MINI PROJECT II

I changed the dataset for project 2. The data set I have used is Red Wine Quality Dataset. The dataset has 12 attributes out of which 1 is categorical and remaining are numerical. I did label encoding to the categorical dataset and made it a dataset of numerical values.

Dataset: Red Wine Quality Dataset
Number of rows: 1000 (Took a subset from original 6400 as MDS takes a lot of time)
Number of attributes: 13 (11 numerical, 1 categorical)

Data related to red and white variants of the Portuguese "Vinho Verde" wine. The attributes of the Wine, based on physicochemical tests, are given in each row.

Attributes are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, quality (score between 0 and 10)

Task 1: data clustering and decimation (30 points)

Implement random sampling and stratified sampling (remove 75% of data)

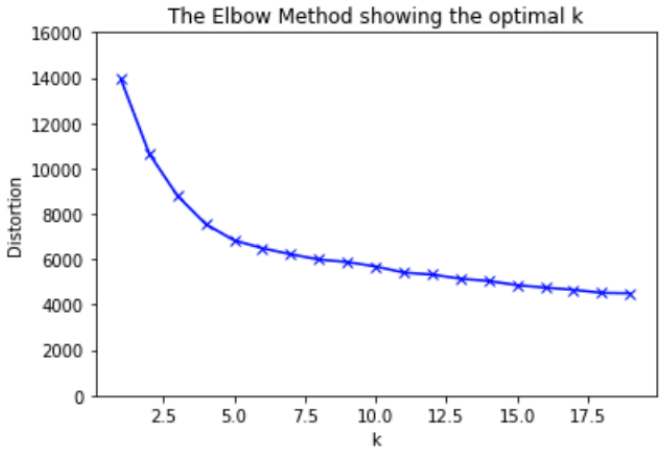
Random Sampling

I did random sampling using .sample function of the python dataframe. As I had to reduce 75% of dataset, I used the fraction to retain as 0.25

```
random_sample = data.sample(frac=0.25)
```

Stratified Sampling

For stratified sampling, I first plotted number of clusters vs total_error and found out the elbow point to find out the apt value for number of clusters. I did data normalization before computing the clusters so that distances are not skewed towards a single attribute.



It turns out to be 5. So, I made 5 clusters and extracted 25% of values from each of the clusters. After these steps, I obtained ~250 rows from the original 1000 rows.

```
# Add cluster to the df
kmeanModel = KMeans(n_clusters=5).fit(X)
kmeanModel.fit(X)
data['cluster'] = kmeanModel.fit_predict(X)
cluster_sample = data
cluster_sample = cluster_sample.groupby('cluster').apply(lambda x: x.sample(frac=0.25))
```

Task 2: dimension reduction on both org and 2 types of reduced data (30)

