VISUALIZATION MINI PROJECT II

REPORT SAKSHI GUPTA-112552239

DATASET

The dataset that I chose for my assignment is the FIFA 2019 Player Dataset. This dataset contains all the relevant information about European Football players that participated in FIFA World Cup 2019. After combining and filtering columns based on relevance to the assignment, I picked out the final 11 attributes (and 600 rows) which were:

Categorical attributes: Body Type, Height, Preferred foot, Weight.

Numerical attributes: Age, Value, Crossing, Finishing, Dribbling, Sprint Speed, GKDiving.

I have used Python, JavaScript, HTML, CSS, Ajax and d3 version 3 for this assignment. Also, I used flask as the server to host the website.

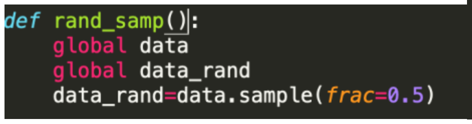
I used Label Encoding instead of One Hot Encoding (initial plan) as the scatter plot matrix was not very appealing and informative in case of One Hot Encoding.

ASSIGNMENT TASKS

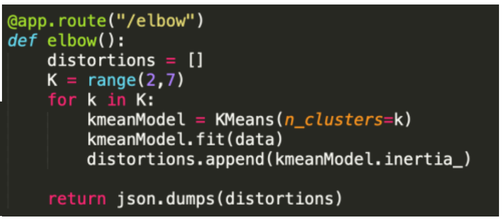
Task 1: Data clustering and decimation

- Implement random sampling and stratified sampling (remove 50% or 75% of data)
- The latter includes the need for k-means clustering (optimize k using elbow)

RANDOM SAMPLING:
I used random.sample() to perform random sampling and I removed 50% of my data by setting frac=0.5.

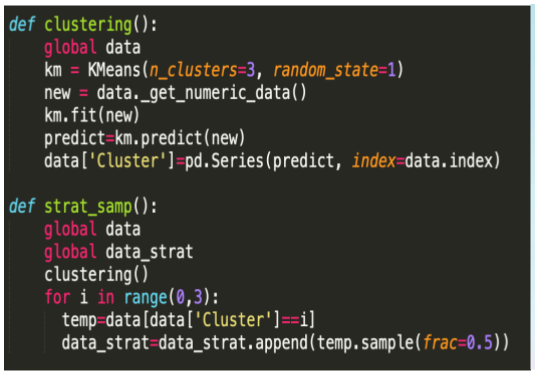


```python
def rand_samp():
    global data
    global data_rand
    data_rand=data.sample(frac=0.5)
```

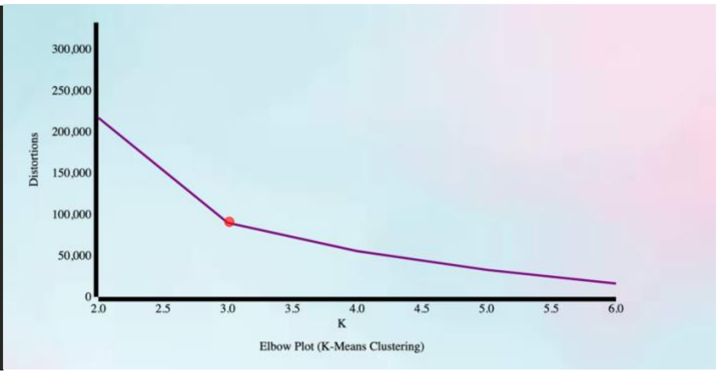


```python
@app.route("/elbow")
def elbow():
    distortions = []
    K = range(2,7)
    for k in K:
        kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(data)
        distortions.append(kmeanModel.inertia_)

    return json.dumps(distortions)
```

STRATIFIED SAMPLING:
First I performed K-Means Clustering (Elbow method) to get the number of clusters for my data and the answer I got was n_clusters=3. Then I grouped the data according to the 3 clusters that were formed and then I sampled 50% of the data from each cluster group. Below are the code snippets and elbow plot.



