Report

Data Visualization: Mini Project 2

Author: Rohit Rawat

SBU ID: 112963417

Dataset:

The dataset is collaboration of two data files: Meter dataset of a house for a year and weather data of the same area of the city where that house is located. Meter dataset contains power consumption by all the electrical appliances in the house for every half an hour of a year. Also, Weather dataset contains feature columns about the weather-related data like temperature, humidity, summary of weather, icon, visibility, pressure, cloud Cover, etc. Weather dataset contains per hour records. All the features have numerical values except two features i.e. “icon” and “summary” which are categorical features.

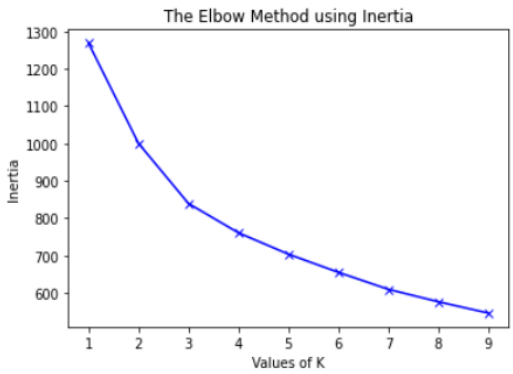
Data Challenges and Preprocessing:

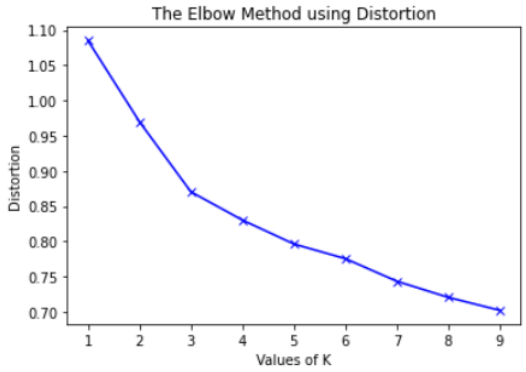
We have encountered following challenges with given datasets:

1. The meter datasets contain per half an hour records, but weather datasets contain per hour records. So, we collaborated both them by converting weather dataset into per half hour records.
2. There are two categorical features (“icon” and “summary”) which need to be handled before processing through PCA and MDS functionalities as they process numerical features only. So, we used label Encoding to handle these categorical columns.
3. There are Non defined (NaN) values in few features of weather dataset, which are 'visibility','pressure','windSpeed','windBearing', and 'cloudCover'. Thus, we replaced the NaN values with mean of that column.
4. We have saved the collaborated data as “/datasets/dataset_withcol.csv”.
5. All the attributes collectively are not standardized or normalized i.e. they contains different range of values in different features. We have standardized the dataset in range of (0,1) using MinMaxScaler() function of sklearn.preprocessing library and saved it as our original dataset named as “/datasets/*dataset_mm.csv”.*

Technology Stack: JavaScript, HTML, CSS, python, Visual Code, d3.js, Flask, VisualCode.

- **Task 1: Generation of Random and Stratified Sampled Data**
    - After standardizing the dataset, I have applied k means clustering on the dataset and generated following Elbow curves using Distortion and Inertia.





    - After observing the curve we concluded that the optimal number of clusters or the k-value of K-means clustering should be 3.
    - Then we have finally clustered the dataset with k=3 clusters and then took out the 25% of data records from each cluster as **Stratified Sampled Data** and saved it as "/datasets/*dataset_strata.csv".*
    - We have also randomly sampled 25% of data records from dataset_mm.csv as **Random Sampled Data** and saved it as "/datasets/*dataset_random.csv".*
    - Up to here we have done all the steps in Google Colaboratory notebook named as *"Assignment2-DataPreprocessing.ipynb".*

- **Task 2: Dimension Reduction using PCA**
    - First, we step up folder structure in Visual code, then we created a Virtual Environment named as '***flaskenv***' to install all the necessary libraries, including **"Flask".** Flask is a library of python which helps in generating a production similar environment and server for developing the functionalities as development mode.
    - Then we set the flask environment and app variable by using following commands:

      set FLASK_ENV=development

      Set FLASK_APP=app

      Now, we are ready to use the flask, virtual env (after installing required libraries) and rest of the functionalities.
    - ***"/app.py"*** is the base file of python from where we have called other functionalities of python and rendered the front-end development files i.e. "/*index.html*" file from where we called other JavaScript files.
    - ***"/pca_component.py"*** is the file with all the function definitions in python which we called in /app.py. It contains functions related to Task 2 and Task3 of python.

