Machine Learning 1

Tasnia Sharia (PID A15931128)

10/21/2021

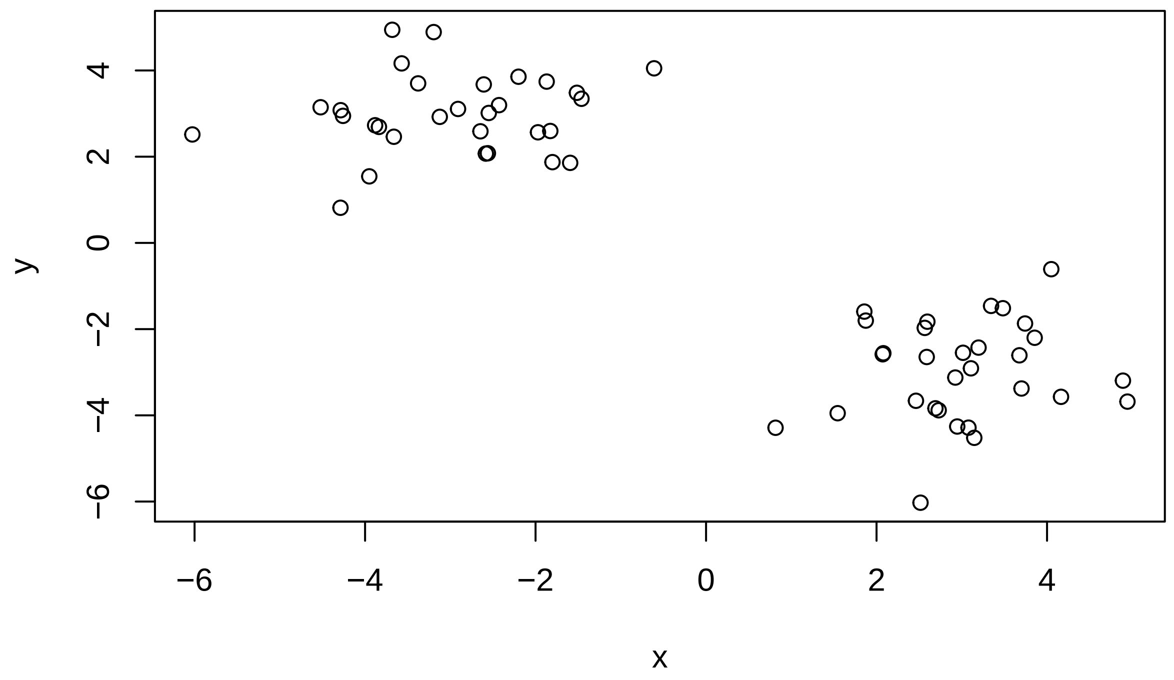
First is clustering methods

Kmeans clustering

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

First make up some 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)
```



Q. Can we use kmeans() to cluster this data setting k to 2 and nstart to 20?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  2.988375 -2.959826
## 2 -2.959826  2.988375
##
## 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 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 1 1
##
## Within cluster sum of squares by cluster:
## [1] 64.22261 64.22261
##  (between_SS / total_SS =  89.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
km$size
```

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

Q. What 'component' of your result object details cluster assignment/membership?

```
km$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 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 1 1
```

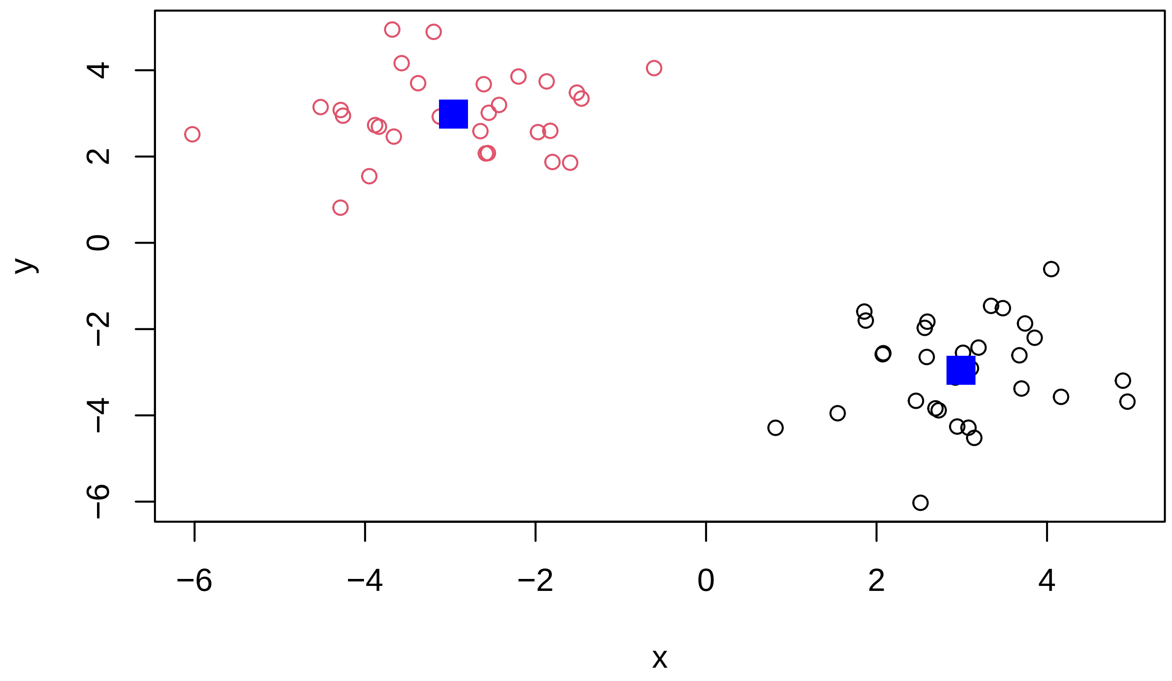
Q. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1  2.988375 -2.959826
## 2 -2.959826  2.988375
```

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

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



Hierarchical Clustering

A big limitation with k-means is that we have to tell it K (the number of clusters we want).

Analyze this same data with hclust()

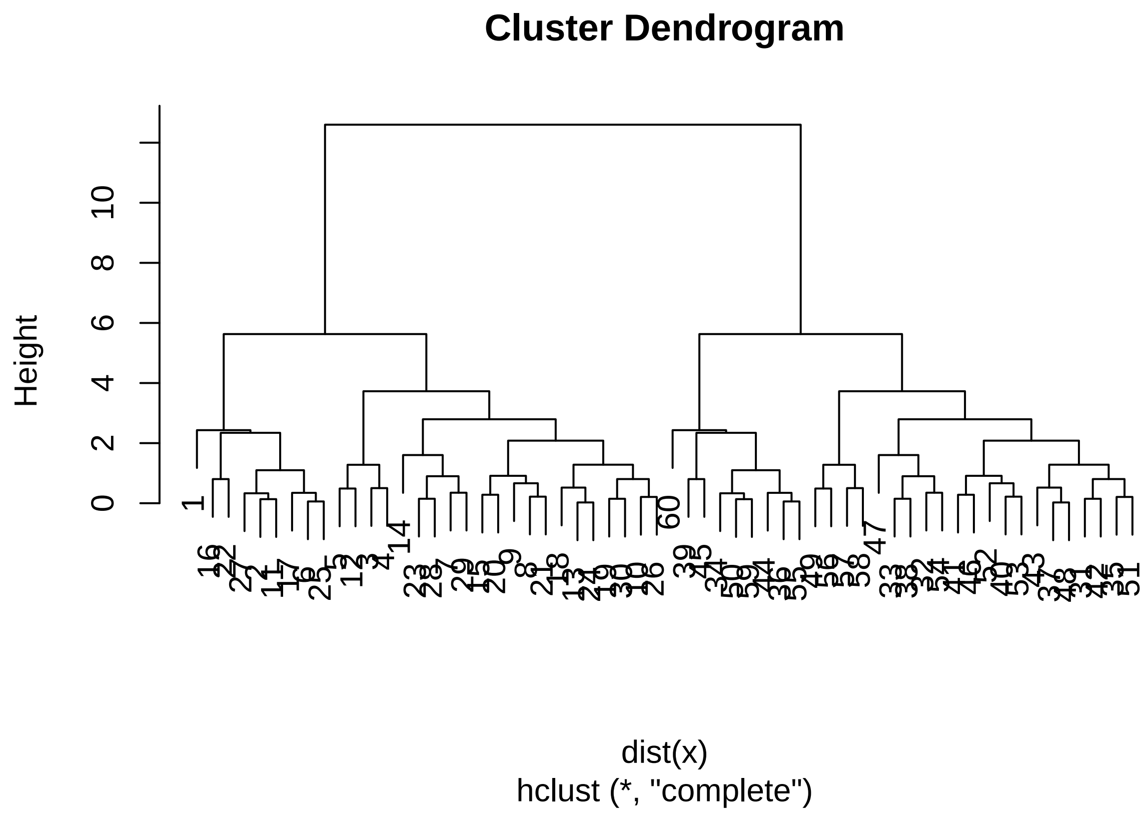
Demonstrate the use of dist(), hclust(), plot(), and cutree() function to do clustering. Generate dendeograms and return cluster assignment membership vector.

