Machine Learning

Tom Quach (A15549142)

10/21/2021

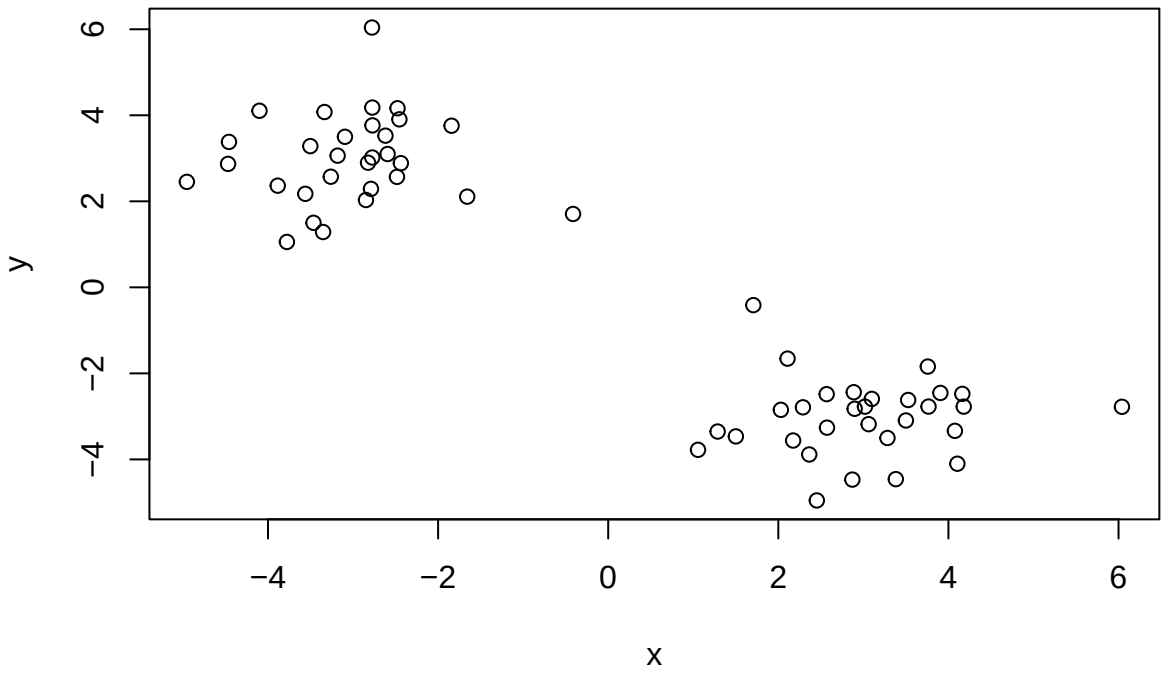
First up is clustering methods

#Kmeans clustering

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

First make up some data where we know what the answer should be:

```
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 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  2.987637 -3.031157
## 2 -3.031157  2.987637
##
## 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 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
##
## Within cluster sum of squares by cluster:
## [1] 55.19042 55.19042
##  (between_SS / total_SS =  90.8 %)
##
## 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 size?

```
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 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
```

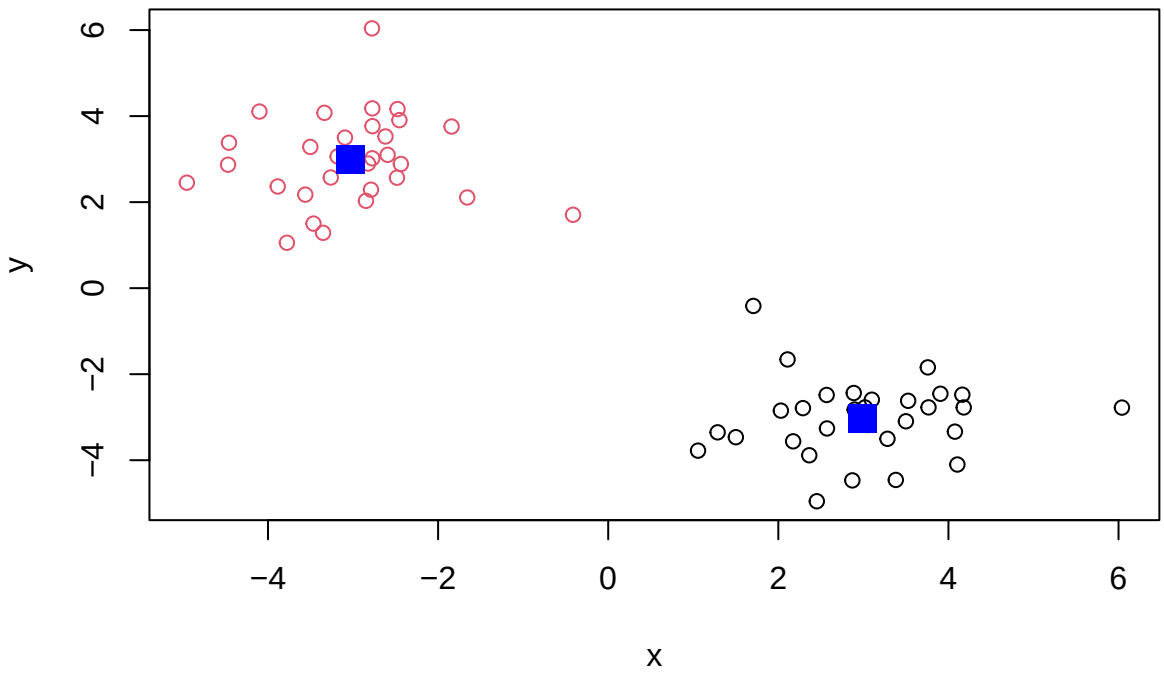
- cluster assignment/membership?
- cluster center?

```
km$centers
```

```
##           x         y
## 1  2.987637 -3.031157
## 2 -3.031157  2.987637
```

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

```
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()

