

**Name = Tariq Bilal**

**Class = Bs(cs)**

**Id = 13588**

Q1. Implement a code of Genetic Algorithm in any language and show the output?

Ans )

```
import
java.util.Random;

/**
 *
 * @author Vijini
 */

//Main class
public class SimpleDemoGA {

    Population population = new Population();
    Individual fittest;
    Individual secondFittest;
    int generationCount = 0;

    public static void main(String[] args) {

        Random rn = new Random();

        SimpleDemoGA demo = new SimpleDemoGA();

        //Initialize population
        demo.population.initializePopulation(10);
```

```

//Calculate fitness of each individual
demo.population.calculateFitness();

System.out.println("Generation: " + demo.generationCount + " Fittest:
" + demo.population.fittest);

//While population gets an individual with maximum fitness
while (demo.population.fittest < 5) {
    ++demo.generationCount;

    //Do selection
    demo.selection();

    //Do crossover
    demo.crossover();

    //Do mutation under a random probability
    if (rn.nextInt()%7 < 5) {
        demo.mutation();
    }

    //Add fittest offspring to population
    demo.addFittestOffspring();

    //Calculate new fitness value
    demo.population.calculateFitness();

    System.out.println("Generation: " + demo.generationCount + "
Fittest: " + demo.population.fittest);
}

System.out.println("\nSolution found in generation " +
demo.generationCount);
System.out.println("Fitness: "+demo.population.getFittest().fitness);

```

```

        System.out.print("Genes: ");
        for (int i = 0; i < 5; i++) {
            System.out.print(demo.population.getFittest().genes[i]);
        }

        System.out.println("");

    }

    //Selection
    void selection() {

        //Select the most fittest individual
        fittest = population.getFittest();

        //Select the second most fittest individual
        secondFittest = population.getSecondFittest();
    }

    //Crossover
    void crossover() {
        Random rn = new Random();

        //Select a random crossover point
        int crossOverPoint =
rn.nextInt(population.individuals[0].geneLength);

        //Swap values among parents
        for (int i = 0; i < crossOverPoint; i++) {
            int temp = fittest.genes[i];
            fittest.genes[i] = secondFittest.genes[i];
            secondFittest.genes[i] = temp;
        }
    }

```

```

}

//Mutation
void mutation() {
    Random rn = new Random();

    //Select a random mutation point
    int mutationPoint = rn.nextInt(population.individuals[0].geneLength);

    //Flip values at the mutation point
    if (fittest.genes[mutationPoint] == 0) {
        fittest.genes[mutationPoint] = 1;
    } else {
        fittest.genes[mutationPoint] = 0;
    }

    mutationPoint = rn.nextInt(population.individuals[0].geneLength);

    if (secondFittest.genes[mutationPoint] == 0) {
        secondFittest.genes[mutationPoint] = 1;
    } else {
        secondFittest.genes[mutationPoint] = 0;
    }
}

//Get fittest offspring
Individual getFittestOffspring() {
    if (fittest.fitness > secondFittest.fitness) {
        return fittest;
    }
    return secondFittest;
}

```

```

//Replace least fittest individual from most fittest offspring
void addFittestOffspring() {

    //Update fitness values of offspring
    fittest.calcFitness();
    secondFittest.calcFitness();

    //Get index of least fit individual
    int leastFittestIndex = population.getLeastFittestIndex();

    //Replace least fittest individual from most fittest offspring
    population.individuals[leastFittestIndex] = getFittestOffspring();
}

}

```

```

//Individual class
class Individual {

    int fitness = 0;
    int[] genes = new int[5];
    int geneLength = 5;

    public Individual() {
        Random rn = new Random();

        //Set genes randomly for each individual
        for (int i = 0; i < genes.length; i++) {
            genes[i] = Math.abs(rn.nextInt() % 2);
        }
    }
}

```

```

        fitness = 0;
    }

    //Calculate fitness
    public void calcFitness() {

        fitness = 0;
        for (int i = 0; i < 5; i++) {
            if (genes[i] == 1) {
                ++fitness;
            }
        }
    }
}

//Population class
class Population {

    int popSize = 10;
    Individual[] individuals = new Individual[10];
    int fittest = 0;

    //Initialize population
    public void initializePopulation(int size) {
        for (int i = 0; i < individuals.length; i++) {
            individuals[i] = new Individual();
        }
    }

    //Get the fittest individual
    public Individual getFittest() {
        int maxFit = Integer.MIN_VALUE;
        int maxFitIndex = 0;
        for (int i = 0; i < individuals.length; i++) {
            if (maxFit <= individuals[i].fitness) {

