Assignment 4

on

CSCI 6610 Visual Analytics

Submitted by

Christian Gang Liu (B000415613/gn417126@dal.ca)
……

In partial fulfillment of the requirements for the Course CSCI 6610 Human Computer Interaction

Computer Science

2019-11-19

DEPARTMENT OF Computer Science

Present to Teaching Assistant : Mateus Pereira; Leonardo Milhomem Franco Christino
&
Professor : Fernando Paulovich, PhD

DALHOUSIE UNIVERSITY

DEPARTMENT OF Computer Science

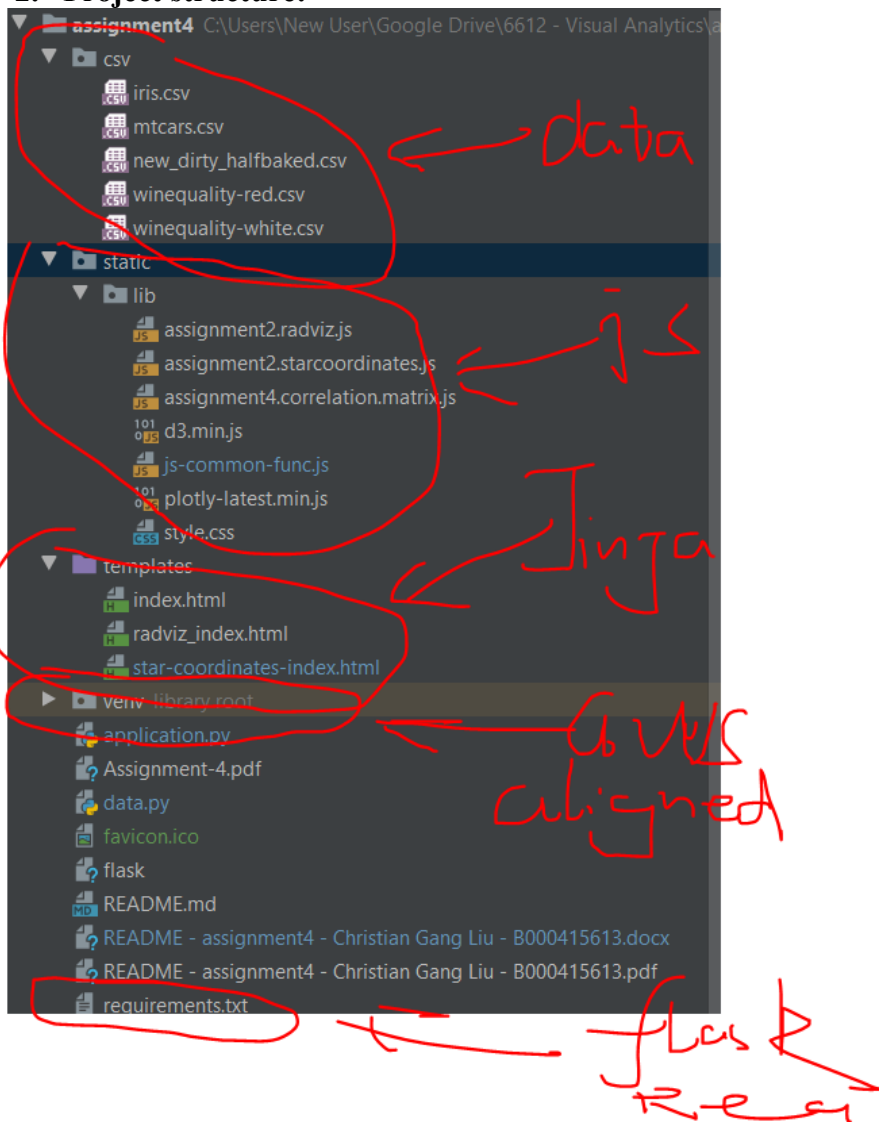
Important Notes: because I am principle programmer of our "outliers" group project implementation, so you might find similar coding between my assignment 4 and project.

1. **Project technical used:**

General speaking, the project is based on JS and Python 3.4+ as required by assignment 4 instruction.

1, Flask: A python framework used for quick REST services establishment;
2, D3: A Javascript library used for generating the interactive ML graphic charts;
3, Plotly: A popular javascript library which based on D3 provides the complicated chart view (here I only used it for show the tabular table view on the page, this requirement is not asked by assignment, I used it only to demonstrate how data comes from backend framework (Flask)
4, Others: JQuery / Bootstrap (only for UI layout and event triggering)
5, IDE: PyCharm
6, AWS (extra work): public cloud deployment:
http://assignment4-dev.us-east-2.elasticbeanstalk.com/

2. **Project structure:**



As you can see above, I constructed project structure to comply with the AWS elastic sever standardization, in order to deploy on AWS for public access.

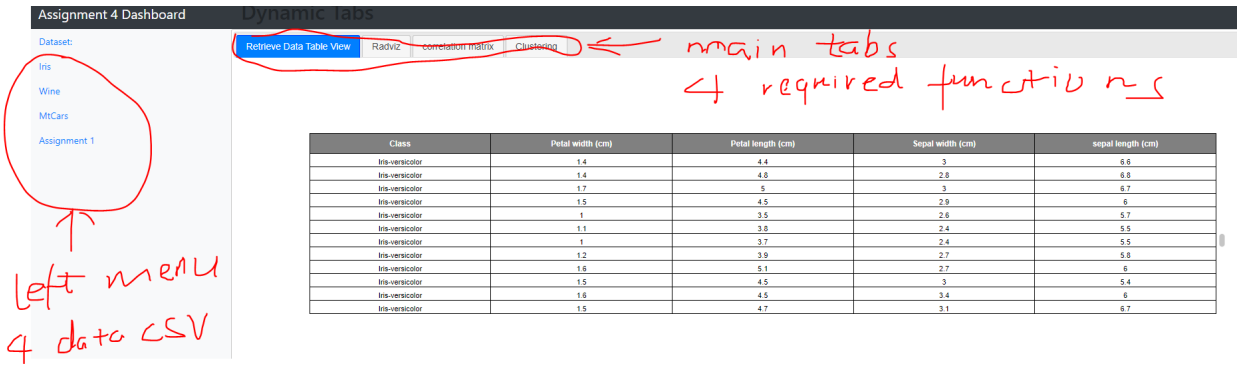
I utilized FLASK framework to quickly establish this lightweight project, like what I did for my group project as well (http://flask-env.zsewpnnzda.us-east-2.elasticbeanstalk.com/).

In order to simplify the assignment 4 review work. I planned to deploy it on AWS for quickly reviewing. Anyhow you can always open my project source folder to execute command: python application.py to exam it on local.

3. **Execution (the execution instruction is also mentioned in the README file under project folder:**

4. **General UI introduction (the details will be described specifically to each of questions of assignment 4):**

UI layout introduction:



a. Left Menu: they are four different tables which are required by assignment 4:
   i. Iris
   ii. Wine
   iii. Mtcars
   iv. Assignment 1 dataset (bonus)

b. Main Menu:
   i. Tabular view on Html page (extra work): it shows tabular view directly from JSON response of Flask REST services:
   ii. Radviz: The assignment 2 functionality for each of tables;
   iii. Correlation Matrix: Heatmap of each table;
   iv. Clustering: there will be a comparison between original labels (classification) and clustering by K-means, as well as a measurement score showing on the page.

   Notes: the Bonus requirement of assignment 1 dataset will share the same functionalities.

5. **The specific answers to the assignment 4 questions:**

[100 (+30)] Requirements:

Code will be marked based on functionality, structure, reusability, best practices, and documentation.

