Name : **Melita Saldanha**
SBU ID : **112551230**

Course : **CSE 564 – Visualization and Visual Analytics**
Mini Project : **2 – World Statistics Visual Data Analytics**

Report

Data Processing:

The dataset consists of various features about each country in the world. Each country had data for the years 1950-2019, and also the difference in values based on gender.

1. **Filtering Data:** Since the dataset consists of around 20,000 rows which takes very long to process during visualization, the data has been filtered to consist of data of each country only for the years 2005 and 2010. We will analyze the data for these 2 years. The data set looks like this for every country:

Country	Year	Gender	Population_Density	Life_Expectancy	……
United States	2005	Male	17.4	65.5	……
United States	2005	Female	17.4	71.2	……
United States	2010	Male	17.6	65.6	……
United States	2010	Female	17.6	71.4	……

2. **Cleaning Individual Datasets:** If a row had more than 5 missing values it was deleted. Since the data is a time-dependent data, for the rest of the rows forward fill and back fill methods were used to propagate non-null values forward or backward.
3. **Label Encoding:** For K-means clustering required in stratified sampling, and for PCA, the datasets should only contain numeric values. Therefore, label encoding was performed on the data for the three categorical columns: Country, Year and Gender.

Code Implementation:

The entire page has been design using HTML, CSS, JavaScript, JQuery and D3.js. Following are its features:

1. **Menu to choose attribute and update chart**
   The top navigation bar is used to choose the attribute that is to be displayed. The attributes have been divided into categories to be easily accessible.

Menu Bar

2. **Sampling of Data**
   The entire dataset has 728 rows and 18 columns. We then sample the data to get 25% of the original data.

   Sampling based on Country:
   Since the data can be considered as a time-series data because we have data for every country for 2 years 2005 and 2010, if we perform sampling on the entire data without condition, for one country we may have data for 2005,

another might have 2010, one might have data for male population and another for females. This sample wouldn't be right as it is not comparable.
Therefore, the data has been sampled based on grouping by country. If a country has been chosen in the sample, the sample will have the data for 2005, 2010's male and female population for that country.

The sampling is done by using 2 methods:

a. **Random Sampling:** Got a set of all countries, randomly shuffled the set, then took 25% of the countries in the set. If a country is present in the sampled set, the 4 rows from the original data corresponding to that country are added to the randomly sampled data list.
b. **Stratified Sampling:** Got a set of all countries, found mean of the four rows for that country from the original data, then sampled 25% of the rows from this set. If a country is present in the sampled set, the 4 rows from the original data corresponding to that country are added to the randomly sampled data list.

```
@app.route('/', methods=['GET', 'POST'])
def getData():

    global data_complete, data_random, data_stratified

    # Original Data
    data = data_read("data/Final_Dataset_Shortened.csv")
    data_complete = label_encoding(data)
    n = len(data_complete)

    # Random Sampling Data
    data_random = random_sampling(25)

    # Stratified Sampling Data
    data_stratified = stratified_sampling(25)
```

Request for sampling of data on body load

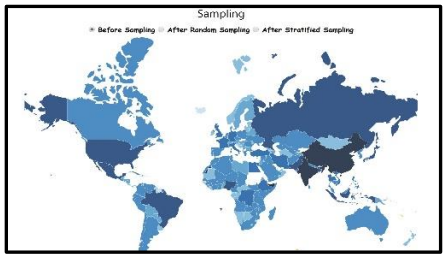
```
def random_sampling(percentage):
    data = pd.DataFrame()
    country_list = list(data_complete.Country.unique())
    sampled_country_list = random.sample(country_list,
        int(len(country_list)*percentage/100))

    for index, row in data_complete.iterrows():
        if(row['Country'] in sampled_country_list):
            data = data.append(row, ignore_index=True)

    return data
```