```

```

        maxFit = individuals[i].fitness;
        maxFitIndex = i;
    }
}
fittest = individuals[maxFitIndex].fitness;
return individuals[maxFitIndex];
}

//Get the second most fittest individual
public Individual getSecondFittest() {
    int maxFit1 = 0;
    int maxFit2 = 0;
    for (int i = 0; i < individuals.length; i++) {
        if (individuals[i].fitness > individuals[maxFit1].fitness) {
            maxFit2 = maxFit1;
            maxFit1 = i;
        } else if (individuals[i].fitness > individuals[maxFit2].fitness)
    {
        maxFit2 = i;
    }
    }
    return individuals[maxFit2];
}

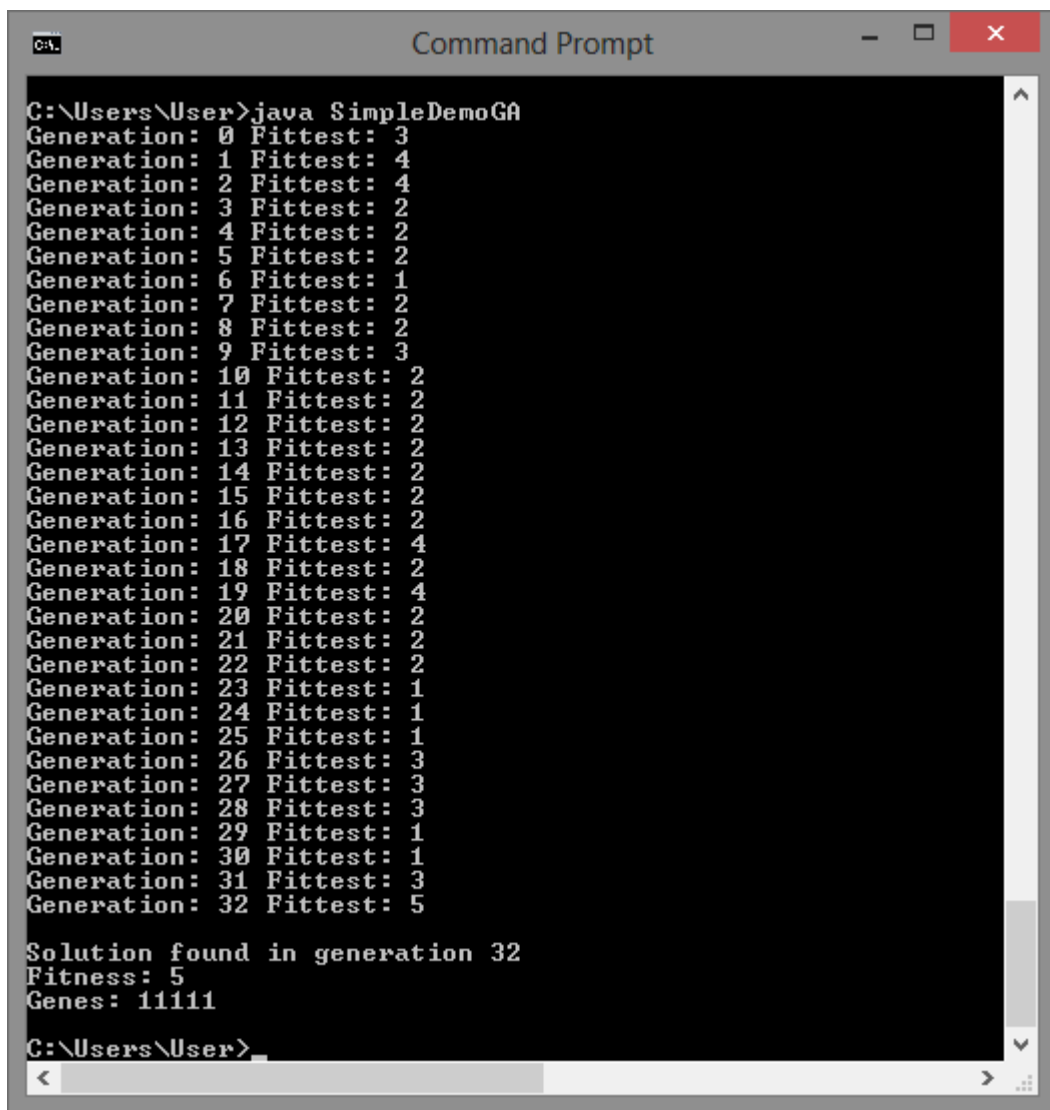
//Get index of least fittest individual
public int getLeastFittestIndex() {
    int minFitVal = Integer.MAX_VALUE;
    int minFitIndex = 0;
    for (int i = 0; i < individuals.length; i++) {
        if (minFitVal >= individuals[i].fitness) {
            minFitVal = individuals[i].fitness;
            minFitIndex = i;
        }
    }
    return minFitIndex;
}

//Calculate fitness of each individual
public void calculateFitness() {

