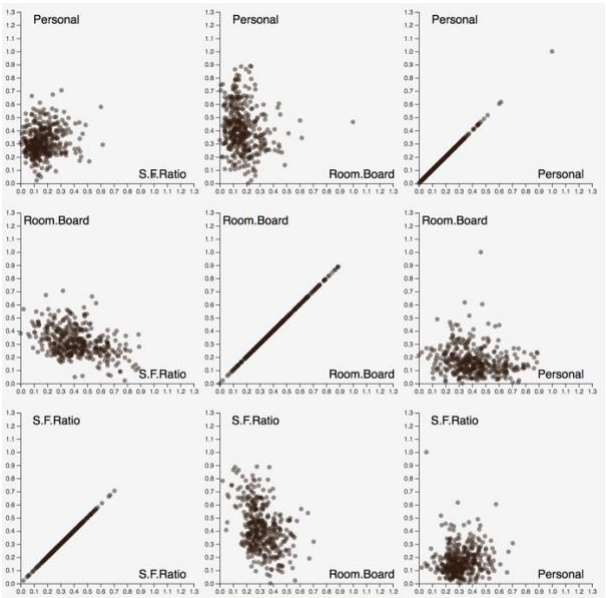
One thing to note is the fade-in fade-out effect for all charts. For instance, if I move my mouse over a specific scatter plot chart, the opacity will be 1.0; if the mouse is moved out of the chart, its opacity will reduced to 0.5 as shown in the right below picture, in which the middle top sub scatterplot is highlighted because user's mouse is over it while other 8 charts become a little transparent.





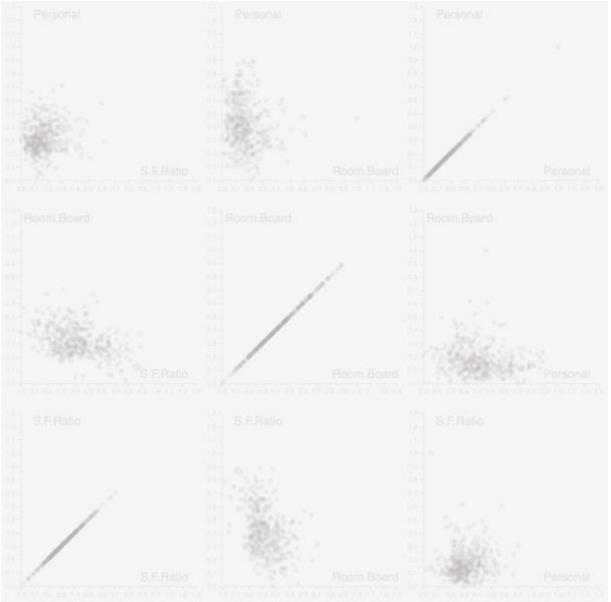
Sending a POST request is done by pressing the 

button. One interesting thing to note is that when the "Resample Data" button is pressed, all the charts including scree, scatter plot and scatterplot matrix's opacity will be reduced from 1 to less than 1. The idea is to give user a sense that these charts are inactive that the data is not fresh, so the frontend is waiting for the backend to compute and send new data. Once the frontend receives those data, it will update the visualization with new data. This effect is illustrated in the below picture (left: before resampling, right: after resampling and waiting for the data).



Scatter Matrix for Sampled Data

Resample Data



Scatter Matrix for Sampled Data

Resample Data