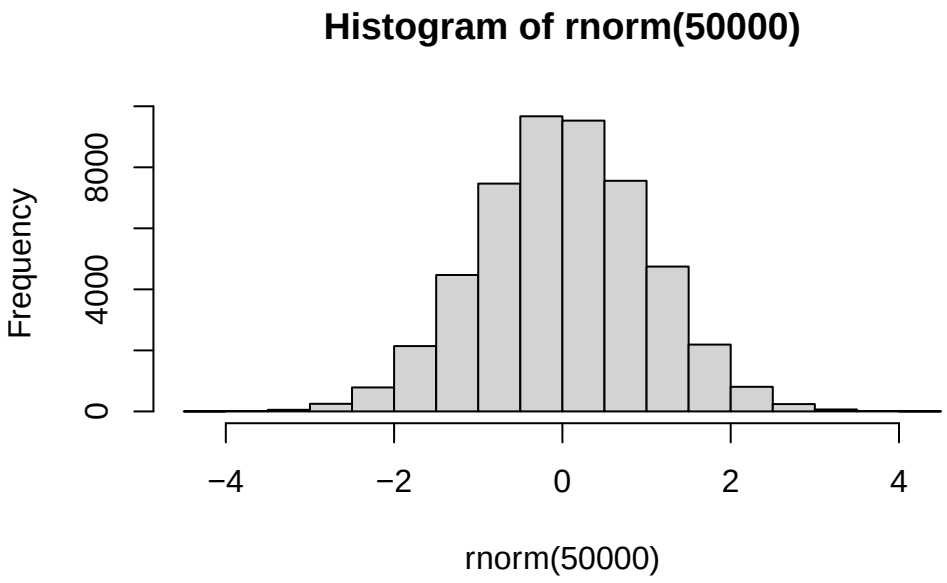
Class 7: Machine Learning

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

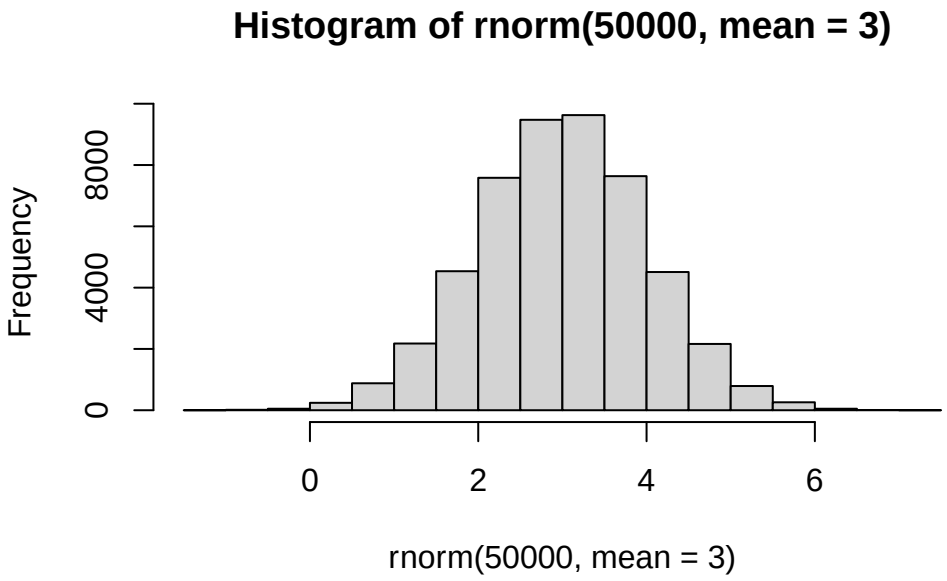
Kmeans clustering

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 aat the results.

```
hist( rnorm(50000) )
```



```
hist( rnorm(50000, mean =3) )
```



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

```
a <- rnorm(30, mean=3)
b <- rnorm(30, mean=-3)
c<-c(a,b)
c
```