```

```
        for (int i = 0; i < individuals.length; i++) {
            individuals[i].calcFitness();
        }
        getFittest();
    }
}
```

## Output



```
C:\Users\User>java SimpleDemoGA
Generation: 0 Fittest: 3
Generation: 1 Fittest: 4
Generation: 2 Fittest: 4
Generation: 3 Fittest: 2
Generation: 4 Fittest: 2
Generation: 5 Fittest: 2
Generation: 6 Fittest: 1
Generation: 7 Fittest: 2
Generation: 8 Fittest: 2
Generation: 9 Fittest: 3
Generation: 10 Fittest: 2
Generation: 11 Fittest: 2
Generation: 12 Fittest: 2
Generation: 13 Fittest: 2
Generation: 14 Fittest: 2
Generation: 15 Fittest: 2
Generation: 16 Fittest: 2
Generation: 17 Fittest: 4
Generation: 18 Fittest: 2
Generation: 19 Fittest: 4
Generation: 20 Fittest: 2
Generation: 21 Fittest: 2
Generation: 22 Fittest: 2
Generation: 23 Fittest: 1
Generation: 24 Fittest: 1
Generation: 25 Fittest: 1
Generation: 26 Fittest: 3
Generation: 27 Fittest: 3
Generation: 28 Fittest: 3
Generation: 29 Fittest: 1
Generation: 30 Fittest: 1
Generation: 31 Fittest: 3
Generation: 32 Fittest: 5

Solution found in generation 32
Fitness: 5
Genes: 11111
C:\Users\User>
```

Q2. Implement a code of Fuzzy logic in any language and show the output?



```
Ans) #include <iostream>
```

```
#include <cmath>
```

```
#include <cstring>
```

```
const double cdMinimumPrice =0;
```

```
const double cdMaximumPrice =70;
```

```
using namespace std;
```

```
class CFuzzyFunction
```

```
{
```

```
protected :
```

```
    double dLeft, dRight;
```

```
    char  cType;
```

```
    char* sName;
```

```
public:
```

```
    CFuzzyFunction(){};
```

```
    virtual ~CFuzzyFunction(){ delete [] sName; sName=NULL;}
```

```
    virtual void
```

```
    setInterval(double l,
```

```
                double r)
```

```
    {dLeft=l; dRight=r;}
```

```
    virtual void
```

```
    setMiddle( double dL=0,
```

```
              double dR=0)=0;
```

```
    virtual void
```

```
    setType(char c)
```

```
{ cType=c;}
```

```
virtual void
```

```
setName(const char* s)
```

```
{
```

```
    sName = new char[strlen(s)+1];
```

```
    strcpy(sName,s);
```

```
}
```

```
bool
```

```
isDotInInterval(double t)
```

```
{
```

```
    if((t>=dLeft)&&(t<=dRight)) return true; else return false;
```

```
}
```

```
char getType(void)const{ return cType;}
```

```
void
```

```
getName() const
```

```
{
```

```
    cout<<sName<<endl;
```

```
}
```

```
virtual double getValue(double t)=0;
```

```
};
```

```
class CTriangle : public CFuzzyFunction
```

```
{
```

```
private:
```

```
    double dMiddle;
```

```

public:
    void
    setMiddle(double dL, double dR)
    {
        dMiddle=dL;
    }

    double
    getValue(double t)
    {
        if(t<=dLeft)
            return 0;
        else if(t<dMiddle)
            return (t-dLeft)/(dMiddle-dLeft);
        else if(t==dMiddle)
            return 1.0;
        else if(t<dRight)
            return (dRight-t)/(dRight-dMiddle);
        else
            return 0;
    }
};