1. [10 Marks] Create a backend that will provide the data and metadata that can be used to display the visualization
    a. Use an HTTP request to retrieve data from the back-end and use it to generate the visualization on the front-end.
    b. Tip: You can return, along with the data, some metadata like column names or other information that could be useful to handle and/or display the data in the front-end.

Answer:
Front-end (JS retrievals JSON data, sends to html page):

Class	Petal width (cm)
Iris-versicolor	1.4
Iris-versicolor	1.4
Iris-versicolor	1.7
Iris-versicolor	1.5
Iris-versicolor	1
Iris-versicolor	1.1
Iris-versicolor	1
Iris-versicolor	1.2
Iris-versicolor	1.6
Iris-versicolor	1.5
Iris-versicolor	1.6
Iris-versicolor	1.5

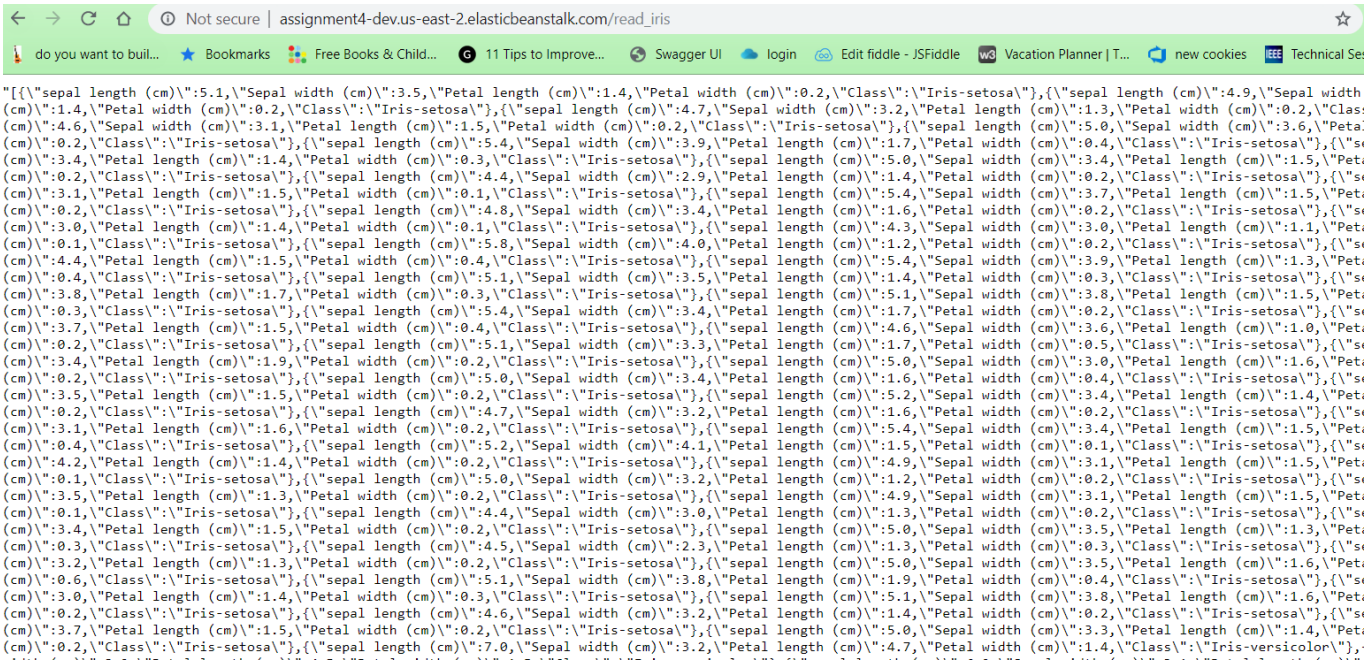
Petal length (cm)	Sepal width (cm)	sepal length (cm)
4.4	3	6.6
4.8	2.8	6.8
5	3	6.7
4.5	2.9	6
3.5	2.6	5.7
3.8	2.4	5.5
3.7	2.4	5.5
3.9	2.7	5.8
5.1	2.7	6
4.5	3	5.4
4.5	3.4	6
4.7	3.1	6.7