Intrinsic dimensionality of the data using PCA

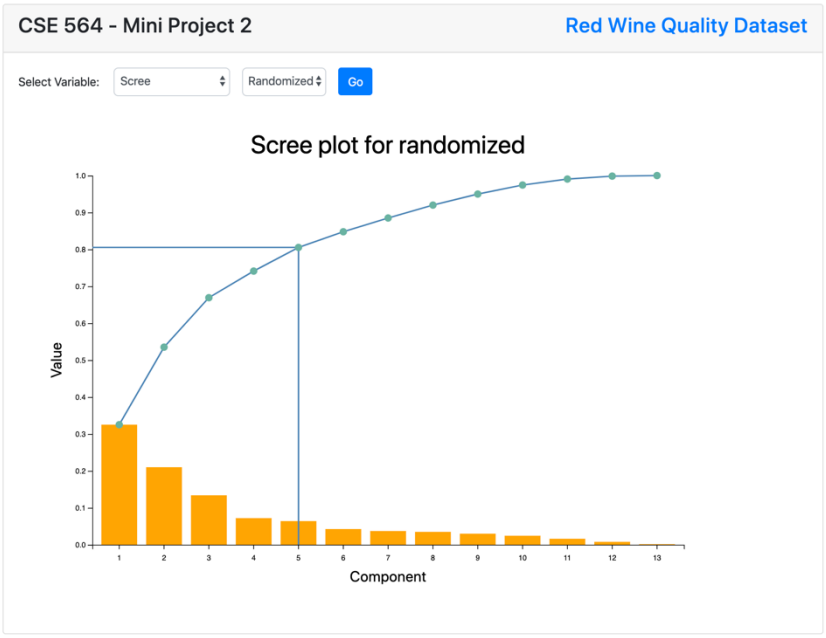
Answer: 5

I found the intrinsic dimensionality of the dataset using two methods – elbow and marking 75% of cumulative sum of eigen values of top PCA components. The following code shows how the PCA component calculation is done.

```
def do_pca(data, n_components):
    ''' Get PCA scree plot ka data '''
    my_model = PCA(n_components=n_components)
    my_model.fit_transform(StandardScaler().fit_transform(data))
    data = []
    for x in range(len(my_model.explained_variance_ratio_)):
        data.append({"x":x+1, "y":my_model.explained_variance_ratio_[x],
        "z":my_model.explained_variance_ratio_.cumsum()[x]})
    return data
```

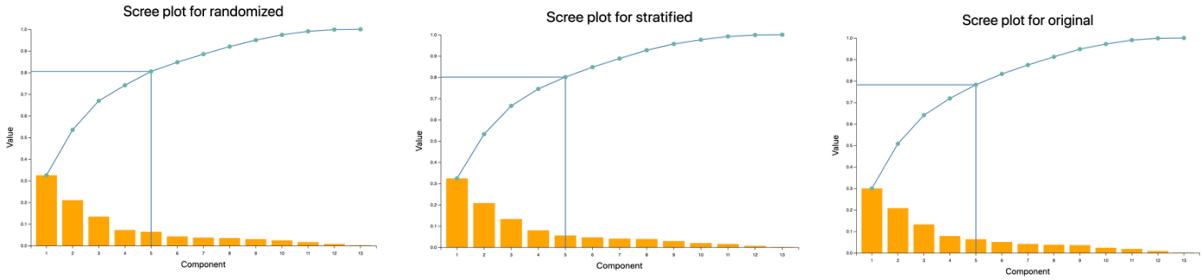
Scree plot visualization and mark the intrinsic dimensionality

I used bar charts in the d3 (similar to assignment 1) to plot the scree plot of the dataset. For marking the intrinsic dimensionality in the scree plot, I used the PCA component till which we capture at least 75% of the information and then used line plots to mark the intrinsic dimensionality. This is used as elbow doesn't seem to capture much of the information and there was lot of variation in top 3 PCA loaded attributes.



Scree plots before/after sampling to assess the bias introduced

We see that the number of components that capture 75% of the variance are same in all the 3 scree plots. Hence the **bias introduced is not that high**. Moreover, the plots for MDS, PCA and scatterplot matrix are similar. So there is not much of a bias.



The three attributes with highest PCA loadings

The top PCA loaded attributes are obtained by using the squared sum of loadings of the pca components on each of the attributes. The top three attributes are**: Residual Sugar, Total Sulfur Dioxide, Free Sulfur Dioxide**

```
def scatter_plot_matrix(inp):
    pca = PCA(n_components=5)
    pca.fit(inp)
    loadings = np.sum(np.square(pca.components_), axis = 0)
    data = []
    columns = inp.columns[[loadings.argsort()[-3:][::-1]]]
    temp = inp[columns]
    temp = temp[(np.abs(stats.zscore(temp)) < 3).all(axis=1)]
    print(temp)
    for i in range(len(temp[columns[0]])):
        try:
            data.append({columns[0]:temp[columns[0]][i],
            columns[1]: temp[columns[1]][i], columns[2]: temp[columns[2]][i]})
        except:
            pass
    return data
```

