Machine Learning 1

Camryn McCann (PID: A15437387)

10/21/2021

First up is clustering methods

#Kmeans clustering

The functon in base R to do Kmeans clustering is called *kmeans()*

First, generate some example data for clustering where we know what the answer should be

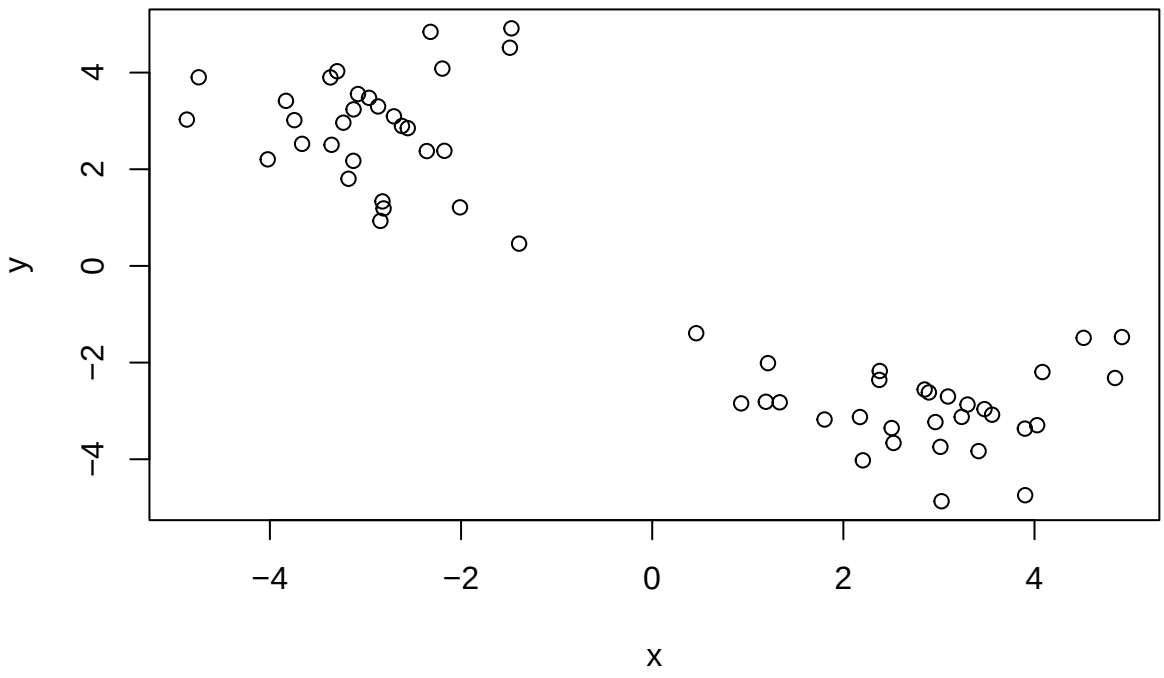
```
tmp <- c(rnorm(30,-3), rnorm(30,3))
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
##                x          y
##  [1,] -2.1742070  2.3807519
##  [2,] -3.0784496  3.5559357
##  [3,] -2.5577487  2.8503556
##  [4,] -3.1270594  2.1736156
##  [5,] -1.3926957  0.4603820
##  [6,] -3.2313542  2.9629990
##  [7,] -4.0227508  2.2042620
##  [8,] -3.7442592  3.0152781
##  [9,] -3.3669762  3.8994011
## [10,] -3.1245969  3.2382141
## [11,] -2.6182357  2.8944476
## [12,] -2.3585253  2.3760859
## [13,] -4.8690284  3.0274682
## [14,] -3.2959126  4.0276008
## [15,] -2.8225194  1.3343611
## [16,] -1.4725223  4.9148719
## [17,] -2.3194169  4.8424127
## [18,] -2.8106851  1.1891939
## [19,] -2.0111723  1.2111208
## [20,] -2.1958061  4.0827237
## [21,] -2.9627681  3.4765926
## [22,] -2.7021794  3.0945688
## [23,] -3.3546605  2.5054114
## [24,] -3.6634410  2.5251223
## [25,] -1.4887020  4.5140994
## [26,] -4.7446899  3.9022275
## [27,] -2.8682901  3.2991067
## [28,] -3.1776548  1.8029560
## [29,] -2.8439938  0.9308418
## [30,] -3.8317273  3.4149423
```

```
## [31,]  3.4149423 -3.8317273
## [32,]  0.9308418 -2.8439938
## [33,]  1.8029560 -3.1776548
## [34,]  3.2991067 -2.8682901
## [35,]  3.9022275 -4.7446899
## [36,]  4.5140994 -1.4887020
## [37,]  2.5251223 -3.6634410
## [38,]  2.5054114 -3.3546605
## [39,]  3.0945688 -2.7021794
## [40,]  3.4765926 -2.9627681
## [41,]  4.0827237 -2.1958061
## [42,]  1.2111208 -2.0111723
## [43,]  1.1891939 -2.8106851
## [44,]  4.8424127 -2.3194169
## [45,]  4.9148719 -1.4725223
## [46,]  1.3343611 -2.8225194
## [47,]  4.0276008 -3.2959126
## [48,]  3.0274682 -4.8690284
## [49,]  2.3760859 -2.3585253
## [50,]  2.8944476 -2.6182357
## [51,]  3.2382141 -3.1245969
## [52,]  3.8994011 -3.3669762
## [53,]  3.0152781 -3.7442592
## [54,]  2.2042620 -4.0227508
## [55,]  2.9629990 -3.2313542
## [56,]  0.4603820 -1.3926957
## [57,]  2.1736156 -3.1270594
## [58,]  2.8503556 -2.5577487
## [59,]  3.5559357 -3.0784496
## [60,]  2.3807519 -2.1742070
```

Let's plot it now!

```
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 -2.941068  2.870245
## 2  2.870245 -2.941068
##
## 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 2 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
##
## Within cluster sum of squares by cluster:
## [1] 58.13368 58.13368
##  (between_SS / total_SS =  89.7 %)
##
## 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/membersip?