- **<u>Task 2(i):</u>** ***def pca(dataset):*** It takes the dataset as the input parameter and returns the list of pca_component, explained variance ratio, cumulative explained variance ratio, and pca object fit_transform on the dataset.

- **<u>Task 2(ii) and Task2(iii):</u>** Under app.py we have declared different functions which passes data to HTML file and then get processed there using JavaScript.

```
#   Task 2(ii): produce scree plot visualization and mark the intrinsic dimensionality and
#   Task 2(iii): show the scree plots before/after sampling to assess the bias introduced
@app.route("/")
def index():
    return render_template("index.html")
```

Following @app.route("/task2") sends data for scree plot and intrinsic dimensionality which we got in Task2(i) above.

```
@app.route("/task2")
def get_data():

    data = {
    "pcomponent_mm":    pcomponent_mm,  "expl_var_ratio_mm":  expl_var_ratio_mm,"cum_expl_var_ratio_mm":  cum_expl_var_ratio_mm,
    "pcomponent_random":    pcomponent_random,  "expl_var_ratio_random":  expl_var_ratio_random,"cum_expl_var_ratio_random":  cum_expl_va
    "pcomponent_strata":    pcomponent_strata,  "expl_var_ratio_strata":  expl_var_ratio_strata,"cum_expl_var_ratio_strata":  cum_expl_va
    }

    return data
```

- **<u>Task2(iv):</u>** ***def task2_top3attr(pca, colnames):*** It takes pca object and list of all column names as input and returns list of top 3 columns name with highest PCA loading.

```
#   Task 2(iv):   Top 3 attributes with highest loading

#   Getting column names
df_forcolm = pd.read_csv('datasets/dataset_withcol.csv')
colnames = list(df_forcolm.columns)
colnames.remove('Unnamed: 0')   #Removing default index column

#task2_top3attr() from pca_component.py to obtain top 3 attributes
top3_attr_mm    =   task2_top3attr(pca_mm,colnames)
top3_attr_random    =   task2_top3attr(pca_random,colnames)
top3_attr_strata    =   task2_top3attr(pca_strata,colnames)
```

- **Task 3: Visualization of both original and 2 types of reduced data**

  - **Task3(i):** Following @app.route("task3") sends data for Scatterplot of top two PCA components where pcomponent is obtained from Task 2(i).

```
#   Task 3(i):  visualize the data projected into the top two PCA vectors via 2D scatterplot
@app.route("/task3")
def get_data_task3():

    data = {"pcomponent_mm":    np.transpose(pcomponent_mm[0:2]).tolist(),
        "pcomponent_random":    np.transpose(pcomponent_random[0:2]).tolist(),
        "pcomponent_strata":    np.transpose(pcomponent_strata[0:2]).tolist()
        }

    return data
```

  - **Task 3(ii):** This calculates the MDS components with Euclidean distance metric using following function:
    def task3_get_MDS_data(dataset,metric_type): It returns Multidimensional Scaling components calculate according to the *metric_type* passed in it i.e. (Euclidean or Correlation).
    And @app.route("/tasks_MDS_Euclidean") sends the data for Scatterplot of two MDS components with Euclidean distance metric.

```
#   Task 3(ii): MDS calculations
#   for Euclidean
metric_type =   'euclidean'
mds_euclidean_mm    =   task3_get_MDS_data(df_mm,metric_type)
mds_euclidean_random    =   task3_get_MDS_data(df_random,metric_type)
mds_euclidean_strata    =   task3_get_MDS_data(df_strata,metric_type)

@app.route("/task3_MDS_Euclidean")
def get_task3_MDS_Euclidean():

    data = {"mds_euclidean_mm":    mds_euclidean_mm,
        "mds_euclidean_random":    mds_euclidean_random,
        "mds_euclidean_strata":    mds_euclidean_strata
        }

    return data
```

  - Similarly, following function calculates the MDS components with Correlation metric using ***def task3_get_MDS_data(dataset,metric_type)*** function with metric_type = "correlation" and sends the data for Scatterplot of two MDS components with Correlation metric.

```
#   for Correlation
metric_type =   'correlation'
mds_correlation_mm    =   task3_get_MDS_data(df_mm,metric_type)
mds_correlation_random    =   task3_get_MDS_data(df_random,metric_type)
mds_correlation_strata    =   task3_get_MDS_data(df_strata,metric_type)

print("org",min(mds_correlation_mm),max(mds_correlation_mm))
print("random",min(mds_correlation_random),max(mds_correlation_random))
print("strata",min(mds_correlation_strata),max(mds_correlation_strata))

@app.route("/task3_MDS_Correlation")
def get_task3_MDS_Correlation():

    data = {"mds_correlation_mm":    mds_correlation_mm,
        "mds_correlation_random":    mds_correlation_random,
        "mds_correlation_strata":    mds_correlation_strata
        }

    return data
```

