Machine Learning 1

Dominique Lie (A15470100)

10/21/2021

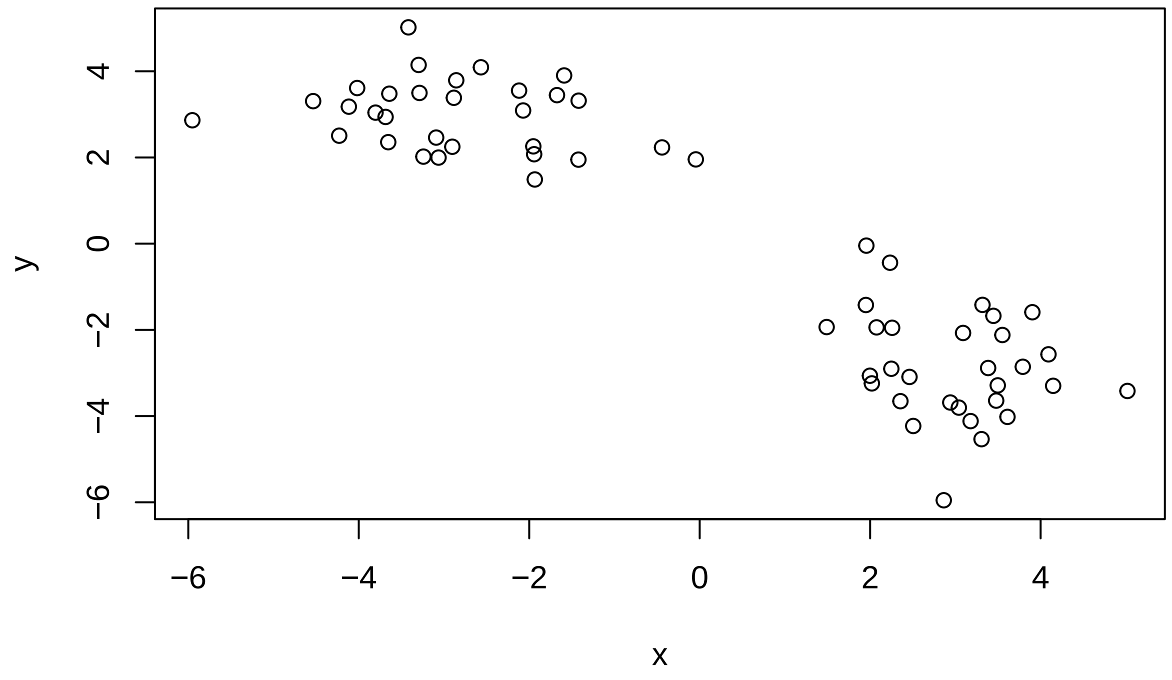
First up is clustering methods

Kmeans clustering

The function in base R to do Kmeans clustering is called 'kmeans()'

Generate some example data where we know what the answer should be:

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

plot(x)
```



```r
#rnorm generates random data that is normalized
```

Q. Can we use kmeans() to cluster the data?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -2.828659  2.973690
## 2  2.973690 -2.828659
##
## 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 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] 65.30045 65.30045
##  (between_SS / total_SS =  88.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```r
km$size
```

```
## [1] 30 30
```

Q. What 'component' of your result object details -cluster size? (refer previous question) -cluster assignment/membership? -cluster center?

```r
km$cluster
```

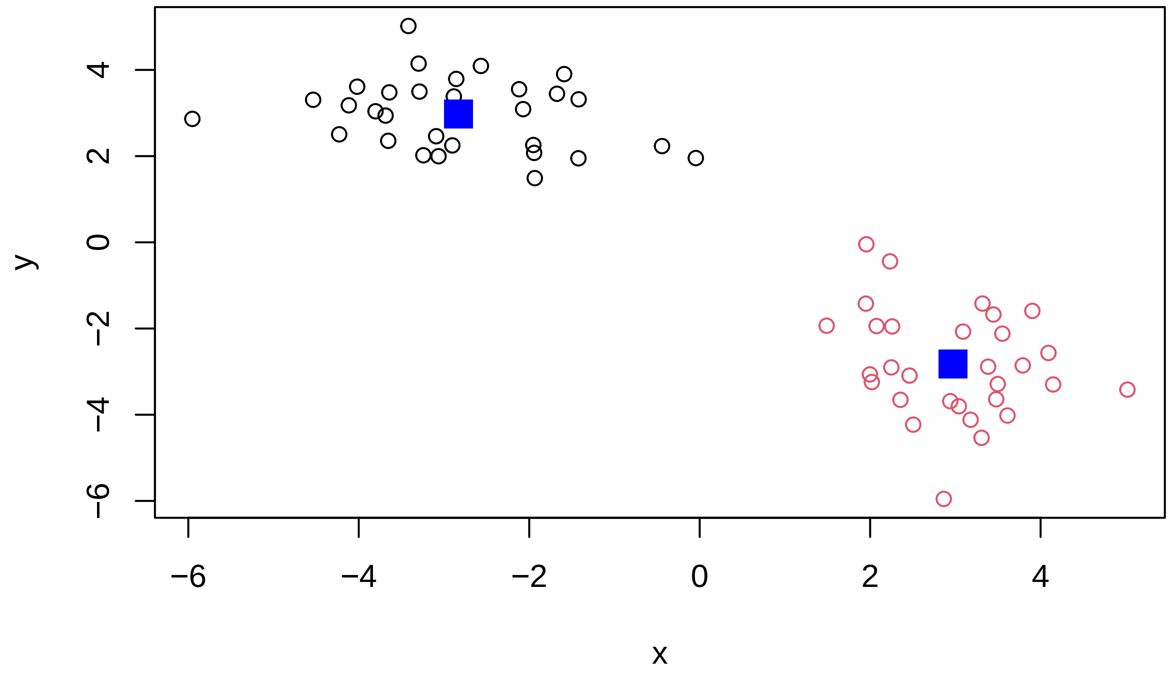
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```r
km$centers
```

```
##           x         y
## 1 -2.828659  2.973690
## 2  2.973690 -2.828659
```

Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```r
plot(x, col = km$cluster)
points(km$centers, col = "blue", pch = 15, cex = 2)
```



hclust

A big limitation with kmeans is that we have to tell it K (the number of clusters we want) Analyze this same data with hclust()