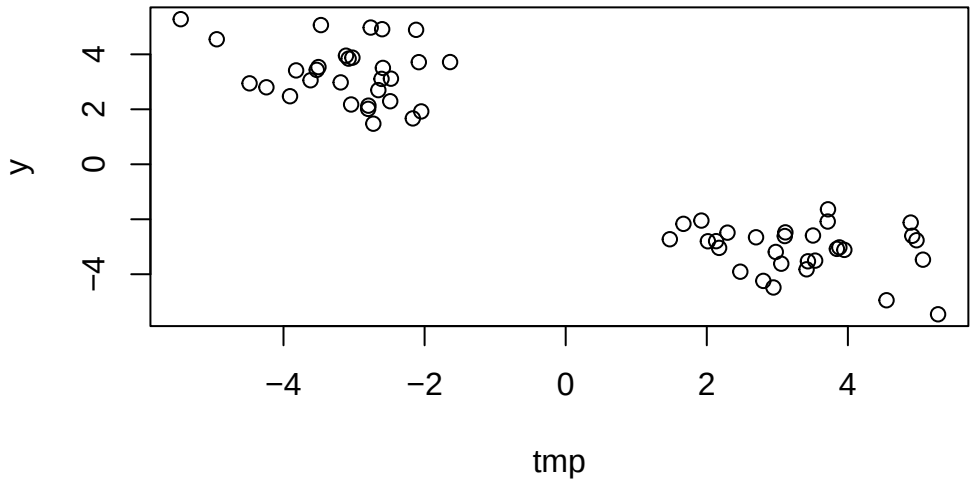
```
 [1]  1.1986328  0.7170577  2.2607876  3.7683486  3.2177567  1.6750286
 [7]  2.7407846  4.2379561  3.7599113  2.8800973  3.5595601  3.4528372
[13]  4.2028922  3.9032403  1.5894150  1.8519086  2.6521472  5.0425378
[19]  3.2492074  3.2437127  3.8612791  1.6327027  3.6780634  2.3519409
[25]  1.5433504  2.3847815  4.4240118  2.0282457  3.2827718  4.7294152
[31] -3.3849129 -2.7027476 -1.9039785 -1.2173619 -2.1025637 -4.1653089
[37] -3.0768385 -3.1260047 -1.8420060 -2.3558362 -2.9203543 -2.4134344
[43] -4.1894806 -4.3364479 -2.9496446 -3.5932300 -2.4025450 -2.0995388
[49] -4.1301124 -3.4951157 -5.4738147 -1.0874672 -3.5780736 -3.0490969
[55] -3.5141954 -3.1747685 -1.9403076 -2.4049477 -4.1164370 -1.2915785
```

Can shorten code doing this way.

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

Get the reverse to make another vector

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



Now run the Kmeans function to see how they cluster

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

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

Cluster means:
        tmp         y
1  3.317390 -3.099637
2 -3.099637  3.317390

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 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] 55.2345 55.2345
 (between_SS / total_SS =  91.8 %)

Available components:

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

Whats in this results?

```
attributes(k)
```

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

$class
[1] "kmeans"
```

What are the cluster centers?

```
k$centers
```

```
        tmp         y
1  3.317390 -3.099637
2 -3.099637  3.317390
```

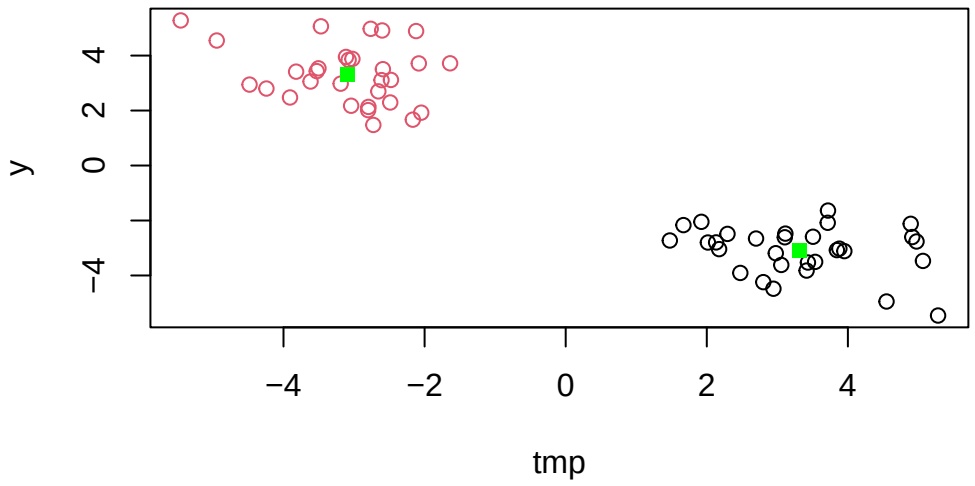
What are the clustering results?