```
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 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
```

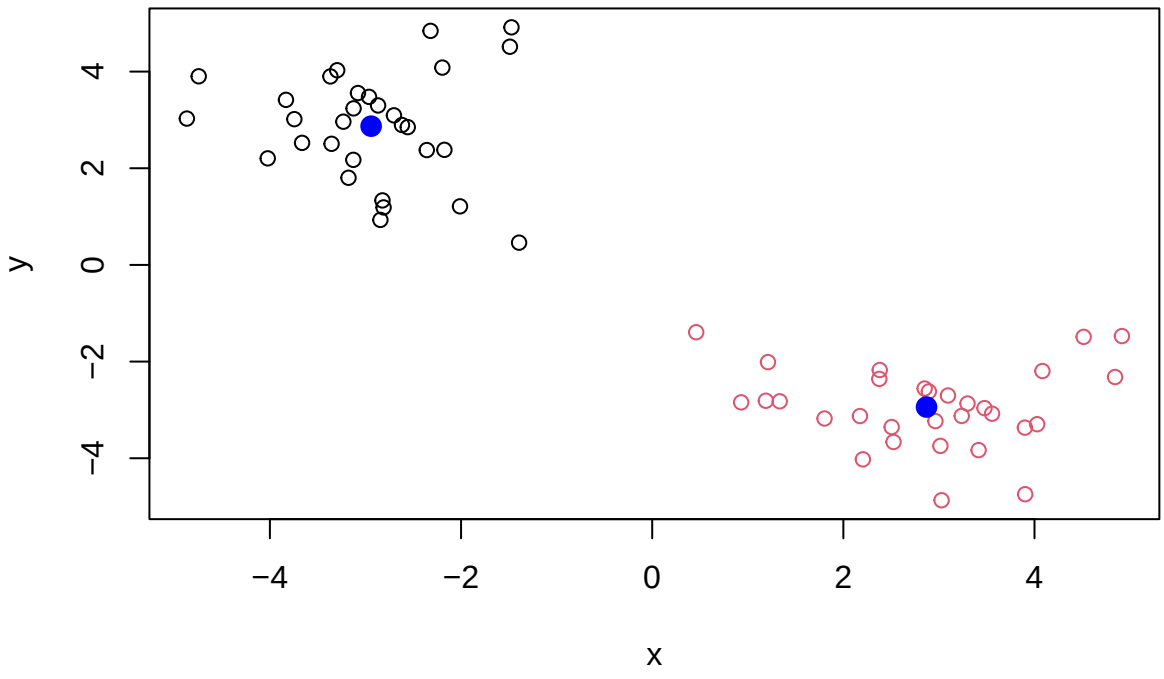
What 'component' of your result object details cluster centers?

```
km$centers
```

```
##           x         y
## 1 -2.941068  2.870245
## 2  2.870245 -2.941068
```

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

```
plot(x, col= km$cluster)
points(km$centers, col = "blue", pch = 16, cex = 1.5)
```



A big limitation with kmeans is that we have to tell it "k" (number of clusters we want).

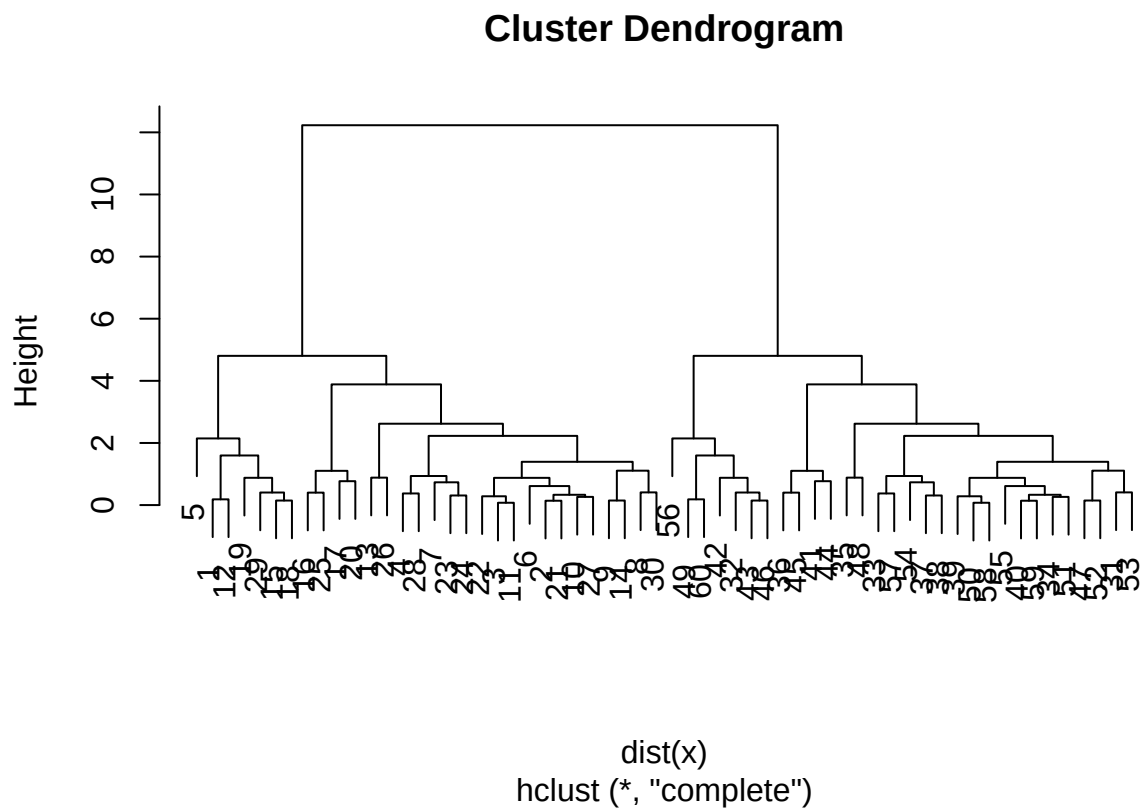
#hclust clustering Now, lets use the same data with *hclust()* We will demonstrate the use of dist(), hclust(), plot(), and cutree() functions to do clustering.

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

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

Now we can use hc to make a plot, it will result in a dendrogram!