Random Sampling

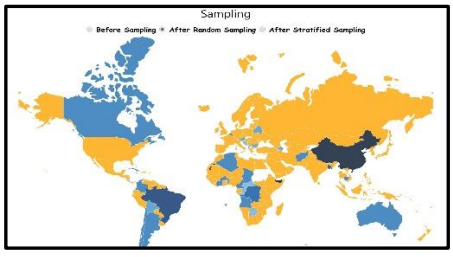
```
def stratified_sampling(percentage):

    # Group by Country
    col_list = data_complete.columns
    df_group = pd.DataFrame(columns=col_list)
    grp = data_complete.groupby('Country')
    for name, group in grp:
      mean = group.mean()
      df_group = df_group.append(pd.DataFrame([[name, mean[0], mean[1], mean[2], mean[3],

    # Stratified Sampling
    data_as_np = (df_group.groupby('Country').mean()).values
    sum_of_square_error_np = np.asarray(generate_elbow_data(data_as_np))

    idxOfBestPoint = get_elbow_point(sum_of_square_error_np, 10)
    optimal_k = int(sum_of_square_error_np[idxOfBestPoint][0])
    kmeans = KMeans(n_clusters = optimal_k, random_state = 0).fit(data_as_np)

    clusters = {}
    for i in range(len(kmeans.labels_)):
        try:
            clusters[kmeans.labels_[i]].append(i)
        except KeyError:
            clusters[kmeans.labels_[i]] = [i]

    stratified_samples = []
    for cluster in clusters.values():
        cluster = shuffle(cluster, n_samples=int((len(cluster)*percentage)/100),
                                    random_state = 0)
        stratified_samples += [data_as_np[i] for i in cluster]

    # Final DF based on Country Names sampled
    temp_df = pd.DataFrame(stratified_samples)
    new_df = pd.DataFrame()
    for i in range(len(temp_df)):
        new_df = new_df.append(data_complete[data_complete['Country'] == temp_df[0][i]],
                               ignore_index=True)

    return new_df
```

Stratified Sampling

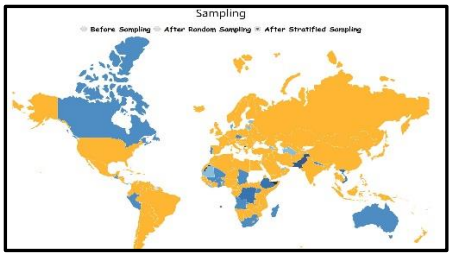
Given below if a map of all the countries. In the graph for random and stratified sampling, the countries marked with shades of blue are a part of the sampled data, and the countries marked with orange have not been considered in the sample.



Original Data



Random Sampled Data



Stratified Sample Data

3. Elbow Plot for Stratified Sampling

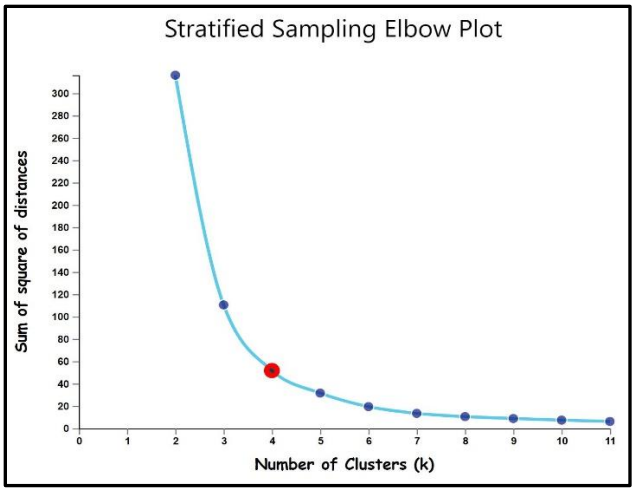
To perform stratified sampling, we need to divide the entire data into clusters using K-means clustering algorithm. For this, we need to know the optimal value of k i.e. the optimal number of clusters. In order to do this, we make clusters 2 to 11, and check the sum of square of the distances which gives us a plot which looks like an arm. The point which looks like an elbow in the arm i.e. the point after which the curve begins to flatten is the optimal k (number of clusters).

```
$.ajax({
  url: "http://127.0.0.1:5000/getElbowData",
  type: 'GET',
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  crossDomain: true,
  success: function(dt) {

    // Close spinner
    document.getElementById("spinner").style.display="none";

    // Data Received
    data = dt.elbow_data;
    knee = dt.knee;
    n = data.length;
```