- **Task3(iii): *def get_top3_pca_atttr_data (dataset, colnames, top3_attr):*** It returns data of top 3 columns out of the datasets which we got from *function task2_top3attr()* which is used to plot the scatterplot matrix.

```
#   Task 3(iii):     visualize the scatterplot matrix of the three highest PCA loaded attributes
top3_attr_data_mm  = get_top3_pca_atttr_data(df_mm,colnames,top3_attr_mm)
top3_attr_data_random  = get_top3_pca_atttr_data(df_random,colnames,top3_attr_random)
top3_attr_data_strata  = get_top3_pca_atttr_data(df_strata,colnames,top3_attr_strata)


@app.route("/task3_ScatterPlotMatrix")
def get_task3_ScatterPlotMatrix():

    data = {"top3_attr_data_mm":    top3_attr_data_mm,
        "top3_attr_data_random":    top3_attr_data_random,
        "top3_attr_data_strata":    top3_attr_data_strata,
        }

    return data
```

- **JavaScripts and their functionalities:**
  - After passing everything from app.py file using “app.route()”, index.html call the javascripts where functions get that data and process it to plots bargraph, line chart, scatter plot and scatterplot matrix.
  - **“/index.html”** calls all the javascripts as follows:

```
<div id="Content" align ="center" style="margin-top: 30;">
  <script src="{{url_for('static', filename='task3_drawScatterPlotMatrix.js')}}"></script>
  <script src="{{url_for('static', filename='task3_drawScatterPlot.js')}}"></script>
  <script src="{{url_for('static', filename='task2_getbarchart.js')}}"></script>
  <script src="{{url_for('static', filename='getdataFromPython.js')}}"></script>
  <script src="{{url_for('static', filename='main.js')}}"></script>
</div>
```

- **"/static/main.js":** This file handles the dropdown functionalities and calls respective javascript functions of other .js files.

```
d3.select("#selectButton")
  .selectAll('myOptions')
  .data(tasks)
  .enter()
  .append('option')
  .text(function (d) { return d; }) // text showed in the menu
  .attr("value", function (d) { return d;})

d3.select("#selectButton").on("change", function(d) {
    taskname = d3.select(this).property("value")

    if (taskname == "Task2: Scree plot and Intrinsic Dimensionality")
      {task2_getBarChartData();}

    else if(taskname  ==  "Task3: Scatterplot PCA Components")
      { task3_getScatterPlotData();}

    else if(taskname  ==  "Task3: Scatterplot MDS (Euclidean)")
      { task3_MDS_Euclidean();}

    else if(taskname  ==  "Task3: Scatterplot MDS (Correlation)")
      { task3_MDS_Correlation();}

    else if(taskname == "Task3: Scatterplot Matrix (Original Data)")
      { task3_ScatterPlotMatrix("ORIGINAL");}
c
    else if(taskname == "Task3: Scatterplot Matrix (Random Sampled Data)")
      { task3_ScatterPlotMatrix("RANDOM");}

    else if(taskname == "Task3: Satterplot Matrix (Stratified Sampled Data)")
      { task3_ScatterPlotMatrix("STRATIFIED");}
})
```

- The names of the ***taskname*** values are pretty intuitive about their functionality. For example: ***Task2: Scree plot and Intrinsic Dimensionality*** drop down ***taskname*** calls some

function ***task2_getBarChartData ()*** which helps in calling other function which draws the Scree plot which is a combination of Bar chart and line chart.

- ***"/static/getdataFromPython.js":*** This is the important function which catches all the python data sent via app.route functions. It catches the data using jQuery code which looks as follows:

```
function task2_getBarChartData(){
    $.get("/task2", function(response){

      d3.select("#cleansheet").remove();
      d3.select("#cleansheet").remove();
      d3.select("#cleansheet").remove();
      drawBarChart(response);
    });
  }

  function task3_getScatterPlotData(){
    $.get("/task3", function(response){

      d3.select("#cleansheet").remove();
      d3.select("#cleansheet").remove();
      d3.select("#cleansheet").remove();
      drawScatterPlot(response.pcomponent_mm,  color_data[0]);
      drawScatterPlot(response.pcomponent_random,  color_data[1]);
      drawScatterPlot(response.pcomponent_strata,  color_data[2]);
    });
  }
```

- Here "***response"*** catches the response from python file i.e list or dictionary or list of dictionary or dictionary of list. ***d3.select("#cleansheet").remove()*** remove whatever is under that tag of this id which is basically svg. And ***drawBarChart()*** actually draws the scree plot.

