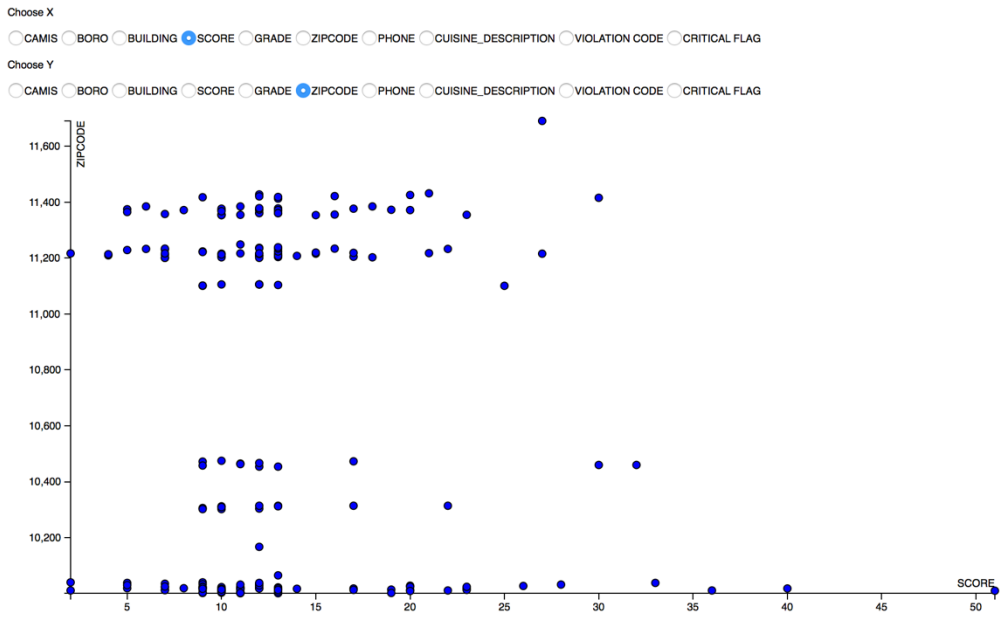
CSE332 Lab 3 Report

For more detailed implementation see in code.

1 Bivariate Scatterplot



Pros:

The bivariate plot graphs the relationship between two variables. The plot permits you to see the degree and pattern of relation between the two variables. Each point on the plot shows the X and Y scores for a single subject.

The Pattern of bivariate scatterplot shows the certain relationship (linear or non-linear) between x and y variable.

Cons:

The two variables X and Y must be quantitative. It doesn't work well with categorical data like mine. Also the bivariate Scatterplot only shows parts of relationships in data.

What I learn:

The example shows that the restaurant inspection score doesn't has any linear relation with the restaurant's zip code.

What it can't show:

Since my data was almost made up of categorical data. The bivariate plot doesn't help much here. It also can't show

Implementation Detail:

Step 1.

Use clean.py to extract 10 numeric attributes and store in "result.csv":

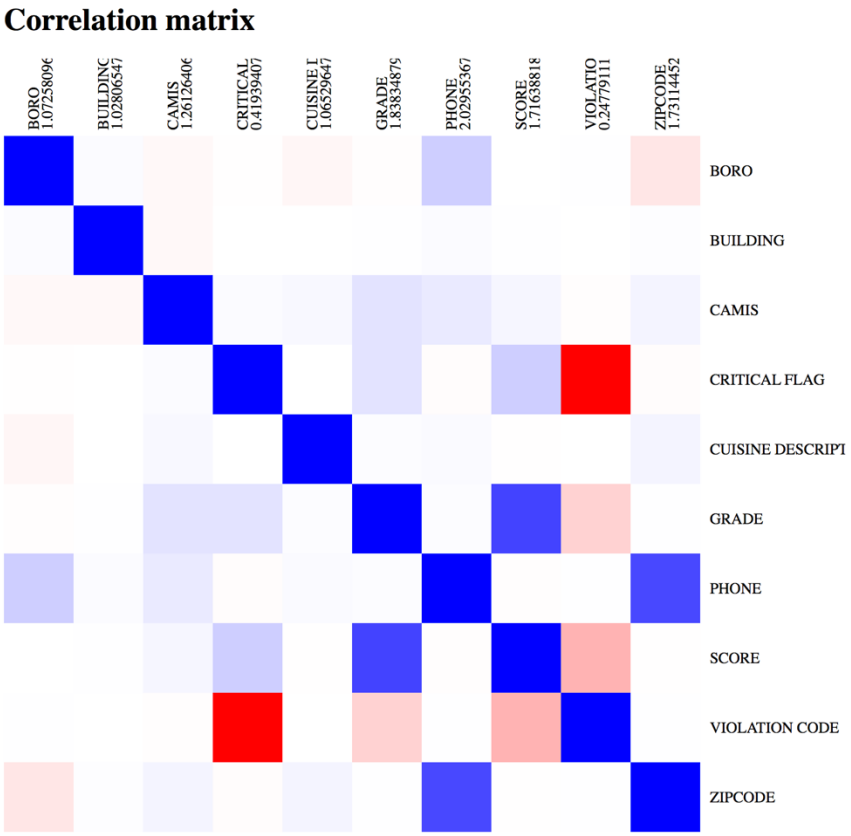
```
with open('data.csv') as csvfile:
    reader = csv.reader(csvfile)
    spamwriter.writerow( next(reader,None))
    for row in reader :
        if row[0]=='' or row[2]=='' or row[3] == '' or r
            continue
        else:
            row[2] = BORO.index(row[2])
            row[7] = CUISINE_TYPE.index(row[7])
            row[10] = row[10][:-1]
            row[12] = Critical_flag.index(row[12])
            row[14] = Grade.index(row[14])


        spamwriter.writerow(row)
result.close()
csvfile.close()
```