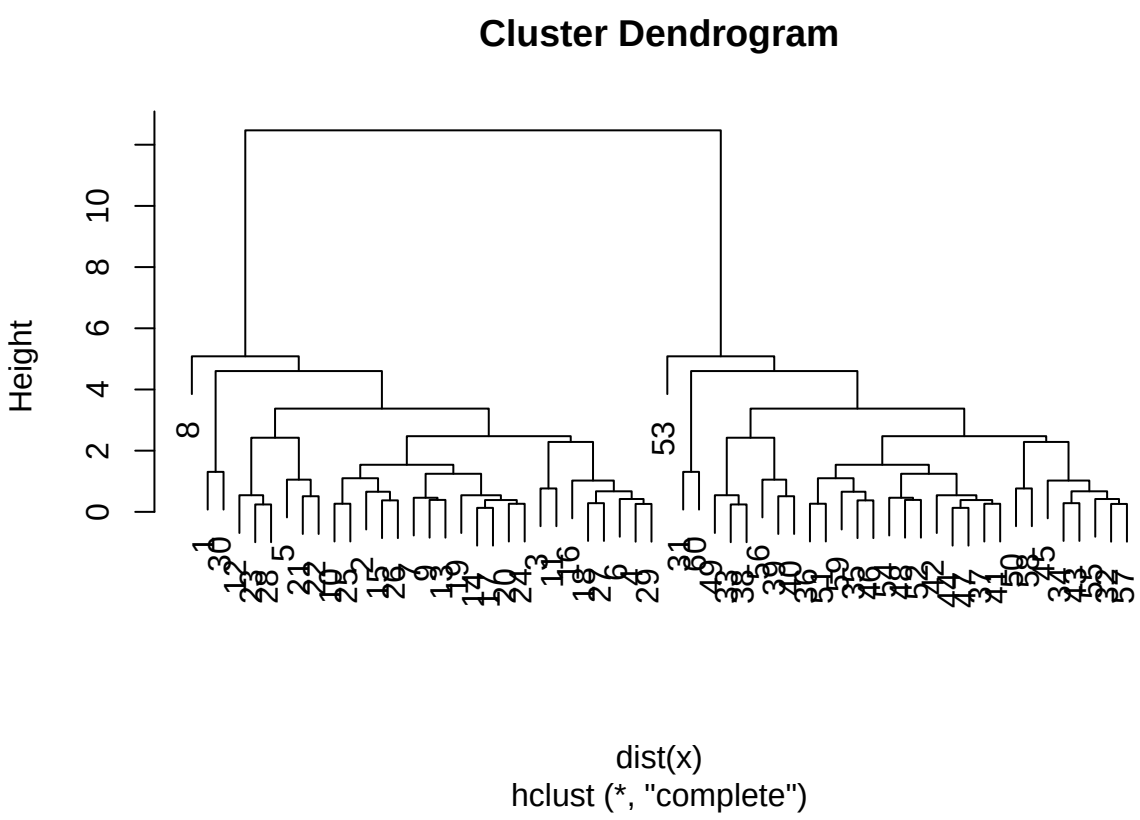
Demonstrate the use of dist(), hclust(), plot(), and cutree(), functions to do clustering. Generate dendrograms and return cluster assignment membership vector

```
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=6)
```

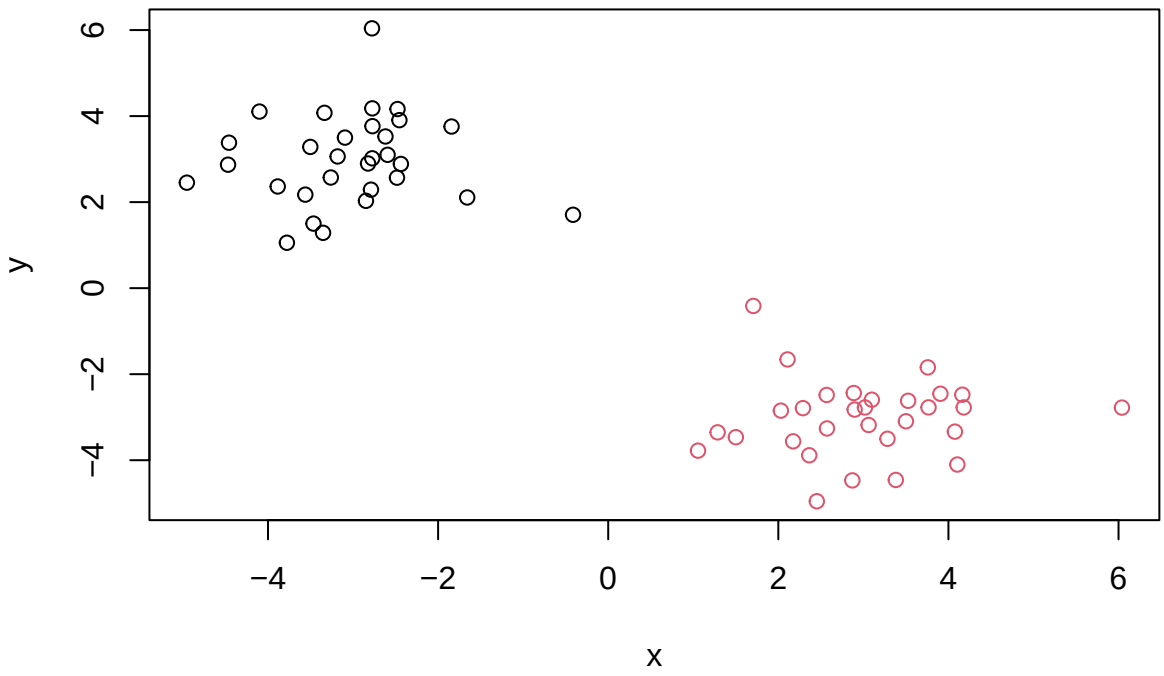
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

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

Make our results plot

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



##PCA Lab Report

Importing and reading the data

```
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 questions?

```
dim(x)
```

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

> 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?

We can see that there are 17 rows and 5 columns in my new data frame x

There should only be a 17 x 4 dimensions. We have an extra first column that we need to fix

```
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
```

As we run the code each time, we are deleting more data from our data frame which is really dangerous

```
url <- "https://tinyurl.com/UK-foods"
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
```

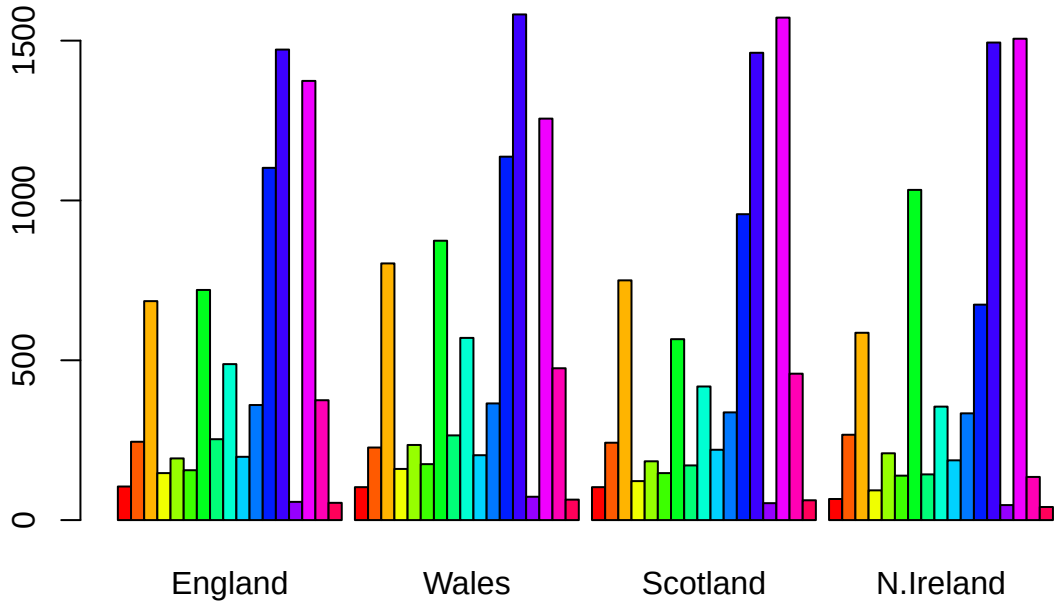
Rechecking the dimension of the data fram

