CSE 564
(VISUALIZATION)

Lab 2 PROJECT REPORT

Submitted By: Rohit Bassi (112505198)

Advisor: Klaus Mueller

In this assignment, there were 3 tasks

TASK1: data clustering and decimation

1. Took a dataset : datafile.csv
2. I used python-flask server as my backend and for front-end I used D3
3. Now, I started with the task 1 i.e random sampling and stratified sampling

RANDOM SAMPLING:
I took 500 as a sample size.
Here is the code for random sampling

```
#RANDOM SAMPLING
def random_sampling():
    rand = random.sample(range(len(df)), size)
    for i in rand:
        random_samples.append(data[i])
```

STRATIFIED SAMPLING/ADAPTIVE SAMPLING:
Here I used the concept of K-means clustering and find the optimal K and then divided the data into K groups or clusters.
Below is the code for K-means and adaptive sampling data.

```
def kelbow():
    #using kmeas method, inbuilt function in python
    distortions = []
    X=df[ftrs]
    K = range(1,10)
    for k in K:
        kmeanModel = KMeans(n_clusters=k).fit(X)
        kmeanModel.fit(X)
        distortions.append((k,sum(np.min(cdist(X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0]))
    coordinates=np.asarray(distortions)
    # print(coordinates)
    #this is method to find optimal K, refered this from the link professor gave
    first=coordinates[0]
    last=coordinates[-1]
    lineVec=last-first
    lineVecNorm = lineVec / np.sqrt(np.sum(lineVec**2))
    vecFromFirst = coordinates - first
    scalarProduct = np.sum(vecFromFirst * npm.repmat(lineVecNorm, 9, 1), axis=1)
    vecFromFirstParallel = np.outer(scalarProduct, lineVecNorm)
    vecToLine = vecFromFirst - vecFromFirstParallel
    distToLine = np.sqrt(np.sum(vecToLine ** 2, axis=1))
    idxOfBestPoint = np.argmax(distToLine)
    # print(distToLine)
    # print(idxOfBestPoint)
    # plt.plot(K, distortions,'bx-')
    # plt.xlabel('k')
    # plt.ylabel('Distortion')
    # plt.title('The Elbow Method showing the optimal k')
    # plt.show()
    kvalue= distortions[idxOfBestPoint][0]
    # print(kvalue)
```

Here , kvalue =3 in my case (optimal K)

```
kvalue= distortions[idxOfBestPoint][0]
# print(kvalue)

#dividing the data in K groups/clusters
check=KMeans(n_clusters=kvalue).fit(X)
list2=check.labels_
# print(list2)
df['kcluster'] = pd.Series(list2)
clustering={}
for i in range(len(list2)):
    try:
        clustering[list2[i]].append(i)
    except KeyError:
        clustering[list2[i]]=[i]
# print(clustering)
# print (len(clustering[0]))
# print (len(clustering[1]))
# print (len(clustering[2]))
#k=3
global adaptive_samples
#getting the cluster and storing the data in 3 clusters
kcluster0 = df[df['kcluster'] == 0]
kcluster1 = df[df['kcluster'] == 1]
kcluster2 = df[df['kcluster'] == 2]
#sampling the data with 40% samples
temp1=kcluster0[ftrs].sample(frac=0.4)
temp2=kcluster1[ftrs].sample(frac=0.4)
temp3=kcluster2[ftrs].sample(frac=0.4)

#SO this is my adaptive samples data!! yipee! -TASK1 done
adaptive_samples = pd.concat([temp1, temp2, temp3])
# print(len(adaptive_samples))
```

Here I found the Adaptive sampling data

TASK2 :dimension reduction (use decimated data
-> First, find the intrinsic dimensionality of the data using PCA
In PCA, firstly get Eigen values and Eigen vectors
We also need covariance matrix (one can take correlation matrix) as well

```python
#This is calculation of eigen values and eigen vector using in built libraries
def vector(data):
    # print(data)
    #note-either correaltion or covariance will work,will get same result
    R = np.cov(data.T) # first find the covariance!
    print("-----")
    # print(R)
    evals, evecs = LA.eigh(R) #then calculate the eigenvalues and vectors
   # sort eigenvalue in decreasing order
    idx = np.argsort(evals)[::-1]
    evecs = evecs[:,idx]
    # sort eigenvectors according to same index
    evals = evals[idx]
    return evals,evecs
```

