# Class 7: Machine Learning

Taylor F. (A59010460)

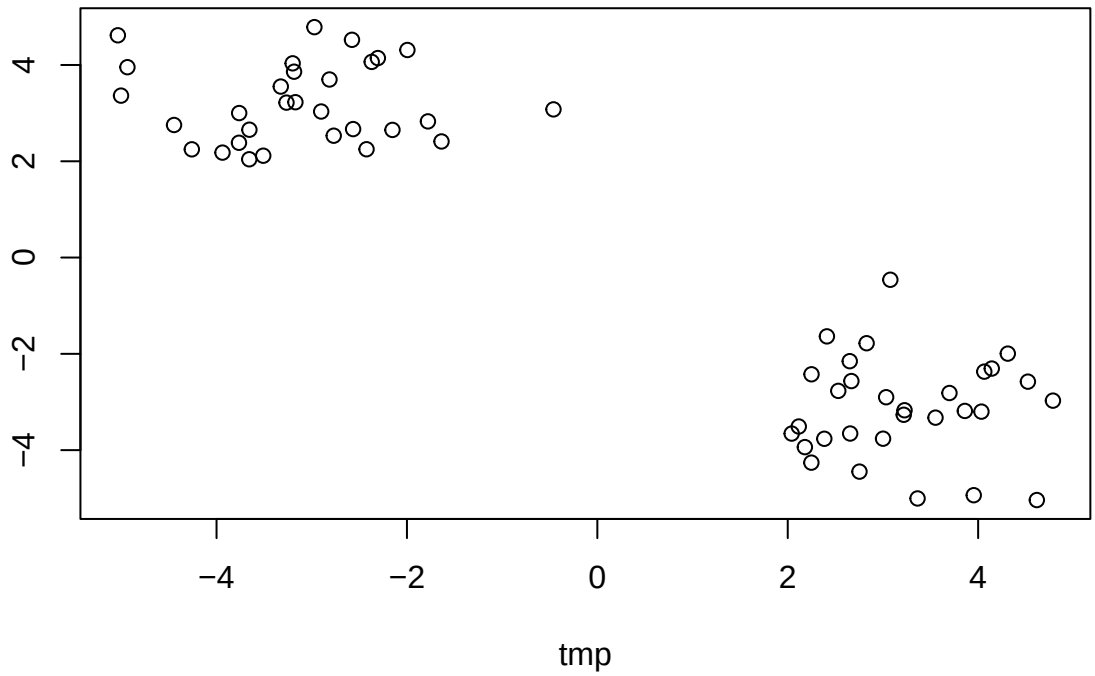
2/9/2022

kmeans()

## Generate some example data for clustering

```
# rnorm(n, mean = 0, sd = 1), where n is the number of observations, mean is where the numbers will "ce

# generate a set of data where 30 numbers are generated that center around -3 and 30 numbers that are c
tmp <- c(rnorm(30,-3), rnorm(30,3))
x <- cbind(tmp, rev(tmp))
plot(x)
```




We are now going to perform k-means clustering on the data

```r
k <- kmeans(x, centers = 4, nstart = 10)
k
```

```
## K-means clustering with 4 clusters of sizes 19, 19, 11, 11
##
## Cluster means:
##         tmp
## 1 -2.519533  3.414090
## 2  3.414090 -2.519533
## 3  2.846644 -4.178732
## 4 -4.178732  2.846644
##
## Clustering vector:
##  [1] 1 1 1 4 1 1 1 4 1 1 1 1 4 1 4 1 1 4 4 4 1 4 1 1 1 4 1 4 4 1 2 3 3 2 3 2 2 2
## [39] 3 2 3 3 3 2 2 3 2 3 2 2 2 2 3 2 2 2 3 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 19.60203 19.60203 10.35989 10.35989
##  (between_SS / total_SS =  95.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

How many points in each cluster (i.e. the size of the cluster)

```r
k$size
```

```
## [1] 19 19 11 11
```

```r
#any of the "available components" from the kmeans() output can be analyzed in this way
```

How many centroids in each cluster?