```
dim(x)
```

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

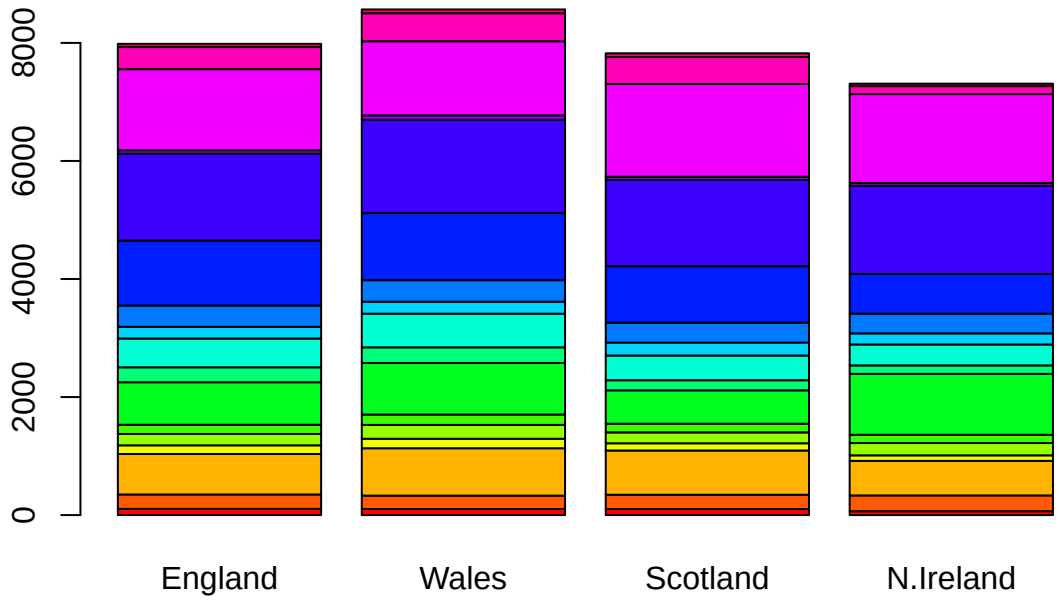
Perfect! The better way seems to be the latter code we just performed, 'row.name()=1'. The first method will delete our data each time we run it

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



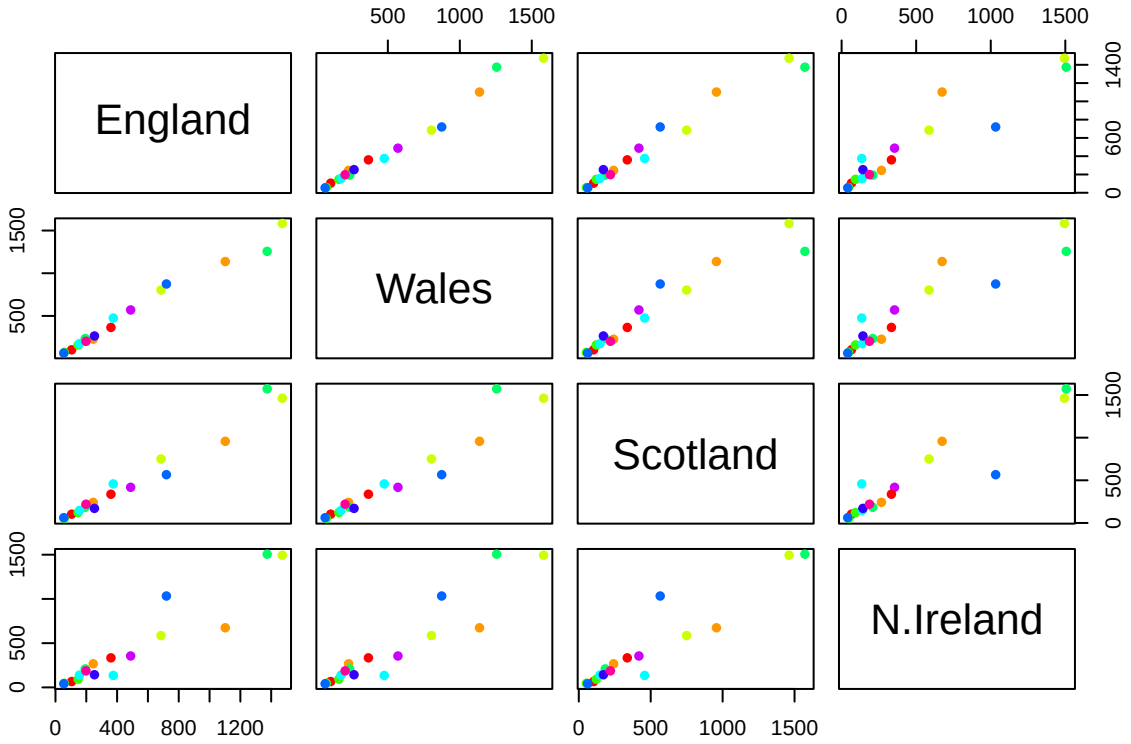
Q3: Changing what optional argument in the above barplot() function results in the following plot?

```
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)
```



Each individual graphs plot two countries against one another in the x and y axis. The y axis show a constant country while the x axis has different countries. If the point lies in the diagonal line, that means there is less variance of that specific point between two countries

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

For the other countries, the points are more evenly distributed along an imaginary diagonal line. However, the data set with N. Ireland is more clustered towards the bottom left of each data graph.

#PCA to the rescue

The main function in base R for PCA is 'prcomp()'

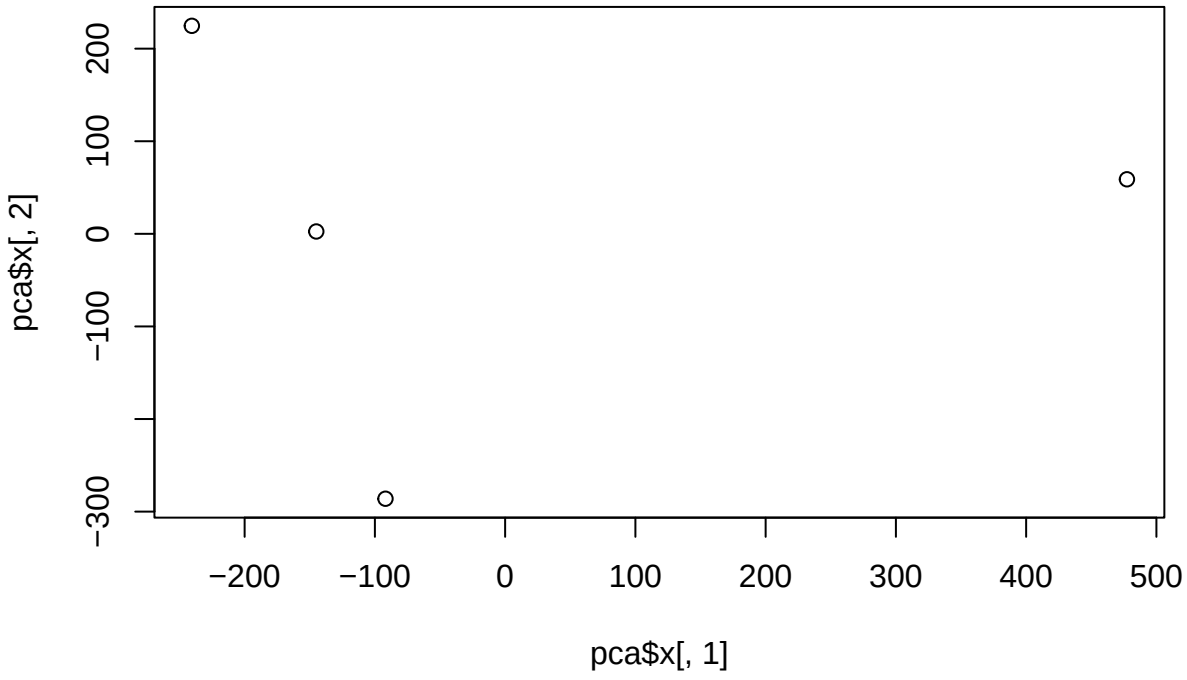
```
pca <- prcomp(t(x))
summary(pca)
```

