Class07_Unsupervised_Machine_Learning1

Krysten Jones (A10553682)

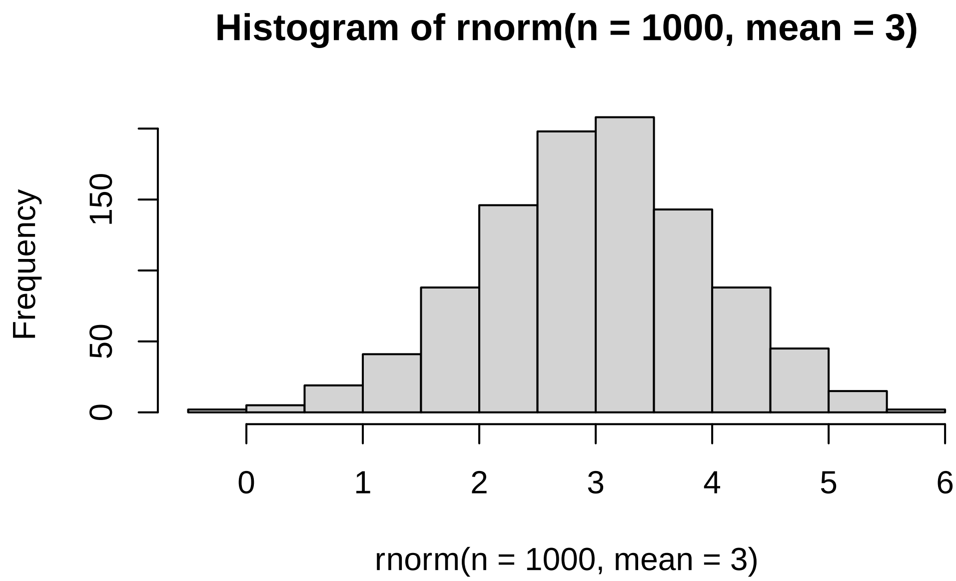
Today we will start with k-means clustering which is one of the most popular means of clustering along with UMAP and t-SNE. K-means is fast and computes many things for you. The challenge with this is you have to define the number of clusters (represented by K) for your data.

Lets try this out on some madeup data using `rnorm(n=x, mean = y)`. This function will randomly give back a set of numbers (defined by x here) from a normal distribution with the central mean being y. You don't can also code it as `rnorm(x,y)` and r will assume based on the order of arguments, but for clarity it is often better to write the first form

```
rnorm(10,3)
```

```
 [1] 2.5949272 3.4522711 0.9973678 3.6140099 5.1324777 2.9688123 2.0614066
 [8] 2.2447915 4.6528272 2.0402560
```

```
hist(rnorm(n=1000, mean =3))
```



We can also combine multiple vectors in the rnorm function. The code below should give you 60 datapoints

```
tmp <- c(rnorm(30,3), rnorm(30,-3))
tmp
```

```
 [1]  4.3905648  3.3818380  1.7223547  3.9888557  3.8477142  2.8476103
 [7]  2.4605586  4.0781301  1.7230475  2.2898489  2.1572686  2.7033445
[13]  3.5308666  2.9369977  2.3362544  4.3902521  3.1800745  4.1712229
[19]  3.4656873  3.7696900  3.0664364  2.5721636  5.0527608  3.3081824
[25]  3.2413157  3.2004187  2.8081589  2.8311566  1.5366185  1.8335831
[31] -2.0713806 -3.3315724 -2.5550030 -3.8470936 -3.4385719 -4.0345927
[37] -3.2053785 -1.4924427 -1.4599830 -2.0640190 -2.5405664 -3.6169189
[43] -2.9091769 -0.6604499 -0.8082855 -1.9122156 -4.9011213 -2.2173231
[49] -3.5417310 -2.8027591 -2.7275283 -4.6969762 -1.7544863 -3.8996736
[55] -1.5261682 -2.9838627 -3.7861962 -3.2759987 -2.9154850 -1.3947266
```

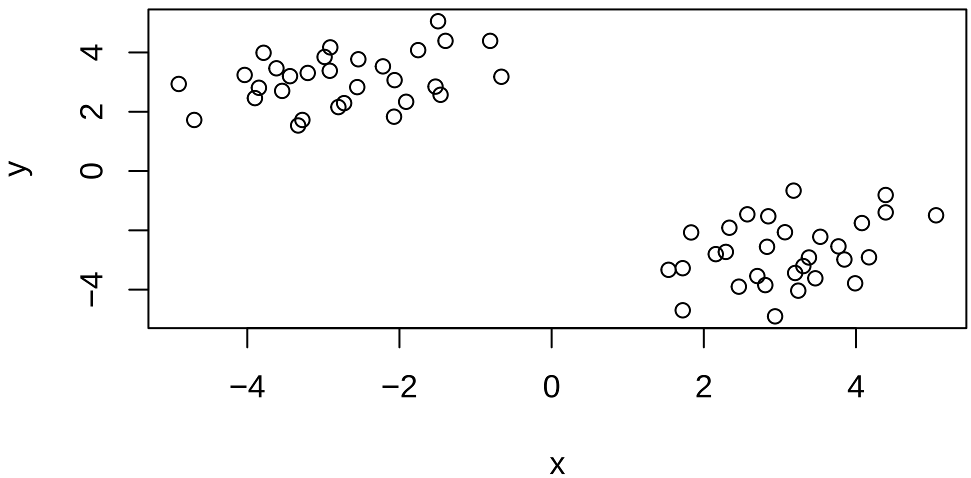
We can also make matrices using the `cbind()` function which will put the arguments in the ( ) as columns as compared to `rbind()` which will add them as rows. The "x" and "y" labels here are arbitrary and whatever you write here will be added as labels at the top of the columns or rows respectively. The `rev` here is asking for the reverse of the vector in the ( ).

```r
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
              x          y
 [1,]  4.3905648 -1.3947266
 [2,]  3.3818380 -2.9154850
 [3,]  1.7223547 -3.2759987
 [4,]  3.9888557 -3.7861962
 [5,]  3.8477142 -2.9838627
 [6,]  2.8476103 -1.5261682
 [7,]  2.4605586 -3.8996736
 [8,]  4.0781301 -1.7544863
 [9,]  1.7230475 -4.6969762
[10,]  2.2898489 -2.7275283
[11,]  2.1572686 -2.8027591
[12,]  2.7033445 -3.5417310
[13,]  3.5308666 -2.2173231
[14,]  2.9369977 -4.9011213
[15,]  2.3362544 -1.9122156
[16,]  4.3902521 -0.8082855
[17,]  3.1800745 -0.6604499
[18,]  4.1712229 -2.9091769
[19,]  3.4656873 -3.6169189
[20,]  3.7696900 -2.5405664
[21,]  3.0664364 -2.0640190
[22,]  2.5721636 -1.4599830
[23,]  5.0527608 -1.4924427
[24,]  3.3081824 -3.2053785
[25,]  3.2413157 -4.0345927
[26,]  3.2004187 -3.4385719
[27,]  2.8081589 -3.8470936
[28,]  2.8311566 -2.5550030
[29,]  1.5366185 -3.3315724
[30,]  1.8335831 -2.0713806
[31,] -2.0713806  1.8335831
[32,] -3.3315724  1.5366185
[33,] -2.5550030  2.8311566
[34,] -3.8470936  2.8081589
[35,] -3.4385719  3.2004187
[36,] -4.0345927  3.2413157
[37,] -3.2053785  3.3081824
[38,] -1.4924427  5.0527608
```

```
[39,] -1.4599830  2.5721636
[40,] -2.0640190  3.0664364
[41,] -2.5405664  3.7696900
[42,] -3.6169189  3.4656873
[43,] -2.9091769  4.1712229
[44,] -0.6604499  3.1800745
[45,] -0.8082855  4.3902521
[46,] -1.9122156  2.3362544
[47,] -4.9011213  2.9369977
[48,] -2.2173231  3.5308666
[49,] -3.5417310  2.7033445
[50,] -2.8027591  2.1572686
[51,] -2.7275283  2.2898489
[52,] -4.6969762  1.7230475
[53,] -1.7544863  4.0781301
[54,] -3.8996736  2.4605586
[55,] -1.5261682  2.8476103
[56,] -2.9838627  3.8477142
[57,] -3.7861962  3.9888557
[58,] -3.2759987  1.7223547
[59,] -2.9154850  3.3818380
[60,] -1.3947266  4.3905648
```

```
plot(x)
```



The main function in R for k-means clustering is called `kmeans()`. It requires 3 arguments the first being what dataset to use (here represented by x), the number of clusters assigned by `centers=`, and the number of iterations to run it which is defined by the `nstart=`

```
k <- kmeans(x, centers=2, nstart =20)
k
```

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1  3.094099 -2.745723
2 -2.745723  3.094099

Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 56.74546 56.74546
 (between_SS / total_SS =  90.0 %)
```

```
Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

Looking at the readouts, it will give you a variety of pieces of information

The center location for the mean of the values in each cluster (you can also do complete, single, or average by changing the arguments).

Then the clustering vector will tell you which group each value is in (here we told it to make 2 clusters so it will be either 1 or 2).