```
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 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 your data `x` showing your clustering results and the center point for each cluster?

```r
plot(x, col=k$cluster)
points(k$centers, pch=15, col="green")
```



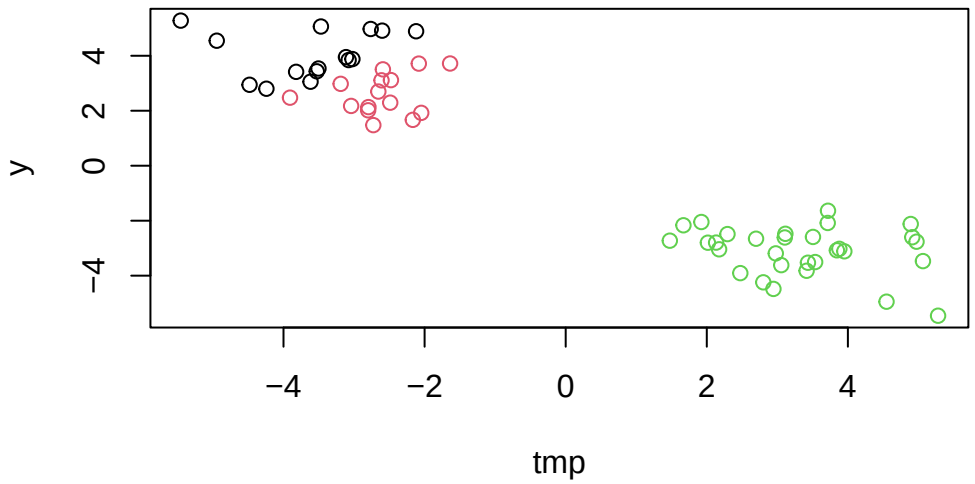
Q. Run kmeans and cluster into 3 groups and plot the results?

```r
k2<- kmeans(x,centers = 3, nstart=20)
k2
```

```
K-means clustering with 3 clusters of sizes 15, 15, 30

Cluster means:
        tmp         y
1 -3.584605  4.035169
2 -2.614669  2.599610
3  3.317390 -3.099637

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

Within cluster sum of squares by cluster:
[1] 21.23908 11.48338 55.23450
```

```
 (between_SS / total_SS =  93.5 %)

Available components:

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

