Class 7: Machine Learning 1

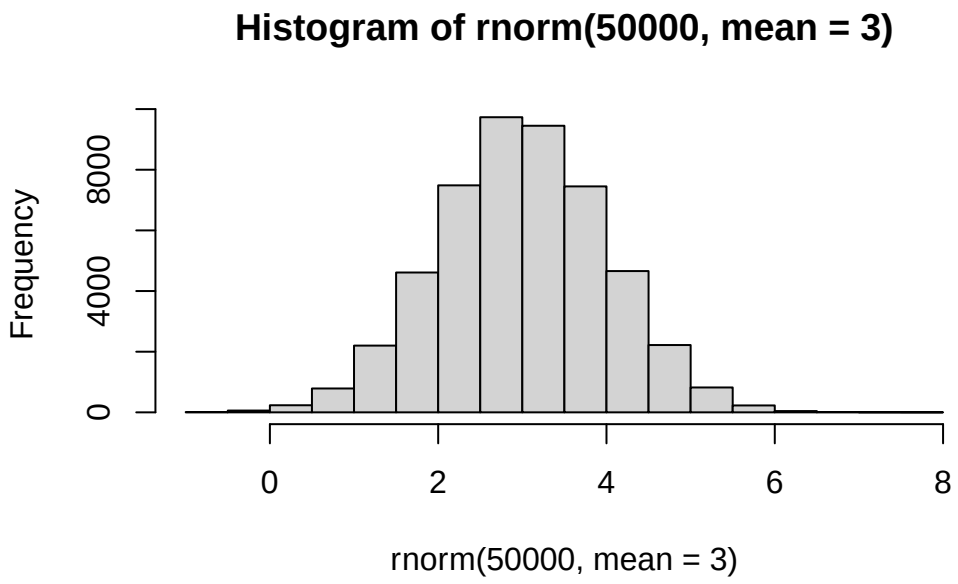
Renny (PID: A98061553)

Today we are going to explore some core machine learning methods. Namely clustering and dimensionality reduction.

Kmeans clustering

The main function for k-means in "base" R is called `kmeans()`. Let's first make up some data to see how kmeans works and to get at the results.

```
#`rnorm()` function creates a set of data.
# Using a histogram should show a normal distribution of numbers.
# Setting a mean will designate where the data centers around.
hist(rnorm(50000, mean = 3))
```



Make a vector named `tmp` with 60 total points half centered at +3, and half centered at -3.

```
tmp <- c(rnorm(30, mean=3), rnorm(30, mean=-3))
#concatenate with c()
tmp
```

```
 [1]  1.466519  3.812420  4.749723  3.397087  3.104786  4.668047  2.964529
 [8]  3.276466  3.241307  3.000117  2.551791  2.760786  4.038456  4.838694
[15]  4.162330  3.969073  1.708370  3.805460  3.799120  3.056776  1.565378
[22]  1.913869  1.927576  2.856482  3.519744  4.227997  3.110882  2.912791
[29]  3.096500  1.730243 -3.817585 -4.625122 -1.529090 -2.169112 -2.433119
[36] -2.566150 -3.445369 -2.579906 -2.640729 -3.026085 -4.356768 -3.180158
[43] -3.960831 -3.071770 -3.192617 -3.239795 -3.339077 -4.380060 -3.012061
[50] -2.683982 -2.421187 -5.066415 -3.741939 -3.362440 -1.427841 -2.829026
[57] -4.676601 -3.153941 -3.960047 -2.375317
```

