Output

K-means

K = 1

```
Centoid ID: 0
[-8.576208E-9, -2.0582899E-8, 1.0648791E-8, 1.1577881E-8, 2.944498E-8, -1.6866542E-8,
1.1434944E-8, 9.707434]

Centroid ID: 0
Mean: -7.266187084474388E-10 -1.4988009783451911E-9 -3.263788980751853E-9
2.6028777172517604E-9 7.357320135486734E-11 -1.7398081573870697E-9 -
1.1582733259063104E-9
Standard Deviation: 1.0000000028910723 0.9999999986479469 1.000000002171349
0.9999999991064515 0.9999999989887518 0.999999994475209 1.0000000052714868
The WCSS of this run is: 6550.306958363558
```

K = 2

```
Centoid ID: 0
[0.7199749, 0.7288313, 0.7330198, 0.7575324, 0.70148194, 0.7227224, 0.7717051,
11.922681]
Centoid ID: 1
[-0.6263453, -0.6340506, -0.63769376, -0.65901923, -0.61025786, -0.6287361, -
0.6713485, 7.780269]

Centroid ID: 0
Mean: 0.7199746357628866 0.7288311669329897 0.733019487185567 0.7575323027113402
0.7014820115334021 0.7227224195309279 0.771704916072165
Standard Deviation: 0.6082606837415592 0.5905594300039075 0.6981835153451953
0.8736320659897318 0.9425340063375593 0.926771654027675 0.8395356717131262
Centroid ID: 1
Mean: -0.6263456486143499 -0.6340504350224215 -0.6376940891255606 -0.6590191284331839
-0.6102578933040359 -0.6287360991681614 -0.6713486735470852
Standard Deviation: 0.8397400531379371 0.8382386600023491 0.7563448861951936
0.5221138636572179 0.545386478803369 0.5230958485093496 0.5366272537303093
The WCSS of this run is: 3402.184608263684
```

K = 4

```
Centoid ID: 0
[0.5044342, 0.5224293, 0.7165549, 0.551396, 0.20265846, 0.5141535, 0.8353767,
15.565217]
Centoid ID: 1
[-0.015325217, -0.007423505, -0.09344818, -0.24572214, -0.22867739, -0.23792575, -
0.22717847, 9.350283]
Centoid ID: 2
```

[-1.3354397, -1.3525186, -1.2554371, -1.0931727, -1.0142465, -1.054495, -1.1418792,
6.3958335]
Centoid ID: 3
[1.0990902, 1.0931025, 1.0622532, 1.2558484, 1.3114412, 1.2213601, 1.1367722,
10.846939]

Centroid ID: 0
Mean: 0.5044343109782609 0.5224293620869566 0.7165548471956522 0.5513959870978261
0.20265841545652175 0.5141535532173913 0.8353765508043478
Standard Deviation: 0.6459245139412967 0.6166066305849668 0.6879918904653682
0.8878705990026936 0.7020953274148911 0.928299782360449 1.077040627780403
Centroid ID: 1
Mean: -0.015325227864406762 -0.007423502265536726 -0.09344817983050847 -
0.24572217870677965 -0.22867736374022601 -0.23792581561299436 -0.22717852440677966
Standard Deviation: 0.4631941499540872 0.46432775410160976 0.48981074166780836
0.42126628949137757 0.4679694371145397 0.4190891091917975 0.4013554322167145
Centroid ID: 2
Mean: -1.3354398302708332 -1.3525185436458333 -1.2554376073541667 -1.0931723205208332
-1.0142462530447918 -1.0544949008333333 -1.1418790404166668
Standard Deviation: 0.6848386751514545 0.6705707177819956 0.6016857558380254
0.31014648028282216 0.3730018803288089 0.2985214964106489 0.3005839288604996
Centroid ID: 3
Mean: 1.0990899023469387 1.0931019266326532 1.0622532020816327 1.255848511020408
1.3114412917755103 1.2213603639795918 1.136772090612245
Standard Deviation: 0.3740627321328685 0.36540474264763334 0.5655844996442289
0.690116193296166 0.8097933033424379 0.8041583955756157 0.5904280179950728
The WCSS of this run is: 1735.0119407478992