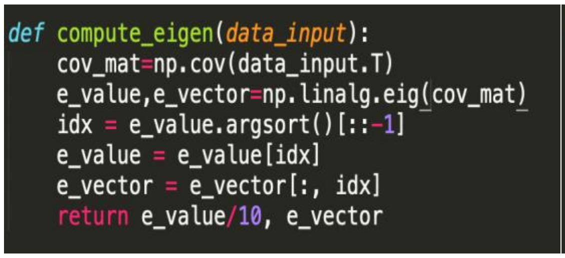
```python
def clustering():
    global data
    km = KMeans(n_clusters=3, random_state=1)
    new = data._get_numeric_data()
    km.fit(new)
    predict=km.predict(new)
    data['Cluster']=pd.Series(predict, index=data.index)

def strat_samp():
    global data
    global data_strat
    clustering()
    for i in range(0,3):
      temp=data[data['Cluster']==i]
      data_strat=data_strat.append(temp.sample(frac=0.5))
```



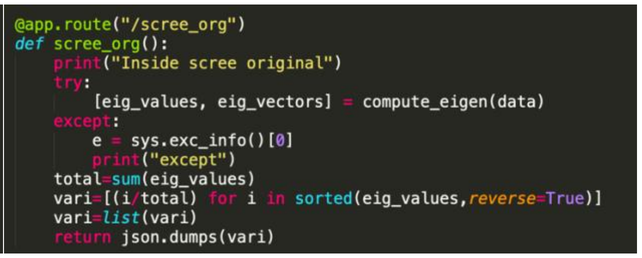
Task 2: Dimension reduction on both org and 2 types of reduced data

- Find the intrinsic dimensionality of the data using PCA
- Produce scree plot visualization and mark the intrinsic dimensionality
- Show the scree plots before/after sampling to assess the bias introduced

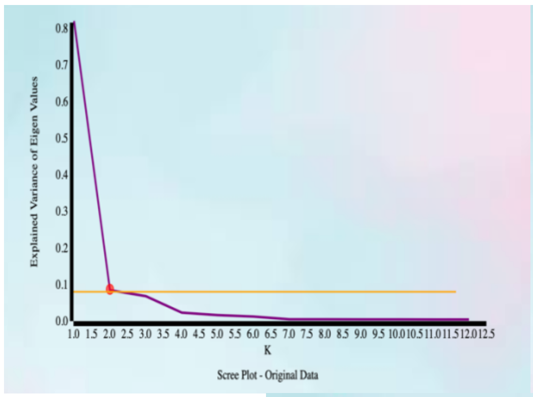
In order to find the intrinsic dimensionality of the data, I computed the eigen values and eigen vectors for both the original and 2 types of reduced data through compute_eigen(). This function returns the eigen values and vectors for the input data by computing the covariance matrix and then uses np.linalg.eig() to create pairs of the values and vectors that gives me a 11X11 matrix in my case. Then we sort the eigen values in order to get the top k dimensions that best describe the data. After this I calculate the explained variance based on the eigen values and send them to my JS file to plot the scree plot. I have just used a line plot to display my scree plot.
Below are the code snippets and the 3 plots- Original data, Data after Random Sampling and Data after Stratified Sampling