Step 2:

Use d3 to read data from “result.csv” and visualize it

```
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(x_axis_name);

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(y_axis_name);

svg.selectAll(".dot")
.data(global_data)
.enter().append("circle")
.attr("class","dot")
.attr("r",3.5)
.attr("cx",function(d){return x(d[x_axis_name
  ])})
.attr("cy",function(d){return y(d[y_axis_name
  ])})
.attr("fill","blue");
```

2 Correlation Matrix



Pros:

Shows all correlation between attributes. The color also shows the degree of correlations. It also sums the correlation of that attribute on its column

Cons:

Half of the matrix is unnecessary since the matrix is symmetric along its diagonal. It also only show pair-wise relation of the attributes.

What I learn:

It shows that my data has strong negative correlation with violation code and critical flag. That means the lower the code is the higher chance that the critical flag is 1.

What it doesn't show:

It only shows the degree of the correlation not the statistics.

Implementation Detail:

Step 1.

Use correlation.py to calculate the pair-wise correlation and store them in a json file.

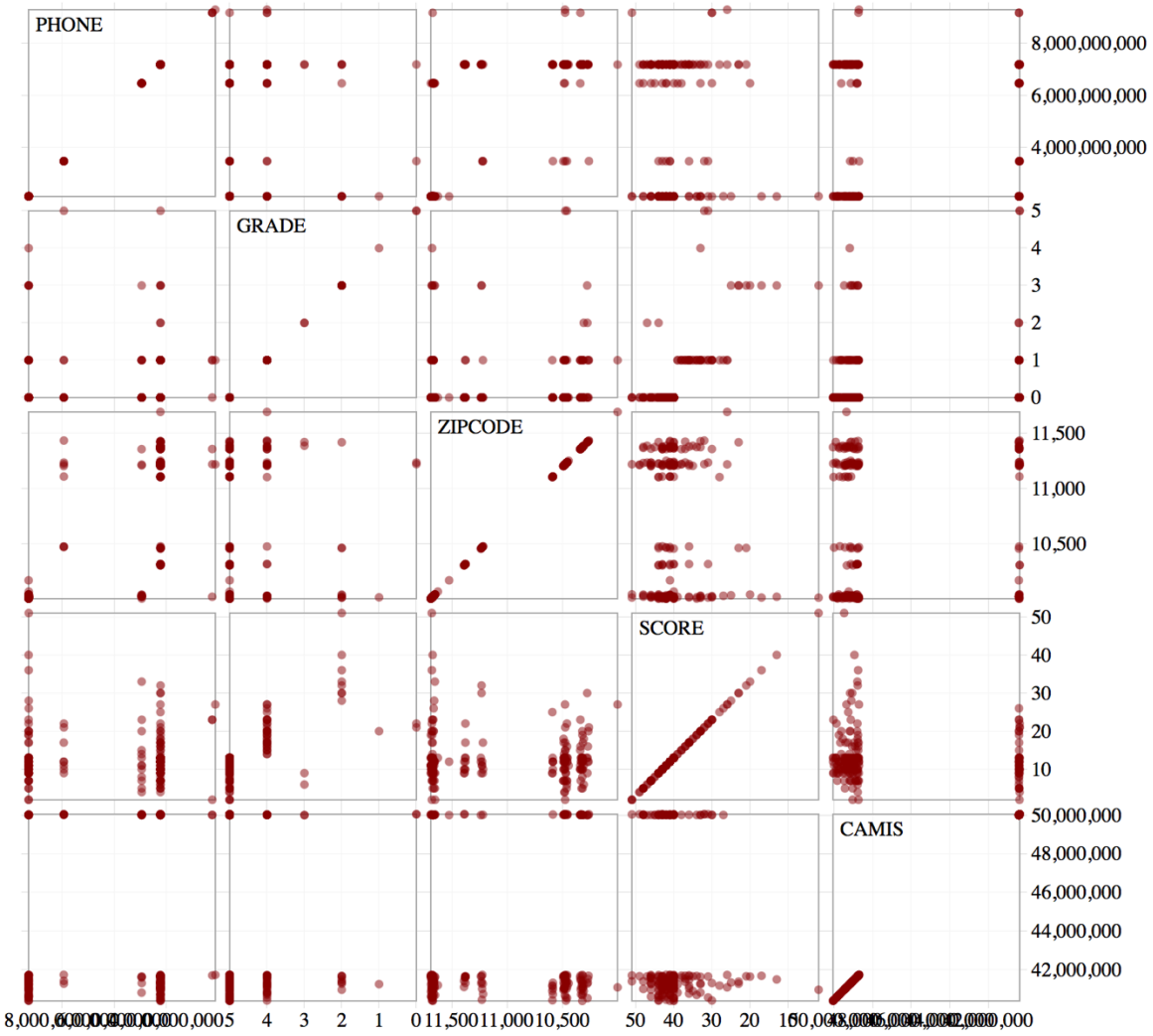
```
for name in names:
    print '{'
    print '"name":"'+name+'"'
    print '},'
print ']'

print '}]'

print '"links":['
for attr in result_ttr:
    for attr2 in attr:
        print'{'
        print '"source":'+str(result_ttr.index(attr))+',
        print '"target":'+str(attr.index(attr2))+','
        print '"value":'+str(attr2)
        print '},'
print "]"
```