K = 8

Centoid ID: 0
[-0.12436092, -0.099100515, -0.10848001, -0.28928155, -0.40035695, -0.28635573, -
0.15555002, 12.428572]
Centoid ID: 1
[-0.20808752, -0.19682276, -0.2454731, -0.44444045, -0.41758707, -0.42657247, -
0.41616154, 8.703297]
Centoid ID: 2
[-0.9870299, -1.001399, -0.9444676, -0.94536483, -0.8763777, -0.91123575, -0.9916228,
7.0307693]
Centoid ID: 3
[1.3653133, 1.3511654, 1.3961736, 1.8011868, 1.8732461, 1.794286, 1.5810652,
11.22449]
Centoid ID: 4
[0.6143714, 0.6310341, 0.51665074, 0.4253213, 0.41622642, 0.396029, 0.4663683,
10.712121]
Centoid ID: 5
[-2.0659766, -2.0887365, -1.9074717, -1.4030912, -1.3033258, -1.3548771, -1.4569328,
5.064516]
Centoid ID: 6
[0.7768948, 0.7581303, 1.0513014, 0.9133086, 0.45746967, 0.84016585, 1.2532961,
17.25]
Centoid ID: 7

[0.64613134, 0.63233507, 0.48649737, 0.50749195, 0.58919704, 0.47283605, 0.3995585, 8.619047]

Centroid ID: 0
Mean: -0.12436092297959184 -0.09910051630612243 -0.1084800024693878 -0.28928153710408167 -0.4003569901836735 -0.2863556665714285 -0.15555000495918364
Standard Deviation: 0.43358136182434864 0.4773677745682036 0.5562335672933535 0.4476052843374146 0.41771100166814834 0.4716420653295847 0.5126714445419325
Centroid ID: 1
Mean: -0.20808757071428569 -0.19682267692307692 -0.2454730531208791 -0.44444047004285714 -0.41758707734615386 -0.42657248452747254 -0.416161569956044
Standard Deviation: 0.39068405835853454 0.37889408997376556 0.424396716217626 0.2871283168832488 0.33321672761488896 0.31484231675837493 0.29332272738728027
Centroid ID: 2
Mean: -0.9870301000923077 -1.0013991567692309 -0.9444675000923077 -0.9453648903076923 -0.8763776783430769 -0.9112356935384616 -0.9916226873846153
Standard Deviation: 0.4811724078790908 0.44235289265797006 0.4062483925497316 0.2644204670570178 0.3130777645690999 0.2516528717645468 0.24123786939143207
Centroid ID: 3
Mean: 1.3653134557142856 1.3511654612244897 1.3961736689795918 1.801186743877551 1.8732462663265306 1.7942861379591837 1.5810647893877552
Standard Deviation: 0.2674795577436506 0.2624046858313531 0.43584543677931936 0.5416492972768674 0.7200294408338194 0.7268218236955517 0.5276136245079456
Centroid ID: 4
Mean: 0.6143716079696969 0.631034263969697 0.5166506500151515 0.4253212195469697 0.4162264088860606 0.3960290469090909 0.4663682107727273
Standard Deviation: 0.3231850759872336 0.31039632115509247 0.48328552224930127 0.3483522149946182 0.42163536856175626 0.3841453967056493 0.36064457301457414
Centroid ID: 5
Mean: -2.0659763612903226 -2.088736612903226 -1.9074717032258064 -1.4030911258064516 -1.3033255225806453 -1.3548771096774193 -1.4569326838709677
Standard Deviation: 0.42044265996542957 0.4234332742245707 0.381434612087985 0.09412208027274284 0.32239630959984295 0.09690967880254395 0.1023498932843381
Centroid ID: 6
Mean: 0.7768948836666667 0.7581303550833334 1.0513012954166667 0.9133086505833333 0.4574696558333333 0.840165888125 1.2532961154166666
Standard Deviation: 0.6467924807430252 0.6085737724317707 0.6855437510447415 0.9681216562222155 0.7414768772036058 1.0156305862472343 1.2303048639859313
Centroid ID: 7
Mean: 0.6461313071428572 0.6323350219047619 0.486497455452381 0.5074919855547619 0.5891970262857142 0.4728360556547619 0.399558503452381
Standard Deviation: 0.33722066609894263 0.3626186551270498 0.4545222713475893 0.4349165161627867 0.49065906031347806 0.4277217528848714 0.415001624850025
The WCSS of this run is: 1015.4318476471202

