CSE 564 Visualization, Spring 2020, Lab 2

Tianao Wang 112819772

April 5, 2020

1 Youtube Link

Youtube Link

2 Google Drive Link

Please use stony brook edu mail to open the google drive link!
Google Drive Link

3 Data Preprocessing

The data I use is all Pokémon stats until generation 6 which is a famous IP in Nintendo. The data file has 11 columns and 721 rows. I do some data preprocessing to make the output graph more beautiful.

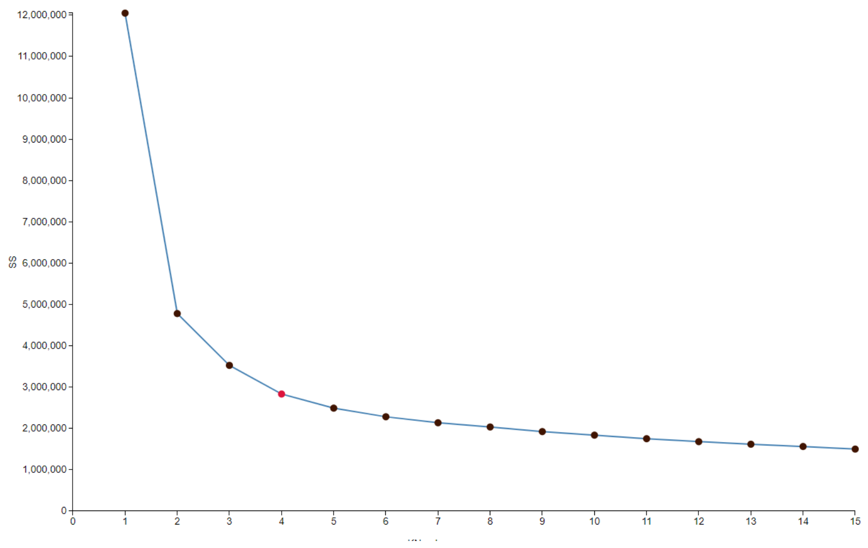
(1) I change some categorical data into numerical variables.

(2) I give up some data which are not useful for the anlysis such as id and name.

4 Program's Capabilities

4.1 k-means elbow

I use line chart to show the sum of square error of different K values in K-Means. Then I find the elbow which is shown in red in the graph and decide the k into 4. Using this K=4, I can do the stratified sampling.

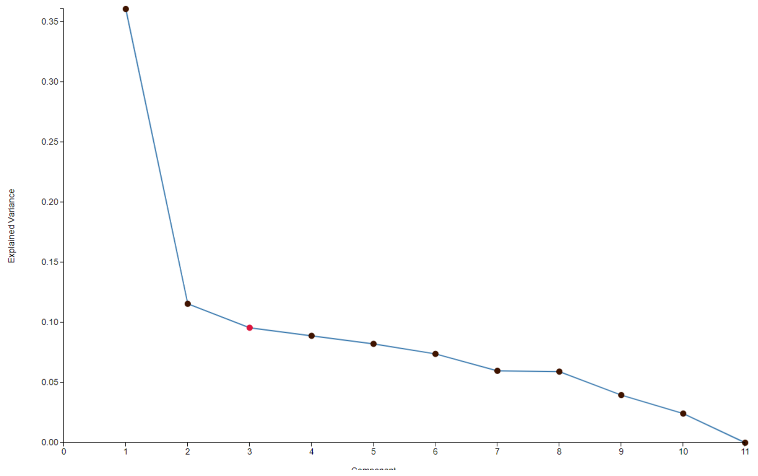


4.2 Sampling

I implement the random sampling and stratified sampling. Random sampling is very simple, I just need to remove the 75% of original data. When doing stratified sampling, I first categorize the data into 4 groups(according to the kmenas elbow). Then I remove 75% data of each group and then add them into the final result.

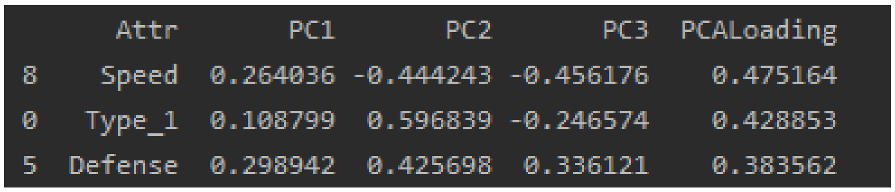
4.3 Draw scree plot and find the intrinsic dimensionality

I change PCA parameter n_components into different values and find the reasonable intrinsic dimensionality. I use the kaiser rule the get the intrinsic dimensionality and mark it in the red.