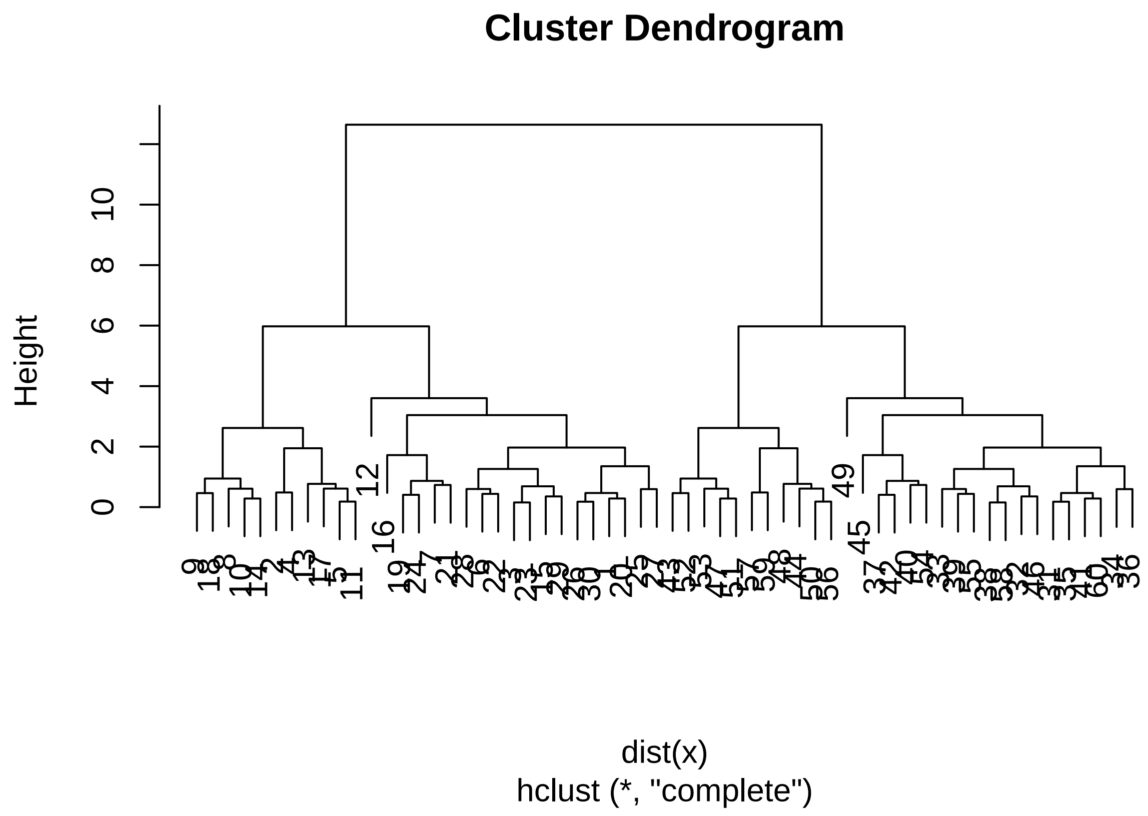
Demonstrate the use of dist(), hclust(), plot() , and cutree() functions to do clustering, Generate dendogras and return cluster assignment/membership vector…

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

```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

There is a plot method for hclust result objects. Let's see it.

```r
plot(hc)
```



To get our cluster membership vector we have to do a wee bit more work. We have to "cut" the tree where we think it makes sense. For this we use the 'cutree()' function

```r
cutree(hc, h = 6)
```

```
##  [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 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
```

You can also call cutree() setting k = the number of grps/clusters you want.

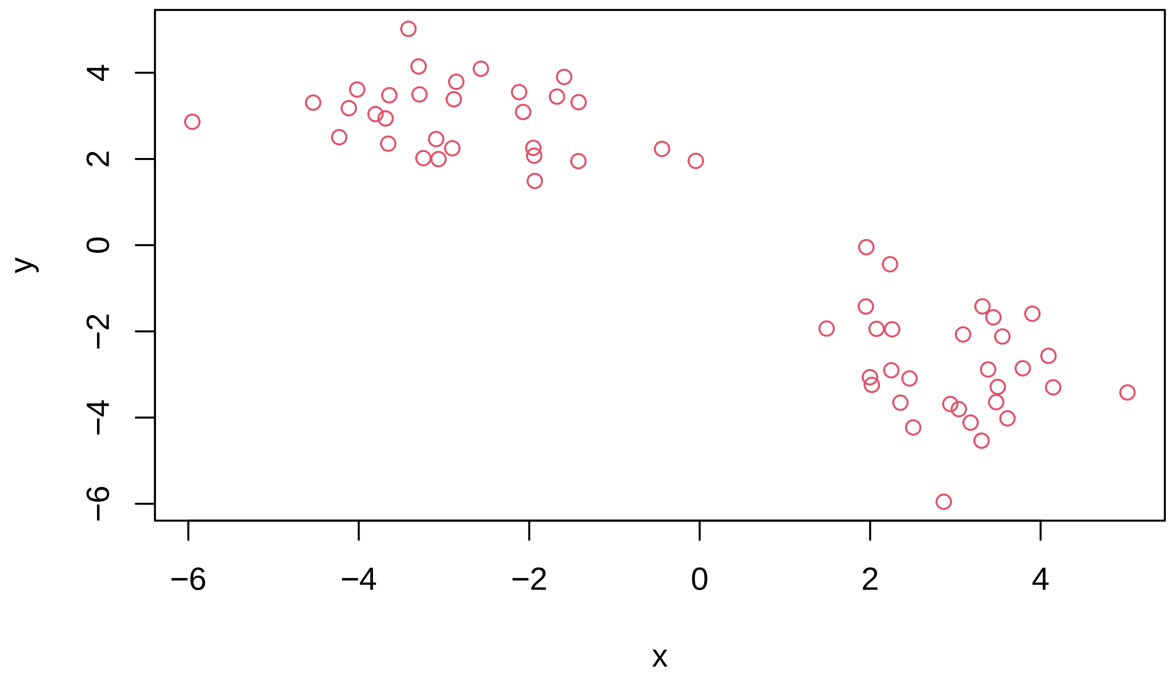
```r
cutree(hc, k = 2)
```

```
##  [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 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
```

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

Make our results plot

```r
plot(x, col = 2)
```



Principal Component Analysis

Data import

```r
url <-  "https://tinyurl.com/UK-foods"
x <- read.csv(url)
```

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

```r
nrow(x)
```

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

```r
ncol(x)
```

```
## [1] 5
```

```r
rownames(x) <-  x[,1]
x <-  x[,-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
```

Not a great method because rerunning code will keep removing columns

```
dim(x)
```

```
## [1] 17  4
```

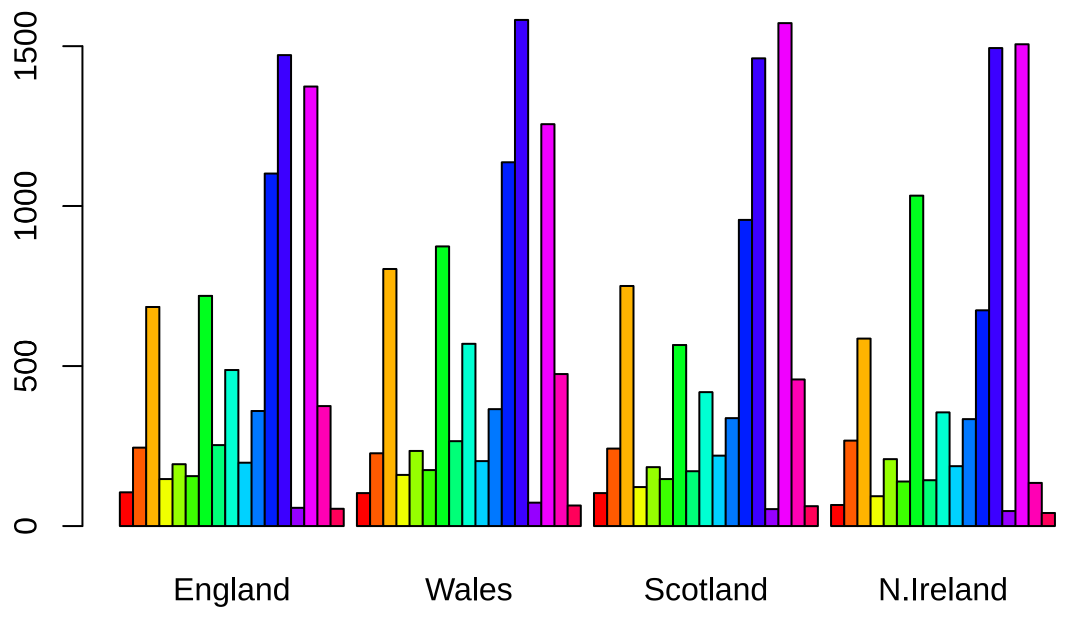
```
read.csv(url, row.names = 1)
```

