Class 7 Machine Learning 1

Shreyas Sankaranarayanan

In this document we will start to explore some key achine learning methods. We will begin with clustering which involves finding groupings in data. We will then move on to dimensionality reduction.

Clustering

Let's start with **"k-means"** clustering. The main functoin in Base R for this is `kmeans()`.

```
# Make up some data
rnorm(1000, mean = 3, )
```

```
   [1]  2.8746716  1.6980453  2.6733497  4.5393964  1.2639175  0.2048850
   [7]  1.4689862  1.8166323  3.6213650  3.0895463  3.5233872  2.3880217
  [13]  3.2894135  5.9123586  2.3532110  1.3542547  1.4852896  5.0730665
  [19]  2.6429525  2.6047347  4.2207672  1.6411502  2.7650459  4.5329135
  [25]  2.4930379  2.0003268  4.9804552  2.0448210  3.9176655  2.1024438
  [31]  4.4452242  3.2966268  4.3221573  3.2820404  2.8146271  2.3159917
  [37]  4.8973624  2.4362908  2.7797261  2.3399006  2.4514870  1.6328191
  [43]  2.0040619  2.3601361  2.9092665  2.9107384  3.3600359  1.9200721
  [49]  3.3946007  4.0609766  1.3102632  2.8578400  2.7159356  3.0915521
  [55]  3.5469360  3.7794700  4.2908096  2.6149767  3.5623776  0.8841569
  [61]  3.2461954  3.2317885  2.1465898  2.7696507  3.8848459  1.6767148
  [67]  3.7279805  3.0473246  3.0812862  3.2172748  4.3768642  3.2694389
  [73]  3.5578934  3.1668622  2.8327892  2.9272490  4.0206461  4.0344812
  [79]  2.5084254  3.5382084  2.9869925  1.9822156  2.6947133  3.5216204
  [85]  2.8867070  2.0276238  1.8999742  3.0919266  1.8552373  4.4742858
  [91]  2.7648413  2.7315546  2.7426368  4.0891493  2.6203508  3.2748791
  [97]  2.9771683  1.9565858  3.4761215  2.5706172  2.3114597  3.6675852
 [103]  0.8913229  3.6936189  2.3722170  4.6347701  2.8308444  4.1054334
 [109]  4.4707562  1.3659486  1.6682229  2.5071944  3.6661541  3.7634358
 [115]  3.3115995  0.7888608  3.9731254  3.2368243  3.3117910  4.5269856
```

```
[121]  1.5491558  4.8332232  1.8785229  2.9291314  3.9337244  2.8017831
[127]  2.3679526  2.8952256  3.3162090  2.9179702  3.5062746  0.9228504
[133]  3.2339269  4.3196591  0.6816519  3.1033643  2.3927853  1.9959526
[139]  4.4328618  3.7418744  3.0099112  2.5865323  3.0443719  2.6394081
[145]  1.5372175  2.2014810  3.0213932  3.2895837  2.3240522  4.7626216
[151]  2.2603860  3.4999821  3.4550911  3.7615120  3.0402225  1.1781911
[157]  2.6782531  3.9792358  3.6223159  1.8368084  1.0803227  1.9184458
[163]  2.4988736  4.3684394  2.9320594  2.1967003  2.8428726  2.9945492
[169]  4.3261311  2.1307840  4.1856787  1.8910408  1.9837869  2.9328464
[175]  3.6442449  3.6680576  1.1372964  2.6114408  2.6376566  1.1223127
[181]  2.0120476  4.2707998  4.8461116  3.2411934  3.0967088  3.3069276
[187]  3.3148148 -0.5915540  1.8005495  2.2981563  3.8802507  1.8583576
[193]  2.6318910  3.1980112  2.1270565  3.2326944  2.3597868  3.3051789
[199]  2.3918458  2.5411629  2.5585726  3.6393580  3.6103607  3.1185572
[205]  3.6761788  1.5848312  3.6403361  2.8855476  2.3170715  2.1470588
[211]  1.9920108  3.0710575  1.7104673  2.8711989  2.9348990  3.5027737
[217]  3.9048589  4.1924043  3.0853438  3.7472629  3.8505720  4.6486360
[223]  4.0933643  1.9646536  4.0839908  3.1613443  3.3065460  3.9383208
[229]  2.2179804  3.4409041  2.3716856  2.4852266  3.8437111  3.1057057
[235]  3.2116994  2.2473110  2.4797318  2.5007719  4.7816130  4.4174538
[241]  1.7917215  3.3967897  3.4019245  3.0899395  2.4265795  2.8933797
[247]  4.1661042  1.6235761  3.6020616  1.7808814  2.6568612  0.4352351
[253]  3.9450311  3.6435329  2.5048517  1.3072158  2.0832243  1.6649420
[259]  2.6588860  4.2862468  2.7325270  1.7865323  3.4294293  2.7145716
[265]  2.2463982  3.4163822  3.2183385  4.2889983  3.7319864  3.6643579
[271]  2.3551319  3.7099839  2.7827479  1.5075479  4.4259308  2.7967447
[277]  4.2472014  3.6563891  4.1090868  3.2527978  2.8001164  3.2332368
[283]  4.2945821  2.9649033  3.7392635  4.0479089  2.5876511  4.5760858
[289]  3.2083944  4.3306472  2.6154717  3.2300870  2.3709424  3.6181833
[295]  2.1280017  3.3740890  2.3272907  2.6502199  5.7803509  2.6795807
