**Visualization Lab-2 Report:**

I chose the data set representing the unemployment rate of blue collar worker in a state. It has 4877 number of rows and 18 columns. Since 11 of these columns were not having a numerical value, I have stripped off those columns. Please find the implementation details task wise as follows:

## Task1: data clustering and decimation:

First after reading the data I have performed min-max normalization.

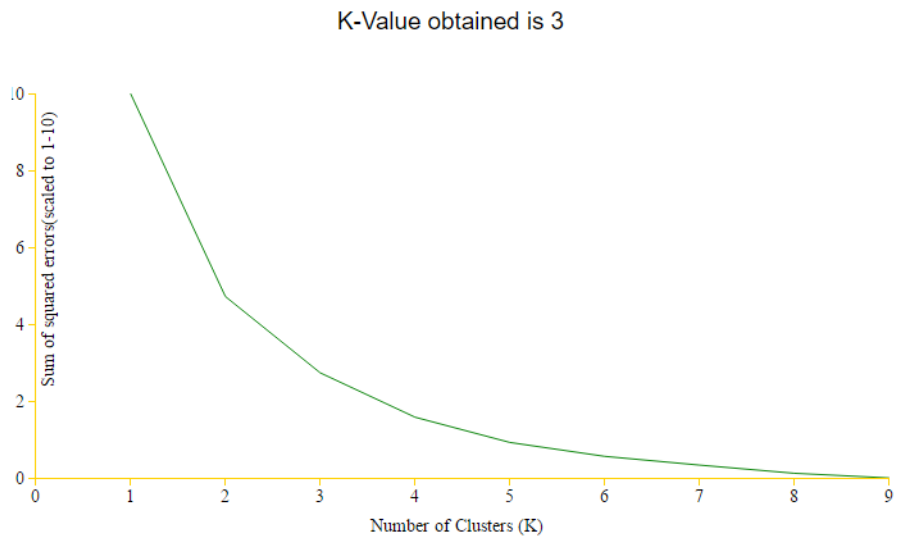
To perform random sampling, I randomly picked the 20% of the data. To perform stratified sampling, I have first generated the scree plot for K=1 to number of columns again SSE and found that the best k for the given data is 3. After clustering the data into three cluster I have chosen equal amount of sample from each of the sets.

here is the sample code.

```
def random_sampling(dataFrame,fraction):
    '''A method to generate random sample by taking fraction of the given samples'''
    #print(dataFrame.index)
    rows = random.sample((list)(dataFrame.index), (int)(len(dataFrame)*fraction))
    return dataFrame.ix[rows]
    #print(dataFrame)

def stratified_sampling(dataFrame,fraction,clust_count):
    '''Method to perform clustering and then sampling, Cluster generation is done using scikit library of
    python'''
    k_cluster=Kcluster.KMeans(n_clusters=clust_count).fit(dataFrame)
    total_len= len(dataFrame)
    total=total_len *fraction
    t1=total/3
    t2=t1
    t3=t1
    strate_sample=[]
    for i in range(total_len):
        if k_cluster.labels_[i]==0 and t1>0:
            strate_sample.append(dataFrame.ix[i])
            t1-=1
        elif k_cluster.labels_[i]==1 and t2>0:
            strate_sample.append(dataFrame.ix[i])
            t2-=1
        elif k_cluster.labels_[i]==2 and t3>0:
            strate_sample.append(dataFrame.ix[i])
            t3-=1
        else :
            break;
```

```
def plot_elbow(dataFrame):
    '''Method to produce k clusters and check SSE values'''
    sse=[]
    for i in range(1,10):
        k_cluster=Kcluster.KMeans(n_clusters=i).fit(dataFrame)
        sse.append(-k_cluster.score(dataFrame))
    out = open(data_dir+'elbow.csv', 'w')
    i=1
    min_max=preprocessing.MinMaxScaler(feature_range=(0,10))
    std_sse=min_max.fit_transform(sse)
    #print(std_sse)
    out.write("K,SSE\n")
    for row in std_sse:
        out.write('%d,%f' % (i,row))
        out.write('\n')
        i+=1;
    out.close()
```

And here is the plot of the k-means plotted for different k to obtain the elbow.



