Mini Project #2 Report : CSE 564

Data Choosen : IMDB Movie Data

Data Cleaning :

The csv data that is being used for this lab had many redundant columns to make more sense out of the data I choose to use few selected columns. The code that I used to select columns is below

```
def removeExtraColumns(inputFile,outputFile):
   keepingIndex = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
   count = 0
   with open(inputFile, 'r', encoding="utf8") as input:
      with open(outputFile, 'w+') as output:
         for line in input:
            newRow = []
            print(count)
            count += 1
            row = line.split(",")
            for idx,item in enumerate(row):
               if idx in keepingIndex:
                  newRow.append(item)
            if len(newRow) == 17:
               output.write(','.join(str(x) for x in newRow))
```

After retaining only few selected columns, I noticed that the range of the values for different columns was very different from each other so I normalized the data by subtracting each column by its mean and divide by the variance. The python code used is below.

```
def norm_data(inputFile, outputFile):
   df = pd.read_csv(inputFile, encoding="ISO-8859-1")
   result = df.copy()
   result = result.set_index("movie_title")
   for feature_name in result.columns:
      max_value = df[feature_name].max()
      min_value = df[feature_name].min()
      result[feature_name] = (result[feature_name] - min_value)/(max_value -
min_value)
   result.to_csv(outputFile)
```

After normalizing I discovered few blank values in the data, Since those were very few I removed those rows from the data.

data clustering and decimation:

Our first task was to randomly sample the data.

Random Sampling:

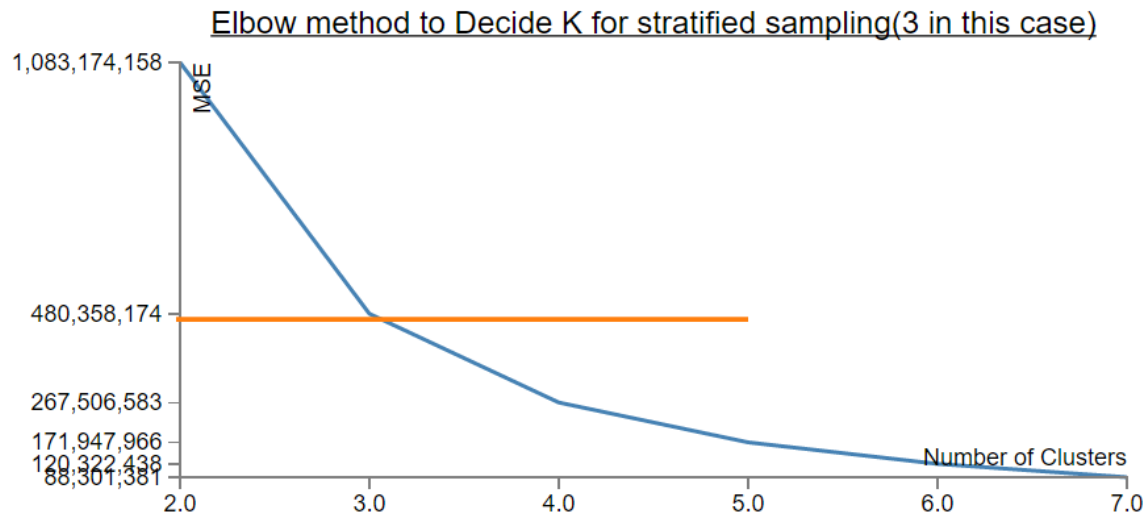
One way of doing it is using randomized sampling. I generated a random number between 0 and the number of rows and kept the rows which had those index as row numbers. This way I reduced the data size by half initially it was ~3k after random sampling it was ~1.5k.

The code is as follows

```
def randomSample(inputFile):
    df = pd.read_csv(inputFile, encoding="ISO-8859-1")
    print(df.head())
    random_sampled_df = df.sample(n=1500)
    print(random_sampled_df.head())
    print(len(random_sampled_df))
    random_sampled_df.to_csv("data/random_sampled.csv",index=False)
```

stratified sampling:

In Stratified sampling we first do the clustering of data and then pick the representative number of samples from each data. The first task in this case is to decide the number of clusters. I used elbow method to decide the number of clusters for k – means clustering. Below is the elbow plot that I got.



The code for generating below graph :

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
    body { font: 12px Arial;}
    path {
        stroke: steelblue;
        stroke-width: 2;
        fill: none;
    }
    .axis path,
    .axis line {
        fill: none;
        stroke: grey;
        stroke-width: 1;
        shape-rendering: crispEdges;
    }

</style>
<body>
<script type="text/javascript" src="js/d3.v3.js"></script>
<script>
    var margin = {top: 30, right: 20, bottom: 30, left: 100},
        width = 600 - margin.left - margin.right,
        height = 270 - margin.top - margin.bottom;

    var x = d3.scale.linear().range([0,width]);
    var y = d3.scale.linear().range([height, 0]);


    var valueline = d3.svg.line()
        .x(function(d) { return x(d.K); })
        .y(function(d) { return y(d.MSE); });

    var svg = d3.select("body")
        .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");

    d3.csv("data/clustering_elbow.csv", function(error, data) {
        data.forEach(function(d) {
            d.K = parseInt(d.K);
            d.MSE = parseFloat(d.MSE);
        });
        xvalues = [];
        data.forEach(function (d) {
            xvalues.push(d.K);
        });
        yvalues = [];
        data.forEach(function (d) {
            yvalues.push(d.MSE);
        });
        var yAxis = d3.svg.axis().scale(y)
            .orient("left").tickValues(yvalues);

        var xAxis = d3.svg.axis().scale(x)
            .orient("bottom").tickValues(xvalues);

        x.domain([d3.min(data, function(d) { return d.K; }), d3.max(data, function(d)
{ return d.K; })]);
        y.domain([d3.min(data, function(d) { return d.MSE; }), d3.max(data,
```

By looking at the above plot I decided to make 3 clusters.

Below is the python code that I wrote from making 3 clusters.