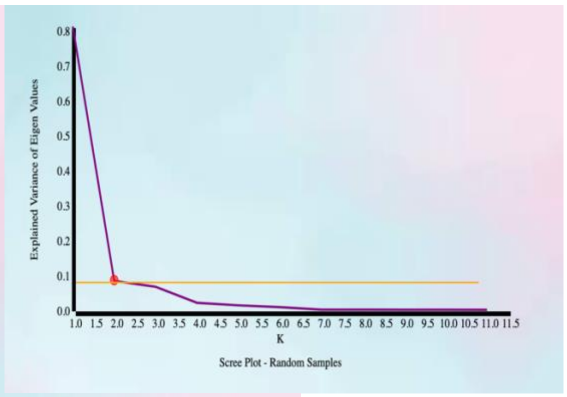


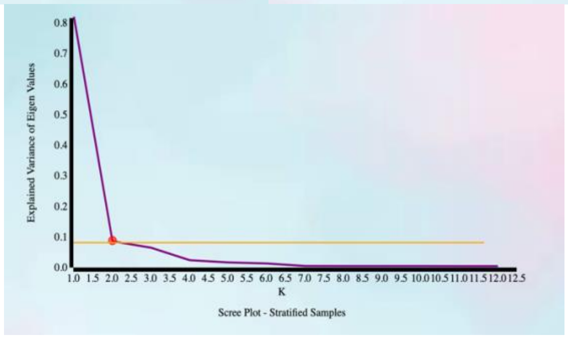
```python
def compute_eigen(data_input):
    cov_mat=np.cov(data_input.T)
    e_value,e_vector=np.linalg.eig(cov_mat)
    idx = e_value.argsort()[::-1]
    e_value = e_value[idx]
    e_vector = e_vector[:, idx]
    return e_value/10, e_vector
```



```python
@app.route("/scree_org")
def scree_org():
    print("Inside scree original")
    try:
        [eig_values, eig_vectors] = compute_eigen(data)
    except:
        e = sys.exc_info()[0]
        print("except")
    total=sum(eig_values)
    vari=[(i/total) for i in sorted(eig_values,reverse=True)]
    vari=list(vari)
    return json.dumps(vari)
```

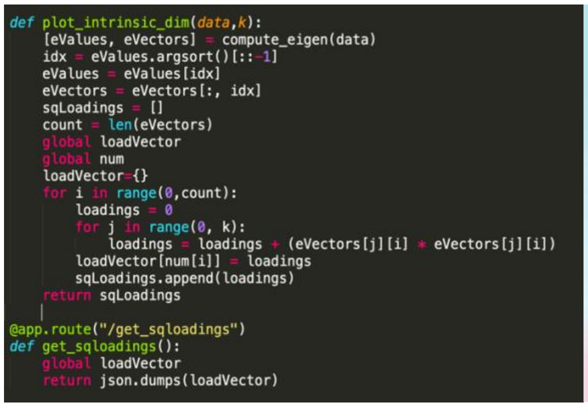




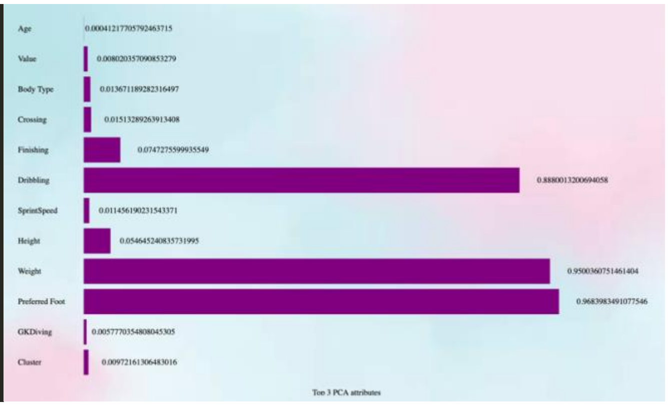


As you can see on actually plotting the three plots, there is very minute difference in the plots and all three plots give the same intrinsic dimensionality of 2. So, my observation is that my data is such that even after performing two types of sampling and taking away 50% of the data, the bias produced is quite less and all the three types of data can be represented just by using the top 2 dimensions.