```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.

```
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.

```
cutree(hc, h=8)
```

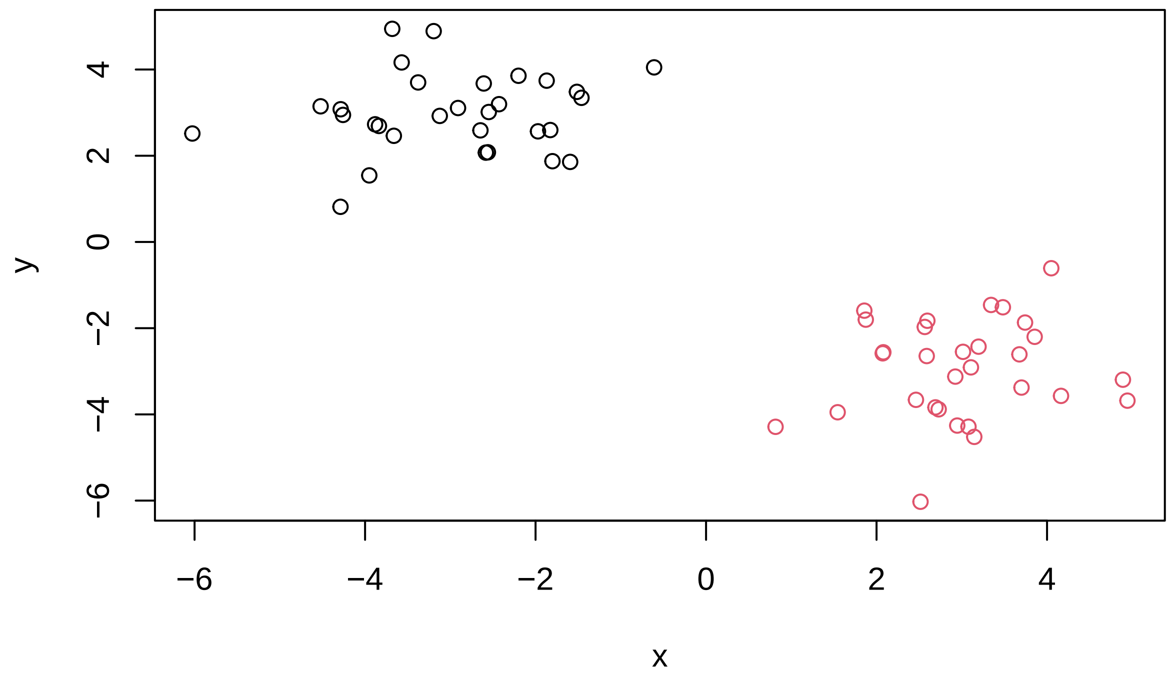
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

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

Make our results plot

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



Now, We will examine the PCA of UK food data

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

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

```
nrow(x)
```

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

```
ncol(x)
```

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

```
dim(x)
```

```
## [1] 17  5
```

The dim() function returns the # of rows and columns. Using nrow() or ncol() functions also provide the answer

```
## Preview the first 6 rows
head(x)
```

```
##                X England Wales Scotland N.Ireland
## 1         Cheese     105   103      103        66
## 2  Carcass_meat      245   227      242       267
## 3    Other_meat      685   803      750       586
## 4           Fish     147   160      122        93
## 5 Fats_and_oils      193   235      184       209
## 6         Sugars     156   175      147       139
```

```
## Preview the last 6 rows
tail(x)
```

```
##                    X England Wales Scotland N.Ireland
## 12      Fresh_fruit    1102  1137      957       674
## 13          Cereals    1472  1582     1462      1494
## 14         Beverages     57    73       53        47
## 15      Soft_drinks    1374  1256     1572      1506
## 16 Alcoholic_drinks     375   475      458       135
## 17    Confectionery      54    64       62        41
```

The data should be 17 by 4 dimensions. There is an extra first column that needs to be fixed.

```
# Note how the minus indexing works
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
```

Now let's check the dimensions again

```
dim(x)
```

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

Alternative approach to setting the correct row-names

```
x <- read.csv(url, row.names=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
```

```
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?

The second approach to solving the 'row-names problem' where we include the row.names argument is more convenient.

```
# What would happened if we ran row names(x) <- x[,1] again
rownames(x) <- x[,1]
head(x)
```

```
##     England Wales Scotland N.Ireland
## 105     105   103      103        66
## 245     245   227      242       267
## 685     685   803      750       586
## 147     147   160      122        93
## 193     193   235      184       209
## 156     156   175      147       139
```

By running rownames(x) <- x[,1] multiple times, the first column from the previous run would be removed. This would cause a problem of having data being removed after each run when the code is rewritten.

Now we go back to the original data with the right number of dimensions and correct headings.

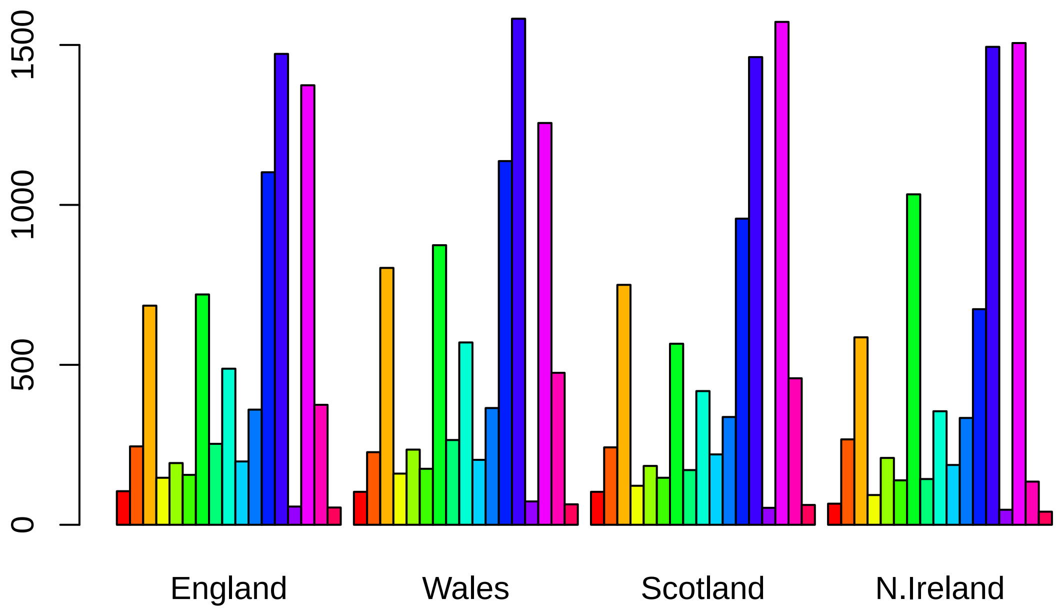
```
x <- read.csv(url, row.names=1)
head(x)
```

```
##                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
```

```
View(x)
```