```
def strat_sampling(inputFile, outFile):
   with open(outFile,"w+") as output:
      df = pd.read_csv(inputFile,encoding="ISO-8859-1")
      indexs = df["movie_title"]
      df.pop("movie_title")
      df["index"] = df.index
      dataArray = df.values
      kmeans = KMeans(n_clusters=3)
      kmeans.fit(dataArray)
      centroids = kmeans.cluster_centers_
      labels = kmeans.labels_
      cluster1 = np.where(labels == 0)[0]
      cluster2 = np.where(labels == 1)[0]
      cluster3 = np.where(labels == 2)[0]
      np.random.shuffle(cluster1)
      strata1 = cluster1[:500]
      np.random.shuffle(cluster2)
      strata2 = cluster2[:500]
      np.random.shuffle(cluster3)
      strata3 = cluster3[:500]
      combinedArray = np.concatenate((dataArray[strata1],dataArray[strata2]),
axis=0)
      combinedArray =
np.concatenate((combinedArray,dataArray[strata3]),axis=0)
      finaldf = pd.DataFrame(combinedArray, columns=df.columns)
      finaldf["movie_title"] = indexs[finaldf["index"]]
      movie_list = []
      for index in finaldf["index"]:
         movie_list.append(indexs[index])
      finaldf["movie_title"] = movie_list
      finaldf.pop("index")
      finaldf = finaldf.set_index("movie_title")
      finaldf.to_csv(outFile)
```

As we can see in above code I first built the 3 clusters and checked the size of each cluster, Since in my case all 3 clusters had almost equal size I decided to pick 500 samples from each cluster for that I shuffled each cluster and picked top 500 for making my sample size around ~1.5k.

dimension reduction

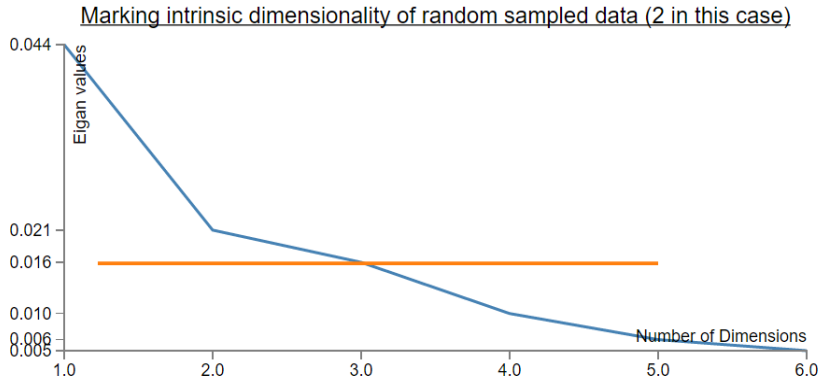
Finding the intrinsic dimensionality of the data using PCA :

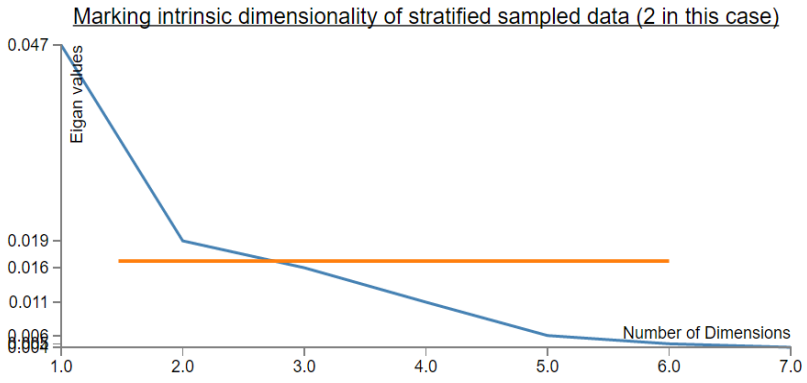
For finding the intrinsic dimension of the data I first did the principle component analysis of the data and checked the eigen values. Plotted the graph for different eigen values. In my case the intrinsic dimensionality to be found was 2 both for randomly sampled data as well as stratified sampled data.

Principle component analysis :