It will then give you the within cluster sum of squares. This is the Euclidian distance between the center of a cluster and a point in the cluster, squared and then repeated and summed for all points in the cluster. The more clusters you have then, the smaller these numbers will be.

You can also ask it to just give you specific portions as indicated in the questions below. Don't forget you can always check the options using the `?kmeans` command.

> Q1. How many points are in each cluster

```
k$size
```

```
[1] 30 30
```

> Q2. What is the clustering result (i.e. membership vector)?

```
k$cluster
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

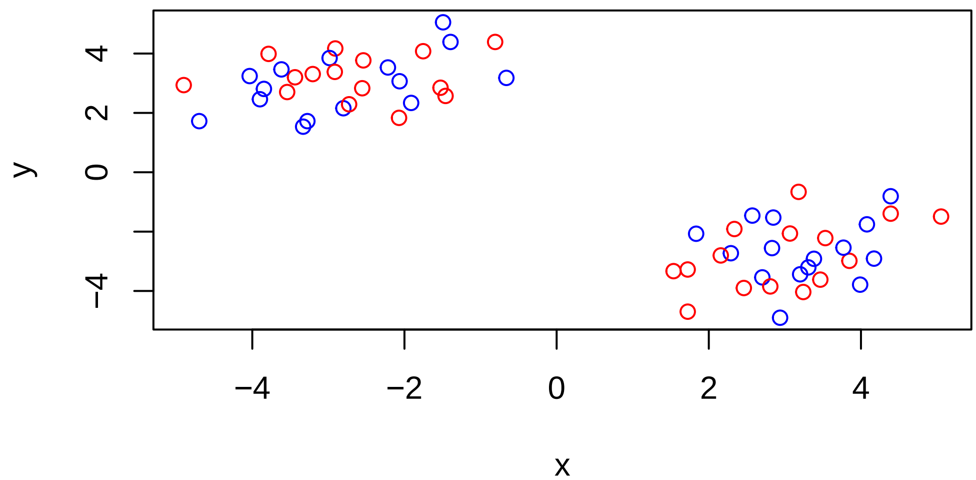
> Q3.What are the cluster centers?

```
k$centers
```

```
          x         y
1  3.094099 -2.745723
2 -2.745723  3.094099
```

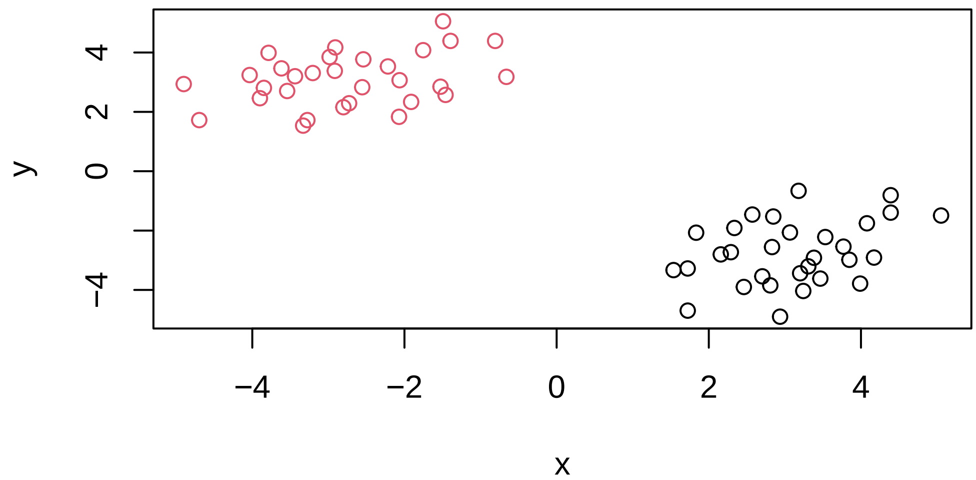
> Q4. Make a plot of our data colored by clustering results with optionally the cluster centers shown.

```
plot(x, col=c("red", "blue"))
```



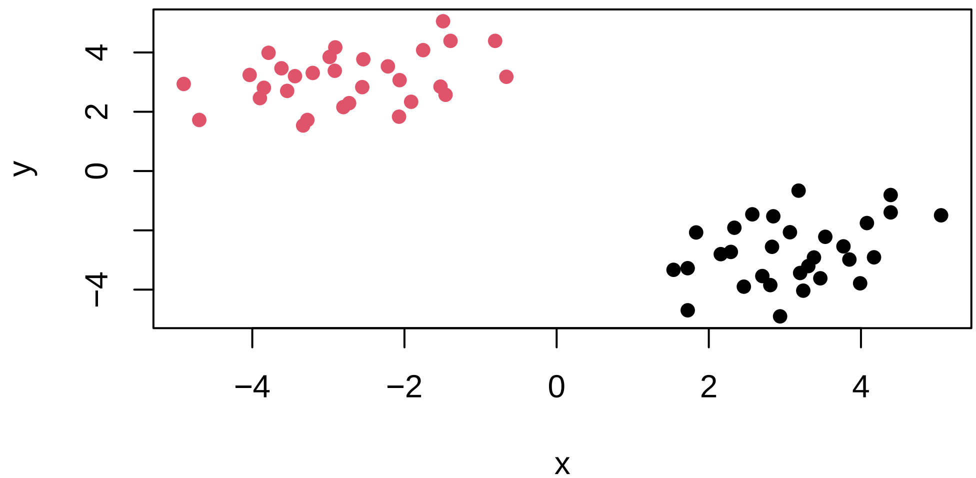
This will color by data points in the vector order, but we want to color by cluster. How do we do this?

```
plot(x, col=k$cluster)
```



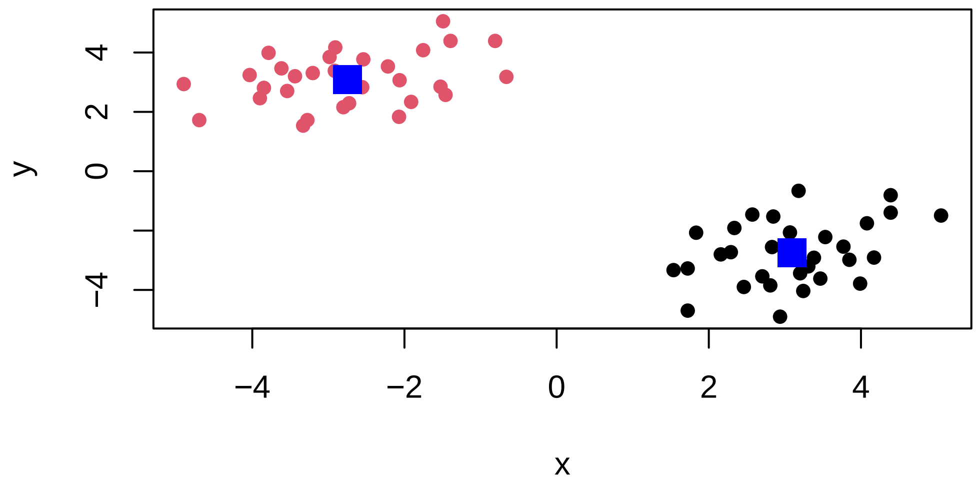
Now we have 2 clusters. What if we want solid circles? We would use the point character or `pch` argument

```
plot(x, col=k$cluster, pch=16)
```



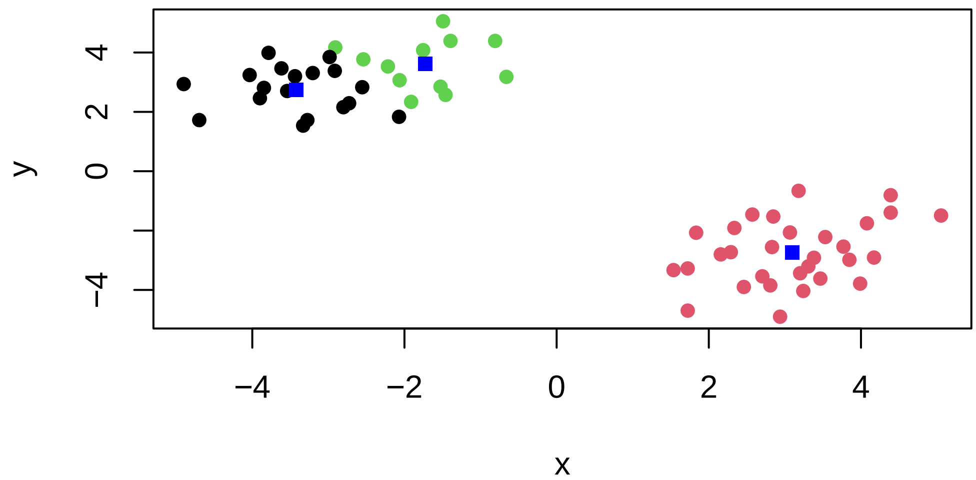
Now we want to include the centers on our graph. Here we'll use the `points` function which will add points to our graph in the same format as the `plot` function, but with an additional argument `cex=` this will determine the size of the point shape that you defined in `pch=`, just like how we had to define it in the kmeans function.