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

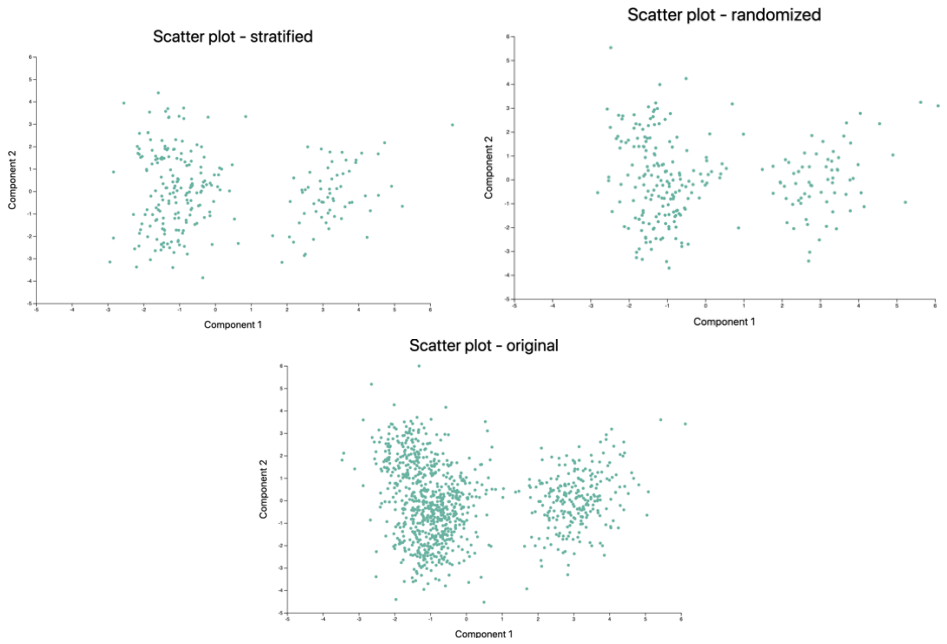
Data projected into the top two PCA vectors via 2D scatterplot

I computed the top 2 pca vectors using the PCA function provided by the sklearn library. The top 2 pca vectors thus obtained, are sent to the front end of the web application using flask and the the plot is rendered using d3. **The plots obtained for all 3 sets of data samples are similar suggesting less bias.**

```
def do_pca_2(data, n_components = 2):
    ''' Get PCA scatter plot ka data '''
    my_model = PCA(n_components=n_components)
    X_PCA = my_model.fit_transform(StandardScaler().fit_transform(data))
    data = []
    Xax, Yax = X_PCA[:,0], X_PCA[:,1]
    for x in range(len(Xax)):
        data.append({"x":Xax[x], "y":Yax[x]})
    return data
```

The code for redirection to the webpage is shown below

```
@app.route("/pca/<sample>", methods = ['POST', 'GET'])
def pca(sample):
    if request.method == 'POST':
        return redirect("/" + request.form['selectControl'] + "/" + request.form['selectControl1'])
    data = do_pca_2(getSample(sample), len(getSample(sample).columns))
    meta_data = {"x_label": "Component 1", "y_label": "Component 2", "title": "Scatter plot - " + sample}
    return render_template('scatter.html', data=data, meta_data = meta_data, selected=["pca", sample])
```