```
def finding_intrinsic_dimen_randomsample(inputFile, outputFile):
   with open(outputFile, 'w') as output:
      df = pd.read_csv(inputFile,encoding="ISO-8859-1")
      df = df.set_index("movie_title")
      cov_mat = np.cov(df.values.T)
      eig_vals, eig_vecs = np.linalg.eig(cov_mat)
      for val in eig_vals:
         print(val)
```

Below are the elbow plots :





Below is the d3 code used to plot above elbow graphs.

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
body { font: 12px Arial;}
path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}
.axis path,
.axis line {
    fill: none;
    stroke: grey;
    stroke-width: 1;
    shape-rendering: crispEdges;
}
</style>
<body>
<script type="text/javascript" src="js/d3.v3.js"></script>
<script>
    var margin = {top: 30, right: 20, bottom: 30, left: 50},
        width = 600 - margin.left - margin.right,
        height = 270 - margin.top - margin.bottom;
    var x = d3.scale.linear().range([0,width]);
    var y = d3.scale.linear().range([height, 0]);
    var valueline = d3.svg.line()
        .x(function(d) { return x(d.n); })
        .y(function(d) { return y(d.eigan_value); });
    var svg = d3.select("body")
        .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");
    d3.csv("data/random_sample_intrinsic_dimen6.csv", function(error, data) {
        data.forEach(function(d) {
            d.n = parseInt(d.n);
            d.eigan_value = parseFloat(d.eigan_value);
        });
        xvalues = [];
        data.forEach(function (d) {
            xvalues.push(d.n);
        });
        yvalues = [];
        data.forEach(function (d) {
            yvalues.push(d.eigan_value);
        });
        var yAxis = d3.svg.axis().scale(y)
            .orient("left").tickValues(yvalues);
        var xAxis = d3.svg.axis().scale(x)
            .orient("bottom").tickValues(xvalues);
        x.domain([d3.min(data, function(d) { return d.n; }), d3.max(data, function(d)
{ return d.n; })]);
        y.domain([d3.min(data, function(d) { return d.eigan_value; }), d3.max(data,
function(d) { return d.eigan_value; })]);
        svg.append("text")
            .attr("x", (width / 2))
            .attr("y", 0 - (margin.top / 2))
            .attr("text-anchor", "middle")
            .style("font-size", "16px")
            .style("text-decoration", "underline")
            .text("Marking intrinsic dimensionality of random sampled data (2 in this
```

```
case)");
        svg.append("path")
            .attr("class", "line")
            .attr("d", valueline(data));
        svg.append("g")
            .attr("class", "x axis")
            .attr("transform", "translate(0," + height + ")")
            .call(xAxis);
        svg.append("g")
            .attr("class", "y axis")
            .call(yAxis);
        svg.append("line")
            .style("stroke", "#FF7F0E")
            .style("stroke-width", "2.5px")
            .attr("x1", x(0)+130)
            .attr("y1", 150)
            .attr("x2", x(0) + width)
            .attr("y2", 150);
        svg.append("text")
            .attr("text-anchor","end")
            .attr("x",width)
            .attr("y",height-6)
            .text("Number of Dimensions");
        svg.append("text")
            .attr("text-anchor","end")
            .attr("y",6)
            .attr("dy",".75em")
            .attr("transform","rotate(-90)")
            .text("Eigan values");
    });

</script>
</body>
```

obtaining the three attributes with highest PCA loadings :

For this I first calculated the square sum of feature loadings. And plotted the scree plot to mark the top the attributes with highest loadings.

Below is the python code to calculate the square sum of loadings.

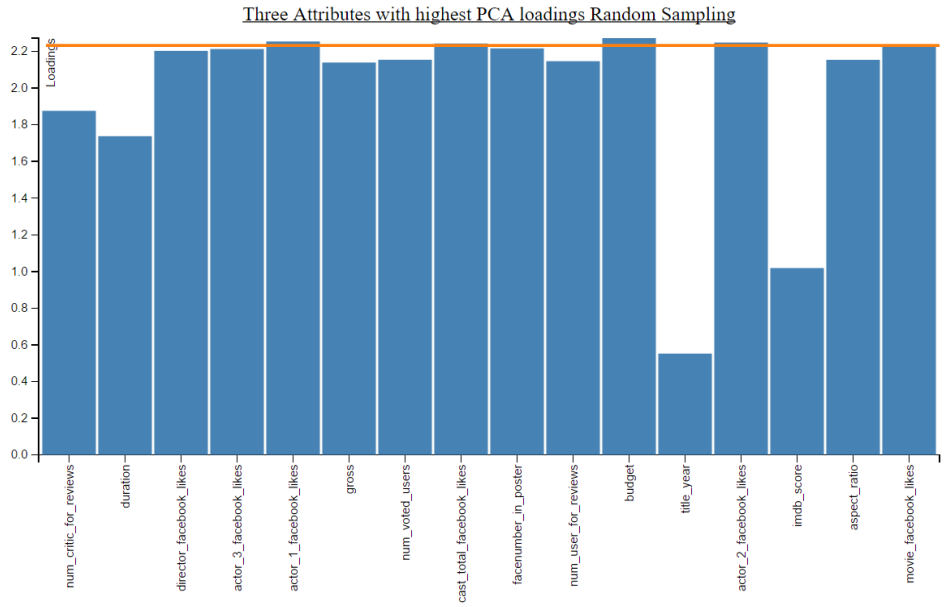
```
def FindingPCA(inputFile, outputFile):
   with open(outputFile,'w') as output:
      df = pd.read_csv(inputFile,encoding="ISO-8859-1")
      df = df.set_index("movie_title")
      pca = decomposition.PCA()
      pca.fit(df.values)
      i=np.identity(df.shape[1])
      coef = pca.transform(i)
      corrDf = pd.DataFrame(coef, columns=['PC-1','PC-2','PC-3','PC-4','PC-
5','PC-6','PC-7','PC-8','PC-9','PC-10','PC-11','PC-12','PC-13','PC-14','PC-
15','PC-16'], index=df.columns)
      corrDf.to_csv(outputFile)
```

```
def Calculate_sq_sum_Loadings(inputFile,outputFile):
   with open(outputFile, 'w') as output:
      df = pd.read_csv(inputFile, index_col=0)
      df['sque_sum_loadings'] = np.square(df).sum(axis=1)
      df.to_csv(outputFile)
```

After generating the values of square sum loading I plotted the bar chart for different loadings and marked the top 3 attributes for random as well as stratified sampled data.