```
plot(x, col=k$cluster, pch=16)
points(k$centers, col="blue", pch=15, cex=2)
```



Q5. Run kmeans again, but cluster into 3 groups and plot the results

```
k3 <- kmeans(x, centers=3, nstart =20)
plot(x, col=k3$cluster, pch=16)
points(k3$centers, col="blue", pch=15, cex=1)
```



The challenge here is that even if there aren't really the number of clusters you assigned in k, it will make that number of clusters anyway.

> How do you know how many iterations to call?

Until you stop getting different answers or get impatient. You won't know ahead of time how many iterations will be sufficient for your dataset.

Scree Plots

Scree plots are used to determine your desired number of clusters. If this is a straight line, this means that there are no clear groupings. These measure the total sum of squares on the y-axis and the number of clusters on the x-axis. At a certain point the sum of squares doesn't drastically decrease with increasing number of clusters. This point is often called the "elbow" point and is usually the number of clusters you want to define the kmeans.

Hierarchical Clustering

Has an advantage in that it can help visualize structure in your data rather than imposing a structure as you do with `kmeans`.

The main function in "R base" is called `hclust()` for hierarchical clustering. As always, its helpful to check the help file `?hclust`. This shows you the arguments that you can use in the clustering.

```
?hclust
```

```
starting httpd help server ... done
```

The first argument required is `d` which is a measure of dissimilarity that you must calculate, but can be based on a variety of things. Two optional arguments include the `method=` which can be "complete" which is the maximum, "single" which is the minimum, or "average".

```
#by default, if you make a distance matrix it will be based on Euclidian distance, but can
dist(x)
```

