Class 7: Machine Learning 1

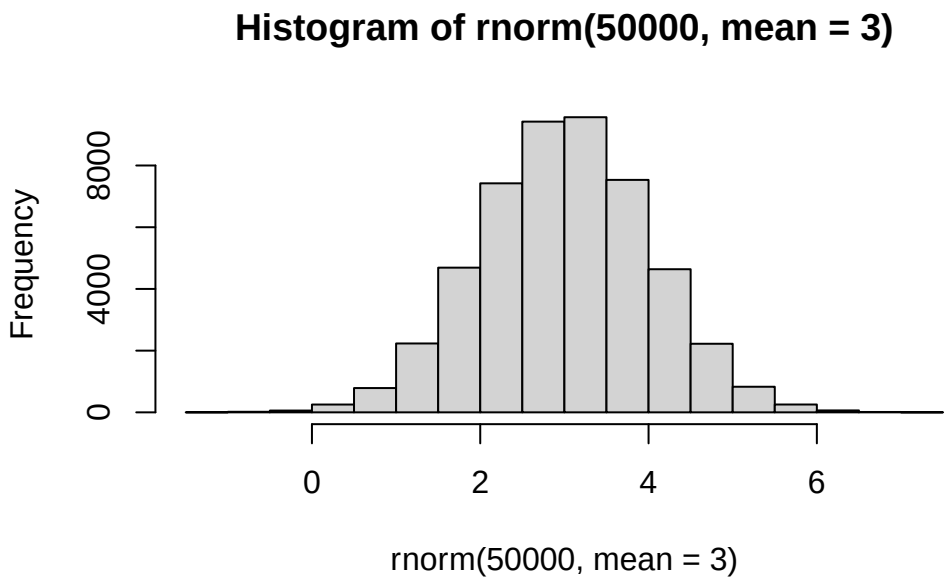
Renny (PID: A98061553)

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]  3.2571589  3.9042588  2.6805335  4.3402606  3.9390494  2.9348342
 [7]  1.1468530  2.3844804  1.7005579  0.8843498  2.4410831  3.1731559
[13]  4.3220970  2.6056600  2.3171993  2.1106479  1.7259074  3.1610465
[19]  3.4810843  3.7546315  3.5313461  2.3443775  1.8584787  2.3008404
[25]  4.1104899  3.0257942  2.2993113  3.1007131  4.2517645  2.4116182
[31] -1.9651286 -2.5656801 -3.6736183 -0.9291021 -2.4849853 -2.5410512
[37] -4.3114206 -4.4137057 -3.9002630 -4.1873300 -1.5417322 -3.3477878
[43] -3.1482239 -2.9932951 -2.7972842 -4.0816037 -2.1848624 -3.0463050
[49] -1.7481727 -2.6545545 -3.2688337 -3.3021171 -0.4646766 -2.1243470
[55] -2.9833890 -5.4321167 -5.1590235 -2.8225929 -1.3773854 -2.8641253
```

Reverse tmp using the reverse function `rev()`

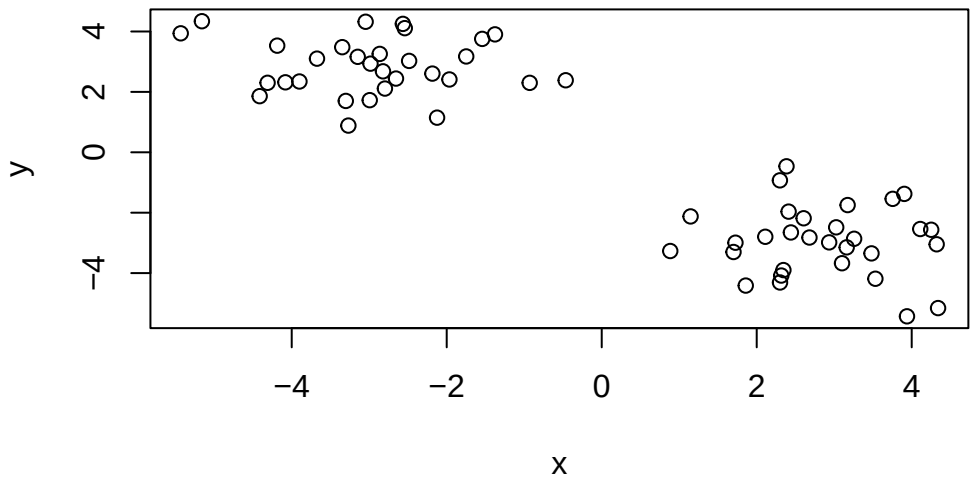
```
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
              x          y
 [1,]  3.2571589 -2.8641253
 [2,]  3.9042588 -1.3773854
 [3,]  2.6805335 -2.8225929
 [4,]  4.3402606 -5.1590235
 [5,]  3.9390494 -5.4321167
 [6,]  2.9348342 -2.9833890
 [7,]  1.1468530 -2.1243470
 [8,]  2.3844804 -0.4646766
 [9,]  1.7005579 -3.3021171
[10,]  0.8843498 -3.2688337
[11,]  2.4410831 -2.6545545
[12,]  3.1731559 -1.7481727
[13,]  4.3220970 -3.0463050
[14,]  2.6056600 -2.1848624
[15,]  2.3171993 -4.0816037
[16,]  2.1106479 -2.7972842
[17,]  1.7259074 -2.9932951
[18,]  3.1610465 -3.1482239
[19,]  3.4810843 -3.3477878
[20,]  3.7546315 -1.5417322
[21,]  3.5313461 -4.1873300
[22,]  2.3443775 -3.9002630
[23,]  1.8584787 -4.4137057
[24,]  2.3008404 -4.3114206
[25,]  4.1104899 -2.5410512
[26,]  3.0257942 -2.4849853
[27,]  2.2993113 -0.9291021
[28,]  3.1007131 -3.6736183
[29,]  4.2517645 -2.5656801
[30,]  2.4116182 -1.9651286
[31,] -1.9651286  2.4116182
[32,] -2.5656801  4.2517645
[33,] -3.6736183  3.1007131
[34,] -0.9291021  2.2993113
[35,] -2.4849853  3.0257942
[36,] -2.5410512  4.1104899
[37,] -4.3114206  2.3008404
[38,] -4.4137057  1.8584787
[39,] -3.9002630  2.3443775
[40,] -4.1873300  3.5313461
[41,] -1.5417322  3.7546315
[42,] -3.3477878  3.4810843
```