## Task 2: dimension reduction (use decimated data)

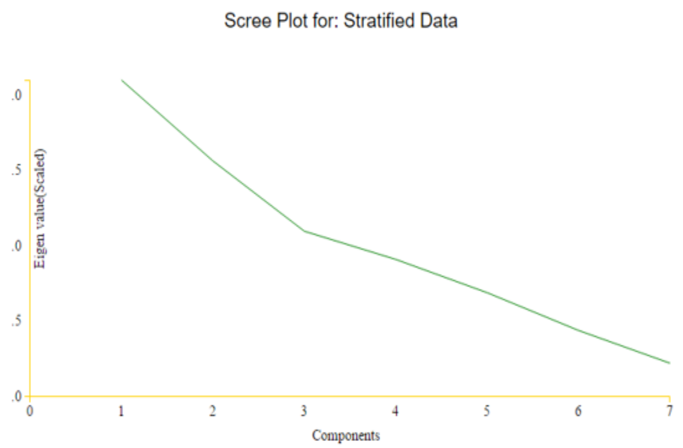
- find the intrinsic dimensionality of the data using PCA
- produce scree plot visualization and mark the intrinsic dimensionality
- obtain the three attributes with highest PCA loadings

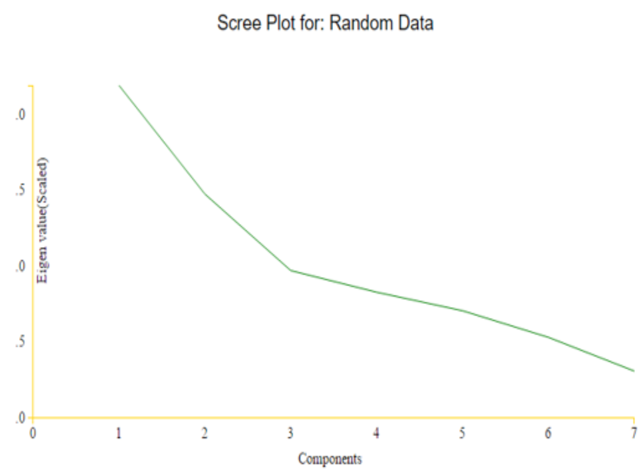
To obtain the intrinsic dimensionality of I have calculated the eigen values of all the components and plotted them to obtain the scree plot. Following is the code snippet

```
#eigen values for a particular data column
def scree(dataframe):
    '''generating sum of '''
    #print(adaptiveSample)
    X_std = StandardScaler().fit_transform(dataframe)
    cor_mat1 = np.corrcoef(X_std.T)
    eig_vals, eig_vecs = np.linalg.eig(cor_mat1)
    sorted(eig_vals,reverse=True)
    return eig_vals
```

```
def generate_loadings(dataframe):
    '''Method creates two outputs,
    First it calculates the sum of squared loading for all the components and
    then selects the top three components out of them'''
    std_input = StandardScaler().fit_transform(dataframe)
    pca = PCA(n_components=3)
    pca.fit_transform(std_input)

    loadings = pca.components_
    squared_loadings = []
    a = np.array(loadings)
    a = a.transpose()
    for i in range(len(a)):
        squared_loadings.append(np.sum(np.square(a[i])))
    df_attributes = pd.DataFrame(pd.DataFrame(dataframe).columns)
    df_attributes.columns = ["attributes"]
    df_sqL = pd.DataFrame(squared_loadings)
    df_sqL.columns = ["s_loadings"]
    sample = df_attributes.join([df_sqL])
    sample = sample.sort_values(["s_loadings"], ascending=[False])
    #saving all the loadings with columns
    sample.to_csv(data_dir+"loadings.csv", sep=',')
    top3 = sample.head(n = 3)
    lst= top3['attributes'].values.tolist()
    min_max=preprocessing.MinMaxScaler(feature_range=(0,10))
    df_top=dataframe.ix[:, lst]
    #save top3 attributes data in file
    std_np_top=min_max.fit_transform(df_top)
    std_df_top=pd.DataFrame(std_np_top)
    #np.savetxt("3loadings.csv", np_col, delimiter=",")
    std_df_top.to_csv(data_dir+"3loadings.csv", sep=',',index=False)
    #return sample
```

Here is the scree plot visualization:



Here is the plot of “sum of squared loading of all the components”.

