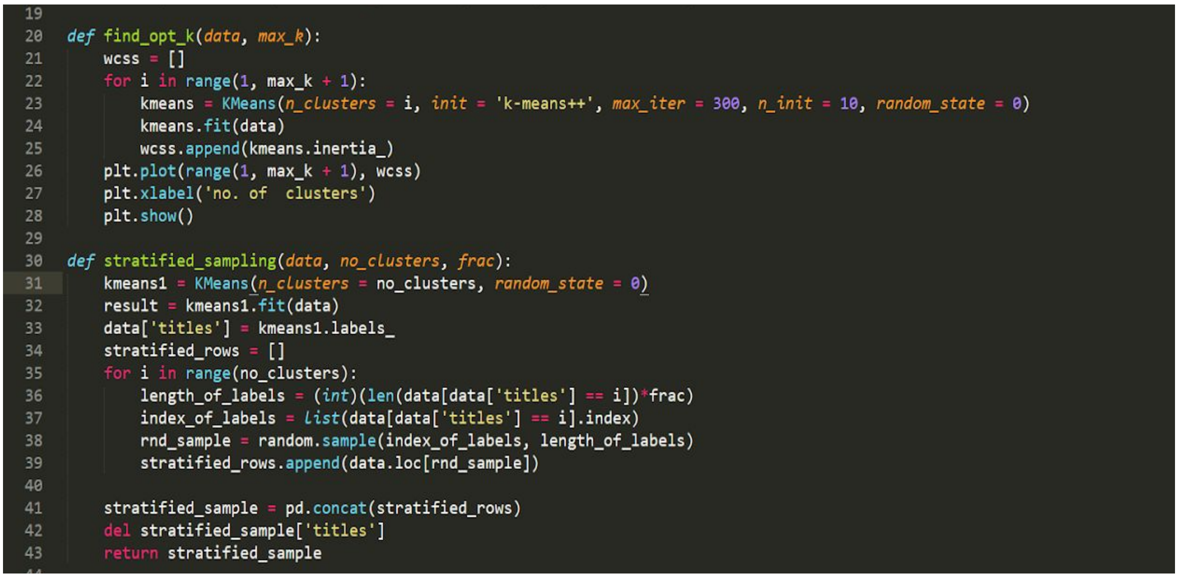
Link for video: https://youtu.be/GeUfMmVI-90

Visualization HW2 Report

1. Chosen Dataset:
   Video Games Sales which lists all the various names and platforms a game is released on along with the user ratings, user scores, critic ratings, critic scores, Age rating and the developer of the game.
2. Task 1:
   - 1a and b: This task involved performing random and stratified sampling on the dataset
   - For stratified sampling we first need to apply kmeans clustering algorithm on the data to obtain clusters and then extract samples from individual clusters.
   - The code for this is snapshotted and pasted below:-



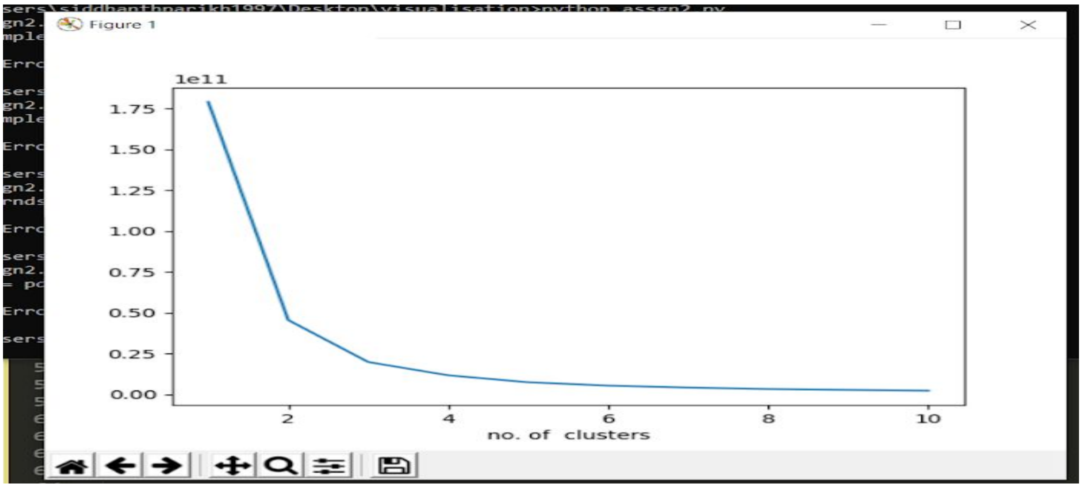
```python

def find_opt_k(data, max_k):
    wcss = []
    for i in range(1, max_k + 1):
        kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
        kmeans.fit(data)
        wcss.append(kmeans.inertia_)
    plt.plot(range(1, max_k + 1), wcss)
    plt.xlabel('no. of  clusters')
    plt.show()

def stratified_sampling(data, no_clusters, frac):
    kmeans1 = KMeans(n_clusters = no_clusters, random_state = 0)
    result = kmeans1.fit(data)
    data['titles'] = kmeans1.labels_
    stratified_rows = []
    for i in range(no_clusters):
        length_of_labels = (int)(len(data[data['titles'] == i])*frac)
        index_of_labels = list(data[data['titles'] == i].index)
        rnd_sample = random.sample(index_of_labels, length_of_labels)
        stratified_rows.append(data.loc[rnd_sample])

    stratified_sample = pd.concat(stratified_rows)
    del stratified_sample['titles']
    return stratified_sample
```