[301]  3.9180959  5.1695534  3.5944963  3.2203912  3.2080602  2.4114811
[307]  4.9438410  2.2297060  3.1110888  4.1116594  2.7631679  2.9249800
[313]  2.5904350  3.2945491  2.9003987  3.2003889  3.5452316  4.6707244
[319]  4.1246553  2.1310074  4.0616968  4.0678494  1.8477201  4.4580926
[325]  4.7418405  1.3914488  4.3885993  3.9908002  2.0880398  3.1011490
[331]  3.2195991  3.4784987  3.0797271  4.0062752  2.7294941  3.2922748
[337]  4.2987506  5.6301846  2.1079066  3.4781669  4.0893516  2.4607021
[343]  2.9145147  3.8849215  3.3030173  3.3235278  2.2495863  1.3858839
[349]  1.5484181  3.1221676  3.0700484  3.2480980  1.9870394  2.5961533
[355]  3.1024519  2.6785767  3.4139245  3.2010263  2.7239905  4.6999186
[361]  3.0143099  3.5274616  3.7935774  2.2826447  3.3201342  4.0066285
[367]  2.1390705  3.2269806  4.3943523  3.5827701  1.9498236  2.3665512
[373]  2.5279275  2.5082658  2.4858782  3.3945910  3.2434646  3.0567436
```

```
[379] 2.0593651 2.4590465 3.6936460 2.0437614 3.1501763 1.1220409
[385] 2.6931411 2.7974346 0.3994333 3.6992752 4.4461722 2.3465696
[391] 3.5883640 2.8360993 2.9562676 3.7887960 4.2364804 3.2029421
[397] 1.4682626 3.8952536 4.9885096 2.1211044 4.4789667 2.4977686
[403] 2.9058385 2.1942809 2.4414983 2.8287061 2.5122908 4.6418158
[409] 2.8907441 2.5572074 1.4105482 2.4912876 3.2028166 3.5909987
[415] 4.6818708 2.6943497 2.6973965 1.3789362 3.9212886 0.6680640
[421] 3.9230495 4.0910340 3.7185324 1.8684262 2.8105994 2.1778714
[427] 3.3064643 3.1975561 2.6644457 3.1862554 3.5382038 2.3379277
[433] 3.0543207 2.9266859 2.1748822 2.9619244 3.1226074 1.8645645
[439] 4.1733428 2.0597605 2.5352382 1.2594938 1.6614925 3.0377121
[445] 3.7000197 3.8494968 1.9131267 3.1151206 3.0781302 2.7386159
[451] 4.1616454 4.5306489 2.8811661 4.4077022 3.4305832 1.0951826
[457] 3.0111836 3.5820439 1.8690713 4.2730332 2.3860390 3.4973843
[463] 3.6712685 2.4532480 2.2883730 3.8279934 3.9778560 3.5969983
[469] 3.6082207 1.9947493 2.9394177 5.1561906 2.3170257 4.8891981
[475] 2.1456731 3.2387840 2.3427380 2.0393871 4.3271885 2.5874208
[481] 3.7691820 2.2672610 2.8229860 2.4671741 4.4229544 2.7756845
[487] 1.3324820 4.4546854 1.8306753 1.6013153 3.3002433 3.8286251
[493] 3.0236168 4.2325644 3.1275138 3.3853684 2.2692390 3.2590201
[499] 1.8081331 3.1491563 3.9762747 1.4672524 3.1401052 2.1785544
[505] 2.5640351 2.9723297 3.6299688 4.4604586 4.3322245 1.5035311
[511] 3.0209212 2.9289473 2.2117100 3.4506221 2.4557550 3.9526728
[517] 2.8842977 2.3142072 2.6147592 1.0899642 2.9532073 2.5174709
[523] 3.2174100 4.0617151 4.1897835 2.5808775 3.8005556 4.2514056
[529] 3.1801999 2.5196723 2.7086765 2.8581934 1.6747591 3.0323961
[535] 3.4258860 3.1769690 3.1128156 2.6339842 0.6284412 2.2610325
[541] 2.7632306 3.5240427 2.2345230 2.8025553 4.5588476 3.8461545
[547] 3.8470171 2.3128223 4.6546779 2.9000304 1.1361605 2.4345141
[553] 3.2640838 3.3287540 3.3726847 2.8925948 1.7511439 4.5614897
[559] 3.1194445 1.9919010 3.6374437 3.5358748 3.6132601 4.3512796
[565] 2.8539703 3.5561808 4.1823690 4.4486658 4.6283644 1.9734230
[571] 2.0060461 3.0175669 2.7760413 2.5450058 2.9551194 3.1731272
[577] 4.2872098 1.8508663 2.4224364 2.1631349 2.2192783 3.0443892
[583] 3.1481767 3.5481648 2.9461439 4.2729301 4.2157522 3.6967498
[589] 3.8495548 1.3576335 2.1660602 2.3314078 2.9713080 2.9018099
[595] 1.9204328 3.4725423 2.8083240 2.0214267 2.9323999 2.3972170
[601] 3.4928772 3.3453851 3.6231150 3.7172261 4.2254461 2.3812781
[607] 2.9319285 2.8748995 2.7409784 3.9643390 2.5039574 2.9575994
[613] 4.3100526 4.3768469 2.5505208 2.3099415 1.5059420 2.8033252
[619] 2.3182198 1.9596608 3.1903739 4.3398760 2.9541161 3.8314409
[625] 2.5860882 2.3986195 3.1638346 3.7844935 3.1655439 1.9365805
[631] 1.9868145 2.2750789 2.8252497 3.6741015 4.0141953 3.3234723
```

```
[637] 2.8087327 3.8852507 4.0900343 2.2260226 3.1431604 3.8706198
[643] 4.1949431 2.9199556 5.1497356 1.7690404 2.1118888 3.3719129
[649] 3.0929496 3.6352151 3.2415098 2.1796981 2.0792937 1.8752520