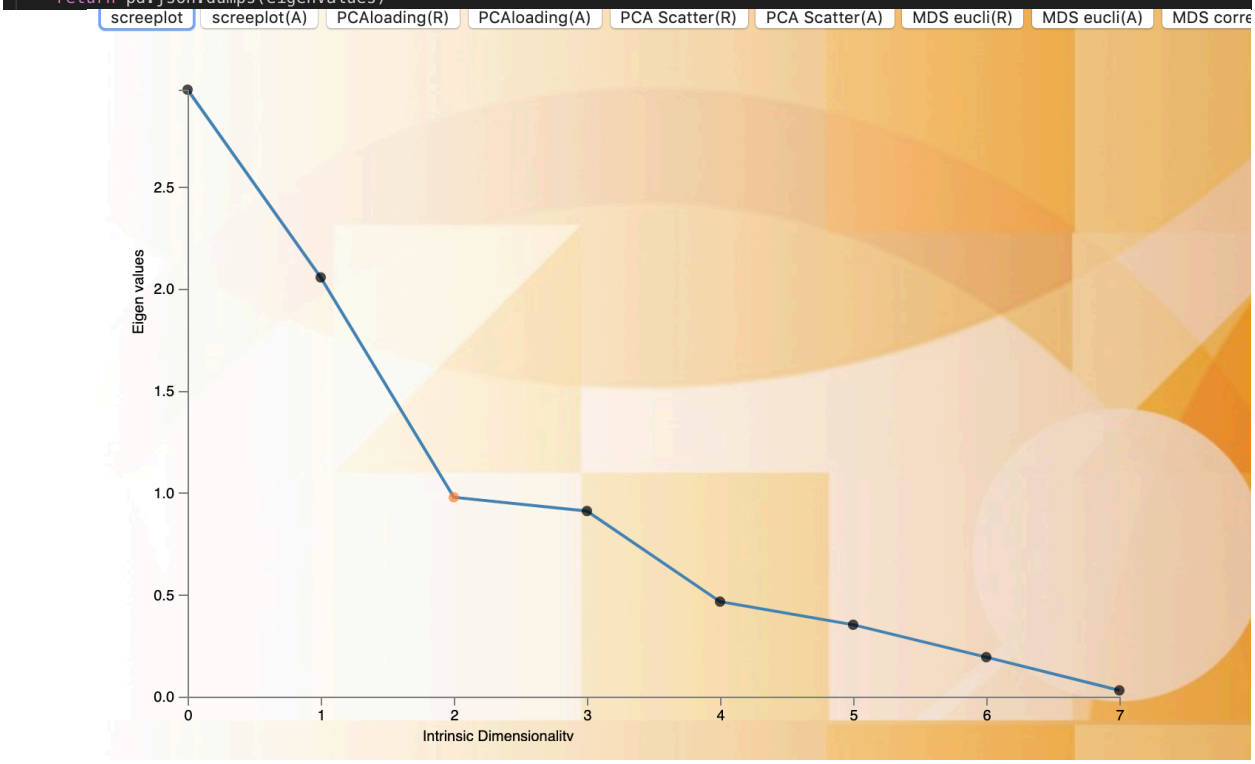
Now below is the code for plotting scree plot with Eigen values to get intrinsic dimensionality for both random and adaptive sample