```
[43,] -3.1482239  3.1610465
[44,] -2.9932951  1.7259074
[45,] -2.7972842  2.1106479
[46,] -4.0816037  2.3171993
[47,] -2.1848624  2.6056600
[48,] -3.0463050  4.3220970
[49,] -1.7481727  3.1731559
[50,] -2.6545545  2.4410831
[51,] -3.2688337  0.8843498
[52,] -3.3021171  1.7005579
[53,] -0.4646766  2.3844804
[54,] -2.1243470  1.1468530
[55,] -2.9833890  2.9348342
[56,] -5.4321167  3.9390494
[57,] -5.1590235  4.3402606
[58,] -2.8225929  2.6805335
[59,] -1.3773854  3.9042588
[60,] -2.8641253  3.2571589
```

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  2.849986 -2.943824
2 -2.943824  2.849986

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 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

Within cluster sum of squares by cluster:
[1] 63.84903 63.84903
 (between_SS / total_SS =  88.7 %)

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  2.849986 -2.943824
2 -2.943824  2.849986
```

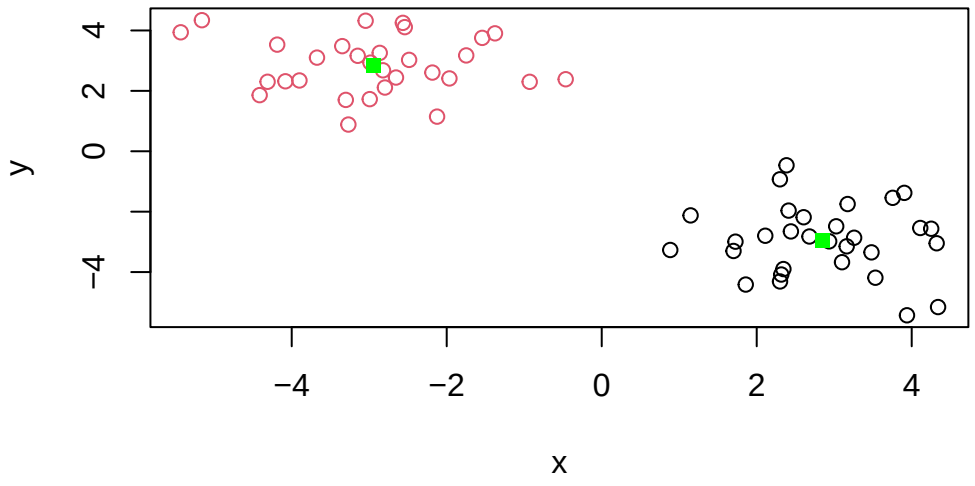
What is my clustering result? I.e. what cluster does each point reside in? Use $cluster to check.

```
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 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
```