```
plot(hc)
```



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

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

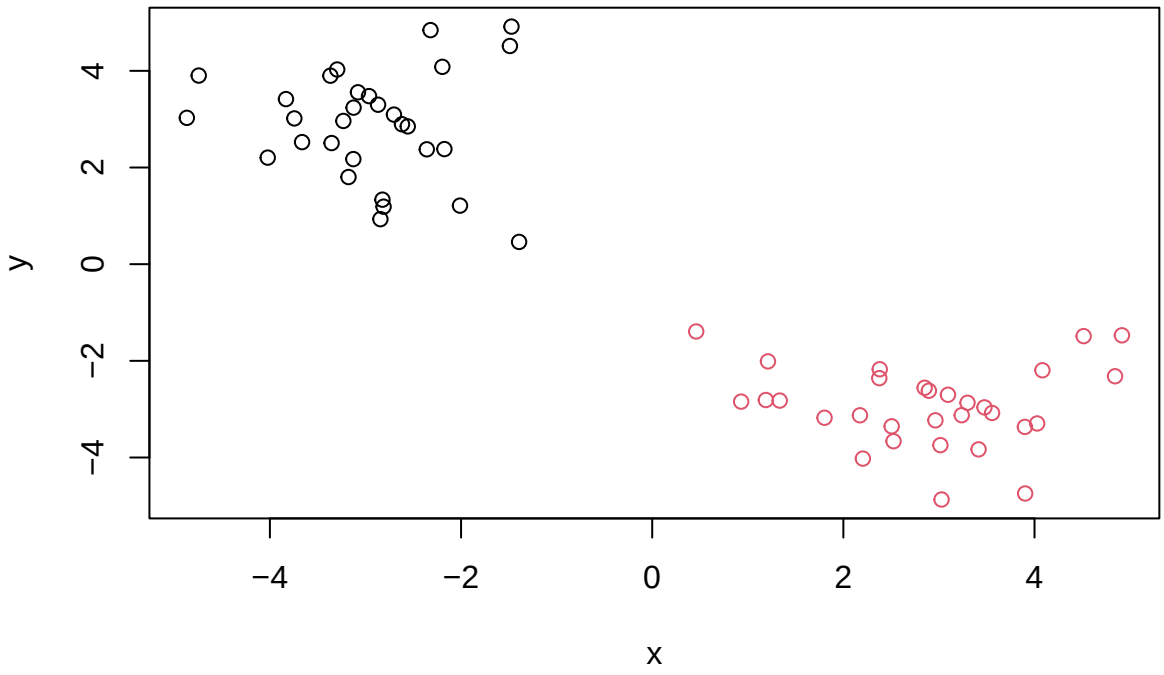
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

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

Make our resluts a plot!

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



#PCA- Principal Component Analysis

First let's import 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?

```
nrow(x)
```

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

```
ncol(x)
```

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

Next, we should check our data, using the 'View(x)'

The row names are set improperly! Lets fix this

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

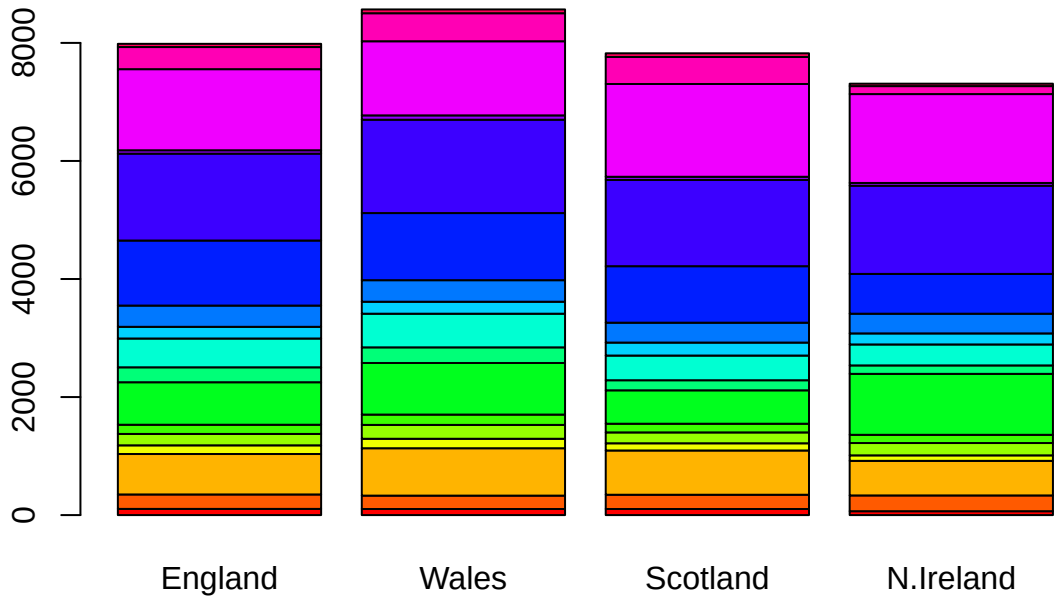
```
##                England Wales Scotland N.Ireland
## Cheese             105   103      103        66
## Carcass_meat       245   227      242       267
## Other_meat         685   803      750       586
## Fish               147   160      122        93
## Fats_and_oils      193   235      184 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

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 to fix the row-names problem using the approach in which the row names are corrected while we have Rstudio read the data file. The other method in which we could have removed the first column using x [,-1] would work the first time, however it is risky because it would delete columns every time we run it, which we do not want to occur.

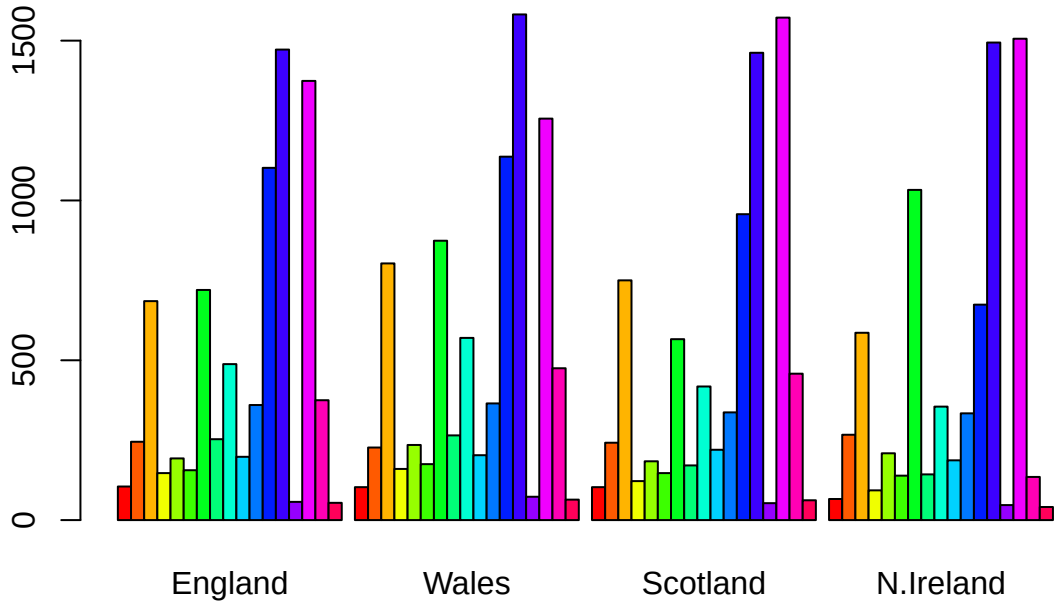
Q3: Changing what optional argument in the above barplot() function results in the following plot?

