VISUALIZATION MINI PROJECT-2

https://prathak07.github.io/Visualization-Mini-Project-2/

DATASET: **EUROPEAN SOCCER DATABASE** (PLAYER ATTRIBUTES FOR EUROPEAN PROFESSIONAL FOOTBALL) CONTAINING **10849 ROWS AND 36 COLUMNS**

Architecture of the Project:

1. The data is processed using Python code inside "**src**" folder : **GenVisData.py**
2. The python code process the '**Football_Players.csv**' inside "**data**" folder, which is passed as argument during runtime.
3. After processing it creates processed data in folder "**data/processed/**" containing all the files which can be used for plotting the various graphs using D3.js
4. Each graph in "**index.html**" can be plotted by clicking the respective buttons, and each graph plots random as well as stratified sample data in single plot. **(Random: Green ; Stratified: Red)**

Main Function of Python Code:

```
def main(filename):
    data_frame = pandas.read_csv("../data/"+filename)
    data_frame = data_frame.drop(data_frame.columns[0],axis=1)
    data_frame = data_frame.fillna(value=0)
    random_sample = random_sampling(data_frame,0.05)
    find_k_for_kmean(data_frame)
    # found the elbow at 3 thus, number of clusters to be created are 3
    stratified_sample = stratified_sampling(data_frame,3,0.05)
    col_list = data_pca(random_sample,stratified_sample)
    find_pca2(random_sample,stratified_sample,'pca2.csv')
    find_MDS_euclidean(random_sample,stratified_sample,'mds_euclidean.csv')
    find_MDS_correlation(random_sample,stratified_sample,'mds_correlation.csv')
    find_pca3(random_sample,stratified_sample,'pca3.csv',col_list)


if __name__=='__main__':
    if(len(sys.argv)<2):
        print "Command format: python GenVisData.py <csv_filename>"
    else:
        main(sys.argv[1])
```

FIGURE 1 MAIN FUNCTION OF PYTHON CODE

Task1: data clustering and decimation

```
def random_sampling(data_frame,fraction):
    print "Doing Random Sampling with fraction "+str(fraction)
    rows = random.sample(data_frame.index,(int)(len(data_frame)*fraction))
    # print rows
    # print data_frame.iloc[rows]
    return data_frame.iloc[rows]


def find_k_for_kmean(data_frame):
    print "Getting elbow for k-mean clustering by clustering the data with various different k (1-10)"
    i=1
    dic = []
    while(i<=10):
        km = cluster.KMeans(n_clusters=i)
        km.fit(data_frame)
        dic.append((i,km.inertia_))
        i+=1
    file_name = directory+'kmean.csv'
    print "Creating file "+file_name
    dicDF = pandas.DataFrame(dic)
    dicDF.columns = ['cluster_size','value']
    dicDF.to_csv(file_name,sep=',',index=False)
    print "K by elbow method is "+str(3)


def stratified_sampling(data_frame,no_of_clusters,fraction):
    print "Doing stratified sampling where no. of clusters "+str(no_of_clusters)+" and fraction "+str(fraction)
    kmean = cluster.KMeans(n_clusters=no_of_clusters)
    kmean.fit(data_frame)
    data_frame['kmean'] = kmean.labels_
    # from each cluster get some values
    result_data_frame = pandas.DataFrame([],columns=data_frame.columns.values)
    for i in range(no_of_clusters):
        temp = data_frame.loc[data_frame['kmean']==i]
        length = len(temp.index)
        random_index = random.sample(range(length),int(length*fraction))
        random_data_frame = temp.iloc[random_index]
        result_data_frame = result_data_frame.append(random_data_frame)
    result_data_frame = result_data_frame.drop('kmean',1)
    return result_data_frame
```

FIGURE 2 CODE FOR RANDOM AND STRATIFIED SAMPLING

- ♣ implement random sampling and stratified sampling
- ♣ the latter includes the need for k-means clustering (optimize k using elbow)

Random Sampling and Stratified Sampling is performed using python.
For stratified sampling first value of k is calculated using elbow method as below.