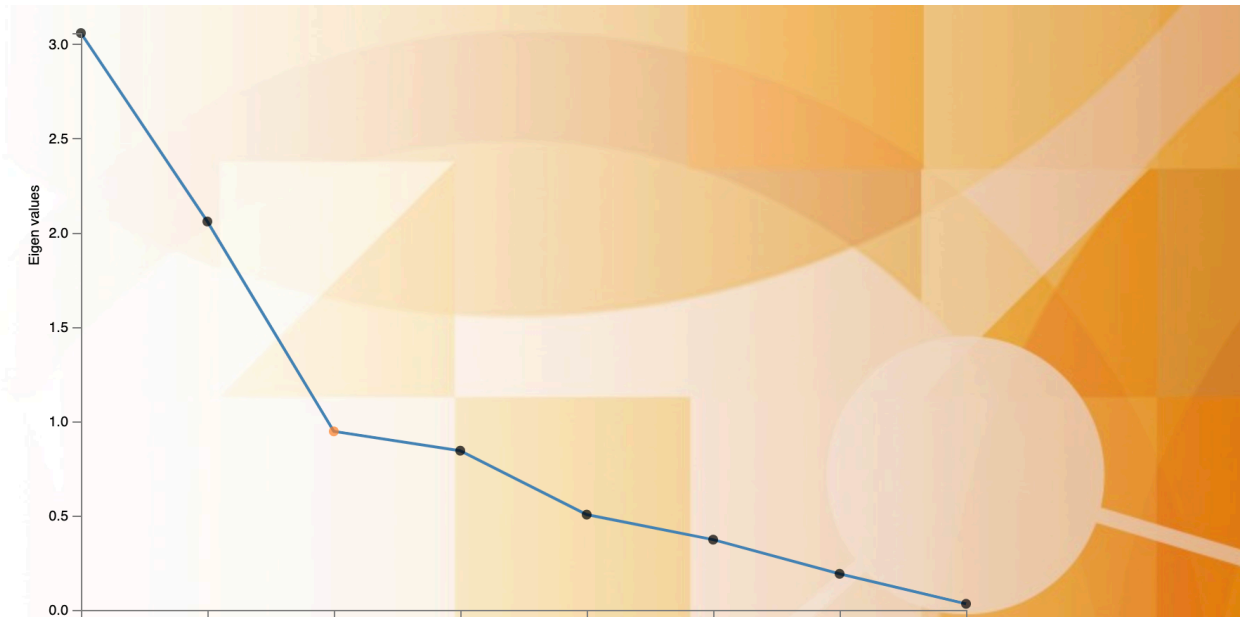
```python
#this function is used to calculate intrinstic dimensionality using scree plot of adaptive samples
@app.route("/screeR")
def scree_adaptive():

    [eigenValues, eigenVectors] = vector(adaptive_samples[ftrs])#ftrs is the attributes that is used in this assignment
    # chart_data = pd.json.dumps(eigenValues)
    # data = {'chart_data': chart_data}
    return pd.json.dumps(eigenValues)

#this function is used to calculate intrinsic dimensionality using screeplot of random samples
@app.route("/screeA")
def scree_adaptive1():
    [eigenValues, eigenVectors] = vector(df[ftrs])

    # chart_data = pd.json.dumps(eigenValues)
    # data = {'chart_data': chart_data}
    return pd.json.dumps(eigenValues)
```



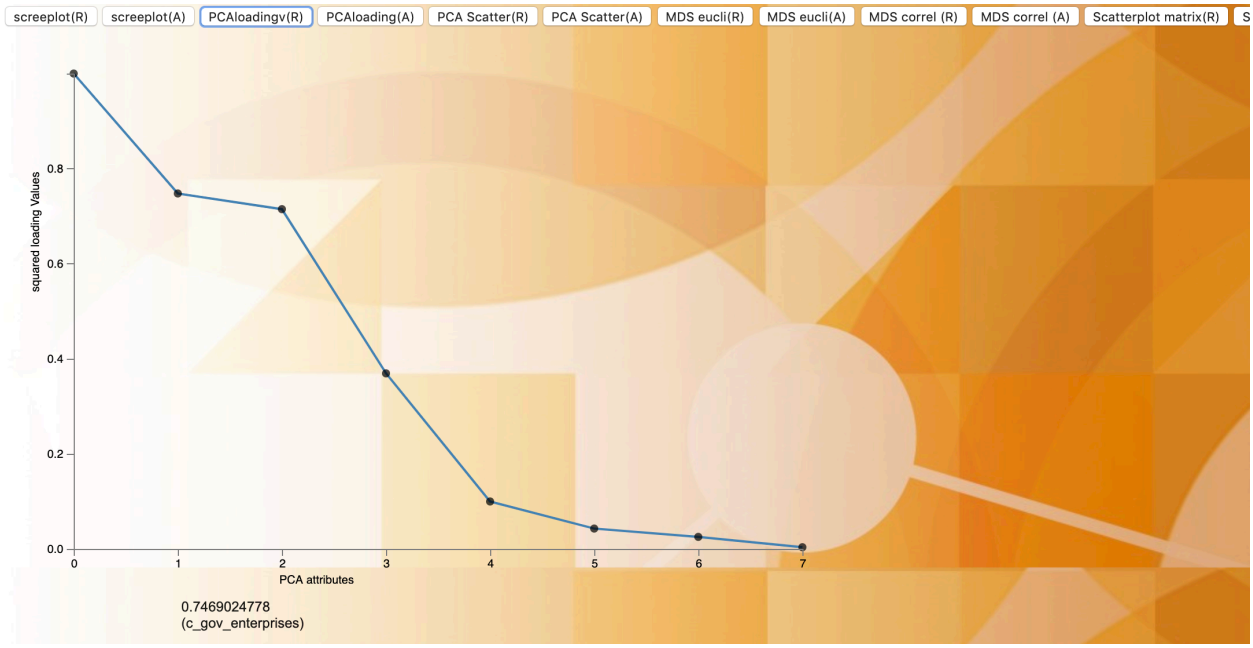


The above and below plots are for before and after sampling(scree plots)-**(red dot is not there in video because I added the color after I recorded and uploaded the video)