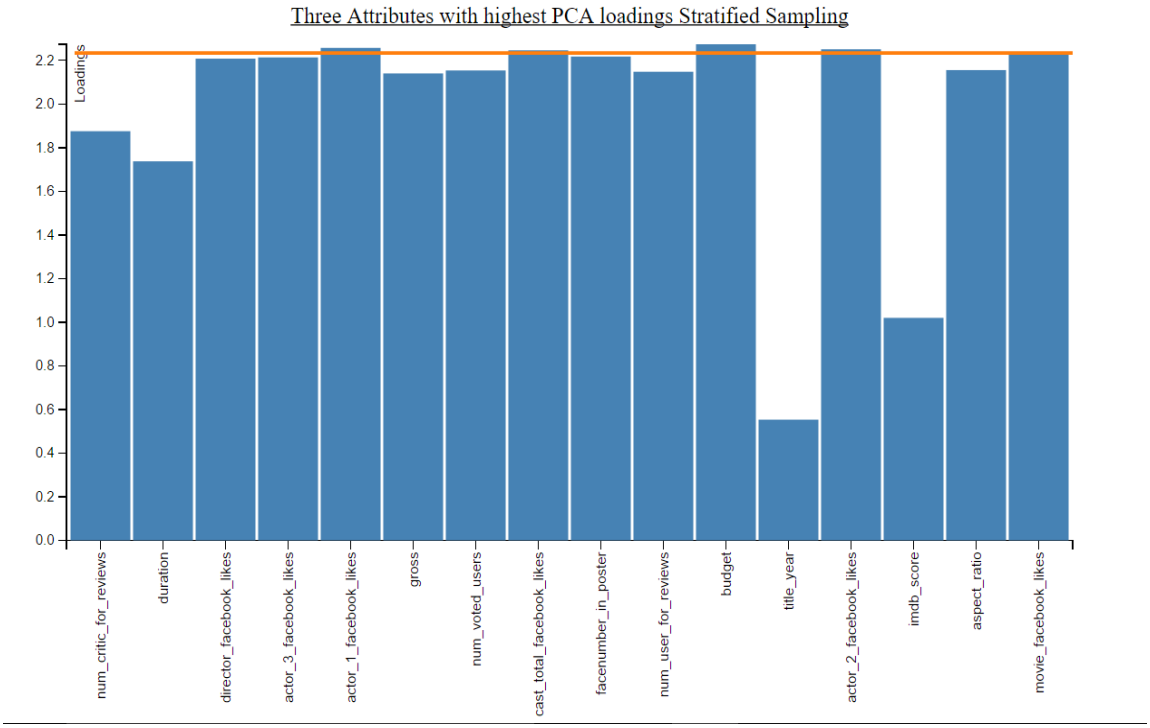
Three Attributes with highest PCA loadings Random Sampling

- actor_1_facebook_likes
- budget
- actor_2_facebook_likes



Three Attributes with highest PCA loadings Stratified Sampling

- actor_1_facebook_likes
- budget
- actor_2_facebook_likes



As we can see many of the attributes were almost close

But since we had to choose 3 among them I tried to fit a line and decided.

Below id d3 code used to generate above scree plots.

```
<!DOCTYPE html>
<meta charset="utf-8">

<head>
    <style>
        .axis {
            font: 10px sans-serif;
        }
        .axis path,
        .axis line {
            fill: none;
            stroke: #000;
            shape-rendering: crispEdges;
        }
    </style>
</head>
<body>
<div>
    Three Attributes with highest PCA loadings Stratified Sampling
    <ul>
        <li>actor_1_facebook_likes</li>
```

```
        <li>budget</li>
        <li>actor_2_facebook_likes</li>
    </ul>
</div>
<script type="text/javascript" src="js/d3.v3.js"></script>

<script>

    var margin = {top: 30, right: 20, bottom: 125, left: 40},
        width = 800 - margin.left - margin.right,
        height = 500 - margin.top - margin.bottom;

    var x = d3.scale.ordinal().rangeRoundBands([0, width], .05);
    var y = d3.scale.linear().range([height, 0]);
    var xAxis = d3.svg.axis()
        .scale(x)
        .orient("bottom")

    var yAxis = d3.svg.axis()
        .scale(y)
        .orient("left")
        .ticks(10);
    var svg = d3.select("body").append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");

    svg.append("text")
        .attr("x", (width / 2))
        .attr("y", 0 - (margin.top / 2))
        .attr("text-anchor", "middle")
        .style("font-size", "16px")
        .style("text-decoration", "underline")
        .text("Three Attributes with highest PCA loadings Stratified Sampling");

    svg.append("line")
        .style("stroke", "#FF7F0E")
        .style("stroke-width", "2.5px")
        .attr("x1", x(0)+190)
        .attr("y1", 130)
        .attr("x2", x(0) + width)
        .attr("y2", 130);
    d3.csv("data/strat_sqr_sum_loadings.csv", function(error, data) {
        data.forEach(function(d) {
            d.features = d.features;
            d.sqr_sum_loadings = parseFloat(d.sqr_sum_loadings);
        });

        x.domain(data.map(function(d) { return d.features; }));
        y.domain([0, d3.max(data, function(d) { return d.sqr_sum_loadings; })]);

        svg.append("g")
            .attr("class", "x axis")
            .attr("transform", "translate(0," + height + ")")
            .call(xAxis)
            .selectAll("text")
            .style("text-anchor", "end")
            .attr("dx", "-.8em")
            .attr("dy", "-.55em")
            .attr("transform", "rotate(-90)" );
```

```
        svg.append("g")
            .attr("class", "y axis")
            .call(yAxis)
            .append("text")
            .attr("transform", "rotate(-90)")
            .attr("y", 6)
            .attr("dy", ".71em")
            .style("text-anchor", "end")
            .text("Loadings");

        svg.selectAll("bar")
            .data(data)
            .enter().append("rect")
            .style("fill", "steelblue")
            .attr("x", function(d) { return x(d.features); })
            .attr("width", x.rangeBand())
            .attr("y", function(d) { return y(d.sqr_sum_loadings); })
            .attr("height", function(d) { return height - y(d.sqr_sum_loadings); });
        svg.append("line")
            .style("stroke", "#FF7F0E")
            .style("stroke-width", "2.5px")
            .attr("x1", 6)
            .attr("y1", 6)
            .attr("x2", width)
            .attr("y2", 6);
    });
</script>

</body>
```

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

I projected the data into top 2 Principle components and plotted the scatter plot using D3.