4.4 Three attributes with highest PCA loadings

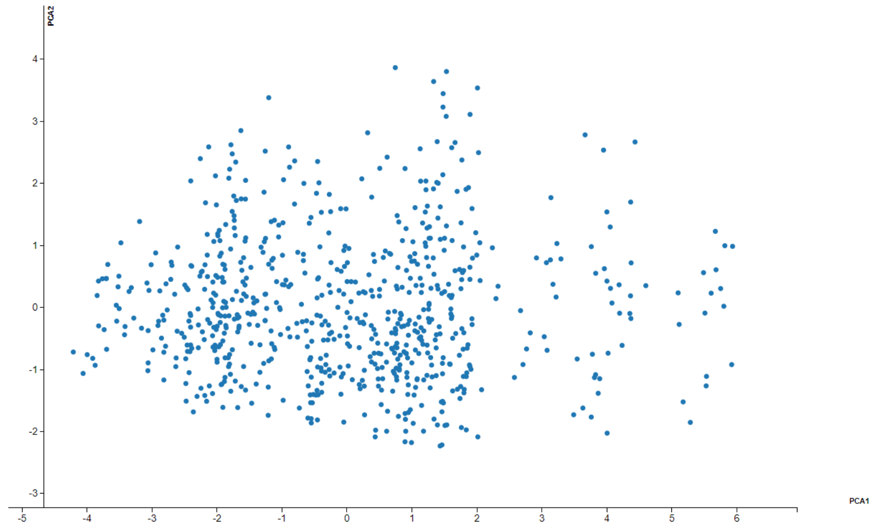
I let n_components=3 and calculate the PCALoading by sum of square of PC1,PC2,PC3. Then sort the result and show the top three highest PCA loading.



```
     Attr       PC1       PC2       PC3  PCALoading
8    Speed  0.264036 -0.444243 -0.456176    0.475164
0   Type_1  0.108799  0.596839 -0.246574    0.428853
5  Defense  0.298942  0.425698  0.336121    0.383562
```

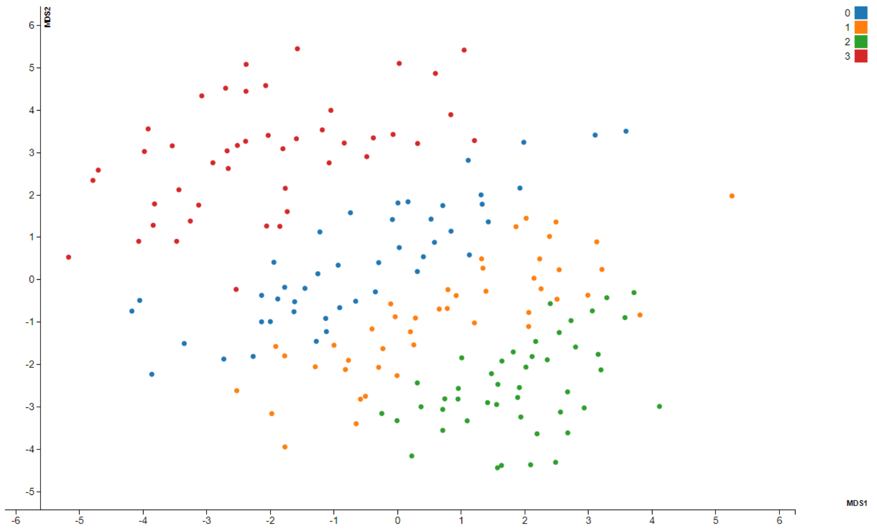
4.5 2D Scatter plot of top two PCA vectors

I use PCA method in sklearn library and let n_components=2 to get the x and y values and use d3 to draw a 2D scatter plot for the data.



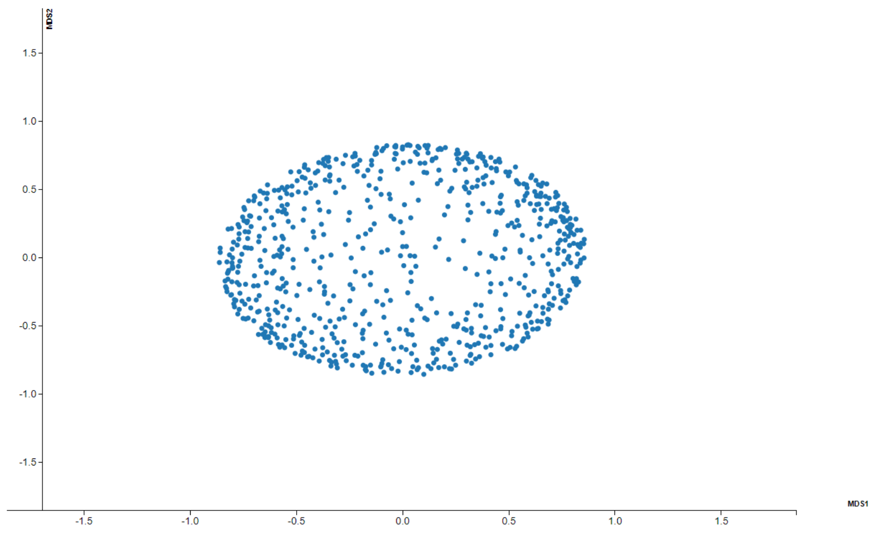
4.6 2D Scatter plot of MDS(Euclidian)

I use MDS method in sklearn libray and let n_components=2,dissimilarity=”euclidean” to get the x and y values to draw the 2D scatter plot.