Now I have get top 3 highest PCA loadings,
Each eigen vector tells us about the contribution of a variable

```
counter = len(eigenVectors)
for cols in range(0, counter):
    loadings = 0
    for row in range(0, k):
        loadings = loadings + eigenVectors[row][cols] * eigenVectors[row][cols]
    squaredLoadings.append(loadings)
```

Correlations between variables and the principal axes are known as loadings



Now I have made a plot between squared loading values and PCA attribute
(note- I have sorted the attribute in decreasing order)
Top 3 PCA attributes are:-
1."g_restrictions_sale_real_property"
2. "c_gov_enterprises"
3."b_impartial_courts"

```
squaredLoading=generate11(data,3)
print(squaredLoading)
print(sorted(squaredLoading))
print(list3)
list6=[]
# pca_filter_attr=squaredLoading
# print(pca_filter_attr)
# print(reversed(squaredLoading))

#this method is used for getting top 3 pca attributes indexes!!
sortlist=sorted(squaredLoading)
for i in sortlist:
    print (i)
    for j,k in zip(squaredLoading,range(len(squaredLoading))):
        if j==i:
            print(k)
            list6.append(k)
print(list6[::-1])

pca_filter_attr=list6[::-1]
print(pca_filter_attr)
```

TASK3:visualization (use dimension reduced data)

a) visualize the data projected into the top two PCA vectors via 2D scatterplot

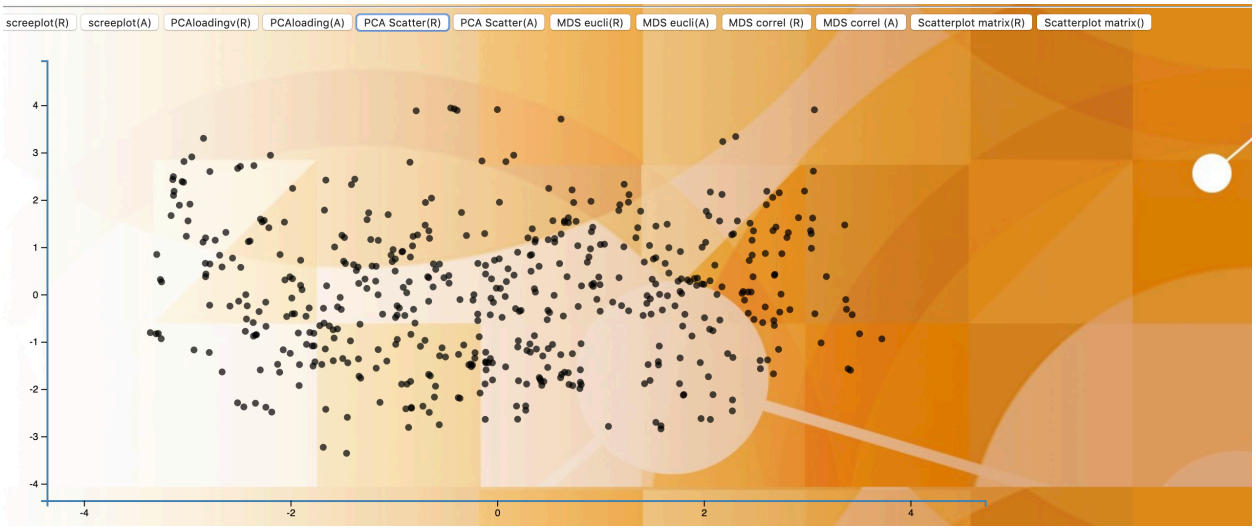
I already have the top 3 PCA attributes and now I have to plot top 2 PCA vectors via scatterplot.
I calculated pca filter attire -that means index of top pca attributes:- code->>