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



This is unhelpful. . . lets try again

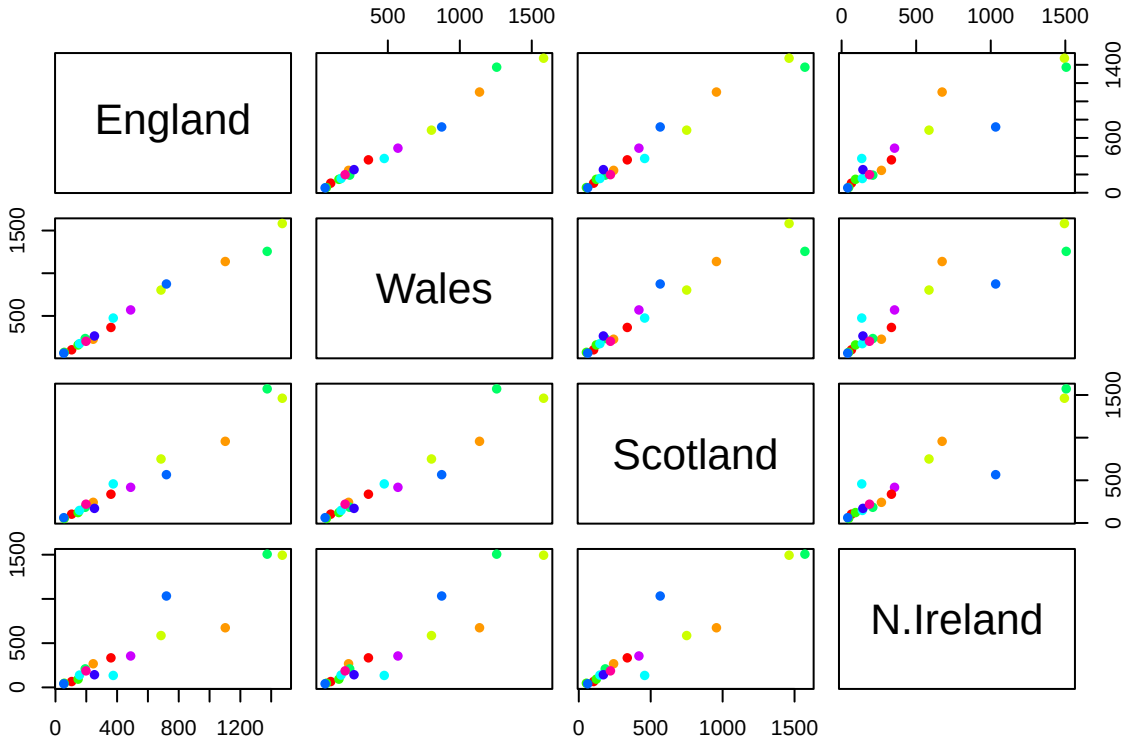
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



Setting beside=T or TRUE tells the function that we want the bars side by side.

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)
```



If a given point lies on the diagonal or a given plot, it means that the paired countries have similar values for that specific variable in the data.

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

When each of the other countries is paired with N. Ireland, the plot is not as good of a diagonal line, meaning its is less similar to the other countries. However the other countries have diagonals when paired to each other, meaning they are more similar in this data-set. This plot makes it difficult for us to discover the main difference because it does not provide us with enough details.

#PCA to the rescue

The main function in base R for PCA is *prcomp()* This wants the transpose of our data, meaning we need to flip the rows and columns.

```
t(x)
```

```
##             Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
## England        105           245         685  147            193    156
## Wales          103           227         803  160            235    175
## Scotland       103           242         750  122            184    147
## N.Ireland       66           267         586   93            209    139
##             Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
## England                720        253        488                 198
## Wales                  874        265        570                 203
```

