<u>CSE 564 - VISUALIZATION</u>

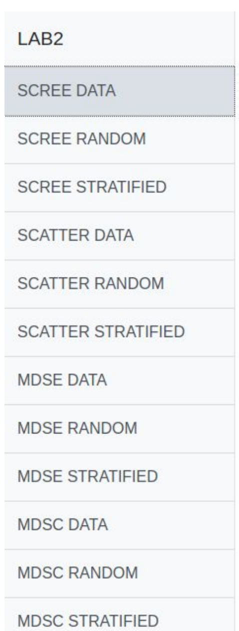
LAB 2

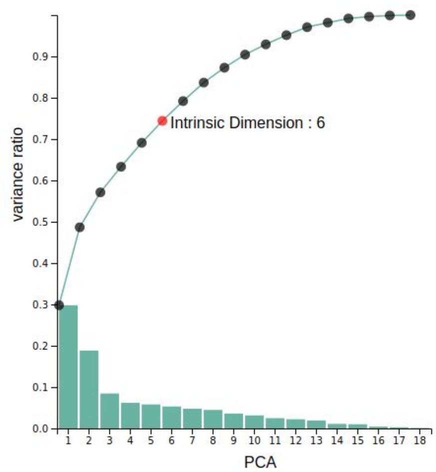
This is the report for lab 2 and contains a brief explanation of it.

Dataset : The dataset is taken from kaggle (FIFA 19 players), and has 18 features and 572 samples.

Technologies used: A client sever system is used, Having flask as a backend, all the data processing is done in python and then the result are sent to frontend Where they are visualized using D3.js.

The Loading page : I have taken a simple fixed side menu bar(only certain tasks were needed to be displayed), a bootstrap template.





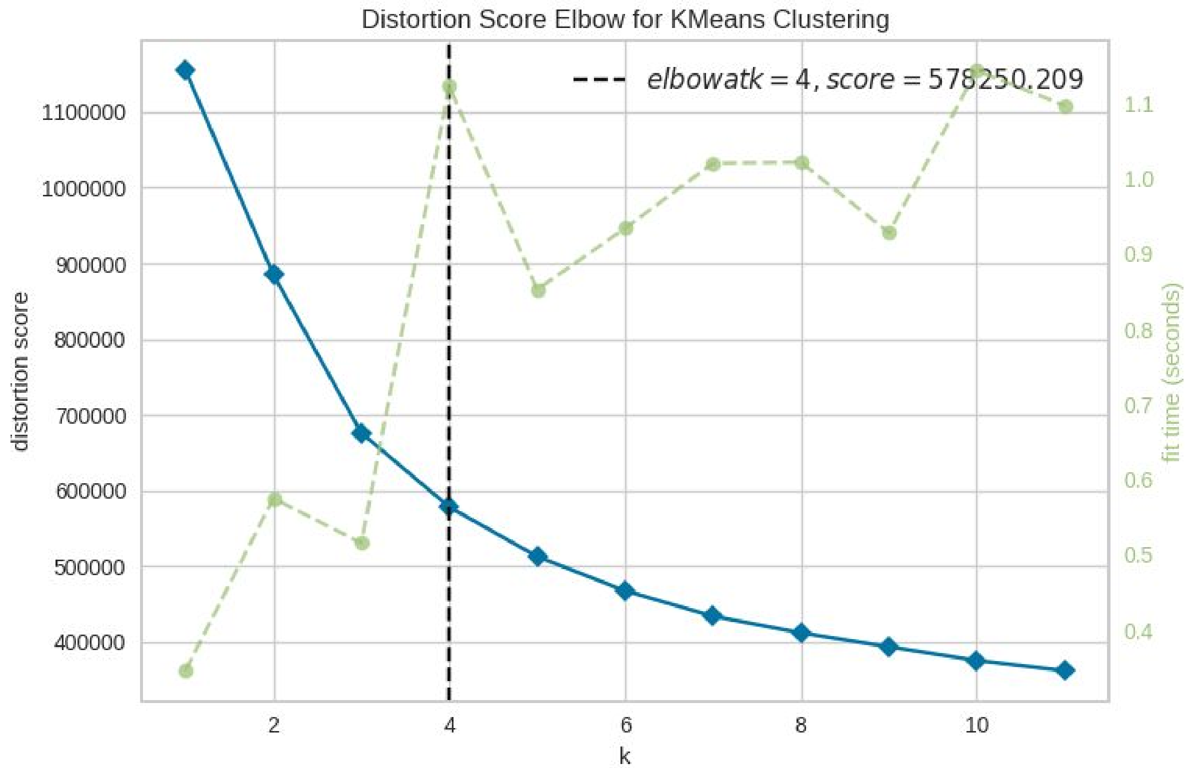
Task `1:

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

```python
@app.route('/')
def hello_world():
    clean_data()
    return render_template("index.html")


def randomSample():
    return data.sample(frac=0.25)

#Use of Kelbow visualizer
def elbowCheck():
    mat = data.values
    mat = mat.astype(float)
    model = KMeans()
    visualizer = KElbowVisualizer(model, k=(1,12))
    visualizer.fit(mat)
    visualizer.show(outpath="static/images/kmeans.png")

```

Random sampling and stratified sampling is done on the data and only 25% of the data is kept, for stratified sampling KMeans clustering is used and to get the best cluster size Kelbow visualizer is used.

```python
def stratifiedSample():
    #From elbow check the cluster size is best when K=4
    global frame
    global sample_done
    global colors

    if not sample_done:
        nCluster = 4
        mat = data.values
        mat = mat.astype(float)
        kmeans = KMeans(n_clusters=nCluster)
        kmeans.fit(mat)
        cInfo = kmeans.labels_
        #save clusters and recreate dataframe
        cluster = [[],[],[],[]]
        for i in range(0,len(data.index)):
            cluster[cInfo[i]].append(data.loc[i]);

        temp = []
        colors = []

        for i in range(0,nCluster):
            tempFrame = pd.DataFrame(cluster[i]).sample(frac=0.25)
            dfSize = tempFrame.shape[0]
            for j in range(0,dfSize):
                colors.append(i)
            temp.append(tempFrame)
        frame = pd.concat(temp)
        sample_done = True

    return frame