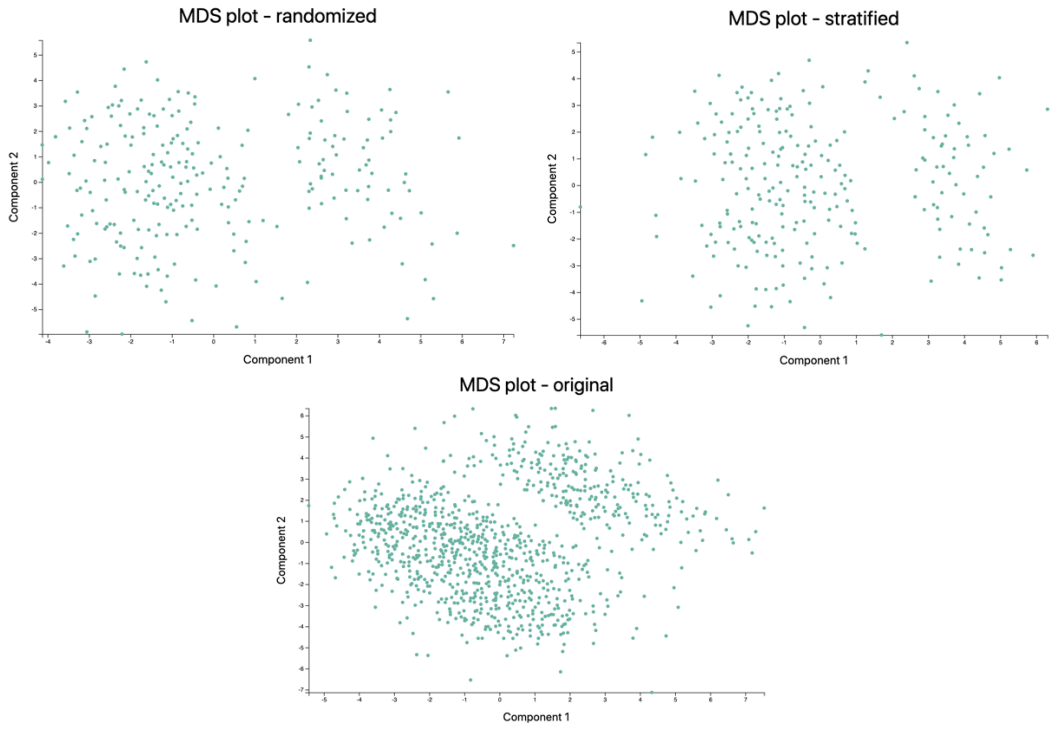


Data via MDS (Euclidian & correlation distance) in 2D scatterplots

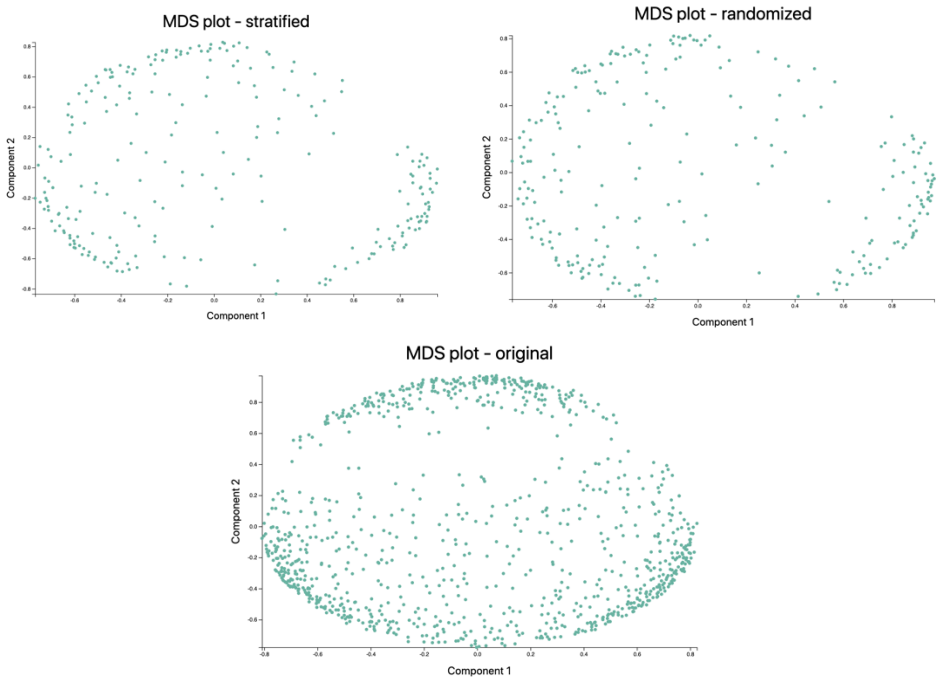
Multidimensional scaling (MDS) is used for visualizing the level of similarity of individual cases of a dataset by translating information about the pairwise 'distances' among a set of n objects or individuals" into a configuration of n points mapped into an abstract Cartesian space. **The plots obtained for all 3 sets of data samples are similar suggesting less bias.**

```
def mds(inp, met):
    distances = pairwise_distances(StandardScaler().fit_transform(inp), metric=met)
    data = []
    print(inp)
    mds = MDS(n_components=2, dissimilarity="precomputed", random_state=23423)
    d = mds.fit_transform(distances)
    d = d[(np.abs(stats.zscore(d)) < 3).all(axis=1)]
    for p in d:
        data.append({"x":p[0], "y":p[1]})
    return data
```

I used **pairwise_distances** function provided by **sklearn.metrics** to get the pairwise distances between the data points. The metric used for computing the distances is selected as Euclidean/ correlation based on the user input. Since MDS takes a lot of time to run, I precomputed the values and stored them at the startup of the server.