[655] 3.5436986 3.7244269 2.9848658 3.5206337 2.8474401 3.1842909
[661] 1.7654415 2.3101030 3.9131212 4.3476669 1.7972720 2.9262553
[667] 2.9464779 1.0952120 2.5164165 2.6112894 1.1509299 4.2609699
[673] 4.5474956 2.9279485 3.2667104 3.2336669 3.4823483 2.7034965
[679] 2.3075117 2.2051504 2.7033354 2.4040562 4.1360758 1.8800713
[685] 3.5501138 4.2880409 2.8096268 3.1484082 2.3562595 3.2360291
[691] 2.8172697 0.8572463 3.9623064 3.5768907 1.3584014 2.9281729
[697] 3.5852351 3.2244084 2.7635730 0.7171862 4.6730533 3.6681134
[703] 2.3844114 3.0156382 3.1245508 2.8961823 1.9481417 2.5375705
[709] 2.3362151 2.8212738 0.5941080 2.8394609 2.1939496 2.3941654
[715] 1.6145893 2.9693839 2.8142243 1.2361869 2.5494997 3.9994370
[721] 2.9233647 3.4090345 3.6333642 3.9769065 2.5791826 3.2340046
[727] 2.2552647 3.0532482 2.7141825 4.4273994 2.4861673 3.4988794
[733] 3.6718086 2.6576950 1.9829046 3.6638852 3.5247434 4.0638174
[739] 2.8010487 3.7965984 4.0544365 3.4839185 2.6661619 2.0202581
[745] 1.3915796 2.9784258 0.9943525 3.0296794 2.0336303 2.7173520
[751] 2.6070983 4.5973779 3.1513140 3.7508314 0.6294553 2.7552944
[757] 3.6399250 3.9494505 3.0289977 3.4979430 3.8242328 2.0165443
[763] 3.7162453 4.7816558 2.2756456 2.1471428 1.5270437 3.7729328
[769] 1.6142991 2.8889046 1.9446939 3.8123911 2.5629094 3.2458766
[775] 2.7576082 3.5819276 3.1436233 2.4387850 3.7020959 2.8378700
[781] 2.5006984 2.6733118 1.2020973 2.4096363 2.0034876 1.1140702
[787] 2.1305582 4.9151276 1.7382142 3.4326978 2.8479833 2.1962021
[793] 2.6927175 2.8709902 4.6023297 2.3266356 3.3291509 2.0480993
[799] 3.5022021 2.3034350 2.1704054 0.9748239 2.4177502 2.9880352
[805] 2.0216311 3.5581824 2.5698633 1.7671906 3.0102602 3.0881431
[811] 3.9647718 1.7391866 3.5411890 3.6769114 3.6040341 1.2304032
[817] 3.1899993 3.9640734 1.8656499 2.5938480 3.1979349 3.1038426
[823] 2.1777010 3.5673524 2.7769229 3.5201537 3.3624201 2.2335913
[829] 4.1591496 3.7207027 2.7807845 1.6784479 2.9628806 3.5494105
[835] 4.0700659 3.0036142 2.8586858 1.4083141 3.0739770 4.3497643
[841] 3.1086893 3.4753190 3.0727431 3.5989936 4.0430045 2.6988640
[847] 2.8965844 3.2317529 3.9520287 3.6525173 3.1839338 1.4305700
[853] 3.3330191 2.4491811 4.5169294 3.5786768 4.0925607 3.0029641
[859] 3.7799019 3.1535319 2.9859759 3.3641246 2.1150829 4.0240684
[865] 3.7098629 3.5160591 3.8604727 3.8926825 4.0145510 3.3219419
[871] 2.2868183 3.7783691 3.3879507 2.3753322 2.4093828 1.4197556
[877] 3.7640038 2.0441836 4.3187793 5.1221820 3.4442656 2.9715264
[883] 2.7011101 2.4402468 3.0597035 3.9961661 2.4222638 0.9062828
[889] 5.1630251 3.8182325 2.5504188 2.6986061 2.9744847 4.0078090
```

```
 [895]  3.1760469  2.9532980  3.4613943  2.6661356  3.8131846  2.0630504
 [901]  3.0902450  2.8878584  2.3271337  3.3710618  3.4324212  2.1803341
 [907]  1.1691189  2.9008293  2.6694719  2.8638878  2.7280397  1.6636235
 [913]  2.3666633  4.8146361  3.4381745  1.3673203  2.8977594  3.8347500
 [919]  1.0962643  3.6555622  3.6405139  4.0731440  3.0450765  1.9349989
 [925]  3.5762594  3.3274452  3.0046418  3.1160413  4.6239132  2.3495348
 [931]  4.2567347  2.5787465  2.6537362  3.0748882  3.5245930  4.8465475
 [937]  5.1013963  2.9234833  3.9482546  3.6306755  2.1225218  1.9306047
 [943]  3.1541051  2.5870005  2.8147477  2.2334575  4.1329614  2.1661373
 [949]  4.1642023  3.6473532  2.2754110  2.4147588  2.1122891  2.7641245
 [955]  2.0693579  1.8536265  1.0719140  2.2821163  3.5028556  0.5178372
 [961]  1.2615349  4.0135809  3.2147314  2.2918324  3.4975869  1.9345356
 [967]  3.8470178  1.4555513  5.2433358  3.1042070  3.2373087  4.0517608
 [973]  4.2213922  2.8805723  3.4679938  4.9471234  2.5739313  4.0314247
 [979]  4.7675089  2.6119777  2.4815325  3.8434142  1.7118540  3.6701789
 [985]  2.2693696  2.3911428  2.5876232  3.0127146  3.3399743  3.1089496
 [991]  2.6933405  2.5268251  3.5271795  4.7483548  3.4081626  5.0255342
 [997]  2.6615844  2.9119588  2.8773498  2.7818028
```

```
library(ggplot2)
tmp <- c(rnorm(30,-3), rnorm(30, +3))
x <- cbind(x = tmp, y= rev(tmp))
```