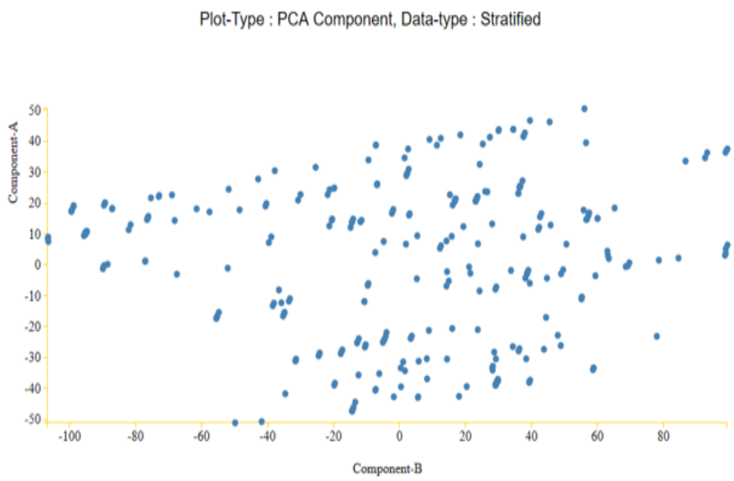


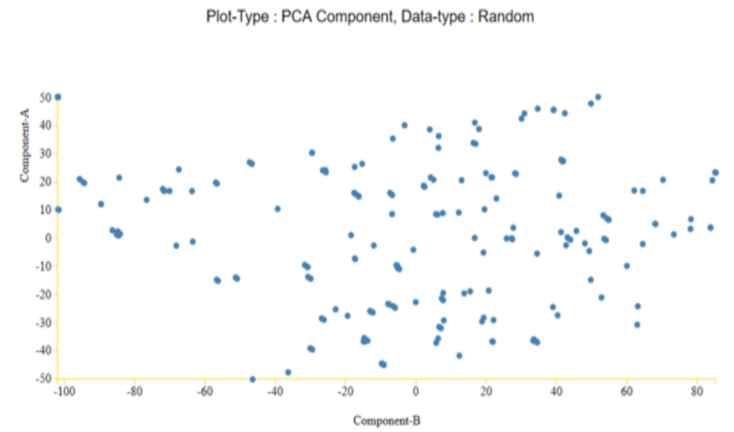
## Task 3: visualization (use dimension reduced data)

- 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

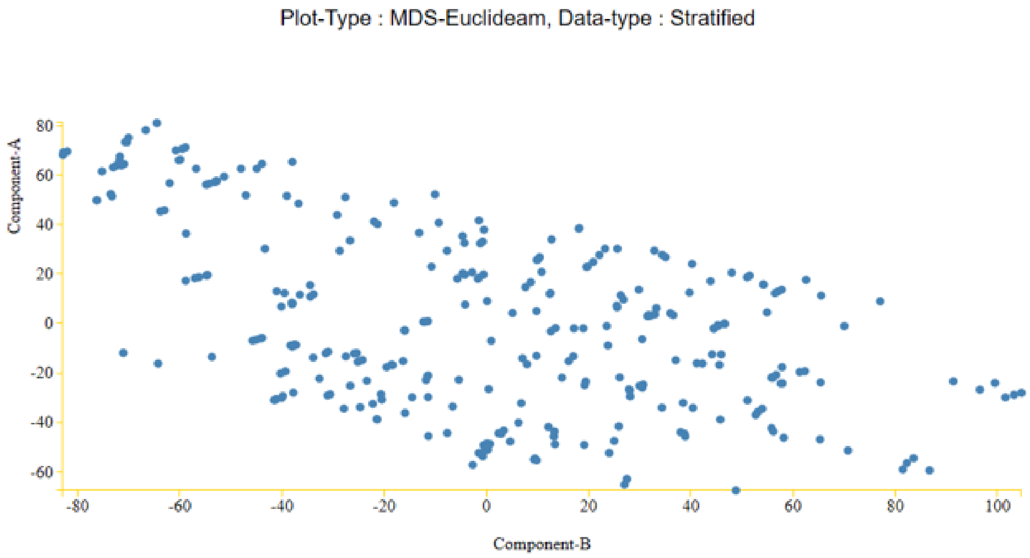
I have used the standard python library to compute the top two PCA components of the data.