```

```

class CTrapezoid : public CFuzzyFunction

```

```

{
private:
    double dLeftMiddle, dRightMiddle;

public:
    void
    setMiddle(double dL, double dR)

```

```

    {
        dLeftMiddle=dL; dRightMiddle=dR;
    }

    double
    getValue(double t)
    {
        if(t<=dLeft)
            return 0;
        else if(t<dLeftMiddle)
            return (t-dLeft)/(dLeftMiddle-dLeft);
        else if(t<=dRightMiddle)
            return 1.0;
        else if(t<dRight)
            return (dRight-t)/(dRight-dRightMiddle);
        else
            return 0;
    }
};

```

```

int
main(void)
{
    CFuzzyFunction *FuzzySet[3];

    FuzzySet[0] = new CTrapezoid;
    FuzzySet[1] = new CTriangle;
    FuzzySet[2] = new CTrapezoid;

    FuzzySet[0]->setInterval(-5,30);
    FuzzySet[0]->setMiddle(0,20);

```

```
FuzzySet[0]->setType('r');
```

```
FuzzySet[0]->setName("low_price");
```

```
FuzzySet[1]->setInterval(25,45);
```

```
FuzzySet[1]->setMiddle(35,35);
```

```
FuzzySet[1]->setType('t');
```

```
FuzzySet[1]->setName("good_price");
```

```
FuzzySet[2]->setInterval(40,75);
```

```
FuzzySet[2]->setMiddle(50,70);
```

```
FuzzySet[2]->setType('r');
```

```
FuzzySet[2]->setName("to_expensive");
```

```
double dValue;
```

```
do
```

```
{
```

```
    cout<<"\nInput the value->"; cin>>dValue;
```

```
    if(dValue<cdMinimumPrice) continue;
```

```
    if(dValue>cdMaximumPrice) continue;
```

```
for(int i=0; i<3; i++)
```

```
{
```

```
    cout<<"\nThe dot="<<dValue<<endl;
```

```
    if(FuzzySet[i]->isDotInInterval(dValue))
```

```
        cout<<"In the interval";
```

```
    else
```

```
        cout<<"Not in the interval";
```

```
    cout<<endl;
```

```
cout<<"The name of function is"<<endl;
```

```

FuzzySet[i]->getName();
cout<<"and the membership is=";

cout<<FuzzySet[i]->getValue(dValue);

}

}

while(true);

return EXIT_SUCCESS;

```

### }Output

The screenshot shows a C++ IDE with the following source code in 'FUZZY main.cpp':

```

1 #include <iostream>
2 #include <cmath>
3 #include <cstring>
4
5 const double cdMinimumPrice =0;
6 const double cdMaximumPrice =76;
7
8 using namespace std;
9
10 class CFuzzyFunction

```

The execution output window shows the following text:

```

Input the value->12
The dot=12
In the interval
The name of function is
low_price
and the membership is=1
The dot=12
Not in the interval
The name of function is
good_price
and the membership is=0
The dot=12
Not in the interval
The name of function is
Absor_to_expensive
and the membership is=0
Input the value->_

```

Q4. Give solved example of hierarchical Clustering?

Ans) **Example:**

**Agglomerative Hierarchical Clustering**

## Example of Complete Linkage Clustering

Grouping begins by figuring a separation between each pair of units that you need to bunch. A separation lattice will be symmetric (in light of the fact that the separation among x and y is equivalent to the separation among y and x) and will have zeroes on the corner to corner (on the grounds that each thing is separation zero from itself). The table underneath is a case of a separation framework. Just the lower triangle is appeared, on the grounds that the upper triangle can be filled in by reflection.