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

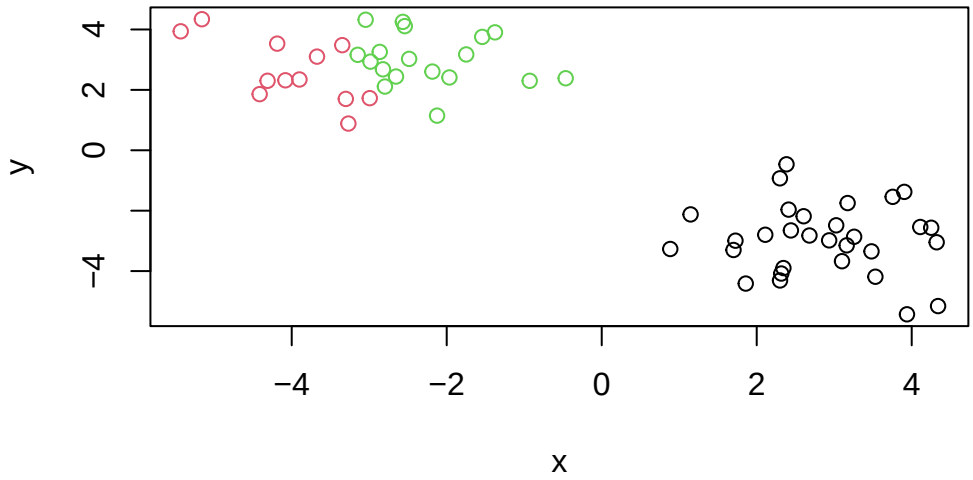
```
#Plot by k$cluster, which shows two distinct clusters.
plot(x, col = k$cluster)
points(k$centers, pch=15, col="green")
```




```
# `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] 127.6981
```

```
k3$tot.withinss
```

```
[1] 104.1425
```

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.

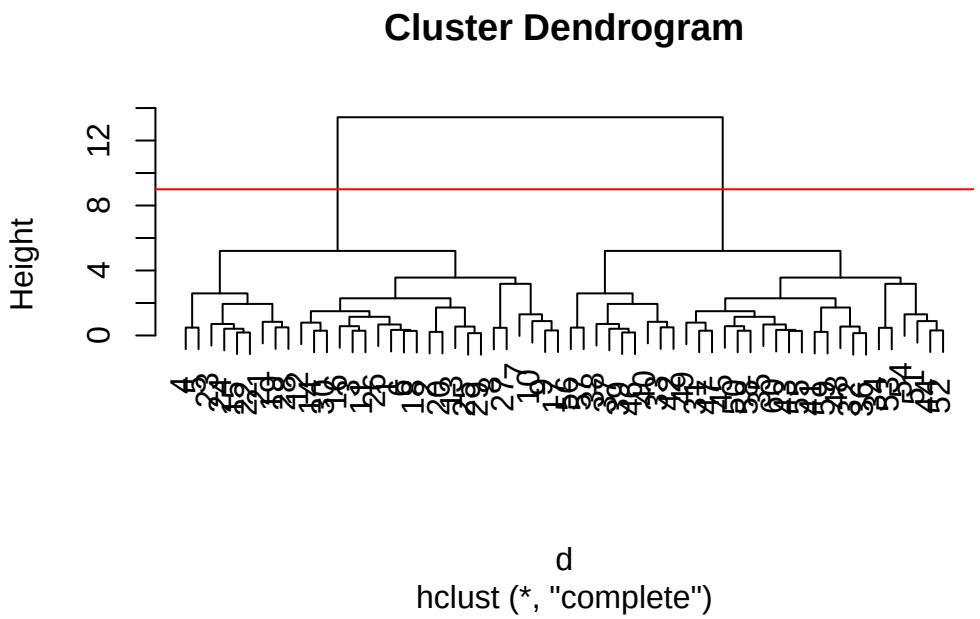
We can calculate a distance matrix in multiple ways, but we will use the `dist()` function. Distance is by default measured as Euclidean distance).