```
##                     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
```

> 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?

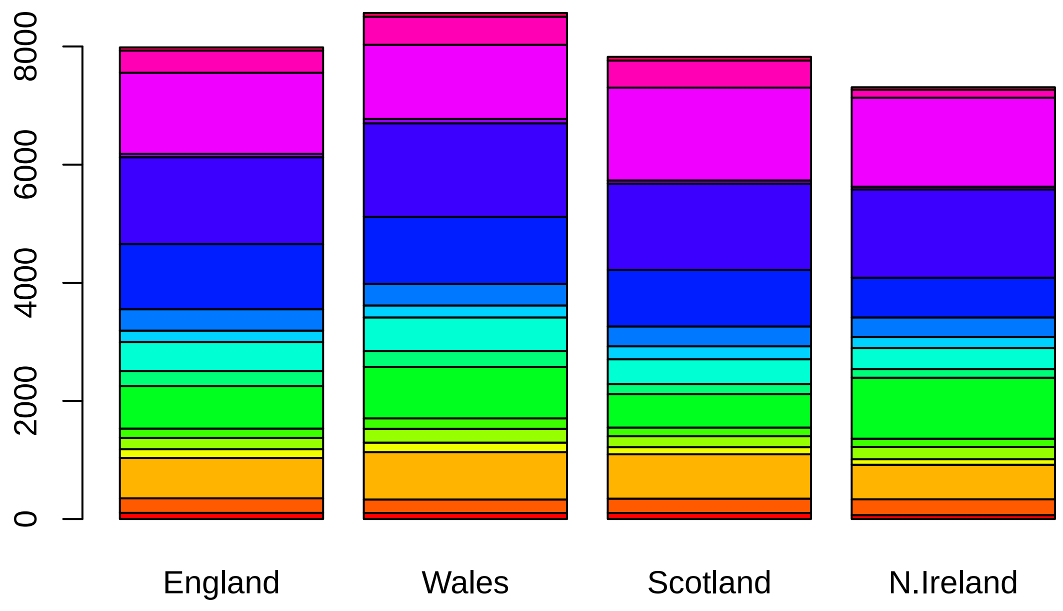
If you run the first code block multiple times you will keep losing columns. Using the row.names argument is more effective because you will prevent loss of data.

```
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?

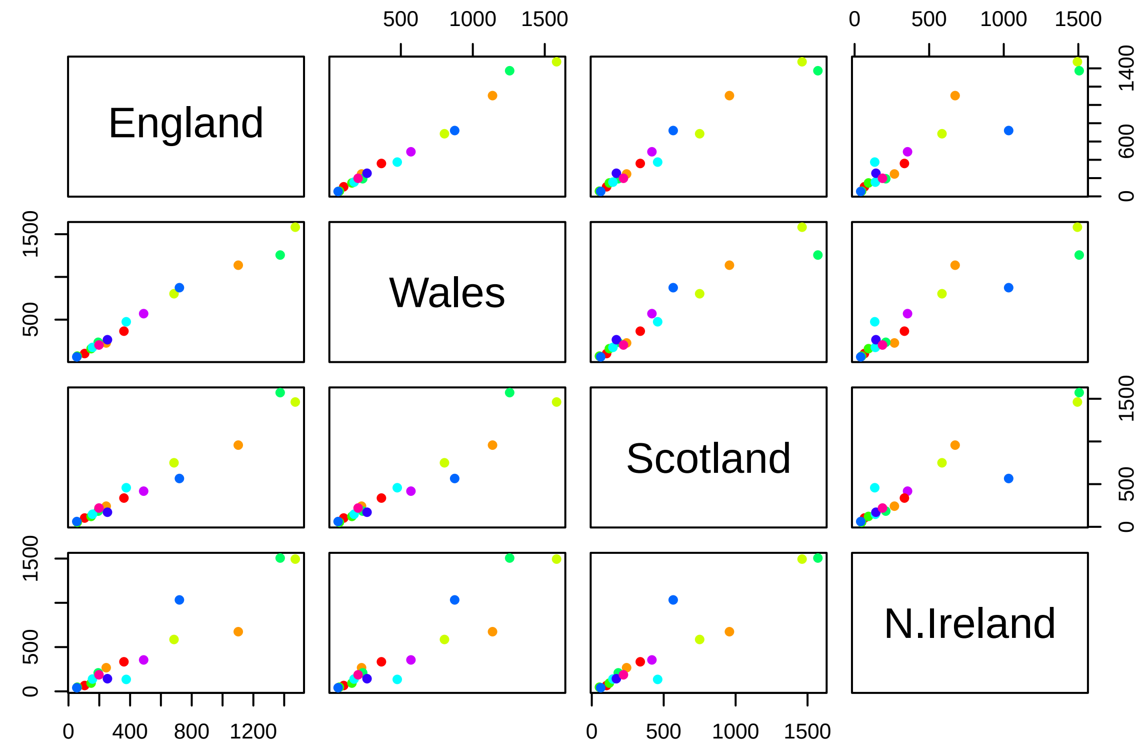
```
barplot(as.matrix(x), col = rainbow(nrow(x)))
```



Remove the 'beside = T' argument

Q5 (mislabeled is Q4) Generating all pairwise plots. 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?

The points that are not on the diagonal (the blue and orange points) are different than the other countries.

PCA to the rescue

The main function in base R is 'prcomp()' This want's the transpose of our data

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

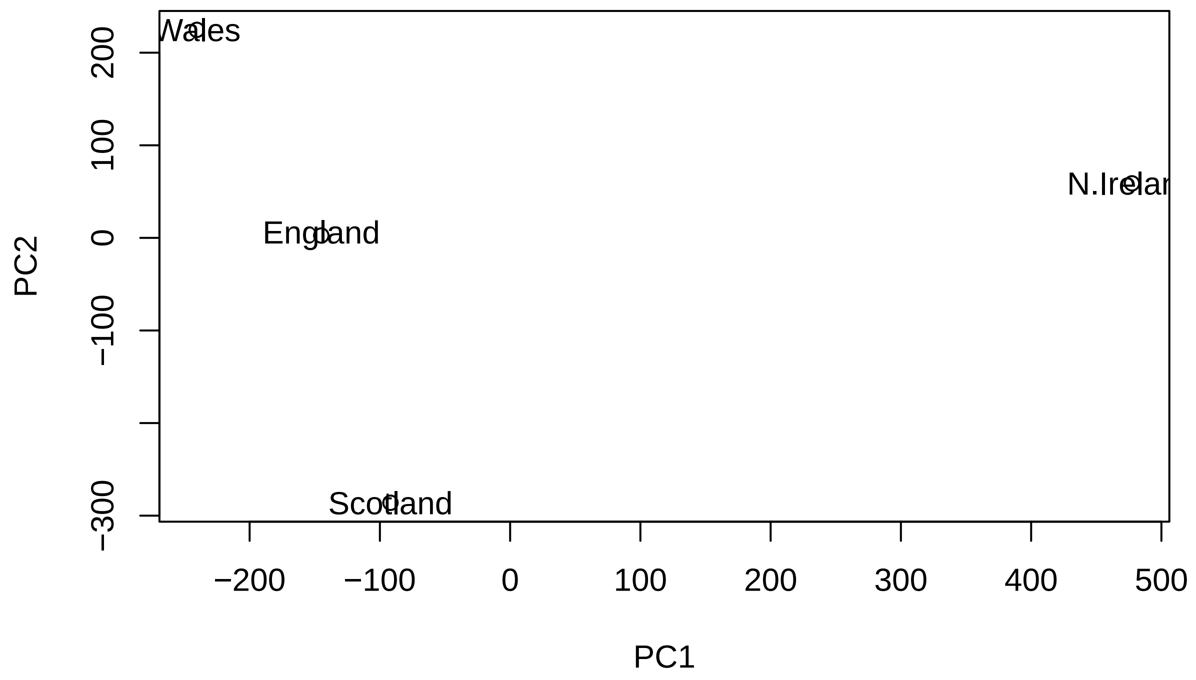
```r
attributes(pca)
```

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

```
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2", plim = c(-270, 500))
```

```
## Warning in plot.window(...): "plim" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "plim" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "plim" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "plim" is not a
## graphical parameter

## Warning in box(...): "plim" is not a graphical parameter

## Warning in title(...): "plim" is not a graphical parameter
```

```
text(pca$x[,1], pca$x[,2], colnames(x))
```