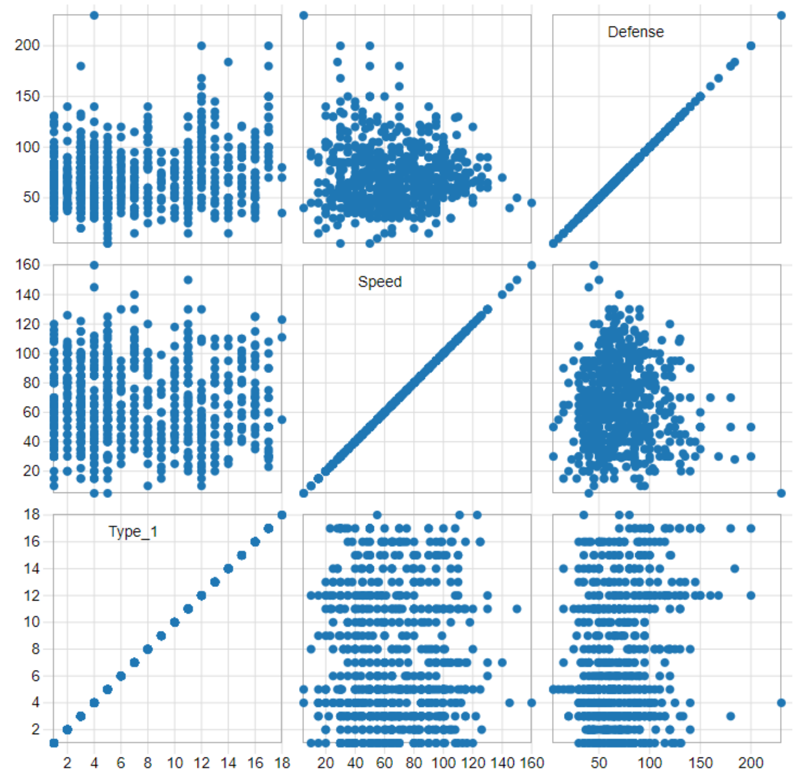
4.7 2D Scatter plot of MDS(Correlation distance)

I calculate the pairwise_distances as a matrix and use it to calculate MDS by the parameter n_components=2,dissimi
Then I use the data to draw the 2D scatter plot.



4.8 Scatter plot matrix of the three highest PCA loaded attributes

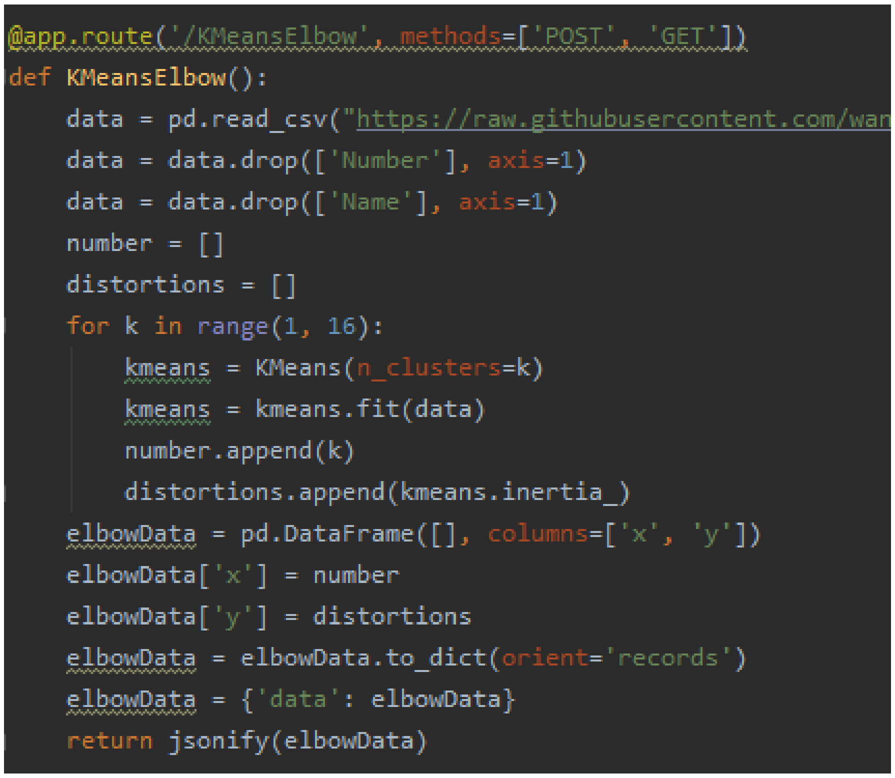
I calculate the top three PCA loading attributes and delete all other attributes. Using these three types data to draw the scatter plot matrix.