   - The elbow curve looked something like this :-

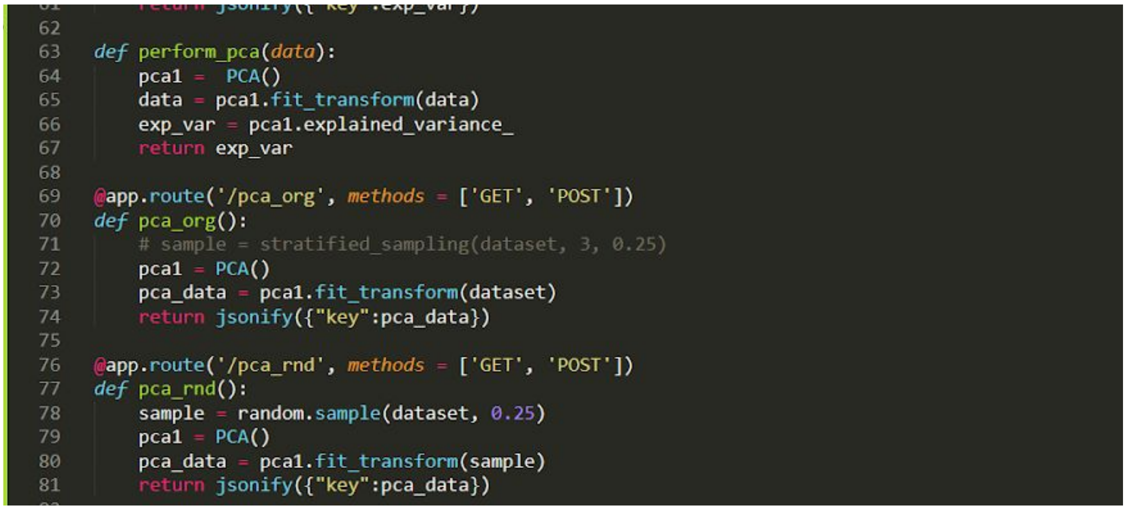


```
sers\siddhanthparikh1997\Desktop\visualisation>python assgn2.py
```

Task 2:

2a) Dimension reduction

- Intrinsic Dimensionality of data is calculated using the PCA plot for both random and adaptive data samples. The Scree plot visualization is produced and intrinsic dimensionality is marked. It comes out to be 3 in both the cases as shown below the code snippet