Python got a inbuilt PCA function
Here is the code below:-

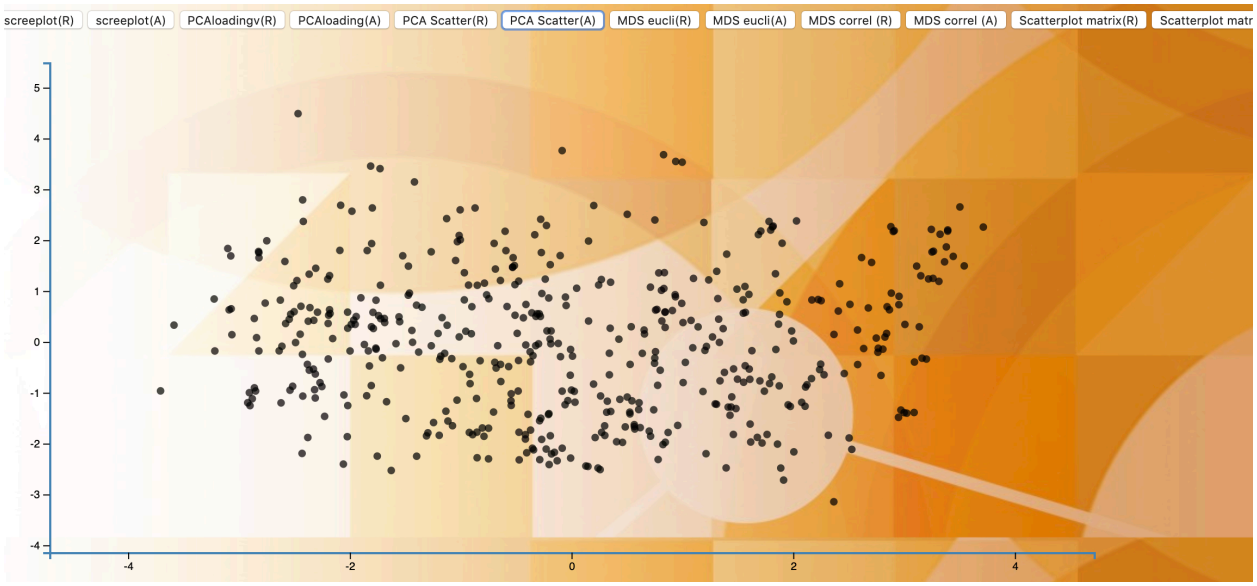
```
@app.route('/scatterrandom')
def pcaR():
    col_content = []
    #basic implementation

    # pca = PCA(n_components=2)
    # principalComponents = pca.fit_transform(x)
    # principalDf = pd.DataFrame(data = principalComponents
    #      , columns = ['principal component 1', 'principal component 2'])
    pca = PCA(n_components=2) #take n component as 2
    X = random_samples
    X = pca.fit_transform(X)
    col_content = pd.DataFrame(X) #making coloums
    for i in range(0, 2): #loop upto 2
        #i have calculated the pca_filter_attr below in the code!
        #it contains the index of pca attri (with highest values indexes at the first)
        col_content[ftrs[pca_filter_attr[i]]] = dfor[ftrs[pca_filter_attr[i]]][:size]

    # return col_content.to_json
    return pd.json.dumps(col_content)
```