5 Implementation Details

5.1 k-means elbow

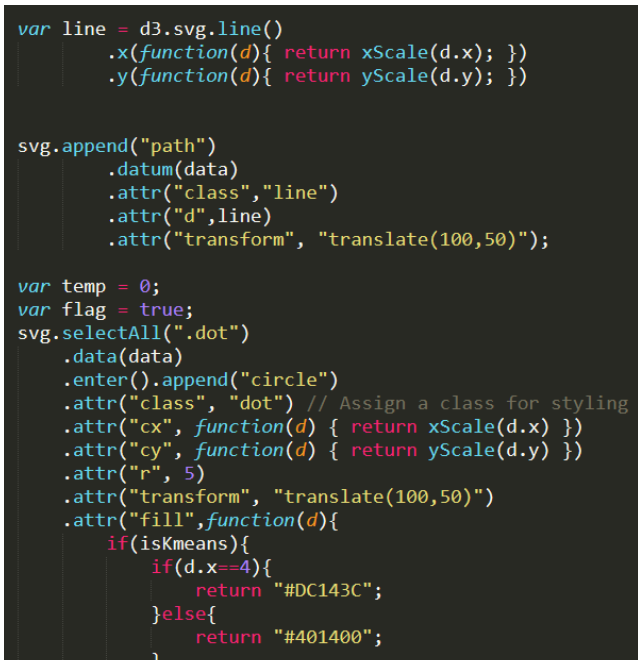
First, I drop some useless data such as id Number and Name. Then I try K values from 1 to 15 and record the inertia of each kmeans. After these processes, I return the result to the website and use d3 to draw a line chart.



```python
@app.route('/KMeansElbow', methods=['POST', 'GET'])
def KMeansElbow():
    data = pd.read_csv("https://raw.githubusercontent.com/wan
    data = data.drop(['Number'], axis=1)
    data = data.drop(['Name'], axis=1)
    number = []
    distortions = []
    for k in range(1, 16):
        kmeans = KMeans(n_clusters=k)
        kmeans = kmeans.fit(data)
        number.append(k)
        distortions.append(kmeans.inertia_)
    elbowData = pd.DataFrame([], columns=['x', 'y'])
    elbowData['x'] = number
    elbowData['y'] = distortions
    elbowData = elbowData.to_dict(orient='records')
    elbowData = {'data': elbowData}
    return jsonify(elbowData)
```

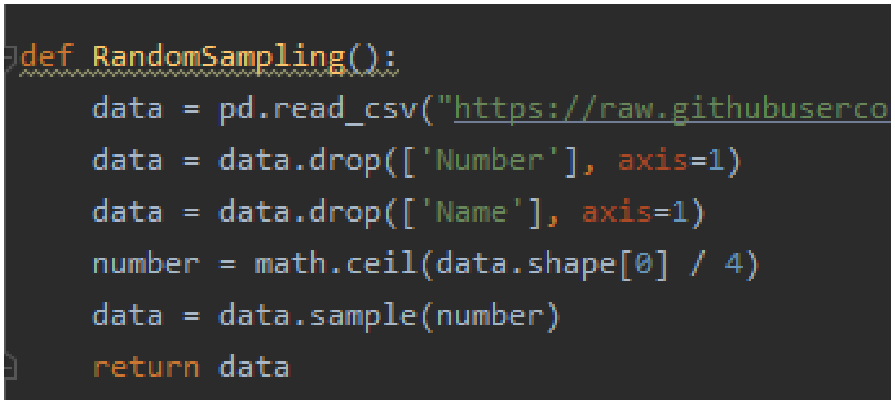
5.2 line chart

Once I get the data from the server, I can call the function DrawLineChart to draw a line chart for the data. I define the scale of x and y, also define the axis of x and y. Then draw the dots according to the data and draw paths between the dots.