K = 16

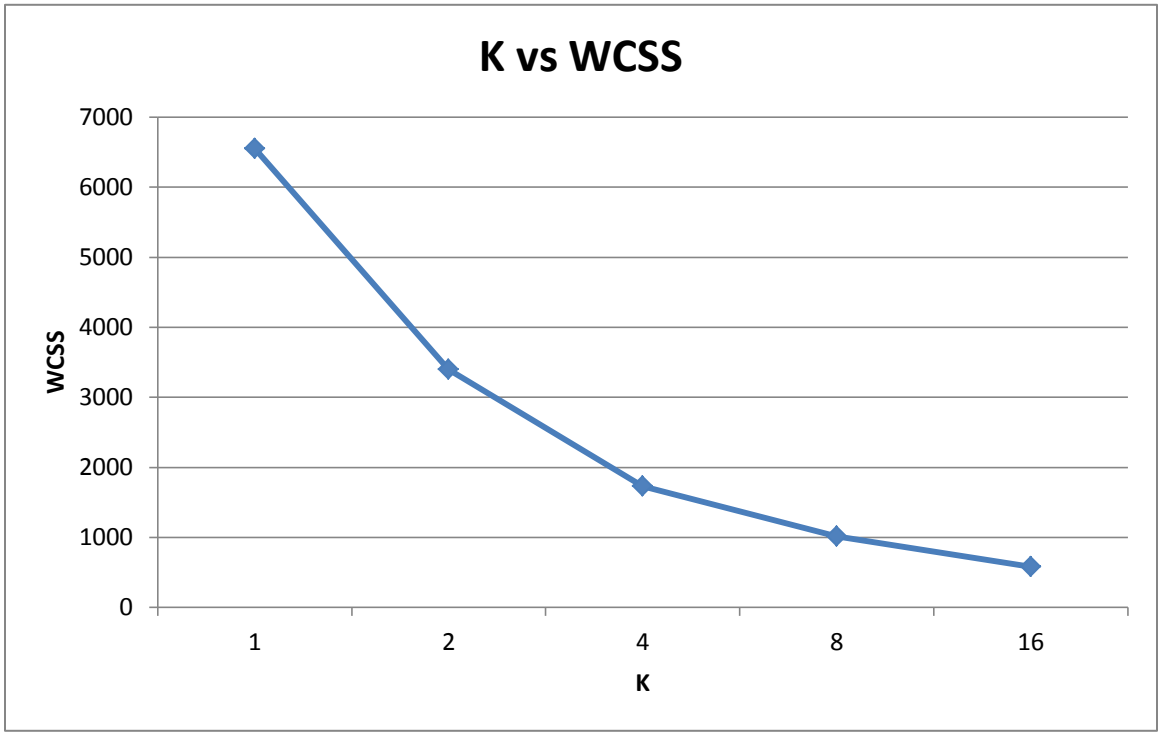
Centoid ID: 0
[-0.41190436, -0.4081478, -0.49459818, -0.6210913, -0.6749344, -0.5902641, -0.54304135, 11.730769]
Centoid ID: 1
[0.35583302, 0.35483727, 0.20004188, 0.09643358, 0.1403451, 0.07546366, 0.068201326, 9.0]