```



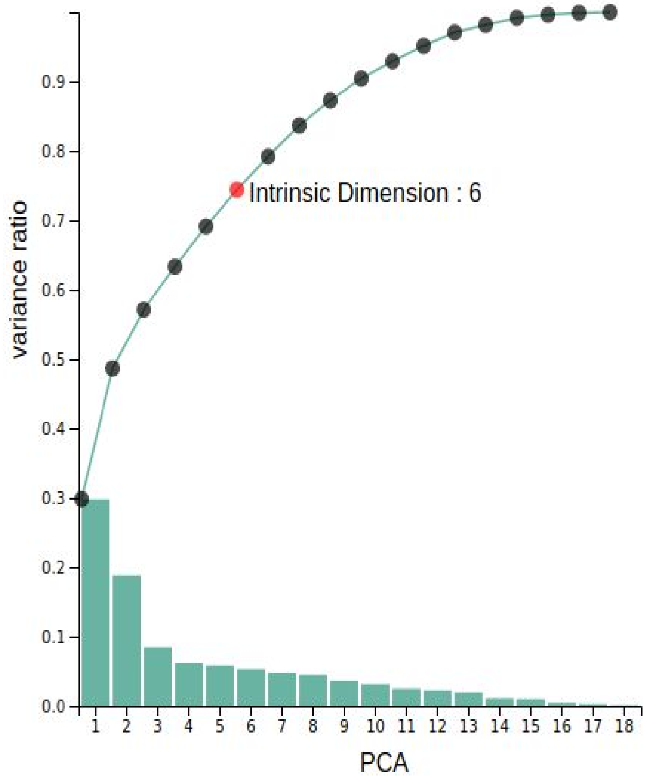
Task 2:

1. *Find the intrinsic dimensionality of the data using PCA*
2. *Produce scree plot visualization and mark the intrinsic dimensionality*
3. *Show the scree plots before/after sampling to assess the bias introduced*
4. *Obtain the three attributes with highest PCA loadings*

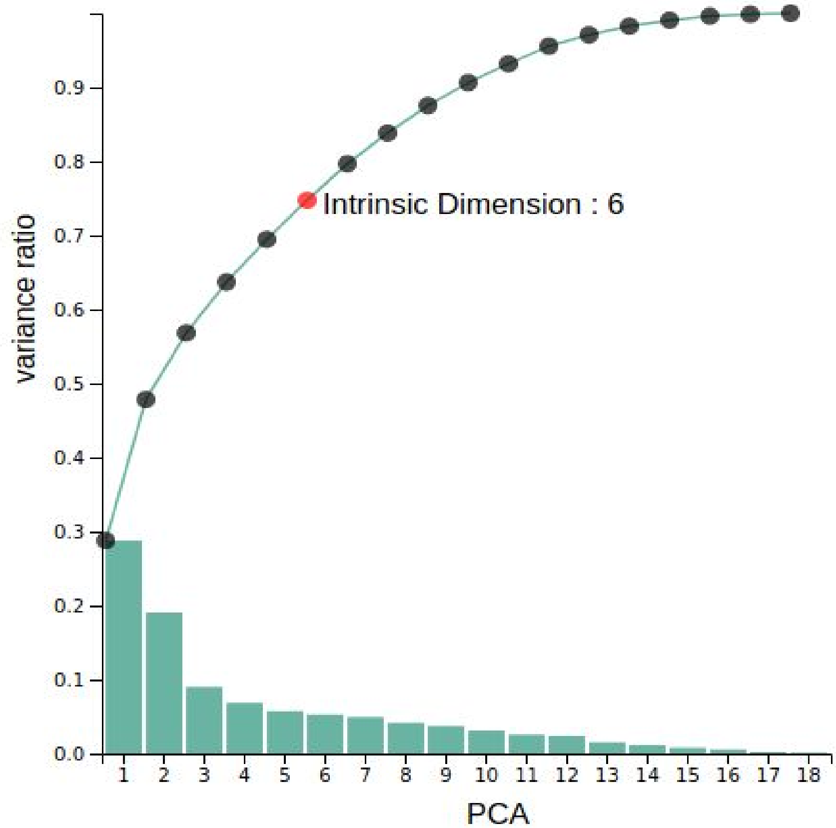
By using the PCA technique Intrinsic dimension of the data is found, when 75% of variance is accumulated we stop and consider it to be the intrinsic dimension.

Scree Plots with Intrinsic Dimension.