Now let's try the `kmeans()` algorithm

```
km <- kmeans(x, 2)
km
```

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

Cluster means:
          x         y
1  3.138545 -2.680385
2 -2.680385  3.138545

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

Within cluster sum of squares by cluster:
[1] 44.25586 44.25586
 (between_SS / total_SS =  92.0 %)
```

```
Available components:

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

Q. How many points in each cluster?

```
km$size
```

```
[1] 30 30
```

Q. What component of your result object details cluster assignment/membership?

```
km$cluster
```

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

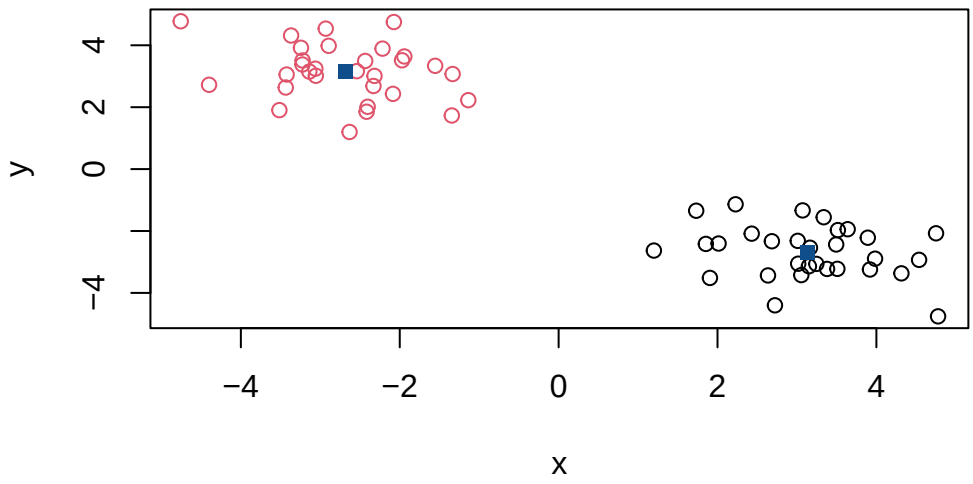
Q. What are center/mean values of each cluster?

```
km$centers
```

```
          x         y
1  3.138545 -2.680385
2 -2.680385  3.138545
```

Q. Make a plot of your data showing your clustering results (groupings clusters and cluster centers)

```
plot(x, col = km$cluster)
points(km$centers, col = "dodgerblue4", pch = 15, cex = 1)
```



Run `kmeans()` again and cluster in 4 groups and plot the results

First we run the `kmeans()` algorithm:

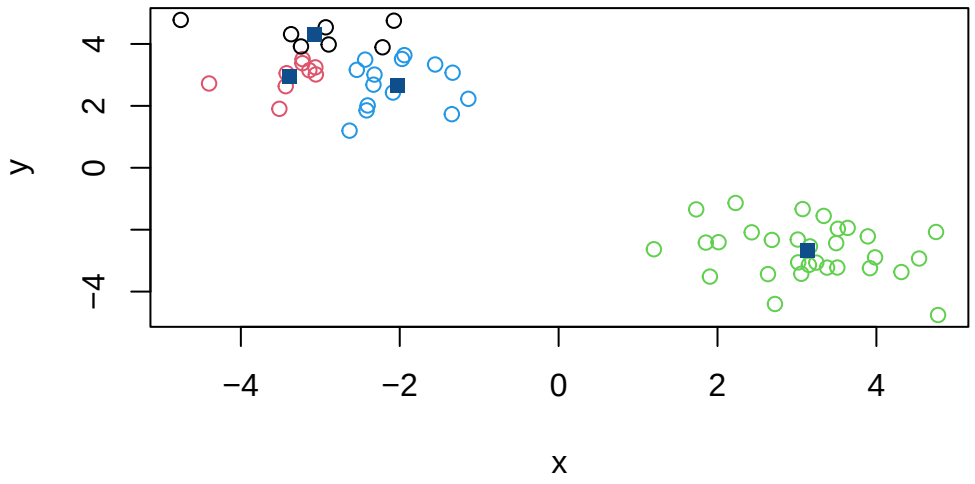
```
km2 <- kmeans(x,4)
km2
```

```
K-means clustering with 4 clusters of sizes 7, 9, 30, 14

Cluster means:
          x         y
1 -3.069348  4.310492
2 -3.386601  2.957190
3  3.138545 -2.680385
4 -2.031908  2.669157

Clustering vector:
 [1] 2 2 1 4 4 1 4 2 1 4 4 2 1 2 2 4 2 4 1 2 4 4 1 4 4 2 4 4 1 4 3 3 3 3 3 3 3 3
[39] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1]  5.633583  3.257897 44.255864 10.934592
 (between_SS / total_SS =  94.2 %)
```

```
Available components:

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

Then we plot the corresponding 4 clusters:

```
plot(x, col = km2$cluster)
points(km2$centers, col = "dodgerblue4", pch = 15, cex = 1)
```




```
nrow(x)
```

```
[1] 60
```

```
km3 <- data.frame(1:60)
z = NULL
for(y in 1:59){
  print(y)
  print(kmeans(x,y)$betweenss/kmeans(x,y)$totss)
```