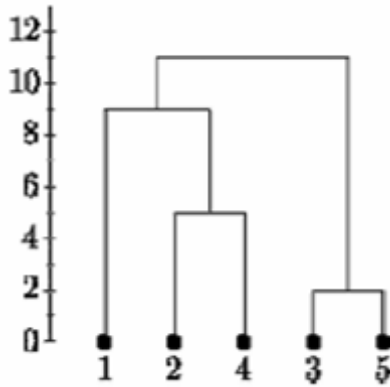
	1	2	3	4	5
1	0				
2	9	0			
3	3	7	0		
4	6	5	9	0	
5	11	10	2	8	0

Now lets start clustering. The smallest distance is between three and five and they get linked up or merged first into a the cluster '35'.

To acquire the new separation lattice, we have to evacuate the 3 and 5 sections, and supplant it by a passage "35" . Since we are utilizing finished linkage grouping, the separation among "35" and each other thing is the limit of the separation between this thing and 3 and this thing and 5. For instance,  $d(1,3)= 3$  and  $d(1,5)=11$ . Along these lines,  $D(1,"35")=11$ . This gives us the new separation framework. The things with the littlest separation get grouped straightaway. This will be 2 and 4.

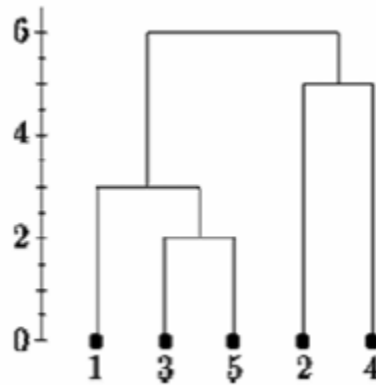
	35	1	2	4
35	0			
1	11	0		
2	10	9	0	
4	9	6	5	0

Proceeding along these lines, after 6 stages, everything is bunched. This is summed up beneath. On this plot, the y-pivot shows the separation between the articles at the time they were grouped. This is known as the bunch tallness. Various representations utilize various proportions of bunch tallness.



Complete Linkage

Below is the single linkage dendrogram for the same distance matrix. It starts with cluster "35" but the distance between "35" and each item is now the minimum of  $d(x,3)$  and  $d(x,5)$ . So  $c(1, "35")=3$ .



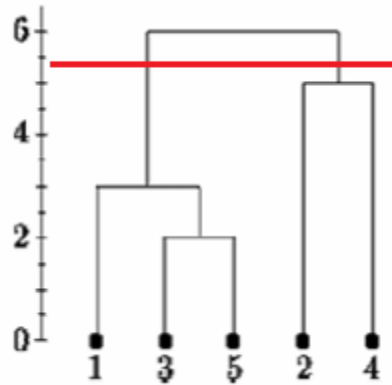
Single Linkage

## Determining clusters

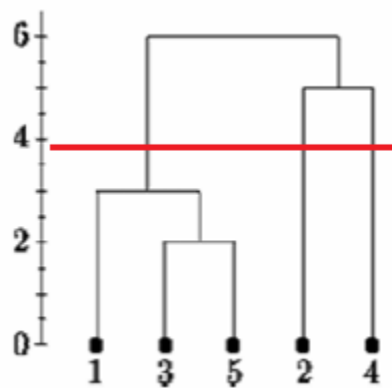
One of the problems with hierarchical clustering is that there is no objective way to say how many clusters there are.

If we cut the single linkage tree at the point shown below, we would say that there are two clusters.





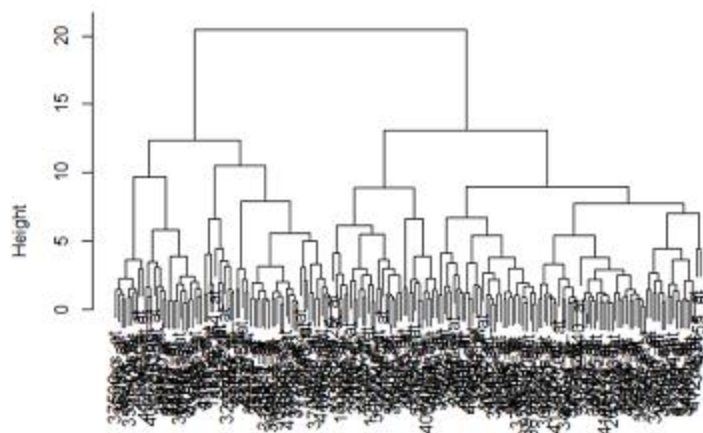
However, if we cut the tree lower we might say that there is one cluster and two singletons.