Reverse tmp using the reverse function `rev()`

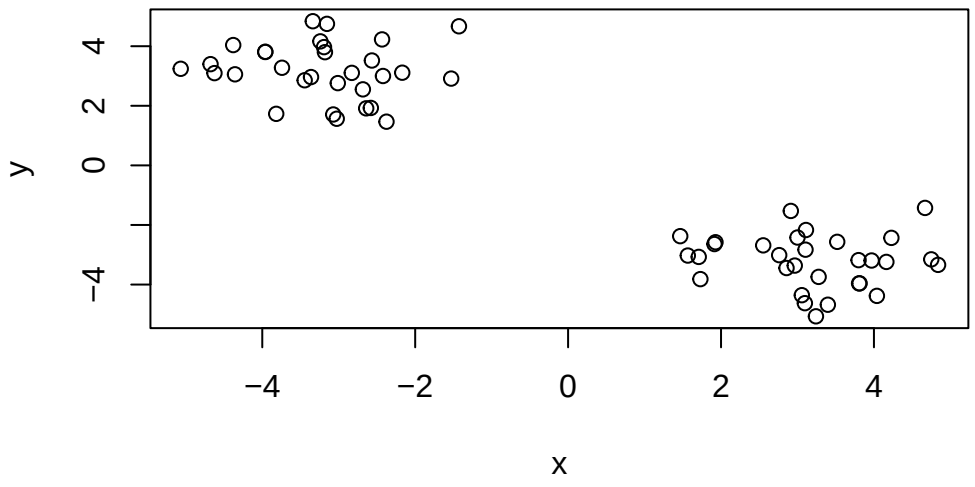
```
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
              x         y
 [1,]  1.466519 -2.375317
 [2,]  3.812420 -3.960047
 [3,]  4.749723 -3.153941
 [4,]  3.397087 -4.676601
 [5,]  3.104786 -2.829026
 [6,]  4.668047 -1.427841
 [7,]  2.964529 -3.362440
 [8,]  3.276466 -3.741939
 [9,]  3.241307 -5.066415
[10,]  3.000117 -2.421187
[11,]  2.551791 -2.683982
[12,]  2.760786 -3.012061
[13,]  4.038456 -4.380060
[14,]  4.838694 -3.339077
[15,]  4.162330 -3.239795
[16,]  3.969073 -3.192617
[17,]  1.708370 -3.071770
[18,]  3.805460 -3.960831
[19,]  3.799120 -3.180158
[20,]  3.056776 -4.356768
[21,]  1.565378 -3.026085
[22,]  1.913869 -2.640729
[23,]  1.927576 -2.579906
[24,]  2.856482 -3.445369
[25,]  3.519744 -2.566150
[26,]  4.227997 -2.433119
[27,]  3.110882 -2.169112
[28,]  2.912791 -1.529090
[29,]  3.096500 -4.625122
[30,]  1.730243 -3.817585
[31,] -3.817585  1.730243
[32,] -4.625122  3.096500
[33,] -1.529090  2.912791
[34,] -2.169112  3.110882
[35,] -2.433119  4.227997
[36,] -2.566150  3.519744
[37,] -3.445369  2.856482
[38,] -2.579906  1.927576
[39,] -2.640729  1.913869
[40,] -3.026085  1.565378
[41,] -4.356768  3.056776
[42,] -3.180158  3.799120
```

```
[43,] -3.960831  3.805460
[44,] -3.071770  1.708370
[45,] -3.192617  3.969073
[46,] -3.239795  4.162330
[47,] -3.339077  4.838694
[48,] -4.380060  4.038456
[49,] -3.012061  2.760786
[50,] -2.683982  2.551791
[51,] -2.421187  3.000117
[52,] -5.066415  3.241307
[53,] -3.741939  3.276466
[54,] -3.362440  2.964529
[55,] -1.427841  4.668047
[56,] -2.829026  3.104786
[57,] -4.676601  3.397087
[58,] -3.153941  4.749723
[59,] -3.960047  3.812420
[60,] -2.375317  1.466519
```

plot(x)

```
plot(x)
```



Try k-means clustering by running `kmeans()` asking for two clusters:

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

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

Cluster means:
          x         y
1 -3.208805  3.174444
2  3.174444 -3.208805

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] 48.86142 48.86142
 (between_SS / total_SS =  92.6 %)

Available components:

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

What is in this result object? Check with `attributes()`.

```
attributes(k)
```

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

$class
[1] "kmeans"
```

Can check individual attributes using function$attribute.