Step 2.

Read the Json file and use d3 to visualize it.

```
  svg.append("rect")
      .attr("class", "background")
      .attr("width", width)
      .attr("height", height);
  var row = svg.selectAll(".row")
      .data(matrix)
      .enter().append("g")
      .attr("class", "row")
      .attr("transform", function(d, i) { return "translate(0," + x(i) + ")"; })
      .each(row);
  row.append("text")
      .attr("x", 730)
      .attr("y", x.rangeBand() / 2)
      .attr("dy", ".32em")
      .attr("text-anchor", "start")
      .text(function(d, i) { return nodes[i].name; });
  var column = svg.selectAll(".column")
      .data(matrix)
      .enter().append("g")
      .attr("class", "column")
      .attr("transform", function(d, i) { return "translate(" + x(i) + ")rotate(-90)"; });
  column.append("text")
      .attr("x", 6)
      .attr("y", x.rangeBand() / 2)
      .text(function(d, i) { return nodes[i].name; });
  column.append("text")
    .attr("x",6)
    .attr("y", x.rangeBand()/2+14)
    .text(function(d,i){return Math.abs(nodes[i].count);});
  function row(row) {
    var cell = d3.select(this).selectAll(".cell")
        .data(row.filter(function(d) { return d.z; }))
        .enter().append("rect")
       .attr("class", "cell")
       .attr("x", function(d) { return x(d.x); })
       .attr("width", x.rangeBand())
       .attr("height", x.rangeBand())
       .style("fill", function(d) { return c(d.z)});
  };
});
```

3 Scatter Plot Matrix



Pros:

Scatterplot matrices are a great way to roughly determine if you have a linear correlation between multiple variable. This is particularly helpful in pinpointing specific variables that might have similar correlation.

Cons:

Scatter plot matrices are not so good for looking at discrete variables

What I learn:

My plot shows top 5 attributes with the greatest correlation sum

Implementation detail:

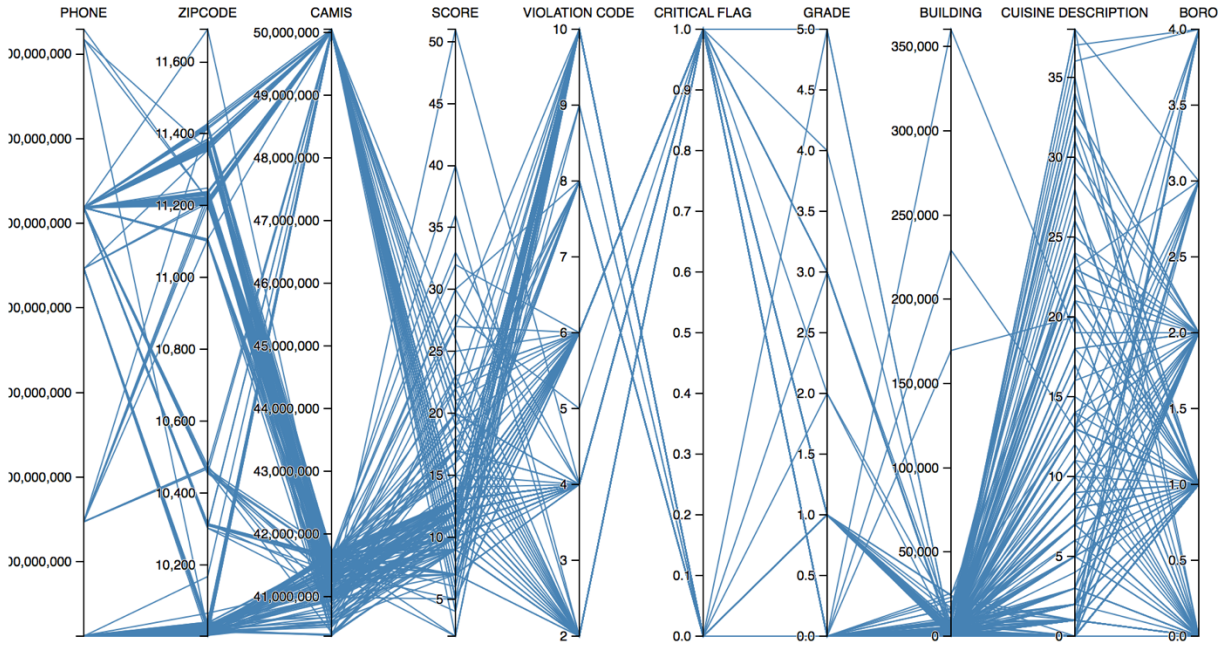
Step 1: make scales, x axis and y axis

```
var svg = d3.select("body").append("svg:svg")
.attr("width",1280)
.attr("height",800)
.append("svg:g")
.attr("transform","translate(359.5,69.5)");
svg.selectAll("g.x.axis")
.data(indexs)
.enter()
.append("svg:g")
.attr("class","x axis")
.attr("transform",function(d,i){
  return "translate(" +i *size +",0)";
})
.each(function(d){
  d3.select(this).call(axis.scale(x[d]).orient("bottom"));
});
svg.selectAll("g.y.axis")
.data(indexs)
.enter().append("svg:g")
.attr("class","y axis")
.attr("transform", function(d,i){
  return "translate(0," +i*size +")";
})
.each(function(d){
  d3.select(this).call(axis.scale(y[d]).orient("right"));
});
var cell = svg.selectAll("g.cell")
.data(cross(indexs, indexs))
.enter().append("svg:g")
.attr("class", "cell")
.attr("transform", function(d) { return "translate(" + d.i * size + "," + d.j * size + ")"; })
.each(plot);
```