Call to python function to get data

```
def generate_elbow_data(data_as_np):

    sum_of_square_dist = []
    for i in range(2, 12):
        kmeans = KMeans(n_clusters = i, random_state = 0).fit(data_as_np)
        sum_of_square_dist.append((i, kmeans.inertia_/10000000))

    return sum_of_square_dist
```

Finding sum of square of distances



Elbow Plot

4. Scree Plot for reduced data

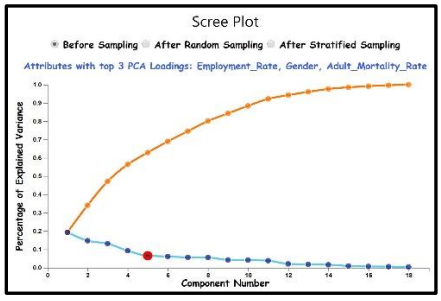
This plot has the percentage of explained variance for each column of the data beginning from the best attribute shown in blue. It also has the cumulative variance shown in orange. The point marked in red is the intrinsic dimensionality which shows the optimum number of dimensions. It also prints the best three attributes for each of each data sample.

```
def generate_cumulative_data(pca):
    cumulative = []
    cum_var = np.cumsum(pca.explained_variance_ratio_)
    for i in range(0, len(cum_var)):
        cumulative.append([i, cum_var[i]])
    return cumulative


def gen_pca_variance_ratio_data(pca):
    evr = pca.explained_variance_ratio_[:18]
    coors = [[i+1, evr[i]] for i in range(len(evr))]
    return coors
```

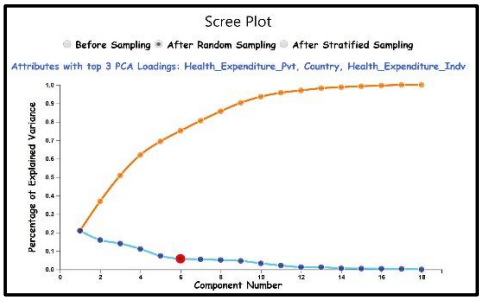
Find variance and cumulative variance

```
def find_intrinsic_dim(pca):
    # uses elbow method
    variance_ratio = pca.explained_variance_ratio_[:18]
    coors_for_elbow = np.asarray([[i+1, variance_ratio[i]] for i in range(18)])
    intrinsic_dim_idx = get_elbow_point(coors_for_elbow, 18)
    return intrinsic_dim_idx
```