```
k$centers
```

```
          x         y
1 -3.208805  3.174444
2  3.174444 -3.208805
```

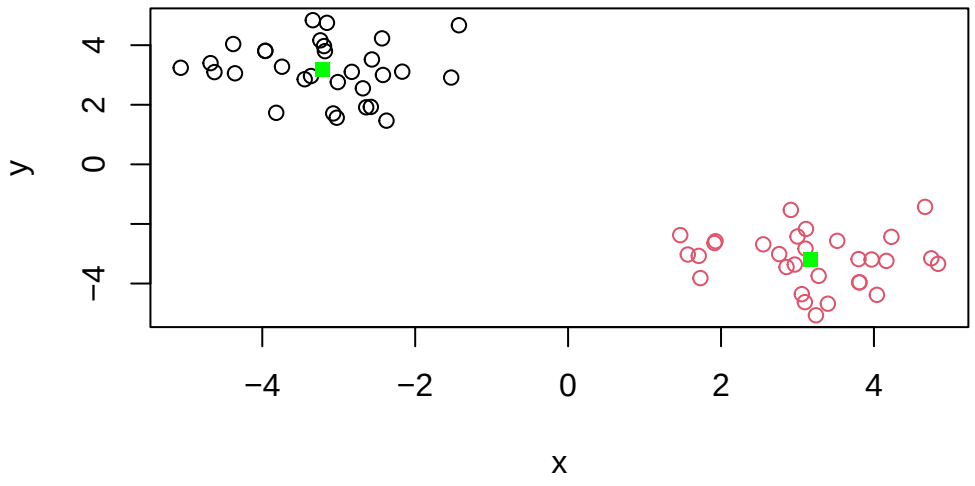
What is my clustering result? I.e. what cluster does each point reside in? Use $cluster to check.

```
k$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. Plot your data **x** showing your clustering results and the center point for each cluster.

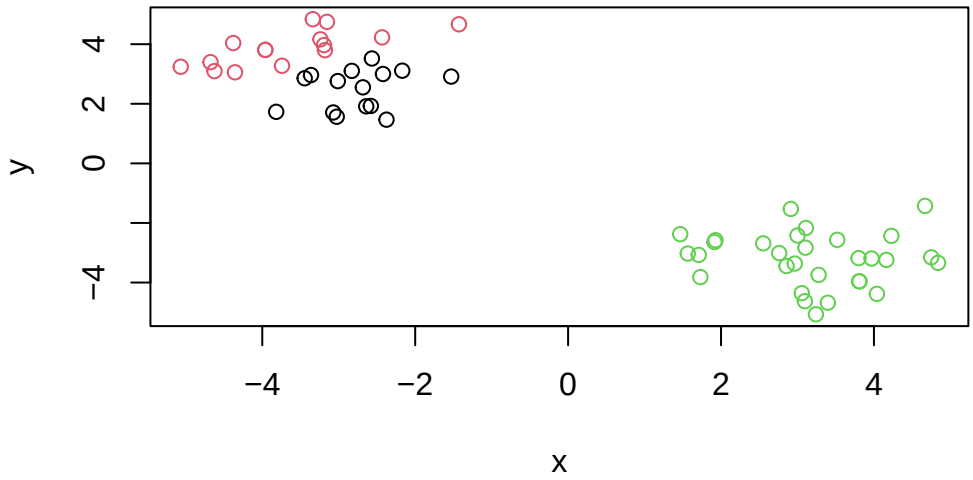
```
#Plot by k$cluster, which shows two distinct clusters.
plot(x, col = k$cluster)
points(k$centers, pch=15, col="green")
```




```
# `points()` function will add points to an existing plot.
# Don't need to add `+`, not necessary in base R; automatically adds code together.
```

Q. Run kmeans and cluster into 3 groups.

```
# Can trick you into thinking the data is a certain way. There will always be an output bu
k3 <- kmeans(x, centers=3, nstart=20)
plot(x, col=k3$cluster)
```




```
# Sum of squares value gets smaller with increasing number of clusters.
k$tot.withinss
```

```
[1] 97.72284
```

```
k3$tot.withinss
```

```
[1] 77.14696
```

The main limitation of k-means clustering (though it is very often employed) is that it imposes a structure on data (i.e. a clustering) that you ask for in the first place, even if that structure is not there.

Hierarchical Clustering

The main function in "base" R for this is called `hclust()`. It wants a distance matrix as input, not the data itself. hclust measures the dissimilarities as produced by distance.