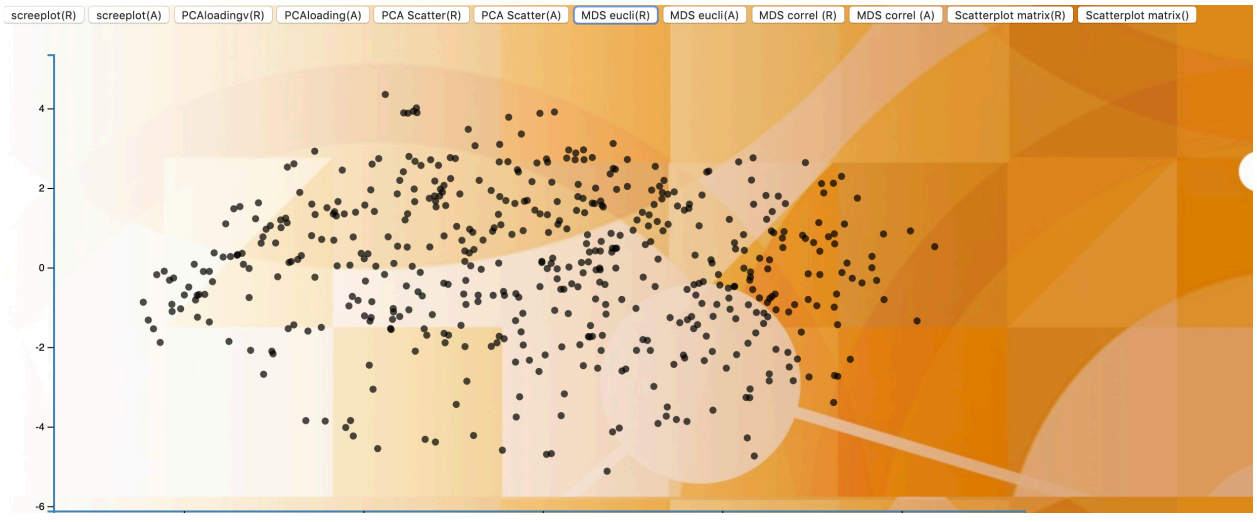


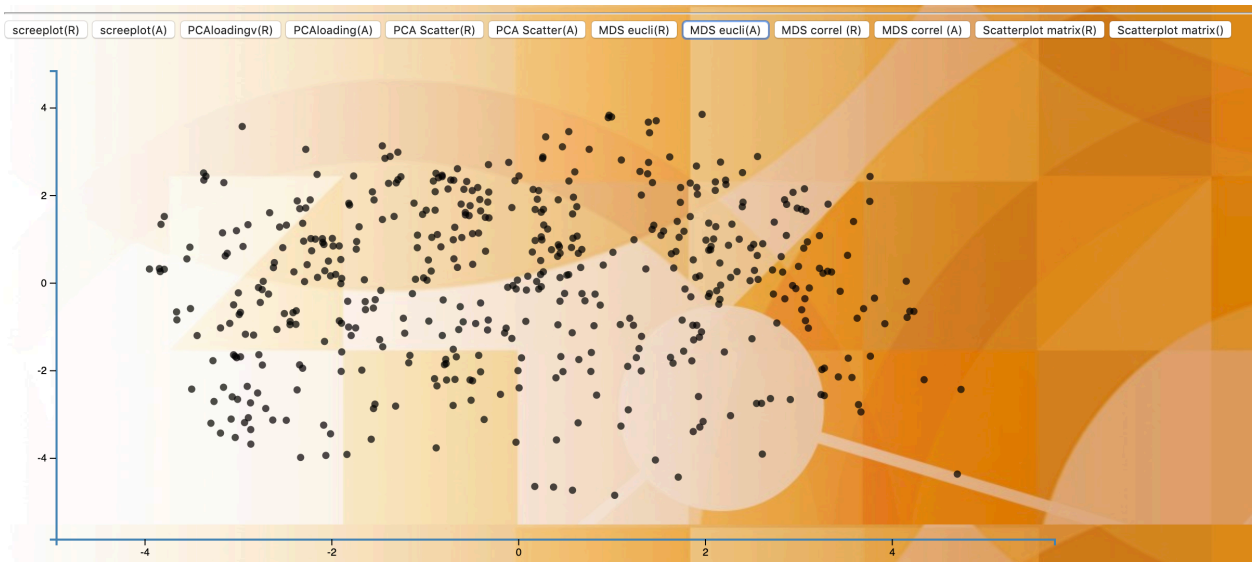
Above and below visualization are for random and adaptive sampling



B)visualize the data via MDS (Euclidian & correlation distance) in 2D scatterplots
MDS (Euclidian & correlation distance) ,it maps distance between variable from N-D array to Lower D array… MDS for Euclidian:-

```
    #Dissimilarity measure to use:
# 'euclidean':Pairwise Euclidean distances between points in the dataset.
# 'precomputed':Pre-computed dissimilarities are passed directly to fit and fit_transform
    dataset1 = manifold.MDS(n_components=2, dissimilarity='precomputed')
     #Compute the distance matrix from a vector array X
    sltry = pairwise_distances(random_samples, metric='euclidean')
    X = dataset1.fit_transform(sltry)
    col_content = pd.DataFrame(X)
```