```
plot(x,col=k2$cluster)
```



The big limitations of kmeans is that it imposes a structure on your data (it will force your data to fit what you told it to, aka. not real). Process requires arbitrarily but systematically (manually) applying until you find the best one.

Hierarchical clustering

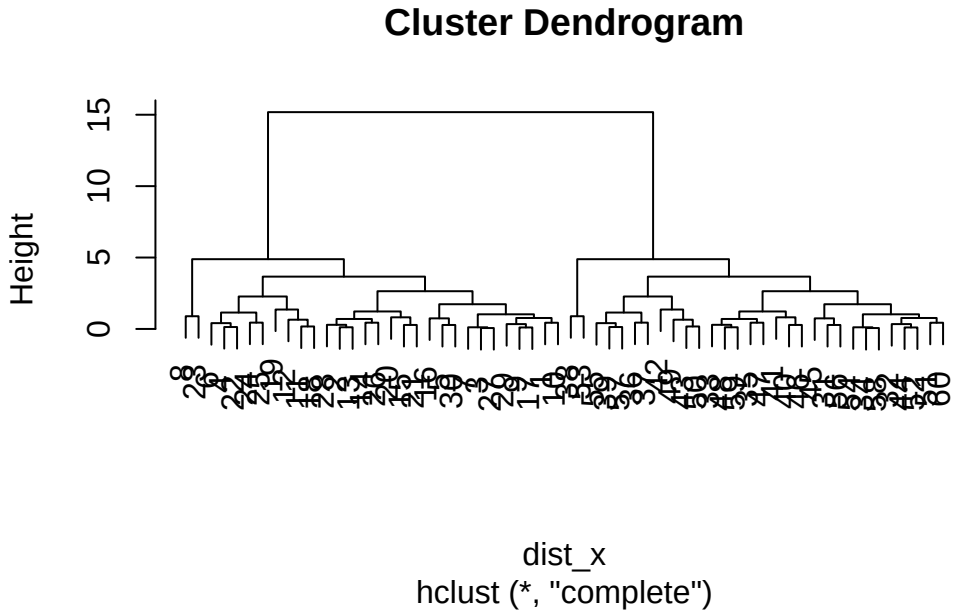
The main function in "base" R for this is called `hclust()`. It wants a distance matric as input not the data itself.

We can calculate a distance matrix in lots of different ways but here we will use the `dist()` function.

```r
dist_x<- dist(x)
# dist_x
hc<-hclust(dist_x)
# hc
```

There is a specific method to graphing hierarchical clustering. Try plotting it as is and see what the result is.

```r
plot(hc)
```



This shows you all the possible clusters based off your data, thereby not forcing your data into anything but takes it by the distance of each point from each other.

Lets learn how to cut your tree to get desired clusters. To get the cluster membership vector (equivalent of k$cluster in kmeans), we cut the tree at a given height we choose. The function to do this is called `cutree()`.

```r
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 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
```

```
# h cuts based off the height you pick in graph where k cuts based off the cluster you wan
```

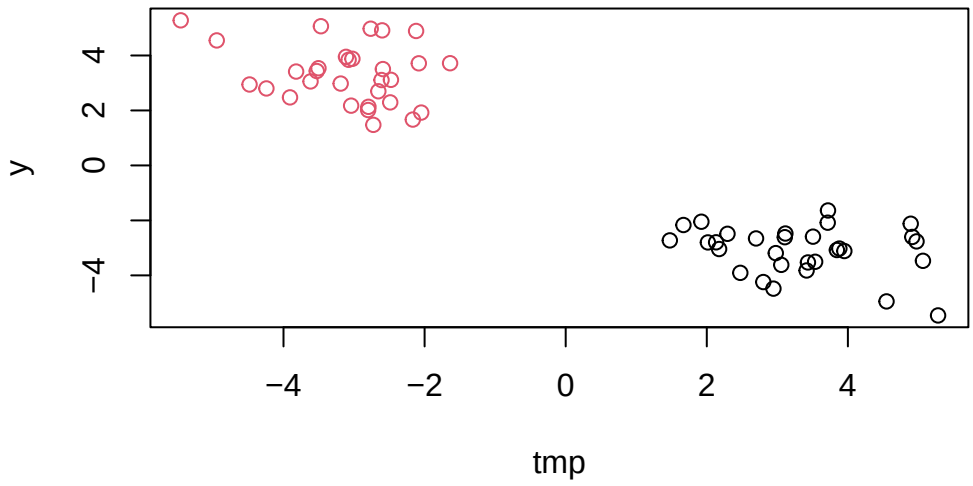
```
cutree(hc,k=2)
```