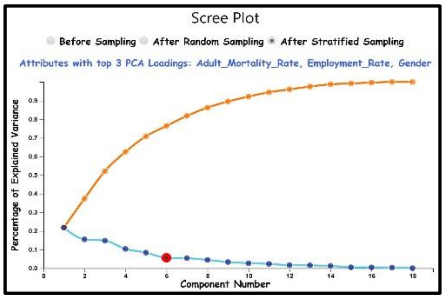
Find intrinsic dimensionality



Plot for Original Data



Plot for Random Sample



Plot for Stratified Sample

5. Dimensions after Reduction

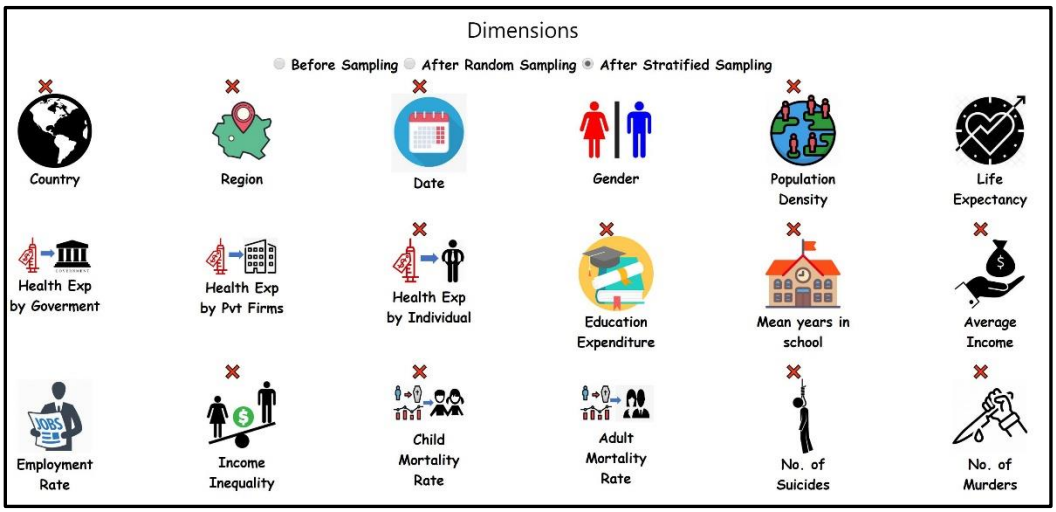
Based on the intrinsic dimensionality we get optimal number of dimensions. We get the best k attributes where k is the intrinsic dimensionality. This view shows all the dimensions of the data. The dimensions mentioned with a cross above them are to be removed, and the dimensions without the cross are the dimensions after reduction.

```
@app.route('/getDimensions', methods=['GET', 'POST'])
def getDimensions():

    data = getDataForRadioButton((request.get_json())['radioValue'])

    pca = compute_pca(data.values)
    intrinsic_dim = find_intrinsic_dim(pca)
    pca_loadings = compute_pca_loadings(pca)
    selected_dimensions = compute_top_k_pca_loadings(pca_loadings, intrinsic_dim+1)

    return jsonify({"dimensions":selected_dimensions})
```

Python Code to get the dimensions after reduction

```
// Display cross for attributes not present
node = $(".col");
for(i=0; i<node.length; i++) {
  id = node[i].id;

  if(dimensions.includes(id))
    document.querySelector("#"+id).querySelector(".cross").style.display = "none";
  else
    document.querySelector("#"+id).querySelector(".cross").style.display = "block";
}
```

JS Code to display a cross on the reduced dimensions



Dimensions after reduction on stratified sample

6. Top 2 PCA vectors in a 2D scatter plot

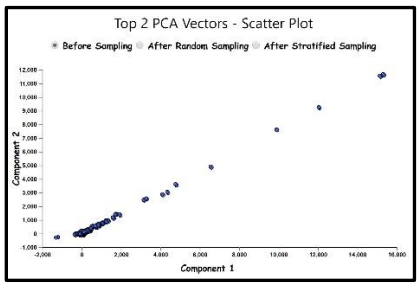
This gives a 2D scatter plot of the top 2 attributes of PCA. We compare the scatter plot for original data, random sample data and stratified sample data.

```
// PCA 2D - Scatter Plot
function PCA_Top2(radioValue) {

  $.ajax({
    url: "http://127.0.0.1:5000/PCA_Top2",
    type: 'POST',
    data: JSON.stringify({radioValue}),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    crossDomain: true,
    success: function(dt) {

      // Data Received
      data = dt.pca_scatter_data;

      // Draw 2D Scatter Plot
      drawScatterPlot(data);
    }
  });

}
```