```
## Scotland              566        171        418                220
## N.Ireland            1033        143        355                187
##           Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
## England              360         1102     1472        57         1374
## Wales                365         1137     1582        73         1256
## Scotland             337          957     1462        53         1572
## N.Ireland            334          674     1494        47         1506
##           Alcoholic_drinks  Confectionery
## England                375             54
## Wales                  475             64
## Scotland               458             62
## N.Ireland              135             41
```

Now we can use PCA!

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

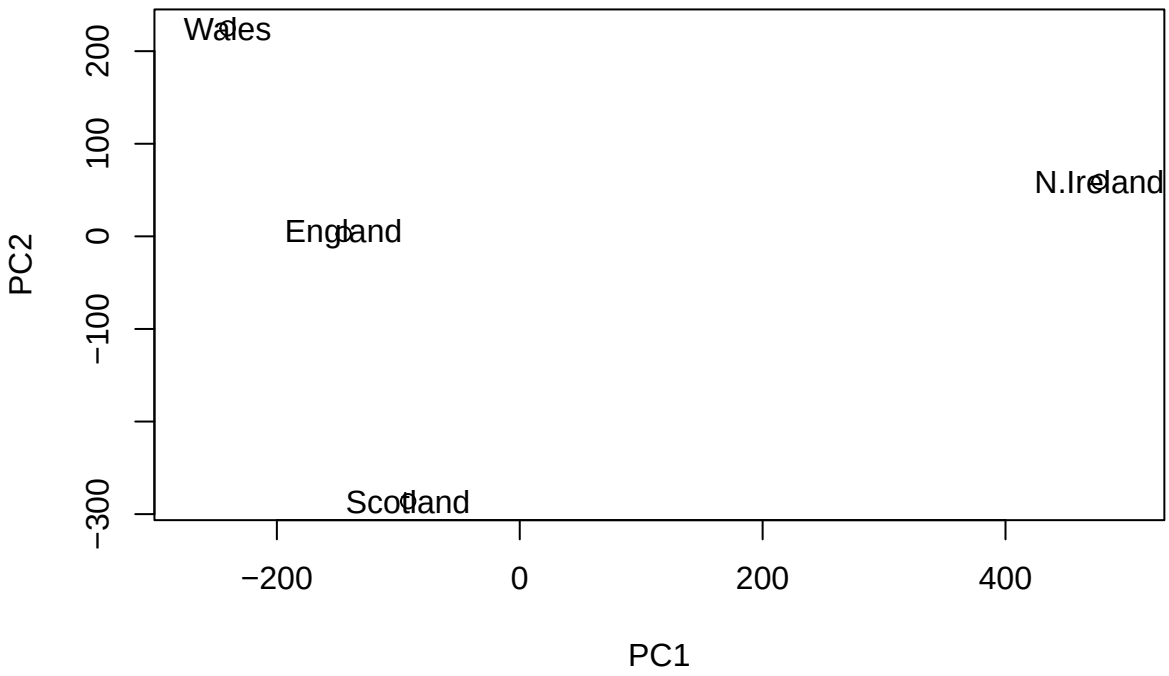
```
## Importance of components:
##                             PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
attributes(pca)
```

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

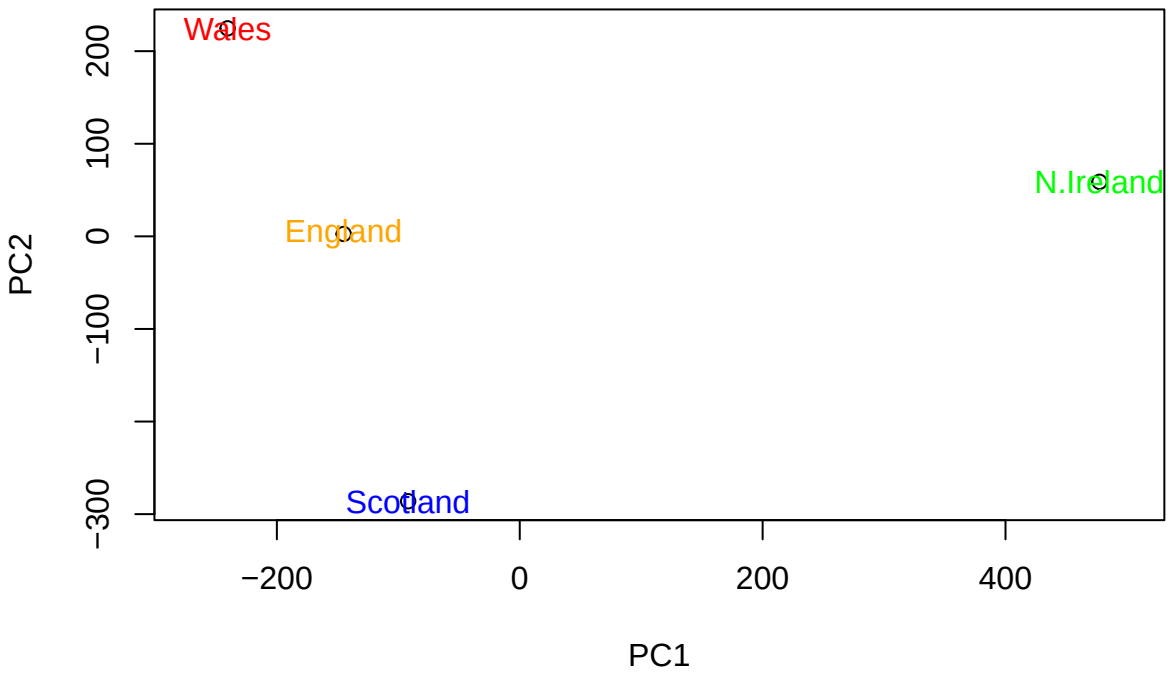
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
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"))
```



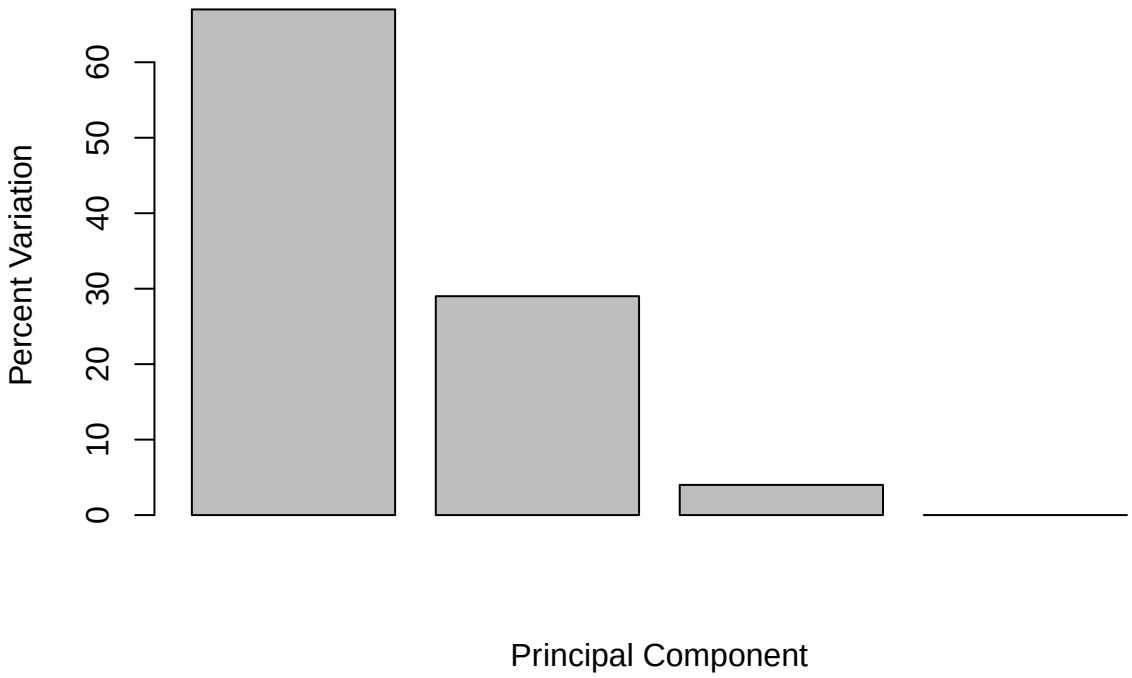
Let's dig deeper into PCA. Lets calculate how much variation each PC counts for.