```r
k$centers
```

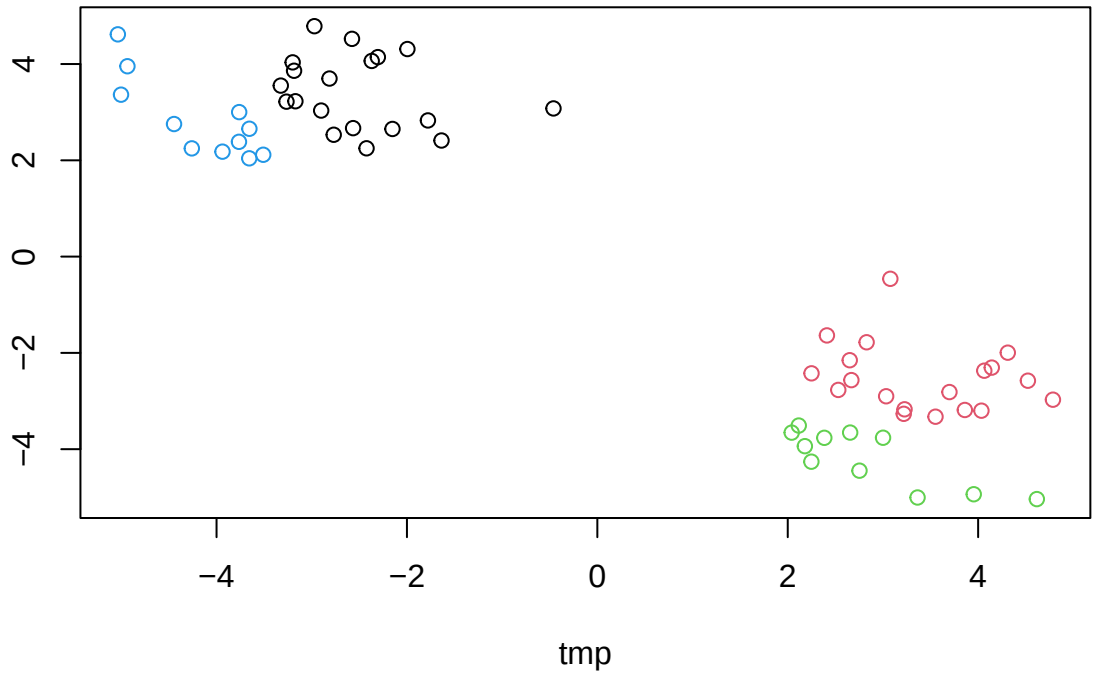
```
##         tmp
## 1 -2.519533  3.414090
## 2  3.414090 -2.519533
## 3  2.846644 -4.178732
## 4 -4.178732  2.846644
```

Plot our results

```r
k$cluster
```

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

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




## Recall a very useful feature of R, called recycling

```
y <- 1:5
cbind(y, "red")
```

```
##      y
## [1,] "1" "red"
## [2,] "2" "red"
## [3,] "3" "red"
## [4,] "4" "red"
## [5,] "5" "red"
```

```
#cbind will assume that, because your data set contains 5 items, you want 5 "reds" to be bound to the d
cbind(y, c("red", "blue"))
```

```
## Warning in cbind(y, c("red", "blue")): number of rows of result is not a
## multiple of vector length (arg 2)
```