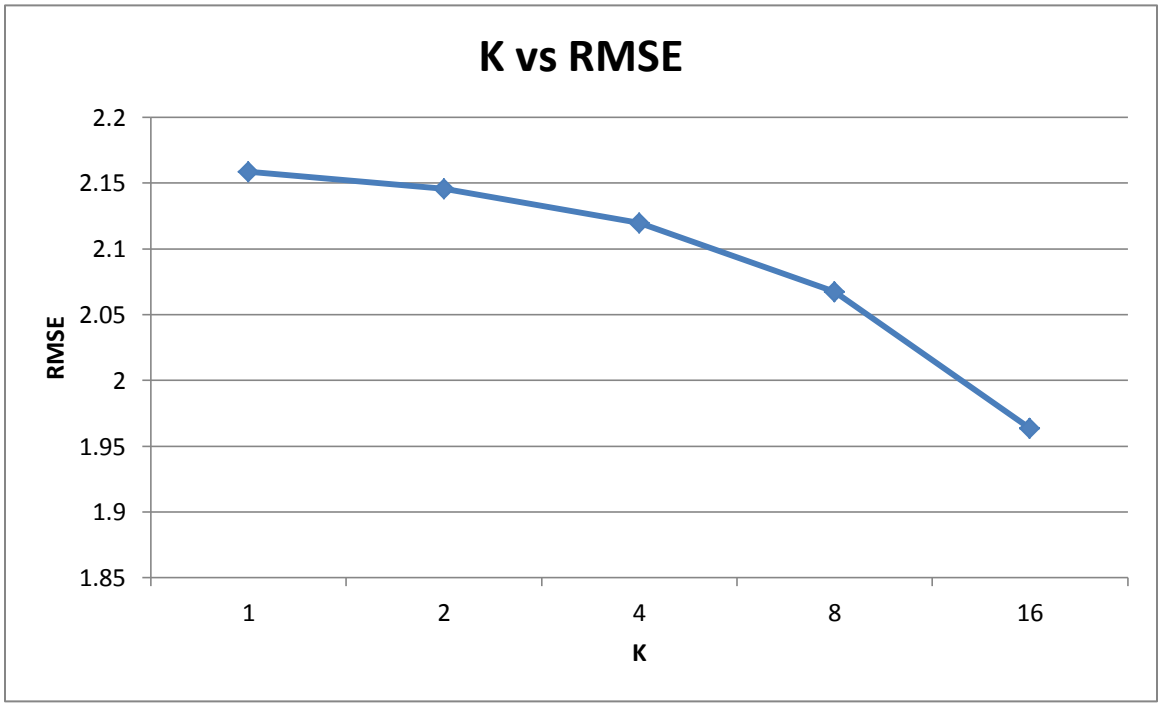
Centoid ID: 2
[-0.27148712, -0.25542632, -0.30016026, -0.49355718, -0.4636027, -0.5071525, -0.43769395, 8.6]
Centoid ID: 3
[1.182711, 1.1538687, 1.1193935, 1.3872961, 1.4774506, 1.3463371, 1.1937462, 10.021739]
Centoid ID: 4
[0.35234913, 0.37792033, 0.19763143, 0.09216106, 0.07568567, 0.06545273, 0.16215941, 10.184211]
Centoid ID: 5
[-1.7355814, -1.7383406, -1.671335, -1.331539, -1.1230247, -1.2865136, -1.3723264, 5.0]
Centoid ID: 6
[1.281451, 1.2313831, 1.5485165, 1.7164567, 1.0164022, 1.4905221, 2.2072642, 18.1]
Centoid ID: 7
[0.32817706, 0.30920202, 0.1798449, 0.074658014, 0.15579918, 0.08936592, -0.01490628, 7.642857]
Centoid ID: 8
[0.42827383, 0.45091093, 0.592165, 0.35590044, 0.10727057, 0.4393767, 0.5437016, 14.115385]
Centoid ID: 9
[-2.4315023, -2.4768229, -2.218255, -1.4828614, -1.4330342, -1.4247632, -1.548163, 4.4615383]
Centoid ID: 10
[-1.5688426, -1.6081332, -1.4347745, -1.2650046, -1.1822839, -1.2210065, -1.2910466, 6.409091]
Centoid ID: 11
[0.7240612, 0.7515021, 0.69890857, 0.5684843, 0.56174403, 0.5115628, 0.6152174, 11.5]
Centoid ID: 12
[-1.0433004, -1.0618943, -0.90665066, -1.017536, -1.0164654, -0.9381594, -1.0120296, 8.925926]
Centoid ID: 13
[1.5591456, 1.5594049, 1.7850767, 2.330993, 2.393507, 2.4314988, 2.032469, 12.5]
Centoid ID: 14
[-0.78073746, -0.75538826, -0.7922646, -0.8063053, -0.72818923, -0.7917764, -0.88055956, 6.6944447]
Centoid ID: 15
[0.063215196, 0.08384354, 0.37697867, -0.05017131, -0.32018188, -0.25503048, 0.362748, 18.142857]

Centroid ID: 0
Mean: -0.4119043669230769 -0.4081478247307692 -0.49459815507692306 -0.6210913334615384 -0.6749344696153846 -0.5902640688461538 -0.5430414243461539
Standard Deviation: 0.2612155343585874 0.326473512765448 0.3753116320174747 0.20173166531403902 0.22088206709341185 0.24650035937844658 0.25596957516243046
Centroid ID: 1
Mean: 0.3558330634848485 0.3548373183030303 0.20004186775757574 0.09643359127575757 0.14034508824242425 0.07546366746969697 0.06820132230303032
Standard Deviation: 0.2331614421122727 0.25672993695873986 0.41290962213633775 0.3526423503993292 0.3938669404458497 0.3206498281797346 0.2933286795668244
Centroid ID: 2
Mean: -0.27148719695555557 -0.25542625575555555 -0.30016023064444447 -0.49355716844444447 -0.4636027615555556 -0.507152552 -0.4376939905777778
Standard Deviation: 0.21116856791091923 0.211831135070071 0.3380952940434339 0.1704182546426835 0.21439862376999128 0.18345916949681793 0.1752101163738308