```
## 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
```

```
attributes(pca)
```

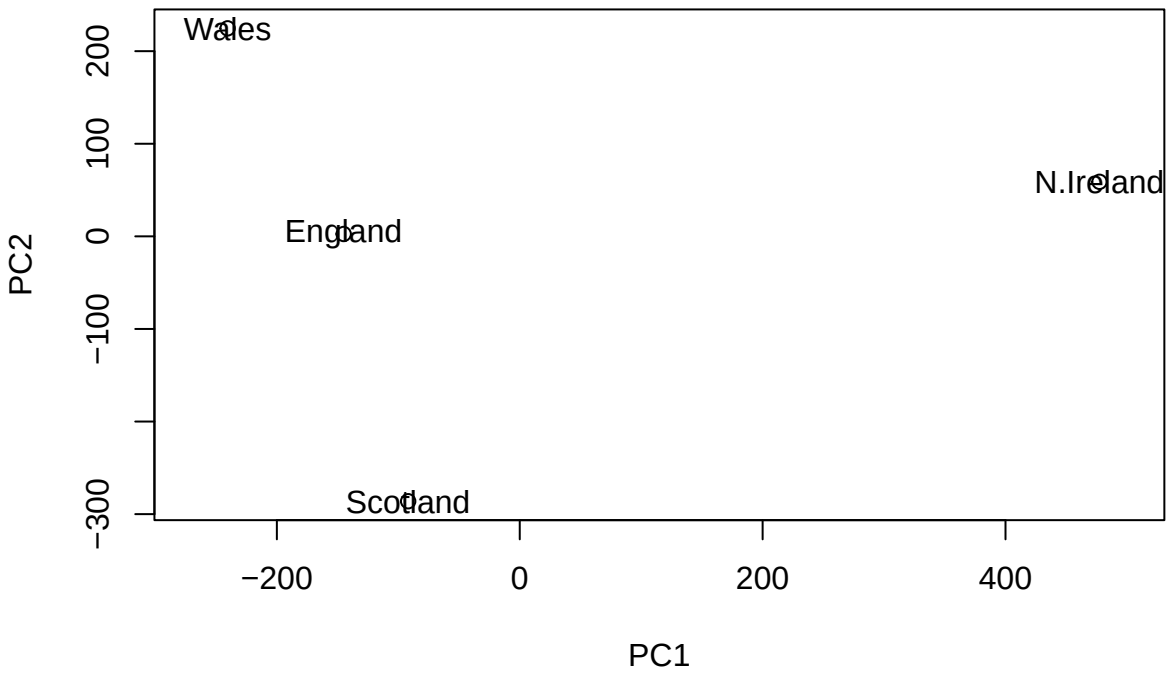
```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

```
plot(pca$x[,1], pca$x[,2])
```



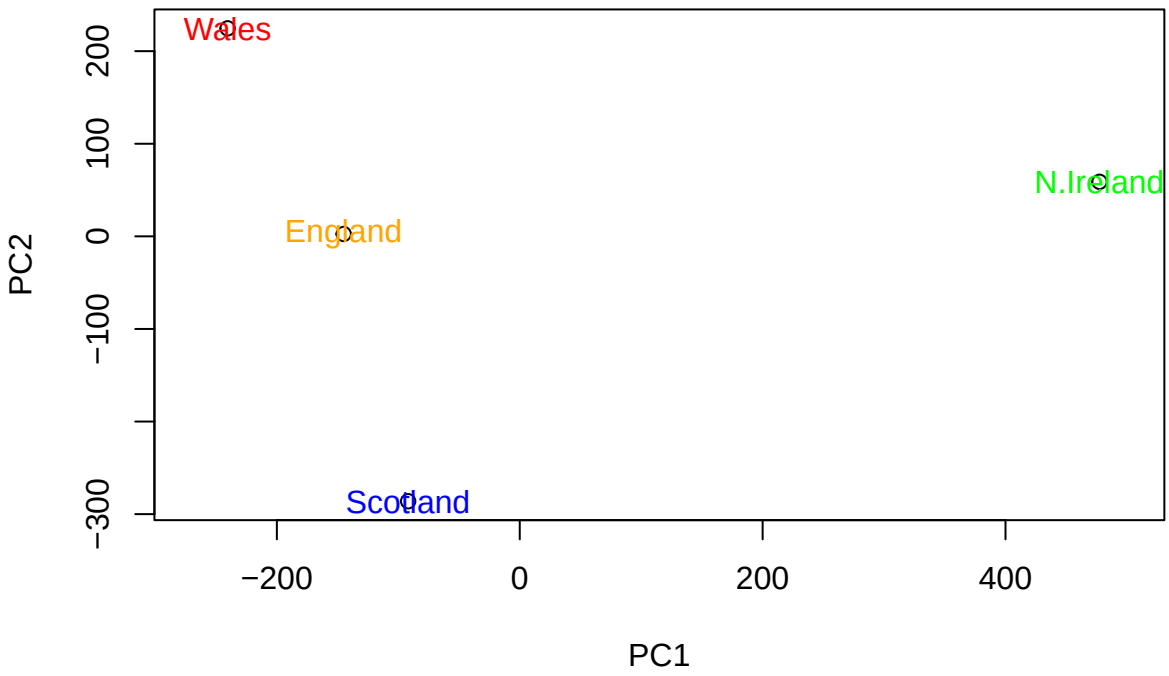
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
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.

```
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", "Green"))
```



We are now finding the standard deviation of our data frame

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

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

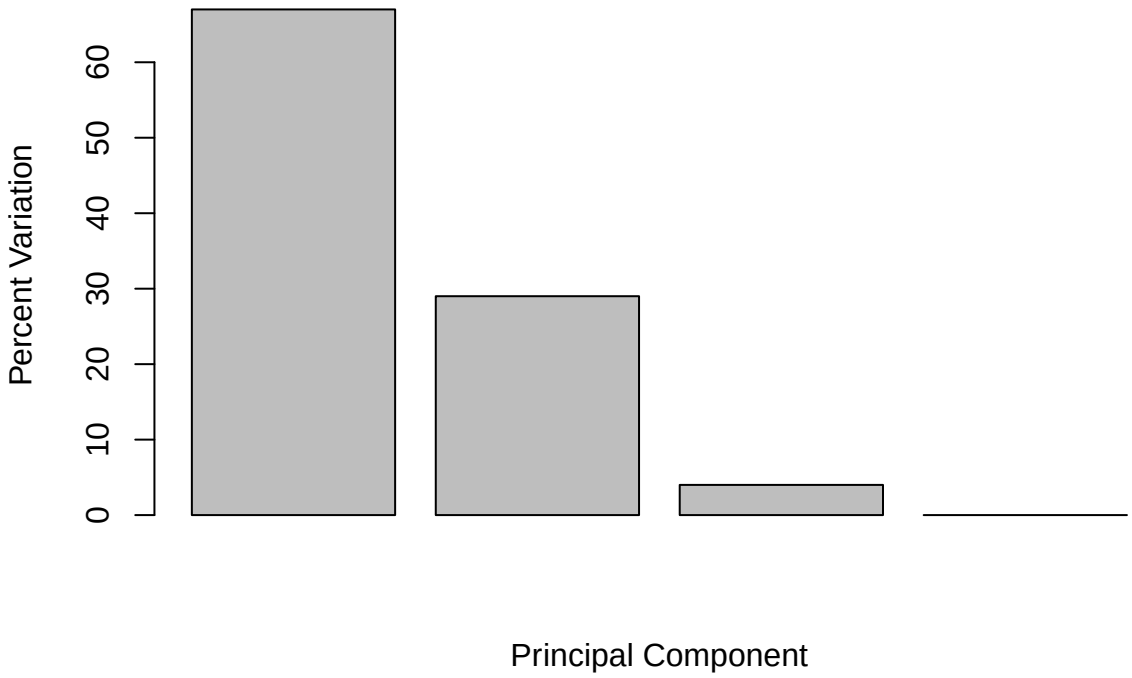
We are now going to store the important components of pca into a vector

```
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
```