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

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

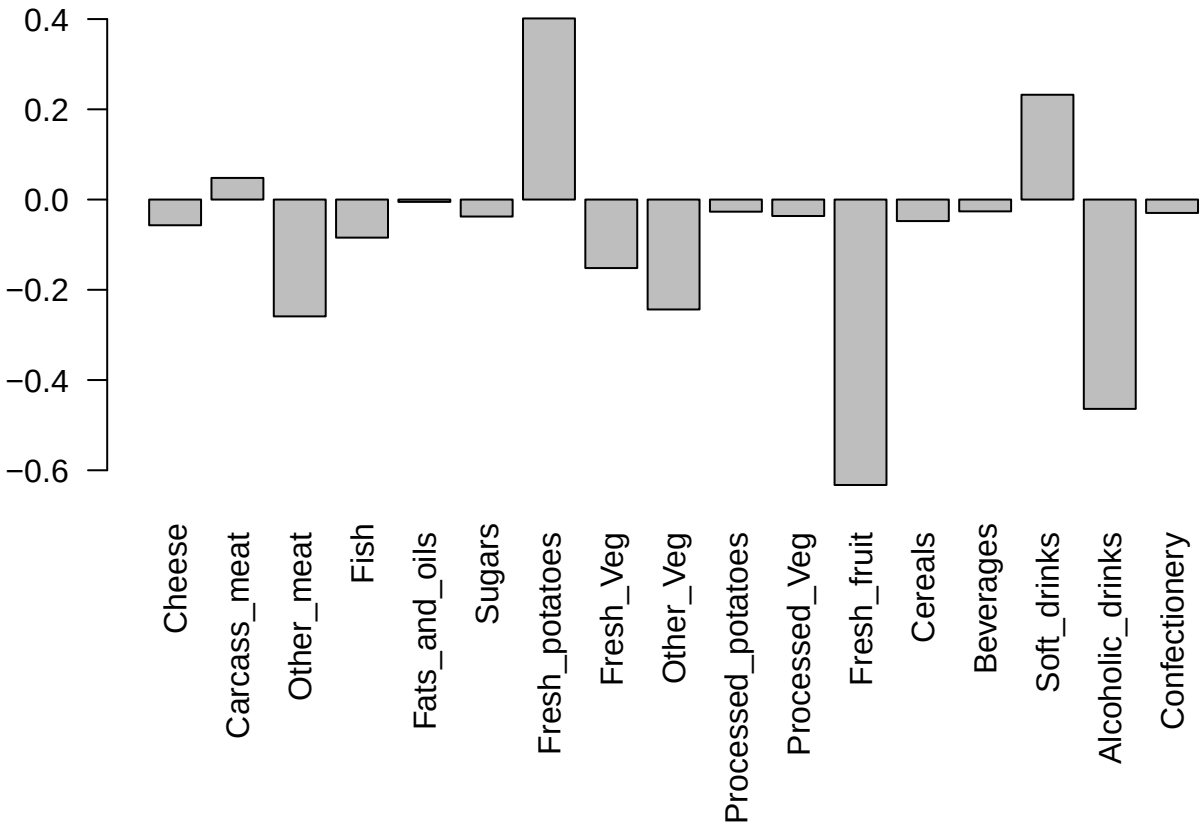
Lets look at this on a graph!

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



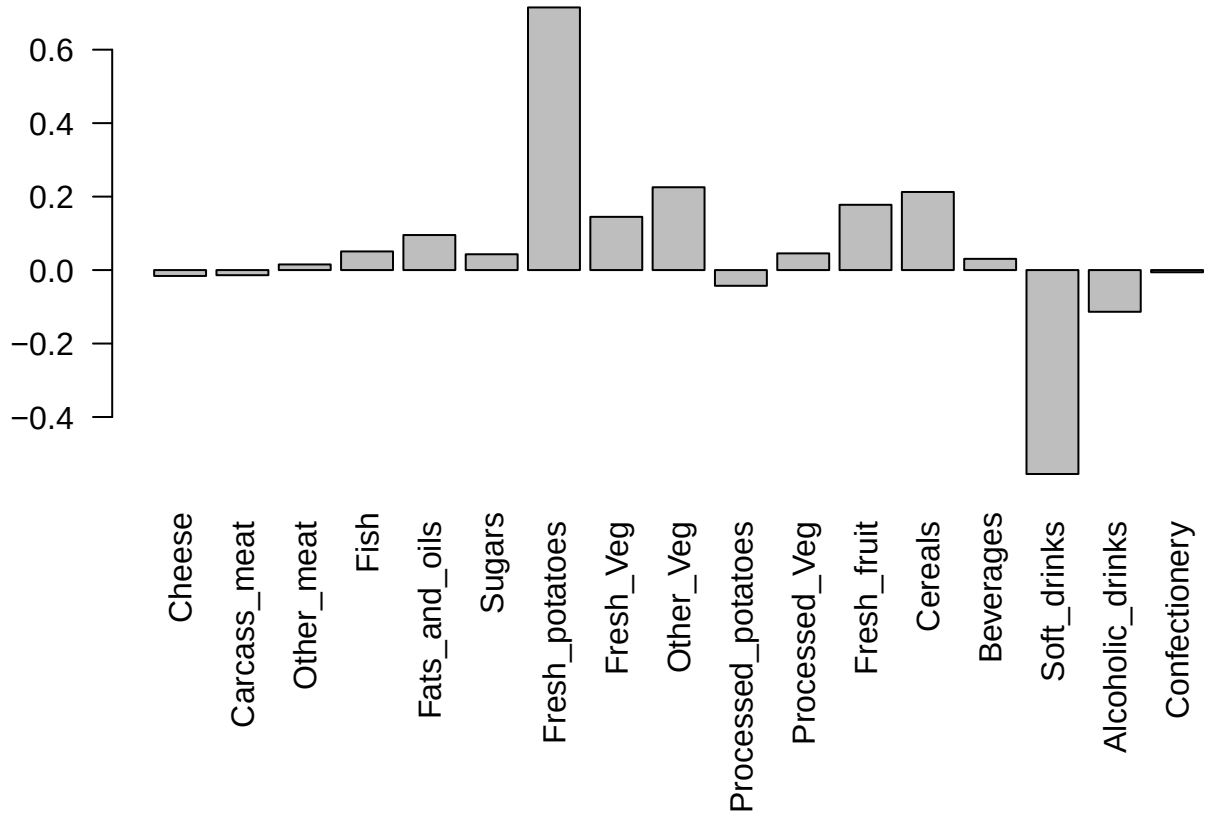
Because PC1 accounts for the most variation, we are going to focus on that.

```
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?

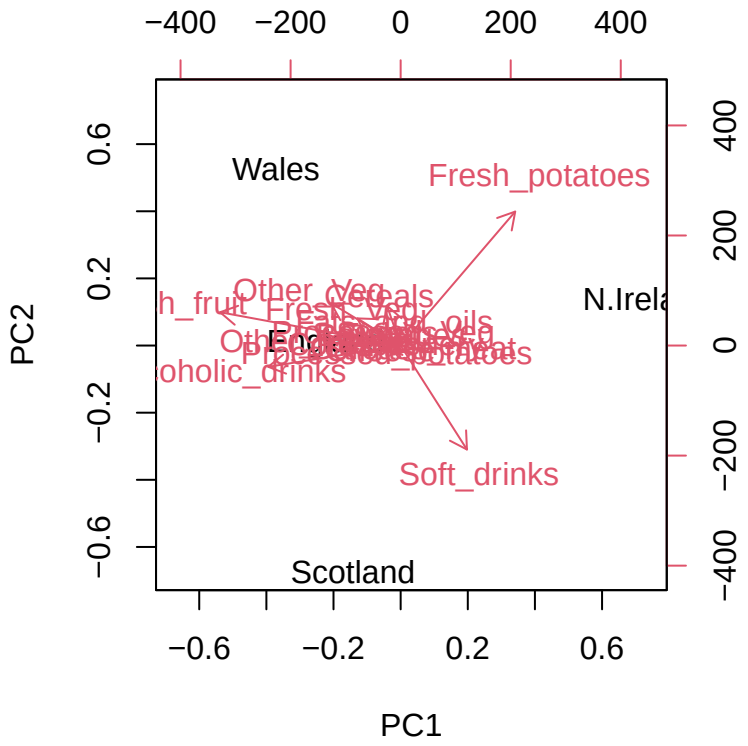
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



The two food groups it features prominently are Fresh_potatoes and Soft_drinks. PC2 mainly tells us about the second axis, explaining the second most variability of the data.

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