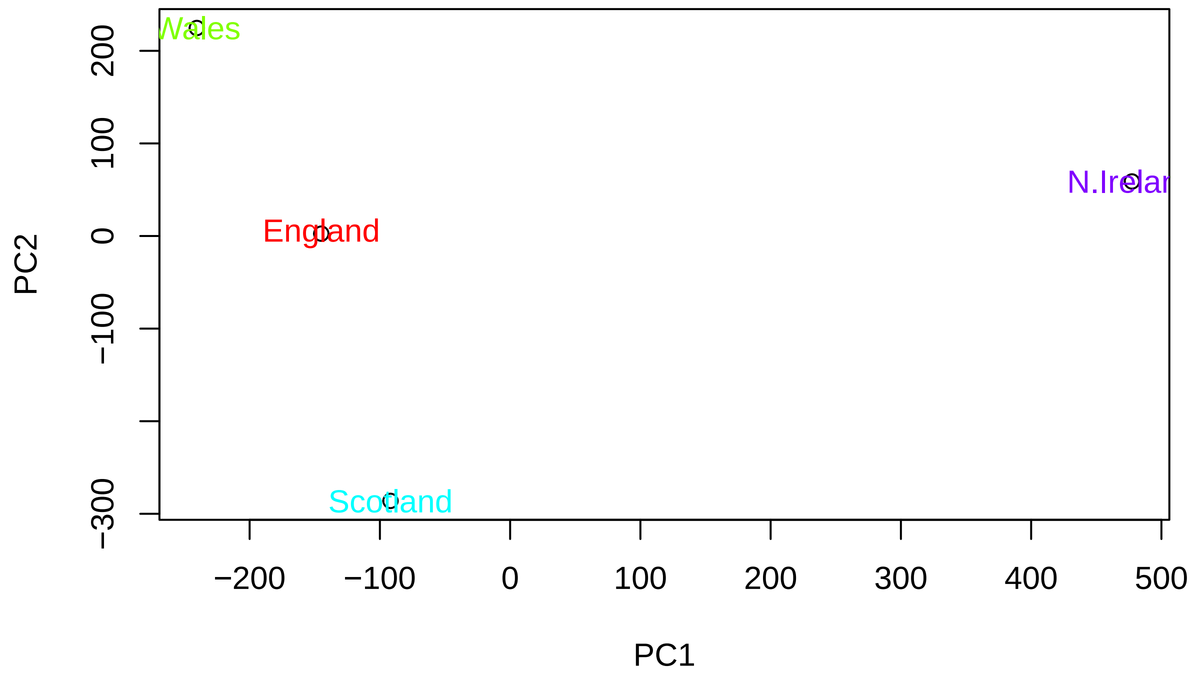


Q8. Customize your plot so that the colors of the country names match the colors in our UK and Irland map and table at start of this document.

```r
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2", plim = c(-270, 500))
```

```
## Warning in plot.window(...): "plim" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "plim" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "plim" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "plim" is not a
## graphical parameter

## Warning in box(...): "plim" is not a graphical parameter

## Warning in title(...): "plim" is not a graphical parameter
```

```r
text(pca$x[,1], pca$x[,2], colnames(x), col = rainbow(4))
```



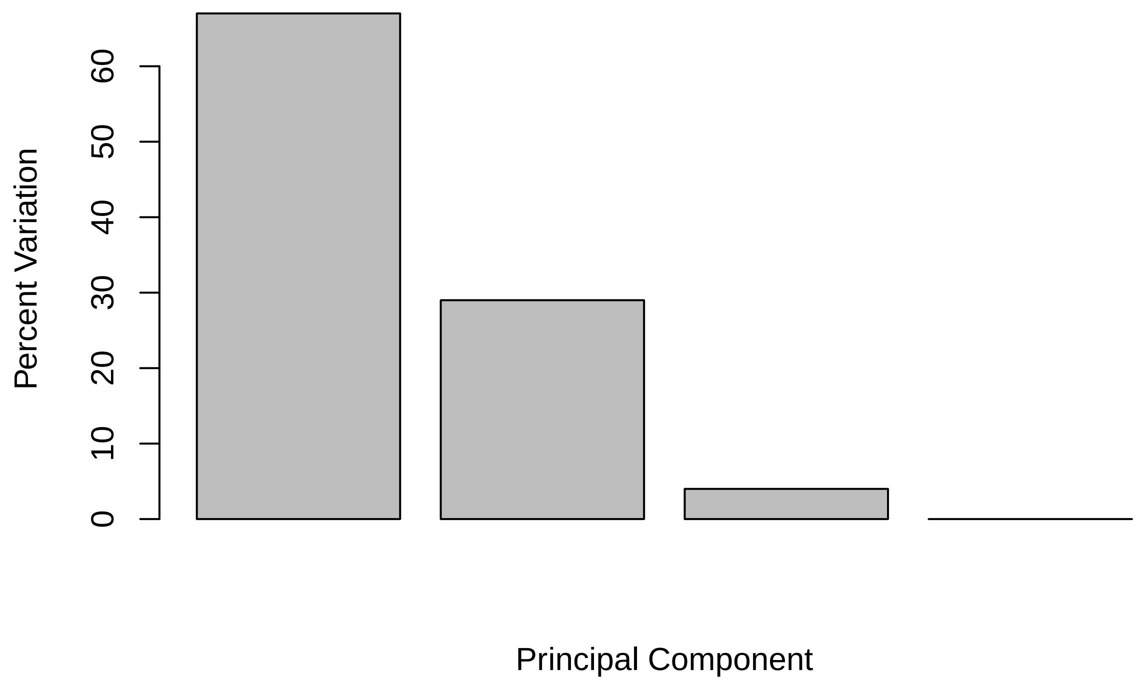
```r
v <-  round(pca$sdev^2/sum(pca$sdev^2)*100)
v
```