- Similarly, ***"/static/getdataFromPython.js"*** file contains other functions like ***task3_getScatterPlotData()*** which calls ***drawScatterPlot()***, which basically draws the scatterplot for Task 3(i) and Task 3(ii) i.e. ***task3_MDS_Euclidean()*** and ***task3_MDS_Correlation()*** call ***drawScatterplot()*** for scatterplot matrix.

- ***"/static/getdataFromPython.js"*** also has this function which named as ***task3_ScatterPlotMatrix()*** which calls another function ***drawScatterPlotMatrix()*** according to the dataset type like original, stratified and random, which draws scatterplot matrix for three highest PCA loading attributes of the dataset.

```
function task3_ScatterPlotMatrix(dataset_type){
  $.get("/task3_ScatterPlotMatrix", function(response){

    d3.select("#cleansheet").remove();
    d3.select("#cleansheet").remove();
    d3.select("#cleansheet").remove();

    if (dataset_type  ==  "ORIGINAL")
      {drawScatterPlotMatrix(response.top3_attr_data_mm,  color_data[0]);}
    else if(dataset_type  ==  "RANDOM")
      {drawScatterPlotMatrix(response.top3_attr_data_random,  color_data[1]);}
    else
      {drawScatterPlotMatrix(response.top3_attr_data_strata,  color_data[2]);}
  });
}
```

- **Scree plot using "/static/task2_getbarchart.js":** This file contains drawBarChart() function which draws the scree plot. In that function the following code is used to create the rectangles or bars of the scree plot.

```
//console.log("after pca")
Pca.selectAll("rect")
  .data(function(d) { return d.data_types; })
.enter().append("rect")
  .attr("width", x1.rangeBand())
  .attr("x", function(d) { return x1(d.name); })
  .attr("y", function(d) { return y(d.value); })
  .attr("height", function(d) { return height - y(d.value); })
  .style("fill", function(d) { return color(d.name); });
```

The following piece of code is used to draw a line chart with cumulative explained variance ratio in the ***scree plot***. Similarly, we obtain the line chart for other two types of datasets.

```
svg.append("path")
   .data([data])
   .attr("class", "line1")
   .style("stroke", color(0))
   .style("fill", "None")
   .style("stroke-width", "2px")
   .attr("d",vline)

var vline = d3.svg.line()
     .x(function(d,i){ return x0(i) + 12; })
     .y(function(d){ return y(d.data_types[1].value2); })
```

And similar piece of code is used to draw the intrinsic dimensionality line from 0.75 explained variance ratio value.

- **Scatter plot using "/static/task3_drawScatterPlot.js":** This file contains the function ***drawScatterPlot()*** which draws the scatter plot using following main functionality:

```
svg.selectAll(".dot")
    .data(two_pca_components)
  .enter().append("circle")
    .attr("class", "dot")
    .attr("r", 3.5)
    .attr("cx", xMap)
    .attr("cy", yMap)
    .style("fill", color_data)
    .style("stroke", "black");
```

Class dot draws circles with radius r and center coordinates are computed using xMap and yMAp functions which map the input values with the axis scales.

- **Scatter plot Matrix using "/static/task3_drawScatterPlot.js":** This file contains the function drawScatterPlotMatrix() which draws the scatter plot matrix for three highest PCA load attributes.
  This function has similar functionalities as drawScatterPlot() like using class dot to draw scatter plot points but instead of drawing one cell with draw multiple cells using following functionality:

```
var cell = svg.selectAll(".cell")
    .data(cross(traits, traits))
  .enter().append("g")
    .attr("class", "cell")
    .attr("transform", function(d) { return "translate(" + (n - d.i - 1) * size + "," + d.j * size + ")"; })
    .each(plot);
```

In each iteration, a cell is formed out of total 9 cells using function *plot()* which further draws the single scatterplot points using *class circle* and draws the *frame* of that one plot using class frame appending *rect.*

```
  function plot(p){
    var cell = d3.select(this);

    x.domain(domainByTrait[p.x]);
    y.domain(domainByTrait[p.y]);

    cell.append("rect")
        .attr("class", "frame")
        .attr("x", padding / 2)
        .attr("y", padding / 2)
        .attr("width", size - padding)
        .attr("height", size - padding)
        .style("fill","None")
        .style("stroke","black")
        .style("font-weight","bold")
        .style("text-transform","capitalize")

    cell.selectAll("circle")
        .data(data)
      .enter().append("circle")
        .attr("cx", function(d) { return x(d[p.x]); })
        .attr("cy", function(d) { return y(d[p.y]); })
        .attr("r", 3.5)
        .style("fill", color_data)
        .style("stroke", "black")
  }
}
```