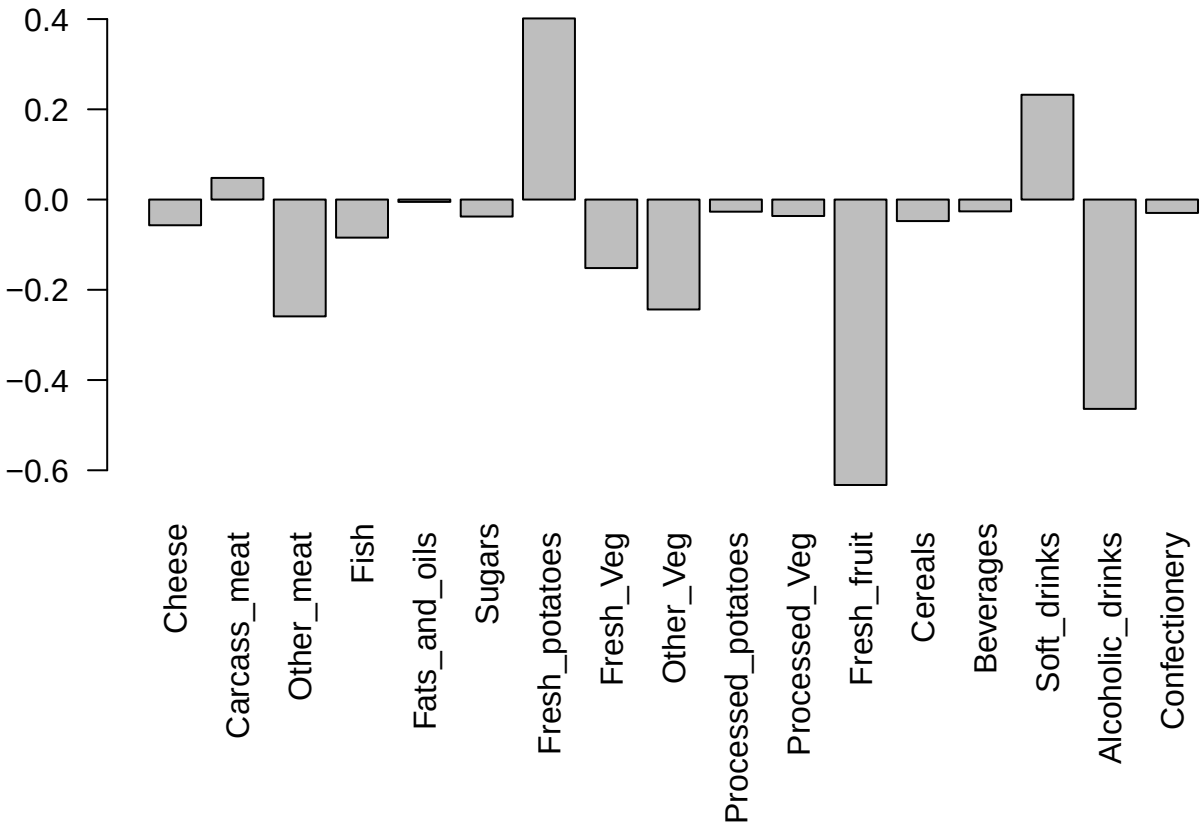
Creating a plot to see which principal component has the most variance

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



Generating a loading plot for pc1 to see what is causing the high vairance

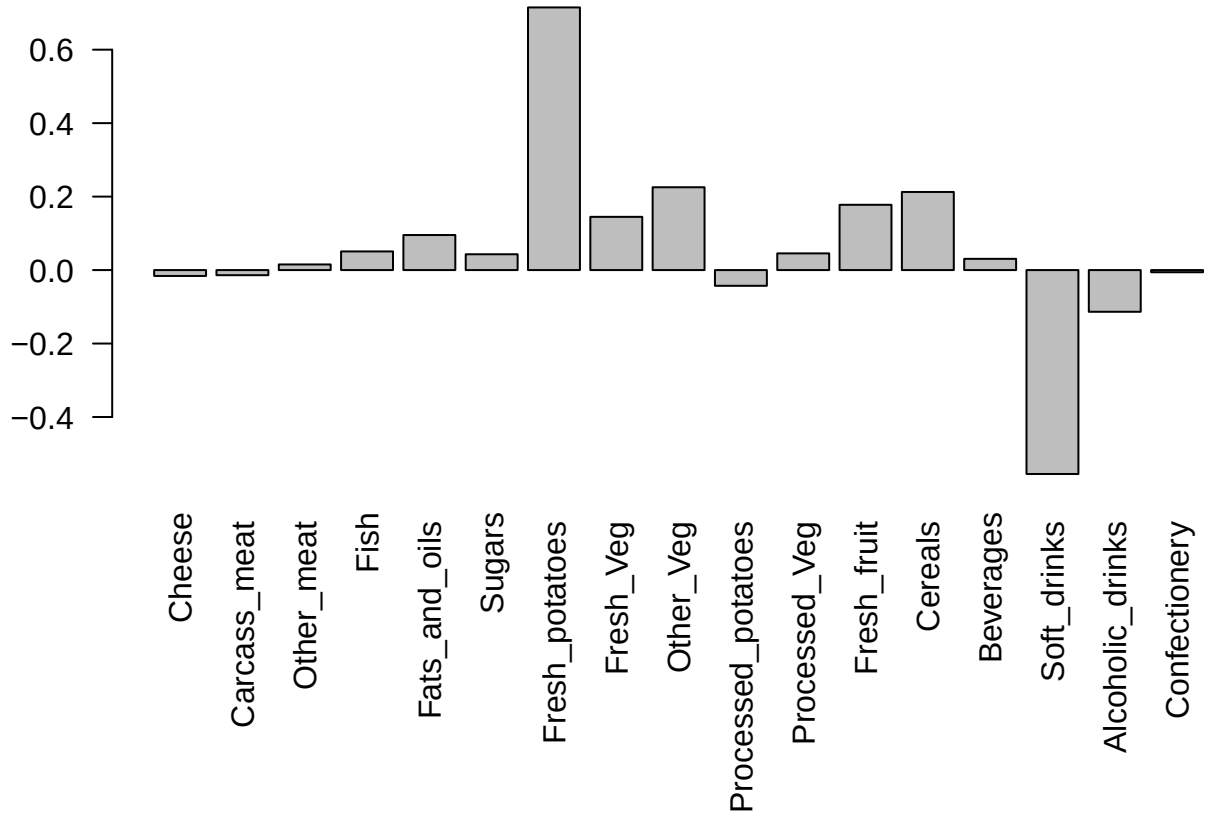
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