Step 2: make dots:

```
function plot(p){
  var cell = d3.select(this);
  cell.append("svg: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("svg:circle")
  .attr("class","dot")
  .attr("cx",function(d){
    return x[p.x](d[p.x]);
  })
  .attr("cy",function(d){
    return y[p.y](d[p.y]);
  })
  .attr("r",3);

};
```

4 Parallel Coordinates



Pros:

Parallel coordinates is a way to represent high dimensional data to a 2-D dimensional. As data is represented in the form of a line. It becomes easy to capture the "trend" shown by data entries from the visualization.

Cons:

A lot data can be overlapped in line representation

What I learn:

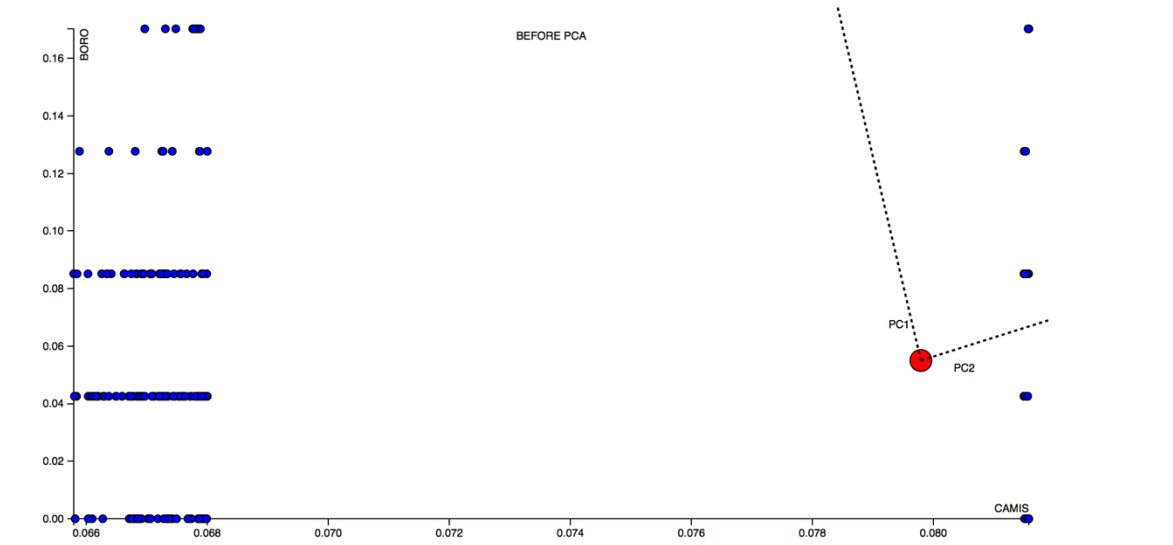
Look at BORO attribute, we can see that BORO[4] (state island) doesn't has much variety in Cuisine type(Cuisine Description) compare than other BOROS