```
biplot(pca)
```



#Using PCA for 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? The samples are the columns and the genes are the rows.

```
ncol(rna.data)
```

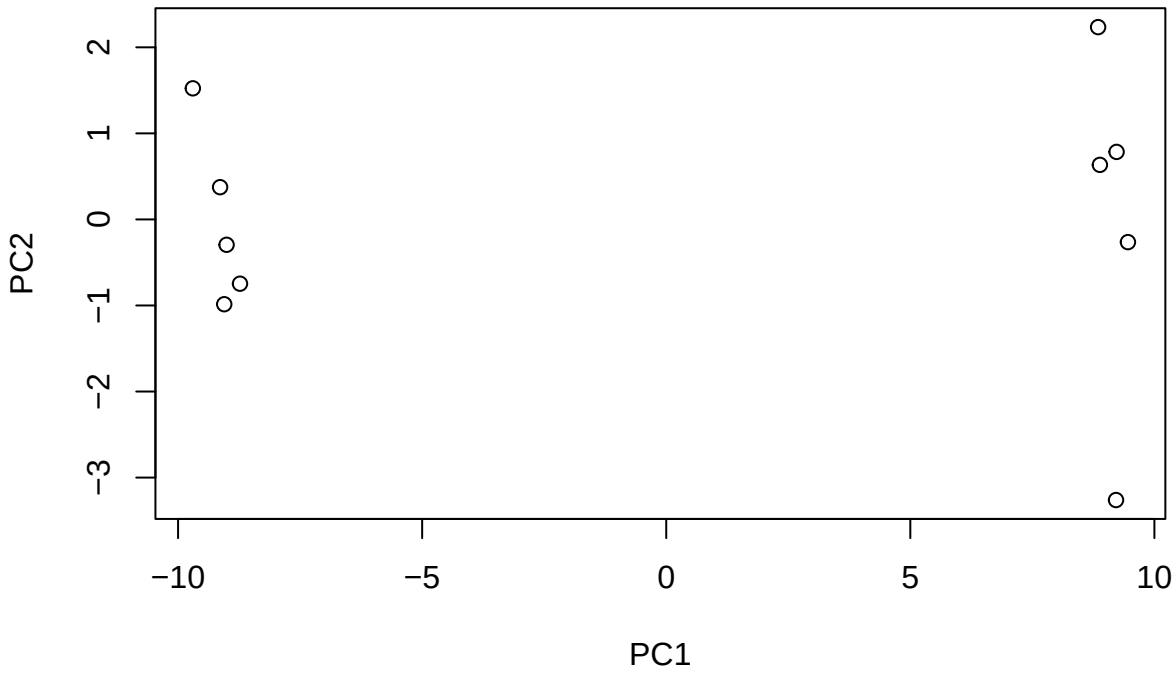
```
## [1] 10
```

```
nrow(rna.data)
```

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

Again lets transpose and plot our data

```
pca2 <- prcomp(t(rna.data), scale=TRUE)
plot(pca2$x[,1], pca2$x[,2], xlab="PC1", ylab="PC2")
```



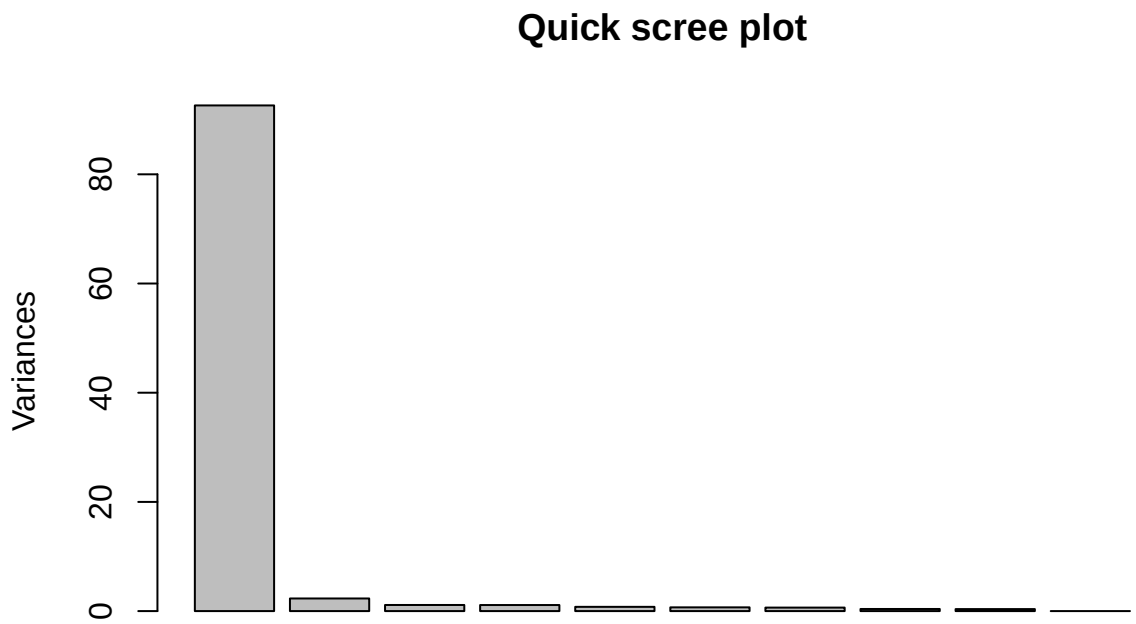
Then, we can look at a numerical summary

```
summary(pca2)
```

```
## 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
```

Similar to the first data set, PC1 is still where "all the action is". Let's look into this further!