```
 [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
```

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

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

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



Principal Component Analysis (PCA): PCAs function by calculating new variables that represent the components of your data that account for the most variation.

Each quadrant represents a portion of the dataset and what is captured. However, quadrants are arbitrary and do not actually represent numbers or categories, but new variables of variance

within your dataset (ie. quadrant 1 has data that meets variance 1, and quadrant 2 has data that meets variance 2).

The motivation is trying to reduce the number of things to look at to gain insight of your data while only losing a small amount of information.

Lets use a real data set to understand PCA:

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

> Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

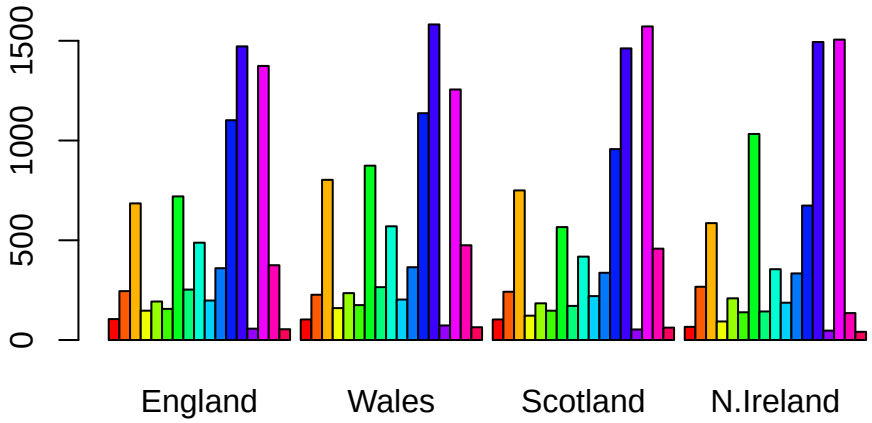
```
dim(x)
```

```
[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?

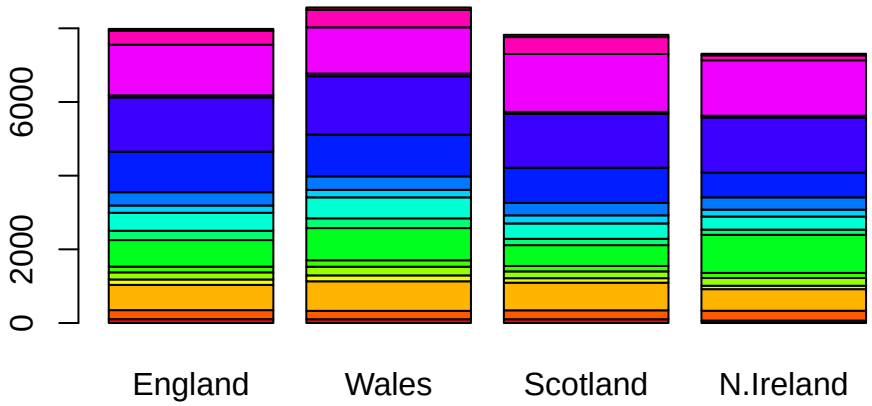
I prefer the reading in to the CSV file as it is straight forward and less work.

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



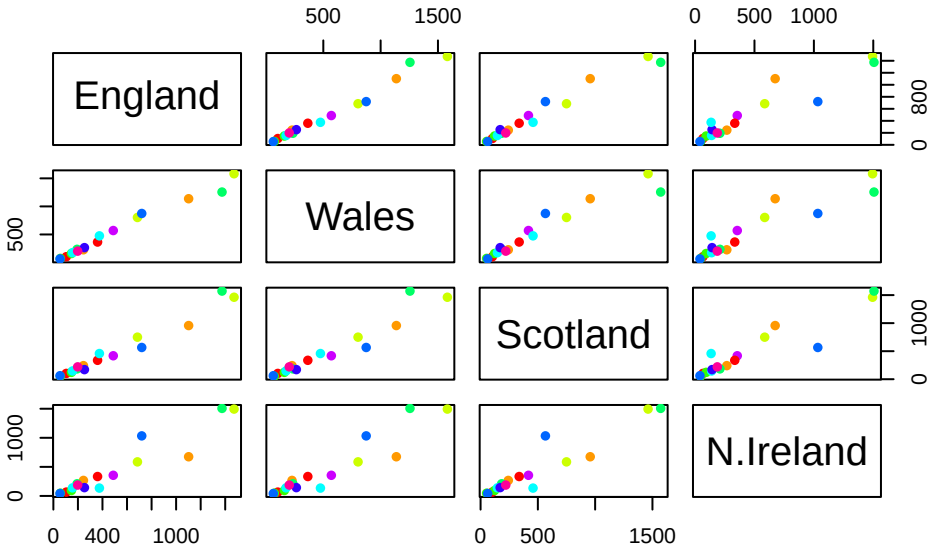
Q3: Changing what optional argument in the above barplot() function results in the following plot?