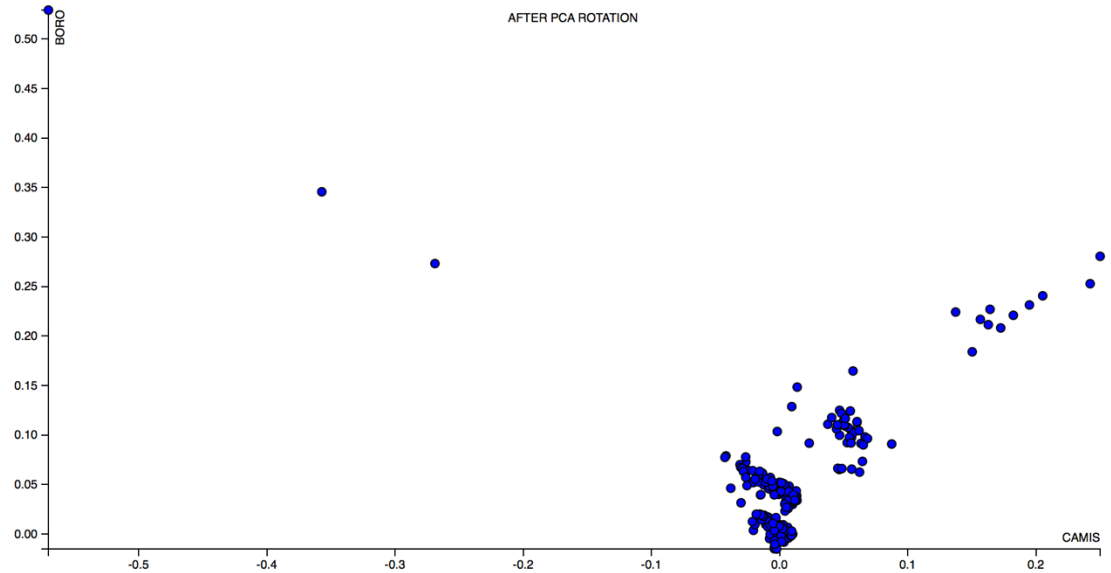
Implementation:

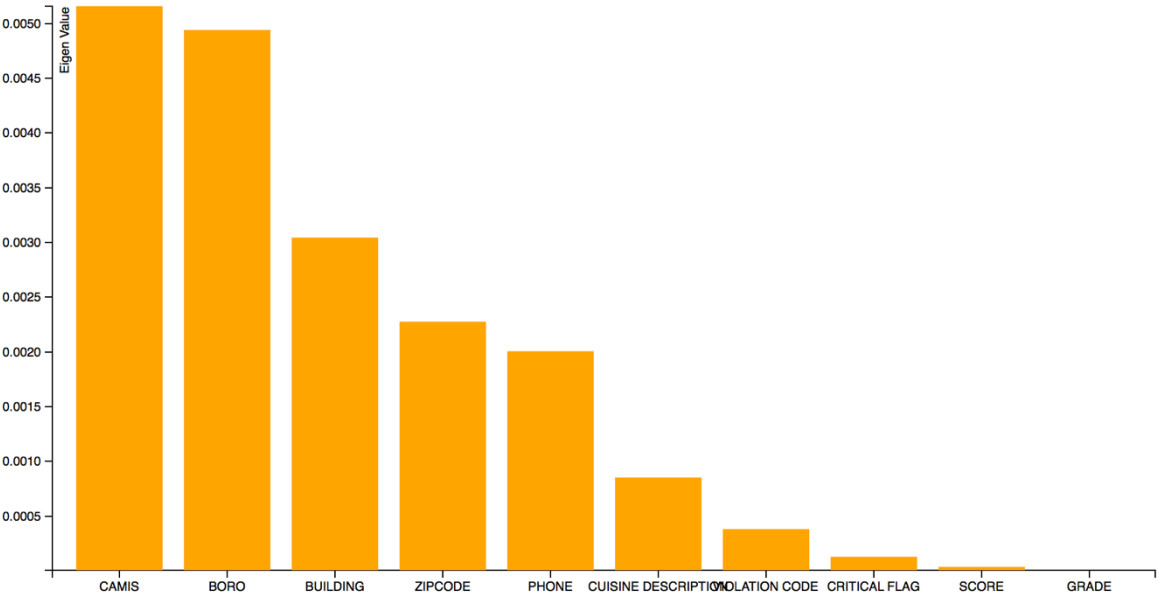
Make scale for different y-axis and visualize the data.

```
d3.csv("/result.csv",function(data){
    indexs = ["PHONE","ZIPCODE","CAMIS","SCORE","
      VIOLATION CODE","CRITICAL FLAG","GRADE","BUILDING"
      ,"CUISINE DESCRIPTION","BORO"];
    x.domain(indexs);
    indexs.forEach(function(index){
      data.forEach(function(d){
        d[index] = +d[index];
      });
      var value = function(d){return d[index];};
      domain = [d3.min(data,value),d3.max(data,value)],
      range = [height ,0];
      y[index] = d3.scale.linear().domain(domain).range(
        range);
    });

    line = svg.append("g")
                  .attr("class","line")
                  .selectAll("path")
                  .data(data)
                  .enter()
                  .append("path")
                  .attr("d",path);



    var g = svg.selectAll(".indexs")
              .data(indexs)
              .enter().append("g")
              .attr("class","indexs")
              .attr("transform",function(d){ return "
                translate("+x(d)+")";});

    g.append("g")
      .attr("class","axis")
      .each(function(d){d3.select(this).call(axis.scale(
        y[d]));})
      .append("text")
      .style("text-anchor","middle")
      .attr("y",-9)
      .text(function(d) {return d;});
});

function path(d) {
  return line(indexs.map(function(p) { return [x(p), y[p
    ](d[p])]; }));
};
```

5 PCA Plot







Pros:

PCA can deal with large datasets. PCA can be applied on all data-sets

Cons:

Non-linear is hard to model with PCA

PCA lost the original meaning of the attributes

What I learn:

I process the data with a python script to obtain the top 2 data PCA

The highest attribute is (CAMIS and BORO). The second plot is the data plot after rotation.