There is no usually settled upon approach to choose where to cut the tree. How about we take a gander at some genuine information. In schoolwork 5 we consider quality articulation in 4 districts of 3 human and 3 chimpanzee minds. The RNA was hybridized to Affymetrix human quality articulation microarrays. We standardized the information utilizing RMA and did a differential articulation examination utilizing LIMMA. Here we chose the 200 most altogether differentially communicated qualities from the examination. We group all the differentially communicated qualities dependent on their mean articulation in every one of the 8 animal categories by mind area medicines

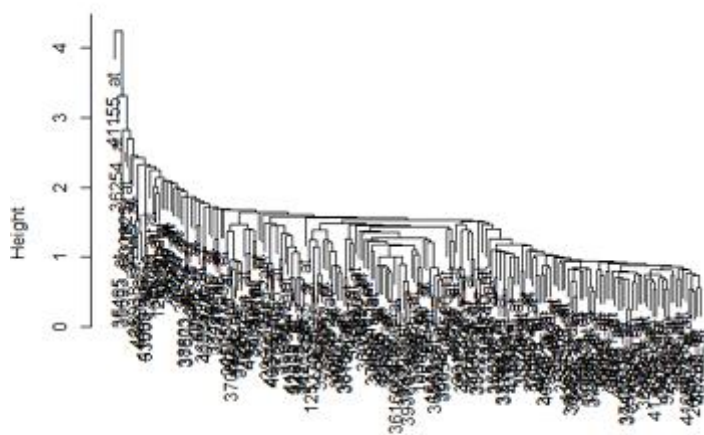
Here are the bunches dependent on Euclidean separation and relationship separation, utilizing total and single linkage grouping.

### Euclidean, Complete

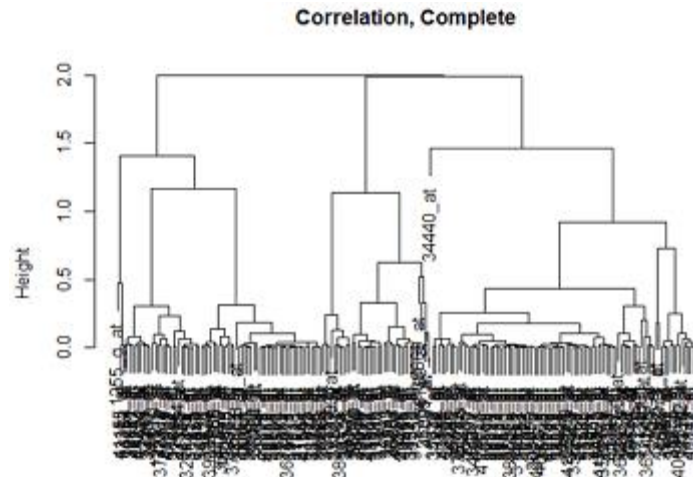


```
dist((geneMeanDE))  
hclust ("complete")
```

### Euclidean, Single

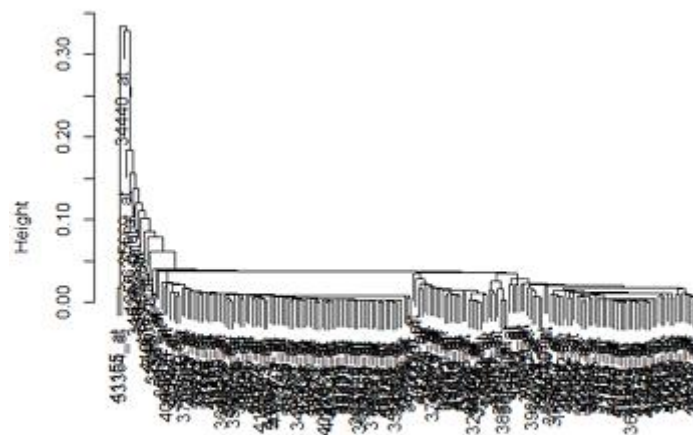


```
dist((geneMeanDE))  
hclust ("single")
```



```
as.dist(1 - cor(t(geneMeanDE)))
hclust("complete")
```