`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)
```

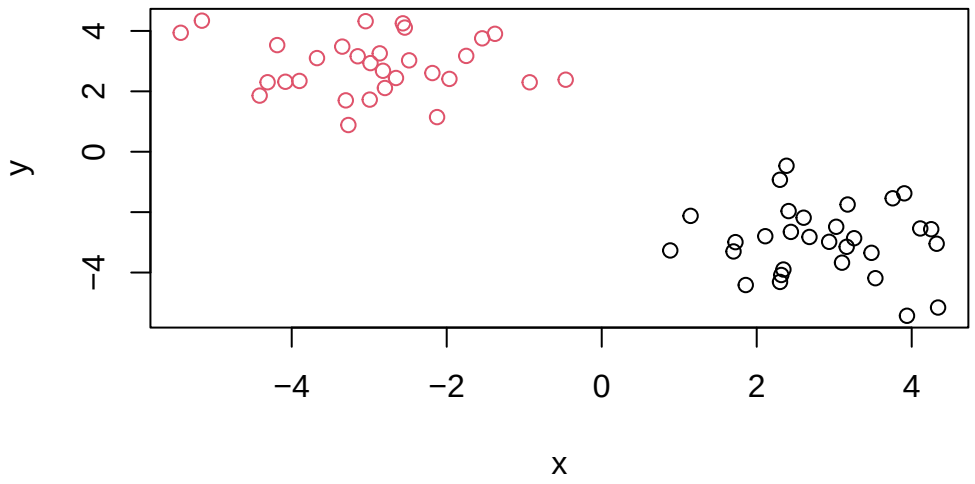
```
 [1] 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 2 2 2 1 1 1 2 1 1 3 3 4 3 3 3 4 4
[39] 4 4 3 4 3 3 3 4 3 3 3 3 3 3 3 3 3 4 4 3 3 3
```

```
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 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.

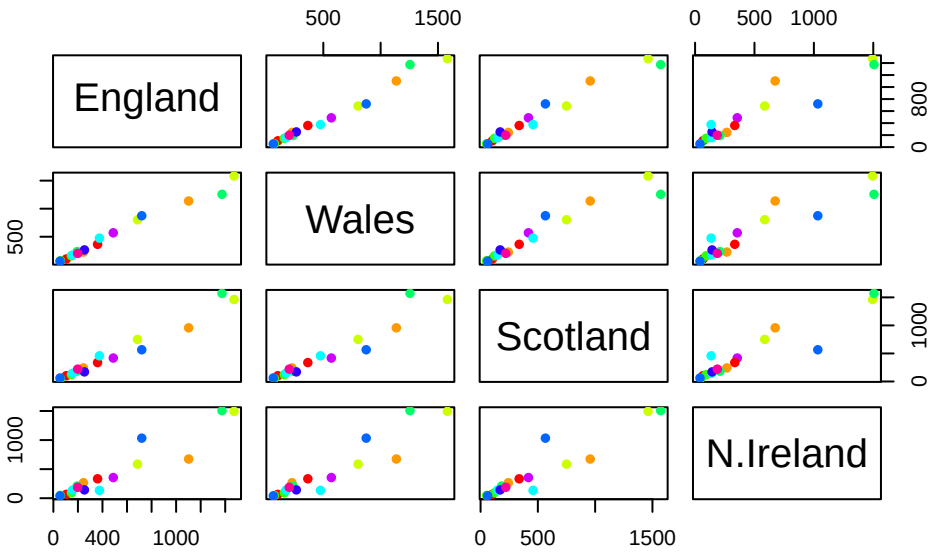
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
x
```

```
                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)
```

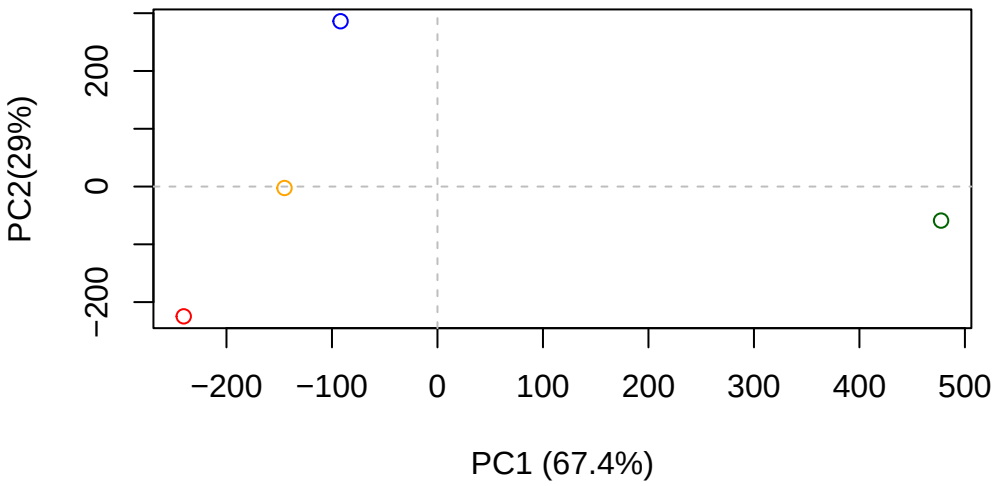
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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%)