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

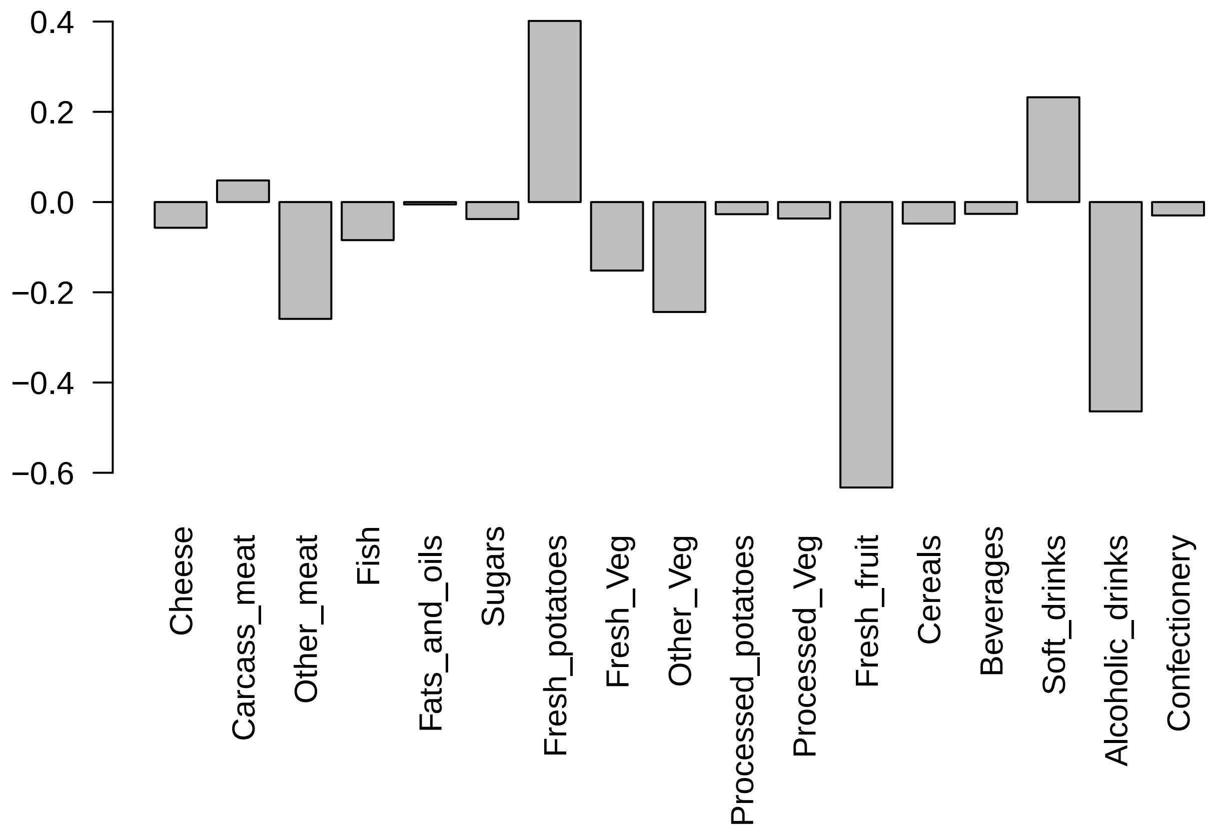
```r
z <- summary(pca)
z$importance
```

```
##                              PC1       PC2      PC3          PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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

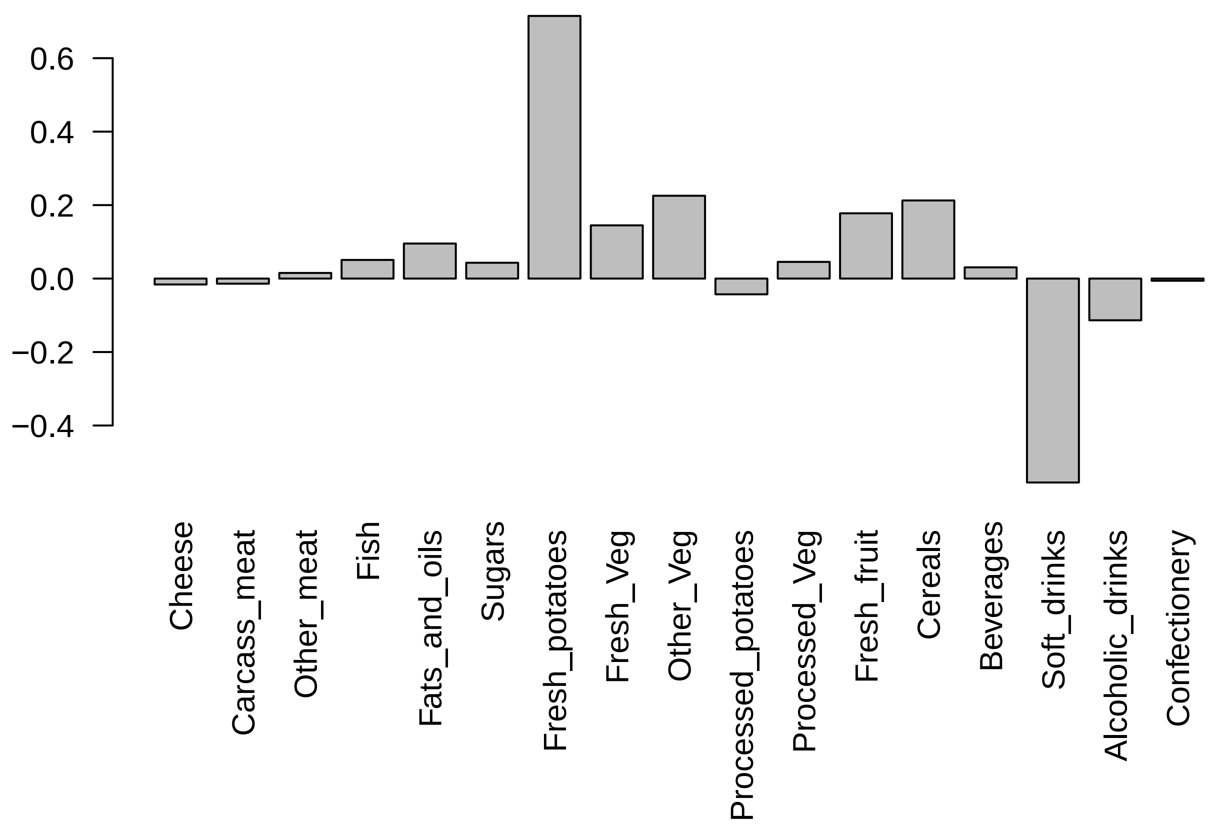


```r
par(mar = c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las =2)
```



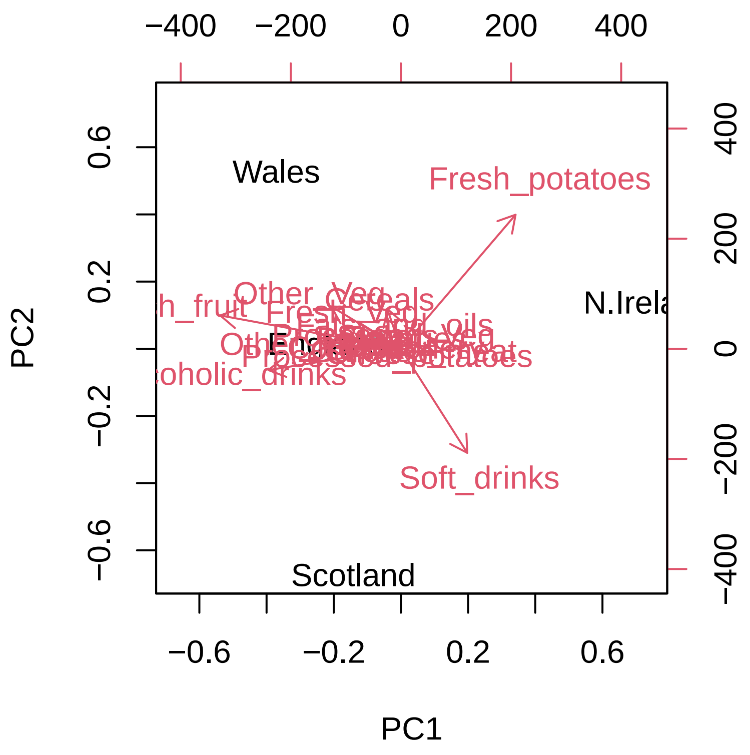
Q9. Generate a similar 'loadings plot' for PC2. What two food groups feature prominantly and what does PC2 mainly tell us about?