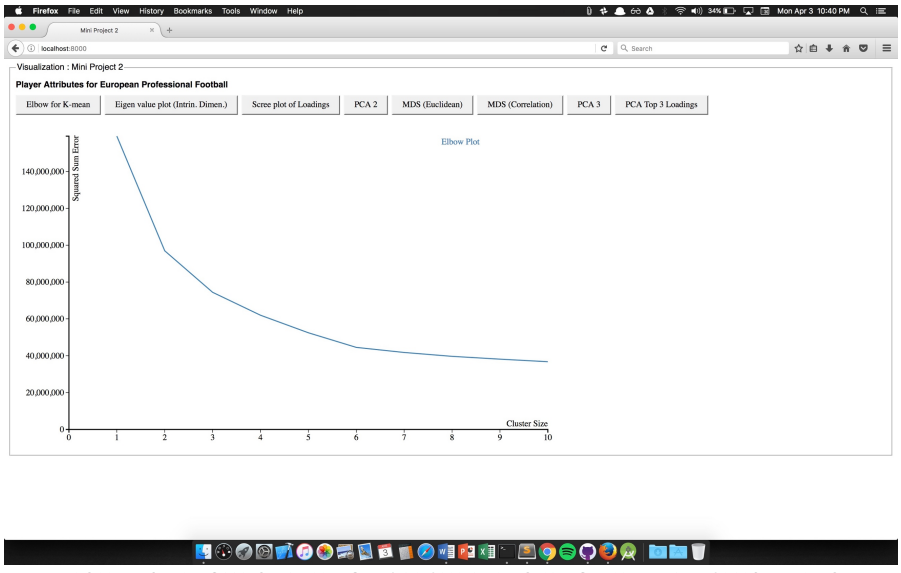

FIGURE 3 ELBOW GRAPH FOR CALCULATING K FOR K-MEAN CLUSTERING

From the above graph, it can be seen the number of clusters to be made is 3. Thus, for stratified sampling number of clusters is passed 3 and fraction of 0.05.

Task 2: dimension reduction (use decimated data)

- ♣ find the intrinsic dimensionality of the data using PCA
- ♣ produce scree plot visualization and mark the intrinsic dimensionality
- ♣ obtain the three attributes with highest PCA loadings

```
def data_pca(df1,df2):
    print "Starting PCA:"
    pca = PCA()
    col_names = df1.columns.values
    print "Random Sampling"
    pca.fit_transform(df1)
    eigen_values_1 = pca.explained_variance_
    top_1 = pca.components_
    file_name = directory+'pca_components_random.csv'
    print "Creating file "+file_name
    topDF = pandas.DataFrame(top_1)
    topDF.to_csv(file_name,sep=',',index=False)
    print "Stratified Sampling"
    pca.fit_transform(df2)
    eigen_values_2 = pca.explained_variance_
    top_2 = pca.components_
    file_name = directory+'pca_components_stratified.csv'
    print "Creating file "+file_name
    topDF = pandas.DataFrame(top_2)
    topDF.to_csv(file_name,sep=',',index=False)
    eigen_tuple_list = []
    for i in range(len(eigen_values_1)):
        eigen_tuple_list.append((i+1,eigen_values_1[i],eigen_values_2[i]))
    file_name = directory+'pca_eigens.csv'
    print "Creating file "+file_name
    tupDF = pandas.DataFrame(eigen_tuple_list)
    tupDF.columns = ['variable','col1','col2']
    tupDF.to_csv(file_name,sep=',',index=False)
    # By observing the Eigen value plot no_of_components is found to be 8
    print "Intrinsic Dimensionality: "+str(8)
    sq_sum_1 = {}
    for i in range(len(top_1[0])):
        sum = 0
        for j in range(8):
            sum += top_1[j][i]**2
        s = col_names[i]
        sq_sum_1[s] = sum
    sq_sum_2 = {}
    for i in range(len(top_2[0])):
        sum = 0
        for j in range(8):
            sum += top_2[j][i]**2
        s = col_names[i]
        sq_sum_2[s] = sum
    tup_list = []
    for i in range(len(col_names)):
        tup_list.append((col_names[i],sq_sum_1[col_names[i]],sq_sum_2[col_names[i]]))
    tup_list = sorted(tup_list, key=lambda x:-1*x[2])
    print "Top 3 PCA Loadings: "+tup_list[0][0]+", "+tup_list[1][0]+' and '+tup_list[2][0]
    top3_col_list = []
    for i in range(3):
        top3_col_list.append(tup_list[i][0])
    file_name = directory+'scree_loadings.csv'
    print "Creating file "+file_name
    tupDF = pandas.DataFrame(tup_list)
    tupDF.columns = ['variable','col1','col2']
    tupDF.to_csv(file_name,sep=',',index=False)
    return top3_col_list
```