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



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)
```



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

N. Ireland eats much more potatoes and less fresh fruit than the other UK countries.

Enter PCA

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

It wants our foots as the columns and the countries as the rows.

`t()` transposes data (needed for this dataset)

`prcomp()` calculates the principal components of data

```
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"
```

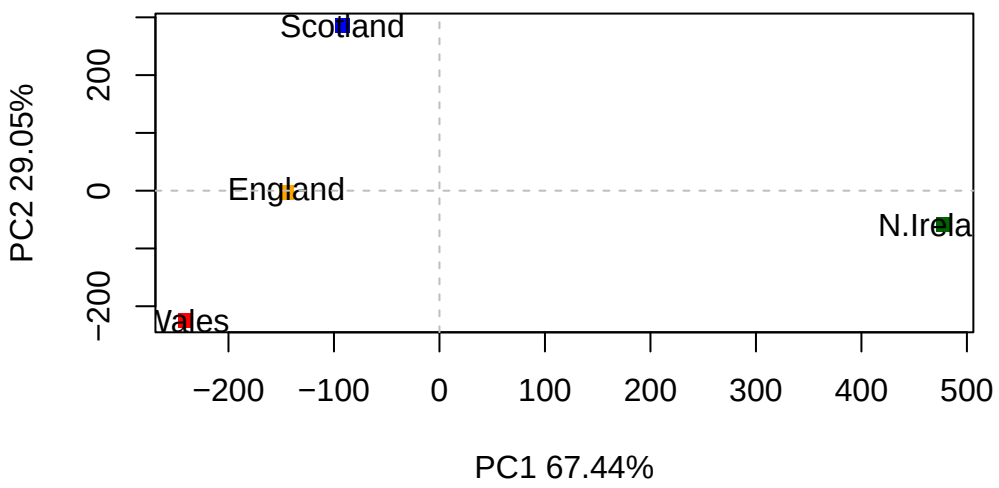
```
pca$x
```

```
                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
```

```
pca$rotation
```

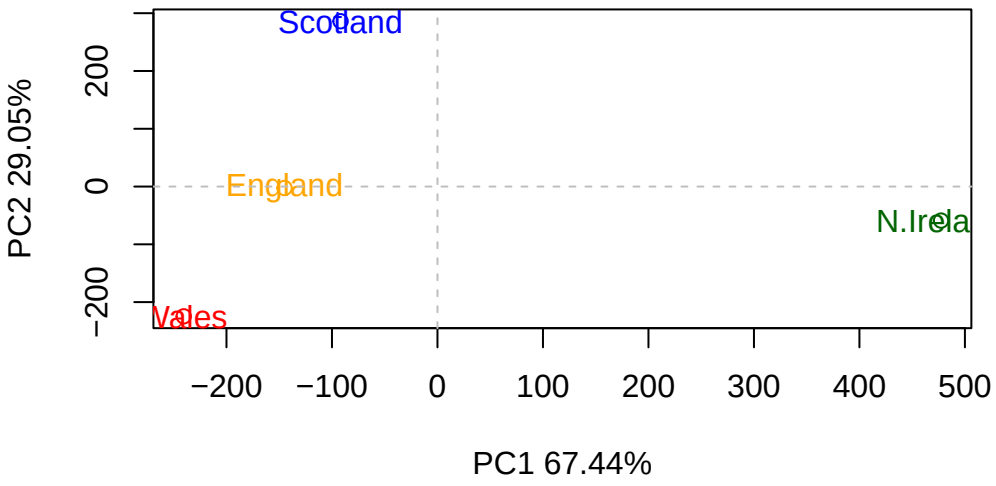
```
                            PC1          PC2         PC3          PC4
Cheese              -0.056955380  0.016012850  0.02394295 -0.409382587
Carcass_meat         0.047927628  0.013915823  0.06367111  0.729481922
Other_meat          -0.258916658 -0.015331138 -0.55384854  0.331001134
Fish                -0.084414983 -0.050754947  0.03906481  0.022375878
Fats_and_oils       -0.005193623 -0.095388656 -0.12522257  0.034512161
Sugars              -0.037620983 -0.043021699 -0.03605745  0.024943337
Fresh_potatoes       0.401402060 -0.715017078 -0.20668248  0.021396007
Fresh_Veg           -0.151849942 -0.144900268  0.21382237  0.001606882
Other_Veg           -0.243593729 -0.225450923 -0.05332841  0.031153231
Processed_potatoes  -0.026886233  0.042850761 -0.07364902 -0.017379680
Processed_Veg       -0.036488269 -0.045451802  0.05289191  0.021250980
Fresh_fruit         -0.632640898 -0.177740743  0.40012865  0.227657348
Cereals             -0.047702858 -0.212599678 -0.35884921  0.100043319
Beverages           -0.026187756 -0.030560542 -0.04135860 -0.018382072
Soft_drinks          0.232244140  0.555124311 -0.16942648  0.222319484
Alcoholic_drinks    -0.463968168  0.113536523 -0.49858320 -0.273126013
Confectionery       -0.029650201  0.005949921 -0.05232164  0.001890737
```