Below is the python code used to get the transformation of the data into top 2 principle components.

```
def get_2D_Transform(inputFile, outputFile):
    df = pd.read_csv(inputFile, encoding="ISO-8859-1")
    movie_list = df["movie_title"]
    df.pop("movie_title")
    dataArray = df.values
    pca = decomposition.PCA(n_components =2)
    pca.fit(df.values)
    arr_2d = pca.transform(df.values)
    df_2d = pd.DataFrame(arr_2d, columns=['PC-1','PC-2'])
    df_2d["movie_title"] = movie_list
    df_2d = df_2d.set_index("movie_title")
    df_2d.to_csv(outputFile)
```

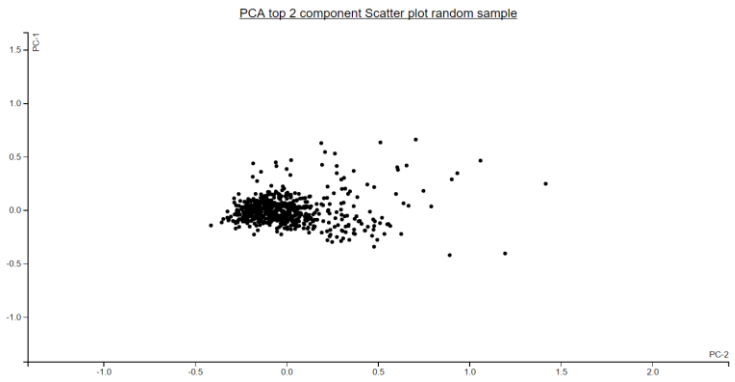
After getting the transformations for both sampled as well as stratified data, I plotted the scatter plots.

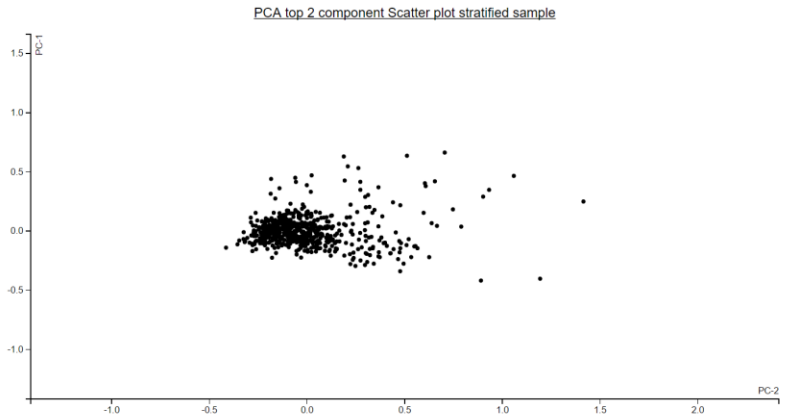
Below is the d3 code to create the scatter plots.

```
<!DOCTYPE html>
<html>
<meta charset="utf-8">
<style>
```

```
    body {
        font: 11px sans-serif;
    }
    .axis path,
    .axis line {
        fill: none;
        stroke: #000;
        shape-rendering: crispEdges;
    }
    .dot {
        stroke: #000;
    }
    .tooltip {
        position: absolute;
        width: 200px;
        height: 28px;
        pointer-events: none;
    }
</style>
<body>
<script type="text/javascript" src="js/d3.v3.js"></script>

<script>
    var margin = {top: 40, right: 20, bottom: 30, left: 40},
        width = 960 - margin.left - margin.right,
        height = 500 - margin.top - margin.bottom;
    var xValue = function(d) { return d.PC_1;},
        xScale = d3.scale.linear().range([0, width]),
        xMap = function(d) { return xScale(xValue(d));},
        xAxis = d3.svg.axis().scale(xScale).orient("bottom");

    var yValue = function(d) { return d.PC_2;},
        yScale = d3.scale.linear().range([height, 0]),
        yMap = function(d) { return yScale(yValue(d));},
        yAxis = d3.svg.axis().scale(yScale).orient("left");

    var svg = d3.select("body").append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

    var tooltip = d3.select("body").append("div")
        .attr("class", "tooltip")
        .style("opacity", 0);

    d3.csv("data/strat_sample_2D.csv", function(error, data) {
        data.forEach(function(d) {
            d.movie_title = d.movie_title;
            d.PC_1 = parseFloat(d.PC_1);
            d.PC_2 = parseFloat(d.PC_2);
        });
        xScale.domain([d3.min(data, xValue)-1, d3.max(data, xValue)+1]);
        yScale.domain([d3.min(data, yValue)-1, d3.max(data, yValue)+1]);
        svg.append("g")
            .attr("class", "x axis")
            .attr("transform", "translate(0," + height + ")")
            .call(xAxis)
            .append("text")
            .attr("class", "label")
            .attr("x", width)
            .attr("y", -6)
            .style("text-anchor", "end")
```

```
            .text("PC-2");
        svg.append("g")
            .attr("class", "y axis")
            .call(yAxis)
            .append("text")
            .attr("class", "label")
            .attr("transform", "rotate(-90)")
            .attr("y", 6)
            .attr("dy", ".71em")
            .style("text-anchor", "end")
            .text("PC-1");
        svg.selectAll(".dot")
            .data(data)
            .enter().append("circle")
            .attr("class", "dot")
            .attr("r", 2)
            .attr("cx", function (d) {
                return xScale(d.PC_1);
            })
            .attr("cy", yMap)
            .on("mouseover", function(d) {
                tooltip.transition()
                    .duration(200)
                    .style("opacity", .9);
                tooltip.html(d.movie_title)
                    .style("left", (d3.event.pageX + 5) + "px")
                    .style("top", (d3.event.pageY - 28) + "px")
                    .style("color","red")
                    .style("font-weight","bold");
            })
            .on("mouseout", function(d) {
                tooltip.transition()
                    .duration(500)
                    .style("opacity", 0);
            });
        svg.append("text")
            .attr("x", (width / 2))
            .attr("y", 0 - (margin.top / 2))
            .attr("text-anchor", "middle")
            .style("font-size", "16px")
            .style("text-decoration", "underline")
            .text("PCA top 2 component Scatter plot stratified sample");
    });

</script>
</body>
</html>
```





visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots:

For MDS I first generated the transformation of given data into MDS dimensions.