```
##      y
```

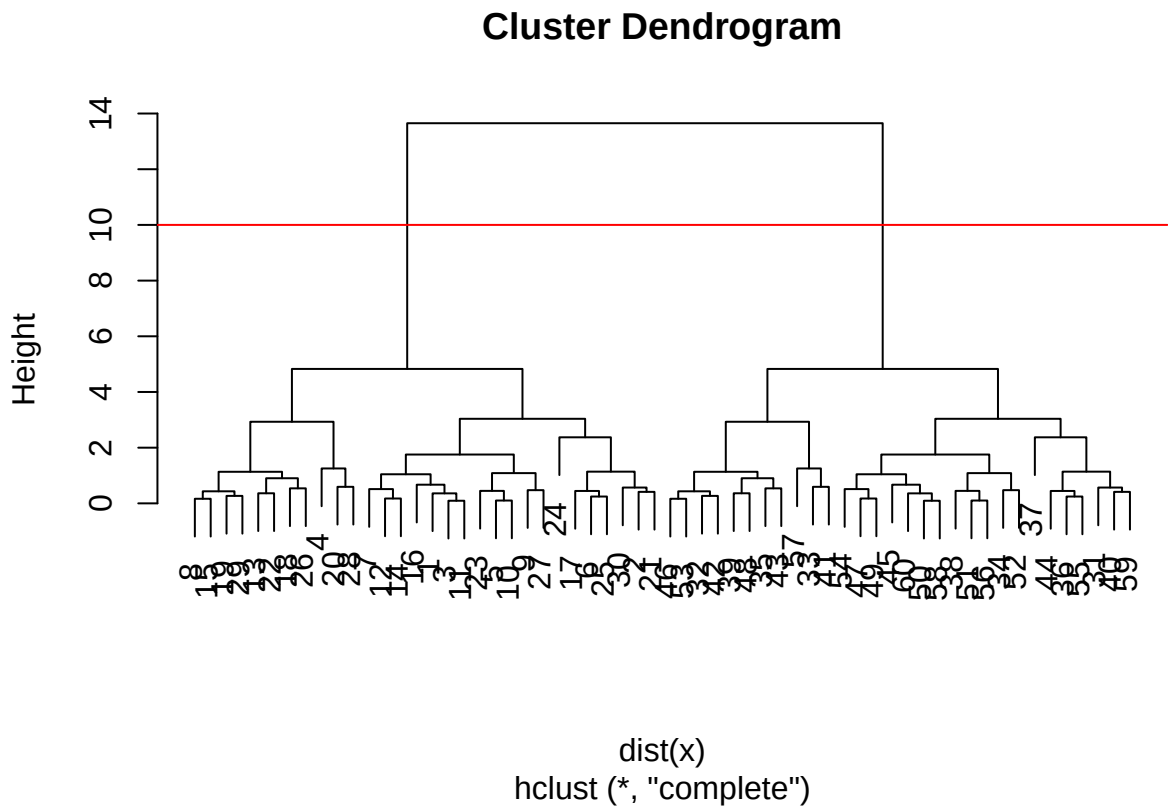
```
## [1,] "1" "red"
## [2,] "2" "blue"
## [3,] "3" "red"
## [4,] "4" "blue"
## [5,] "5" "red"
```

```
#this remains true if you ask it to append red and blue to the data set, but it will spit back an error
```

So we must be careful that when we are coloring, or doing anything that is dependent on lengths of vectors, we are aware of the lengths

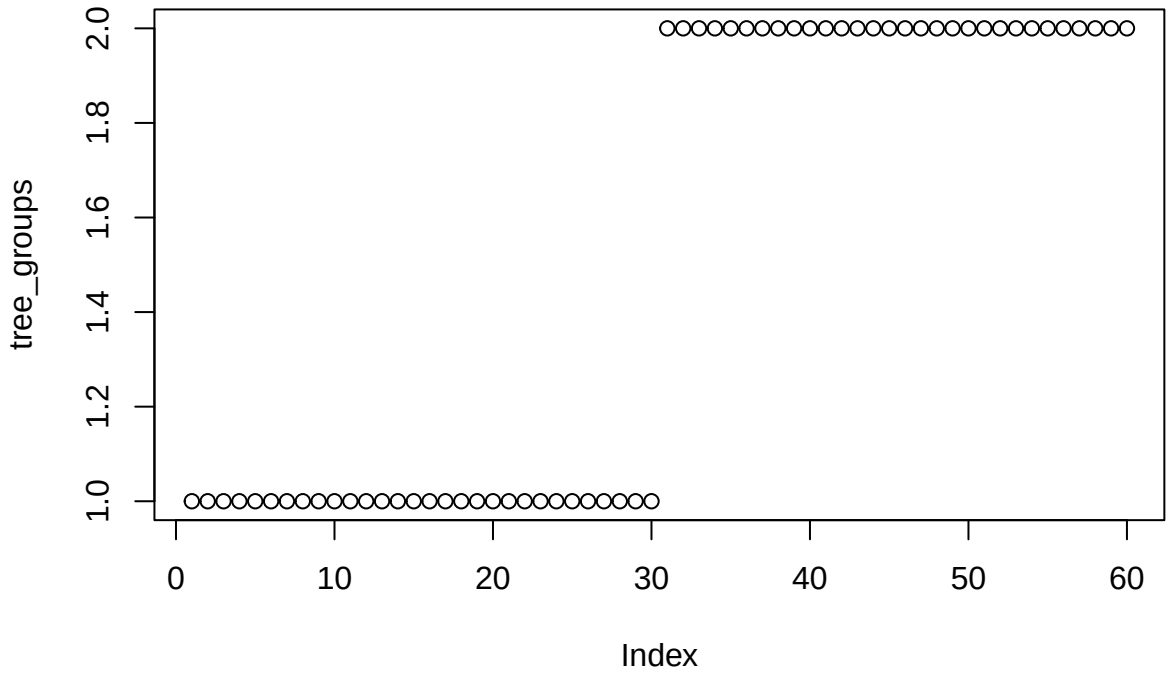
## Hierarchical Clustering

```
#must give hclust() a distance matrix as input, not the raw data
#there is a custom plot method for hclus objects that results in a cluster dendrogram
hc <- hclust(dist(x))
#hc
plot(hc)
abline(h=10, col="red")
```




Now we are going to "cut" the tree to get our two cluster membership vectors. The function to do this is called the "cutree()" function.