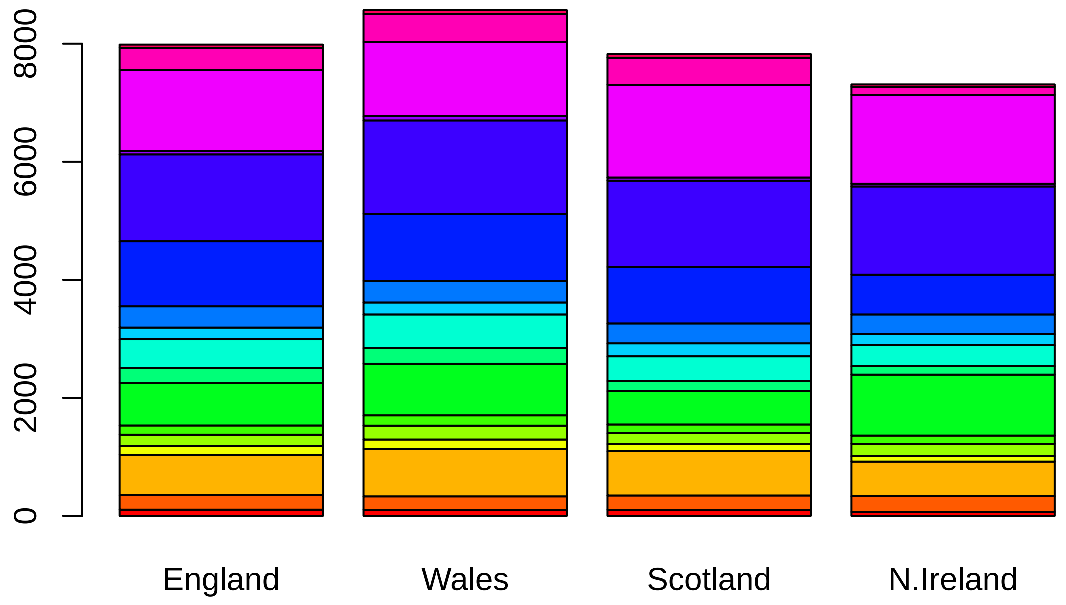
We will spot major differences and trends by generating boxplots and various pairwise plots that may not help as much

```
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?

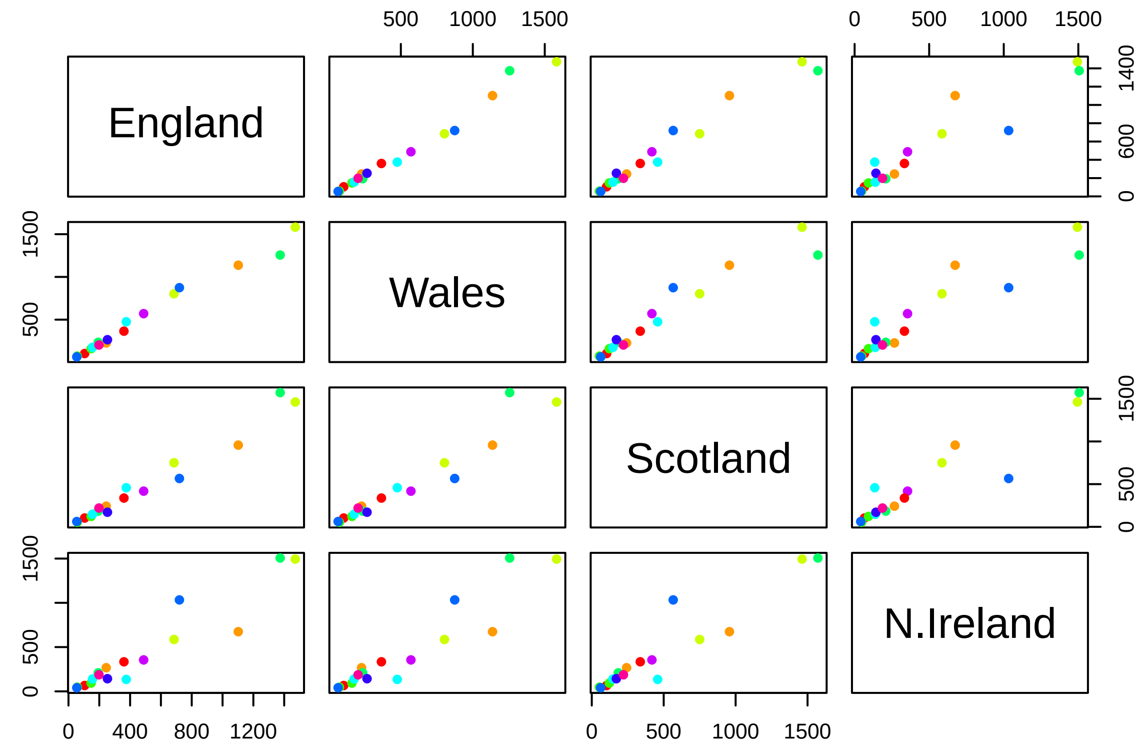
```
barplot(as.matrix(x), beside=FALSE, col=rainbow(nrow(x)))
```



Changing the beside argument can change the plot. Setting beside argument to FALSE creates a stacked boxplot. Leaving the beside argument out would also give the same result because being FALSE is the default. If beside=TRUE were to be added, TRUE would be a numeric vector and includes other components that could be usedul information to the data.

> **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)
```



It is difficult to make sense of this visual dataset. I cannot seem to make out what the figures represent. The axis are very confusing. The diagonal lines appear to show a trend but can't tell what sort of trend.

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

We cannot tell which figures belong to which countries. Does each country have its own respective row or column? Which figures belong to which country?

PCA to the rescue

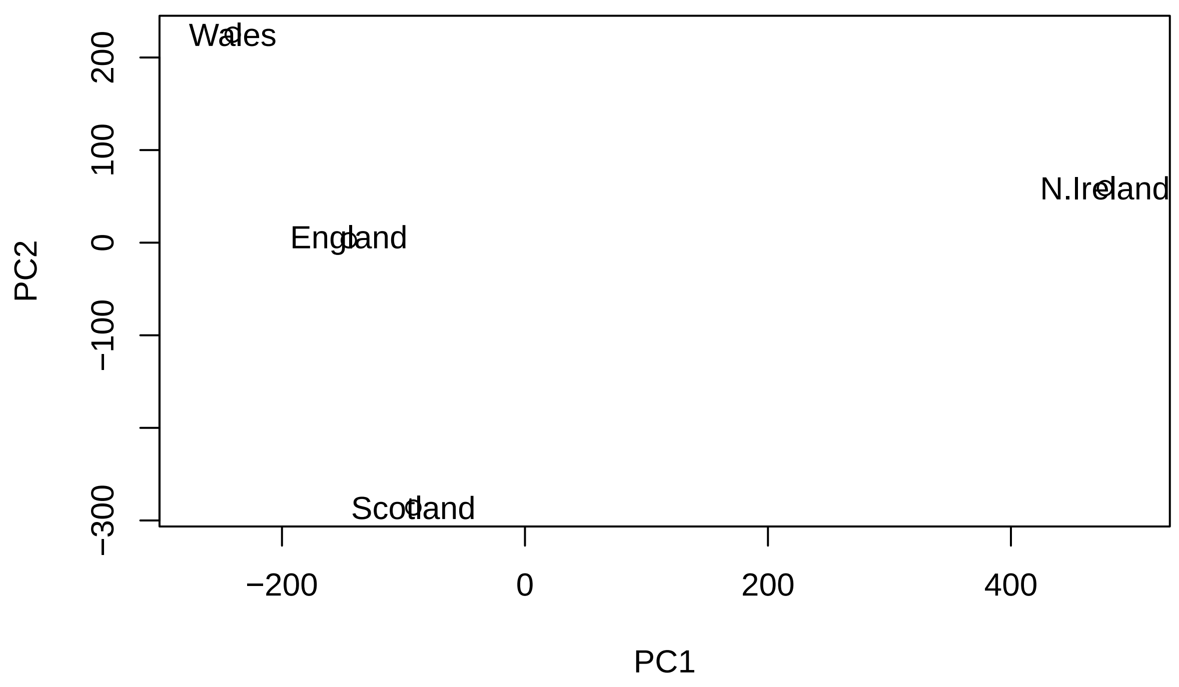
prcomp() expects the observations to be rows and the variables to be columns therefore we need to first transpose our data.frame matrix with the t() transpose function.

```
# Use the prcomp() PCA function
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
```