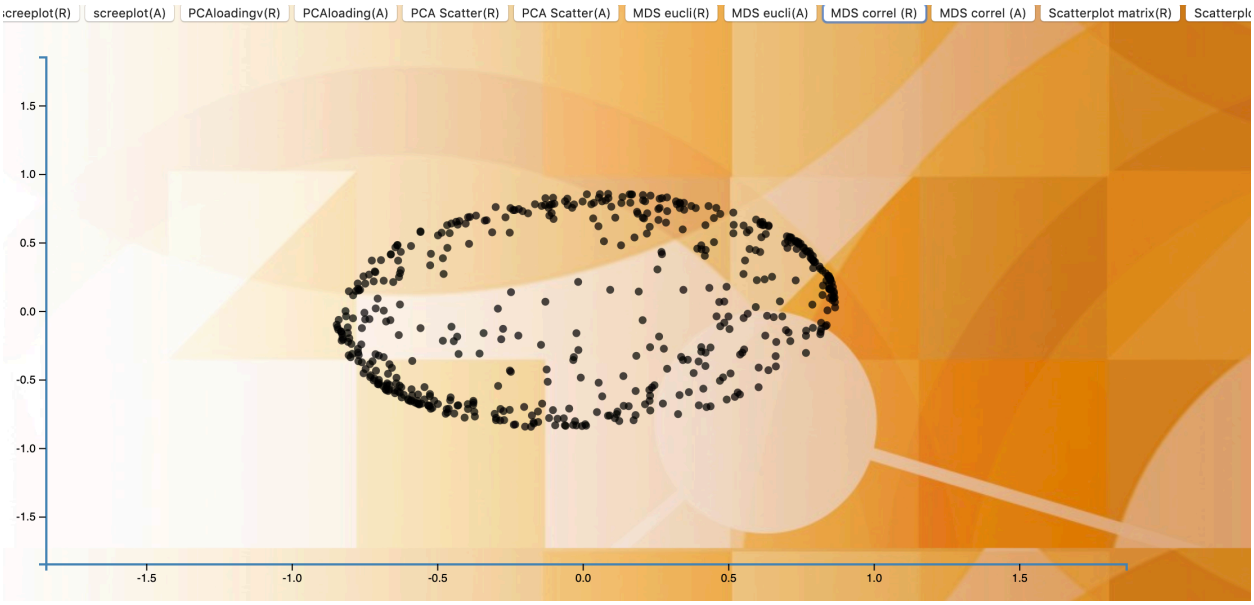


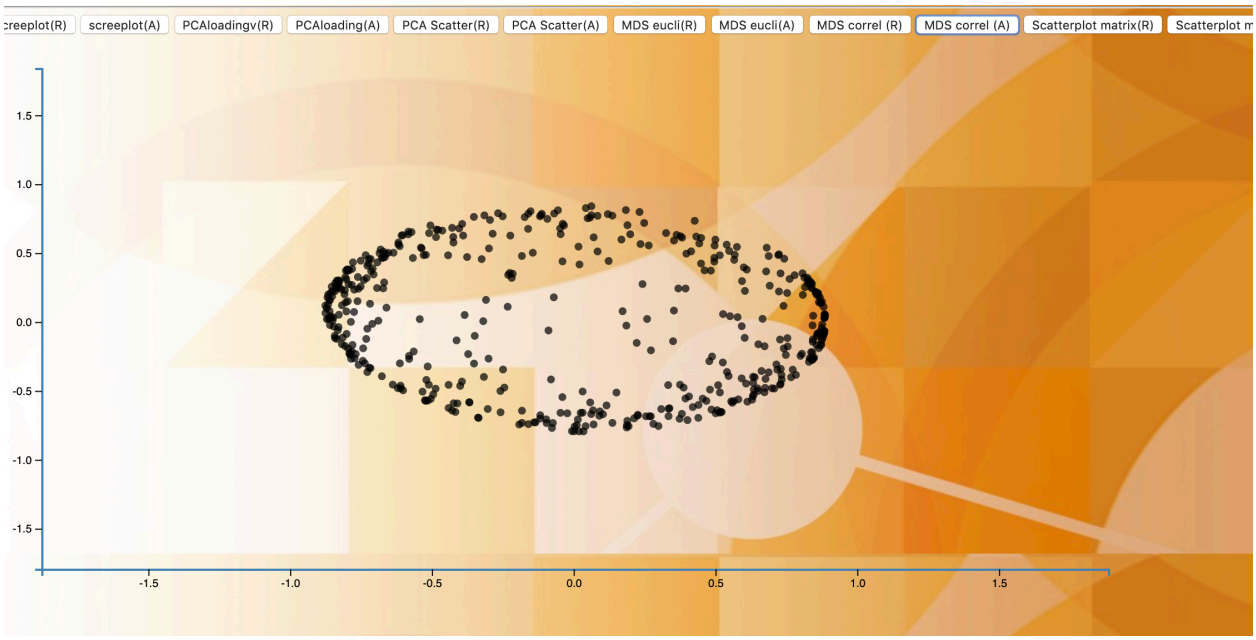


MDS for Correlation:-

```
    #Dissimilarity measure to use:
# 'euclidean':Pairwise Euclidean distances between points in the dataset.
# 'precomputed':Pre-computed dissimilarities are passed directly to fit and fit_transform
    dataset1 = manifold.MDS(n_components=2, dissimilarity='precomputed')
    sltry = pairwise_distances(random_samples, metric='correlation')