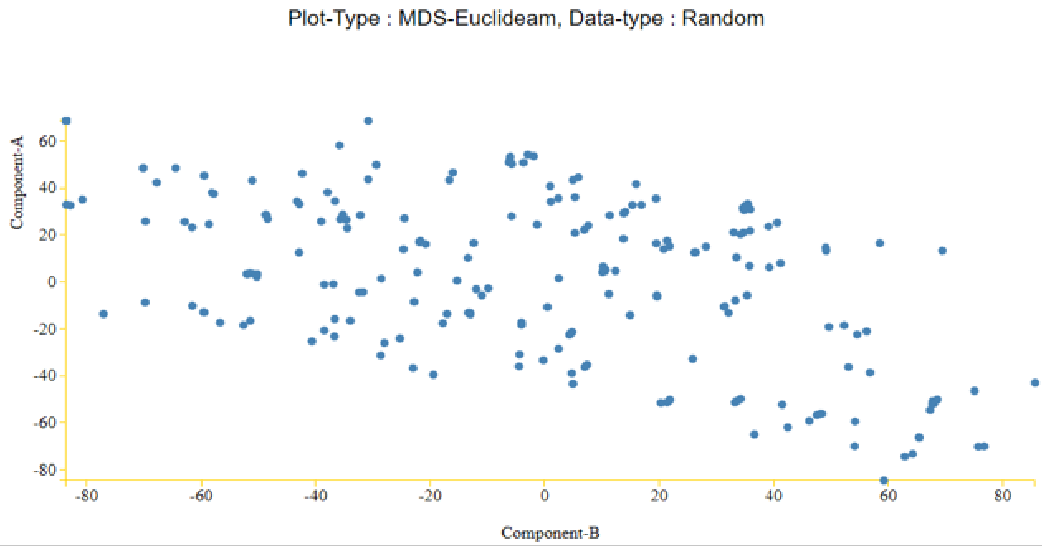
```javascript
//redraw will load the data random/strat for PCA/MDS based on the options selected
function redraw(file,typed)
{

  fileName=file;
 type=typed;

document.getElementById('chart').innerHTML='';
document.getElementById('matrix').innerHTML='';
document.getElementById("screechart").innerHTML="";
document.getElementById("loadings").innerHTML="";

var f,t;
if(file=="pca")
    f="PCA Component";
else if(file=="correlation")
    f="MDS-Correlation";
else if(file=="euclidean")
    f="MDS-Euclideam";
if(type=="strat")
    t="Stratified";
else
    t="Random";

var comment="Plot-Type : "+f+",  Data-type : "+t;
var para= document.createElement('p');
para.textContent=comment;
document.getElementById('chart').appendChild(para);


var arr=[];
 d3.csv("static/"+file+".csv",function(data){
    var i=0;
    data.forEach(function(d){
    //console.log(d);
    arr.push([]);
    arr[i].push(new Array(2));
    if(type=="random")
    {
      arr[i][0]=parseFloat(d.a1);
      arr[i][1]=parseFloat(d.a2);
    }
    else
    {
      arr[i][0]=parseFloat(d.r1);
      arr[i][1]=parseFloat(d.r2);
    }
```



Similarly here are the plot for MDS-Eculidean:

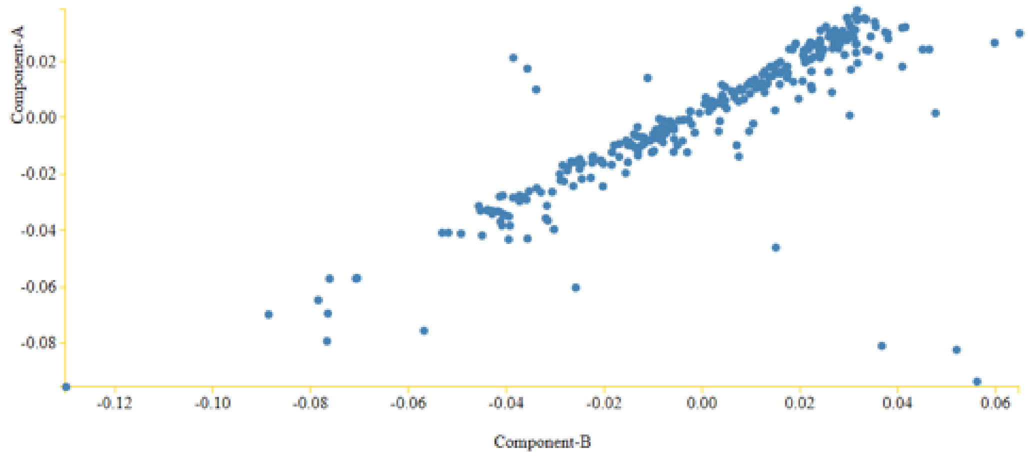


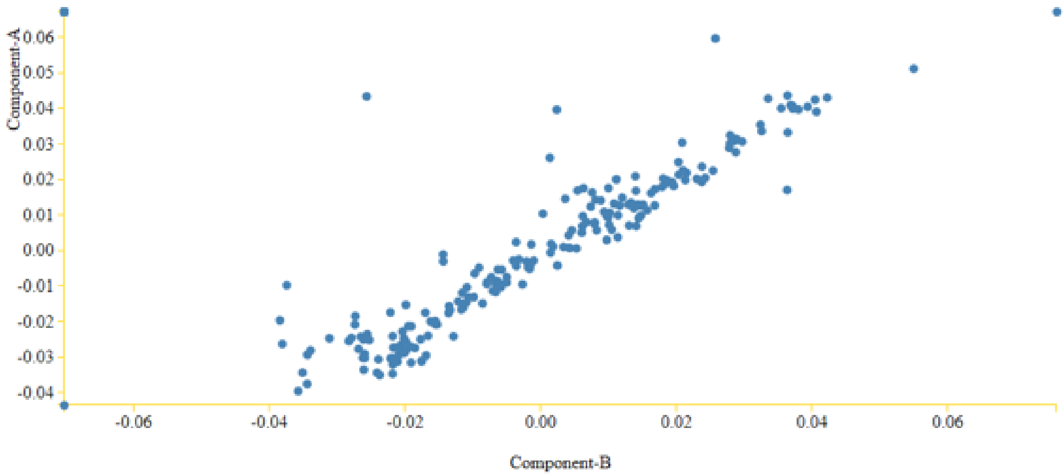

```python
def find_mds(dataframe, type):
    '''Method to computer MDS for different distance type'''

    dis_mat = SK_Metrics.pairwise_distances(dataframe, metric = type)
    mds = MDS(n_components=2, dissimilarity='precomputed')
    return pd.DataFrame(mds.fit_transform(dis_mat))
```

Similary, MDS-Correlation:

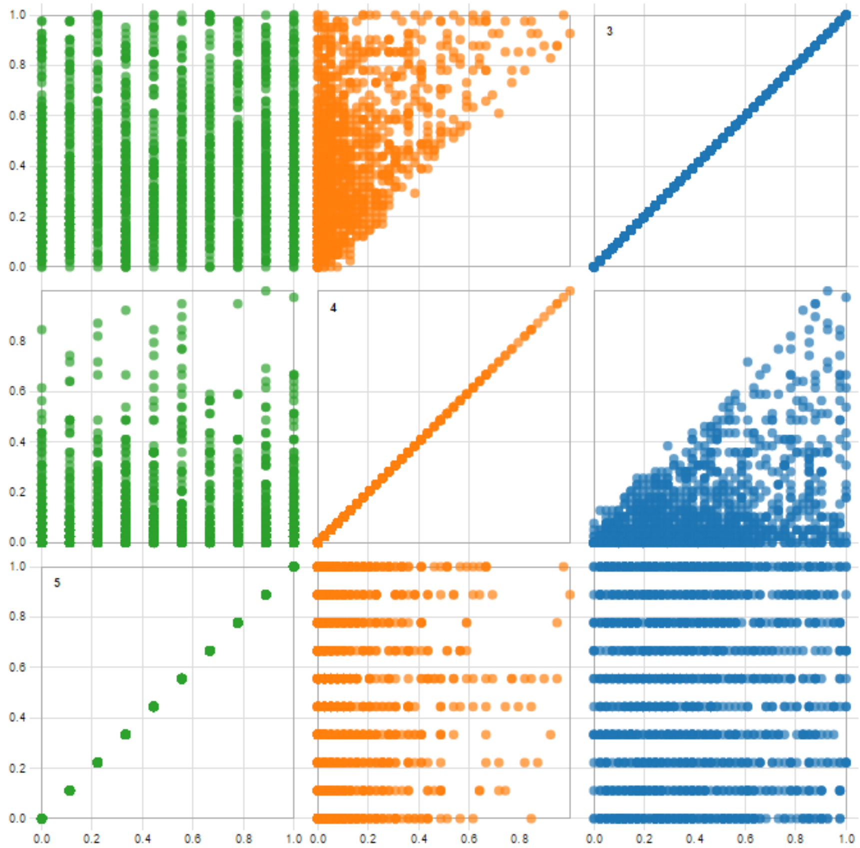
Plot-Type : MDS-Correlation, Data-type : Stratified