**Correlation, Single**



```
as.dist(1 - cor(t(geneMeanDE)))
hclust("single")
```

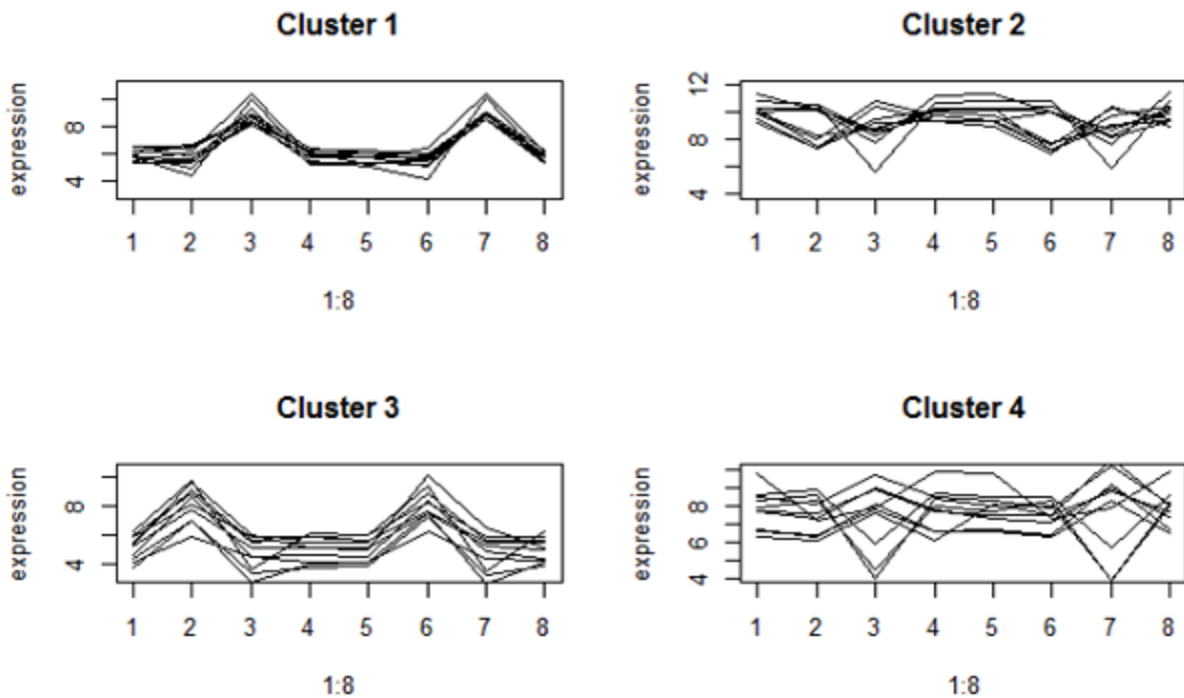
We can see that the grouping design for complete linkage separation will in general make minimized bunches of bunches, while single linkage will in general include each point in turn to the group, making long tacky bunches. As we would anticipate from our conversation of separations, Euclidean separation and connection separation produce altogether different dendrograms.

Progressive grouping doesn't disclose to us what number of bunches there are, or where to slice the dendrogram to shape bunches. In R there is a capacity `cuttree` which will cut a tree into bunches at a predefined stature. Nonetheless, in view of our representation, we may want to cut the long branches at various statures. Regardless, there is a decent measure of subjectivity in figuring out which branches ought to and ought not be sliced to frame separate groups.