```
par(mar = c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las =2)
```



PC1 reduces the data down into one dimension that covers about 67 percent of the data. PC2 covers is another dimension that covers another 29 percent of the data.

```
biplot(pca)
```



# PCA of RNA-seq data

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <-  read.csv(url2, row.names = 1)
head(rna.data)
```

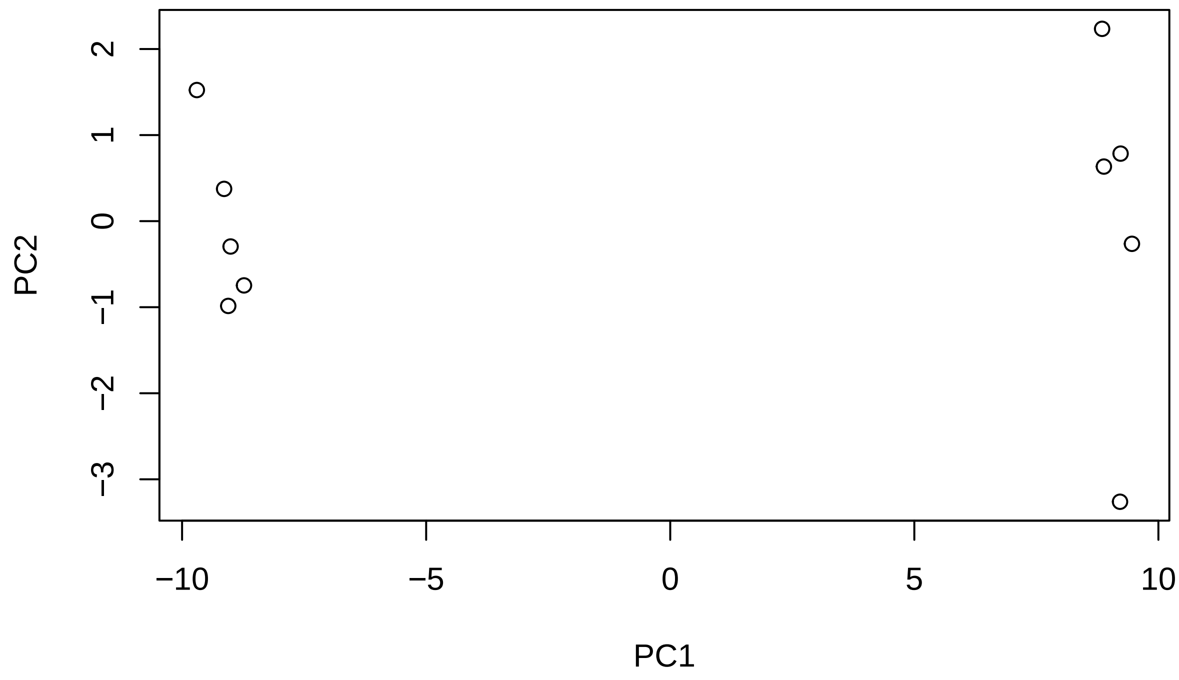
```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

Q10 How many genes and samples are in this data set?

```
dim(rna.data)
```

```
## [1] 100  10
```

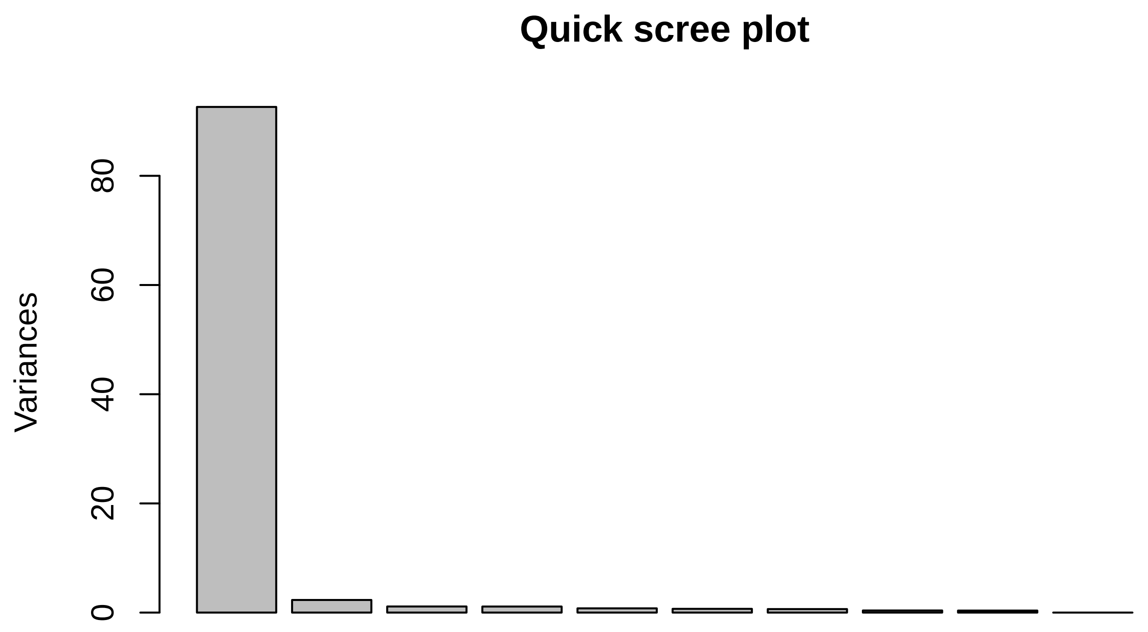
```
# Again we haveto take the transpose of our data
pca <-  prcomp(t(rna.data), scale = TRUE)

#Simple unpolished plot of pc1 and pc2
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2")
```



```r
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
##                            PC8     PC9      PC10
## Standard deviation     0.62065 0.60342 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

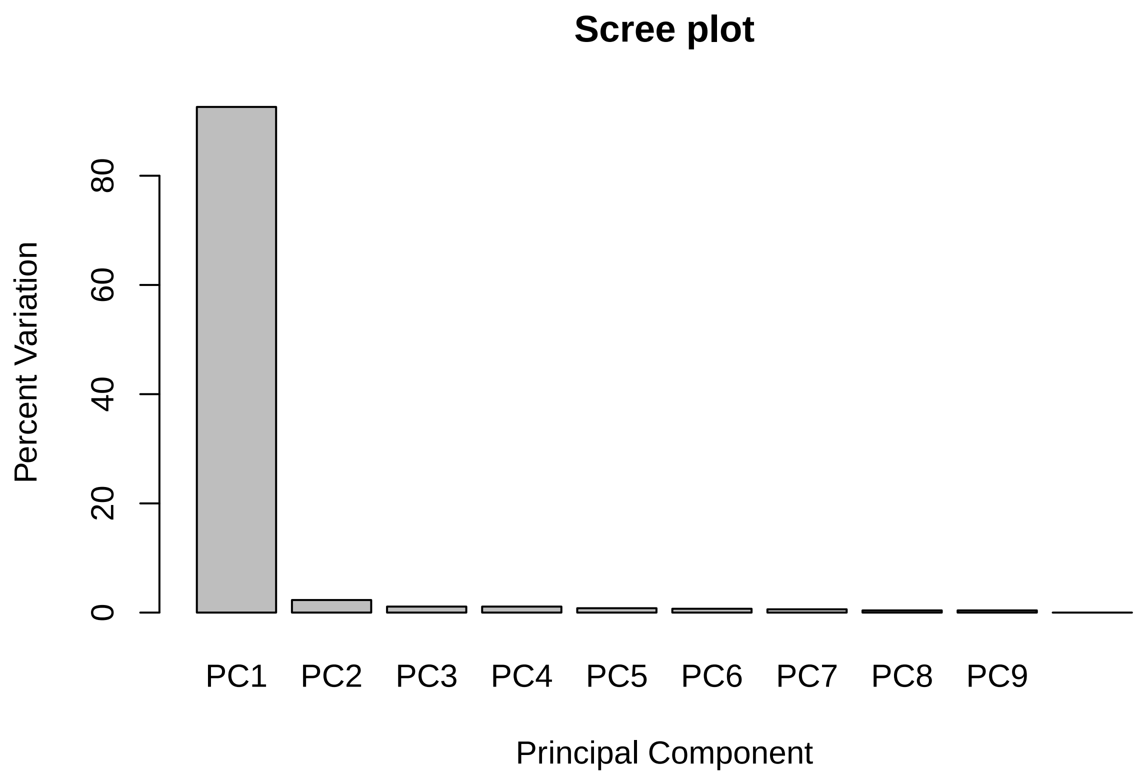
```r
plot(pca, main = "Quick scree plot")
```