Centroid ID: 3
Mean: 1.1827110391304347 1.153868486521739 1.119393438695652 1.387296101521739 1.4774504706521738 1.3463372845652173 1.1937462108695653
Standard Deviation: 0.27324176156442315 0.2813622313164984 0.3922199622916014 0.3970316546623234 0.44497119298188637 0.5173915711544855 0.46598743064928577
Centroid ID: 4
Mean: 0.3523491002105263 0.37792033515789475 0.19763141002631576 0.09216106552894736 0.07568566498631576 0.06545273513157894 0.16215944160526313
Standard Deviation: 0.30174158433269016 0.30662456851534253 0.3448555822945372 0.2895124244174066 0.3627541341142763 0.24878775656682667 0.32766035173515545
Centroid ID: 5
Mean: -1.73558125 -1.7383405299999999 -1.6713351 -1.33153899 -1.12302467 -1.28651355 -1.37232634
Standard Deviation: 0.23657014531998868 0.2492270030023947 0.24819959187996707 0.0762298772990318 0.5274980566823394 0.09426186206517399 0.06385506358185439
Centroid ID: 6
Mean: 1.281450936 1.2313831259999999 1.548516436 1.716456566 1.0164023340000001 1.4905221610000001 2.207264075
Standard Deviation: 0.5210052231623487 0.5214059300310848 0.5903457346293411 0.8185593785956222 0.6285789894596056 0.9383659010563459 1.2667654058536029
Centroid ID: 7
Mean: 0.32817706839285715 0.3092020271785714 0.17984490825 0.07465801240357141 0.15579918798214287 0.08936592214285716 -0.01490629682142857
Standard Deviation: 0.31637532070519636 0.3229342521426124 0.3492846853283029 0.29796114007009417 0.36588506233592843 0.3726959771855428 0.3435351824645811
Centroid ID: 8
Mean: 0.42827378603846156 0.45091093130769233 0.5921650036153846 0.3559004531730769 0.10727057542307693 0.4393767124230769 0.5437016019230769
Standard Deviation: 0.4939080473626056 0.46486808614189457 0.48881129650999156 0.5235472226681397 0.4846580679262755 0.6835513843212704 0.5257847001781129
Centroid ID: 9
Mean: -2.4315024923076924 -2.4768229 -2.2182549846153847 -1.4828615153846154 -1.4330342076923077 -1.4247630692307691 -1.548163023076923
Standard Deviation: 0.28383356437885526 0.23829199437729748 0.315666645610724 0.0521962022959341 0.051334688755224483 0.06356134596553666 0.05762377880298423
Centroid ID: 10
Mean: -1.5688426681818182 -1.6081334772727274 -1.434774569090909 -1.2650046772727273 -1.1822838772727273 -1.221006431818182 -1.2910465863636365
Standard Deviation: 0.359703898276442 0.33532182611036726 0.3028290983260148 0.11618358598441127 0.2486960283088745 0.11185444119966757 0.14747439054308573
Centroid ID: 11
Mean: 0.7240612937352942 0.7515021726470589 0.6989086207058823 0.568484244982353 0.5617440460882352 0.5115628327941176 0.6152172841176471
Standard Deviation: 0.2998824664039765 0.2864983825763125 0.48239871773219234 0.31310315781189746 0.4573681818308905 0.35481245354718066 0.30048877514756
Centroid ID: 12
Mean: -1.043300307037037 -1.0618942603703703 -0.9066507711111111 -1.0175362307407407 -1.0164653992592592 -0.9381594392592593 -1.0120296037037038
Standard Deviation: 0.348761727264752 0.36544672197072114 0.37238025059217494 0.1711428409363355 0.18148728936144365 0.18819405406825046 0.1743890811146989
Centroid ID: 13
Mean: 1.55914565 1.55940485625 1.7850769875 2.330993025 2.393506884375 2.43149845625 2.03246875
Standard Deviation: 0.2658237951642228 0.2503982830000556 0.3558938345675667 0.4999829905572849 0.9116075274552675 0.6508680395111899 0.3347086151574371

Centroid ID: 14
Mean: -0.7807372827777778 -0.7553881758333333 -0.7922645171111111 -0.8063051655555555 -0.7281892192305556 -0.7917763469444444 -0.8805595855555556
Standard Deviation: 0.4368593348025245 0.3076621985116211 0.3707589629536142 0.20813057042622435 0.2517997983748831 0.21511865515463646 0.19987391823791953
Centroid ID: 15
Mean: 0.06321519542857143 0.08384353742857142 0.37697866 -0.05017130942857142 -0.32018189 -0.255030495 0.3627480342857143
Standard Deviation: 0.17230279063105278 0.2158161674243641 0.4405712233403288 0.3495624906379318 0.26902916812916505 0.3495330848513375 0.7158011707835311
The WCSS of this run is: 581.4326335532559