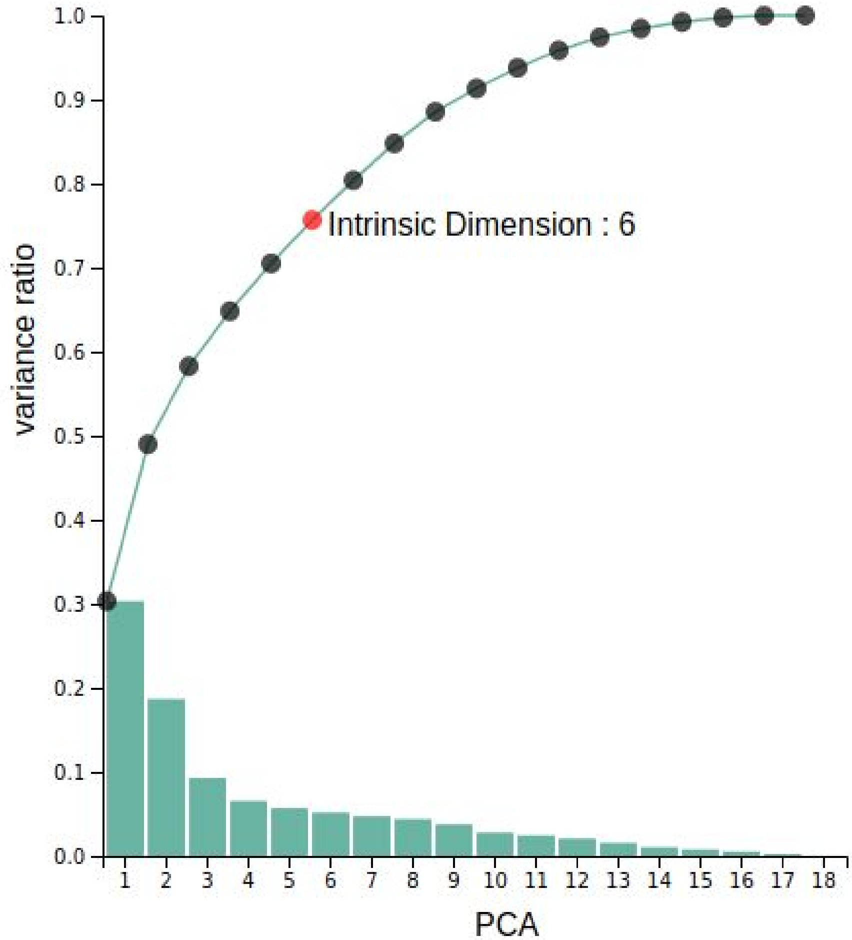
1. Scree Plot original data



2. Scree Plot Random sampling



3. Scree Plot stratified sampling



Observation : We can see that the Intrinsic dimension for the three kind of sampling is Same, we can for now think that the samples are in a good way represnting the data.

Code snippets:

```python

def pcaAnalysis(sample):

    global sq_load
    global intD

    mat = sample.values
    mat = mat.astype(float)
    mat = StandardScaler().fit_transform(mat)
    nComponents = 18
    pca = PCA(n_components=nComponents)
    pca.fit_transform(mat)
    count = 0
    cumsu = 0
    for eigV in pca.explained_variance_ratio_:
        cumsu += eigV
        if cumsu > 0.78:
            break

        count += 1
    intD  = count
    #get the loading matrix
    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)
    sq_loadings = np.square(loadings)
    sq_load = np.sum(sq_loadings[:,0:3],axis=1)
    topPcaLoad()
    return pca.explained_variance_ratio_

```

Common code for doing pca, various types of samples are passed to this function.

Loading Matrix is calculated using the formula.

loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

The top three attributes with highest pca loadings come out to be

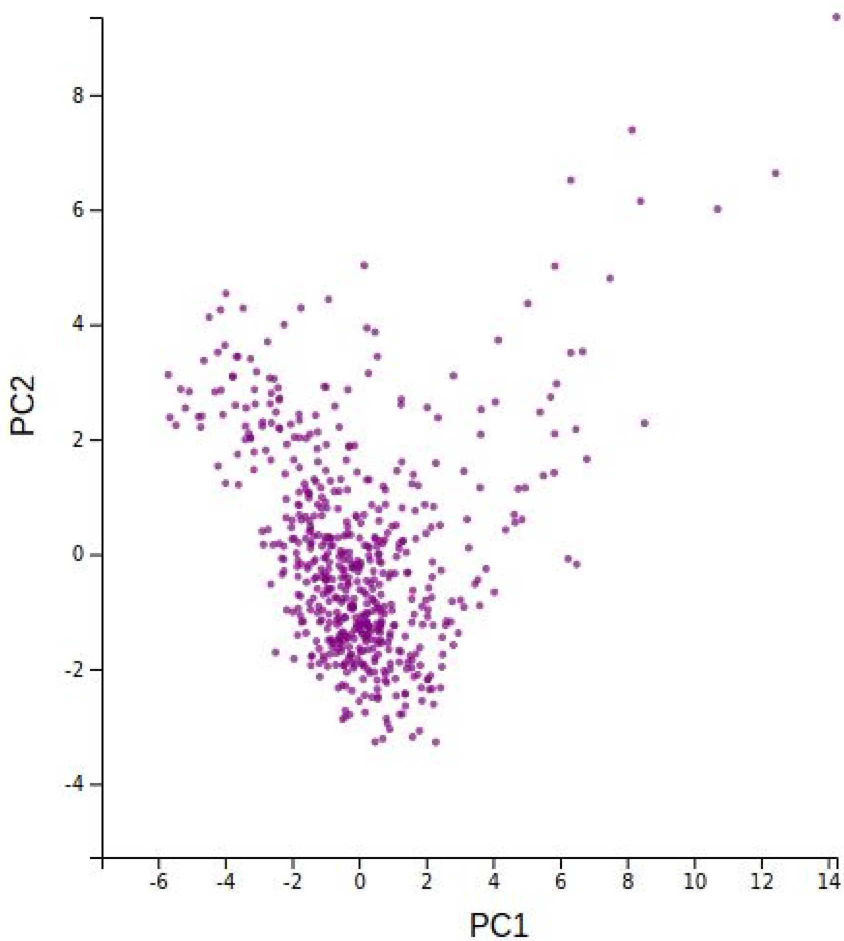
1. 'Value',
2. 'Acceleration',
3. 'Release Clause'

Task 3:

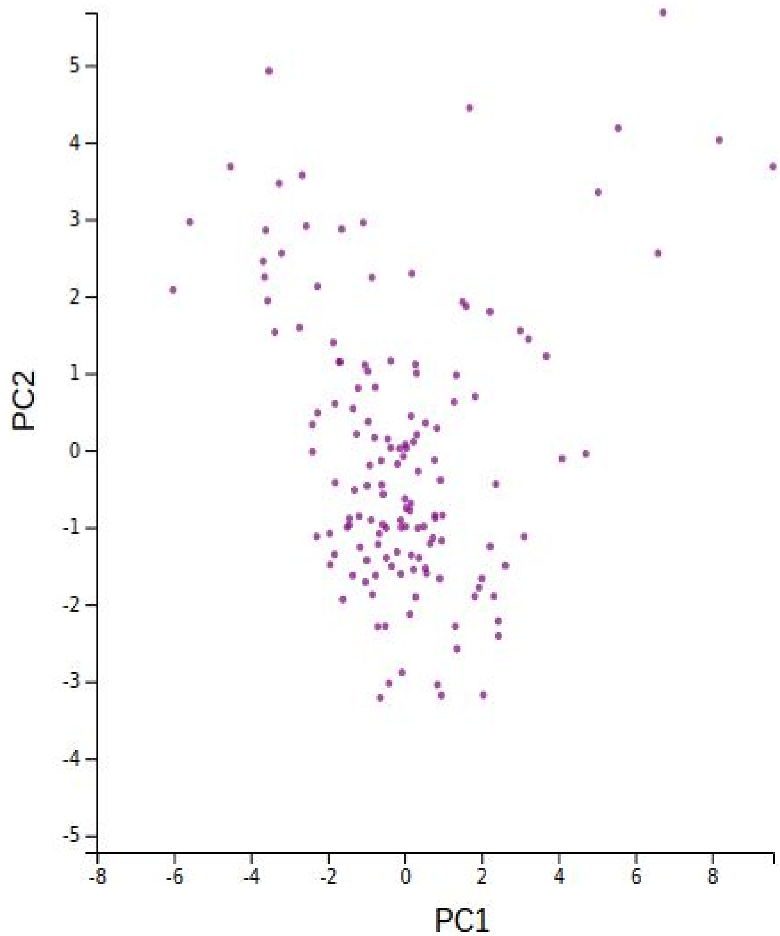
1. *Visualize the data projected into the top two PCA vectors via 2D scatterplot.*
2. *Visualize the data via MDS (Euclidian & correlation distance) in 2D scatterplots.*
3. *Visualize the scatterplot matrix of the three highest PCA loaded attributes.*