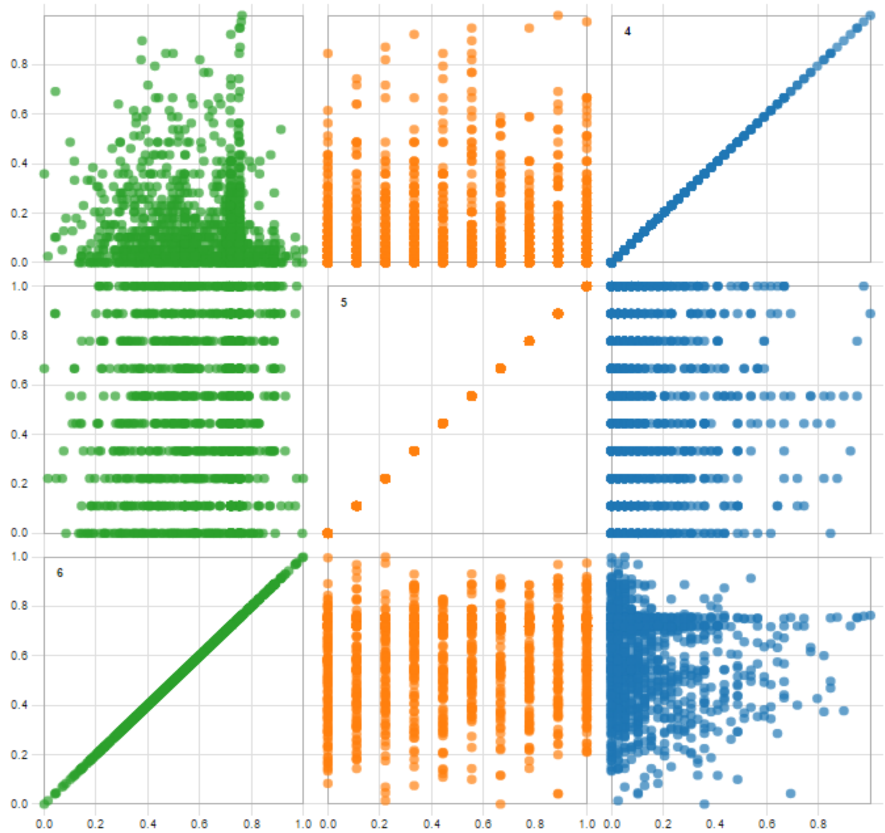

Plot-Type : MDS-Correlation, Data-type : Random



And finally, I plotted Scatter-Matrix Plot of the top-3 loadings for both the cases that Stratified and random :




Scatter-Matrix plot for : Random Data-samples



Flow of program is shown below:

```python
    #generating normalized data samples
    min_max_scaler = preprocessing.MinMaxScaler()
    np_scaled = min_max_scaler.fit_transform(data)
    data = pd.DataFrame(np_scaled)
    #generate the random sample
    rand_sample=random_sampling(data,0.2)

    plot_elbow(data) # based on elbow value of k is chosen

    #k-means cluster+ sampling
    strat_sample=stratified_sampling(data,0.2,3)

    #get the scree plot to check top PCA component
    write_to_file(scree(rand_sample),data_dir+"scree_random.csv")
    write_to_file(scree(strat_sample),data_dir+"scree_strat.csv")

     #generate loading for strat samples
    squared_loading=generate_loadings(strat_sample,"strat")

    #find top3 in strat samples and write to file
    top_attr=getTop3(squared_loading,"strat")
    data.ix[:, top_attr].to_csv(data_dir+"top_strat.csv", sep=',')

    #generate loading for random samples
    squared_loadings=generate_loadings(rand_sample,"random")

    top_attr= getTop3(squared_loadings,"random")
    data.ix[:, top_attr].to_csv(data_dir+"top_random.csv", sep=',')

    #generate PCA data
    createFile(find_pca(rand_sample),find_pca(strat_sample),"pca")

    #generate mds data
    mds_list= ["euclidean","correlation"]
    for type_mds in mds_list:
        createFile(find_mds(rand_sample,type_mds),find_mds(strat_sample,type_mds),type_mds)

    print("done")


app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")
if __name__ == "__main__":
    main()
    app.run(host='127.0.0.1',port=5001,debug=True)
```