FIGURE 4 PYTHON CODE FOR FINDING THE EIGEN VALUES FOR EACH DIMENSION

Above code helps in plotting the Eigen values. Loadings for each of the dimensions of the data is then calculated from it. By checking the Eigen value plot Intrinsic Dimensionality of the data is 8 as at that point the graph becomes flat like x-axis. Below is the graph showing the Eigen Value plot.

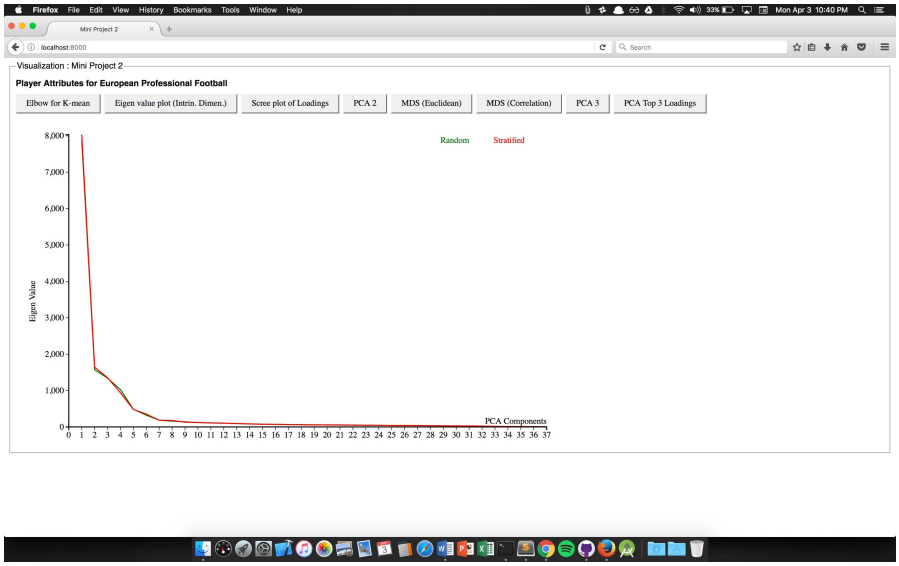

FIGURE 5 EIGEN VALUE PLOT

Below is the graph of Loadings for each of the dimensions of the data, where we can clearly see the first three dimensions corresponds to top 3 loadings.