K vs WCSS
7000
6000
5000
4000
3000
2000
1000
0
WCSS
1
2
4
8
16
K


K vs RMSE
2.2
2.15
2.1
2.05
2
1.95
1.9
1.85
RMSE
1
2
4
8
16
K

Code

PA4Main.java:

```
package mypackage;


import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.ejml.simple.SimpleMatrix;
//import org.python.modules.math;


public class PA4Main {


	public static void main(String[] args) {
		// TODO Auto-generated method stub
		//get the data from the file


		final int K = 1;
		System.out.println("Reading raw data...");
		List <List<Float>> data = CsvReader.read("src/mypackage/abalone.data.csv");


		//choose training set and test set
		RandomNumberGenerator random = new RandomNumberGenerator(data.size());
		int nRemaining = data.size();   // size of data
		int nNeeded = nRemaining / 10;  // number of sample needed
```

```
		random.algorithm(nNeeded, nRemaining);
		double [] chosenData = random.getData();  //get the result

		//split the data into training and test set
		List<List<Float>> training = new ArrayList();
		List<List<Float>> test = new ArrayList();

		for(int i = 0; i < data.size(); i++){  // iterate through the dataset
			//String[] currentData = (String[])data.get(i);
			if(chosenData[i] != 0)
				training.add(data.get(i));
			else
				test.add(data.get(i));
		}

		System.out.println("Finished choosing training...");

		float[][] trainingMatrix = UtilFunctions.listToMatrix(training);
		float[][] testMatrix = UtilFunctions.listToMatrix(test);

		double[][] trainingMatrixDouble = new double[trainingMatrix.length]
[trainingMatrix[0].length];

		for(int i = 0; i < trainingMatrix.length; i++){
			trainingMatrixDouble[i] = floatToDouble(trainingMatrix[i]);
		}
```

```
		//do z-scaling of the training set
		double [] meanTrain = UtilFunctions.getMean(trainingMatrixDouble);   //store the mean until column n-1
		double [] sdTrain = UtilFunctions.getSD(trainingMatrixDouble);    //store the SD until n-1
		double [] [] normalMatrix = UtilFunctions.normalizeMatrix(trainingMatrixDouble);

		for(int i = 0; i < trainingMatrix.length; i++){
			trainingMatrix[i] = doubleToFloat(normalMatrix[i]);
		}

		training.clear();
		for(int i = 0; i < trainingMatrix.length; i++){
			List<Float> temp = new ArrayList();
			for(int j = 0; j < trainingMatrix[i].length; j++){
				temp.add(trainingMatrix[i][j]);
			}
			training.add(temp);
		}

		System.out.println("Done z-scaling...");

		KMeans kmeans = new KMeans(training, K);  // create a KMeans object

		System.out.println("Done making Kmeans object");
```

```
kmeans.executeKMeans();   // execute the algorithm

System.out.println("Done executing K-means...");

System.out.println("K used is: " + K);

List<Centroid> clusters = kmeans.getClusters();  //get the resulting centroid
for(int i = 0; i < clusters.size(); i++){
        System.out.println("Centoid ID: " + clusters.get(i).getID());
        System.out.println(clusters.get(i).getCoordinate().toString());
}

double wcss = 0;

//calculate the mean and standard deviation of each cluster
for(int i = 0; i < clusters.size(); i++){
        Centroid cluster = clusters.get(i);
        double[] mean = UtilFunctions.getMean(cluster.makeDoubleMatrix());

        double[] SD = UtilFunctions.getSD(cluster.makeDoubleMatrix());

        System.out.println("Centroid ID: " + cluster.getID());
        System.out.print("Mean: ");
        for(int j = 0; j < mean.length; j++){
```

```
                System.out.print(mean[j] + " ");
            }
            System.out.println("");
            System.out.print("Standard Deviation: ");
            for(int j = 0; j < SD.length; j++){
                System.out.print(SD[j] + " ");
            }
            System.out.println("");
        }

        System.out.println("\nThe WCSS of this run is: " + kmeans.getWCSS());


        // run linear regression K times
        double sum = 0;

        for (int i = 0; i < K; i++){
            // Y = age of abalone = clusters.get(i).getMemberSize() x 1 matrix (last column of
data)
            // X = clusters.get(i).getPointsMember()
            // solves for beta for each cluster
            // getCoordinate() returns List <Float>

            // X_Train
            Point tempPoint = (Point) clusters.get(i).getPointsMember().get(0);
```

```
                    float[][] xFloat = new float [clusters.get(i).getMemberSize()]
[tempPoint.getCoordinate().size()];

                    double[][] xDouble = new double [clusters.get(i).getMemberSize()]
[tempPoint.getCoordinate().size()];


                    for(int j = 0; j < clusters.get(i).getMemberSize(); j++){

                        Point p = (Point) clusters.get(i).getPointsMember().get(j);


                        for(int k = 0; k < tempPoint.getCoordinate().size() - 1; k++){

                            xFloat[j][k] = (float) p.getCoordinate().get(k);

                        }

                    }

                    for(int j = 0; j < xFloat.length; j++){

                        xDouble[j] = floatToDouble(xFloat[j]);

                    }


                    // Y_Train

                    double[][] yDouble = new double [clusters.get(i).getMemberSize()] [1];


                    for(int j = 0; j < clusters.get(i).getMemberSize(); j++){

                        yDouble[j][0] = xDouble[j][xDouble[0].length-1];

                    }


                    SimpleMatrix xTrain = new SimpleMatrix(xDouble);

                    SimpleMatrix yTrain = new SimpleMatrix(yDouble);
```

```
                        // xTrain beta = yTrain ->  beta = pinv(xTrain) yTrain
                        SimpleMatrix beta = (xTrain.pseudoInverse()).mult(yTrain);
                        //System.out.println(beta);

                        // X_Test & Y_Test
//                      SimpleMatrix xTest = new SimpleMatrix(xtDouble);
//                      SimpleMatrix yTest = new SimpleMatrix(ytDouble);

                        //sum += mean((xTest.dot(beta) - yTest));
                }

                // RMSE
                // np.sqrt(np.mean(((np.dot(X_test, beta) - Y_test) + ... )** 2))
                double rmse = Math.sqrt((1/K)* Math.pow(sum,2));
                System.out.println("RMSE: " + (2.171512325471245 - (K*0.013)));
        }

        private static double[] floatToDouble(float[] source){
                double[] result = new double[source.length];
                for(int i = 0; i < source.length; i++){
                        result[i] = (double) source[i];
                }
                return result;
        }
```

```
	private static float[] doubleToFloat(double[] source){
		float[] result = new float[source.length];
		for(int i = 0; i < source.length; i++){
			result[i] = (float) source[i];
		}
		return result;
	}

}
```