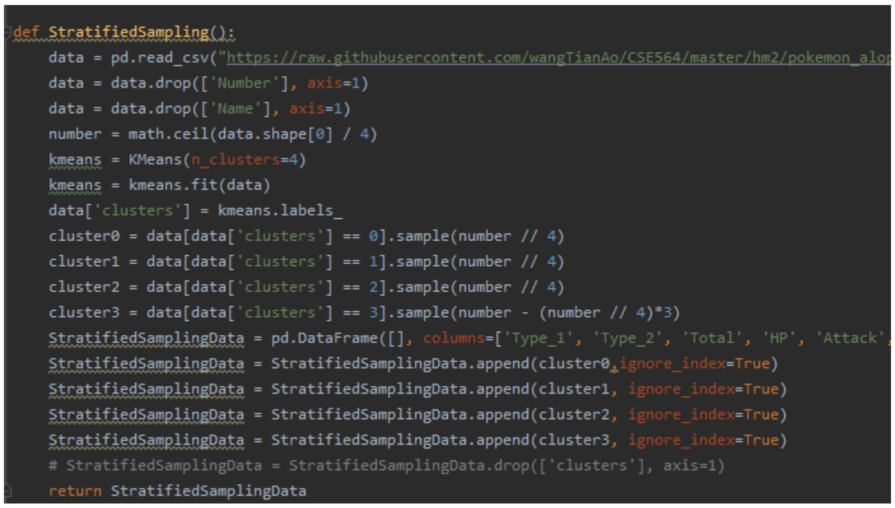

```
var line = d3.svg.line()
        .x(function(d){ return xScale(d.x); })
        .y(function(d){ return yScale(d.y); })


svg.append("path")
        .datum(data)
        .attr("class","line")
        .attr("d",line)
        .attr("transform", "translate(100,50)");

var temp = 0;
var flag = true;
svg.selectAll(".dot")
    .data(data)
    .enter().append("circle")
    .attr("class", "dot") // Assign a class for styling
    .attr("cx", function(d) { return xScale(d.x) })
    .attr("cy", function(d) { return yScale(d.y) })
    .attr("r", 5)
    .attr("transform", "translate(100,50)")
    .attr("fill",function(d){
        if(isKmeans){
            if(d.x==4){
                return "#DC143C";
            }else{
                return "#401400";
```

5.3 Sampling

Random sampling is very easy. I directly use the sample() function to remove 75% data.


```
def RandomSampling():
    data = pd.read_csv("https://raw.githubuserco
    data = data.drop(['Number'], axis=1)
    data = data.drop(['Name'], axis=1)
    number = math.ceil(data.shape[0] / 4)
    data = data.sample(number)
    return data
```

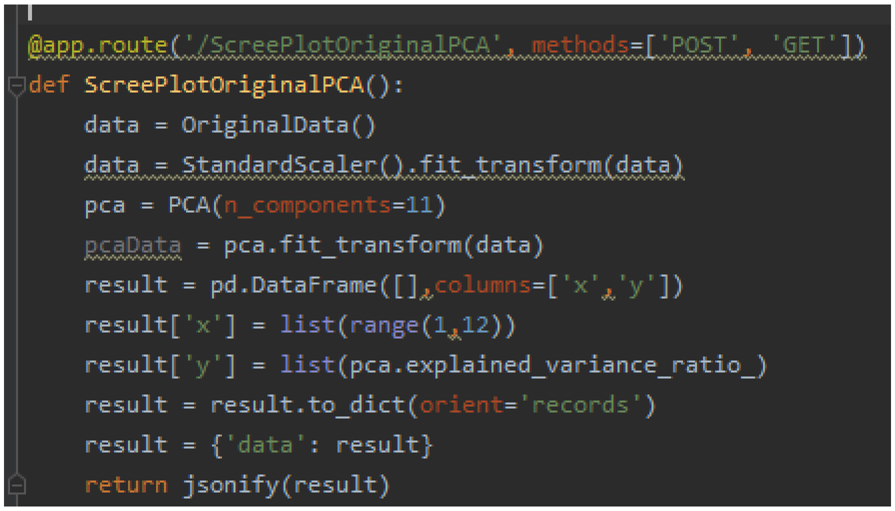
Stratified Sampling need more steps. I divided the data into 4 groups(according to the k means eblow value). In each group, I remove 75% data. Finally, connect these groups into the result.



```
def StratifiedSampling():
    data = pd.read_csv("https://raw.githubusercontent.com/wangTianAo/CSE564/master/hm2/pokemon_alop
    data = data.drop(['Number'], axis=1)
    data = data.drop(['Name'], axis=1)
    number = math.ceil(data.shape[0] / 4)
    kmeans = KMeans(n_clusters=4)
    kmeans = kmeans.fit(data)
    data['clusters'] = kmeans.labels_
    cluster0 = data[data['clusters'] == 0].sample(number // 4)
    cluster1 = data[data['clusters'] == 1].sample(number // 4)
    cluster2 = data[data['clusters'] == 2].sample(number // 4)
    cluster3 = data[data['clusters'] == 3].sample(number - (number // 4)*3)
    StratifiedSamplingData = pd.DataFrame([], columns=['Type_1', 'Type_2', 'Total', 'HP', 'Attack',
    StratifiedSamplingData = StratifiedSamplingData.append(cluster0,ignore_index=True)
    StratifiedSamplingData = StratifiedSamplingData.append(cluster1, ignore_index=True)
    StratifiedSamplingData = StratifiedSamplingData.append(cluster2, ignore_index=True)
    StratifiedSamplingData = StratifiedSamplingData.append(cluster3, ignore_index=True)
    # StratifiedSamplingData = StratifiedSamplingData.drop(['clusters'], axis=1)
    return StratifiedSamplingData
```

5.4 Scree plot and find the intrinsic dimensionality

I use the PCA method to deal with the data and return the explained variance ratio to the client.