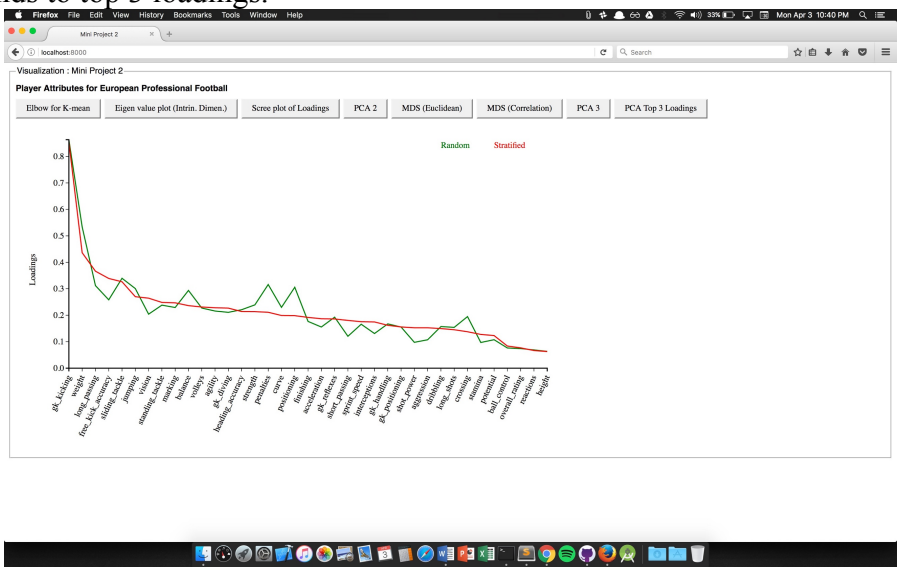

FIGURE 6 SCREE-PLOT SHOWING CONTRIBUTION BY EACH DIMENSION OF THE DATA

Python Code output showing k of k-means and intrinsic dimensionality…

```
(prathakrastogi (master *) src $ python GenVisData.py Football_Players.csv
Doing Random Sampling with fraction 0.05
Getting elbow for k-mean clustering by clustering the data with various different k (1-10)
Creating file ../data/processed/kmean.csv
K by elbow method is 3
Doing stratified sampling where no. of clusters 3 and fraction 0.05
Starting PCA:
Random Sampling
Creating file ../data/processed/pca_components_random.csv
Stratified Sampling
Creating file ../data/processed/pca_components_stratified.csv
Creating file ../data/processed/pca_eigens.csv
Intrinsic Dimensionality: 8
Top 3 PCA Loadings: gk_kicking, weight and long_passing
Creating file ../data/processed/scree_loadings.csv
PCA 2 started
PCA 2 finished
Creating file ../data/processed/pca2.csv
MDS (euclidean) started
MDS (euclidean) finished
Creating file ../data/processed/mds_euclidean.csv
MDS (correlation) started
MDS (correlation) finished
Creating file ../data/processed/mds_correlation.csv
PCA 3 started
PCA 3 finished
Creating file ../data/processed/pca3.csv
Creating file ../data/processed/pca3_loadings.csv
(prathakrastogi (master *) src $ cd ..
(prathakrastogi (master *) VIS Project 2 $ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

FIGURE 7 OUTPUT OF PYTHON CODE

Task 3: visualization (use dimension reduced data)

- ♣ visualize data projected into the top two PCA vectors via 2D scatterplot
- ♣ visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots
- ♣ visualize scatterplot matrix of the three highest PCA loaded attributes

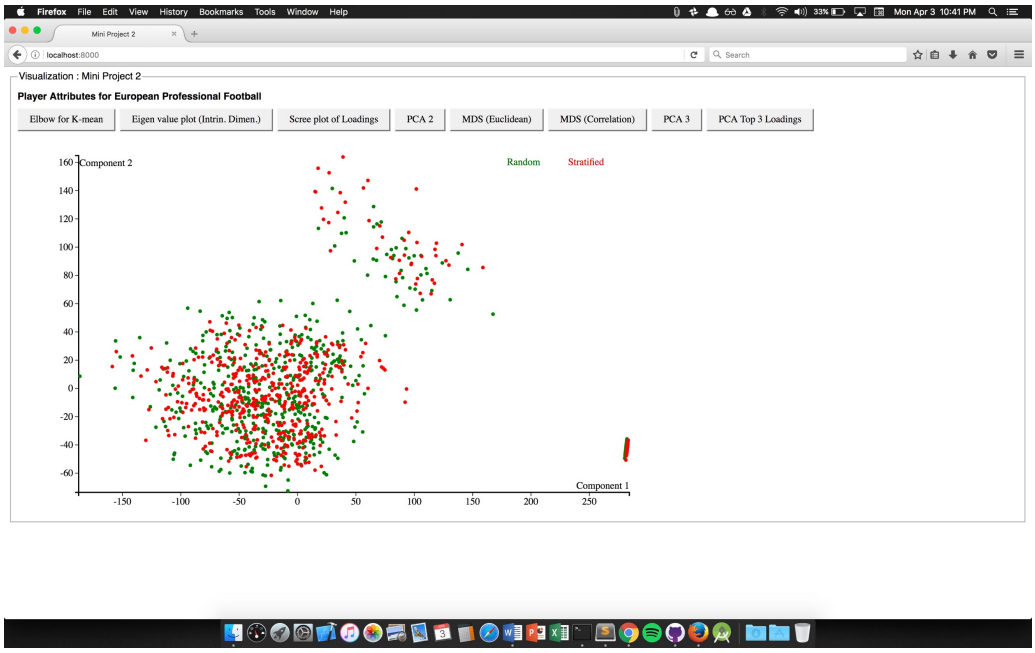

FIGURE 8 DATA PROJECTED ON 2 PCA VECTORS