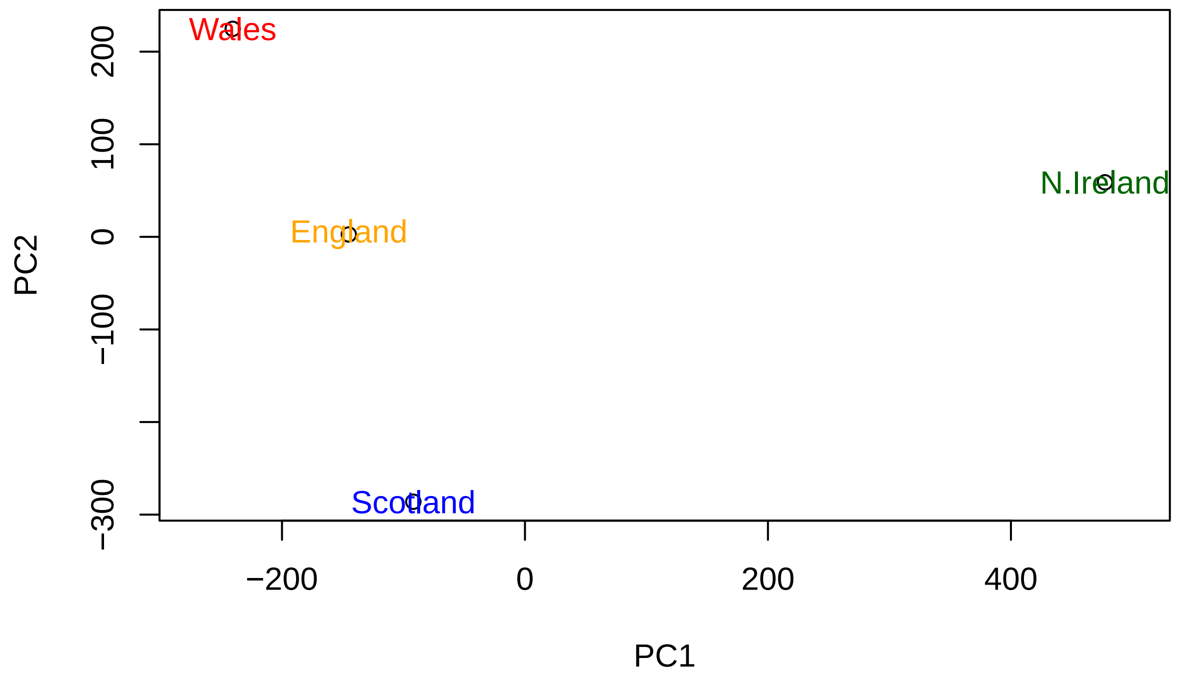
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```r
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
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 Ireland map and table at start of this document.

```r
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=c("orange", "red", "blue", "dark green"))
```



We automatically obtain information about the contributions of each PC to the total variance of the coordinates, which is contained in the Eigenvectors returned from such calculations

We can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for.

For the prcomp() function we can use the summary() command or examine the returned pca$sdev (see below).

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

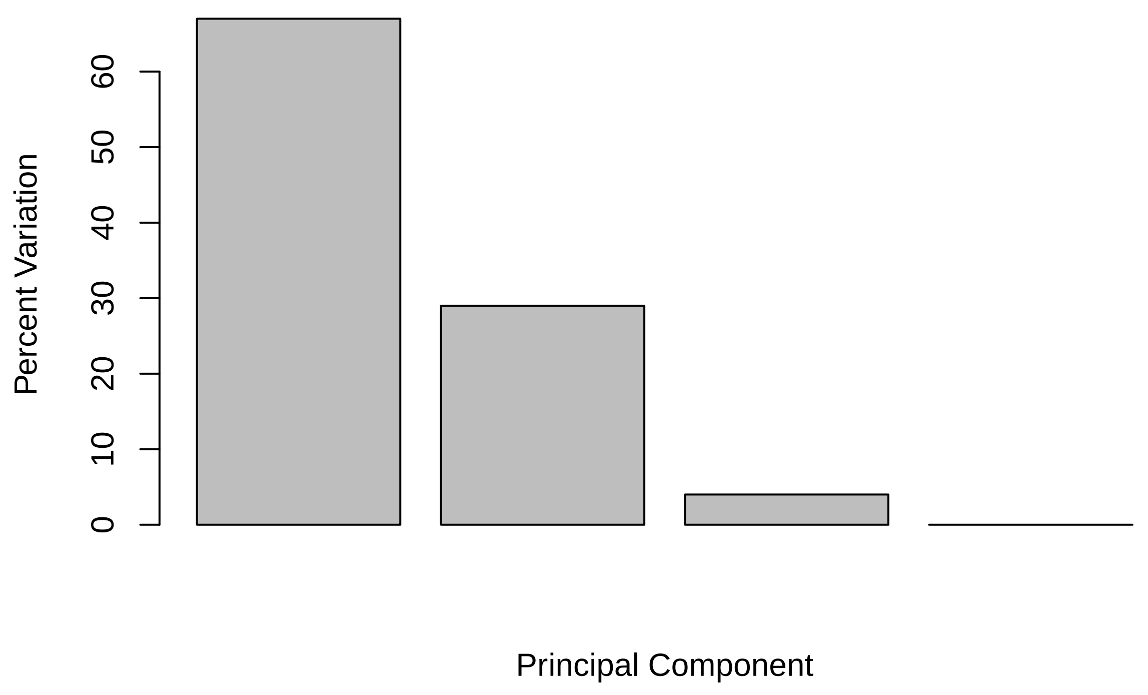
```
## [1] 67 29  4  0
```

```
## summary() command for prcomp function
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
```

This information can be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number)