- Obtain the three attributes with highest PCA loadings
  I used a histogram to plot the PCA Loadings for every column in my dataset and I used the plot_intrinsic_dim() (with n_comp=3) to do so. The initial steps include calculating the eigen pairs as before and then sorting them and then calculating the squared loadings using the square of the eigen vectors. And then I use a load vector to store the squared loadings for each column and then this load vector to d3 to be plotted as a histogram.



```python
def plot_intrinsic_dim(data,k):
    [eValues, eVectors] = compute_eigen(data)
    idx = eValues.argsort()[::-1]
    eValues = eValues[idx]
    eVectors = eVectors[:, idx]
    sqLoadings = []
    count = len(eVectors)
    global loadVector
    global num
    loadVector={}
    for i in range(0,count):
        loadings = 0
        for j in range(0, k):
            loadings = loadings + (eVectors[j][i] * eVectors[j][i])
        loadVector[num[i]] = loadings
        sqLoadings.append(loadings)
    return sqLoadings

@app.route("/get_sqloadings")
def get_sqloadings():
    global loadVector
    return json.dumps(loadVector)
```



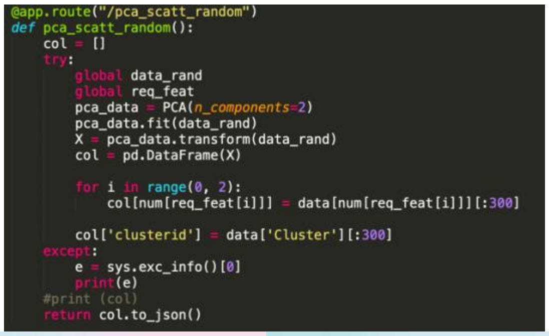
From the plot, we obtain the following top 3 attributes:

1. Preferred Foot
2. Weight
3. Dribbling

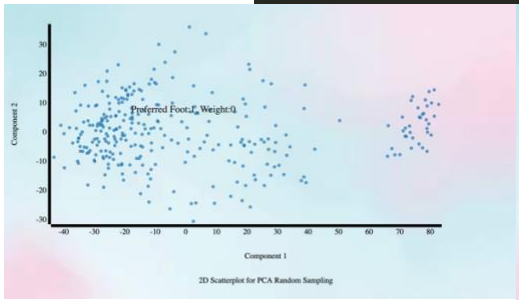
Task 3: Visualization of both original and 2 types of reduced data

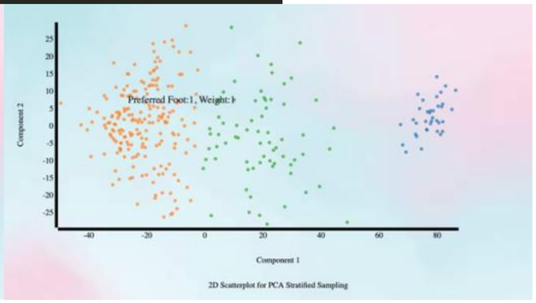
- Visualize the data projected into the top two PCA vectors via 2D scatterplot

For this I first use PCA() and fit the input data according to the model and then create a new dataframe which will store the data points for the top two PCA Vectors (Preferred Foot and Weight in my case) and I also add the clustered column to keep track of the cluster each data point belongs to.



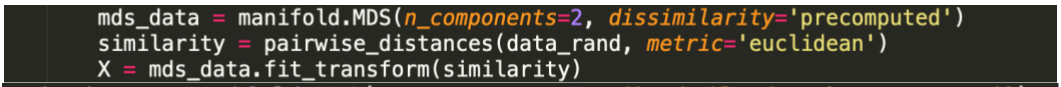
```python
@app.route("/pca_scatt_random")
def pca_scatt_random():
    col = []
    try:
        global data_rand
        global req_feat
        pca_data = PCA(n_components=2)
        pca_data.fit(data_rand)
        X = pca_data.transform(data_rand)
        col = pd.DataFrame(X)

        for i in range(0, 2):
            col[num[req_feat[i]]] = data[num[req_feat[i]]][:300]

        col['clusterid'] = data['Cluster'][:300]
    except:
        e = sys.exc_info()[0]
        print(e)
    #print (col)
    return col.to_json()
```



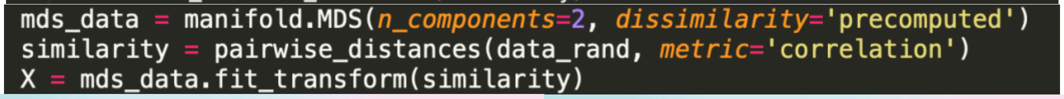


- Visualize the data via MDS (Euclidian & correlation distance) in 2D scatterplots