```
           1         2         3         4         5         6         7
2  1.8248934
3  3.2647404 1.6981917
4  2.4249736 1.0614181 2.3232150
5  1.6792976 0.4708675 2.1453430 0.8146532
6  1.5485430 1.4884893 2.0804103 2.5318309 1.7677900
7  3.1622276 1.3481035 0.9663929 1.5325042 1.6622005 2.4048570
8  0.4764897 1.3537875 2.8044033 2.0336703 1.2507829 1.2515224 2.6867018
9  4.2450561 2.4341933 1.4209777 2.4420088 2.7292795 3.3643224 1.0861004
10 2.4878439 1.1080469 0.7892208 2.0018495 1.5788134 1.3245240 1.1845110
11 2.6401074 1.2297469 0.6427331 2.0789083 1.7001191 1.4512946 1.1380713
12 2.7306300 0.9233296 1.0163438 1.3085496 1.2731060 2.0207192 0.4325135
13 1.1898513 0.7138904 2.0955928 1.6343551 0.8294428 0.9718716 1.9939565
14 3.7957425 2.0348549 2.0288866 1.5327959 2.1225657 3.3761367 1.1090048
15 2.1184868 1.4490668 1.4955859 2.4985784 1.8528191 0.6407164 1.9913415
16 0.5864412 2.3360626 3.6341829 3.0048413 2.2422051 1.7014991 3.6442280
17 1.4157856 2.2640433 2.9943352 3.2286866 2.4174346 0.9273622 3.3181733
18 1.5302518 0.7894101 2.4761893 0.8957793 0.3320178 1.9143311 1.9767287
19 2.4069768 0.7064278 1.7763544 0.5498726 0.7393948 2.1801968 1.0441427
20 1.3032399 0.5394378 2.1754178 1.2647637 0.4501105 1.3708518 1.8870606
21 1.4836672 0.9080047 1.8098206 1.9536508 1.2068586 0.5806620 1.9330587
22 1.8195717 1.6655505 2.0050157 2.7236527 1.9872692 0.2832867 2.4422420
23 0.6693668 2.1947738 3.7779196 2.5284776 1.9174125 2.2054084 3.5375518
24 2.1095051 0.2991043 1.5873994 0.8947990 0.5832356 1.7412277 1.0956788
25 2.8791782 1.1278957 1.6978537 0.7877290 1.2131582 2.5391333 0.7923287
26 2.3651113 0.5536542 1.4869779 0.8616702 0.7910449 1.9446753 0.8717842
```

27	2.9185806	1.0940761	1.2268334	1.1822662	1.3512375	2.3212608	0.3515546
28	1.9437066	0.6581773	1.3226022	1.6900011	1.1033178	1.0289664	1.3948054
29	3.4491131	1.8915507	0.1938721	2.4940229	2.3371063	2.2311844	1.0846217
30	2.6449983	1.7634074	1.2097424	2.7542317	2.2111870	1.1513070	1.9328098
31	7.2234840	7.2312682	6.3639810	8.2648885	7.6317408	5.9568783	7.3081259
32	8.2597934	8.0555015	6.9787867	9.0510233	8.4839205	6.8965905	7.9436801
33	8.1301291	8.2625644	7.4560804	9.3065269	8.6492333	6.9407859	8.3940415
34	9.2478788	9.2204963	8.2483774	10.2414657	9.6310738	7.9753026	9.2076867
35	9.0780362	9.1609098	8.2812527	10.1970323	9.5569505	7.8649037	9.2309832
36	9.6164529	9.6389646	8.6958514	10.6659105	10.0440833	8.3721933	9.6530160
37	8.9339637	9.0623097	8.2239891	10.1038217	9.4517696	7.7465875	9.1682076
38	8.7281080	9.3408539	8.9276625	10.4005669	9.6490721	7.8815205	9.7863323
39	7.0686015	7.3183002	6.6579483	8.3736839	7.6838193	5.9457281	7.5667247
40	7.8462492	8.0895452	7.3866845	9.1430774	8.4590016	6.7242931	8.3065330
41	8.6435975	8.9312059	8.2349393	9.9862289	9.2962682	7.5550355	9.1558937
42	9.3671457	9.4711118	8.5998938	10.5089305	9.8649604	8.1675430	9.5490457
43	9.1796526	9.4761963	8.7699598	10.5310670	9.8412818	8.0994360	9.6939893
44	6.8148040	7.3140917	6.8817614	8.3752593	7.6366004	5.8698557	7.7371524
45	7.7777903	8.4220502	8.0731370	9.4798138	8.7210035	6.9548258	8.9111284
46	7.3242924	7.4570618	6.6863657	8.5033548	7.8409462	6.1297839	7.6162952
47	10.2517933	10.1419411	9.0813963	11.1459870	10.5640291	8.9421860	10.0466116
48	8.2417019	8.5385042	7.8647617	9.5945850	8.9009200	7.1573145	8.7804046
49	8.9283539	8.9166728	7.9663759	9.9410198	9.3246033	7.6624056	8.9234146
50	8.0225045	7.9988793	7.0708591	9.0250099	8.4059519	6.7449522	8.0242795
51	8.0151947	8.0261982	7.1260171	9.0569387	8.4288700	6.7560459	8.0762884
52	9.6074927	9.3157514	8.1362321	10.2856911	9.7553441	8.2145108	9.1019392
53	8.2288403	8.6771239	8.1345949	9.7382547	9.0142328	7.2517206	9.0228574
54	9.1428265	9.0510914	8.0321412	10.0623476	9.4690940	7.8370804	8.9947266
55	7.2804636	7.5697947	6.9319181	8.6268694	7.9299867	6.1854568	7.8370804
56	9.0479482	9.2877882	8.5379019	10.3390227	9.6613088	7.9299867	9.4690940
57	9.7899120	9.9524185	9.1171400	10.9955837	10.3390227	8.6268694	10.0623476
58	8.2760130	8.1139600	7.0687392	9.1171400	8.5379019	6.9319181	8.0321412
59	8.7289136	8.9057596	8.1139600	9.9524185	9.2877882	7.5697947	9.0510914
60	8.1816375	8.7289136	8.2760130	9.7899120	9.0479482	7.2804636	9.1428265

	8	9	10	11	12	13	14
2							
3							
4							
5							
6							
7							
8							
9	3.7689072						

10	2.0358685	2.0493875					
11	2.1882836	1.9433493	0.1524376				
12	2.2548346	1.5151151	0.9131838	0.9188462			
13	0.7167394	3.0686951	1.3418026	1.4931533	1.5616815		
14	3.3471623	1.2309956	2.2678862	2.2385490	1.3793244	2.7487184	
15	1.7490025	2.8514758	0.8166323	0.9083521	1.6703519	1.2329594	3.0486800
16	0.9963514	4.7154953	2.8452041	2.9940174	3.2120681	1.6504334	4.3431847
17	1.4154220	4.2914418	2.2506254	2.3739463	2.9204542	1.5959039	4.2476324
18	1.1584372	3.0314665	1.8901228	2.0167639	1.5983715	0.9427183	2.3433212
19	1.9605463	2.0501993	1.4743173	1.5410437	0.7660416	1.4011127	1.3887723
20	0.8444272	2.9730202	1.4916046	1.6335996	1.4626767	0.4018990	2.5031173
21	1.0579862	2.9558683	1.0214366	1.1714619	1.5216663	0.4890783	2.8400535
22	1.5344926	3.3465091	1.2986041	1.4054129	2.0858770	1.2217510	3.4604243
23	1.0092431	4.6212580	3.0264035	3.1781762	3.1175855	1.6857086	4.0119251
24	1.6425309	2.1765837	1.1248751	1.2193051	0.6920707	1.0128384	1.7358922
25	2.4288154	1.6564692	1.6166961	1.6409059	0.7296065	1.8401926	0.9184123
26	1.8990845	1.9406718	1.1553009	1.2216464	0.5076657	1.2651658	1.4860825
27	2.4478220	1.3783204	1.2337227	1.2305661	0.3228504	1.7828231	1.0618728
28	1.4818131	2.4116292	0.5681364	0.7179889	0.9949714	0.7769310	2.3485046
29	2.9910671	1.3780724	0.9655182	0.8153834	1.1855026	2.2844205	2.1034604
30	2.2668069	2.6279214	0.7991923	0.7998043	1.7083370	1.7035465	3.0372613
31	7.1197418	7.5528729	6.3106307	6.2751209	7.1897150	6.9133938	8.3928598
32	8.1077162	8.0253900	7.0557300	6.9969689	7.8873225	7.8220935	8.9855141
33	8.0639057	8.6587816	7.3737079	7.3448287	8.2621979	7.9072723	9.4842075
34	9.1447746	9.3463108	8.2647382	8.2179539	9.1230117	8.9269125	10.2692208
35	9.0028825	9.4345727	8.2435042	8.2068005	9.1202917	8.8275137	10.3093568
36	9.5275553	9.8064723	8.6963016	8.6527445	9.5608326	9.3291432	10.7192515
37	8.8701811	9.4006355	8.1625565	8.1302884	9.0462325	8.7125317	10.2528753
38	8.7960158	10.2662919	8.6509322	8.6619619	9.5639907	8.8367276	10.8949396
39	7.0278444	7.9354947	6.4921471	6.4787578	7.3968239	6.9172077	8.6708377
40	7.8081554	8.6378497	7.2474956	7.2295729	8.1483514	7.6955090	9.4070262
41	8.6211175	9.4796013	8.0960952	8.0787833	8.9975262	8.5268179	10.2560556
42	9.2986016	9.7541949	8.5583775	8.5225971	9.4366117	9.1316726	10.6281376
43	9.1616859	10.0051216	8.6384396	8.6200519	9.5388707	9.0712557	10.7928303
44	6.8413471	8.2297623	6.6033351	6.6131562	7.5165007	6.8336692	8.8457535
45	7.8507876	9.4332055	7.7628110	7.7803548	8.6745548	7.9049535	10.0178224
46	7.2538542	7.9171630	6.5802159	6.5551629	7.4735606	7.0966338	8.7117436
47	10.1309910	10.1072830	9.1540651	9.0975643	9.9900864	9.8825828	11.0847742
48	8.2199566	9.1227144	7.7124644	7.6975234	8.6159507	8.1291678	9.8825828
49	8.8280541	9.0819954	7.9687957	7.9243784	8.8318705	8.6159507	9.9900864
50	7.9150782	8.2136227	7.0566208	7.0145384	7.9243784	7.6975234	9.0975643
51	7.9166681	8.2839212	7.0956430	7.0566208	7.9687957	7.7124644	9.1540651
52	9.4390536	9.0792846	8.2839212	8.2136227	9.0819954	9.1227144	10.1072830

53	8.2485652	9.4390536	7.9166681	7.9150782	8.8280541	8.2199566	10.1309910
54	9.0228574	9.1019392	8.0762884	8.0242795	8.9234146	8.7804046	10.0466116
55	7.2517206	8.2145108	6.7560459	6.7449522	7.6624056	7.1573145	8.9421860
56	9.0142328	9.7553441	8.4288700	8.4059519	9.3246033	8.9009200	10.5640291
57	9.7382547	10.2856911	9.0569387	9.0250099	9.9410198	9.5945850	11.1459870
58	8.1345949	8.1362321	7.1260171	7.0708591	7.9663759	7.8647617	9.0813963
59	8.6771239	9.3157514	8.0261982	7.9988793	8.9166728	8.5385042	10.1419411
60	8.2288403	9.6074927	8.0151947	8.0225045	8.9283539	8.2417019	10.2517933

	15	16	17	18	19	20	21
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16	2.3318594						
17	1.5096192	1.2191739					
18	2.0883106	2.1122781	2.4574679				
19	2.0449039	2.9568973	2.9702330	0.9993394			
20	1.5651078	1.8400800	1.9704021	0.5450710	1.1184599		
21	0.7457949	1.8246519	1.4081619	1.3909871	1.6034025	0.8495075	
22	0.5100661	1.9313611	1.0043949	2.1580439	2.3346856	1.6129879	0.7804902
23	2.7487481	0.9523596	2.0491866	1.6686057	2.6518298	1.6567540	2.0669262
24	1.6176880	2.6300056	2.5481510	0.9124550	0.4406510	0.8092987	1.1666802
25	2.3072972	3.4247793	3.3746986	1.4598932	0.4741245	1.5847064	1.9783184
26	1.7540078	2.8868859	2.7781965	1.1057667	0.3196484	1.0632421	1.3810674
27	1.9915940	3.4259851	3.2082737	1.6545789	0.6966520	1.6222070	1.8016832
28	0.8112359	2.3413246	1.9264151	1.3860797	1.2370508	0.9386444	0.5444464
29	1.6291075	3.8092259	3.1362148	2.6682501	1.9500588	2.3690291	1.9867145
30	0.5272683	2.8516602	1.9503243	2.4832363	2.2477662	1.9921459	1.2328753
31	5.7843110	6.9808428	5.8136031	7.8398897	7.7696263	7.2973481	6.4489119
32	6.6346603	8.0700154	6.8723107	8.7210684	8.5300461	8.1884897	7.3416011
33	6.8135144	7.8410526	6.7143450	8.8427113	8.8219266	8.2980207	7.4540810
34	7.7791856	8.9962511	7.8366025	9.8479099	9.7343921	9.3072152	8.4578375
35	7.7128236	8.7954644	7.6624269	9.7589003	9.7028288	9.2150935	8.3683592

36	8.1943015	9.3475815	8.2021459	10.2549485	10.1631482	9.7127230	8.8640391
37	7.6132943	8.6393814	7.5182480	9.6472693	9.6156200	9.1027162	8.2580996
38	7.9479442	8.3040930	7.3805957	9.7708525	9.9873119	9.2384337	8.4517413
39	5.8754638	6.7566772	5.6550795	7.8584715	7.9099286	7.3136508	6.4794028