Back-end Flask WSGI endpoint (REST services) – Code snippets:

```
import data
from flask import Flask, jsonify, render_template, request

application = Flask(__name__)
assignment4Data = data.getAssignment4Data()


@application.route('/')
def hello_world():...

'''...'''
@application.route('/read_iris')
def read_iris():...


@application.route('/read_iris_correlation')
def read_iris_correlation():...


@application.route('/read_iris_clustering')
def read_iris_clustering():...


@application.route('/read_wine')
def read_wine():...


@application.route('/read_wine_correlation')
def read_wine_correlation():...
```

```python
@application.route('/read_car_clustering')
def read_car_clustering():...


@application.route('/read_assignment1')
def read_assignment1():...


@application.route('/read_assignment1_correlation')
def read_assignment1_correlation():...


@application.route('/read_assignment1_clustering')
def read_assignment1_clustering():...


@application.route('/show_table')
def show_table():...


if __name__ == '__main__':
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

The returning JSON sample like this: (http://assignment4-dev.us-east-2.elasticbeanstalk.com/read_iris)



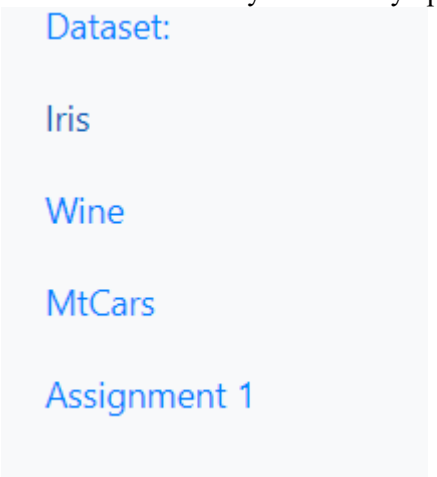
The list of REST services showing below (hostname is http://assignment4-dev.us-east-2.elasticbeanstalk.com):

- @application.route('/')
- @application.route('/read_iris')
- @application.route('/read_iris_correlation')
- @application.route('/read_iris_clustering')
- @application.route('/read_wine')

- @application.route('/read_wine_correlation')
- @application.route('/read_wine_clustering')
- @application.route('/read_car')
- @application.route('/read_car_correlation')
- @application.route('/read_car_clustering')
- @application.route('/read_assignment1')
- @application.route('/read_assignment1_correlation')
- @application.route('/read_assignment1_clustering')
- @application.route('/show_table')

2. [20 Marks] Add an option on the interface to choose a different dataset (iris or winequality)
   a. The backend will return the new dataset

   Front-end: when you click any option in left panel, the back-end will return corresponding response:



Back-end returning:

IRIS: http://assignment4-dev.us-east-2.elasticbeanstalk.com/read_iris
WINE: http://assignment4-dev.us-east-2.elasticbeanstalk.com/read_wine
MTCARS: http://assignment4-dev.us-east-2.elasticbeanstalk.com/read_car
ASSIGNMENT1: http://assignment4-dev.us-east-2.elasticbeanstalk.com/read_assignment1

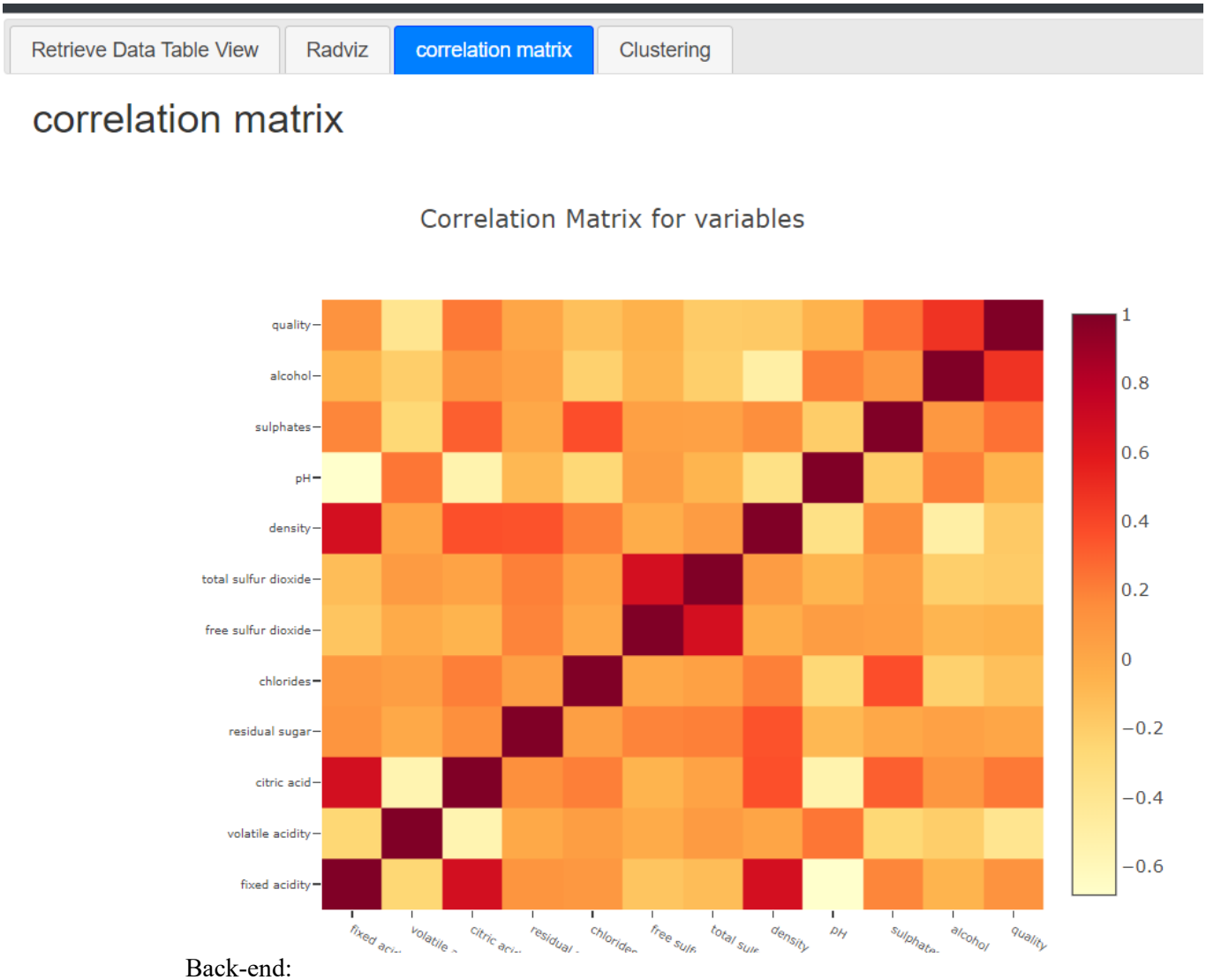
   b. You can keep a state on the front-end and send it on every request to identify the current dataset in use.

      I am using the request parameter /?data=[dataset_name] to identify current dataset using:

Like /?data=[iris|wine|assignment1|car]

3. [20 Marks] When hovering an instance of a given cluster, show (as a tooltip or in other available space) the correlation matrix for instances of that cluster.
    a. The correlation matrix should be calculated and returned by the back-end.

Front-end:



Back-end:
So whenever you click “correlation matrix” tab, it will send REST call like:

In this example:

```
  @application.route('/read_wine_correlation')
def read_wine_correlation():
    return jsonify(data.read_wine_correlation())
```

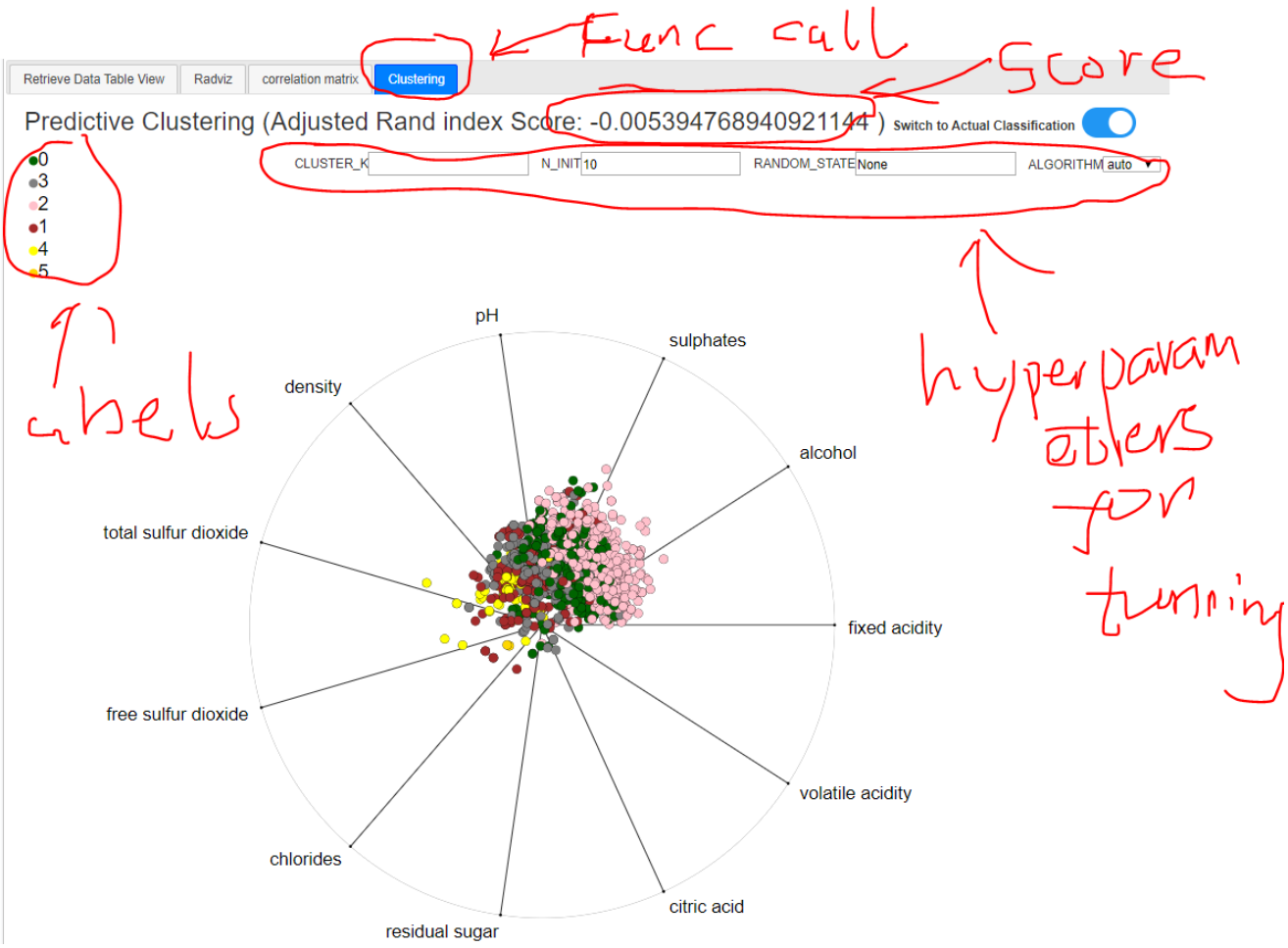
Supposedly we want to calculate win correlation, it will call backend function “read_wine_correlation”:

```
def read_wine_correlation() -> readAssignment4Data:
    rawData = getAssignment4Data().wine
    correlation_result = rawData.corr()
    return correlation_result.to_json(orient='records')
```

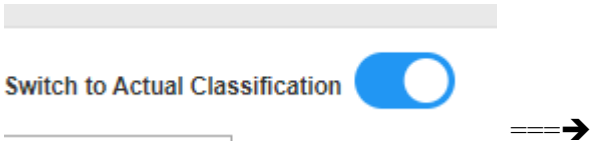
b. This should be displayed on the front-end through a color matrix. See example of such matrix below (Tip: feel free to use libraries to help you):

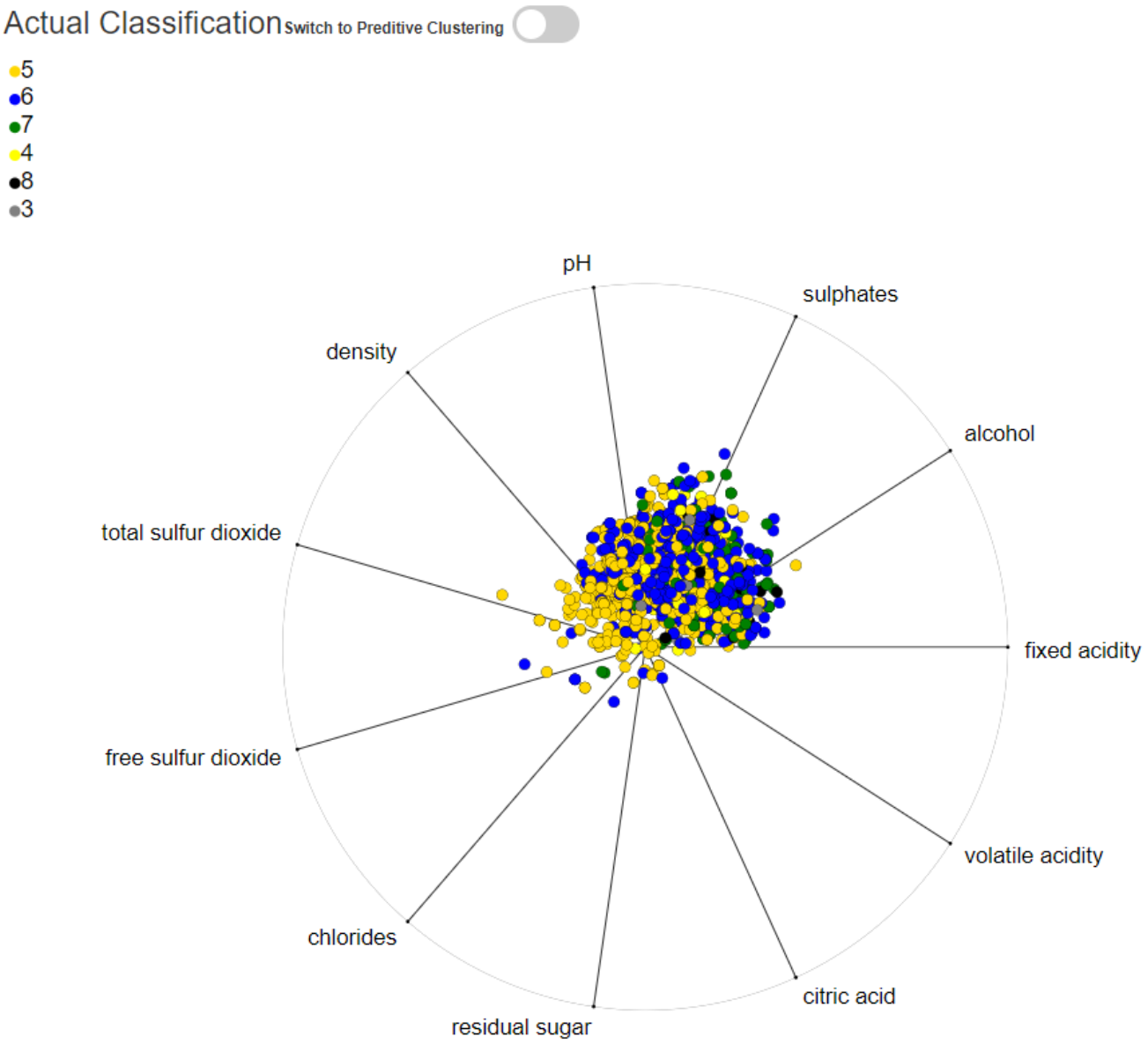
So I utilize the corr() function of pandas to generate correlation heatmap

4. [40 Marks] Implement a button that requests the backend to clusterize the data using one of: K-Means or DBScan
   a. You should color the instances using the clustering information



b. Add a switch button to choose between the color modes: cluster colors or class-based colors.





c. The clusterization should be performed on the same dataset currently seem in the visualization.

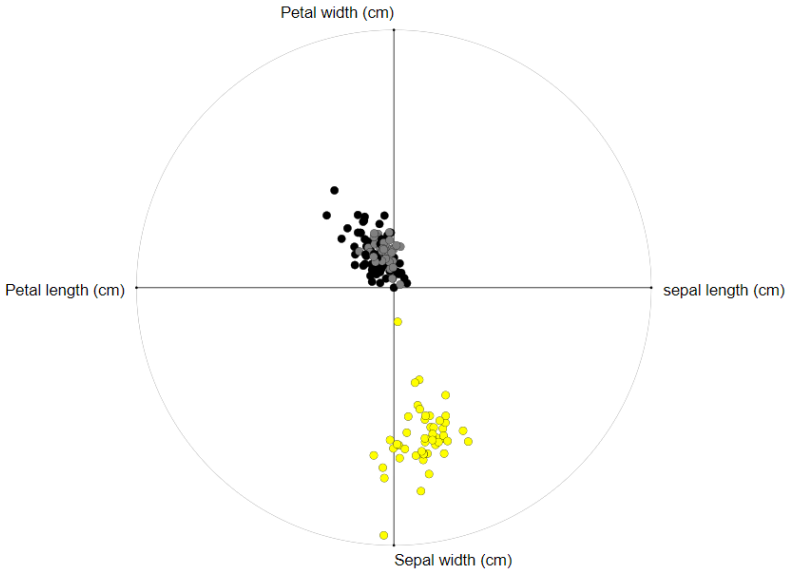
Since **/?data=[dataset_name]** remembers the current dataset.
Clustering tab will go further to call
@application.route('/read_iris_clustering')