Now using PCA and components to be 2 , can produce a scatter plot.

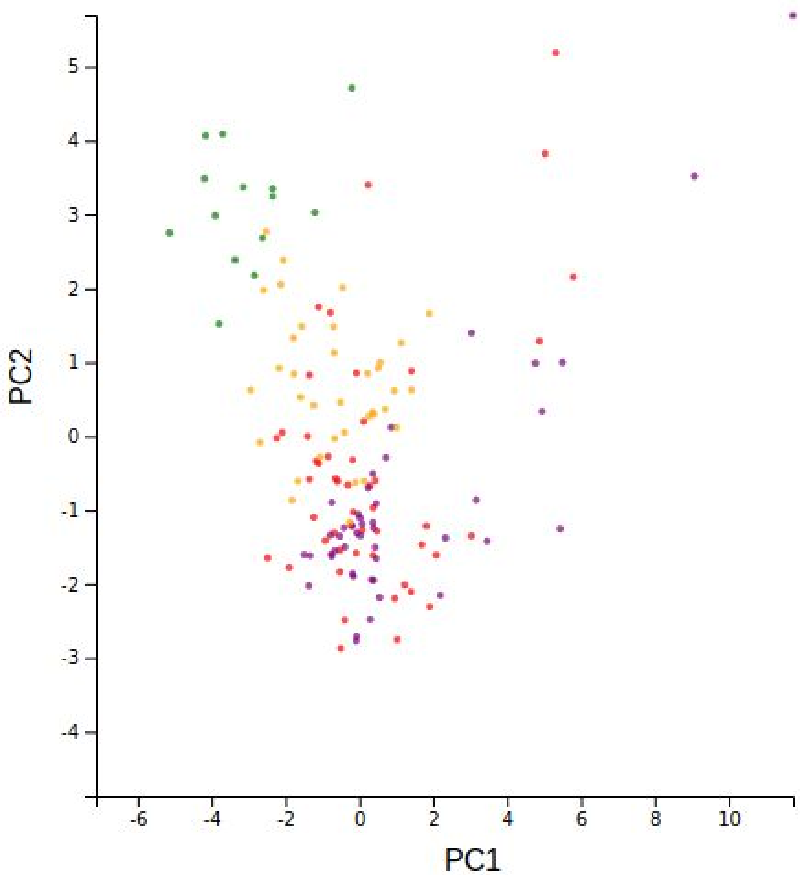
1. Scatter plot original data.



2. Scatter Plot random sampling.



3. Scatter Plot Stratified sampling.

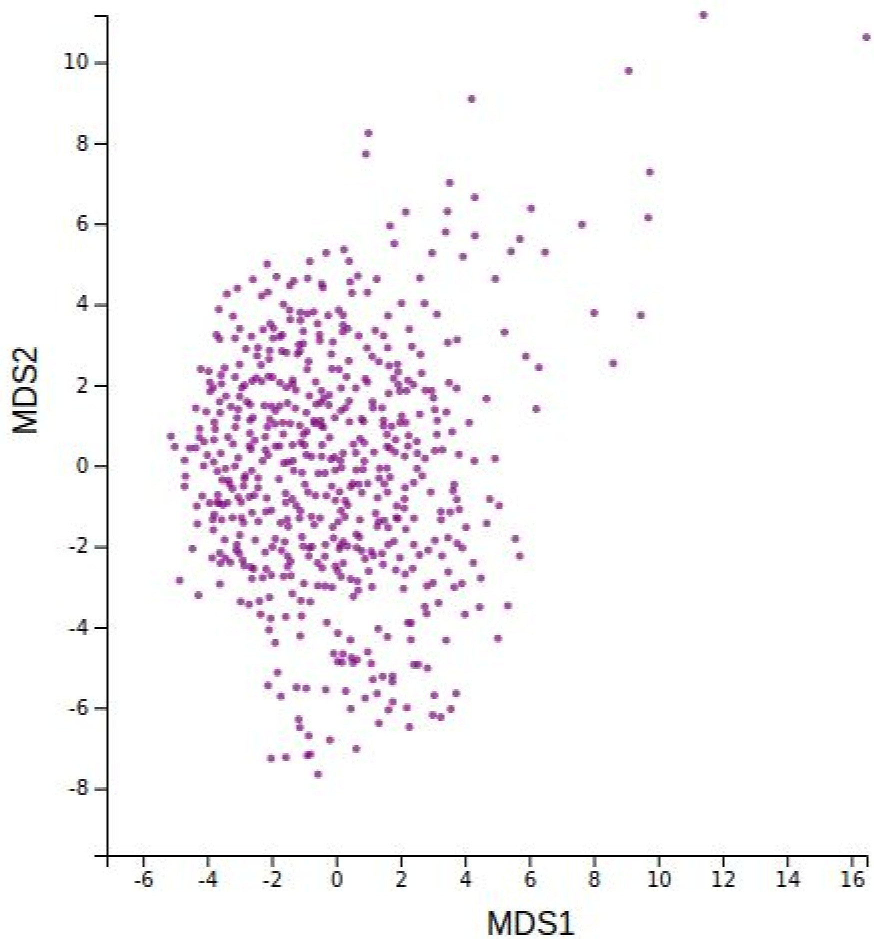


Observation: We can see the data is indeed clustered and looks similar to original data, more clarity about it comes from the fact that the top Pca loadings of the original and stratified data are same.