40	6.6445001	7.5280199	6.4335214	8.6363300	8.6743812	8.0913929	7.2555595
41	7.4878189	8.3062689	7.2354593	9.4686524	9.5203505	8.9240501	8.0913929
42	8.0226001	9.0764327	7.9513601	10.0644943	10.0163178	9.5203505	8.6743812
43	8.0326069	8.8361285	7.7732905	10.0131974	10.0644943	9.4686524	8.6363300
44	5.9086086	6.4355734	5.4313217	7.7732905	7.9513601	7.2354593	6.4335214
45	7.0433820	7.3518423	6.4355734	8.8361285	9.0764327	8.3062689	7.5280199
46	6.0082439	7.0433820	5.9086086	8.0326069	8.0226001	7.4878189	6.6445001
47	8.7117436	10.0178224	8.8457535	10.7928303	10.6281376	10.2560556	9.4070262
48	7.0966338	7.9049535	6.8336692	9.0712557	9.1316726	8.5268179	7.6955090
49	7.4735606	8.6745548	7.5165007	9.5388707	9.4366117	8.9975262	8.1483514
50	6.5551629	7.7803548	6.6131562	8.6200519	8.5225971	8.0787833	7.2295729
51	6.5802159	7.7628110	6.6033351	8.6384396	8.5583775	8.0960952	7.2474956
52	7.9171630	9.4332055	8.2297623	10.0051216	9.7541949	9.4796013	8.6378497
53	7.2538542	7.8507876	6.8413471	9.1616859	9.2986016	8.6211175	7.8081554
54	7.6162952	8.9111284	7.7371524	9.6939893	9.5490457	9.1558937	8.3065330
55	6.1297839	6.9548258	5.8698557	8.0994360	8.1675430	7.5550355	6.7242931
56	7.8409462	8.7210035	7.6366004	9.8412818	9.8649604	9.2962682	8.4590016
57	8.5033548	9.4798138	8.3752593	10.5310670	10.5089305	9.9862289	9.1430774
58	6.6863657	8.0731370	6.8817614	8.7699598	8.5998938	8.2349393	7.3866845
59	7.4570618	8.4220502	7.3140917	9.4761963	9.4711118	8.9312059	8.0895452
60	7.3242924	7.7777903	6.8148040	9.1796526	9.3671457	8.6435975	7.8462492

	22	23	24	25	26	27	28
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							

19							
20							
21							
22							
23	2.4808095						
24	1.8942357	2.4449341					
25	2.6601465	3.1215157	0.8319059				
26	2.0759380	2.6867434	0.2568895	0.5974223			
27	2.3987478	3.2530937	0.8135243	0.4719965	0.5663548		
28	1.1252315	2.4626326	0.8065618	1.5353880	0.9576265	1.2922953	
29	2.1389718	3.9680795	1.7760529	1.8439713	1.6672372	1.3720704	1.5095990
30	0.9588055	3.2708216	1.8602136	2.4157635	1.9332490	2.0255752	1.1086224
31	5.6929852	7.8623048	7.3709451	7.9158214	7.4557015	7.4886577	6.5798600
32	6.6207038	8.9147212	8.1592205	8.6163363	8.2109334	8.1658225	7.3973370
33	6.6859342	8.7505188	8.4152658	8.9853115	8.5108386	8.5651930	7.6171799
34	7.7086898	9.8844617	9.3466897	9.8523498	9.4174878	9.4119484	8.5651930
35	7.6058060	9.7018389	9.3033828	9.8471462	9.3889505	9.4174878	8.5108386
36	8.1087261	10.2463877	9.7711930	10.2896884	9.8471462	9.8523498	8.9853115
37	7.4910209	9.5521131	9.2115662	9.7711930	9.3033828	9.3466897	8.4152658
38	7.6770343	9.2563155	9.5521131	10.2463877	9.7018389	9.8844617	8.7505188
39	5.7023164	7.6770343	7.4910209	8.1087261	7.6058060	7.7086898	6.6859342
40	6.4794028	8.4517413	8.2580996	8.8640391	8.3683592	8.4578375	7.4540810
41	7.3136508	9.2384337	9.1027162	9.7127230	9.2150935	9.3072152	8.2980207
42	7.9099286	9.9873119	9.6156200	10.1631482	9.7028288	9.7343921	8.8219266
43	7.8584715	9.7708525	9.6472693	10.2549485	9.7589003	9.8479099	8.8427113
44	5.6550795	7.3805957	7.5182480	8.2021459	7.6624269	7.8366025	6.7143450
45	6.7566772	8.3040930	8.6393814	9.3475815	8.7954644	8.9962511	7.8410526
46	5.8754638	7.9479442	7.6132943	8.1943015	7.7128236	7.7791856	6.8135144
47	8.6708377	10.8949396	10.2528753	10.7192515	10.3093568	10.2692208	9.4842075
48	6.9172077	8.8367276	8.7125317	9.3291432	8.8275137	8.9269125	7.9072723
49	7.3968239	9.5639907	9.0462325	9.5608326	9.1202917	9.1230117	8.2621979
50	6.4787578	8.6619619	8.1302884	8.6527445	8.2068005	8.2179539	7.3448287
51	6.4921471	8.6509322	8.1625565	8.6963016	8.2435042	8.2647382	7.3737079
52	7.9354947	10.2662919	9.4006355	9.8064723	9.4345727	9.3463108	8.6587816
53	7.0278444	8.7960158	8.8701811	9.5275553	9.0028825	9.1447746	8.0639057
54	7.5667247	9.7863323	9.1682076	9.6530160	9.2309832	9.2076867	8.3940415
55	5.9457281	7.8815205	7.7465875	8.3721933	7.8649037	7.9753026	6.9407859
56	7.6838193	9.6490721	9.4517696	10.0440833	9.5569505	9.6310738	8.6492333
57	8.3736839	10.4005669	10.1038217	10.6659105	10.1970323	10.2414657	9.3065269
58	6.6579483	8.9276625	8.2239891	8.6958514	8.2812527	8.2483774	7.4560804
59	7.3183002	9.3408539	9.0623097	9.6389646	9.1609098	9.2204963	8.2625644
60	7.0686015	8.7281080	8.9339637	9.6164529	9.0780362	9.2478788	8.1301291
	29	30	31	32	33	34	35

2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30	1.2947090						
31	6.3005149	5.5224526					
32	6.8846616	6.3005149	1.2947090				
33	7.3973370	6.5798600	1.1086224	1.5095990			
34	8.1658225	7.4886577	2.0255752	1.3720704	1.2922953		
35	8.2109334	7.4557015	1.9332490	1.6672372	0.9576265	0.5663548	
36	8.6163363	7.9158214	2.4157635	1.8439713	1.5353880	0.4719965	0.5974223
37	8.1592205	7.3709451	1.8602136	1.7760529	0.8065618	0.8135243	0.2568895
38	8.9147212	7.8623048	3.2708216	3.9680795	2.4626326	3.2530937	2.6867434
39	6.6207038	5.6929852	0.9588055	2.1389718	1.1252315	2.3987478	2.0759380
40	7.3416011	6.4489119	1.2328753	1.9867145	0.5444464	1.8016832	1.3810674
41	8.1884897	7.2973481	1.9921459	2.3690291	0.9386444	1.6222070	1.0632421
42	8.5300461	7.7696263	2.2477662	1.9500588	1.2370508	0.6966520	0.3196484
43	8.7210684	7.8398897	2.4832363	2.6682501	1.3860797	1.6545789	1.1057667
44	6.8723107	5.8136031	1.9503243	3.1362148	1.9264151	3.2082737	2.7781965

45	8.0700154	6.9808428	2.8516602	3.8092259	2.3413246	3.4259851	2.8868859
46	6.6346603	5.7843110	0.5272683	1.6291075	0.8112359	1.9915940	1.7540078
47	8.9855141	8.3928598	3.0372613	2.1034604	2.3485046	1.0618728	1.4860825
48	7.8220935	6.9133938	1.7035465	2.2844205	0.7769310	1.7828231	1.2651658
49	7.8873225	7.1897150	1.7083370	1.1855026	0.9949714	0.3228504	0.5076657
50	6.9969689	6.2751209	0.7998043	0.8153834	0.7179889	1.2305661	1.2216464
51	7.0557300	6.3106307	0.7991923	0.9655182	0.5681364	1.2337227	1.1553009
52	8.0253900	7.5528729	2.6279214	1.3780724	2.4116292	1.3783204	1.9406718
53	8.1077162	7.1197418	2.2668069	2.9910671	1.4818131	2.4478220	1.8990845
54	7.9436801	7.3081259	1.9328098	1.0846217	1.3948054	0.3515546	0.8717842
55	6.8965905	5.9568783	1.1513070	2.2311844	1.0289664	2.3212608	1.9446753
56	8.4839205	7.6317408	2.2111870	2.3371063	1.1033178	1.3512375	0.7910449
57	9.0510233	8.2648885	2.7542317	2.4940229	1.6900011	1.1822662	0.8616702
58	6.9787867	6.3639810	1.2097424	0.1938721	1.3226022	1.2268334	1.4869779
59	8.0555015	7.2312682	1.7634074	1.8915507	0.6581773	1.0940761	0.5536542
60	8.2597934	7.2234840	2.6449983	3.4491131	1.9437066	2.9185806	2.3651113

	36	37	38	39	40	41	42
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							

28							
29							
30							
31							
32							
33							
34							
35							
36							
37	0.8319059						
38	3.1215157	2.4449341					
39	2.6601465	1.8942357	2.4808095				
40	1.9783184	1.1666802	2.0669262	0.7804902			
41	1.5847064	0.8092987	1.6567540	1.6129879	0.8495075		
42	0.4741245	0.4406510	2.6518298	2.3346856	1.6034025	1.1184599	
43	1.4598932	0.9124550	1.6686057	2.1580439	1.3909871	0.5450710	0.9993394
44	3.3746986	2.5481510	2.0491866	1.0043949	1.4081619	1.9704021	2.9702330
45	3.4247793	2.6300056	0.9523596	1.9313611	1.8246519	1.8400800	2.9568973
46	2.3072972	1.6176880	2.7487481	0.5100661	0.7457949	1.5651078	2.0449039
47	0.9184123	1.7358922	4.0119251	3.4604243	2.8400535	2.5031173	1.3887723
48	1.8401926	1.0128384	1.6857086	1.2217510	0.4890783	0.4018990	1.4011127
49	0.7296065	0.6920707	3.1175855	2.0858770	1.5216663	1.4626767	0.7660416
50	1.6409059	1.2193051	3.1781762	1.4054129	1.1714619	1.6335996	1.5410437
51	1.6166961	1.1248751	3.0264035	1.2986041	1.0214366	1.4916046	1.4743173
52	1.6564692	2.1765837	4.6212580	3.3465091	2.9558683	2.9730202	2.0501993
53	2.4288154	1.6425309	1.0092431	1.5344926	1.0579862	0.8444272	1.9605463
54	0.7923287	1.0956788	3.5375518	2.4422420	1.9330587	1.8870606	1.0441427
55	2.5391333	1.7412277	2.2054084	0.2832867	0.5806620	1.3708518	2.1801968
56	1.2131582	0.5832356	1.9174125	1.9872692	1.2068586	0.4501105	0.7393948
57	0.7877290	0.8947990	2.5284776	2.7236527	1.9536508	1.2647637	0.5498726
58	1.6978537	1.5873994	3.7779196	2.0050157	1.8098206	2.1754178	1.7763544
59	1.1278957	0.2991043	2.1947738	1.6655505	0.9080047	0.5394378	0.7064278
60	2.8791782	2.1095051	0.6693668	1.8195717	1.4836672	1.3032399	2.4069768

	43	44	45	46	47	48	49
2							
3							
4							
5							
6							
7							
8							
9							
10							

11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							
41							
42							
43							
44	2.4574679						
45	2.1122781	1.2191739					
46	2.0883106	1.5096192	2.3318594				
47	2.3433212	4.2476324	4.3431847	3.0486800			
48	0.9427183	1.5959039	1.6504334	1.2329594	2.7487184		
49	1.5983715	2.9204542	3.2120681	1.6703519	1.3793244	1.5616815	
50	2.0167639	2.3739463	2.9940174	0.9083521	2.2385490	1.4931533	0.9188462
51	1.8901228	2.2506254	2.8452041	0.8166323	2.2678862	1.3418026	0.9131838
52	3.0314665	4.2914418	4.7154953	2.8514758	1.2309956	3.0686951	1.5151151
53	1.1584372	1.4154220	0.9963514	1.7490025	3.3471623	0.7167394	2.2548346

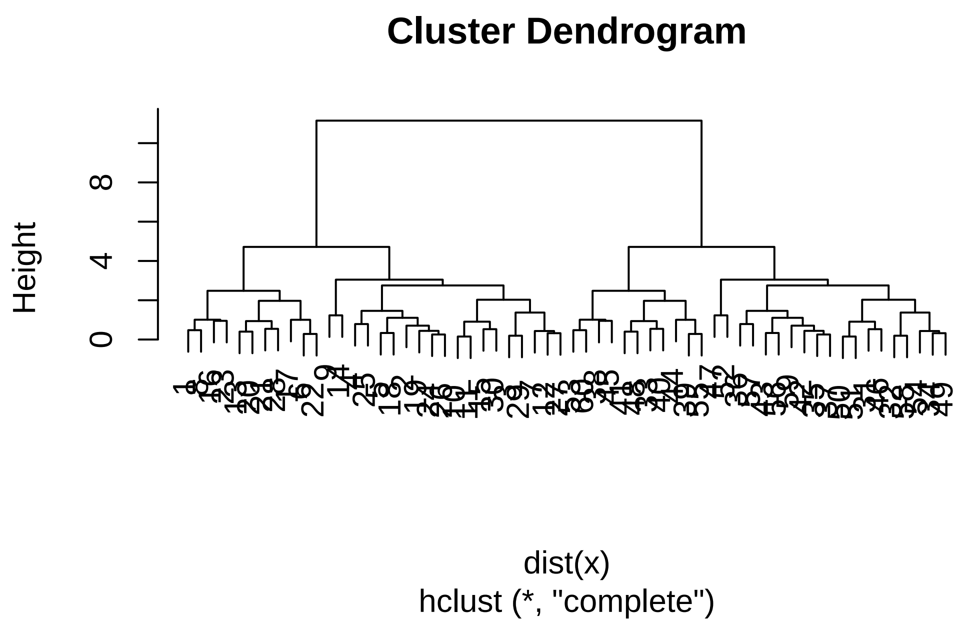
54	1.9767287	3.3181733	3.6442280	1.9913415	1.1090048	1.9939565	0.4325135
55	1.9143311	0.9273622	1.7014991	0.6407164	3.3761367	0.9718716	2.0207192
56	0.3320178	2.4174346	2.2422051	1.8528191	2.1225657	0.8294428	1.2731060
57	0.8957793	3.2286866	3.0048413	2.4985784	1.5327959	1.6343551	1.3085496
58	2.4761893	2.9943352	3.6341829	1.4955859	2.0288866	2.0955928	1.0163438
59	0.7894101	2.2640433	2.3360626	1.4490668	2.0348549	0.7138904	0.9233296
60	1.5302518	1.4157856	0.5864412	2.1184868	3.7957425	1.1898513	2.7306300

	50	51	52	53	54	55	56
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							