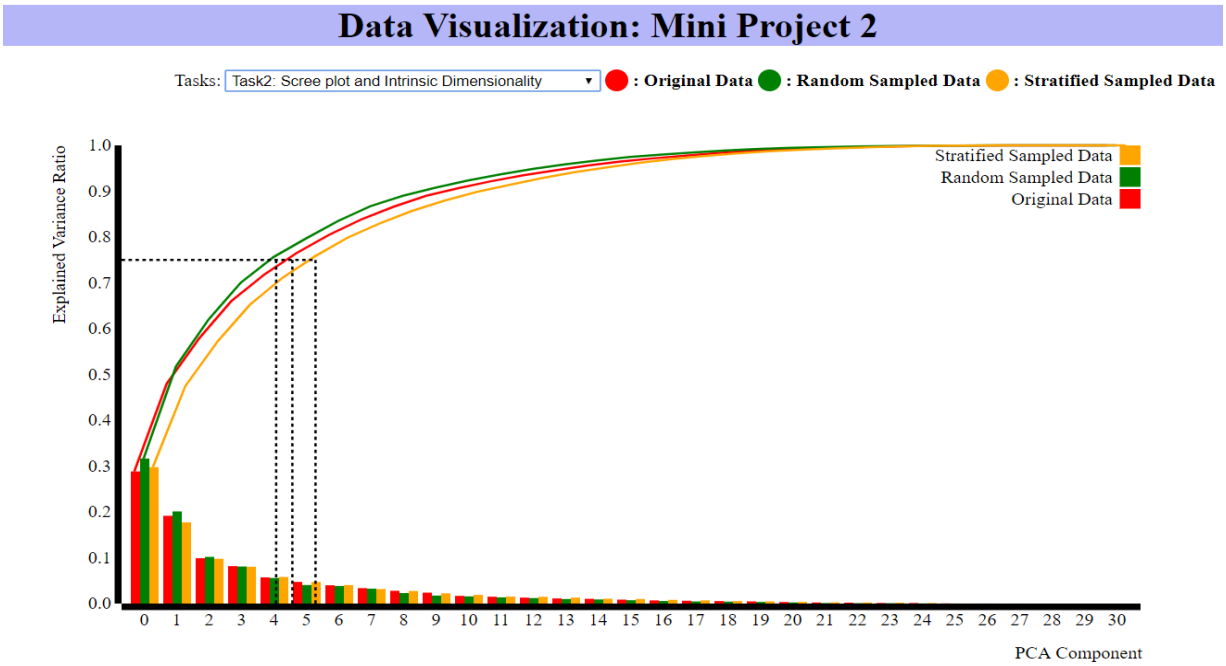
- Also, we have improvised axis drawing functions to draw axes for whole scatter plot. Unlike other single plots we have iterated the class x axis and y axis to draw axes for whole scatter plot.
-

```
svg.selectAll(".x.axis")
    .data(traits)
  .enter().append("g")
    .attr("class", "x axis")
    .attr("transform", function(d, i) { return "translate(" + (n - i - 1) * size + ",0)"; })
    .each(function(d) { x.domain(domainByTrait[d]); d3.select(this).call(xAxis); })

svg.selectAll(".y.axis")
    .data(traits)
  .enter().append("g")
    .attr("class", "y axis")
    .attr("transform", function(d, i) { return "translate(0," + i * size + ")"; })
    .each(function(d) { y.domain(domainByTrait[d]); d3.select(this).call(yAxis); })
```