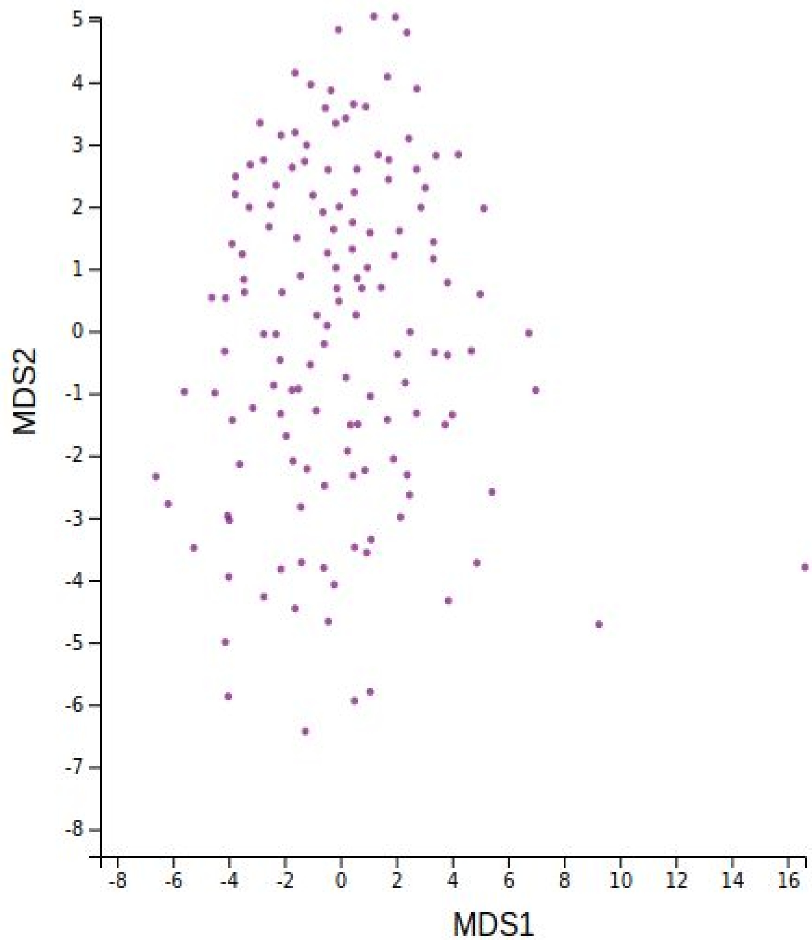
MDS dimension reduction

Euclidean:

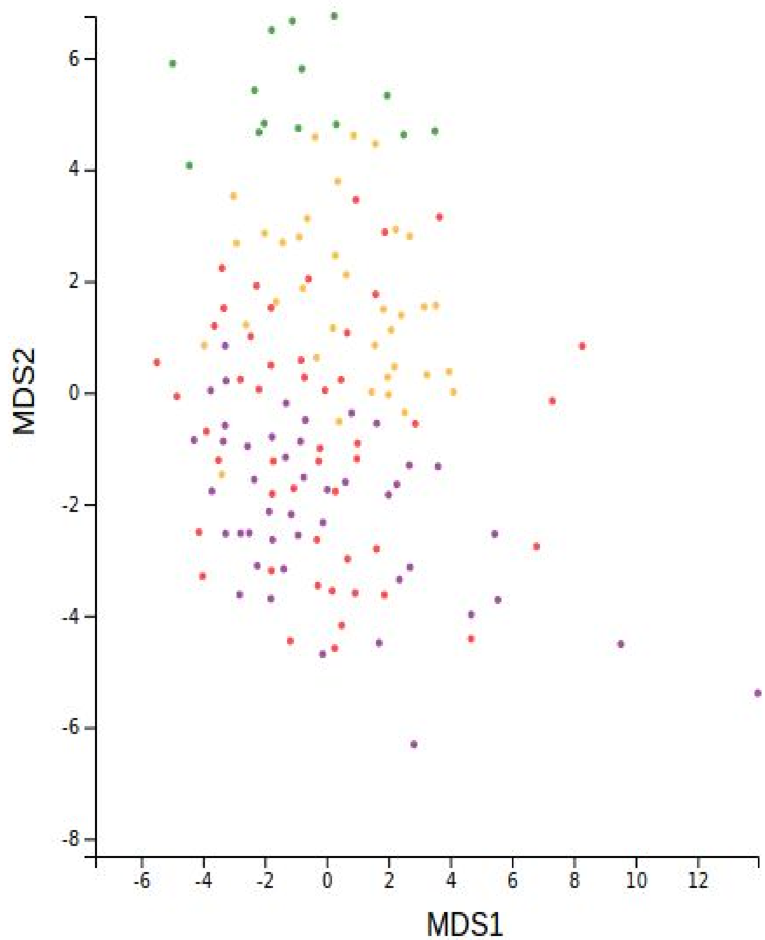
1. Scatter plot Original Data.