Below is the python code I used.

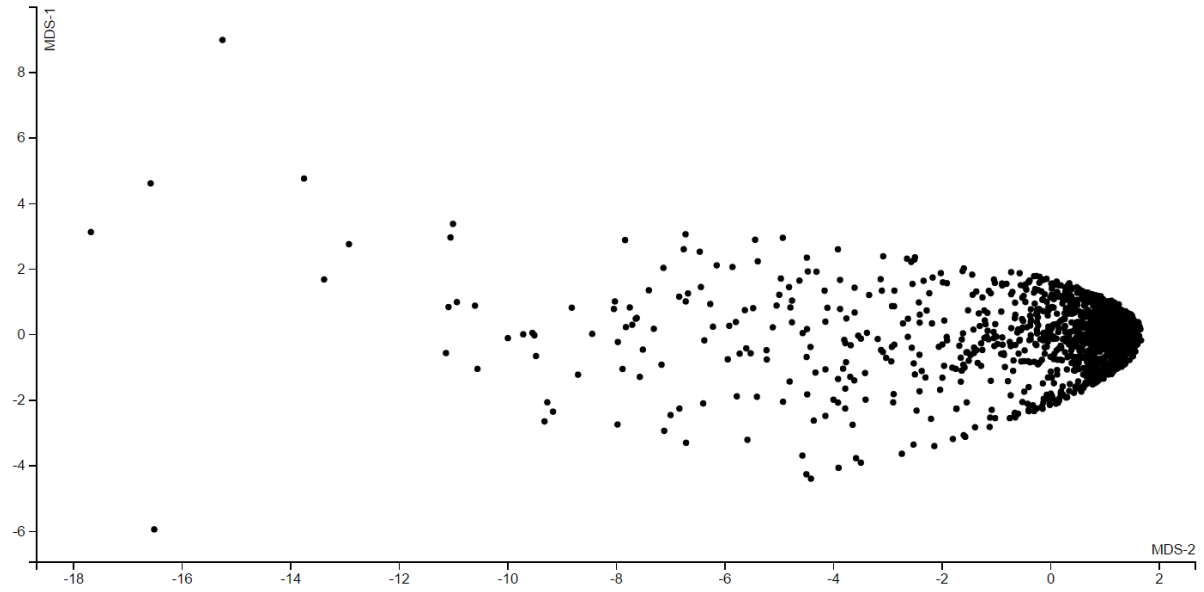
```
def get_MDS_Cordinates(inputFile, outputFile):
    df = pd.read_csv(inputFile, encoding="ISO-8859-1")
    movie_list = df["movie_title"]
    df.pop("movie_title")
    dataArray = df.values
    dis_mat = metrics.pairwise.pairwise_distances(dataArray,
metric="euclidean")
    mds = manifold.MDS(n_components=2)
    arr_2d = mds.fit_transform(dis_mat)
    df_2d = pd.DataFrame(arr_2d, columns=['MDS-1','MDS-2'])
    df_2d["movie_title"] = movie_list
    df_2d = df_2d.set_index("movie_title")
    df_2d.to_csv(outputFile)
```

I did created 4 scatter plots for randomly and stratified sampled data by calculating the euclidean and correlation distance measures.

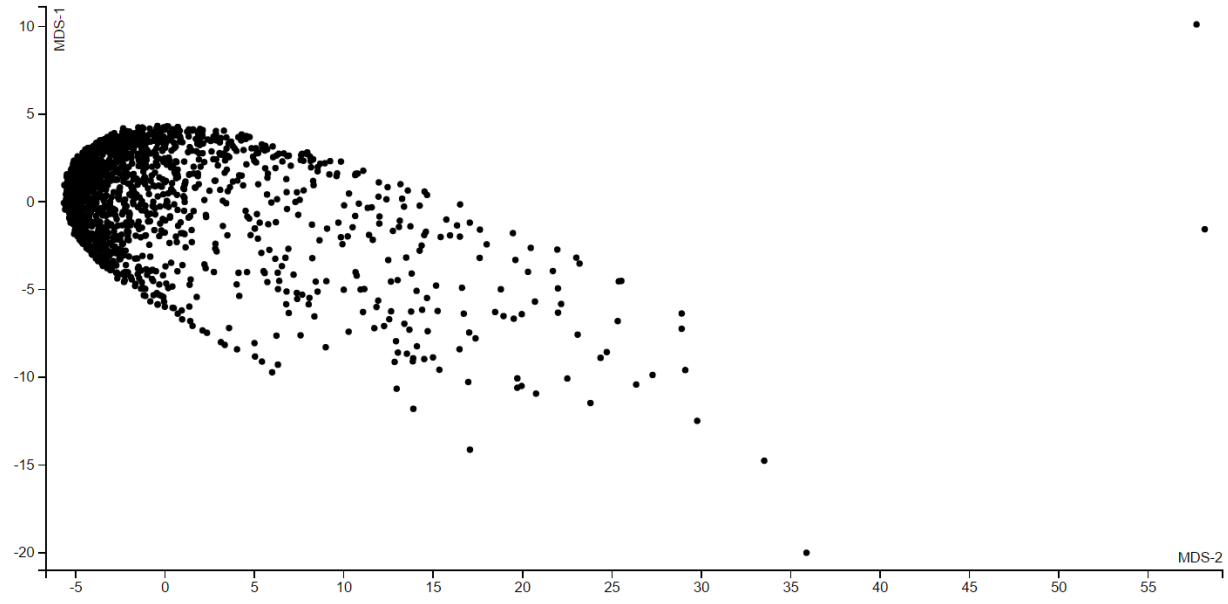
For generating the scatter plots same code as above is used.