- **Results and Analysis:**
  - **Scree plot:**

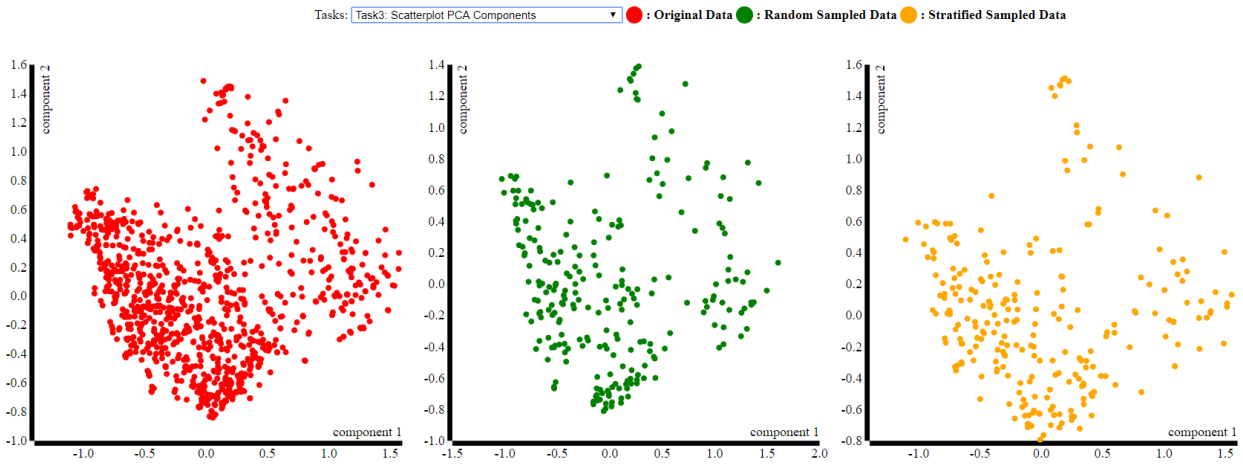
    The dashed line depicts the intrinsic dimensionality from this scree plot. We can analyze from looking this graph that at most 0 to 5 i.e. 6 PCA Components explains the 75% of importance of whole dataset. So, we can reduce our components to 6 from 31 while using the dataset as data modelling or other analysis.



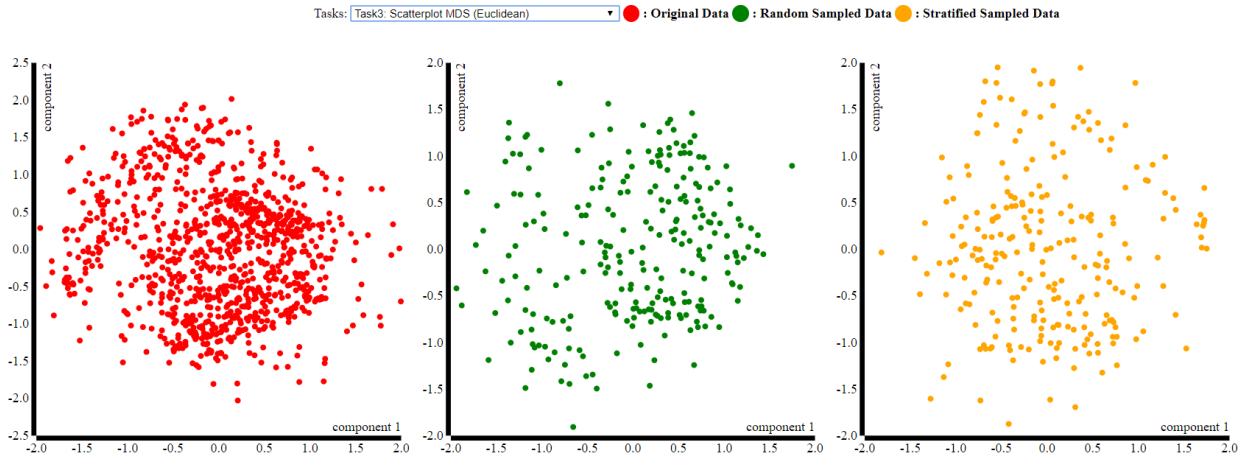
  - **Scatterplot for Top Two PCA Components:**

    We can observe that after taking out top two PCA components, data got spread out properly which indicates the proper explanation of variety of the data after reducing unwanted data dimensions.

    We can also observe some cluster formation which happens in this scenario.



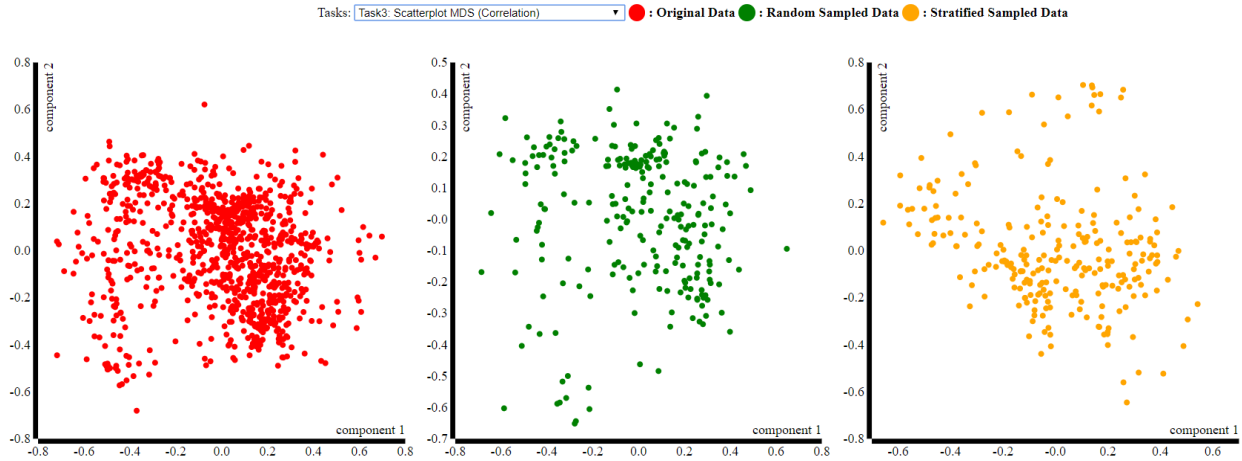
- **Scatterplot for Two MDS Components with Euclidean distance:**