Generating a loadings plot for pc2

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



The two food groups feature prominately are fresh potatoes and soft drinks. This means that soft drinks, a more negative loading score, are pushing countries towards the bottom side of the plot. This also means that fresh potatoes, a more positive loading score, is pushing countires towards the upper side of the plot

##PCA of RNA

Inserting and reading RNA-sequence file

```
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?

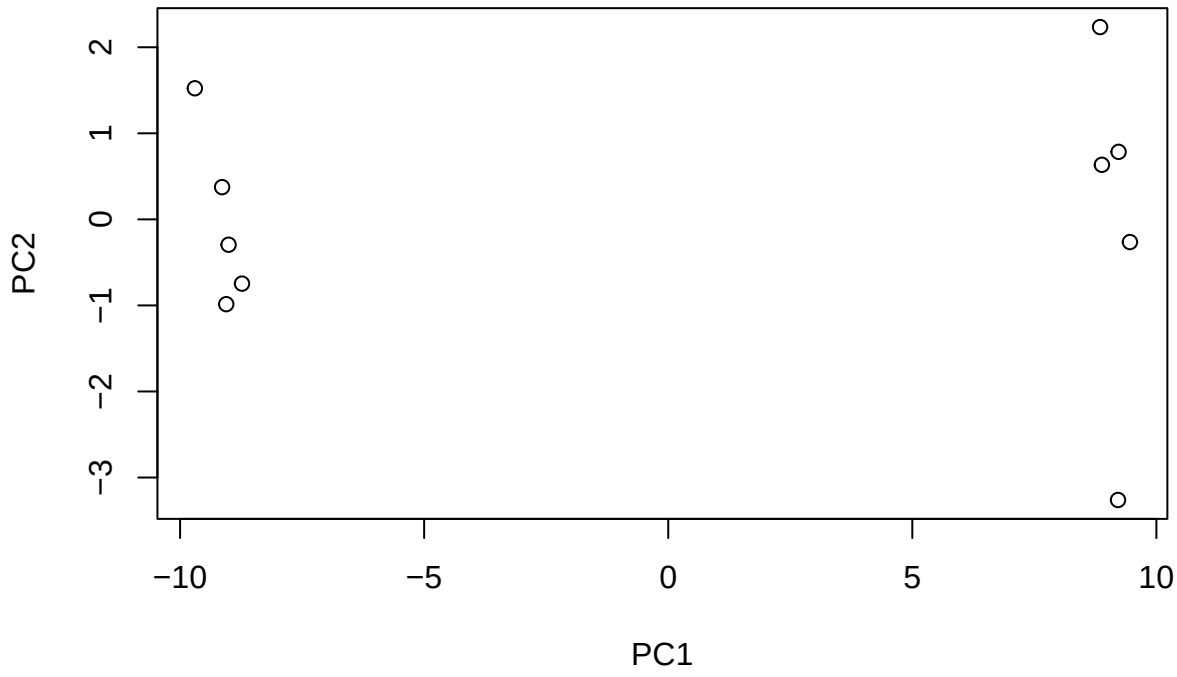
```
dim(rna.data)
```

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

There are a 100 genes and 10 samples

Generating a barplot

```
## 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")
```



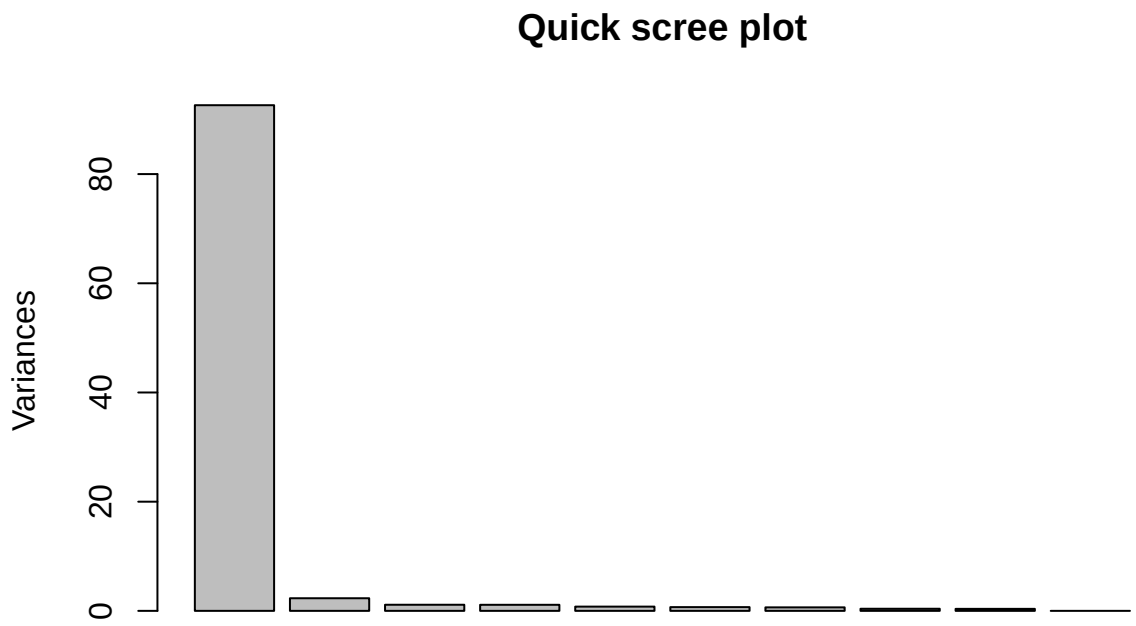
Getting the important details from the pca of RNA-seq

```
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
```

Creating a plot to see which principal componet has what variance

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



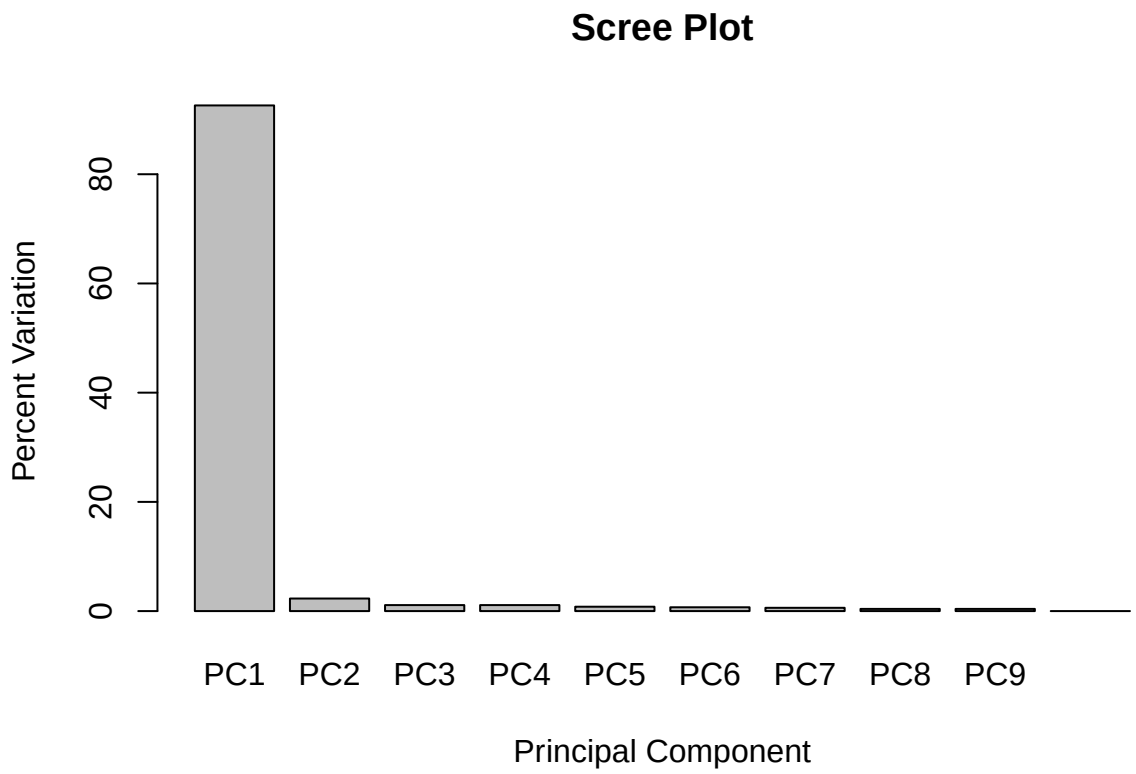
Looking at the exact value of the variance for each principle component