2. Scatter Plot random data.

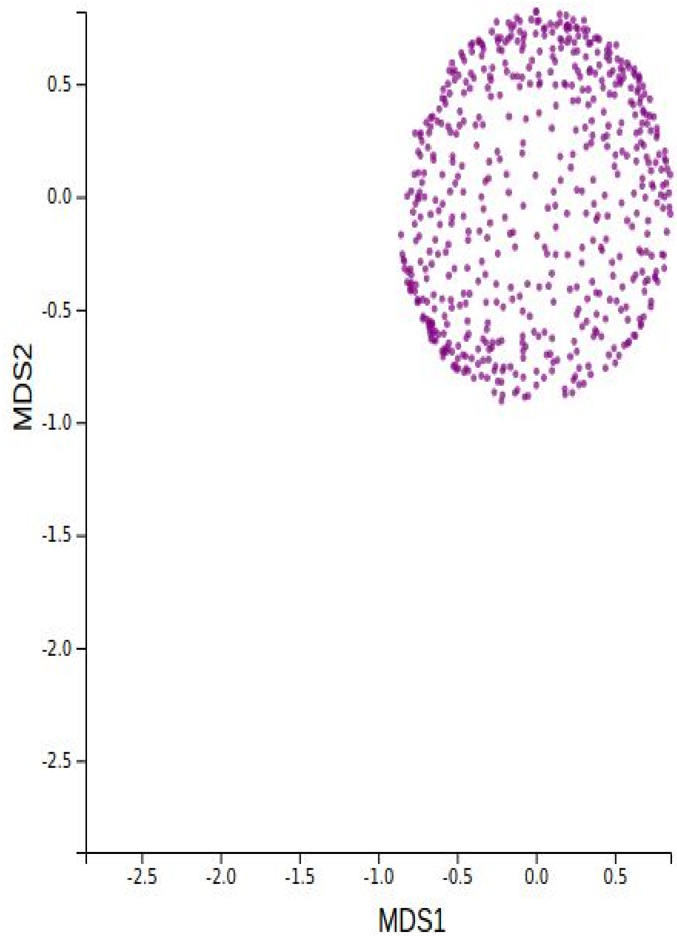


3. Scatter plot Stratified data.

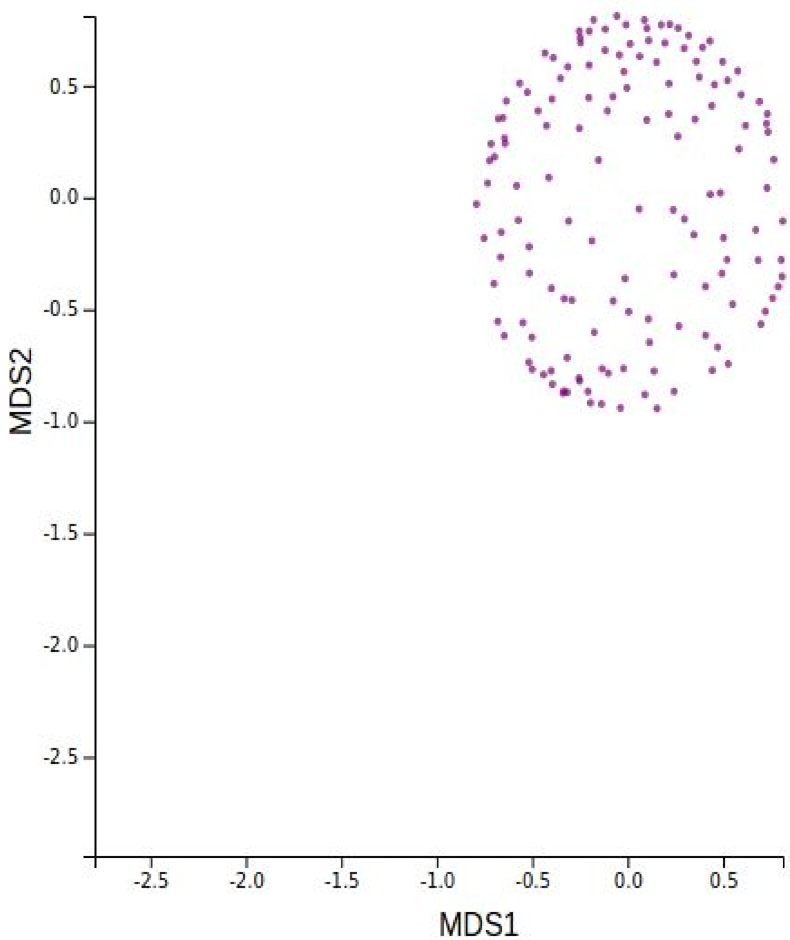


MDS Correlation

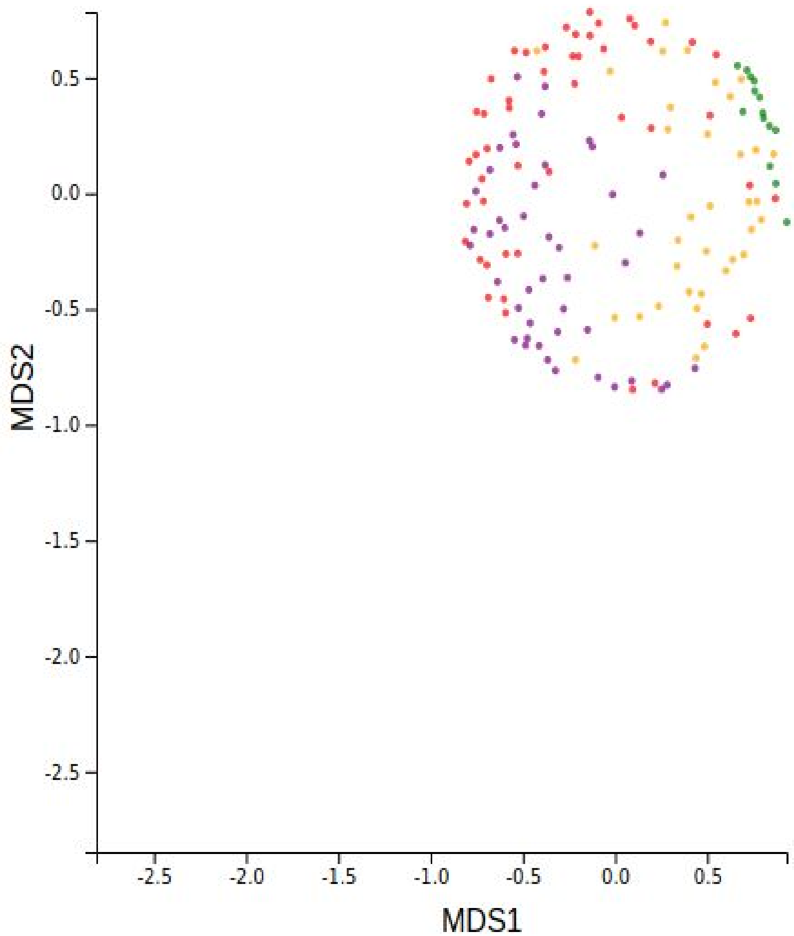
1. Scatter plot original data.