Implementation Detail:

Step 1. I wrote a python script (PCA.py) to calculate eigenvectors and eigenvalues.

```
# compute mean;

mean_0 = numpy.mean(all_samples[0,:])
mean_1 = numpy.mean(all_samples[1,:])
mean_2 = numpy.mean(all_samples[2,:])
mean_3 = numpy.mean(all_samples[3,:])
mean_4 = numpy.mean(all_samples[4,:])
mean_5 = numpy.mean(all_samples[5,:])
mean_6 = numpy.mean(all_samples[6,:])
mean_7 = numpy.mean(all_samples[7,:])
mean_8 = numpy.mean(all_samples[8,:])
mean_9= numpy.mean(all_samples[9,:])
```

```
# compute covariance matrix:

cov_mat = numpy.cov([all_samples[0,:],all_samples[1,:],all_samples[2,:],a

print "Covariance Matrix:\n", cov_mat


eig_val,eig_vec = numpy.linalg.eig(cov_mat)
```

Step 2.

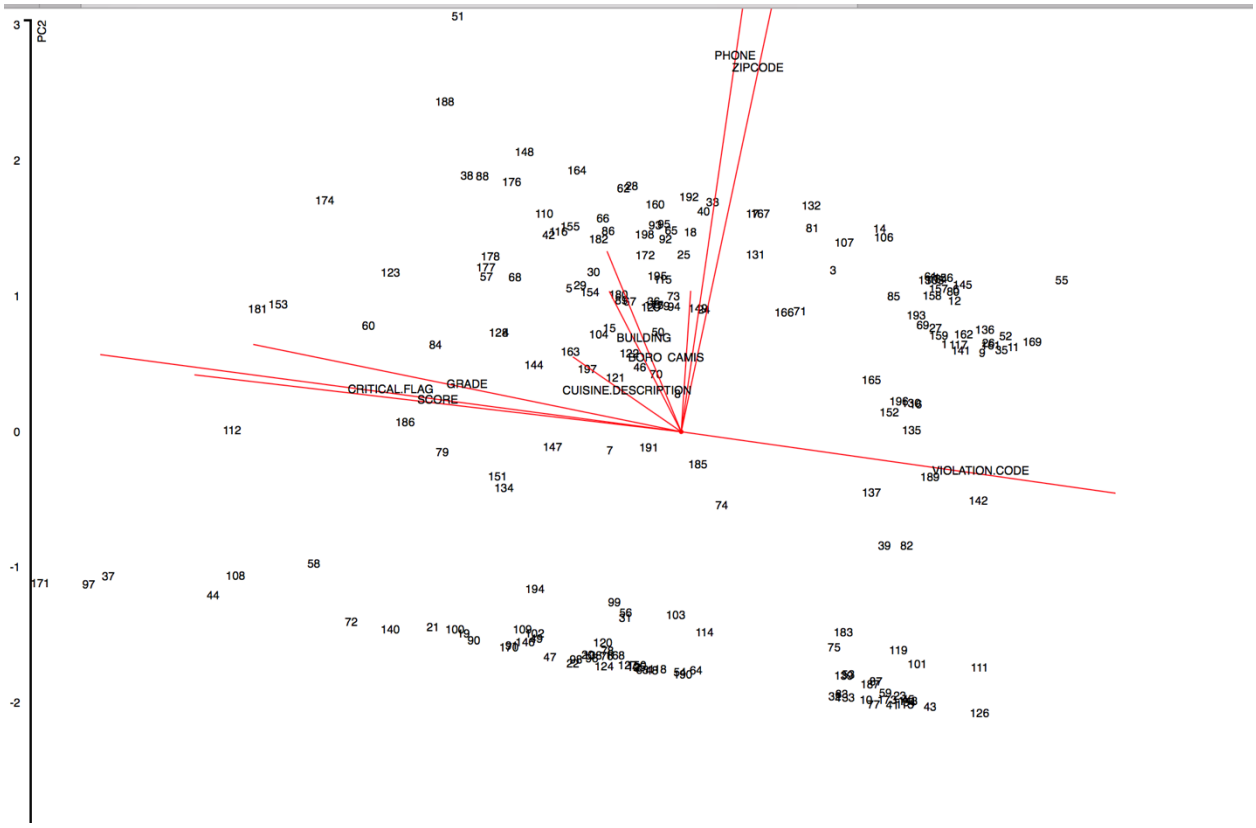
Plot the data as we did in scatter plot using the highest 2 eigenvalue attribute as axis

Step3.