@application.route('/read_wine_clustering')
@application.route('/read_car_clustering')
@application.route('/read_assignment1_clustering')

Page will show corresponding result to current dataset:

Predictive Clustering (Adjusted Rand index Score: 0.7302382722834697) Switch to Actual Classification

1
0
2

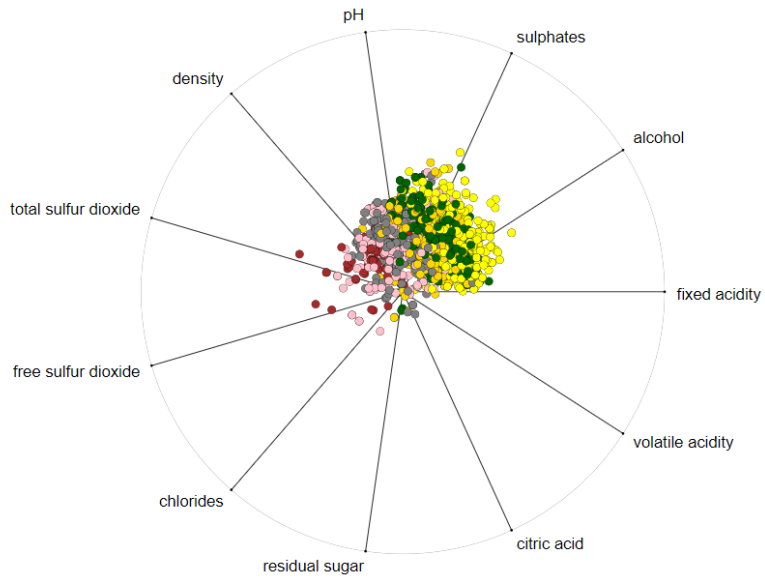
CLUSTER_K | N_INIT 10 | RANDOM_STATE None | BATCH_SIZE 100

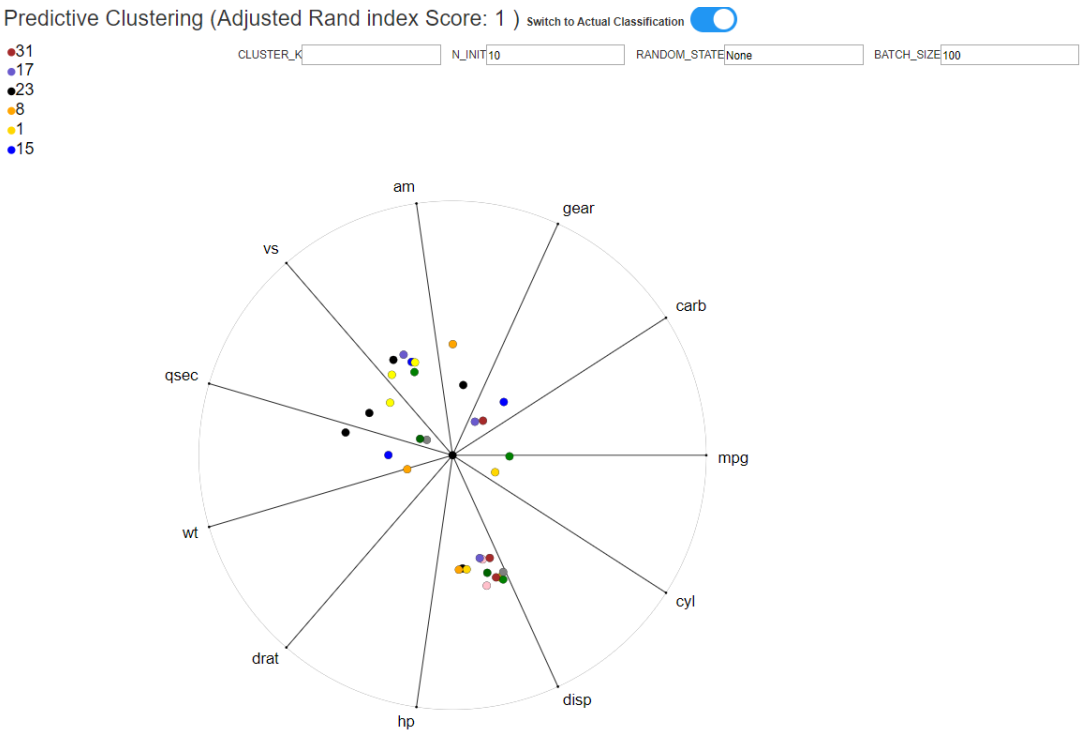


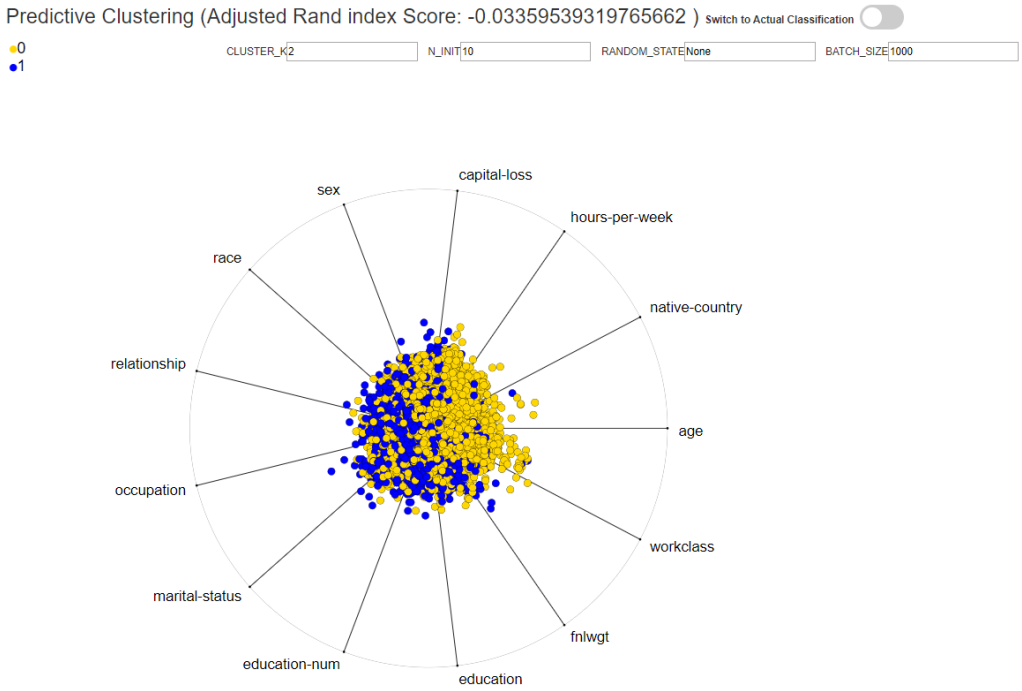
Predictive Clustering (Adjusted Rand index Score: 0.006445156285441907) Switch to Actual Classification

5
3
1
4
0
2

CLUSTER_K | N_INIT 10 | RANDOM_STATE None | BATCH_SIZE 100







d. You just need to implement for <u>one</u> clusterization algorithm.

K-MEAN

e. You may use existing implementations of the clustering algorithms.

SKLearn.cluster.kmean

5. [10 Marks] Add one (or more) options to configure the parameters of the clustering algorithm
   a. Clicking the button should make a new clusterization with the new parameters and update the colors on the visualization.