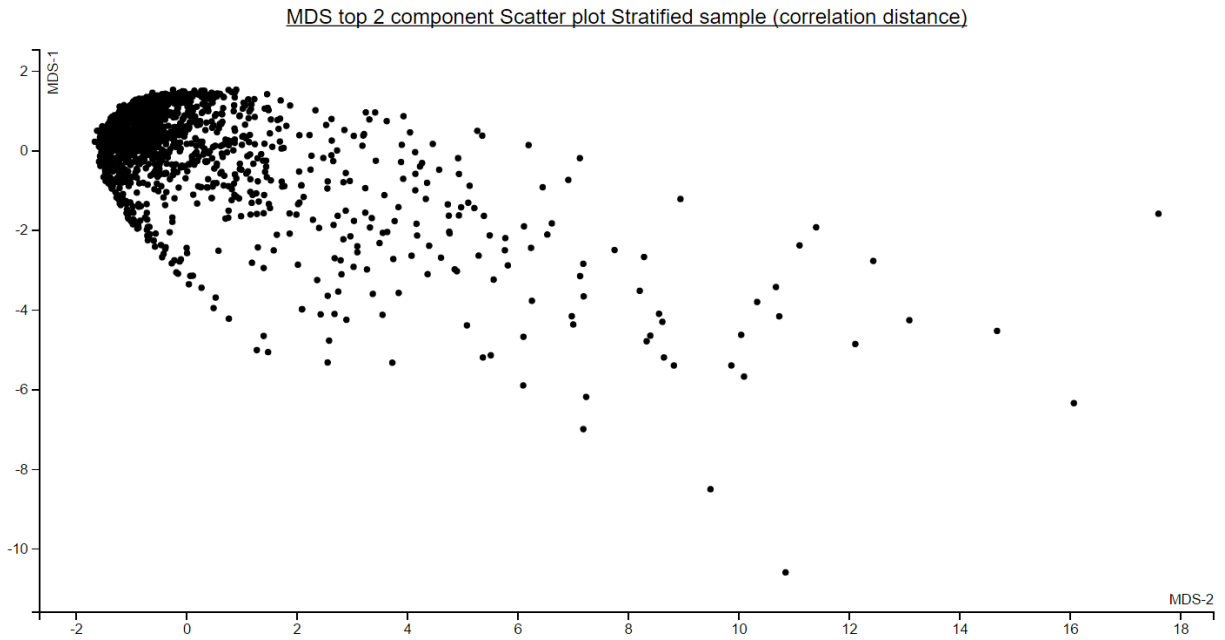
Below are the plots generated.

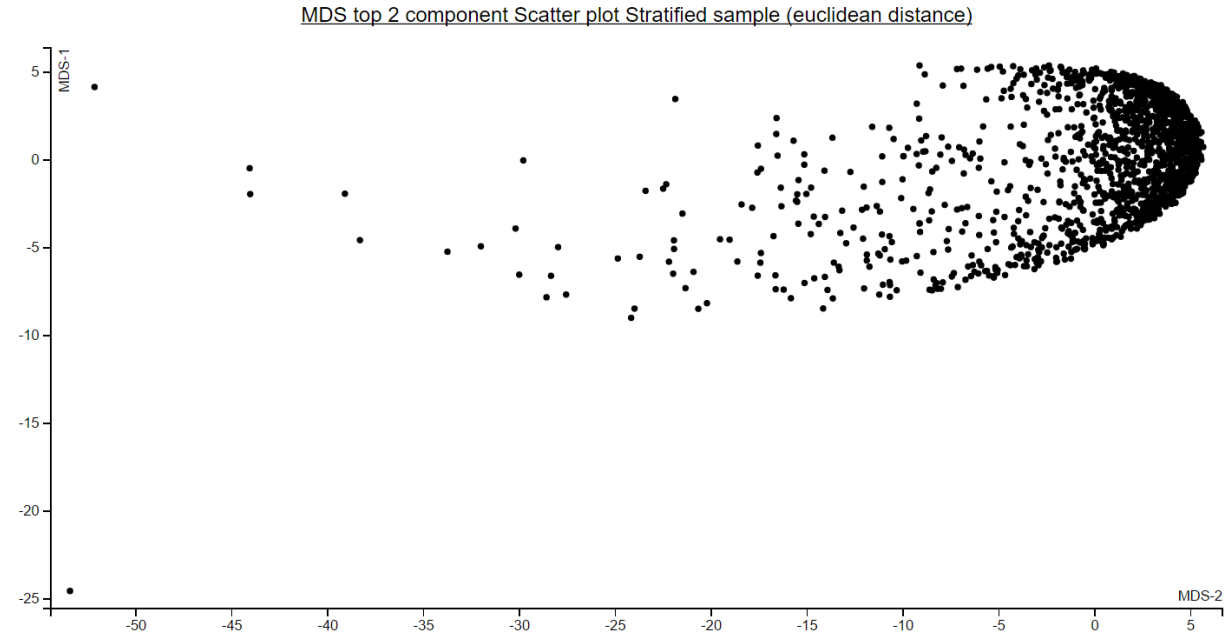
MDS top 2 component Scatter plot random sample (correlation distance)



MDS top 2 component Scatter plot random sample (euclidean distance)







As we can see from above plots both the plots are almost same for randomized as well as stratified sample.

visualize scatterplot matrix of the three highest PCA loaded attributes:

For this I created the data by dropping the other columns except 3 which were having highest loadings.

Below is the d3 code used to generate the scatterplot matrix.