```
#takes the input of an hclust function and a cutoff height
tree_groups <- cutree(hc, h = 10)
plot(tree_groups)
```




# Principal Component Analysis

Principal Component Analysis (PCA) is a very useful method for the analysis of large, multidimensional data sets

## PCA of UK Food Data

Below, we will read/import some data about the food consumption habits of people in the UK

```
url <- "https://tinyurl.com/UK-foods"
food_set <- read.csv(url, row.names=1)
#renames the first column after the row names
```

Let's look at food_set

```
head(food_set)
```

```
##                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
```

How many rows and columns are in food_set

```
nrow(food_set)
```

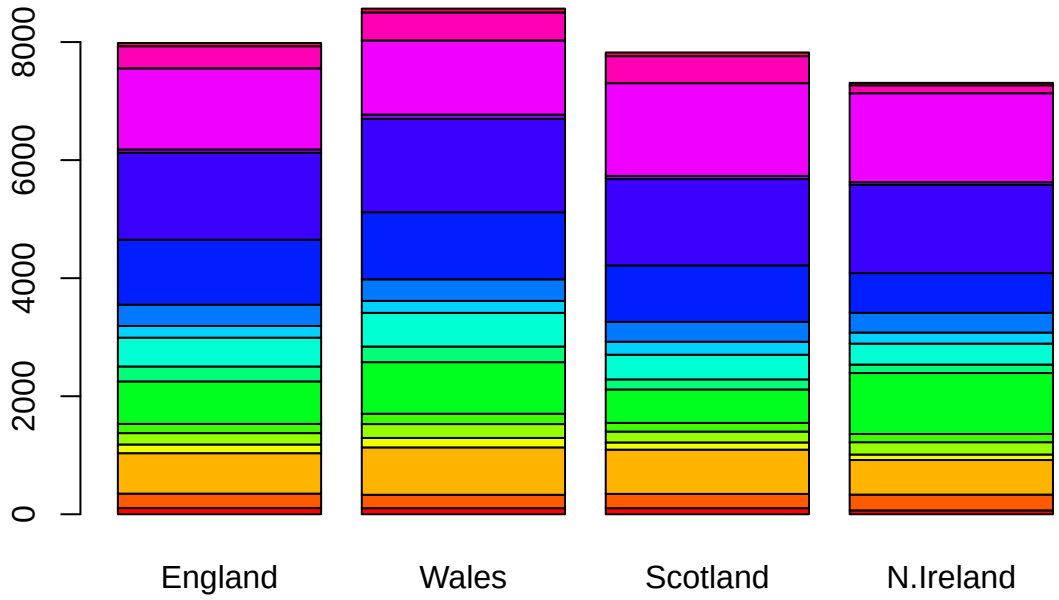
```
## [1] 17
```