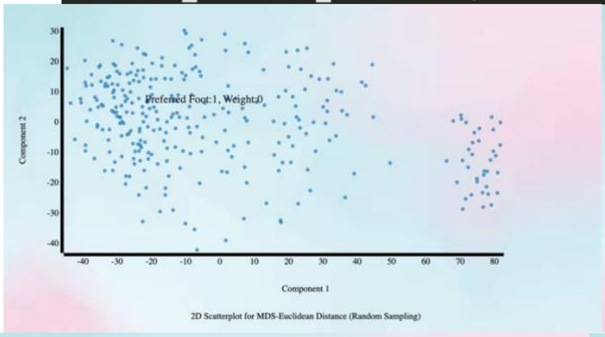
For this I used manifold.MDS() and pairwise_distances with parameters ‘Euclidean’ and ‘Correlation’ and then I fit and transform the input data to this model and the follow the same procedure as before.

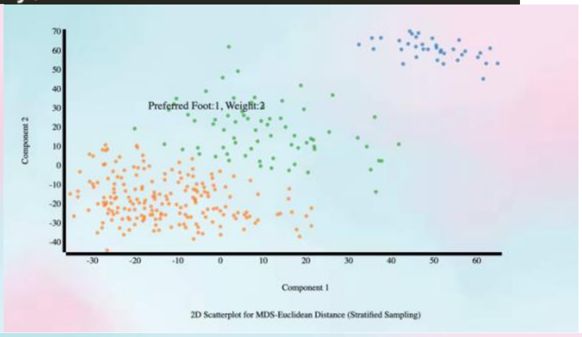


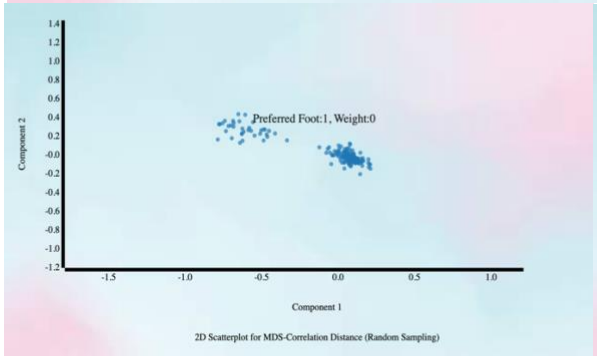
```
mds_data = manifold.MDS(n_components=2, dissimilarity='precomputed')
similarity = pairwise_distances(data_rand, metric='euclidean')
X = mds_data.fit_transform(similarity)
```

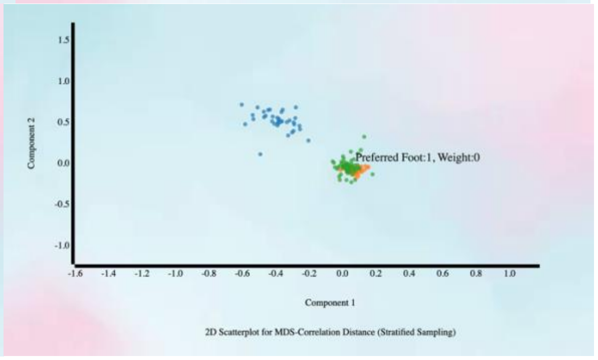


```
mds_data = manifold.MDS(n_components=2, dissimilarity='precomputed')
similarity = pairwise_distances(data_rand, metric='correlation')
X = mds_data.fit_transform(similarity)
```