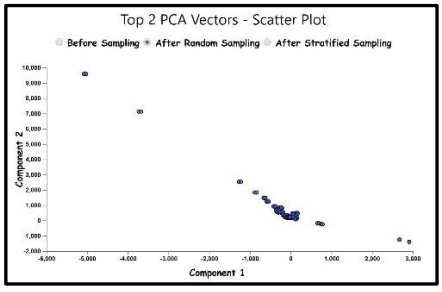
Ajax call to the python function

```
@app.route('/PCA_Top2', methods=['GET', 'POST'])
def PCA_Top2():

    data = getDataForRadioButton((request.get_json())['radioValue'])

    pca = compute_pca(data.values)
    pca_loadings = compute_pca_loadings(pca)
    top2_pca_vectors = np.delete(np.asarray(pca_loadings), [2], 1)
    coors = get_coors_pca_scatterPlot(data.values, top2_pca_vectors)
    coors_l = [{'x':i[0], 'y':i[1]} for i in coors]

    return jsonify({"pca_scatter_data":coors_l})
```

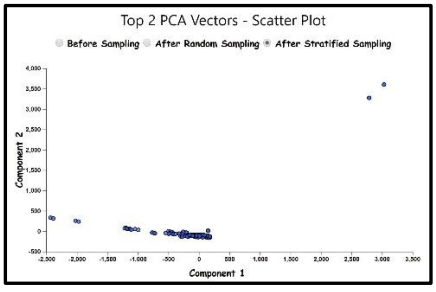
Python function to get scatterplot co-ordinates



Plot for Original Data



Plot of Random Sample



Plot of Stratified Sample

7. MDS using Correlation Distance and Euclidean Distance

This plot shows the difference between MDS using Correlation distance and Euclidean distance of original data, random sample and stratified sample. The original data and stratified sample look similar just with a smaller number of points in stratified sample, while random sample has a random plot which does not resemble the original data.

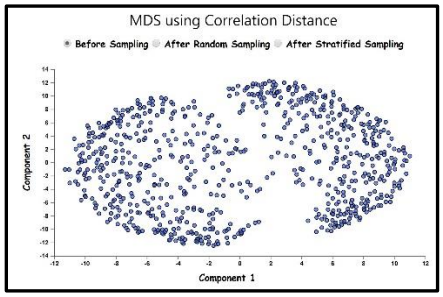
```
// MDS - Scatter Plot
function MDS(radioValue, currentTabName) {

  // Display Spinner
  document.getElementById("spinner").style.display = "block";

  $.ajax({
    url: "http://127.0.0.1:5000/MDS_Distance",
    type: 'POST',
    data: JSON.stringify({radioValue, currentTabName}),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    crossDomain: true,
    success: function(dt) {

      // Close Spinner
      document.getElementById("spinner").style.display = "none";

      // Data Received
      data = dt.data;

      // Draw 2D Scatter Plot
      drawScatterPlot(data);
    }
  });

}
```

Ajax call to get MDS distance from Python

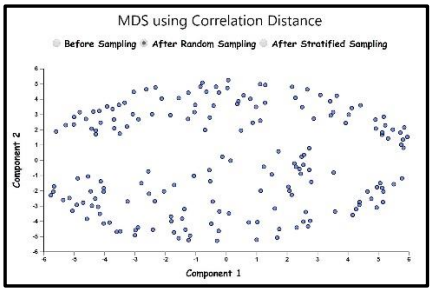
```
def get_dist(data_values, distance):
    df_norm = preprocessing.normalize(data_values)

    scaler = preprocessing.StandardScaler()
    df_scaled = scaler.fit_transform(data_values)

    mds = MDS(n_components=2)

    if(distance=="Correlation"):
        distance = np.corrcoef(df_scaled)
    else:
        distance = euclidean_distances(df_scaled)

    m = mds.fit_transform(distance)
    x = m[:, 0]
    y = m[:, 1]
    coors=[[x[i], y[i]]for i in range(len(x))]
    return coors
```