```
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51  0.1524376
52  1.9433493  2.0493875
53  2.1882836  2.0358685  3.7689072
54  1.1380713  1.1845110  1.0861004  2.6867018
55  1.4512946  1.3245240  3.3643224  1.2515224  2.4048570
56  1.7001191  1.5788134  2.7292795  1.2507829  1.6622005  1.7677900
57  2.0789083  2.0018495  2.4420088  2.0336703  1.5325042  2.5318309  0.8146532
58  0.6427331  0.7892208  1.4209777  2.8044033  0.9663929  2.0804103  2.1453430
59  1.2297469  1.1080469  2.4341933  1.3537875  1.3481035  1.4884893  0.4708675
60  2.6401074  2.4878439  4.2450561  0.4764897  3.1622276  1.5485430  1.6792976
           57         58         59
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58  2.3232150
59  1.0614181  1.6981917
60  2.4249736  3.2647404  1.8248934
```

```r
# there are other arguments you can add to hclust, but only the distance is required
hc <- hclust(dist(x))
plot(hc)
```




```r
# you can then make a specific cut line at a height you define
```

There are two different ways of going about this, "bottom up" or "top down". For "bottom up" you start as each point being its own cluster, then group them based on spacing in a stepwise manner until you only have a single cluster containing all points. The "top down" approach is similar, but in reverse order starting from a single cluster and parsing out to each point being its own cluster.

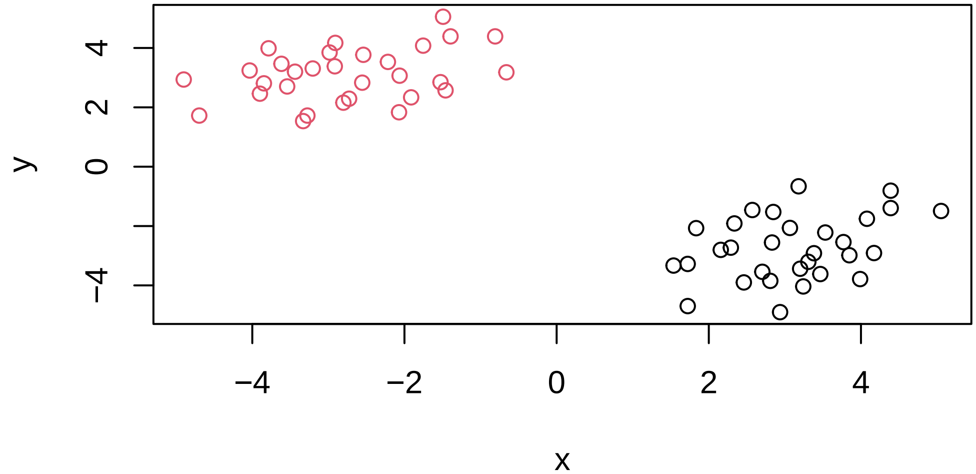
The function to actually cut your group into your desired number of clusters based on height is called `cutree()`. This will take two arguments, first being the hierarchal cluster `hclust` and second being the height at which you want it to cut as represented by `h=`

```r
grps <- cutree(hc, h=8)
grps
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Q6. Plot our hclust results in terms of our data colored by cluster membership

```
plot(x,col=grps)
```



Principal Component Analysis (PCA)

Eiganvector = a principle component. Once you create these, you can get rid of your original axis and only look at the PCA one. It makes it easier to visualize your data. The PCA are measurements of variation (aka spread) of your data. With the highest of amount of variance (or differences) in PCA1 with decreasing variance in sequential order for PCA2, ect. These coordinates do a better job of describing the data than the original coordinates.

Question 1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

```
url <- "https://tinyurl.com/UK-foods"
#make sure to assign the row names here because otherwise it may assume that the first col
x <- read.csv(url, row.names=1)
head(x)
```

```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             156   175      147       139
```

```r
str(x)
```

```
'data.frame':	17 obs. of  4 variables:
 $ England  : int  105 245 685 147 193 156 720 253 488 198 ...
 $ Wales    : int  103 227 803 160 235 175 874 265 570 203 ...
 $ Scotland : int  103 242 750 122 184 147 566 171 418 220 ...
 $ N.Ireland: int  66 267 586 93 209 139 1033 143 355 187 ...
```

So there are 17 rows (called objects) and 4 columns (called variables), but lets double check with less data using the `dim()` function

```r
dim(x)
```

```
[1] 17  4
```

> Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

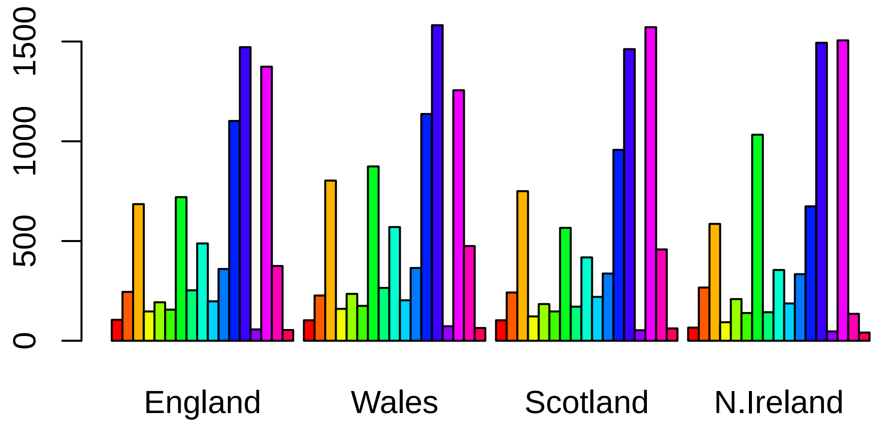
Theoretically, you could use the below as commands

```r
# rownames(x) <- x[,1]
# x <- x[,-1]
```