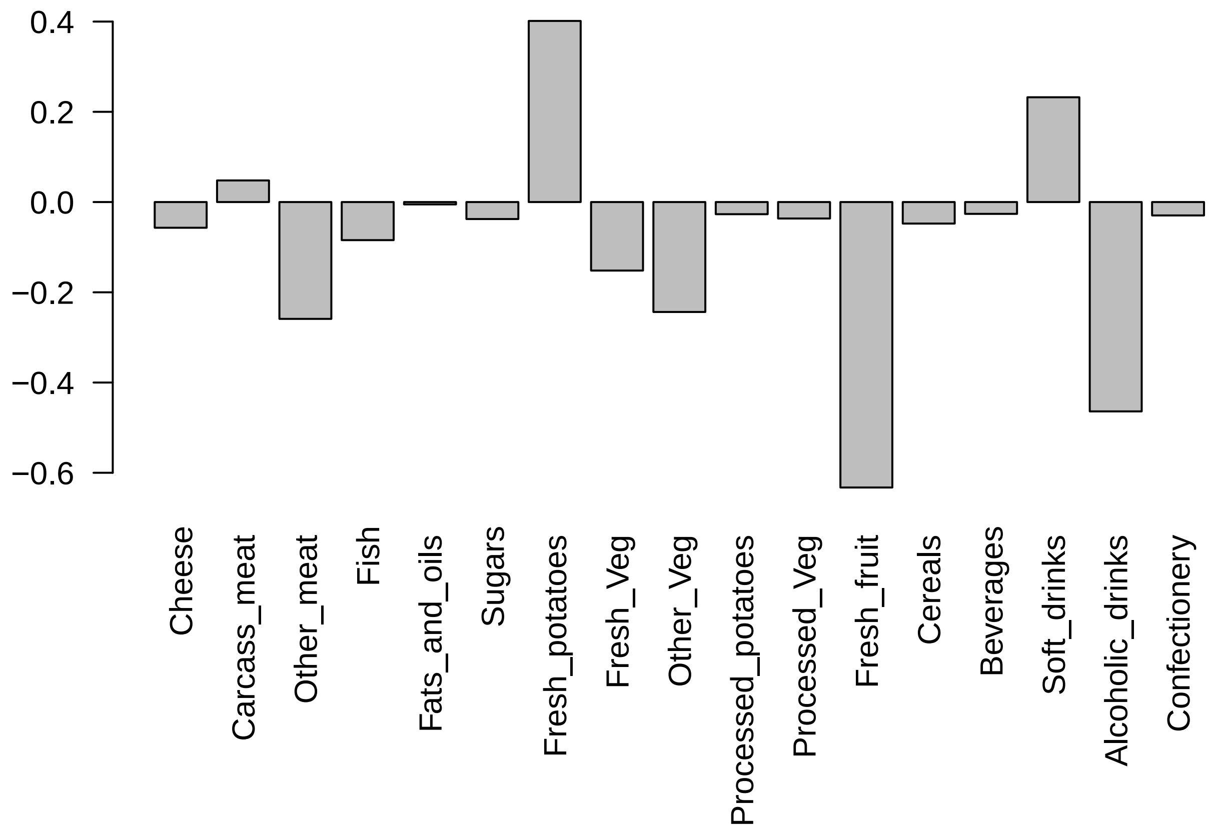
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



Digging deeper variable loadings

Consider the influence of each of the original variables upon the principal components, known as loading scores. We obtain the info using prcomp() component $rotation. Also summarize with a call to biplot()

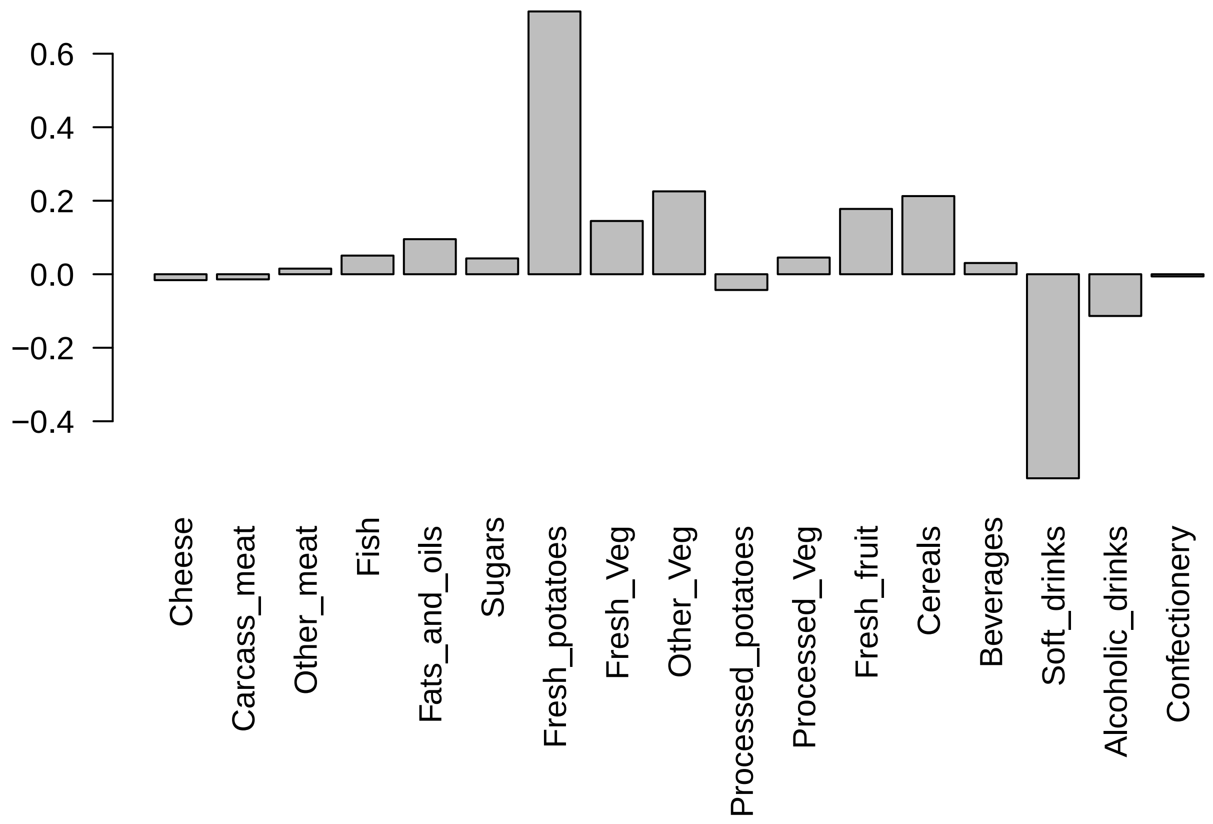
```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



Largest positive loading scores "push" N. Ireland to right positive side of the plot because of Fresh_potatoes and Soft_drinks. High negative scores of Fresh_fruit and Alcoholic_drinks push the other countries to the left side of the plot.

> **Q9:** Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

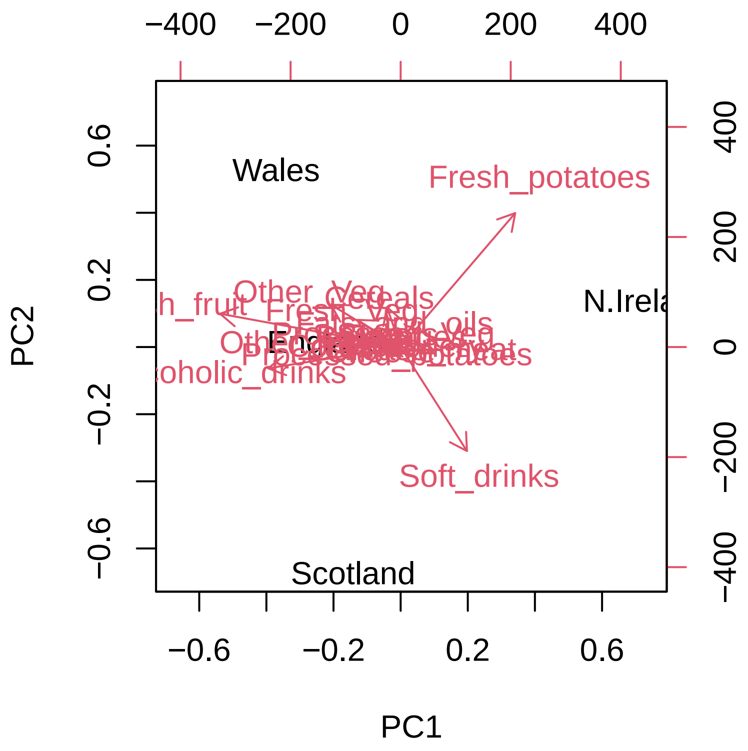
```
## focusing on PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



Negative loading scores include Soft_drinks, Alcholic_drinks, Processes_potatoes. High positive scores include Fresh_potatoes, Other_veg, and Cereals

Another way to see information together with the main PCA plot is in a biplot

```
## The inbuilt biplot() can be useful for small datasets
biplot(pca)
```



There is a central group of foodsin the middle of each PC, with a few that are far from the group. This could be representative of how England, Wales and Scotland were clustered together being "similar", whilst Northern Ireland was the country that was away from the cluster.