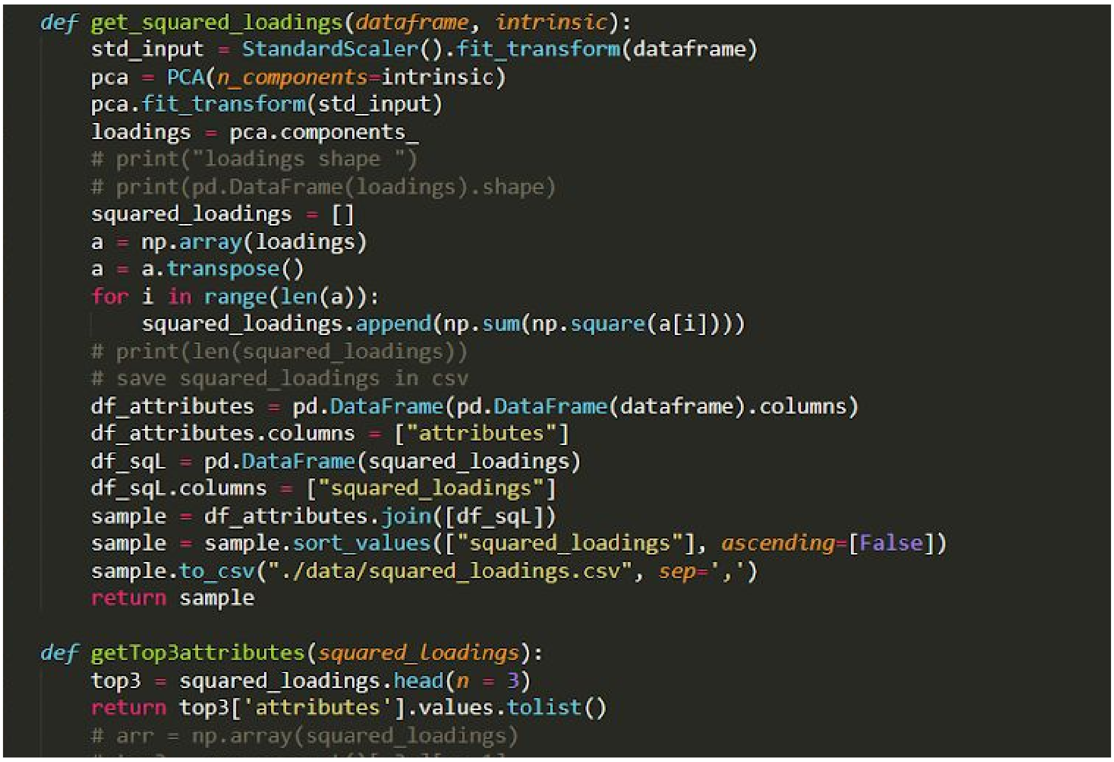
```
    return jsonify({"key":exp_var})

def perform_pca(data):
    pca1 =  PCA()
    data = pca1.fit_transform(data)
    exp_var = pca1.explained_variance_
    return exp_var

@app.route('/pca_org', methods = ['GET', 'POST'])
def pca_org():
    # sample = stratified_sampling(dataset, 3, 0.25)
    pca1 = PCA()
    pca_data = pca1.fit_transform(dataset)
    return jsonify({"key":pca_data})

@app.route('/pca_rnd', methods = ['GET', 'POST'])
def pca_rnd():
    sample = random.sample(dataset, 0.25)
    pca1 = PCA()
    pca_data = pca1.fit_transform(sample)
    return jsonify({"key":pca_data})
```

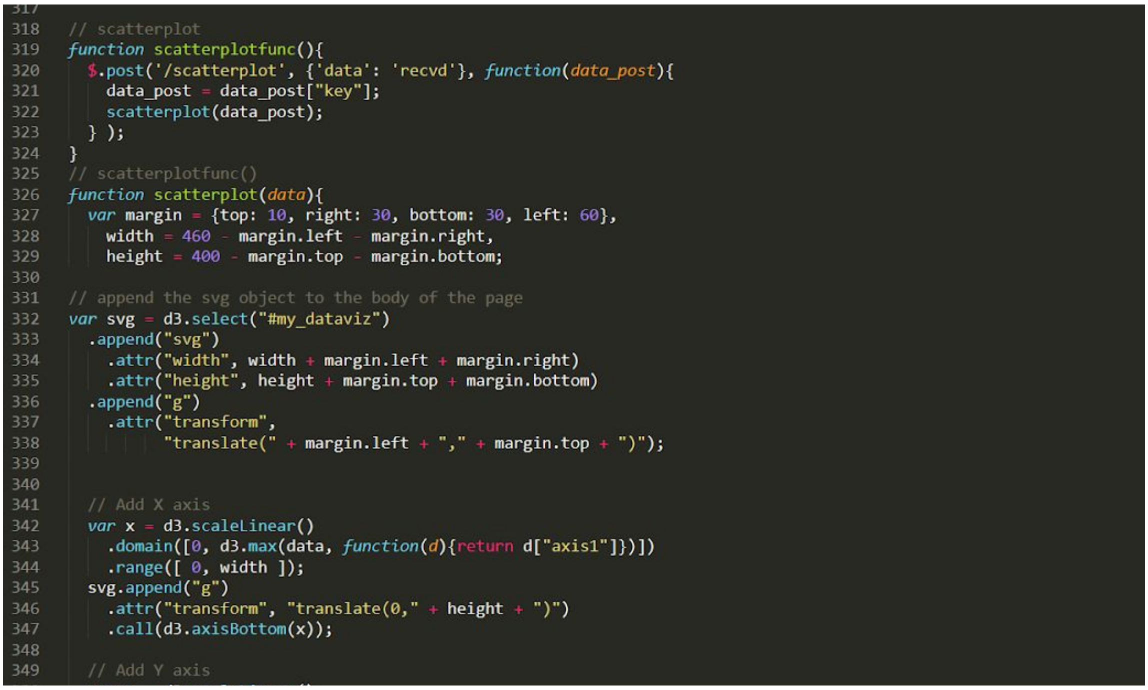
2c) top 3 highest loadings obtained:-



```python
def get_squared_loadings(dataframe, intrinsic):
    std_input = StandardScaler().fit_transform(dataframe)
    pca = PCA(n_components=intrinsic)
    pca.fit_transform(std_input)
    loadings = pca.components_
    # print("loadings shape ")
    # print(pd.DataFrame(loadings).shape)
    squared_loadings = []
    a = np.array(loadings)
    a = a.transpose()
    for i in range(len(a)):
        squared_loadings.append(np.sum(np.square(a[i])))
    # print(len(squared_loadings))
    # save squared_loadings in csv
    df_attributes = pd.DataFrame(pd.DataFrame(dataframe).columns)
    df_attributes.columns = ["attributes"]
    df_sqL = pd.DataFrame(squared_loadings)
    df_sqL.columns = ["squared_loadings"]
    sample = df_attributes.join([df_sqL])
    sample = sample.sort_values(["squared_loadings"], ascending=[False])
    sample.to_csv("./data/squared_loadings.csv", sep=',')
    return sample

def getTop3attributes(squared_loadings):
    top3 = squared_loadings.head(n = 3)
    return top3['attributes'].values.tolist()
    # arr = np.array(squared_loadings)
```