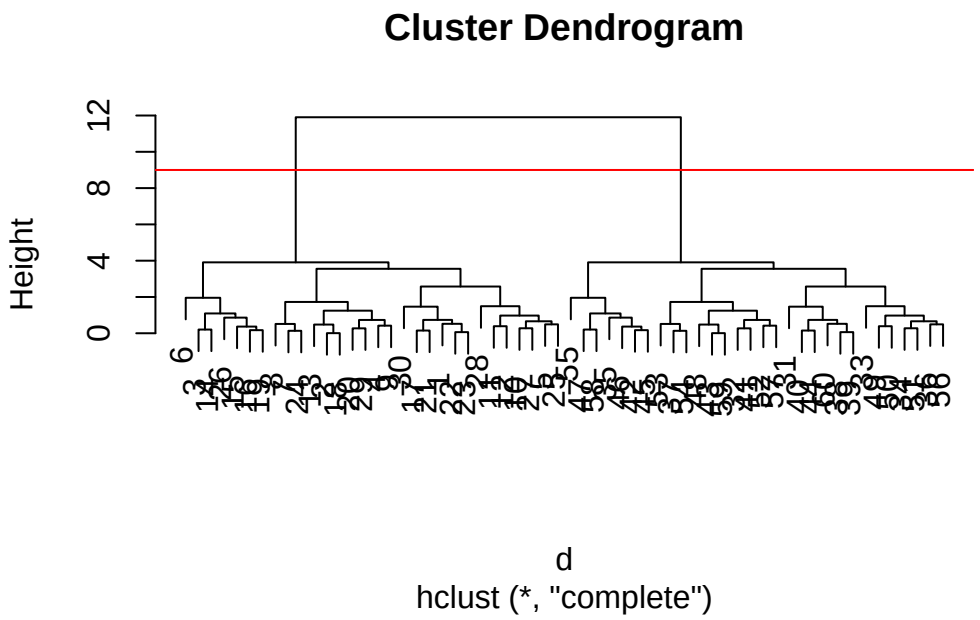
We can calculate a distance matrix in multiple ways, but we will use the `dist()` function. Distance is by default measured as Euclidean distance).

`hclust(dist(x))`

```
#`dist()` takes the distance from all points, to build up a table.
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

```
plot(hc)
# Note that there are no relationships between the labels.
# From this bottom-up approach, the thing that matters is the "crossbar"; it's this part t
abline(h=9, col="red")
```



To get the cluster membership vector (equivalent of k$cluster for k-means) we need to "cut" the tree at a given height that we choose. The function is called `cutree()`.

```
cutree(hc, h=9)
```

```
 [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
```

When choosing not to pick a certain height, and want to instead cut at values when there are certain clusters, then indicate that with `k`.

```
cutree(hc, k=4)
```

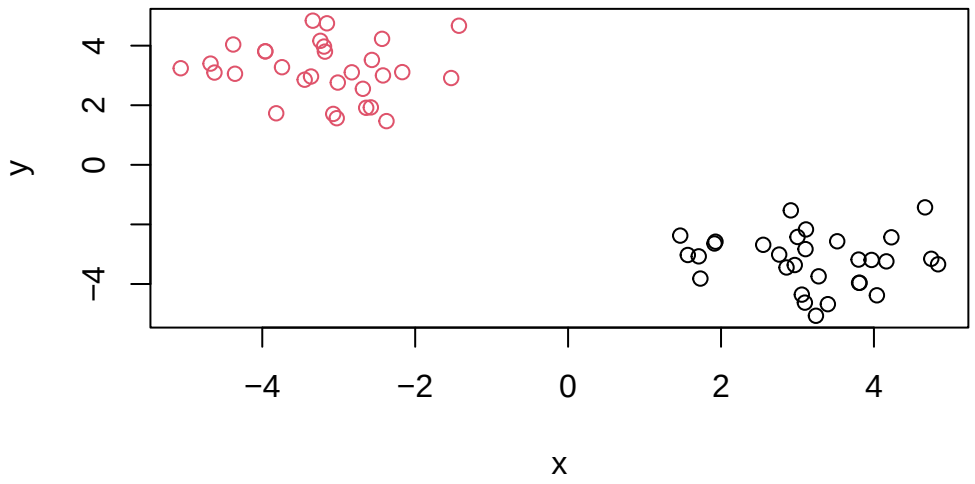
```
 [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 3 3 3 3 4 3 3 3
[39] 3 3 3 4 3 3 4 4 4 3 3 3 3 3 3 3 4 3 3 4 3 3
```