Here I am going to tune up four major different parameters (Quoted from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html)

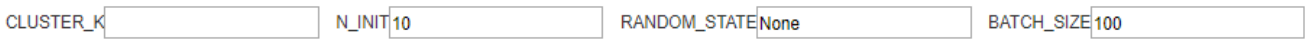
n_clusters : int, optional, default: 8
The number of clusters to form as well as the number of centroids to generate.

batch_size : int, optional, default: 100
Size of the mini batches.

random_state : int, RandomState instance or None (default)
Determines random number generation for centroid initialization and random reassignment. Use an int to make the randomness deterministic.

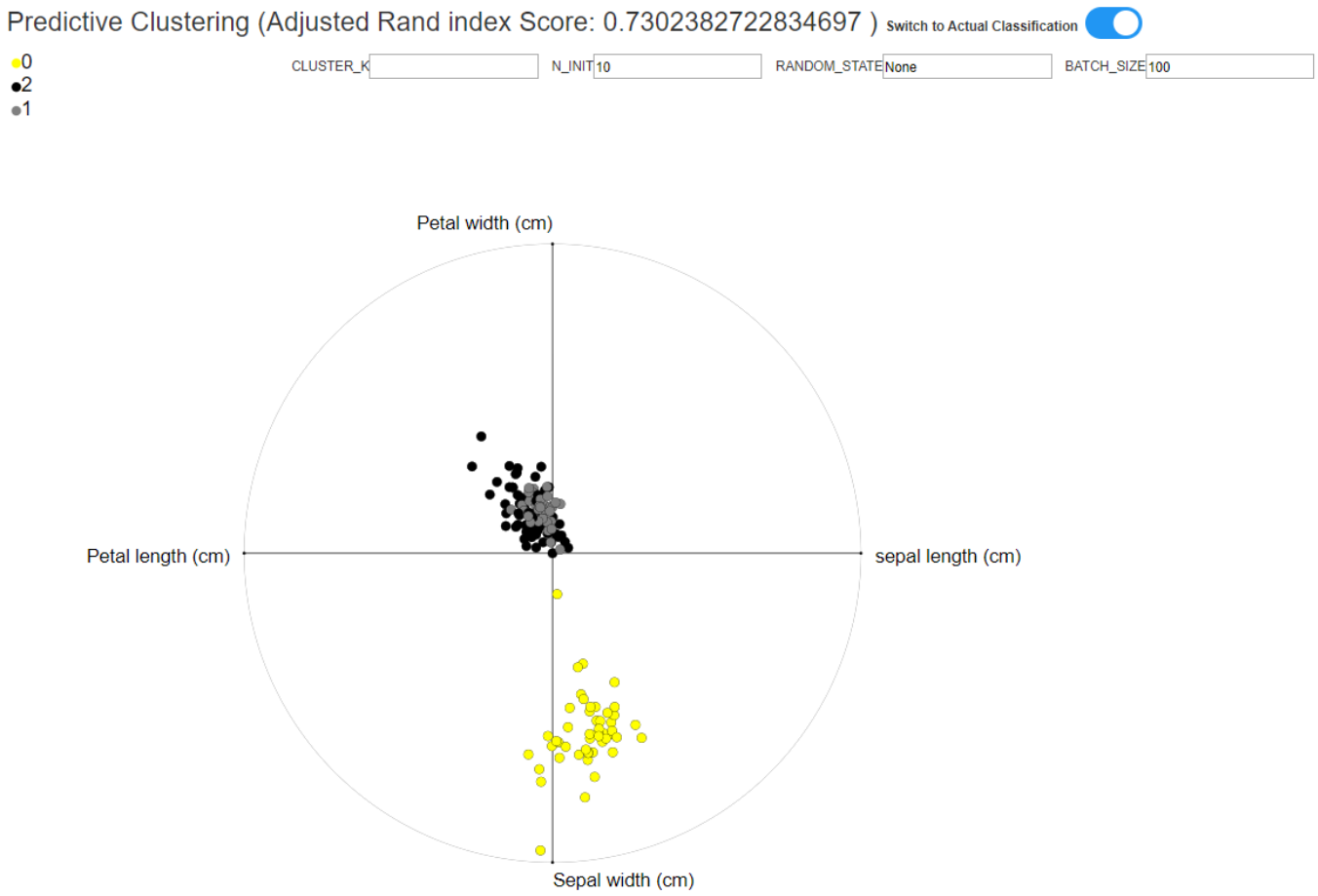
n_init : int, default=3
Number of random initializations that are tried. In contrast to KMeans, the algorithm is only run once, using the best of the n_init initializations as measured by inertia.

Toolbar:

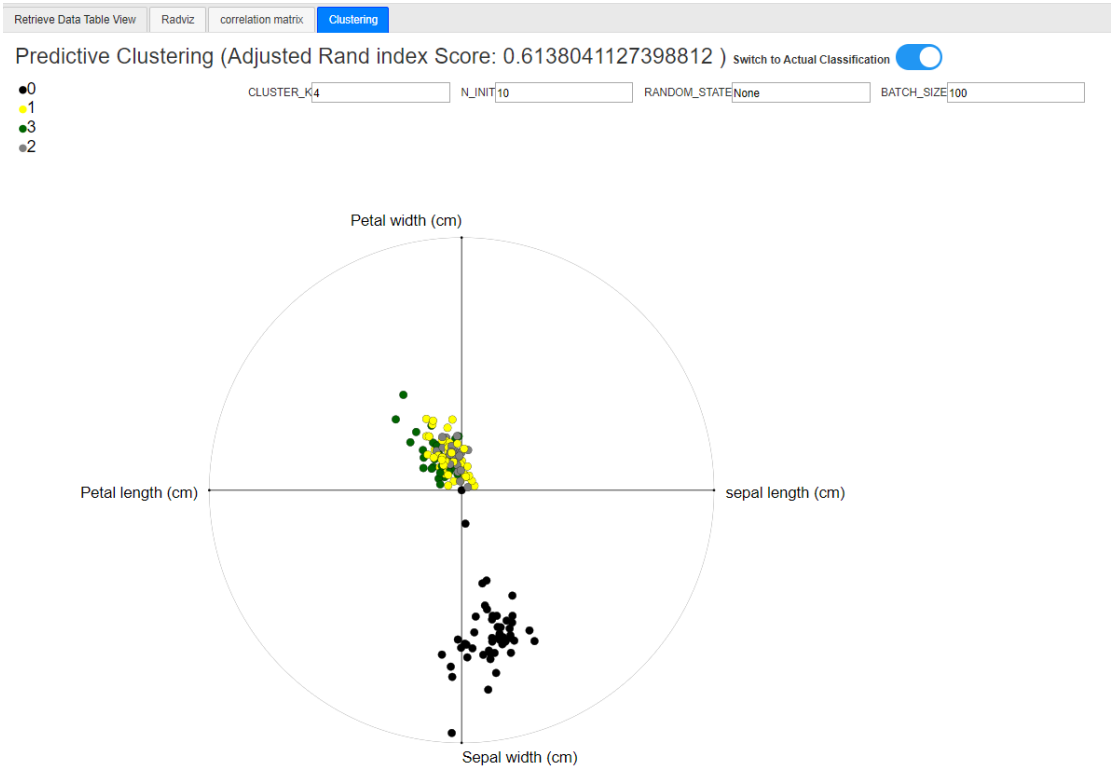


It will allow us to adjust four different major parameters of k-mean , let us see the effects:

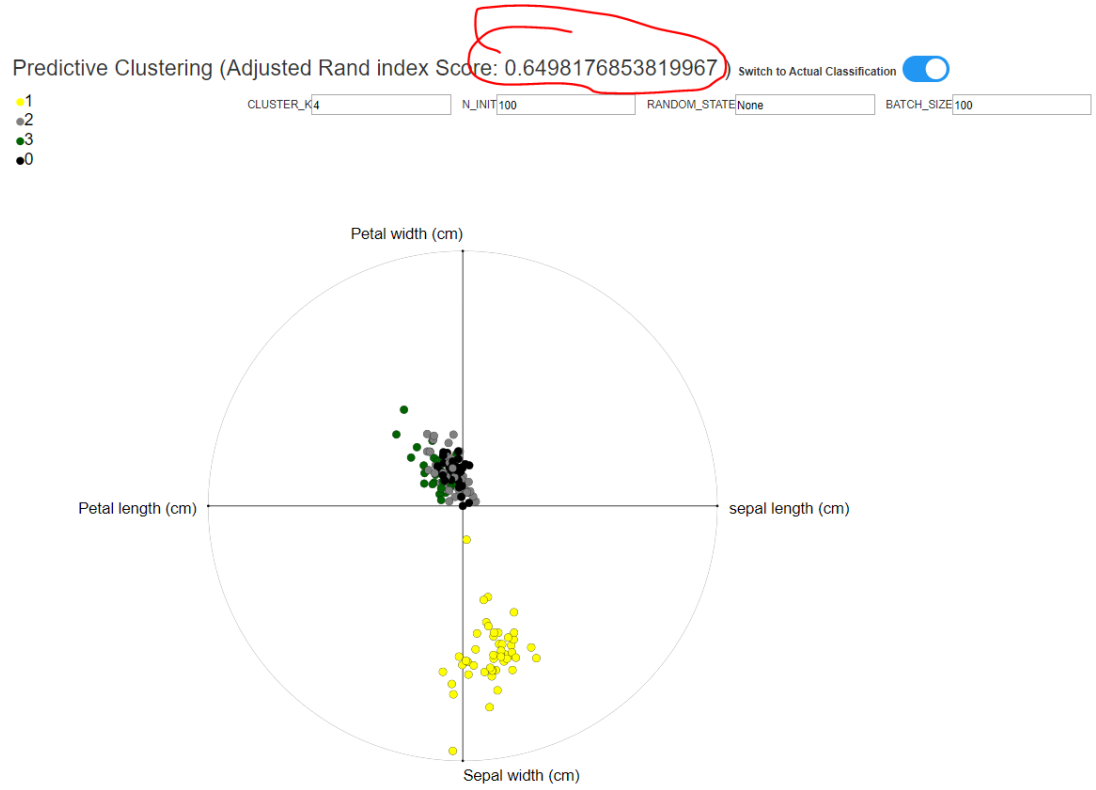
Default one:



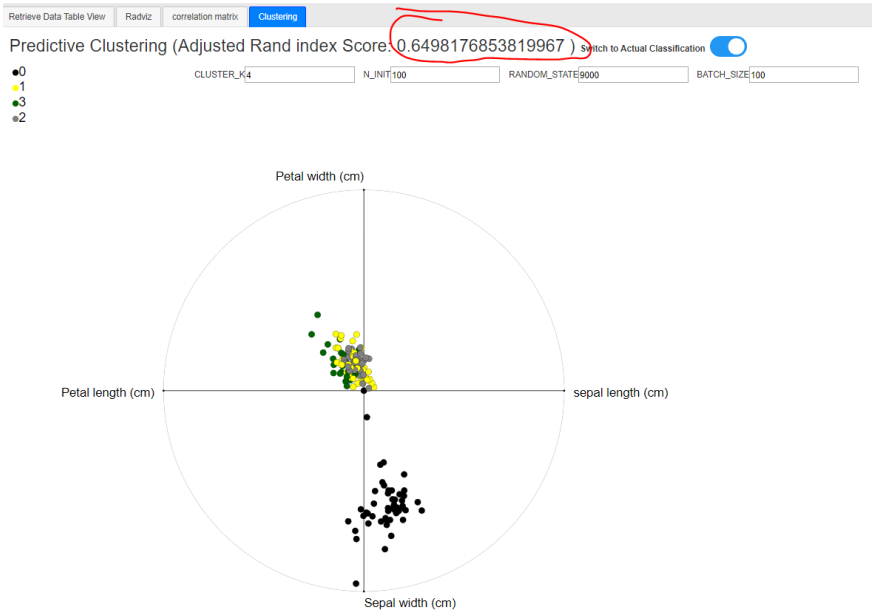
K parameter adjustment:



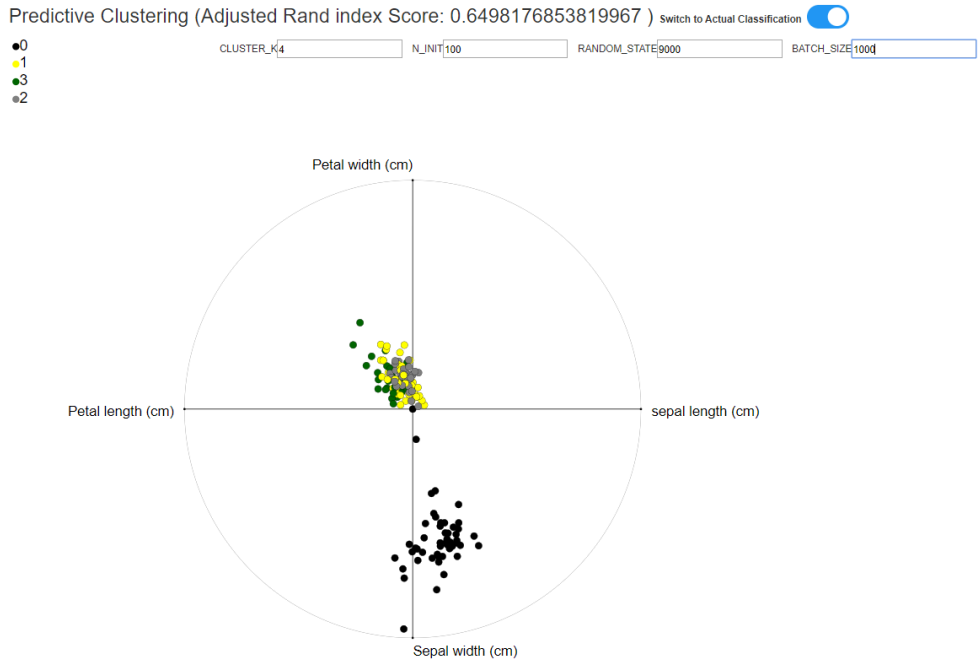
N_init adjustment:



Random_state adjustement:



Batch_size adjustment:



You can see the result will slightly different as we tune the parameters based on the measurement score.

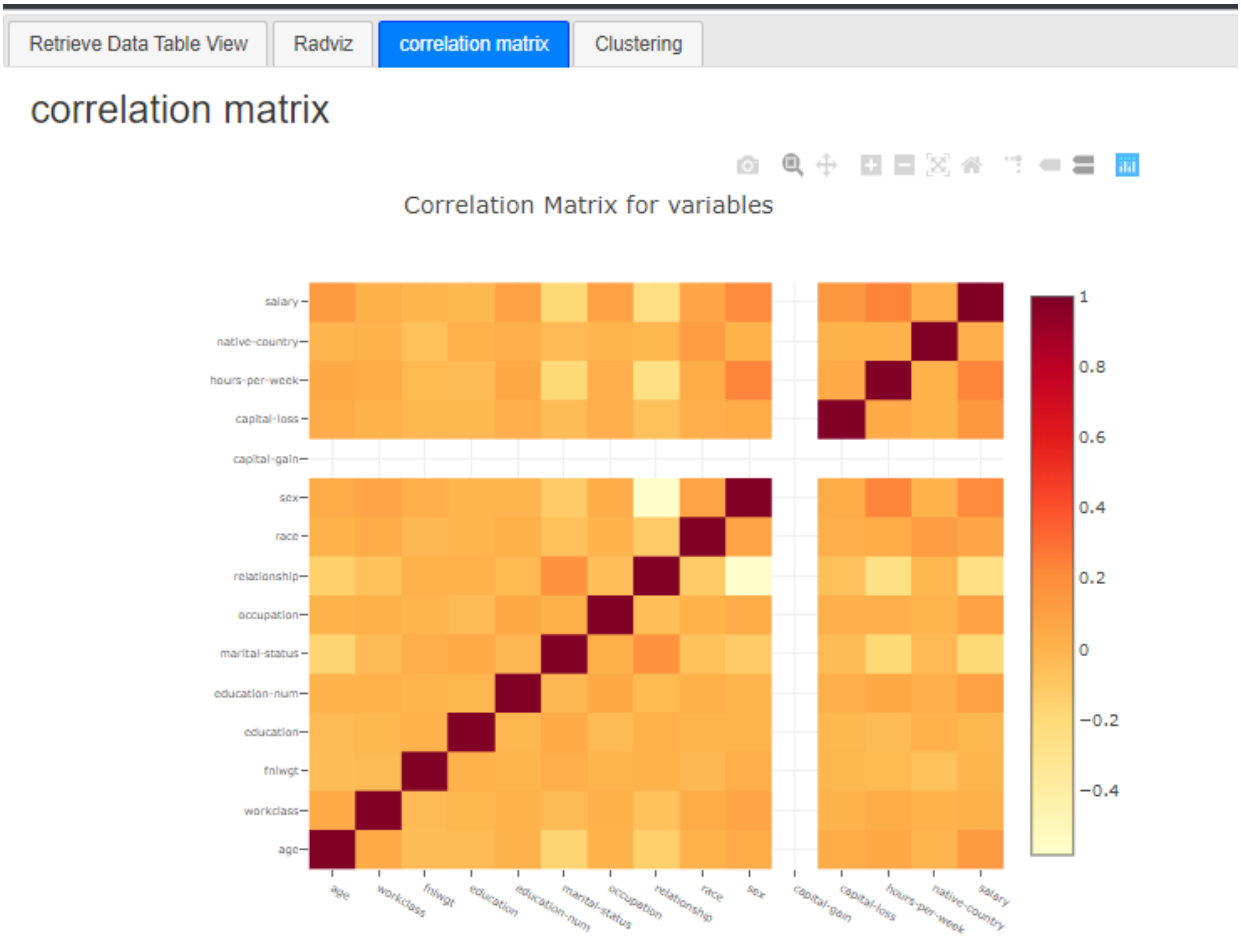
6. [+30 Bonus Marks] Add an option in the interface to see the preprocessed dataset generated by your A1 assignment.
   a. RadViz/StarCoordinates should only show the numerical columns as anchor points

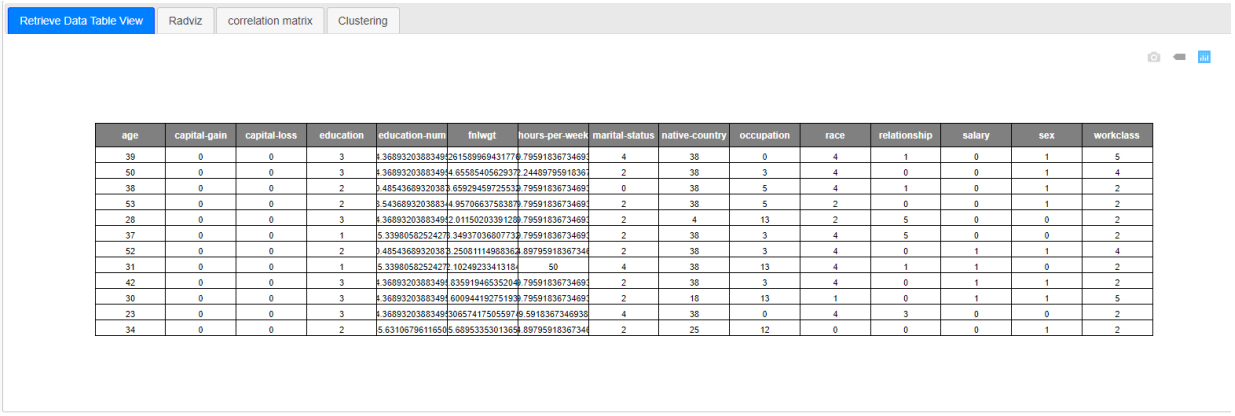
      Using sklearn normalization library to scale and label the columns as digit numbers

   b. The Categorical columns should be shown as the color of the plot. Make an input box selector in the interface to choose the categorical column to be shown as the color.

      Make the salary as label: <50K = 0, >50K = 1

Assignment 1 dataset will share same functionalities as three other datasets:



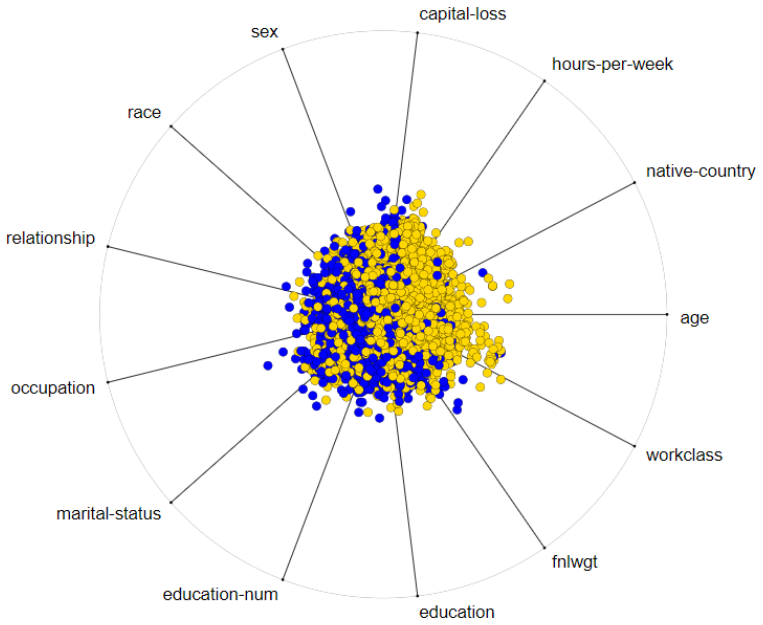


age	capital-gain	capital-loss	education	education-num	fnlwgt	hours-per-week	marital-status	native-country	occupation	race	relationship	salary	sex	workclass
39	0	0	3	4.368932038834951	2.615899694317765	6.795918367346939	4	38	0	4	1	0	1	5