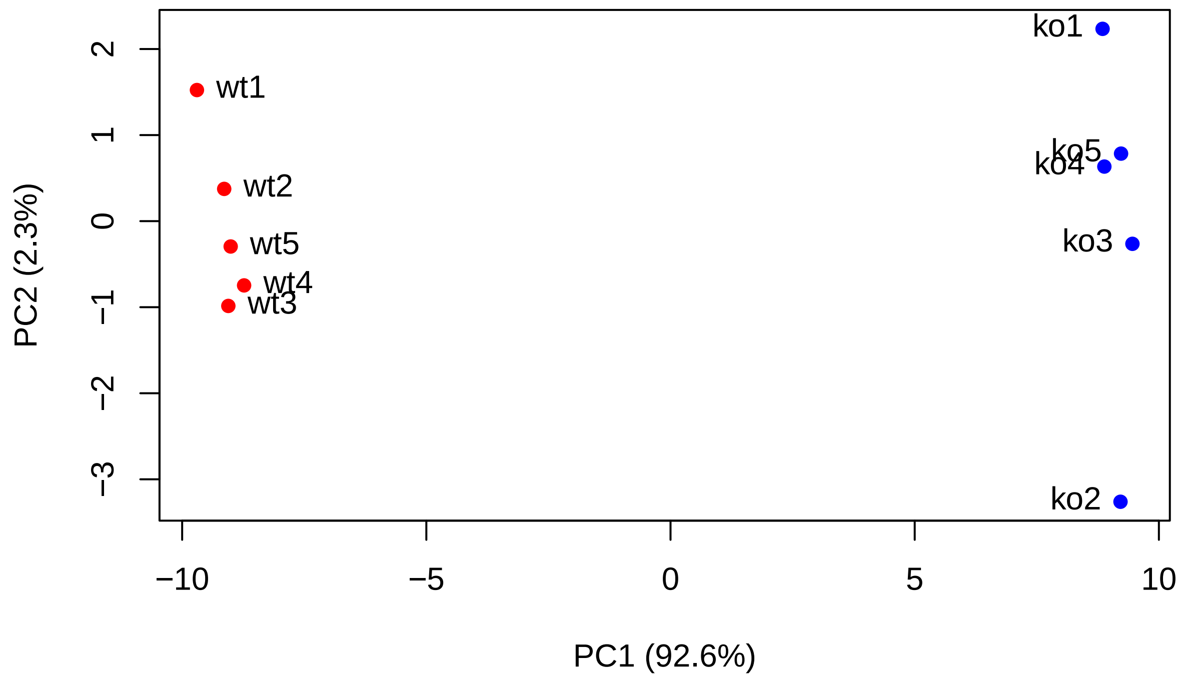
```
pca.var <-  pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

```
barplot(pca.var.per, main = "Scree plot", names.arg = paste0("PC", 1:10), xlab = "Principal Component",
```

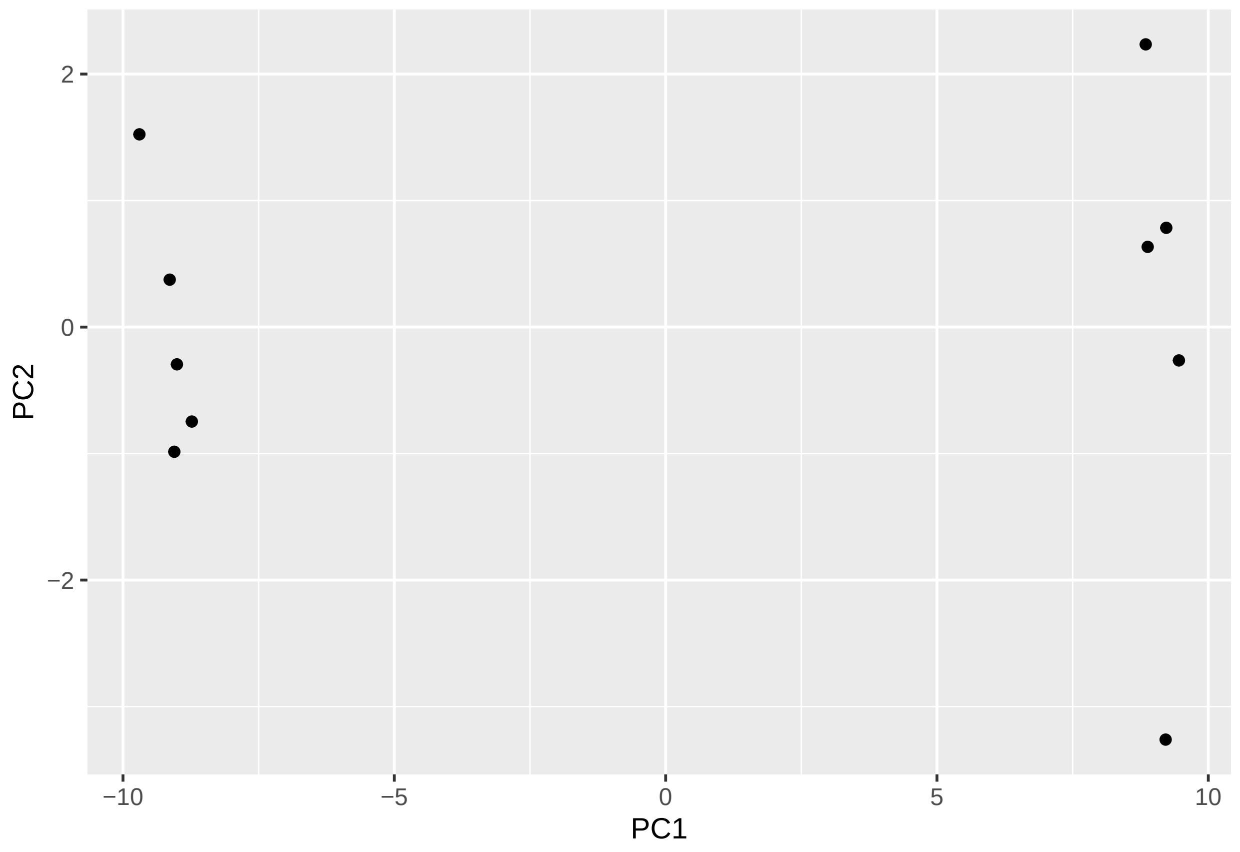