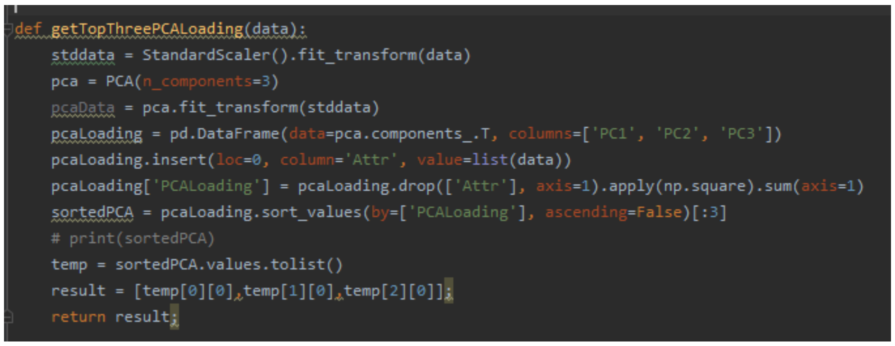


```
@app.route('/ScreePlotOriginalPCA', methods=['POST', 'GET'])
def ScreePlotOriginalPCA():
    data = OriginalData()
    data = StandardScaler().fit_transform(data)
    pca = PCA(n_components=11)
    pcaData = pca.fit_transform(data)
    result = pd.DataFrame([],columns=['x','y'])
    result['x'] = list(range(1,12))
    result['y'] = list(pca.explained_variance_ratio_)
    result = result.to_dict(orient='records')
    result = {'data': result}
    return jsonify(result)
```

At the client, I use d3 to draw the line chart of the returned data. Using Kaiser rule to show the point which represent the intrinsic dimensionality.

5.5 Three attributes with highest PCA loadings

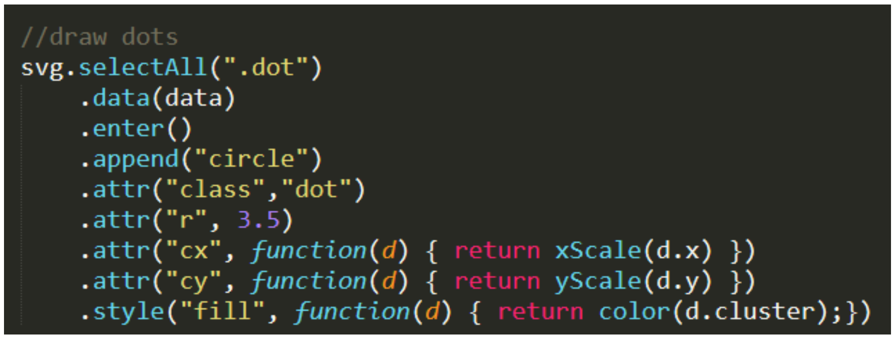
First, I use the intrinsic dimensionality the calculate the PC values and use the sum of square to calculate the pcaLoading of each attribute. After this, I sort the result and return the top3 attributes with highest PCA loadings.



```python
def getTopThreePCALoading(data):
    stddata = StandardScaler().fit_transform(data)
    pca = PCA(n_components=3)
    pcaData = pca.fit_transform(stddata)
    pcaLoading = pd.DataFrame(data=pca.components_.T, columns=['PC1', 'PC2', 'PC3'])
    pcaLoading.insert(loc=0, column='Attr', value=list(data))
    pcaLoading['PCALoading'] = pcaLoading.drop(['Attr'], axis=1).apply(np.square).sum(axis=1)
    sortedPCA = pcaLoading.sort_values(by=['PCALoading'], ascending=False)[:3]
    # print(sortedPCA)
    temp = sortedPCA.values.tolist()
    result = [temp[0][0],temp[1][0],temp[2][0]];
    return result;
```

5.6 Scatter plot

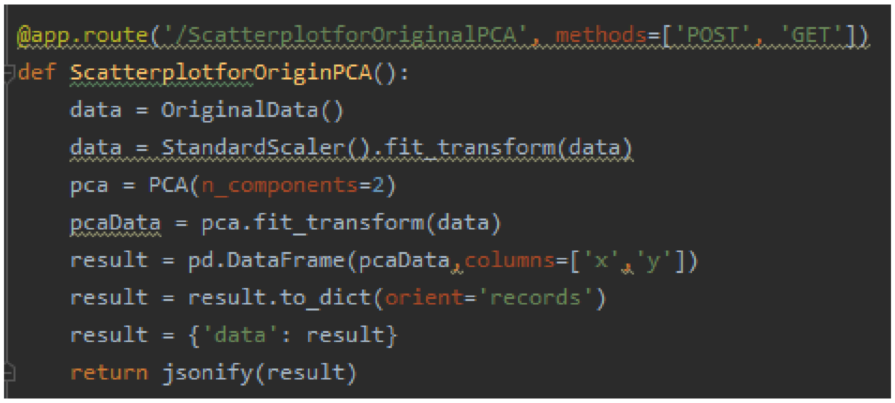
The main task to draw scatter plot is the draw the points of data. I use d3 to draw all these data and show them on the website.



```javascript
//draw dots
svg.selectAll(".dot")
    .data(data)
    .enter()
    .append("circle")
    .attr("class","dot")
    .attr("r", 3.5)
    .attr("cx", function(d) { return xScale(d.x) })
    .attr("cy", function(d) { return yScale(d.y) })
    .style("fill", function(d) { return color(d.cluster);})
```