50	0	0	3	4.368932038834951	4.655854056293726	2.448979591836735	2	38	3	4	0	0	1	4
38	0	0	2	0.485436893203883	3.659294597255326	6.795918367346939	0	38	5	4	1	0	1	2
53	0	0	2	3.543689320388349	4.957066375837977	6.795918367346939	2	38	5	2	0	0	1	2
28	0	0	3	4.368932038834951	2.011502033912887	6.795918367346939	2	4	13	2	5	0	0	2
37	0	0	1	5.339805825242718	3.349370368077328	6.795918367346939	2	38	3	4	5	0	0	2
52	0	0	2	0.485436893203883	3.250811149683628	2.897959183673469	2	38	3	4	0	1	1	4
31	0	0	1	5.339805825242718	3.102492334131849	50	4	38	13	4	1	1	0	2
42	0	0	3	4.368932038834951	3.835919465352049	6.795918367346939	2	38	3	4	0	1	1	2
30	0	0	3	4.368932038834951	4.600944192751939	6.795918367346939	2	18	13	1	0	1	1	5
23	0	0	3	4.368932038834951	4.306574175055974	3.591836734693878	4	38	0	4	3	0	0	2
34	0	0	2	5.631067961165049	5.689533530136599	4.897959183673469	2	25	12	0	0	0	1	2

Predictive Clustering (Adjusted Rand index Score: -0.03359539319765662) Switch to Actual Classification

0
1

CLUSTER_K 2 N_INIT 10 RANDOM_STATE None BATCH_SIZE 1000



Actual Classification Switch to Preditive Clustering

0
1