3a) This task involved getting the top 2 pca vectors and plotting a scatterplot, the code snippet for which is posted below:



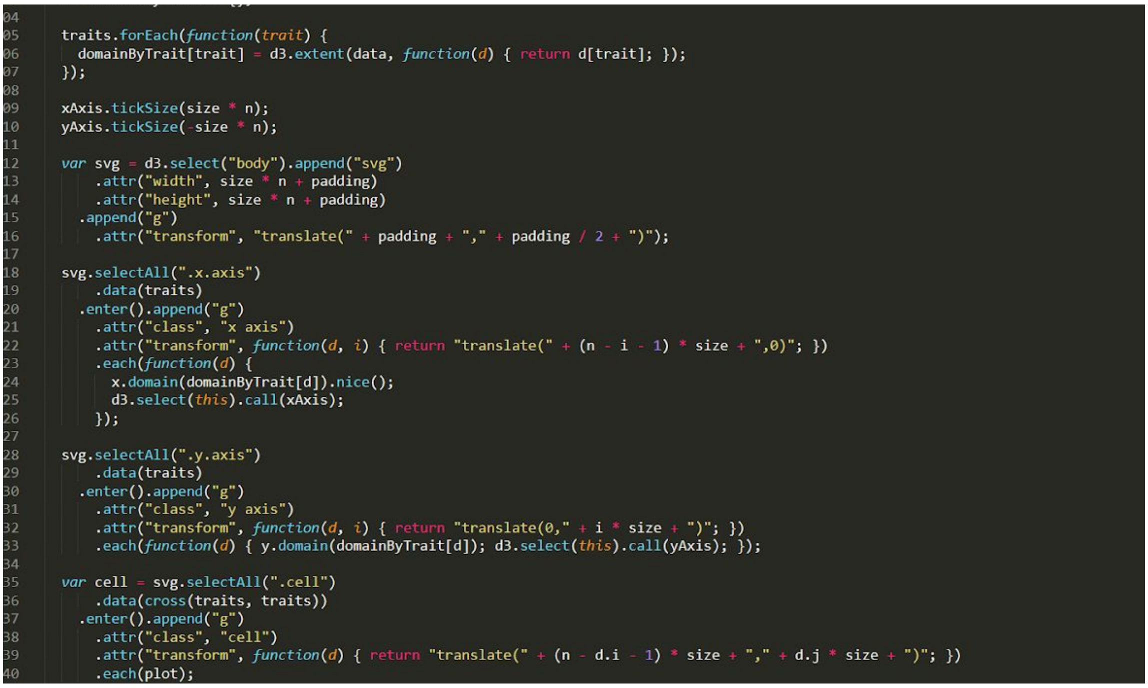
```javascript
// scatterplot
function scatterplotfunc(){
  $.post('/scatterplot', {'data': 'recvd'}, function(data_post){
    data_post = data_post["key"];
    scatterplot(data_post);
  } );
}
// scatterplotfunc()
function scatterplot(data){
  var margin = {top: 10, right: 30, bottom: 30, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
  .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 + ")");


  // Add X axis
  var x = d3.scaleLinear()
    .domain([0, d3.max(data, function(d){return d["axis1"]})])
    .range([ 0, width ]);
  svg.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));

  // Add Y axis
```

3c) This task involved plotting a scatterplotmatrix for the top 3 highest loadings and the code snippet for this is attached bellow



```
traits.forEach(function(trait) {
  domainByTrait[trait] = d3.extent(data, function(d) { return d[trait]; });
});

xAxis.tickSize(size * n);
yAxis.tickSize(-size * n);

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]).nice();
      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);
```