Python code to find distance

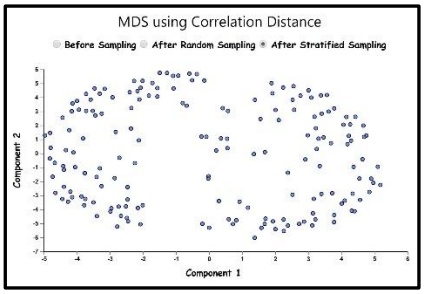
Correlation Distance



Plot for Original Data

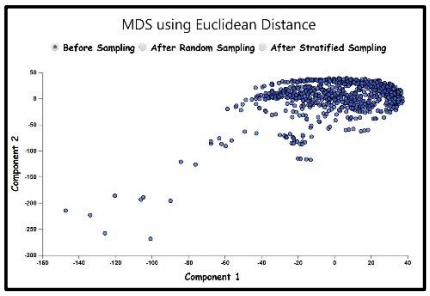


Plot of Random Sample

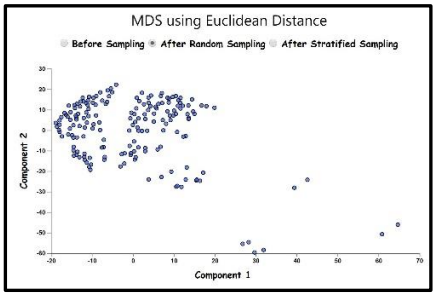


Plot of Stratified Sample

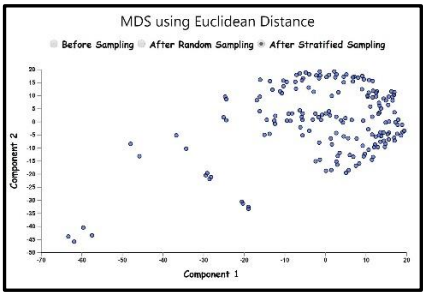
Euclidean Distance



Plot for Original Data



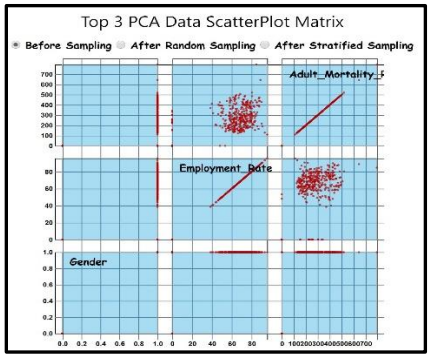
Plot of Random Sample



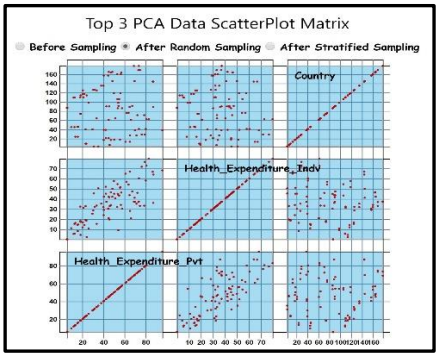
Plot of Stratified Sample

8. Scatterplot matrix of 3 highest PCA loaded attributes

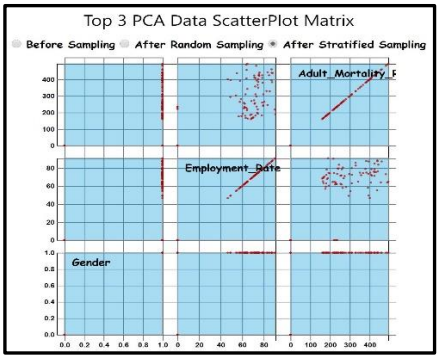
This plot shows a scatterplot matrix of the best 3 attributes. The original data and stratified sample look similar just with a smaller number of points in stratified sample, while random sample has a random plot which does not resemble the original data.



Plot for Original Data



Plot of Random Sample



Plot of Stratified Sample