    X = dataset1.fit_transform(sltry)
    col_content = pd.DataFrame(X)
```

RANDOM sampling

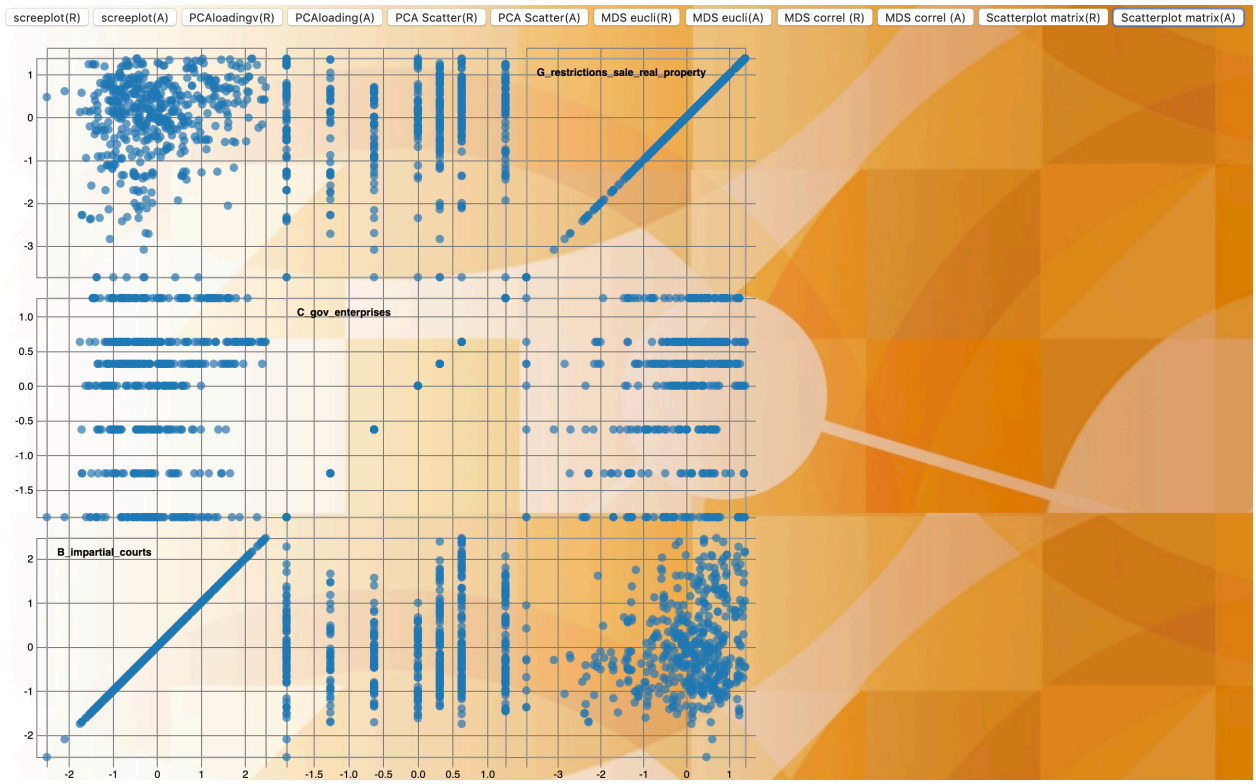




Above is the visualization of adaptive sampling(MDS correlation distance)

C) visualize scatterplot matrix of the three highest PCA loaded attributes

I have already calculated the highest loaded PCA attributes in previous tasks
Below is the scatterplot matrix of adaptive sampling :-



Below is the scatterplot matrix of random sampling :-