```
plot(pca$x[,1],pca$x[,2],xlab="PC1 67.44%",ylab="PC2 29.05%", col=c("orange","red","blue",
text(pca$x[,1], pca$x[,2], colnames(x))
abline(h=0, col="gray", lty=2)
abline(v=0, col="gray", lty=2)
```



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.

```
plot(pca$x[,1],pca$x[,2],xlab="PC1 67.44%",ylab="PC2 29.05%", col=c("orange","red","blue",
text(pca$x[,1], pca$x[,2], colnames(x), col=c("orange","red","blue","darkgreen"))
abline(h=0, col="gray", lty=2)
abline(v=0, col="gray", lty=2)
```



We can manually calculate the variance here using the standard deviation.

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

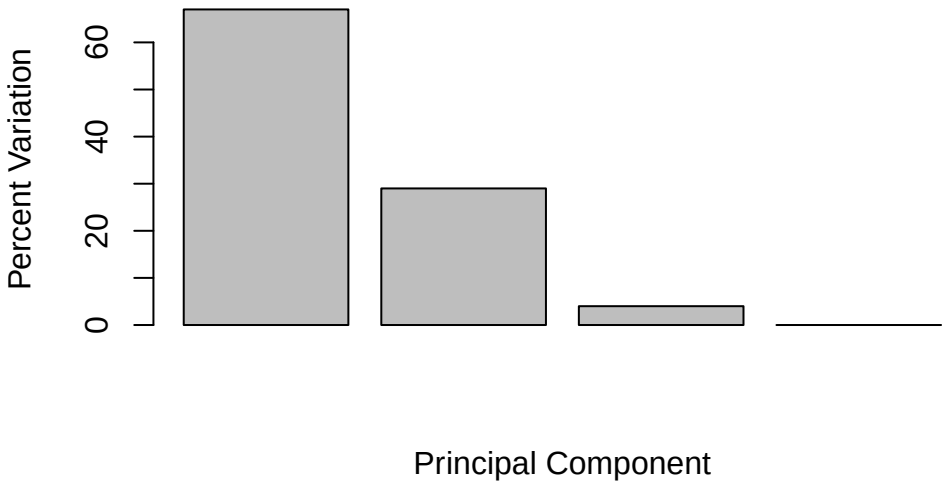
```
[1] 67 29  4  0
```

We can use the `summary()` function to do the same. Given in the second row.

```
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 2.921348e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```


Percent Variation
0
20
40
60
Principal Component