```
def find_pca2(df1,df2,file_name):
    print "PCA 2 started"
    pca = PCA(n_components=2)
    random_sample = pandas.DataFrame(pca.fit_transform(df1))
    stratified_sample = pandas.DataFrame(pca.fit_transform(df2))
    random_sample.columns = ['PC1','PC2']
    stratified_sample.columns = ['PC1','PC2']
    type1 = pandas.DataFrame(numpy.ones([len(random_sample.index),1], dtype=int),columns=['type'])
    random_sample = pandas.concat([random_sample, type1], axis=1)
    type2 = pandas.DataFrame(numpy.ones([len(stratified_sample.index),1], dtype=int)+1,columns=['type'])
    stratified_sample = pandas.concat([stratified_sample, type2], axis=1)
    sample = pandas.concat([random_sample, stratified_sample], axis=0)
    print "PCA 2 finished"
    file_name = directory + file_name
    print "Creating file "+file_name
    sample.to_csv(file_name,sep=',',index=False)
```

FIGURE 9 PYTHON CODE FOR PROJECTING DATA ON 2 PCA VECTORS

FIGURE 10 DATA PROJECED USING MDS (EUCLIDIAN DISTANCE)

```
def find_MDS_euclidean(df1,df2,file_name):
    print "MDS (euclidean) started"
    dis_mat_random = metrics.pairwise_distances(df1, metric = 'euclidean')
    mds = MDS(n_components=2, dissimilarity='precomputed')
    random_sample = pandas.DataFrame(mds.fit_transform(dis_mat_random))
    dis_mat_stratified = metrics.pairwise_distances(df2, metric = 'euclidean')
    mds = MDS(n_components=2, dissimilarity='precomputed')
    stratified_sample = pandas.DataFrame(mds.fit_transform(dis_mat_stratified))
    random_sample.columns = ['PC1','PC2']
    stratified_sample.columns = ['PC1','PC2']
    type1 = pandas.DataFrame(numpy.ones([len(random_sample.index),1], dtype=int),columns=['type'])
    random_sample = pandas.concat([random_sample, type1], axis=1)
    type2 = pandas.DataFrame(numpy.ones([len(stratified_sample.index),1], dtype=int)+1,columns=['type'])
    stratified_sample = pandas.concat([stratified_sample, type2], axis=1)
    sample = pandas.concat([random_sample, stratified_sample], axis=0)
    print "MDS (euclidean) finished"
    file_name = directory + file_name
    print "Creating file "+file_name
    sample.to_csv(file_name,sep=',',index=False)
```

FIGURE 11 PYTHON CODE FOR PROJECTING DATA USING MDS (EUCLIDEAN DISTANCE)

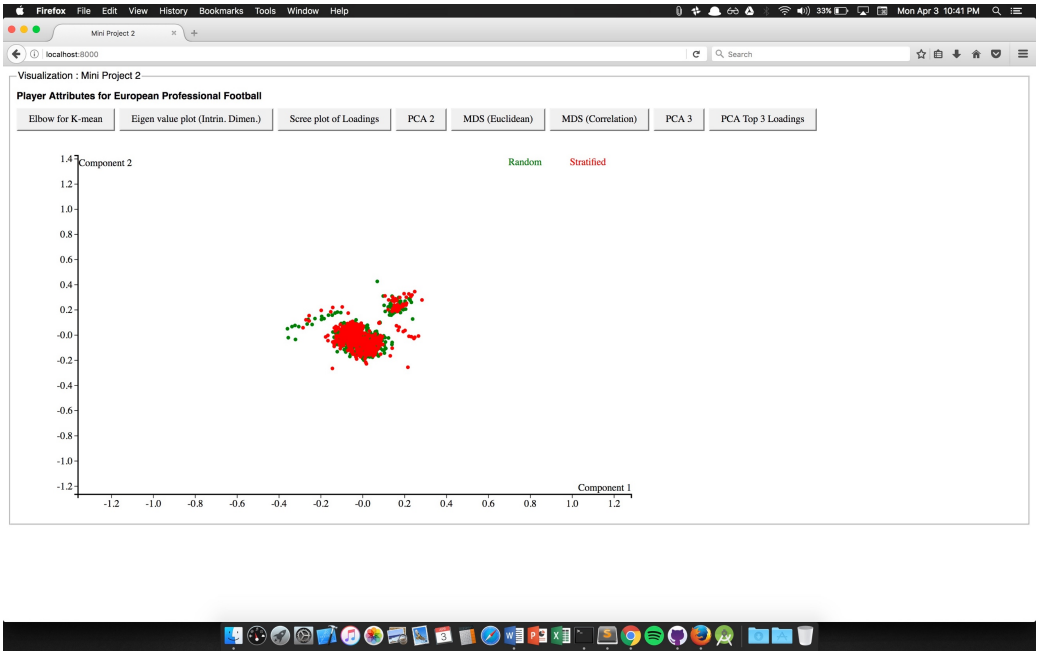

FIGURE 12 DATA PROJECTED USING MDS (CORRELATION DISTANCE)