Now we will look at the PCA of RNA-seq data

```r
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
```

```r
View(rna.data)
```

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

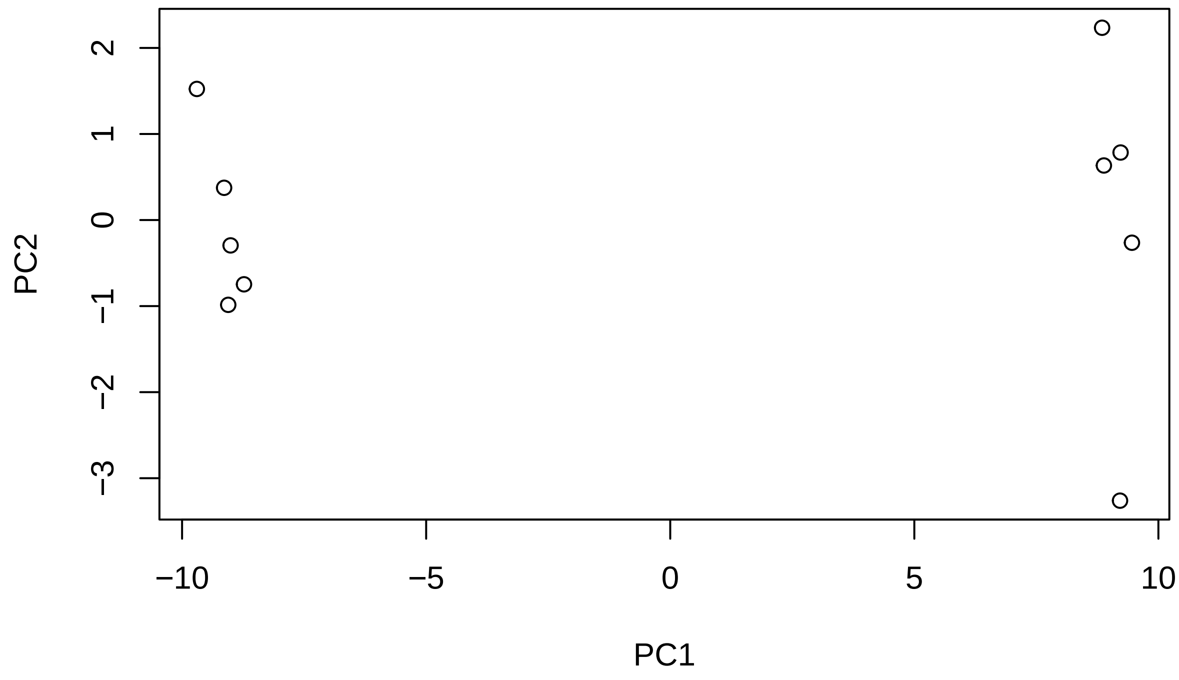
```
dim(rna.data)
```

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

There are 100 genes and 10 samples in this data set.

Do a PCA and plot results rather than use any other plots

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

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



Let's examine a summary of how much variation in the original data each PC accounts for

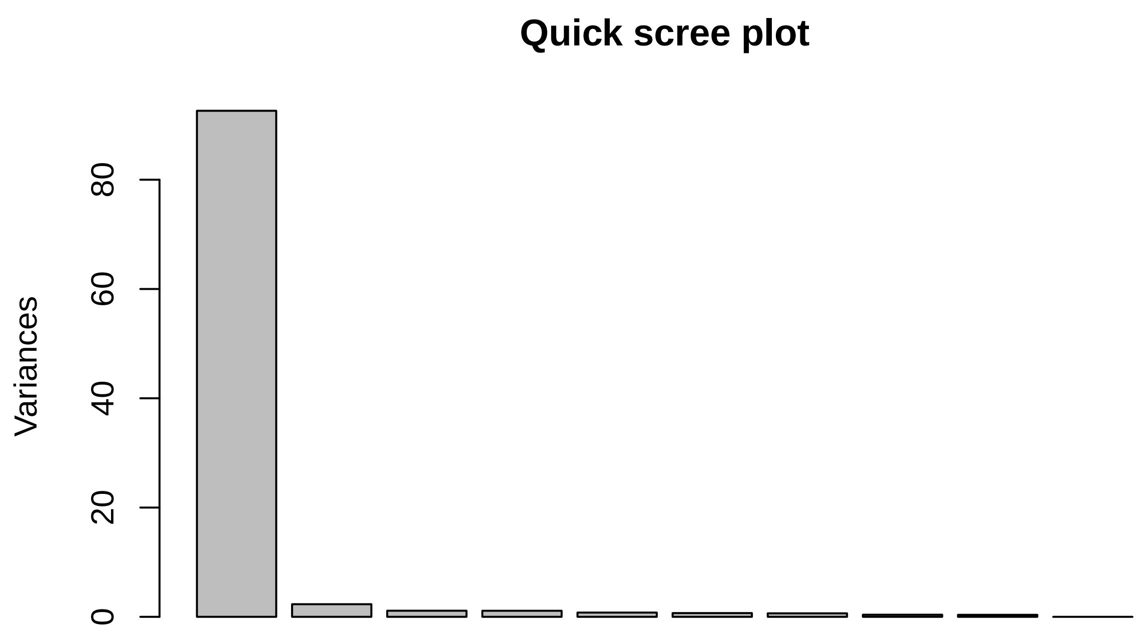
```
summary(pca)
```

```
## 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
```

By viewing the cumulative proportion, we see that PC1 captures 92.6% of the action. This means that we have successfully reduced a 100 diminesional data set down to only one dimension that retains 92.6% of the essential features from the original data. PC1 captures 92.6%, PC1 and PC2 together captures 94.9%.