Above are MDS plots with Euclidean Distance



Above are MDS plots with correlation distance

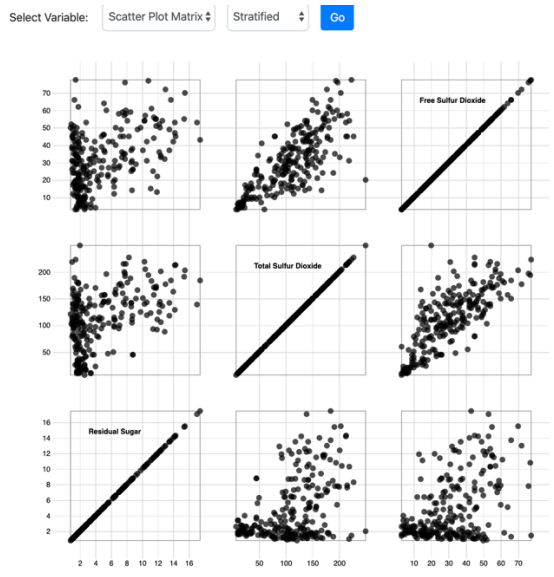
Scatterplot matrix of the three highest PCA loaded attributes

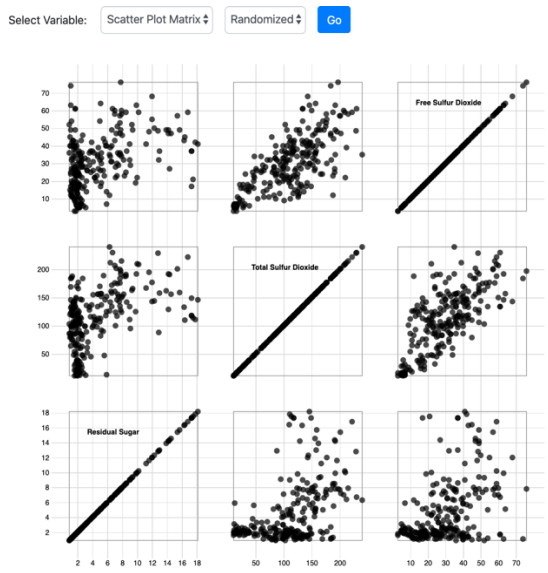
I obtained the top pca loaded attributes by using the squared sum of top pca loadings on each of the attributes and obtained the top three PCA loaded attributes. **The plots obtained for all 3 sets of data samples are similar suggesting less bias.**

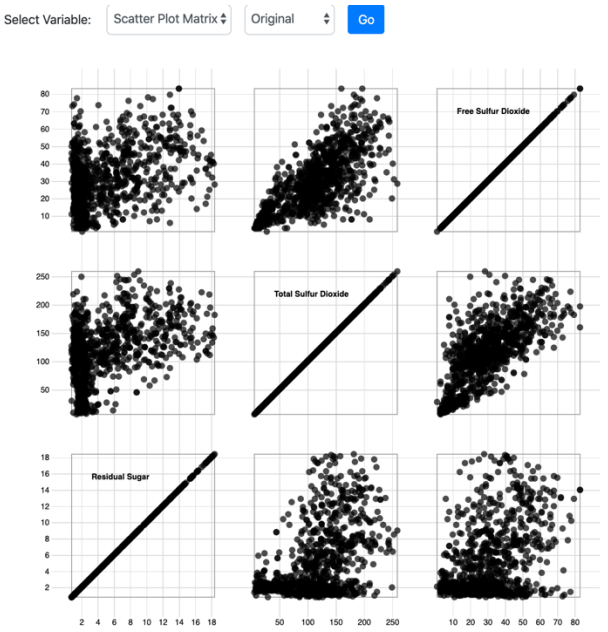
```
    pca = PCA(n_components=5)
    pca.fit(inp)
    loadings = np.sum(np.square(pca.components_), axis = 0)
    data = []
    columns = inp.columns[[loadings.argsort()[-3:][::-1]]]
    temp = inp[columns]
    temp = temp[(np.abs(stats.zscore(temp)) < 3).all(axis=1)]
```

The data is fetched whenever /scattermatrix/<sample> endpoint is hit and the <sample> is used to determine which kind of sampling is to be used. (**Residual Sugar, Total Sulfur Dioxide, Free Sulfur Dioxide**
)

```
@app.route("/scattermatrix/<sample>", methods = ['POST', 'GET'])
def scatterplot(sample):
    if request.method == 'POST':
        print(sample)
        # if request.form['data'] == 'received';
        return redirect("/" + request.form['selectControl'] + "/" + request.form['selectControl1'])
    data = scatter_plot_matrix(getSample(sample))
    return render_template('scatter-matrix.html', data=data, selected=["scattermatrix", sample])
```






Select Variable:
Scatter Plot Matrix
Original
Go
Free Sulfur Dioxide
Total Sulfur Dioxide
Residual Sugar