5.7 2D Scatter plot of top two PCA vectors

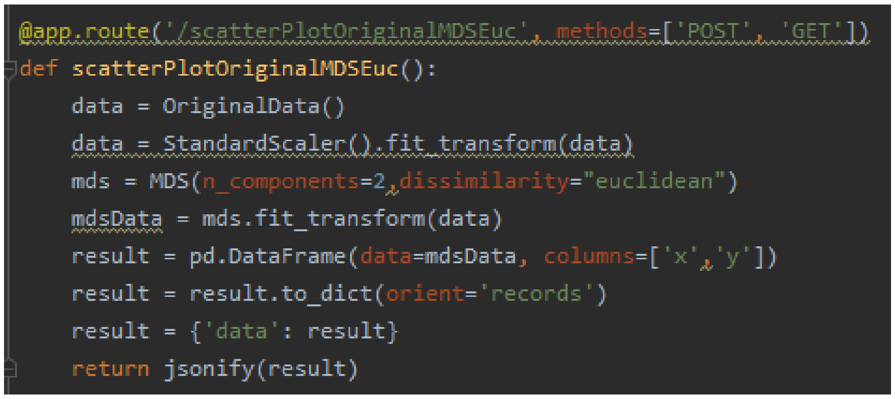
Use PCA method which n_components=2 to calculate the 2D values and return to the client.



```python
@app.route('/ScatterplotforOriginalPCA', methods=['POST', 'GET'])
def ScatterplotforOriginPCA():
    data = OriginalData()
    data = StandardScaler().fit_transform(data)
    pca = PCA(n_components=2)
    pcaData = pca.fit_transform(data)
    result = pd.DataFrame(pcaData,columns=['x','y'])
    result = result.to_dict(orient='records')
    result = {'data': result}
    return jsonify(result)
```

5.8 2D Scatter plot of MDS(Euclidian)

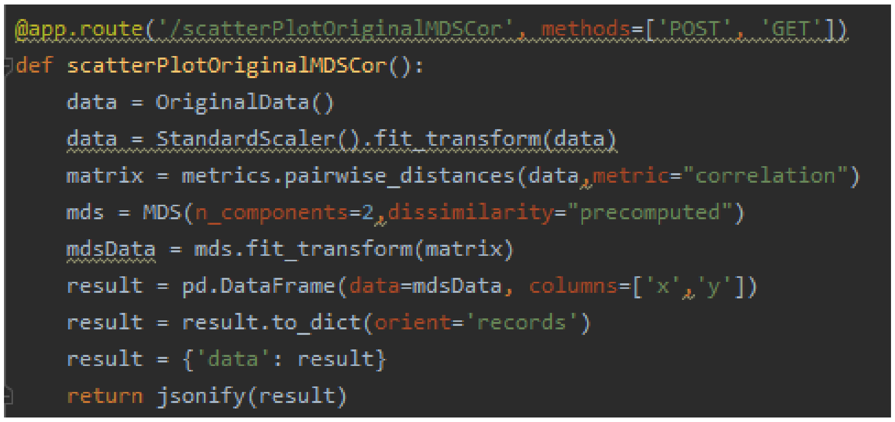
Use MDS method which n_components=2 and dissimilarity="euclidean" to calculate the 2D values. Then return to the client to draw the scatter plot.



```
@app.route('/scatterPlotOriginalMDSEuc', methods=['POST', 'GET'])
def scatterPlotOriginalMDSEuc():
    data = OriginalData()
    data = StandardScaler().fit_transform(data)
    mds = MDS(n_components=2,dissimilarity="euclidean")
    mdsData = mds.fit_transform(data)
    result = pd.DataFrame(data=mdsData, columns=['x','y'])
    result = result.to_dict(orient='records')
    result = {'data': result}
    return jsonify(result)
```

5.9 2D Scatter plot of MDS(Correlation distance)

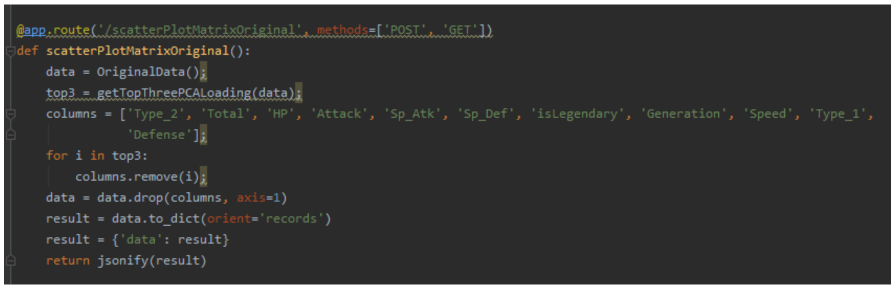
First use a matrix to represent the pairwise distances of all data. Then use MDS method which n_components=2 and dissimilarity="precomputed" to calculate the 2D values. Finally return to the client to draw the scatter plot.