```
## 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
```

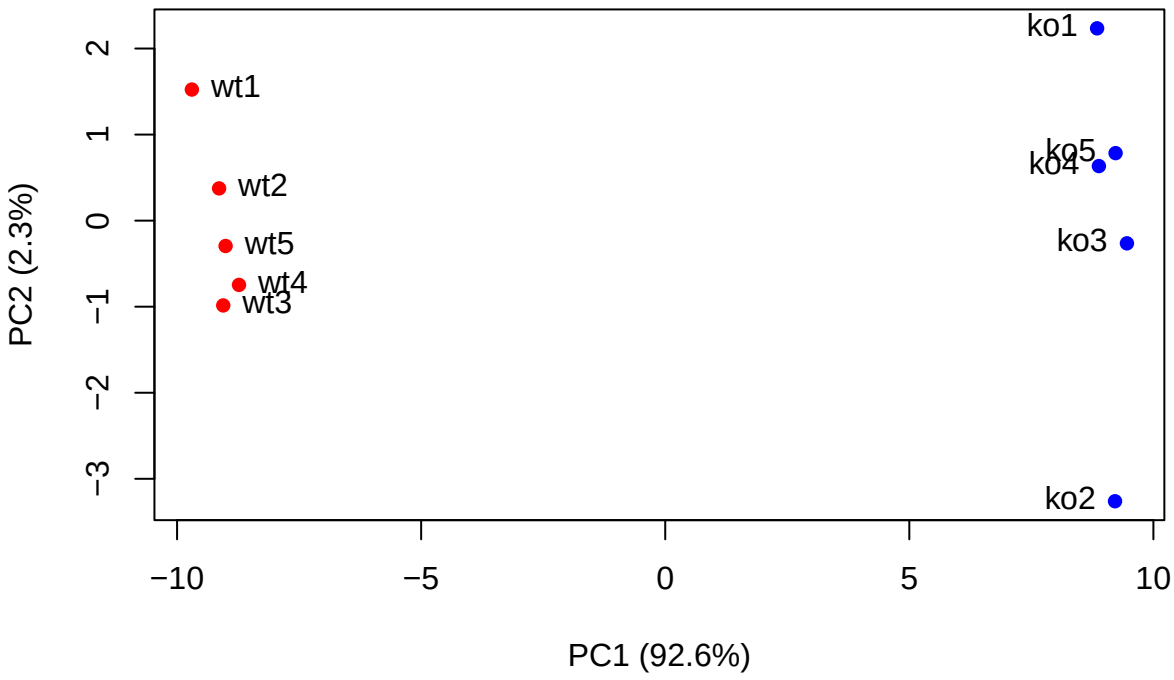
Titling our bar plot

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



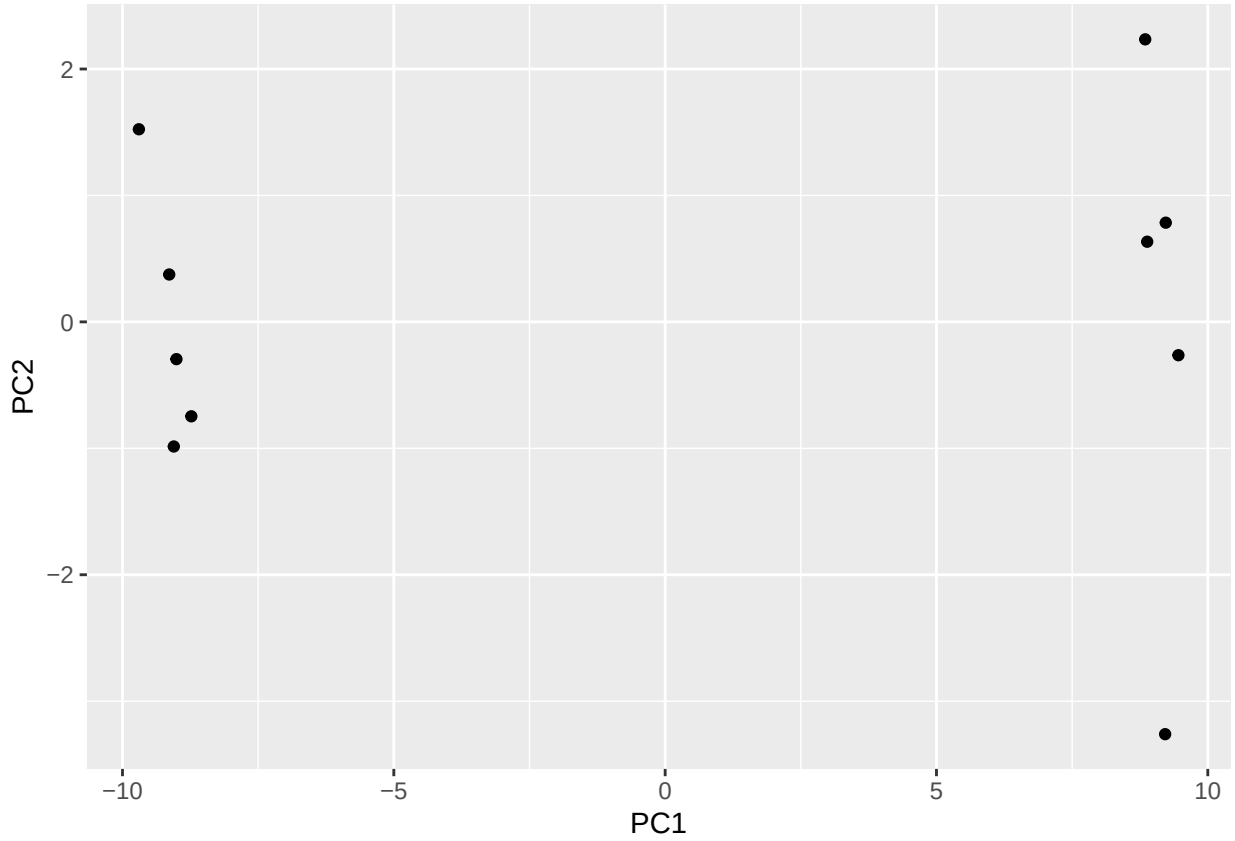
Plotting our PC1 and PC2 against each other to see the majority of the variance more clearly