```
  z <- c(z, kmeans(x,y)$betweenss/kmeans(x,y)$totss)
}
```

```
[1] 1
[1] -2.058966e-16
[1] 2
[1] 0.9198489
[1] 3
[1] 0.9358667
[1] 4
[1] 0.9518846
[1] 5
[1] 0.9577638
[1] 6
[1] 0.9640619
[1] 7
[1] 0.9695948
[1] 8
[1] 0.9715731
[1] 9
[1] 0.979022
[1] 10
[1] 0.9708365
[1] 11
[1] 0.9820717
[1] 12
[1] 0.9838238
[1] 13
[1] 0.9850194
[1] 14
[1] 0.9852791
[1] 15
[1] 0.988944
[1] 16
[1] 0.9795359
[1] 17
[1] 0.9908828
[1] 18
[1] 0.9915034
[1] 19
[1] 0.9931303
[1] 20
```

```
[1] 0.9936377
[1] 21
[1] 0.9935559
[1] 22
[1] 0.995907
[1] 23
[1] 0.9952164
[1] 24
[1] 0.9962257
[1] 25
[1] 0.9962368
[1] 26
[1] 0.9967546
[1] 27
[1] 0.9970368
[1] 28
[1] 0.9976029
[1] 29
[1] 0.9975595
[1] 30
[1] 0.9950355
[1] 31
[1] 0.9980786
[1] 32
[1] 0.9978351
[1] 33
[1] 0.9976357
[1] 34
[1] 0.9980894
[1] 35
[1] 0.9982578
[1] 36
[1] 0.9988097
[1] 37
[1] 0.9986888
[1] 38
[1] 0.9985501
[1] 39
[1] 0.9986367
[1] 40
[1] 0.9987816
[1] 41
[1] 0.9989192
```

```
[1] 42
[1] 0.999292
[1] 43
[1] 0.9993773
[1] 44
[1] 0.9992423
[1] 45
[1] 0.9993077
[1] 46
[1] 0.9994027
[1] 47
[1] 0.9993904
[1] 48
[1] 0.9994219
[1] 49
[1] 0.9995758
[1] 50
[1] 0.9995995
[1] 51
[1] 0.9996318
[1] 52
[1] 0.9997393
[1] 53
[1] 0.9997013
[1] 54
[1] 0.9998936
[1] 55
[1] 0.9998881
[1] 56
[1] 0.9999461
[1] 57
[1] 0.9998529
[1] 58
[1] 0.9999817
[1] 59
[1] 0.999943
```

```
print(z)
```

```
 [1] -2.058966e-16  9.198489e-01  9.358667e-01  9.517022e-01  9.591713e-01
 [6]  9.638958e-01  9.668447e-01  9.542347e-01  9.790220e-01  9.797125e-01
```

```
[11]  9.819254e-01  9.832965e-01  9.870270e-01  9.852628e-01  9.881463e-01
[16]  9.874566e-01  9.898328e-01  9.886068e-01  9.930429e-01  9.929749e-01
[21]  9.901215e-01  9.951296e-01  9.952461e-01  9.960932e-01  9.968172e-01
[26]  9.967713e-01  9.973029e-01  9.969181e-01  9.972800e-01  9.973819e-01
[31]  9.975490e-01  9.976011e-01  9.983296e-01  9.983711e-01  9.984599e-01
[36]  9.985172e-01  9.988674e-01  9.986406e-01  9.989819e-01  9.989819e-01
[41]  9.990291e-01  9.992923e-01  9.992174e-01  9.994097e-01  9.993418e-01
[46]  9.990998e-01  9.993968e-01  9.994734e-01  9.994930e-01  9.997740e-01
[51]  9.996360e-01  9.995908e-01  9.997260e-01  9.998648e-01  9.998884e-01
[56]  9.999413e-01  9.998288e-01  9.998860e-01  9.999883e-01
```

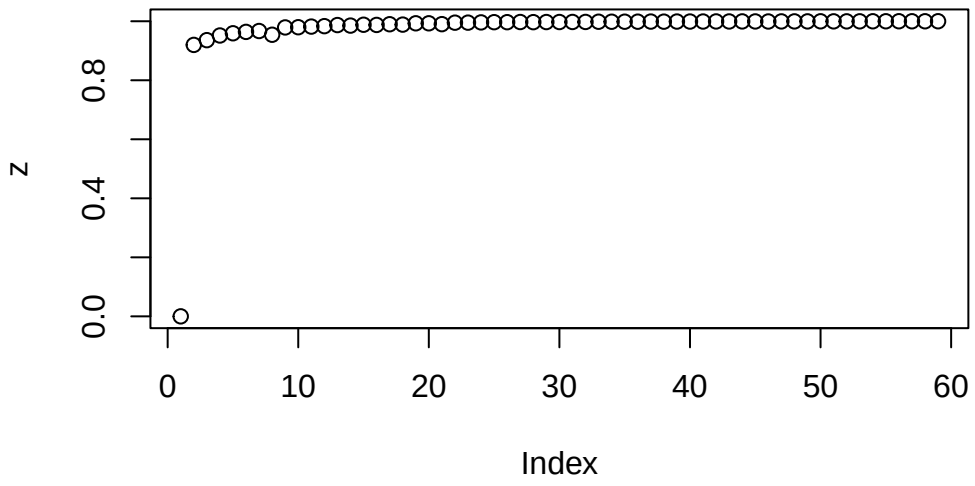
```
length(z)
```

```
[1] 59
```

```
z
```

```
 [1] -2.058966e-16  9.198489e-01  9.358667e-01  9.517022e-01  9.591713e-01
 [6]  9.638958e-01  9.668447e-01  9.542347e-01  9.790220e-01  9.797125e-01
[11]  9.819254e-01  9.832965e-01  9.870270e-01  9.852628e-01  9.881463e-01
[16]  9.874566e-01  9.898328e-01  9.886068e-01  9.930429e-01  9.929749e-01
[21]  9.901215e-01  9.951296e-01  9.952461e-01  9.960932e-01  9.968172e-01
[26]  9.967713e-01  9.973029e-01  9.969181e-01  9.972800e-01  9.973819e-01
[31]  9.975490e-01  9.976011e-01  9.983296e-01  9.983711e-01  9.984599e-01
[36]  9.985172e-01  9.988674e-01  9.986406e-01  9.989819e-01  9.989819e-01
[41]  9.990291e-01  9.992923e-01  9.992174e-01  9.994097e-01  9.993418e-01
[46]  9.990998e-01  9.993968e-01  9.994734e-01  9.994930e-01  9.997740e-01
[51]  9.996360e-01  9.995908e-01  9.997260e-01  9.998648e-01  9.998884e-01
[56]  9.999413e-01  9.998288e-01  9.998860e-01  9.999883e-01
```