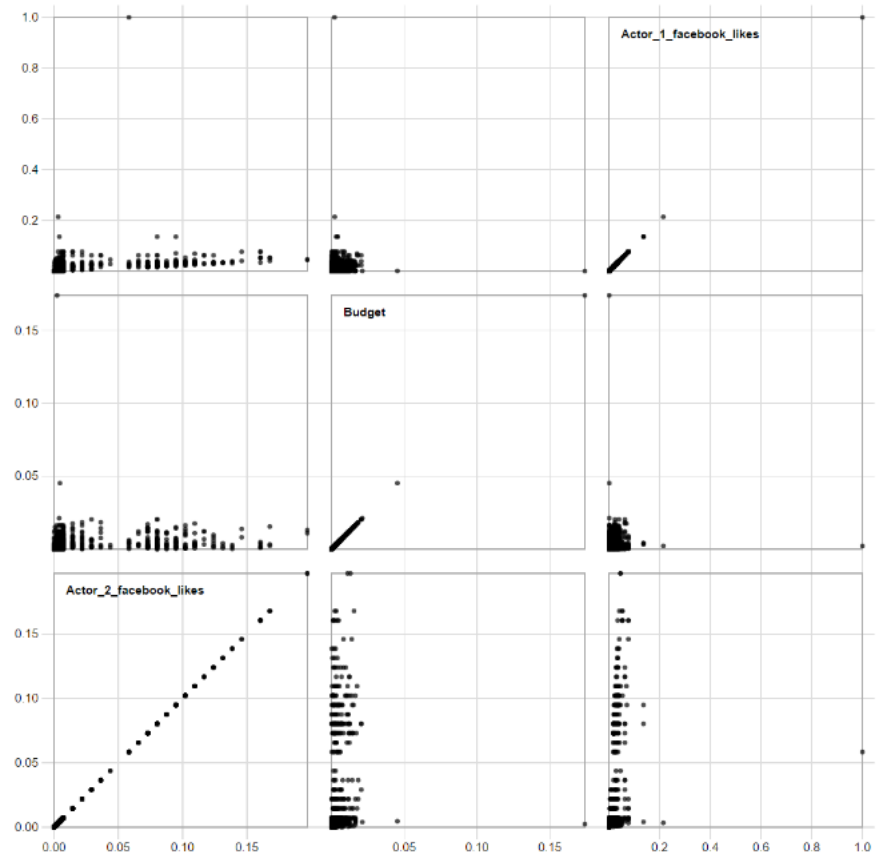
```
<script>
    var width = 960,
        size = 230,
        padding = 20;
    var x = d3.scale.linear()
```

```
        .range([padding / 2, size - padding / 2]);
    var y = d3.scale.linear()
        .range([size - padding / 2, padding / 2]);
    var xAxis = d3.svg.axis()
        .scale(x)
        .orient("bottom")
        .ticks(6);
    var yAxis = d3.svg.axis()
        .scale(y)
        .orient("left")
        .ticks(6);
    d3.csv("data/strat_sample_scatter_matrix.csv", function(error, data) {
        if (error) throw error;
        var domainByTrait = {},
            traits = d3.keys(data[0]).filter(function(d) { return d; }),
            n = traits.length;

        traits.forEach(function(trait) {
            domainByTrait[trait] = d3.extent(data, function(d) { return
parseFloat(d[trait]); });
        });
        xAxis.tickSize(size * n);
        yAxis.tickSize(-size * n);
        var brush = d3.svg.brush()
            .x(x)
            .y(y)
            .on("brushstart", brushstart)
            .on("brush", brushmove)
            .on("brushend", brushend);
        var svg = d3.select("body").append("svg")
            .attr("width", size * n + padding)
            .attr("height", size * n + padding)
            .append("g")
            .attr("transform", "translate(" + padding + "," + padding / 2 + ")");
        svg.selectAll(".x.axis")
            .data(traits)
            .enter().append("g")
            .attr("class", "x axis")
            .attr("transform", function(d, i) { return "translate(" + (n - i - 1) *
size + ",0)"; })
            .each(function(d) { x.domain(domainByTrait[d]);
d3.select(this).call(xAxis); });
        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);
        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; });
        cell.call(brush);
```

```
        function plot(p) {
            var cell = d3.select(this);
            x.domain(domainByTrait[p.x]);
            y.domain(domainByTrait[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 x(d[p.x]); })
                .attr("cy", function(d) { return y(d[p.y]); })
                .attr("r", 2)
        }
        var brushCell;
        function brushstart(p) {
            if (brushCell !== this) {
                d3.select(brushCell).call(brush.clear());
                x.domain(domainByTrait[p.x]);
                y.domain(domainByTrait[p.y]);
                brushCell = this;
            }
        }
        function brushmove(p) {
            var e = brush.extent();
            svg.selectAll("circle").classed("hidden", function(d) {
                return e[0][0] > d[p.x] || d[p.x] > e[1][0]
                    || e[0][1] > d[p.y] || d[p.y] > e[1][1];
            });
        }
        function brushend() {
            if (brush.empty()) svg.selectAll(".hidden").classed("hidden", false);
        }
    });
    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;
    }
    svg.append("text")
        .attr("x", (width / 2))
        .attr("y", 0 - (margin.top / 2))
        .attr("text-anchor", "middle")
        .style("font-size", "16px")
        .style("text-decoration", "underline")
        .text("Elbow method to Decide K for stratified sampling(3 in this case)");

</script>
```

scatterplot matrix of the three highest PCA loaded attributes (randomly sampled values)

- actor_1_facebook_likes
- budget
- actor_2_facebook_likes



scatterplot matrix of the three highest PCA loaded attributes (Straitified sampled values)

- actor_1_facebook_likes
- budget
- actor_2_facebook_likes