```
ncol(food_set)
```

```
## [1] 4
```

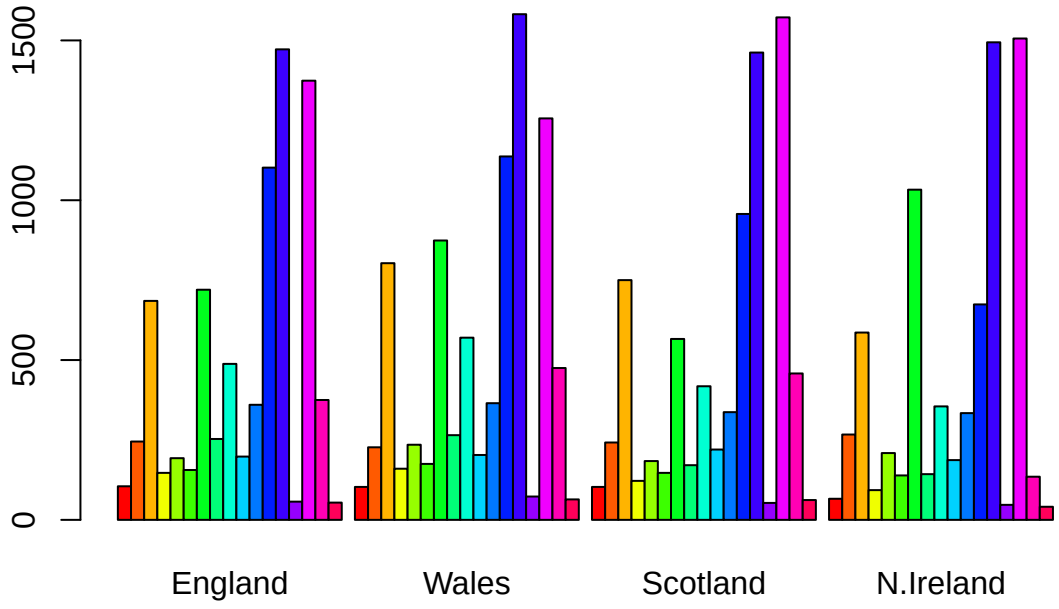
We could plot this with a basic barplot

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



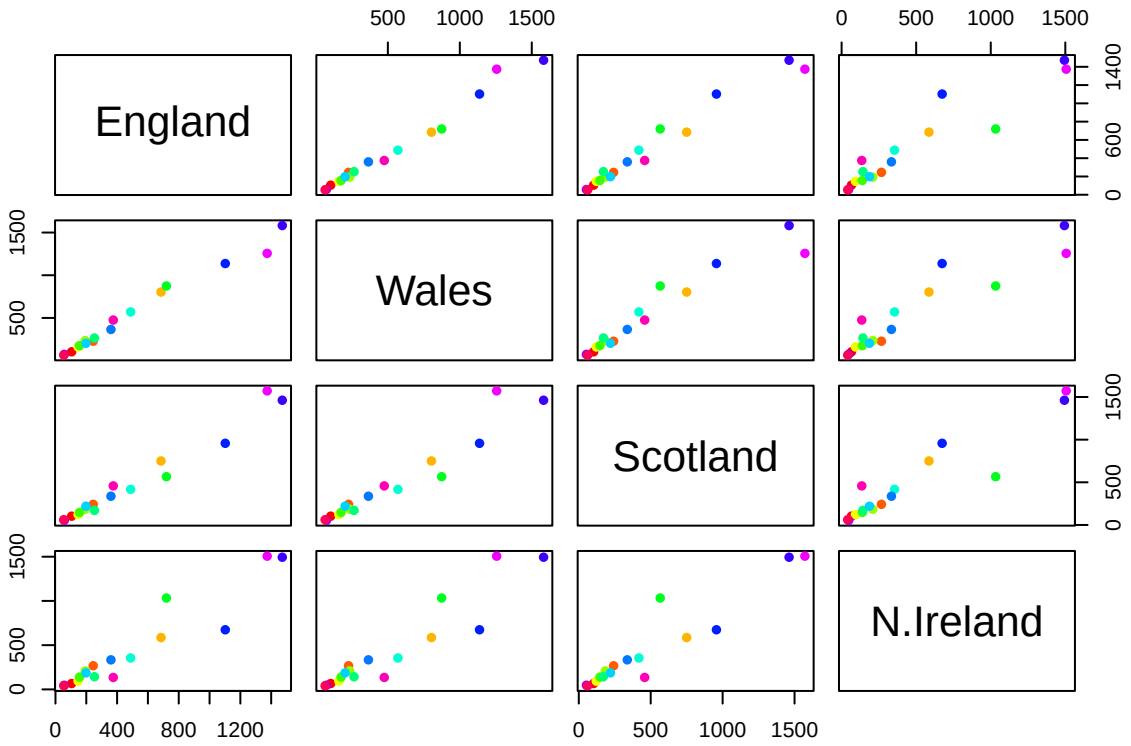

We can turn off the stacking by defining the “beside” criteria

```
barplot(as.matrix(food_set), col = rainbow(nrow(food_set)), beside = TRUE)
```




One plot that might be of more use is called the "pairs plot"

```
pairs(food_set, col=rainbow(nrow(food_set)), pch=16)
```




## PCA to the Rescue

What does PCA tell us about this dataset?