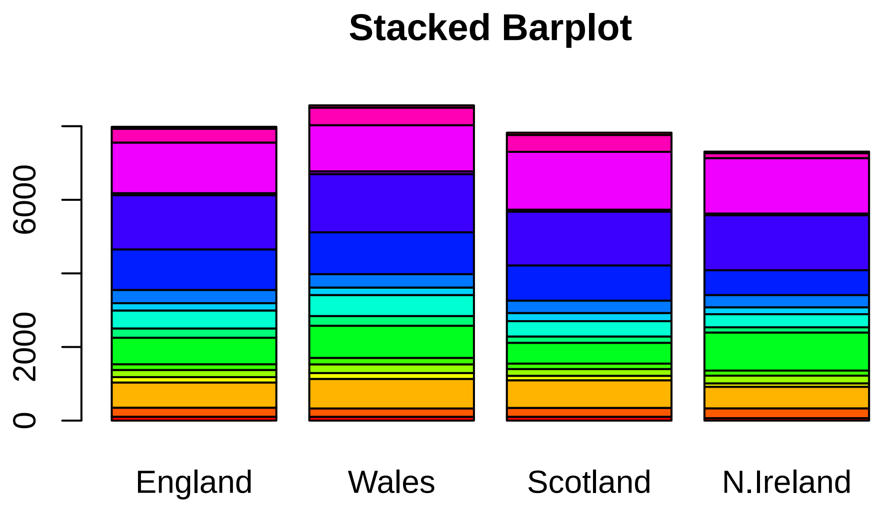
but this would overwrite the first column with the row names and each time you run it, it would overwrite yet another column until you have no data at all (which is bad, so don't do this).

> Q3: Changing what optional argument in the below barplot() function results in the following plot (a stacked barplot)?

```r
# original barplot
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



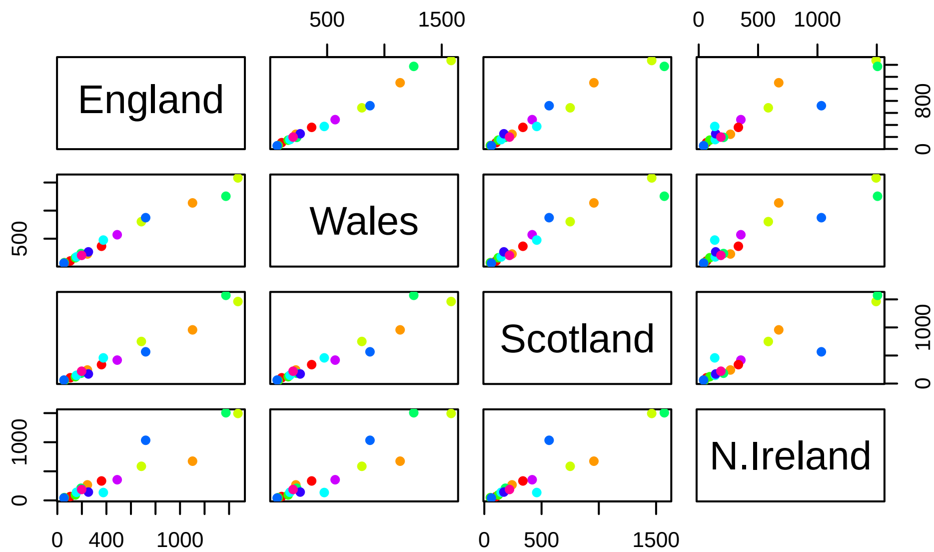

```r
# stacked barplot
barplot(as.matrix(x), main="Stacked Barplot", beside=F, col=rainbow(nrow(x)))
```



So you would change the `beside=` argument to “FALSE”

> Q5: Generating all pairwise plots may help somewhat. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

```
pairs(x, col=rainbow(10), pch=16)
```



The way that this is read is that for the top row England is on the y-axis, for the second row, wales is on the y-axis, third row is Scotland on the y-axis, and bottom row is N. Ireland on the y-axis. For the columns, the first column is England on the x-axis, the second column is Wales on the x-axis, Scotland is on the x-axis in the third column, and N.Ireland is the x-axis for the fourth column.

If a given point lies on the diagonal, that means that there is equal amounts of that type of food consumption in both countries.

> Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

This can be difficult to visualize just looking at the graphs above, but the graphs containing N.Ireland as compared to the other 3 (fourth column and row), the points that are most off of the diagonal are those that are the most different between the two countries. For example, in N.Ireland vs Scotland, the dark blue datapoint is very different between the two.

##PCA to the rescue help me make sense of this data…the main function for PCA in base R is called `prcomp()`. But weirdly, it wants the food names in the columns (aka observations) and the countries in the rows. We can do this by using the transpose function `t()`.

```
Tx <- t(x)
head(Tx)
```

```
          Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
England      105           245         685   147           193     156
Wales        103           227         803   160           235     175
Scotland     103           242         750   122           184     147
N.Ireland     66           267         586    93           209     139
          Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England              720        253        488                 198
Wales                874        265        570                 203
Scotland             566        171        418                 220
N.Ireland           1033        143        355                 187
          Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
England             360         1102     1472         57        1374
Wales               365         1137     1582         73        1256
Scotland            337          957     1462         53        1572
N.Ireland           334          674     1494         47        1506
          Alcoholic_drinks  Confectionery
England                375             54
Wales                  475             64
Scotland               458             62
N.Ireland              135             41
```

```r
pca <- prcomp(t(x))
summary(pca)
```

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

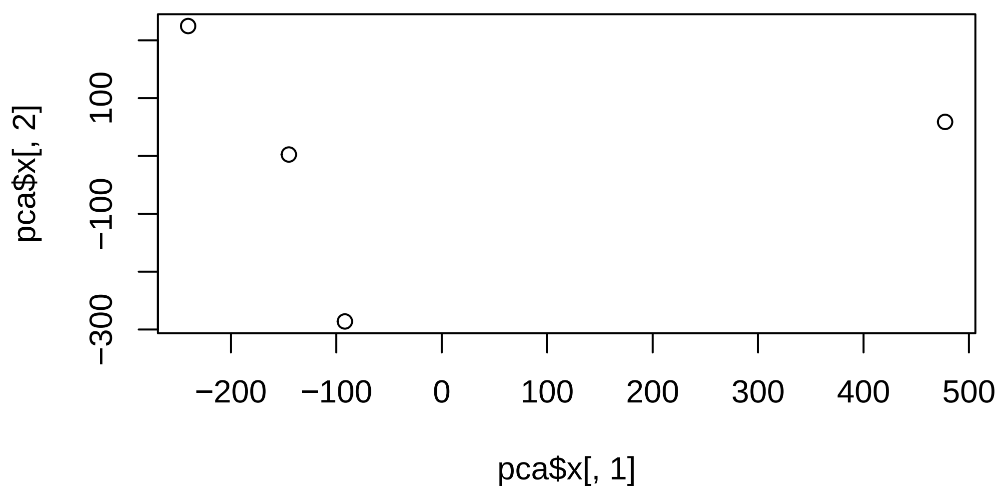
```r
?pca
```

```
No documentation for 'pca' in specified packages and libraries:
you could try '??pca'
```

```r
# we can look at different ways
print(pca$x)
```

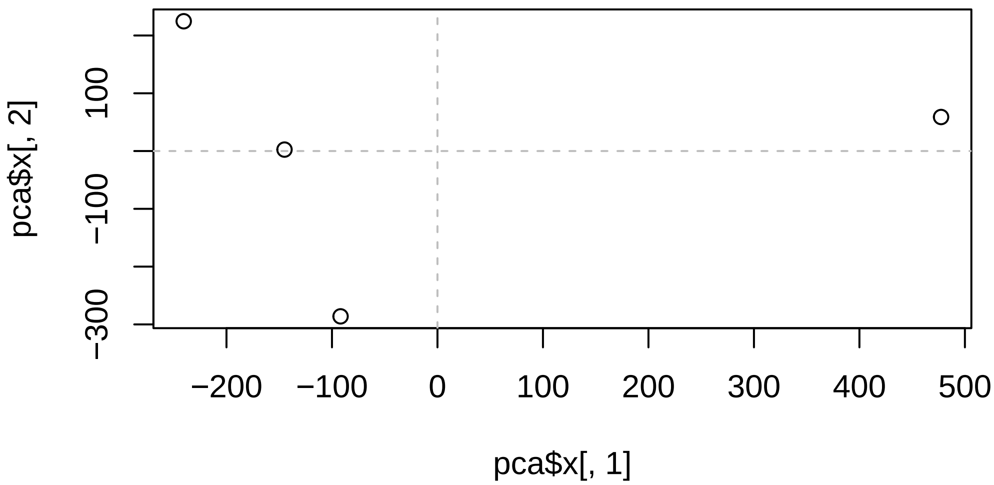
```
                PC1          PC2          PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```r
# now making a plot
plot(pca$x[,1], pca$x[,2])
```



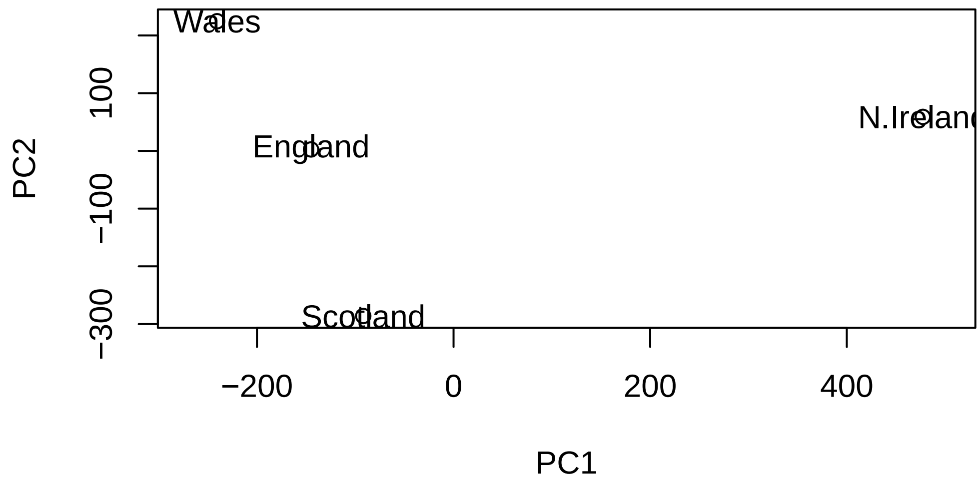
We can also add a line such that we can see where the zero lies using the `abline()` function. This adds a line as opposed to the `point()` function we used before