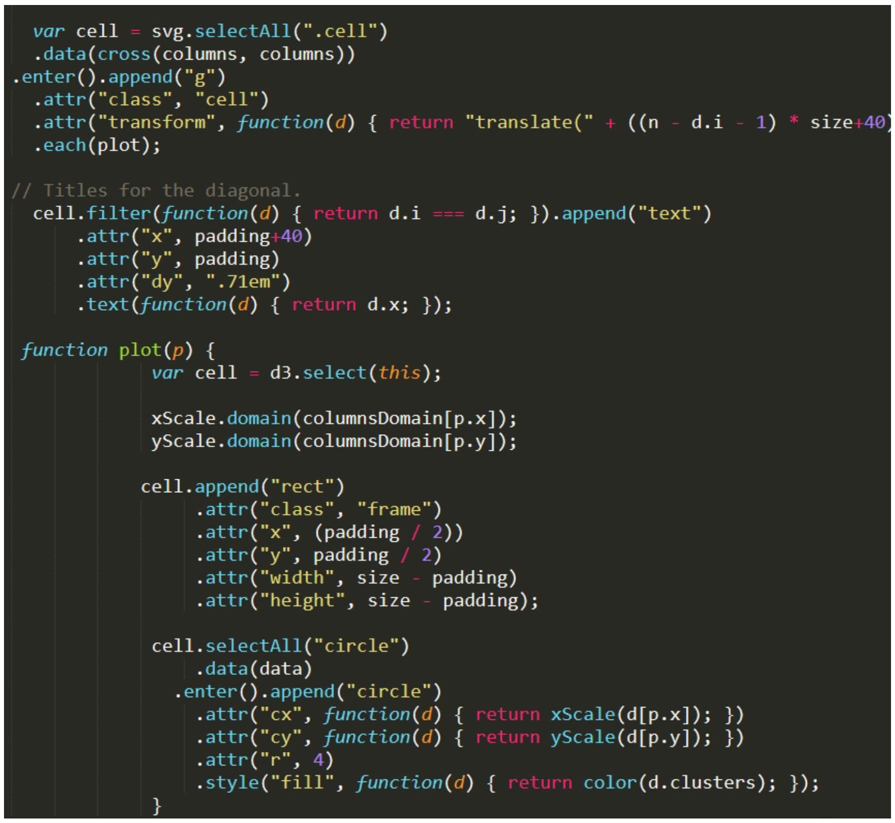
```
@app.route('/scatterPlotOriginalMDSCor', methods=['POST', 'GET'])
def scatterPlotOriginalMDSCor():
    data = OriginalData()
    data = StandardScaler().fit_transform(data)
    matrix = metrics.pairwise_distances(data,metric="correlation")
    mds = MDS(n_components=2,dissimilarity="precomputed")
    mdsData = mds.fit_transform(matrix)
    result = pd.DataFrame(data=mdsData, columns=['x','y'])
    result = result.to_dict(orient='records')
    result = {'data': result}
    return jsonify(result)
```

5.10 Scatter plot matrix of the three highest PCA loaded attributes

First get the top three attributes with highest PCA loading and drop these types data. The client use these three attributes to draw the scatter plot matrix.



```
@app.route('/scatterPlotMatrixOriginal', methods=['POST', 'GET'])
def scatterPlotMatrixOriginal():
    data = OriginalData();
    top3 = getTopThreePCALoading(data);
    columns = ['Type_2', 'Total', 'HP', 'Attack', 'Sp_Atk', 'Sp_Def', 'isLegendary', 'Generation', 'Speed', 'Type_1',
               'Defense'];
    for i in top3:
        columns.remove(i);
    data = data.drop(columns, axis=1)
    result = data.to_dict(orient='records')
    result = {'data': result}
    return jsonify(result)
```



```
  var cell = svg.selectAll(".cell")
  .data(cross(columns, columns))
.enter().append("g")
  .attr("class", "cell")
  .attr("transform", function(d) { return "translate(" + ((n - d.i - 1) * size+40)
  .each(plot);

// Titles for the diagonal.
  cell.filter(function(d) { return d.i === d.j; }).append("text")
      .attr("x", padding+40)
      .attr("y", padding)
      .attr("dy", ".71em")
      .text(function(d) { return d.x; });

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

             xScale.domain(columnsDomain[p.x]);
             yScale.domain(columnsDomain[p.y]);

            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 xScale(d[p.x]); })
                 .attr("cy", function(d) { return yScale(d[p.y]); })
                 .attr("r", 4)
                 .style("fill", function(d) { return color(d.clusters); });
             }
```