2. Scatter plot random data.



3. Scatter plot stratified data.



Code snippts:

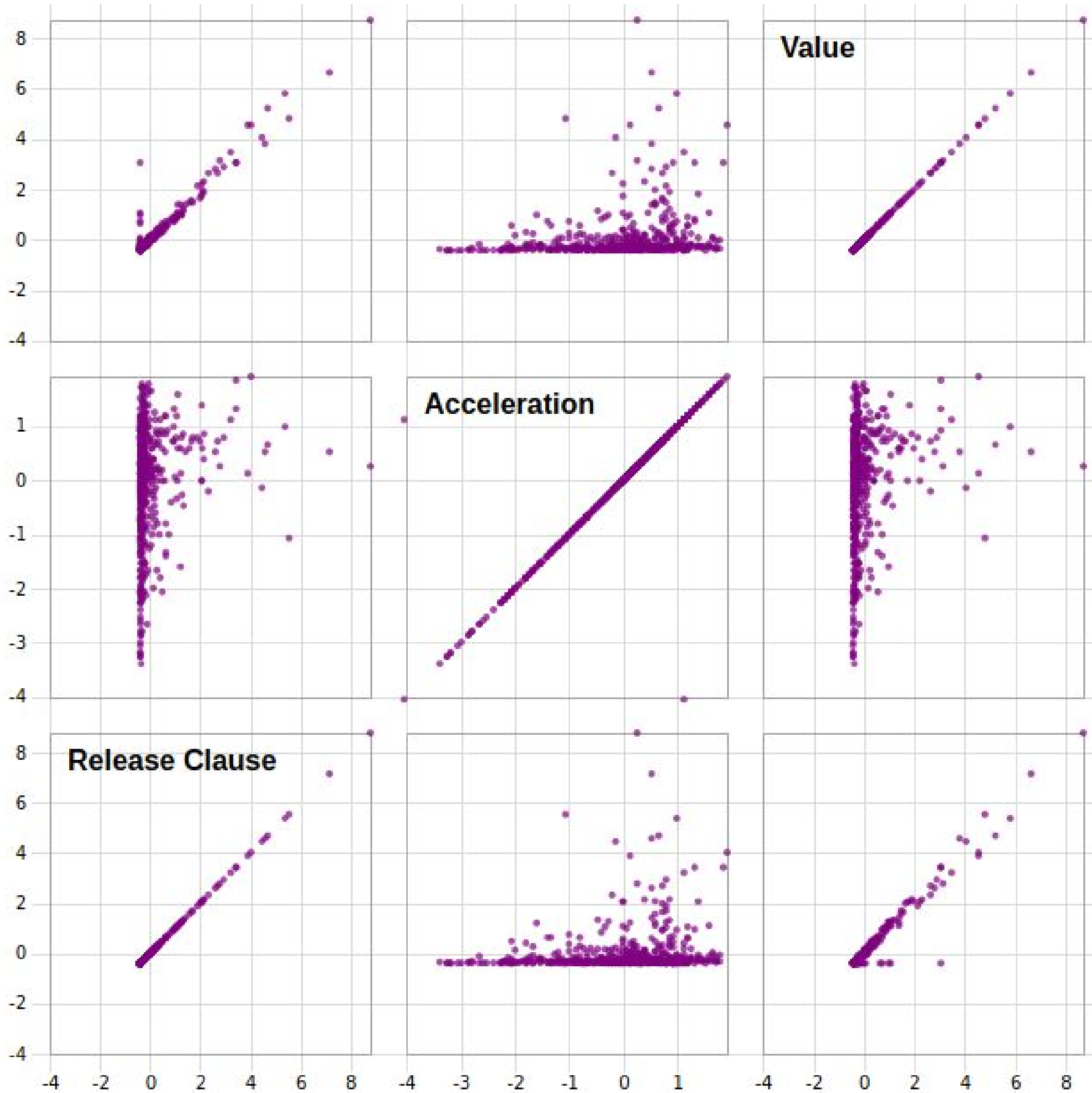
```
#mds Eucledian
def mdsEScatter(sample):

    mat = sample.values
    mat = mat.astype(float)
    mat = StandardScaler().fit_transform(mat)
    mds = manifold.MDS(n_components=2, dissimilarity='precomputed')
    similarity = pairwise_distances(mat, metric='euclidean')
    X = mds.fit_transform(similarity)
    return X
```

```
#mds correlation
def mdsCScatter(sample):

    mat = sample.values
    mat = mat.astype(float)
    mat = StandardScaler().fit_transform(mat)
    mds = manifold.MDS(n_components=2, dissimilarity='precomputed')
    similarity = pairwise_distances(mat, metric='correlation')
    X = mds.fit_transform(similarity)
    return X
```