Quick barplot summary of this Proportion of Variance for each PC

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



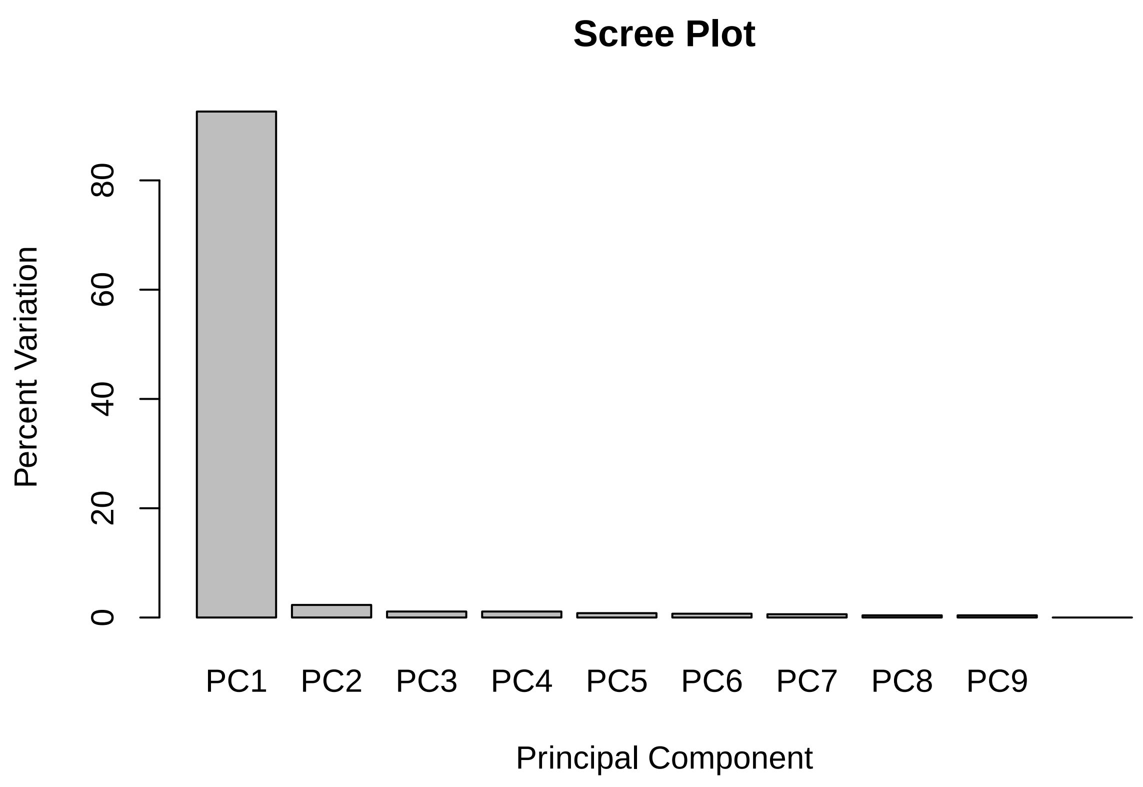
Square of pca$sdev to calculate how much variation in the original data each PC accounts for

```
## Variance captured per PC
pca.var <- pca$sdev^2

## Percent variance is often more informative to look at
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
```

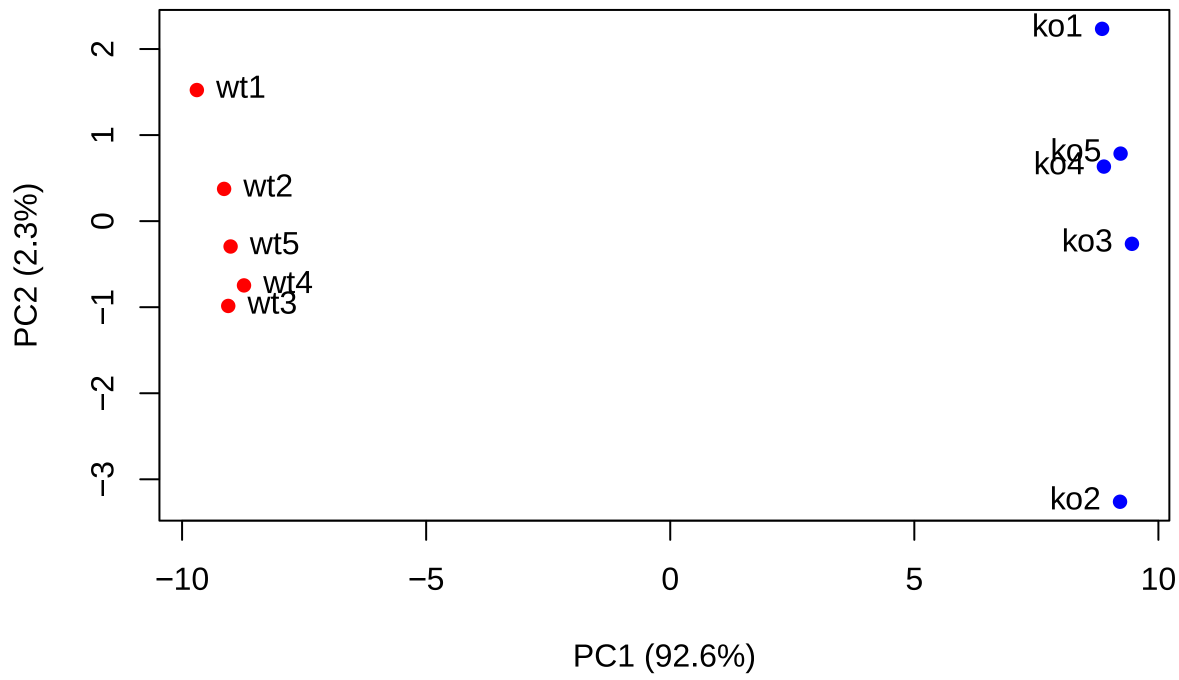
Generate another scree plot

```r
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



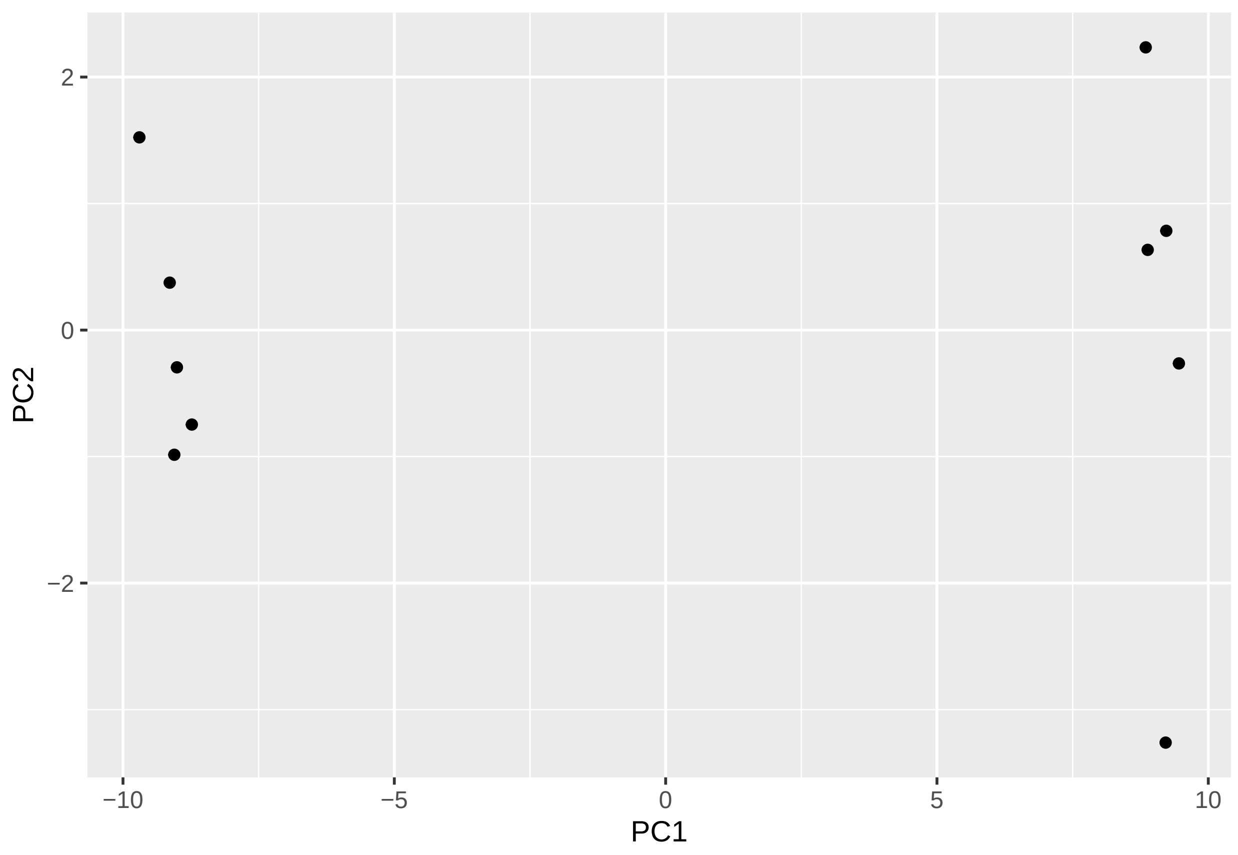
Making the plot appear more useful