```r
plot(pca$x[,1], pca$x[,2])
abline(h=0, v=0, col ="gray", lty =2)
```



Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

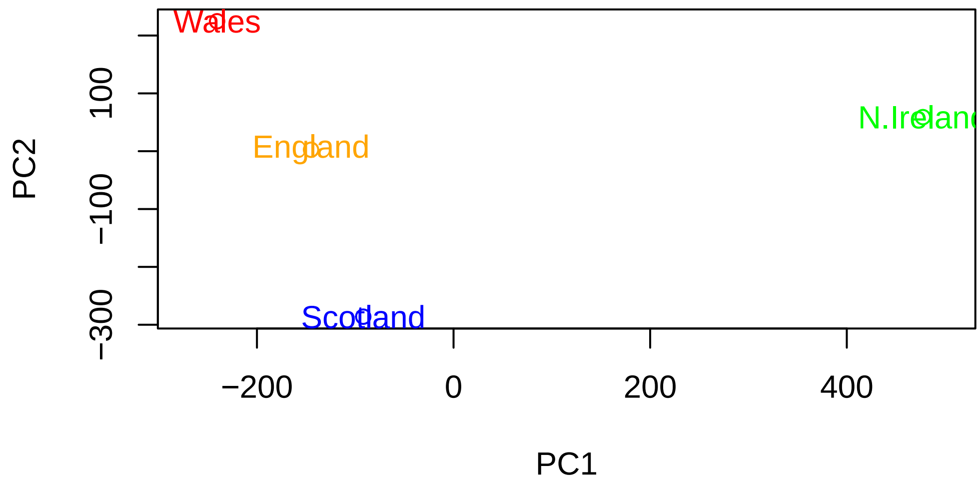
```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```



So here we are telling to plot PCA1 which is in the first column, and PCA2 which is the second column in the PCA plot

> Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

```
# this colors the points
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500), col = c("orange", "re
#this is colors the text
text(pca$x[,1], pca$x[,2], colnames(x), col = c("orange", "red","blue","green"))
```



In our data it's ordered:England, Wales, Scotland, then N. Ireland. We want England =yellow, Wales=red,Scotland=blue, and N.Ireland=green

##Loading Plots

```
summary(pca)
```

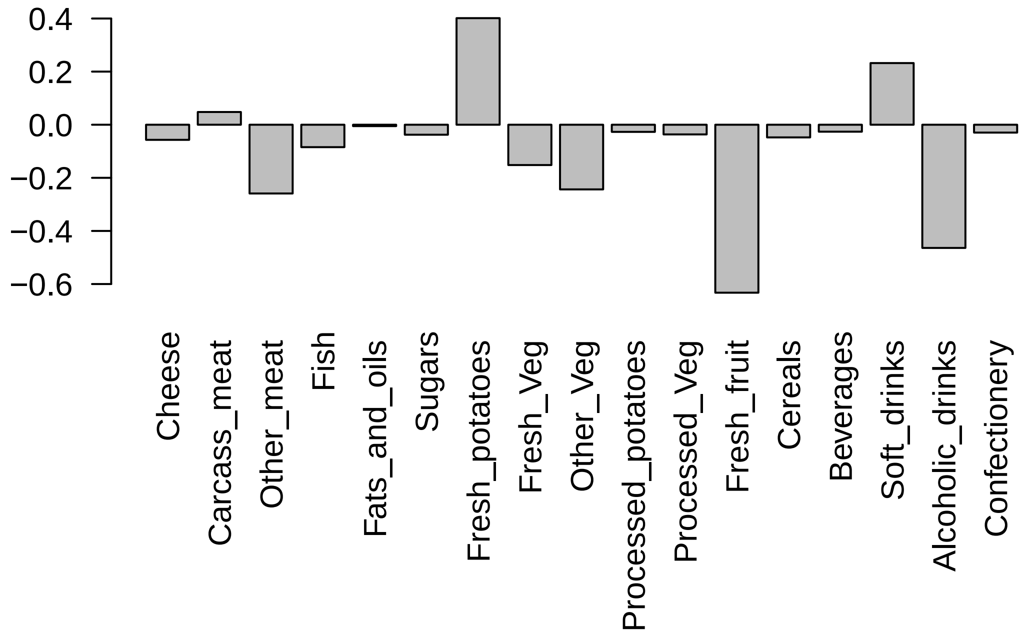
```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

Looking at the summary, the proportion of variance is represented as a decimal, so here 67% of the variance in the data is in PC1, while 29% is in PC2. Together, PC1 and PC2 explain 96.5% of the variance as indiciated by the "cumulative proportion" row.

One of the main results that folks look for is called a "Score Plot" aka "PC Plot". The rotation value includes how much your individual categories (in this case food), determins the overall variance of your data (as opposed to potential confounding variables).

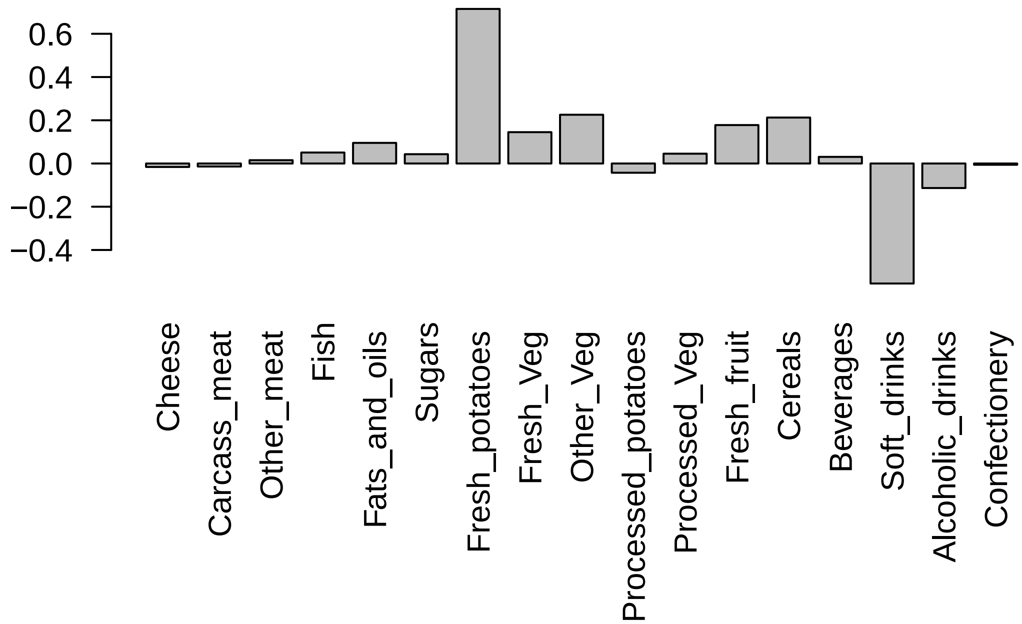
> Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

```
#PC1
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



This plot shows the highest contributes to variation (PC1) with the largest bars observations/foods with high negative scores that push the other countries to the left side of the plot primarily being increased consumption of fresh fruit and alcoholic drinks in other countries. But also the negative values "push" N. Ireland to right positive side of the plot (though a bit less) are primarily potatoes and softdrinks.

```
#PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



PC2 (the second highest level of variance) mainly shows us differences in fresh potatoes and soft drinks where positive values represent pushing values in N.Ireland and negative values "pushes" values in England consumption. For PC2 it seems that fresh potatoes are still consumed more highly in N.Ireland (highest contribute to the "push") and there are more soft drinks consumed in other countries (highest contribute to it's "push").