I have used Flask for creating a webserver. This application works on Port 5001.

Sample code for Matrix-Scatterplot:

```
svg.selectAll(".y.axis")
    .data(traits)
  .enter().append("g")
    .attr("class", "y axis")
    .attr("transform", function(d, i) { return "translate(0," + i * size + ")"; })
    .each(function(d) { y.domain(domainByTrait[d]); d3.select(this).call(yAxis); });

var cell = svg.selectAll(".cell")
    .data(cross(traits, traits))
  .enter().append("g")
    .attr("class", "cell")
    .attr("transform", function(d) { return "translate(" + (n - d.i - 1) * size + "," + d.j * size + ")"; })
    .each(plot);

// Titles for the diagonal.
cell.filter(function(d) { return d.i === d.j; }).append("text")
    .attr("x", padding)
    .attr("y", padding)
    .attr("dy", ".71em")
    .text(function(d) { return d.x; });

function plot(p) {
  var cell = d3.select(this);

  x.domain(domainByTrait[p.x]);
  y.domain(domainByTrait[p.y]);
  p.color;
  cell.append("rect")
      .attr("class", "frame")
      .attr("x", padding / 2)
      .attr("y", padding / 2)
      .attr("width", size - padding)
      .attr("height", size - padding);

  cell.selectAll("circle")
      .data(data)
    .enter().append("circle")
      .attr("cx", function(d) { return x(d[p.x]); })
      .attr("cy", function(d) { return y(d[p.y]); })
      .attr("r", 4)
      .style("fill", function(d) { return color(p.x); });
  }
});

function cross(a, b) {
  var c = [], n = a.length, m = b.length, i, j;
  for (i = -1; ++i < n;) for (j = -1; ++j < m;) c.push({x: a[i], i: i, y: b[j], j: j});
  return c;
}
```

Code for Scree Line plots:

```
// Define the line
var valueline = d3.svg.line()
    .x(function(d,i) { return x(i+1); })
    .y(function(d) { return y(d.K); });
    //.interpolate("basis"); smoothening

// Adds the svg canvas
var svg = d3.select("#screechart")
    .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
      // .attr("id", "scatter")
       .style("margin-left", "2em")
    .append("g")
        .attr("transform",
              "translate(" + margin.left + "," + margin.top + ")");

// Get the data
d3.csv("static/scree_"+file+".csv", function(error, data) {
    data.forEach(function(d) {
        d.K = +d.Eigen_val;

    });

   x.domain([0, data.length]);
   y.domain([0,d3.max(data, function(d) { return d.K; }) ]);

    // Add the valueline path.
    svg.append("path")
        .data([data])
        .attr("class", "line")
        .attr("d", valueline)
        .attr('stroke','green')
        .attr("fill", "none");
    // Add the X Axis
    svg.append("g")
        .attr("class", "x axis")
        .attr("transform", "translate(0," + height + ")")
        .call(xAxis);

    // Add the Y Axis
    svg.append("g")
        .attr("class", "y axis")
        .call(yAxis);

    //put labels
      svg.append("text")
      .attr("transform", "rotate(-90)")
        .attr("y", -60)
        .attr("x", -130)
        .attr("dy", "5em")
        .style("text-anchor", "middle")
        .attr("id", "yLabel")
```

References:


- Elbow in python - http://stackoverflow.com/questions/41540751/sklearn-kmeans-equivalent-of-elbow-method
- Find elbow from graph - http://www.analyticbridge.com/profiles/blogs/identifying-the-number-of-clusters-finally-a-solution
- Spree plot to select principal components - https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/
- loadings - http://stackoverflow.com/questions/21217710/factor-loadings-using-sklearn