```r
## A vector of colors for wt and ko samples
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)))
```



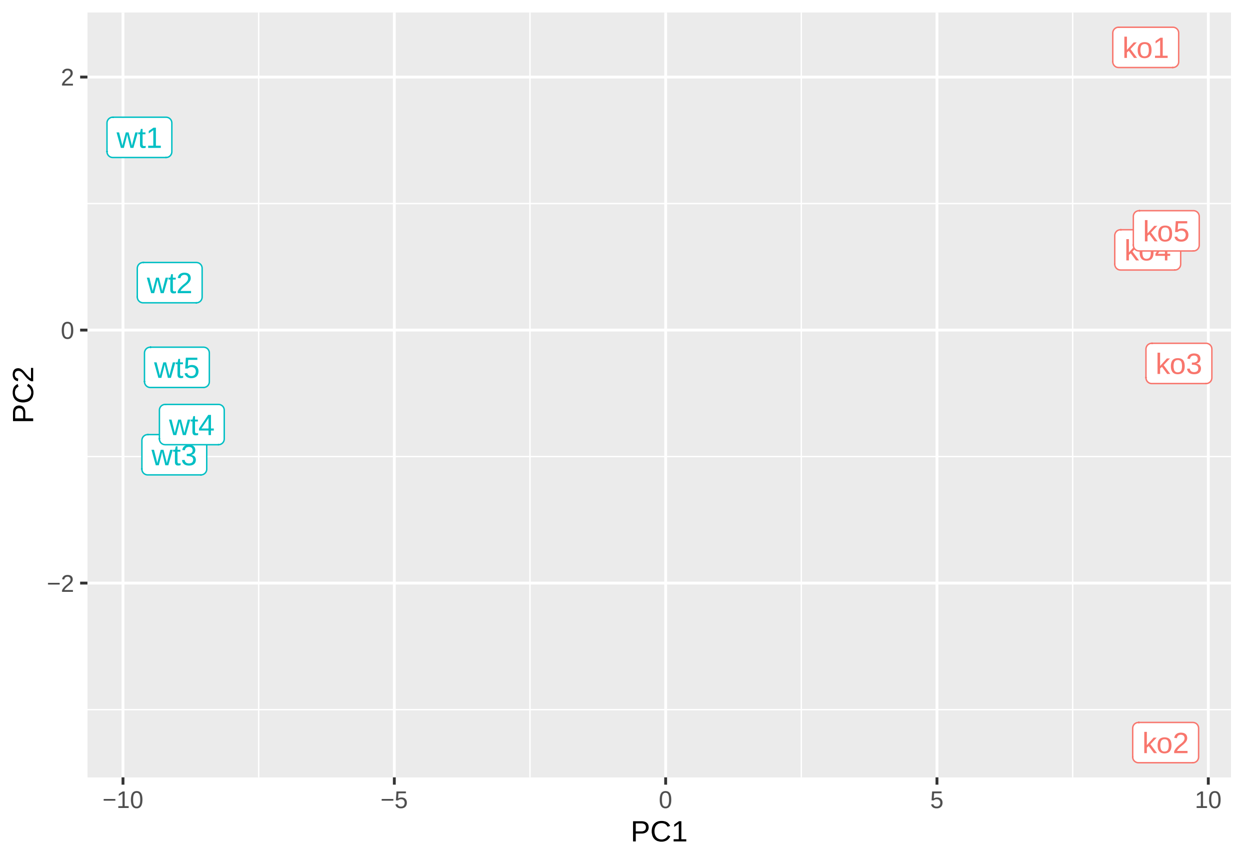
Using ggplot

```r
# Prep ggplot
library(ggplot2)
# Contain PCA results in dataframe
df <- as.data.frame(pca$x)
# Our first basic plot
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



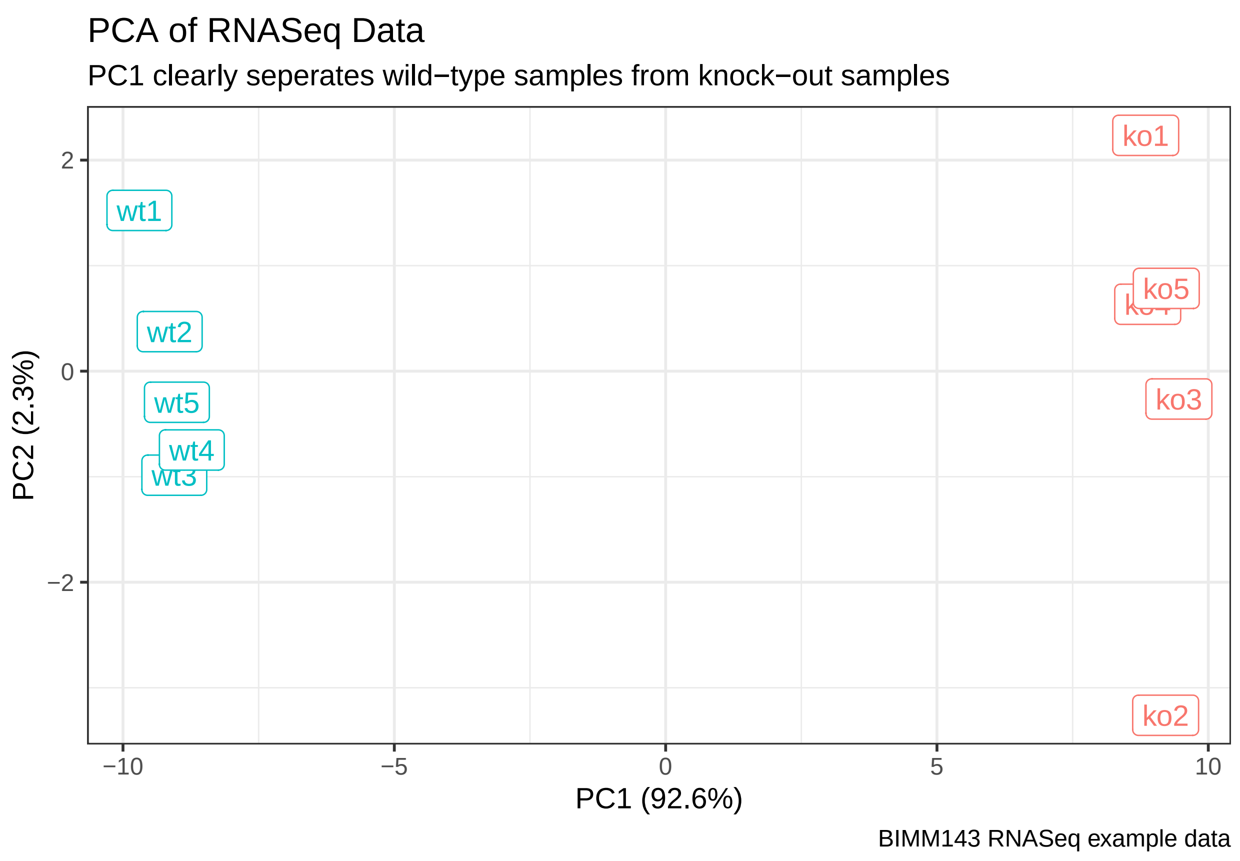
Adding some aesthetics and color

```
# 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
```



More polishing to the plot

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



Let's find the top 10 measurements (genes) that contribute most to pc1 in either direction

```r
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"
```