Draw the Principle Component. I hard coded it using the data from python script

```
// PC1
svg.append("line")
.attr("x1", x(0.07979035))
.attr("y1", y(0.055))
.attr("x2", x(55.1224))
.attr("y2", y(364.77))
.attr("stroke-width", 2)
.attr("stroke", "black")
.style("stroke-dasharray",("3,3"));

//PC2
svg.append("line")
.attr("x1", x(0.07979035))
.attr("y1", y(0.055))
.attr("x2", x(-6.1925))
.attr("y2", y(559.836))
.attr("stroke-width", 2)
.attr("stroke", "black")
.style("stroke-dasharray",("3,3"));
```

6 Biplot



Pros:

The biplot data shows the other eigenvectors compares to the principle components as well as how the data is distributed among the two pca vectors

Cons:

Lost the original meaning of the data. This is only used to lower data demensions

What I learn:

The loading of each attribute

Implementation Detail:

Step 1.

Calculate the loadings and scores of the data attribute using R and store them in “loading.csv” and “scores.csv”

Step2.

Calculate the axis and plot the data from score.

```
d3.csv("score.csv",function(data){
  var PC_max = data[0].PC1;
  var PC_min = data[0].PC2;
  var local_max,local_min;
  data.forEach(function(d){
      local_max = Math.max(d.PC1,d.PC2);
      local_min = Math.min(d.PC1,d.PC2);
      if(local_max>PC_max){
        PC_max = local_max;
      }
      if(local_min<PC_min){
        PC_min = local_min;
      }
  });
  xscale.domain([PC_min,PC_max]);
  console.log(PC_min,PC_max);
  yscale.domain([PC_min,PC_max]);
  var xAxis = d3.svg.axis()
  .scale(xscale)
  .orient("bottom");

  var yAxis = d3.svg.axis()
  .scale(yscale)
  .orient("left");

     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("PC1");


   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("PC2");
```

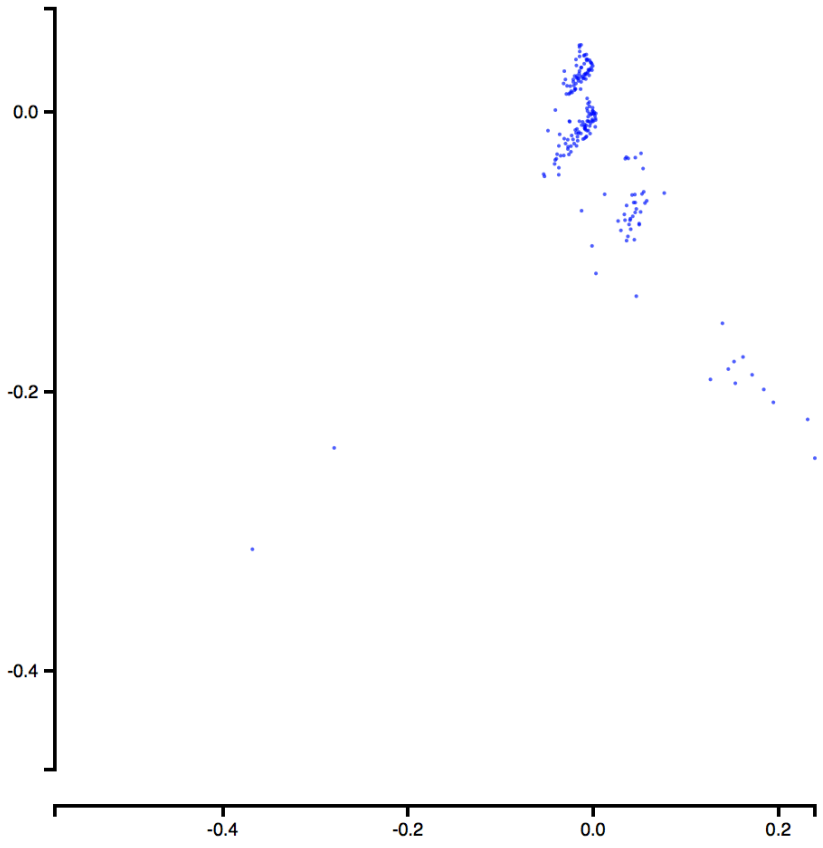
Step3.

Draw the lines from loading.

```
d3.csv("loadings.csv",function(data){
    svg.append("circle")
    .attr("class","dot")
    .attr("r",2)
    .attr("cx",xscale(0))
    .attr("cy",yscale(0))
    .attr("fill","red");
svg.selectAll(".line")
  .data(data)
  .enter()
  .append("line")
  .attr("class","line")
  .attr("x1", xscale(0))
  .attr("y1", yscale(0))
  .attr("x2", function(d){ return xscale(d.PC1*8)})
  .attr("y2", function(d){ return yscale(d.PC2*8);})
  .attr("stroke-width", 1)
  .attr("stroke", "red");
```

7 MDS (Euclidean)

MSD Euclidian



Pros

MDS is a way for us to analyze and kind of distance or similarity between data entries. There similarities can represent in different way.

Cons:

No relationship between data is displayed

What I learn:

From my data we can see that most of the data is clustered together. That means most data are similar to each other

Implementation detail:

Step 1.