```
plot(z)
```

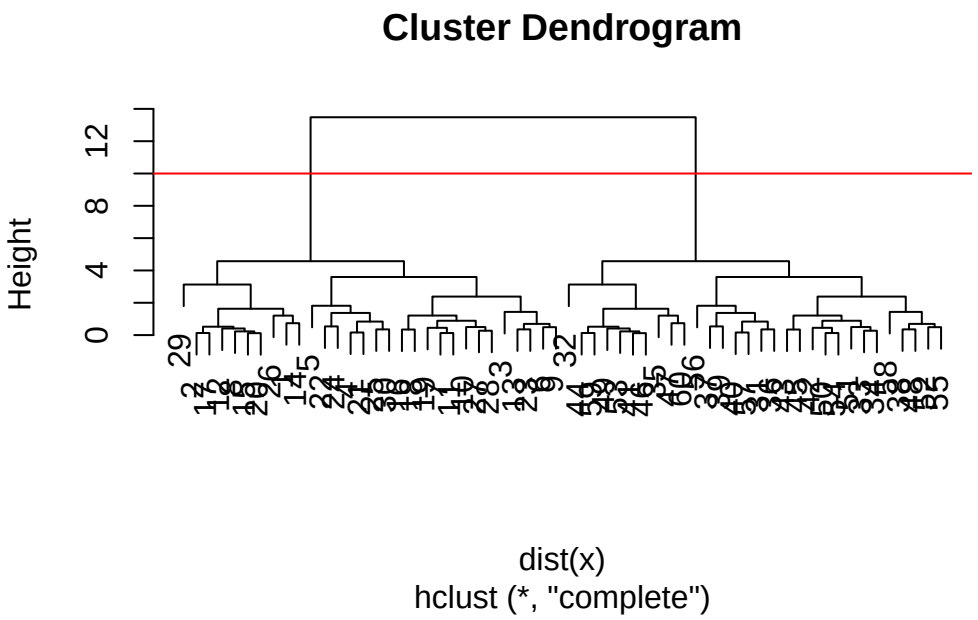


Hierarchical Clustering

This form of clustering aims to reveal the structure in your data by progressively grouping points into a smaller number of clusters over time.

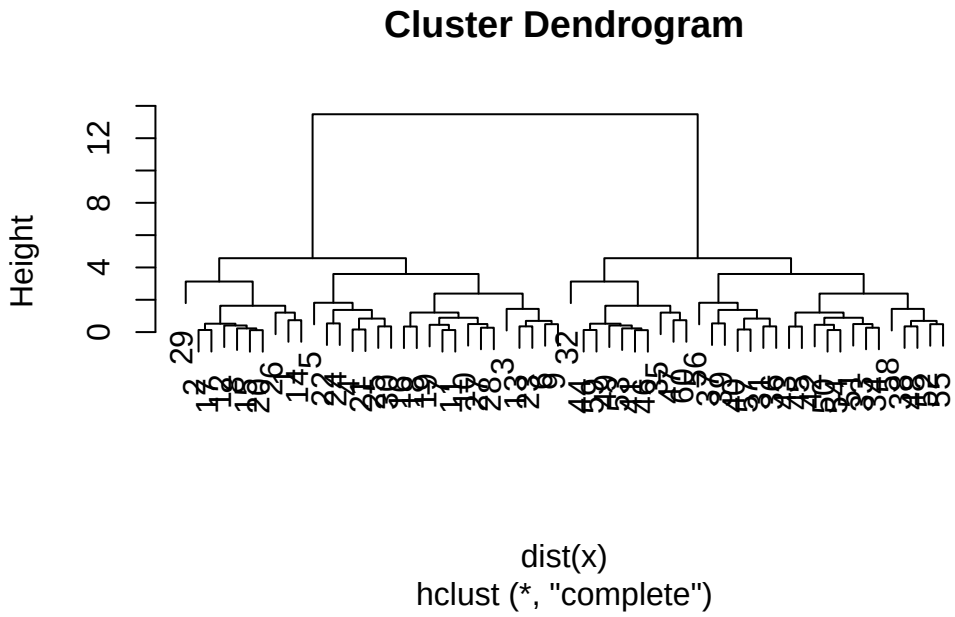
The main function in base R for this is called `hclust()`. This function does not take our input data directly but wants a "distance matrix" that details how dissimilar our input points are to each other.