```
def find_MDS_correlation(df1,df2,file_name):
    print "MDS (correlation) started"
    dis_mat_random = metrics.pairwise_distances(df1, metric = 'correlation')
    mds = MDS(n_components=2, dissimilarity='precomputed')
    random_sample = pandas.DataFrame(mds.fit_transform(dis_mat_random))
    dis_mat_stratified = metrics.pairwise_distances(df2, metric = 'correlation')
    mds = MDS(n_components=2, dissimilarity='precomputed')
    stratified_sample = pandas.DataFrame(mds.fit_transform(dis_mat_stratified))
    random_sample.columns = ['PC1','PC2']
    stratified_sample.columns = ['PC1','PC2']
    type1 = pandas.DataFrame(numpy.ones([len(random_sample.index),1], dtype=int),columns=['type'])
    random_sample = pandas.concat([random_sample, type1], axis=1)
    type2 = pandas.DataFrame(numpy.ones([len(stratified_sample.index),1], dtype=int)+1,columns=['type'])
    stratified_sample = pandas.concat([stratified_sample, type2], axis=1)
    sample = pandas.concat([random_sample, stratified_sample], axis=0)
    print "MDS (correlation) finished"
    file_name = directory + file_name
    print "Creating file "+file_name
    sample.to_csv(file_name,sep=',',index=False)
```

FIGURE 13 PYTHON CODE FOR PROJECTING DATA USING MDS (CORRELATION DISTANCE)

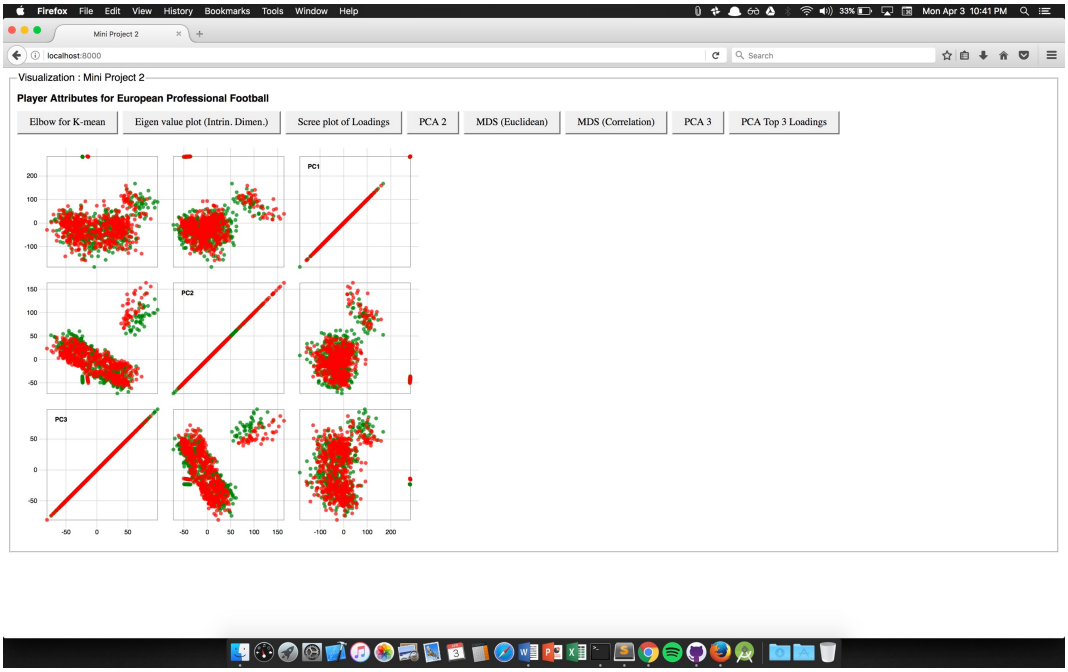

FIGURE 14 SCATTERPLOT MATRIX OF PCA 3

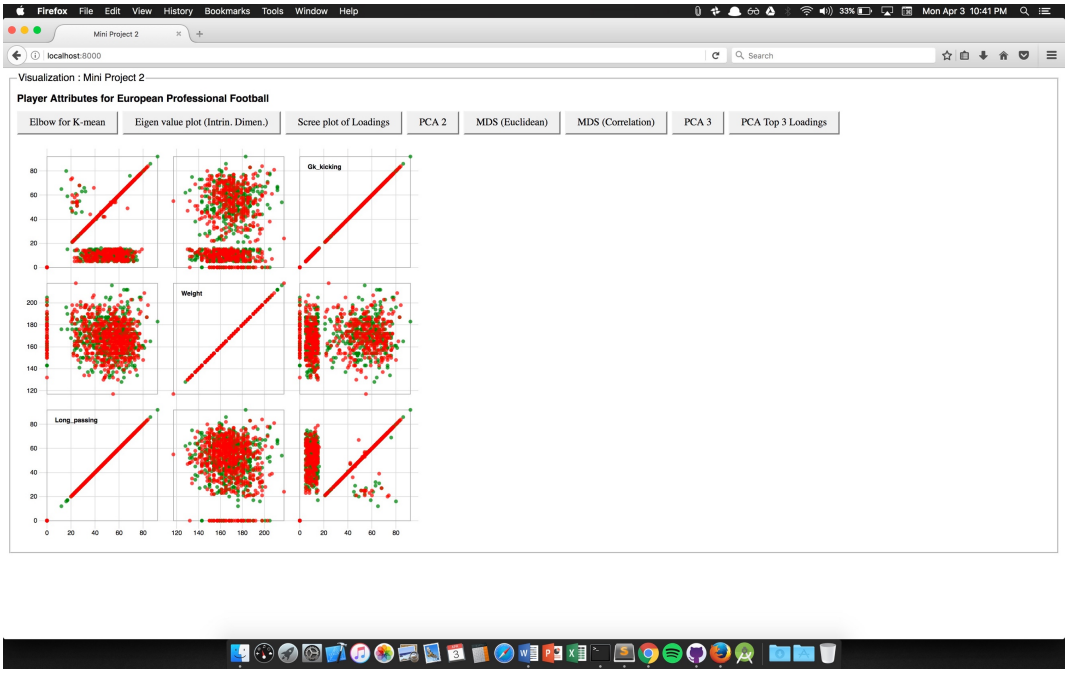

FIGURE 15 SCATTERPLOT MATRIX OF TOP 3 HIGHEST LOADINGS