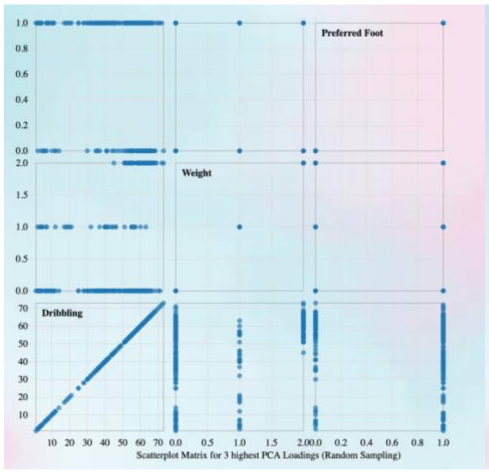


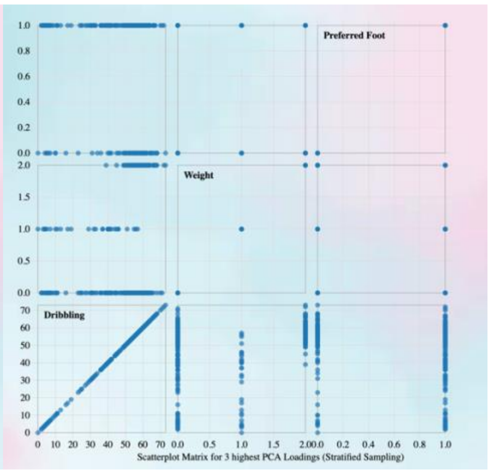




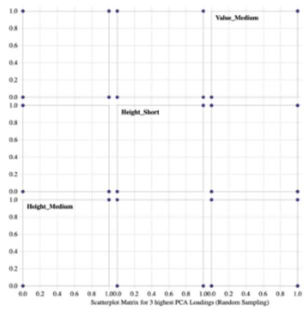
- Visualize the scatterplot matrix of the three highest PCA loaded attributes

For this I did not use any model fitting but just stored the data points for the top two PCA Vectors and added the clustered column to keep track of the cluster each data point belongs to (like before). In this case since I have used label encoding, my top 3 attributes have values range a little better than One Hot Encoding.





This was the scatter plot matrix when I had performed One Hot Encoding. This happened because my top 3 attributes were all one hot encoded data columns and hence had only 0s and 1s as data points.



OBSERVATIONS AND COMMENTS

- In python, I had displayed the scree plot as both a bar plot and a line plot, but I found it easier to locate the elbow using the line plot and hence I used a line plot for my scree plot.
- Encoding my data using two techniques and then visualizing them using the scatter plot matrix helped me understand my data points and their distribution in the dataset. It made me realize how a visualization can change with a change in encoding technique.
- When I used One hot Encoded data, my top 3 PCA Attributes were very different than the current PCA Attributes. All three of them were one hot encoded columns.
- Also, I was hoping that I would see a significant difference between the scree plots before and after sampling but unfortunately for my data, all the plots come out to be more or less the same and so for me the conclusion is that sampling did not produce that significant a bias and the intrinsic dimensionality remained the same before and after sampling.
- Also, while making the scatter plots I noticed that for random sampling different clusters are not colored as in the case of stratified sampling and this fact puzzled me.
- For displaying the PCA Loadings of the attributes I initially plotted the histogram in python and the graph wasn't looking that great because of huge heights of the bars of the top 3 attributes. So, I decided to plot a vertical histogram which makes it easy to understand and appealing to the eye.
- I also used tooltips in the scatterplots so that I could understand the data points in a particular cluster and also understand why they belong to that particular coordinate in the plot.
- For MDS, I kept the dissimilarity metric as 'precomputed' in both, but as we can see when we change the distance metric, the plot changes drastically which actually tells us new things about our data. So, even a simple choice of a metric can have significant effects on visualization.

YOUTUBE LINK

https://youtu.be/66JoKMiobdY

ZIP FILE CONTENTS

- Index.html (inside templates)
- Index.js (inside static)
- Football_wenco.csv (Label encoded data)
- Football_enco.csv (One hot encoded data)
- Bcd.jpg (inside static)
- Main.py
- Report PDF

REFERENCES

- www.w3schools.com
- www.stackoverflow.com
- www.tutorialsteacher.com
- www.bl.ocks.org
- https://medium.com/analytics-vidhya/guide-to-principal-component-analysis-ab04a8a9c305
- https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/