```
## 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)))
```



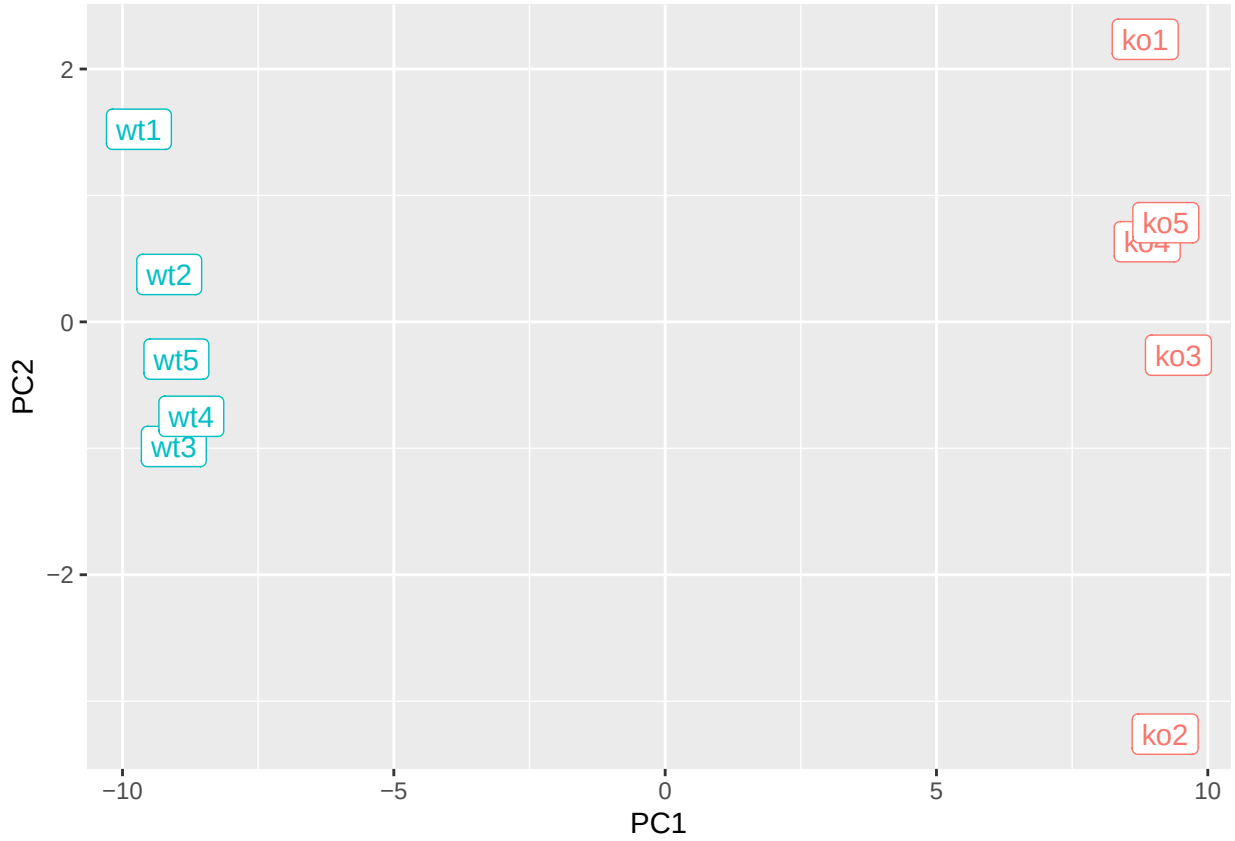
Using ggplot to create a scatter plot of the rna-sequence data

```
library(ggplot2)

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

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



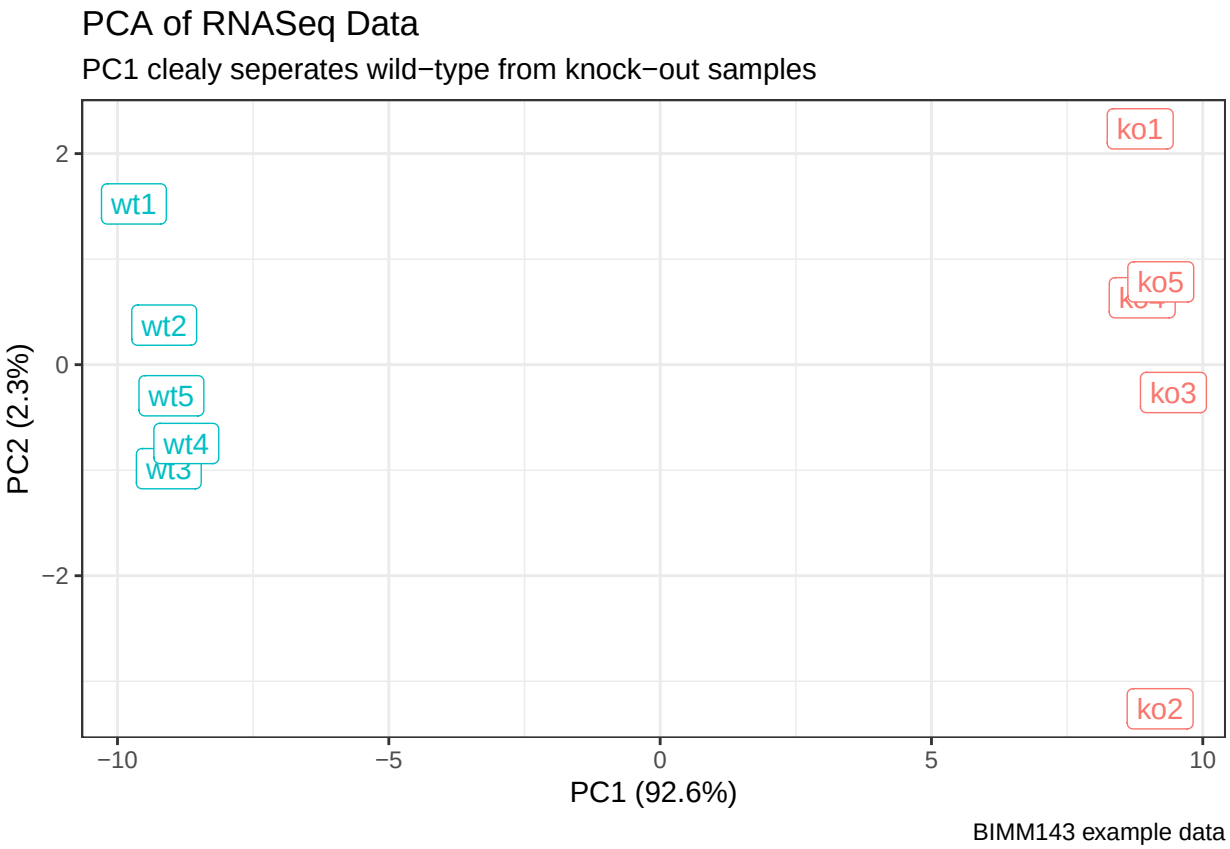
Naming each point to see if they are wild type or knock out

```
# 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
```



Labeling my ggplot

```
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()
```


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
ko1
wt1
ko5
wt2
ko3
wt5
wt4
wt3
ko2
2
0
−2
PC2 (2.3%)
−10
−5
0
5
10
PC1 (92.6%)
BIMM143 example data