```
#prcomp() is the main PCA function in base R
pca <- prcomp(t(food_set))
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
```

```
#We can see that PC1 makes up most of the sample variance, and PC1 and PC2 together explain almost all
```

### PCA Plot

A plot of PC1 vs. PC2 is often called a PCA plot or a "score plot"

```
attributes(pca)
```

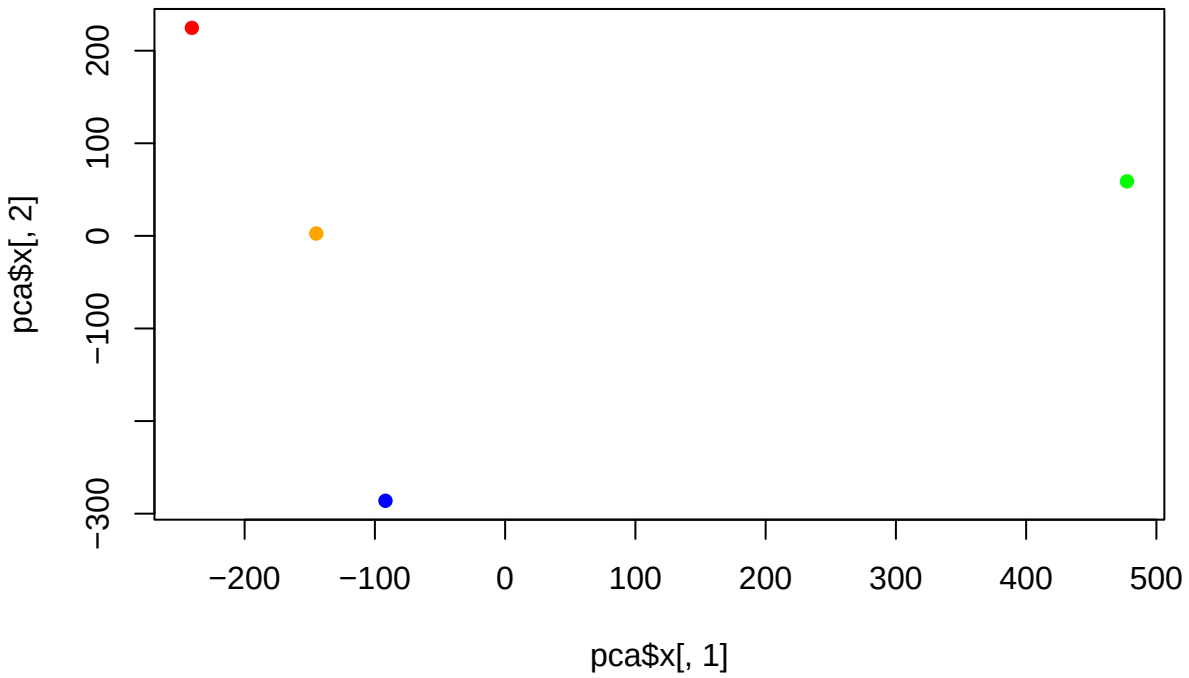
```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

To generate a "score plot" we ant the pca$x component of the resulting object

```
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
```

```
plot(pca$x[,1], pca$x[,2], col = c("orange", "red", "blue","green"), pch = 16)
```




The loadings (a.k.a. weights) tell us how the original variables contribute to the PCs

```
pca$rotation
```

```
##                           PC1          PC2         PC3          PC4
## Cheese            -0.056955380 -0.016012850 -0.02394295 -0.691718038
```

```
## Carcass_meat          0.047927628 -0.013915823 -0.06367111  0.635384915
## Other_meat           -0.258916658  0.015331138  0.55384854  0.198175921
## Fish                 -0.084414983  0.050754947 -0.03906481 -0.015824630
## Fats_and_oils        -0.005193623  0.095388656  0.12522257  0.052347444
## Sugars               -0.037620983  0.043021699  0.03605745  0.014481347
## Fresh_potatoes        0.401402060  0.715017078  0.20668248 -0.151706089
## Fresh_Veg            -0.151849942  0.144900268 -0.21382237  0.056182433
## Other_Veg            -0.243593729  0.225450923  0.05332841 -0.080722623
## Processed_potatoes   -0.026886233 -0.042850761  0.07364902 -0.022618707
## Processed_Veg        -0.036488269  0.045451802 -0.05289191  0.009235001
## Fresh_fruit          -0.632640898  0.177740743 -0.40012865 -0.021899087
## Cereals              -0.047702858  0.212599678  0.35884921  0.084667257
## Beverages            -0.026187756  0.030560542  0.04135860 -0.011880823
## Soft_drinks           0.232244140 -0.555124311  0.16942648 -0.144367046
## Alcoholic_drinks     -0.463968168 -0.113536523  0.49858320 -0.115797605
## Confectionery        -0.029650201 -0.005949921  0.05232164 -0.003695024
```

```r
barplot(pca$rotation[,1], las = 2)
```