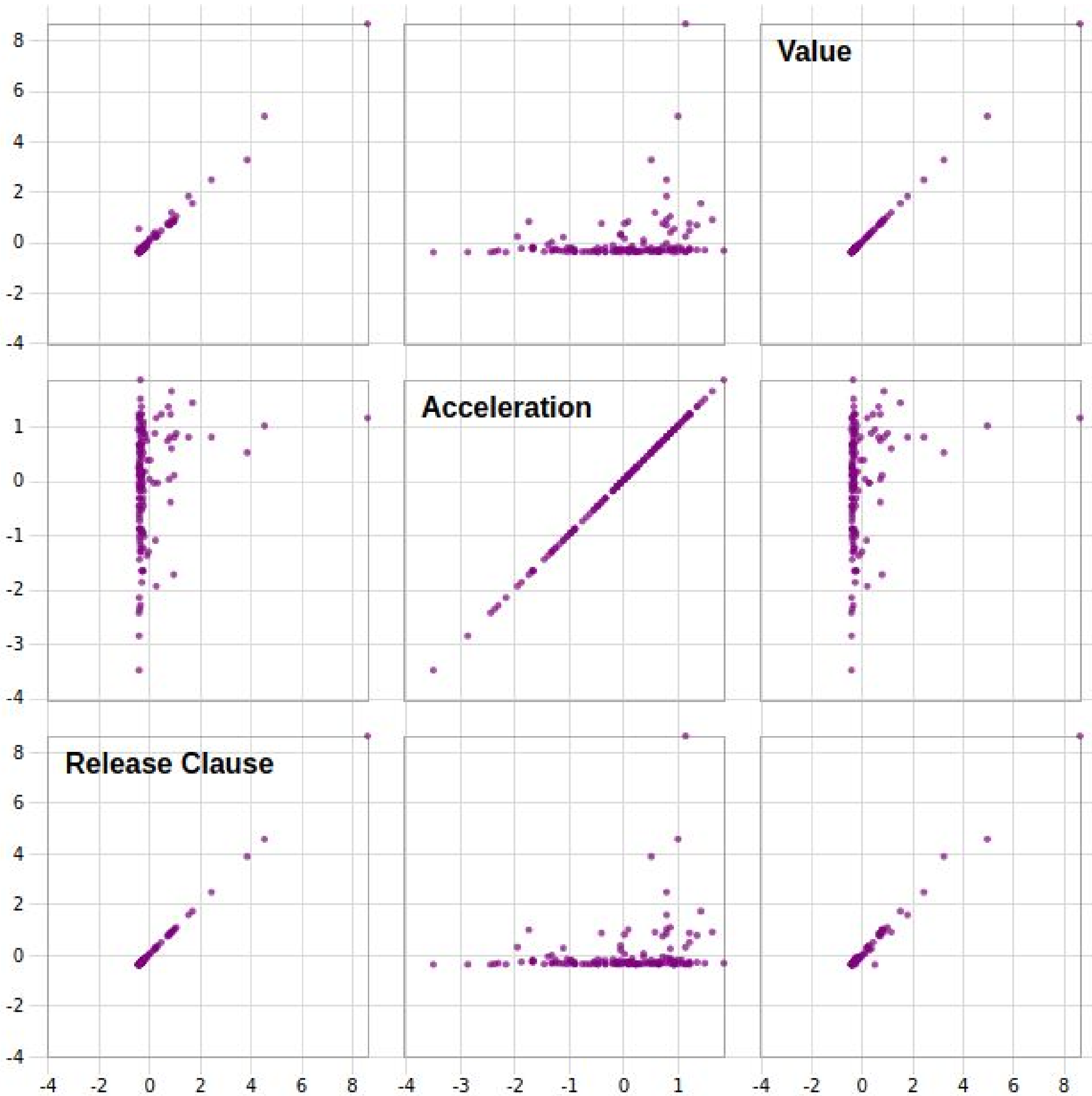
For pca the same techniques as in part 2 is used.

Scatter matrix:

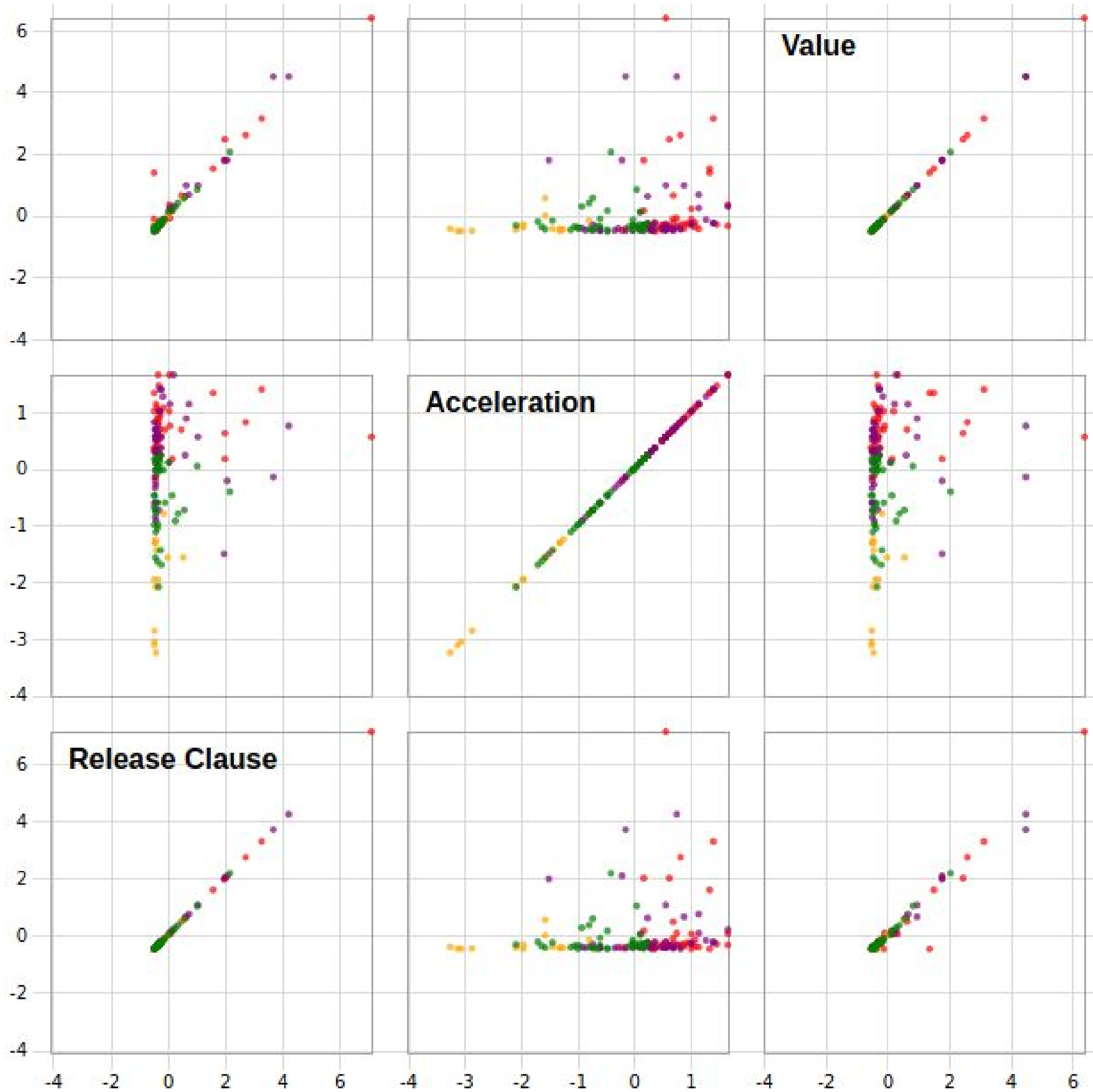
1. Scatter matrix original data.



2. Scatter matrix Random data.



3. Scatter Matrix stratified data.



Observation : A positive correlation exist between ‘Release Claue’ and ‘Value’.