```r
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

plot(pca$x[,1], pca$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%)"))

text(pca$x[,1], pca$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```

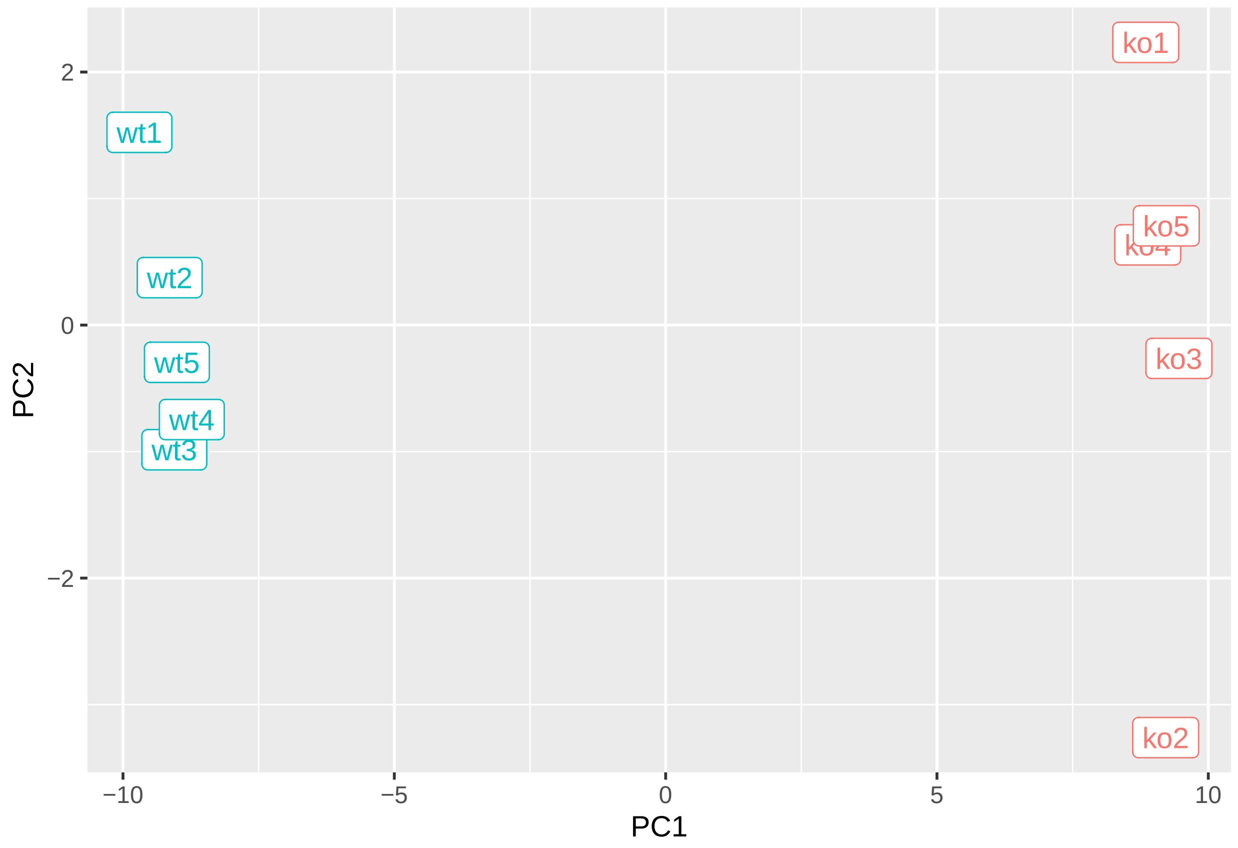


# Use ggplot

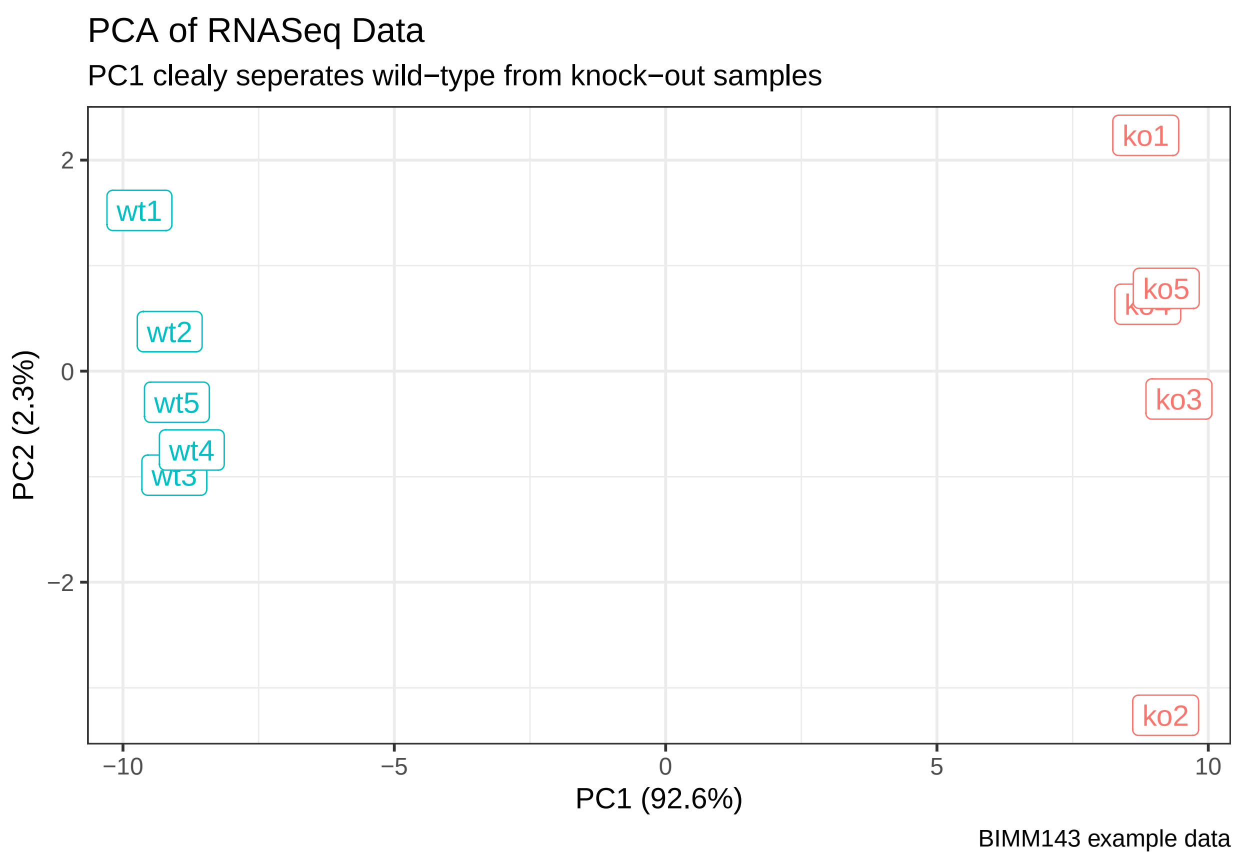
```
library(ggplot2)

df <- as.data.frame(pca$x)

# Our first basic plot
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



```
# Add a 'wt' and 'ko' "condition" column
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)

p <- ggplot(df) +
        aes(PC1, PC2, label=samples, col=condition) +
        geom_label(show.legend = FALSE)
p
```



```
p + labs(title="PCA of RNASeq Data",
       subtitle = "PC1 clealy seperates wild-type from knock-out samples",
       x=paste0("PC1 (", pca.var.per[1], "%)"),
       y=paste0("PC2 (", pca.var.per[2], "%)"),
       caption="BIMM143 example data") +
     theme_bw()
```



```
loading_scores <- pca$rotation[,1]

## Find the top 10 measurements (genes) that contribute
## most to PC1 in either direction (+ or -)
gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## show the names of the top 10 genes
top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes
```

```
##  [1] "gene100" "gene66"  "gene45"  "gene68"  "gene98"  "gene60"  "gene21"
##  [8] "gene56"  "gene10"  "gene90"
```