We can analyze from above scatter plot that MDS with Euclidean distance also gives out dispersed scatterplots. For all the different type of datasets, the two components scatter plot follows similar distribution.

There is no cluster formation observed here unlike the scatter plot of PCA Components.

- **Scatterplot for Two MDS Components with Correlation distance:**

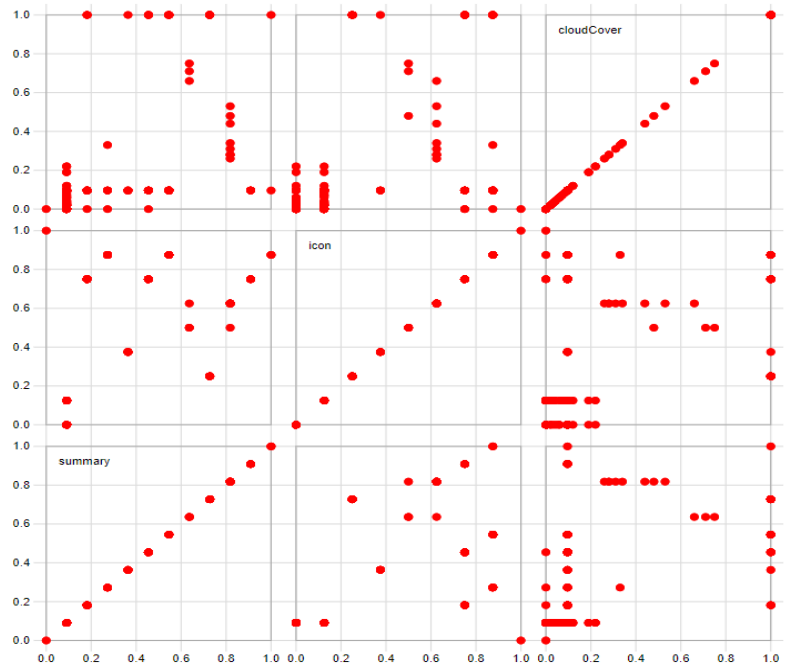


Unlike, other two scatter plots before this one, the points try to follow some distribution which puts them in almost circular dispersion.

Correlation of two components tells about the relation between variability of each other with respect to each other i.e. how strongly or weakly two components are correlated? Stronger the correlation farther the correlation value towards -1 and 1.

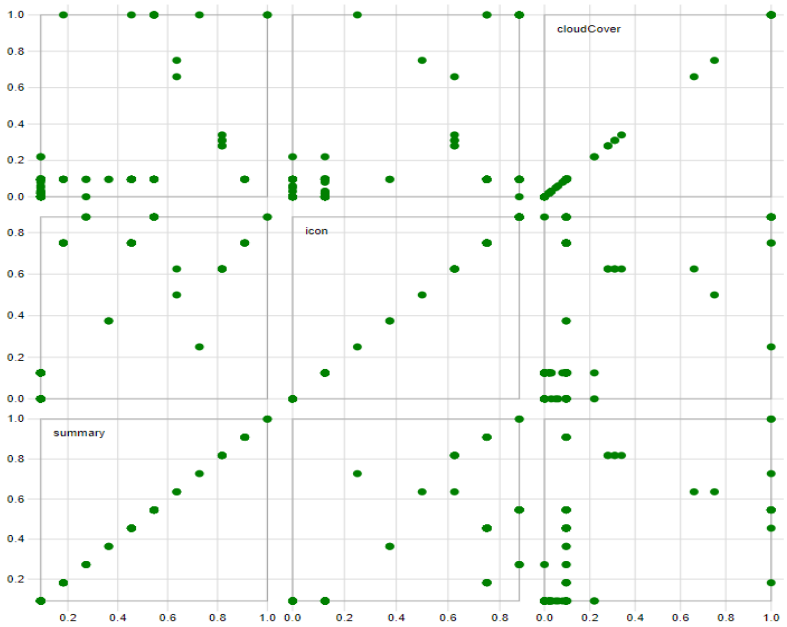
So, we can also observe that x and y values of the components varies from -1 to 1.