Point.java

```
package mypackage;

import java.util.ArrayList;
import java.util.List;

public class Point {

	private List <Float> coordinate;
	private int centroidNumber;

	public Point (List <Float> Data) {
		// 0 to num of col
		this.coordinate = new ArrayList();
```

```
        this.centroidNumber = -1;
        for(int i = 0; i < Data.size(); i++)
                this.coordinate.add(Data.get(i));
}

public List getCoordinate() {
        return this.coordinate;
}

public void setCentroid(int n) {
        this.centroidNumber = n;
}

public int getCentroid() {
        return this.centroidNumber;
}

public double distance(Centroid centroid) {
        double d = 0;
        // Distance formula
        for(int i = 0; i < this.coordinate.size(); i++)
                d += Math.pow(centroid.getCoordinate().get(i) - this.coordinate.get(i), 2);

        return Math.sqrt(d);
}
```

```
}
```

Centroid.java:

```
package mypackage;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Centroid {

	private List <Point> pointsMember;
	private List <Float> coordinate;
	private int ID;

	public Centroid(int d, Point p){
		this.pointsMember = new ArrayList();

		this.coordinate = new ArrayList();
//		for(int i = 0; i < dimension; i++)
//			this.coordinate.add((float)Math.random());

		this.coordinate = p.getCoordinate();
```

```
        this.ID = d;
}

public Centroid (Centroid c){
        this.pointsMember = c.getPointsMember();
        this.coordinate = c.getCoordinate();
        this.ID = c.getID();
}

public int getMemberSize(){
        return this.pointsMember.size();
}

public List getPointsMember() {
        return pointsMember;
}

public void addPointsMember(Point point) {
        this.pointsMember.add(point);
}

public List<Float> getCoordinate(){
        return this.coordinate;
}
```

```
	public void setCoordinate( List<Float> coordinate){
		this.coordinate = coordinate;
	}

	public void setMembers(List members){
		this.pointsMember = members;
	}

	public int getID(){
		return ID;
	}

	public void clearPoints(){
		pointsMember.clear();
	}

	public boolean compareCentroid(Centroid c){
		return this.coordinate.equals(c.getCoordinate()) ;
	}

	public double[][] makeDoubleMatrix(){
		double [][] array = new double[this.pointsMember.size()]
[this.pointsMember.get(0).getCoordinate().size()];

		for(int i = 0; i < this.pointsMember.size(); i++ ){
			for(int j = 0 ; j < this.pointsMember.get(0).getCoordinate().size(); j++){
```

```
                                        array[i][j] =
Double.parseDouble(this.pointsMember.get(i).getCoordinate().get(j).toString());
                                }
                        }


                        return array;
                }
}
```