```
hc <- hclust(dist(x))
plot(hc)
abline(h = 10, col = "red")
```



The print out above is not very useful (unlike kmeans) but thers is a useful `plot()` method

```
plot(hc)
```

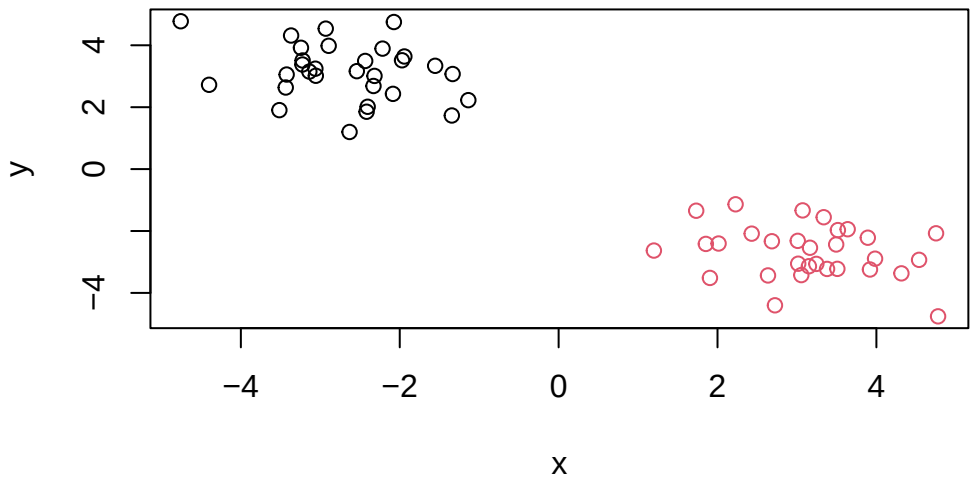


To get my main result (my cluster membership vector) I need to "cut" my tree using function `cutree()`:

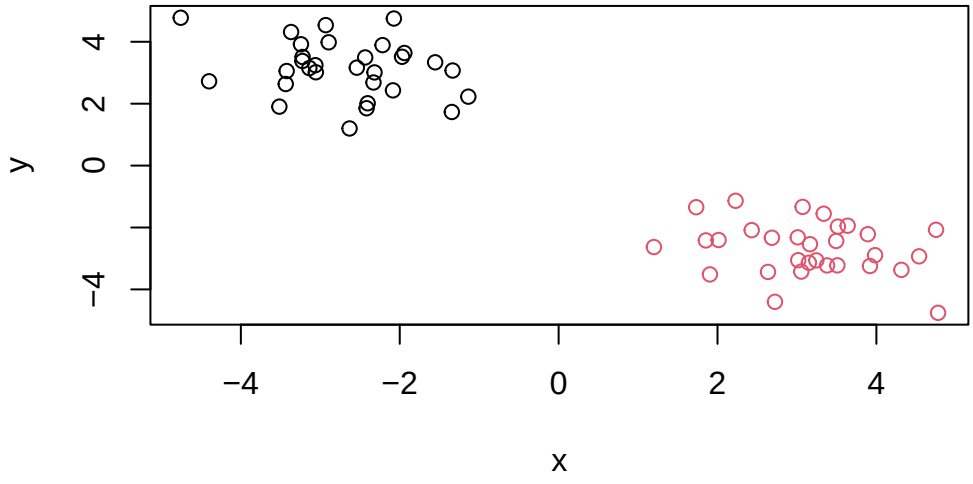
```
grps <- cutree(hc, h=10)
grps
```

```
 [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 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
```

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




```
plot(x, col = cutree(hc, h=5))
```



Principal Component Analysis (PCA)

The goal of PCA is to reduce

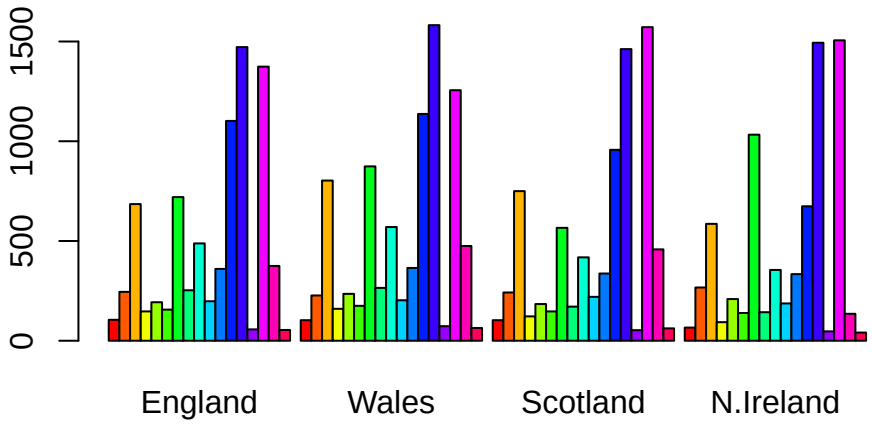
Data import

Read data about crazy eating trends in the UK and N. Ireland

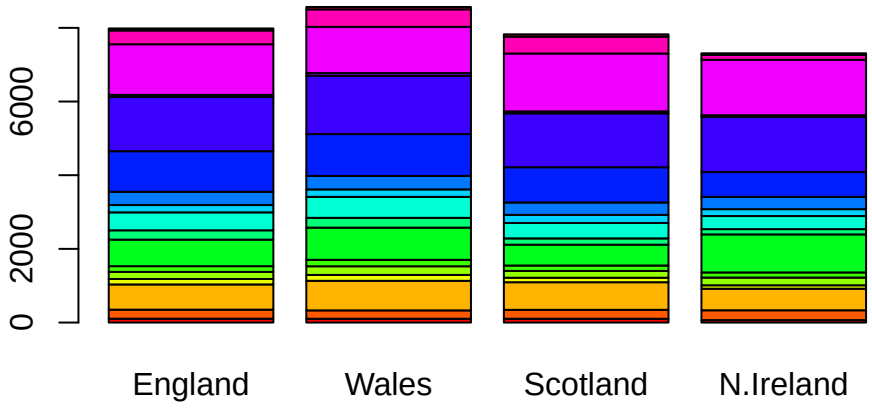
```
url <- "https://tinyurl.com/UK-foods"
x<- read.csv(url,row.names = 1)
head(x)
```