- **Scatterplot Matrix for Original Data:**



From Task2(iv) we have generated 3 attributes or columns of our datasets with highest PCA loadings which in case of original dataset are ***cloudCover***, ***icon,*** and ***summary***. It could be possible that due to high variance in these columns with other attributes, as two of them are categorical attributes, the explained variance ratio is huge for these attributes.

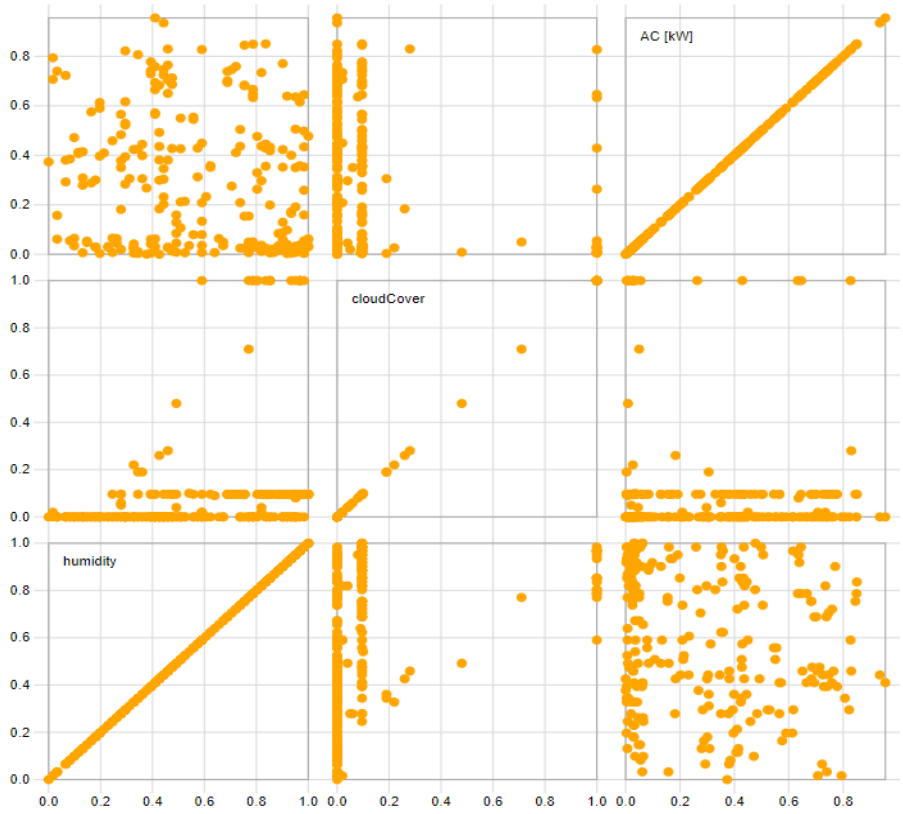
- **Scatterplot Matrix for Random Data:**



Top three attributes for Random Sampled Data is same as for original dataset. This could confirm, up to a limit, that our previous analysis assumption could be true that categorical data varied more with respect to rest of the data.

That's the reason that most of the value points are overlapping on each other as total number of points on plot look lesser than the total count of the points. Most values are equal for these attributes and their MinMax() standardized values distribution between 0 to 1 is almost uniform but varying with larger step values as compare to other attribute like cloudCover thus as compare to other less dispersed and non-uniform columns like cloudCover, their variance is high which results in higher PCA loadings for these attributes in the first place.

- **<u>Scatterplot Matrix for Stratified Data:</u>**



Unlike other two datasets, this has two new attributes with highest PCA loadings which are ***humidity, cloudCover,*** and ***AC [kW].*** This could be explained as follows that as we have clustered and then took out the samples for Stratified dataset, which is nothing but a data reduction technique and which decreased the bias for the categorical attributes unlike other two scatterplot matrices. As it took out the samples from the dataset with consideration of par sampling of all type of clusters.

Conclusion:

I can conclude from this mini project that we have analyzed three type of datasets, obtained from a single dataset against different type of metric systems and functionalities like PCA, MDS with Correlation and Euclidean distance metrics and k means clustering. Using d3.js and python functionalities like flask and other libraries we have developed meaningful graphs and plots like scree plot and scatterplots which helped us in understanding of data and dimension reduction techniques.

References and links:

Youtube link: https://www.youtube.com/watch?v=W6KBtj8sTHE&feature=youtu.be