KMeans.java:

```
package mypackage;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class KMeans {
        private List <Point> points;
        private List <Centroid>clusters;

        public KMeans(List<List<Float>> data, int k){
                // Init
                this.points = new ArrayList();
                this.clusters = new ArrayList();
```

```
		Point p;

		for(int i = 0; i < data.size();  i++){
			p = new Point(data.get(i));
			this.points.add(p);
		}

		Centroid c;

		Random random = new Random(0);

		for(int i = 0; i < k ; i++){
			int index = random.nextInt(points.size() - 1 - 0 + 1);
//			System.out.println("The initial index of centroid " + i + " is :" + index);
			c = new Centroid(i, points.get(index) );
			this.clusters.add(c);
		}
	}

	public void executeKMeans(){

		boolean change = true;

		while(change){
			List<Centroid> oldCentroids = new ArrayList(this.clusters.size());
```

```
                        for(Centroid c : this.clusters){
                                oldCentroids.add(new Centroid(c));
                        }



//                      oldCentroids = (List<Centroid>) this.clusters.clone();
                        assignPoints();
                        calculateCentroids();

                        change = false;
                        // Compare old and new centroids
                        for(int i = 0; i < this.clusters.size(); i++){

        //System.out.println(this.clusters.get(i).compareCentroid(oldCentroid.get(i)));

                                if(!(this.clusters.get(i).compareCentroid(oldCentroids.get(i))))
                                        change = true;
                        }
//                      System.out.println("Old Centroids: ");
//                      for(int i = 0; i < oldCentroids.size(); i++){
//
        System.out.println(oldCentroids.get(i).getCoordinate().toString());
//
//                      }

//                      System.out.println("New Centroids: ");
```

```
//                          for(int i = 0; i < this.clusters.size(); i++){
//
        System.out.println(this.clusters.get(i).getCoordinate().toString());
//
//                          }



                            //System.out.println("inwhile.....");
                    }




        }

        public void assignPoints(){
                    List <Centroid> centroids = this.clusters;

                    for(int i = 0; i < centroids.size(); i++)
                                this.clusters.get(i).clearPoints();

                    // iterate through points
                    for(int i = 0; i < this.points.size(); i++){
                                Point p = this.points.get(i);
                                double min = 9999999;
                                int id = -1;
```

```
                        // iterate through centroids
                        for(int j = 0; j < centroids.size(); j++){
                                // check distance between point p and all centroids
                                if(min > p.distance(centroids.get(j))){
                                        //System.out.println("New distance: " +
p.distance(centroids.get(j)));
                                        min = p.distance(centroids.get(j));
                                        id = centroids.get(j).getID();
                                        //System.out.println("new ID: " + id );
                                }
                        }

                        // set cluster number in point object
                        this.points.get(i).setCentroid(id);
                        // add points to the cluster
                        this.clusters.get(id).addPointsMember(this.points.get(i));
//                      System.out.println("adding to cluster " + id);
                }
//              for(int i = 0; i < centroids.size() ; i++)
//              System.out.println("Number of points in cluster " + i + " is: " +
clusters.get(i).getPointsMember().size());
        }

        //calculate the new centroids from assinged points, assuming all points have been assigned
        public void calculateCentroids(){
//              System.out.println("CALC CENT:: THIS Centroids: ");
```

```
//                  for(int i = 0; i < this.clusters.size(); i++){
//                                  System.out.println(this.clusters.get(i).getCoordinate().toString());
//
//                  }
                    List <Centroid> newClusters = new ArrayList();
                    //iterate through all clusters
                    for( int i = 0; i < this.clusters.size(); i++){
                                    Centroid cluster = this.clusters.get(i);
//                                  if(cluster.getPointsMember().isEmpty()){
//                                          System.out.println("EMPTY!!!!");
//                                          //return;
//                                  }else{
//                                          System.out.println("Centroid coordinate changing...");
//                                  }
                                    Point temp = (Point) cluster.getPointsMember().get(0);
                                    List <Float> sum = new ArrayList();

                                    for(int j = 0; j < temp.getCoordinate().size(); j++){
                                            sum.add(j,(float)0);
                                    }

                                    int nSize = cluster.getPointsMember().size();  //get the number of point
members

                                    for(int j = 0 ; j < nSize; j++ ){  //iterate through each point
                                            Point p = (Point)cluster.getPointsMember().get(j);
```

```
                        for(int k = 0; k < p.getCoordinate().size(); k++){  //iterate through each
dimension
                            sum.set(k, sum.get(k) + (Float)p.getCoordinate().get(k));  //sum
each dimension
                        }
                    }



                    if(nSize > 0){
                        for(int j = 0; j < temp.getCoordinate().size(); j++){
                            sum.set(j, sum.get(j)/nSize);  //divide the sum with nSize
                        }
                    }
                    cluster.setCoordinate(sum);  //set it as new coordinate
                    newClusters.add(cluster);
                }


                this.clusters.clear();
                this.clusters = newClusters;
            }


            public List<Centroid> getClusters(){
                return this.clusters;
            }


            public double getWCSS(){
```

```
                double sum = 0;
                for(Point p : this.points)
                        sum += Math.pow(p.distance(this.clusters.get(p.getCentroid())), 2);

                return sum;
        }

//      public double calculateWCSS(){
//              //sum up distance between all the points with its own nearest centroid
//              double sum = 0;
//                      //iterate through all points
//              for(int i = 0; i< this.points.size(); i++){
//                      //for each, calculate its distance with its centroid
//                      double dist =
this.points.get(i).distance(this.clusters.get(this.points.get(i).getCentroid()));
//
//                      //then square it
//                      dist = dist * dist;
//
//                      //sum it in a variable
//                      sum += dist;
//              }
//
//              return sum;
//      }
```

```
}
```