```
grps <- cutree(hc, k=2)
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 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
```

Q. Plot our data ('x') colored by our hclust result.

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



Principal Component Analysis

We will start with PCA of a tiny dataset of food data.

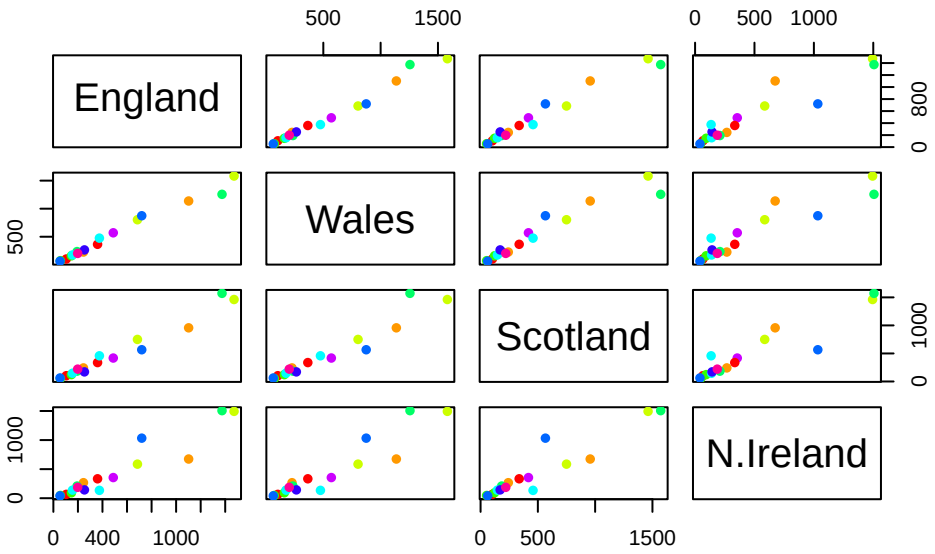
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
x
```

```
                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
Fresh_potatoes	720	874	566	1033
Fresh_Veg	253	265	171	143
Other_Veg	488	570	418	355
Processed_potatoes	198	203	220	187
Processed_Veg	360	365	337	334
Fresh_fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft_drinks	1374	1256	1572	1506
Alcoholic_drinks	375	475	458	135
Confectionery	54	64	62	41

One useful plot in this case (because we only have 4 countries to look across) is a so-called "pairs plot".

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



Enter PCA

The main function to do PCA in "base" R is called `prcomp()`.

It wants our variables as the columns (e.g. the foods in this case) and the countries as the rows. The data will need to be transposed.

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 2.921e-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)
```

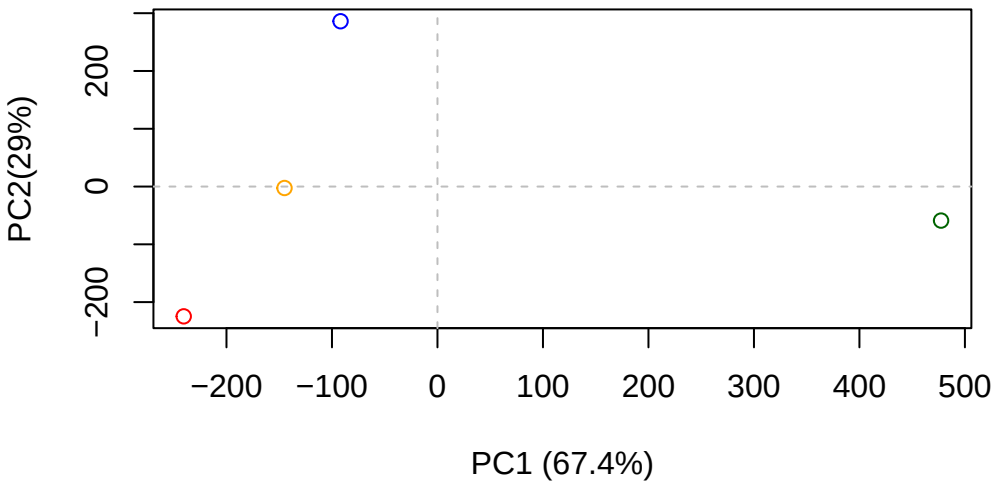
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -9.152022e-15
Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

```
#Gives the values for the new values
```

```
plot(pca$x[,1], pca$x[,2], xlab="PC1 (67.4%)", ylab="PC2(29%)",
     col = c("orange", "red", "blue", "darkgreen"))
abline(h=0, col="gray", lty=2)
abline(v=0, col="gray", lty=2)
```


PC2(29%)
200
0
−200
−200
−100
0
100
200
300
400
500
PC1 (67.4%)