## Understanding the clusters

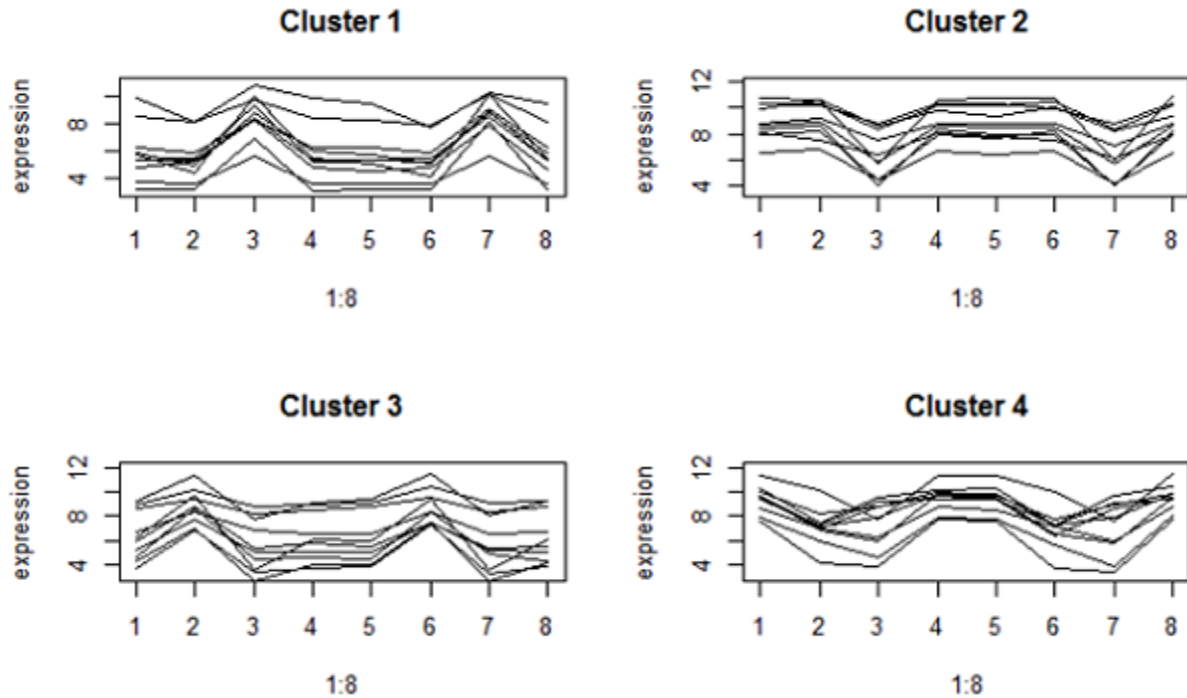
To comprehend the groups we as a rule plot the  $\log_2(\text{expression})$  estimations of the qualities in the bunch, or at the end of the day, plot the quality articulations over the examples. (The numbering in these diagrams are absolutely subjective.) Even however the medicines are unordered, I as a rule interface the focuses originating from a solitary component to make the example more clear. These are called profile plots.

Here is a portion of the profile plots from complete linkage bunching when we utilized Euclidean separation:



These look very tightly packed. However, clusters 2 and 4 have genes with different up and down patterns, because they have about the same mean expression. Cluster 2 are very highly expressed genes.

Here's what we got when we use correlation distance:



These are a lot looser on the y-pivot since relationship centers around the articulation design, not the mean. In any case, all the qualities in a similar group have a pinnacle or valley in similar medicines (which are cerebrum areas by species mixes). Groups 1 and 2 are qualities that are separately higher or lower in the cerebellum contrasted with other cerebrum locales in the two species.

## Selecting a gene list

On a fundamental level it is conceivable to bunch all the qualities, despite the fact that imagining a tremendous dendrogram may be tricky. Generally, some sort of primer examination, for example, differential articulation investigation is utilized to choose qualities for grouping. There are valid justifications to do as such, in spite of the fact that there are additionally a few admonitions.

Regularly in quality articulation, the separation metric utilized is relationship separation. Relationship separation is equivalent to focusing and scaling the information, and afterward utilizing Euclidean separation. When there are methodical treatment impacts, we expect the changeability of quality articulation from treatment to treatment to be a blend of precise treatment impacts and commotion. When there are no treatment impacts, the inconstancy of quality articulation is only because of commotion. Be that as it may, focusing and scaling the information puts all variability on a similar scale. Subsequently qualities that show an example because of chance are not discernable from those that have a precise part.

As we have seen, connection separation has preferred natural understanding over Euclidean separation for quality articulation considers, however a similar scaling that makes it helpful for finding organically important examples of quality guideline presents deceptive outcomes for qualities that don't differentially communicate. Choosing qualities dependent on differential articulation investigation evacuates qualities which are probably going to have just possibility designs. This should upgrade the examples found in the quality bunches.

As an admonition, be that as it may, consider the impacts of quality determination on bunching tests or medicines. The chose qualities are those which test positive in differential articulation investigation. Utilization of those qualities to bunch tests is one-sided towards grouping the examples by treatment.