```
def find_pca3(df1,df2,file_name,col_list):
    print "PCA 3 started"
    pca = PCA(n_components=3)
    random_sample = pandas.DataFrame(pca.fit_transform(df1))
    stratified_sample = pandas.DataFrame(pca.fit_transform(df2))
    random_sample.columns = ['PC1','PC2','PC3']
    stratified_sample.columns = ['PC1','PC2','PC3']
    type1 = pandas.DataFrame(numpy.ones([len(random_sample.index),1], dtype=int),columns=['type'])
    random_sample = pandas.concat([random_sample, type1], axis=1)
    type2 = pandas.DataFrame(numpy.ones([len(stratified_sample.index),1], dtype=int)+1,columns=['type'])
    stratified_sample = pandas.concat([stratified_sample, type2], axis=1)
    sample = pandas.concat([random_sample, stratified_sample], axis=0)
    print "PCA 3 finished"
    file_name = directory + file_name
    print "Creating file "+file_name
    sample.to_csv(file_name,sep=',',index=False)
    random_tuple = []
    for i in range(len(df1)):
        random_tuple.append((df1[col_list[0]].iloc[i],df1[col_list[1]].iloc[i],df1[col_list[2]].iloc[i],'1'))
    random_tuple_df = pandas.DataFrame(random_tuple)
    random_tuple_df.columns = [col_list[0],col_list[1],col_list[2],'type']
    stratified_tuple = []
    for i in range(len(df2)):
        stratified_tuple.append((df2[col_list[0]].iloc[i],df2[col_list[1]].iloc[i],df2[col_list[2]].iloc[i],'2'))
    stratified_tuple_df = pandas.DataFrame(stratified_tuple)
    stratified_tuple_df.columns = [col_list[0],col_list[1],col_list[2],'type']
    sample = pandas.concat([random_tuple_df, stratified_tuple_df], axis=0)
    file_name = directory + 'pca3_loadings.csv'
    print "Creating file "+file_name
    sample.to_csv(file_name,sep=',',index=False)
```

FIGURE 16 PYTHON CODE FOR GENERATING THE DATA FOR SCATTERPLOT MATRIX

SOME OBSERVATIONS

1. Having a large data can provide more detailed info about the pattern in the data, but increase the amount of time it takes to process the data.
2. Having the small fraction of data by random sampling out of original data can significantly change the outlook the data provides.
3. In Football player data, all the fields had some importance as most of the columns are characteristic property of the player.
4. I replaced the blank part in data with 0, but if it is replaced by mean then top 3 loadings may change.
5. Various methods can be used to find the k of k-means like information criteria method, x-means clustering etc. each providing more efficient way to find k for k-means.

REFERENCES:

1 http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
2 http://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html
3 http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
4 https://bl.ocks.org/mbostock/3213173
5 http://bl.ocks.org/weiglemc/6185069
6 https://bl.ocks.org/mbostock/3883245
7 http://bl.ocks.org/d3noob/b3ff6ae1c120eea654b5
8 http://bl.ocks.org/d3noob/8603837