```
              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
```

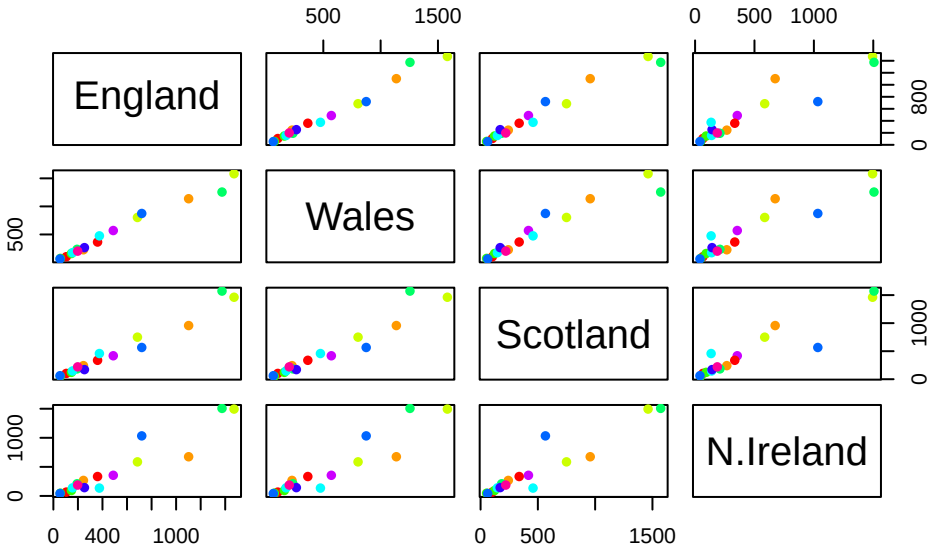
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



```
barplot(as.matrix(x),col=rainbow(nrow(x)))
```




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



If there is a point on the diagonal that means that for that food the countries are very similar in consumption.

The paris plot is useful for small datasets but it can be a lot to interpret and gets intractable for larger datasets.

So PCA to the rescue…

The main dunction to do PCA in base R is called `prcomp()`

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
attributes(pca)
```

```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

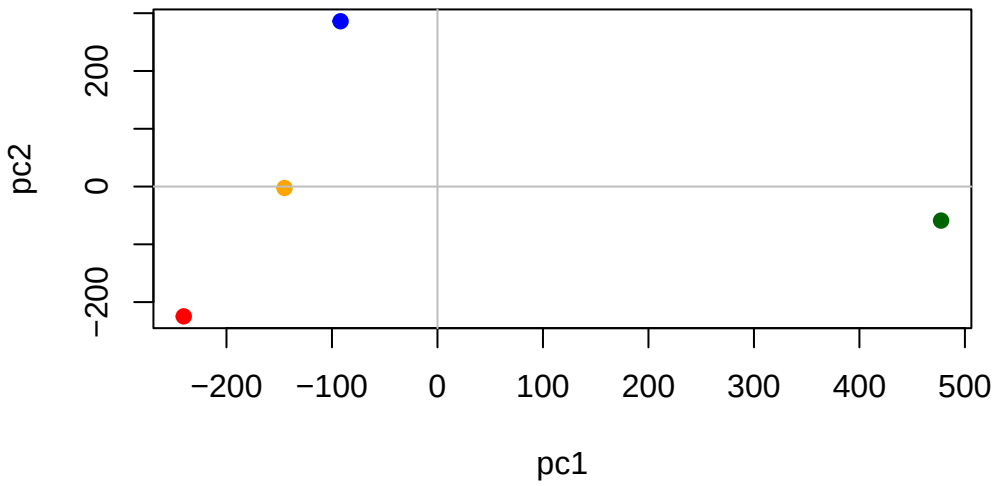
```
#prcomp(x)
```

```
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

A major PCA result viz is called a "PCA plot" (a.k.a a score plot, screen plot, biplot, PC1 v PC2 plot, ordination plot)

```
mycols <- c("orange", "red","blue","darkgreen")
plot(pca$x[,1], pca$x[,2], col = mycols, xlab = "pc1", ylab = "pc2", pch = 19)
abline(h=0, col = "gray")
abline(v=0, col = "gray")
```



Another important output from PCA is called the "loadings' vector or the"rotation' component- this tells us how much the original variables (the foods in this case) contribute to new PCs.

```
pca$rotation
```

	PC1	PC2	PC3	PC4
Cheese	-0.056955380	0.016012850	0.02394295	-0.694538519
Carcass_meat	0.047927628	0.013915823	0.06367111	0.489884628
Other_meat	-0.258916658	-0.015331138	-0.55384854	0.279023718
Fish	-0.084414983	-0.050754947	0.03906481	-0.008483145
Fats_and_oils	-0.005193623	-0.095388656	-0.12522257	0.076097502
Sugars	-0.037620983	-0.043021699	-0.03605745	0.034101334
Fresh_potatoes	0.401402060	-0.715017078	-0.20668248	-0.090972715
Fresh_Veg	-0.151849942	-0.144900268	0.21382237	-0.039901917
Other_Veg	-0.243593729	-0.225450923	-0.05332841	0.016719075
Processed_potatoes	-0.026886233	0.042850761	-0.07364902	0.030125166
Processed_Veg	-0.036488269	-0.045451802	0.05289191	-0.013969507
Fresh_fruit	-0.632640898	-0.177740743	0.40012865	0.184072217
Cereals	-0.047702858	-0.212599678	-0.35884921	0.191926714
Beverages	-0.026187756	-0.030560542	-0.04135860	0.004831876
Soft_drinks	0.232244140	0.555124311	-0.16942648	0.103508492
Alcoholic_drinks	-0.463968168	0.113536523	-0.49858320	-0.316290619

```
Confectionery        -0.029650201  0.005949921 -0.05232164  0.001847469
```

PCA looks to be a super useful method for gaining some insight into high-dimensional data that is difficult to examine in other ways