Use R to calculate the distance matrix and stored in “Euclidean.csv”

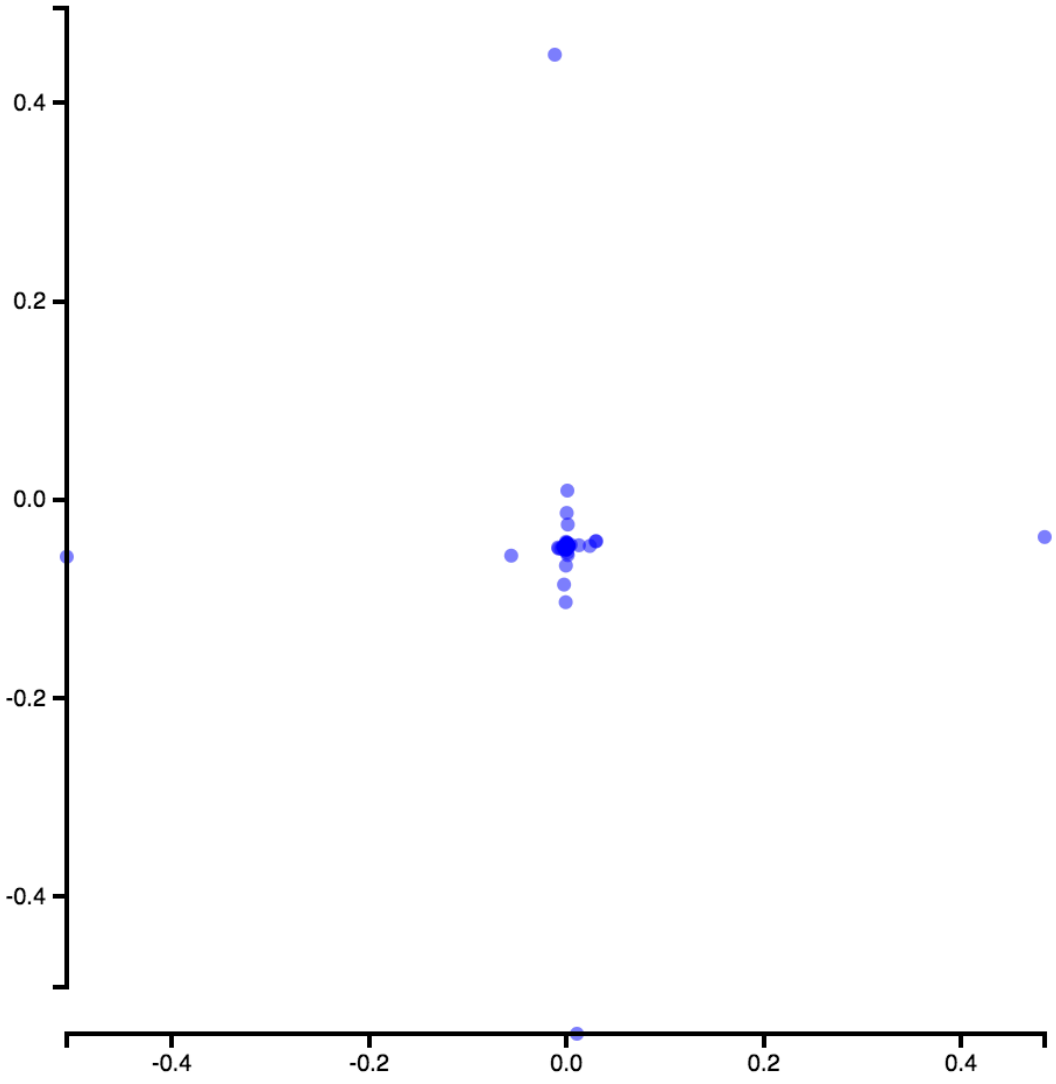
Step2.

Use the data to visualize in d3(similar to scatter plot)

```
});
var x_scale = d3.scale.linear().range([0,nPix]).domain([
  x_min,x_max]);
var y_scale = d3.scale.linear().range([nPix,0]).domain([
  y_min,y_max]);

var xAxis = d3.svg.axis().scale(x_scale).orient("bottom")
  .ticks(4);
svg.append("g").call(xAxis)
.attr("class", "axis")  //Assign "axis" class
.attr("transform","translate(" + mar[0] + "," + (nPix+mar
  [1])  + ")");

var yAxis = d3.svg.axis().scale(y_scale).orient("left").
  ticks(4);
svg.append("g").call(yAxis)
.attr("class", "axis")  //Assign "axis" class
.attr("transform","translate(" + mar[0] + "," + (mar[3])
    + ")");

var dots = sg.selectAll(".datapoint")
.data(data).enter()
.append("circle")
.attr("class", "datapoint")
.attr("cx",function(d) {return x_scale(d.x);})
.attr("cy",function(d) {return y_scale(d.y);})
.attr("id",function(d,i) {return "point" + i})
.attr("r",1);
```

8 MDS(1-|correlation|)

MSD 1- correlation



Similar to MDS Euclidean