```
plot(pca2, main="Quick scree plot")
```

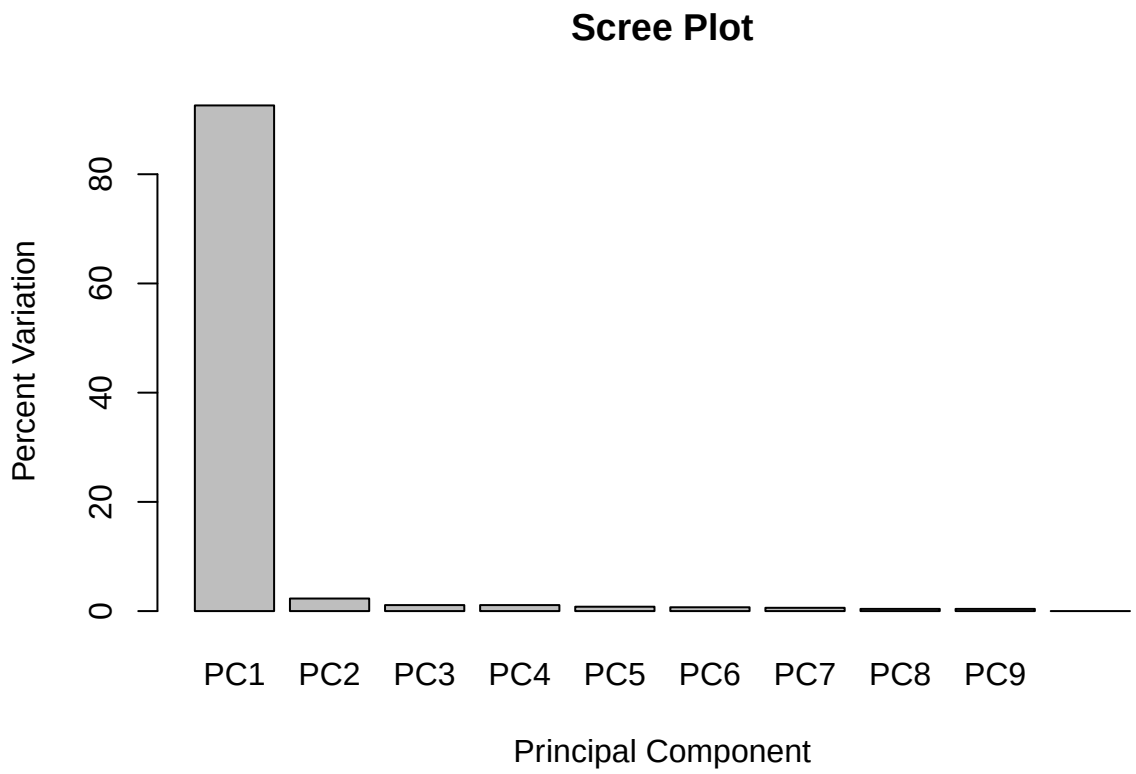



```
## Variance captured per PC
pca.var <- pca2$sdev^2

## 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
```

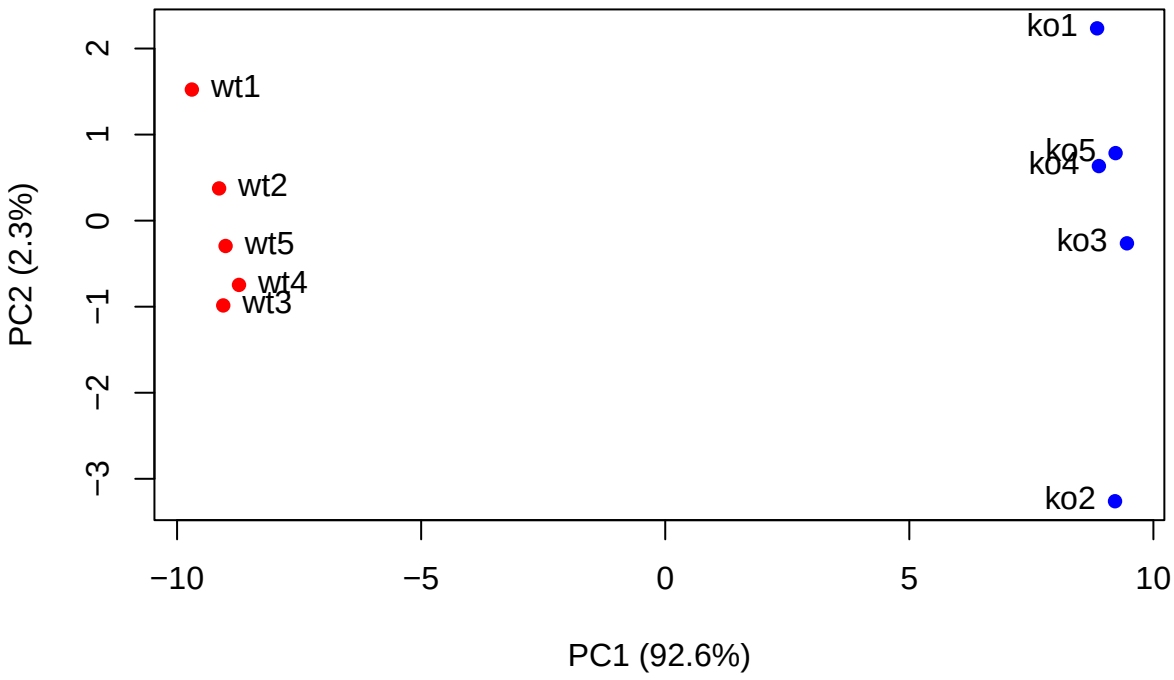
We can use this data to make our own scree plot

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



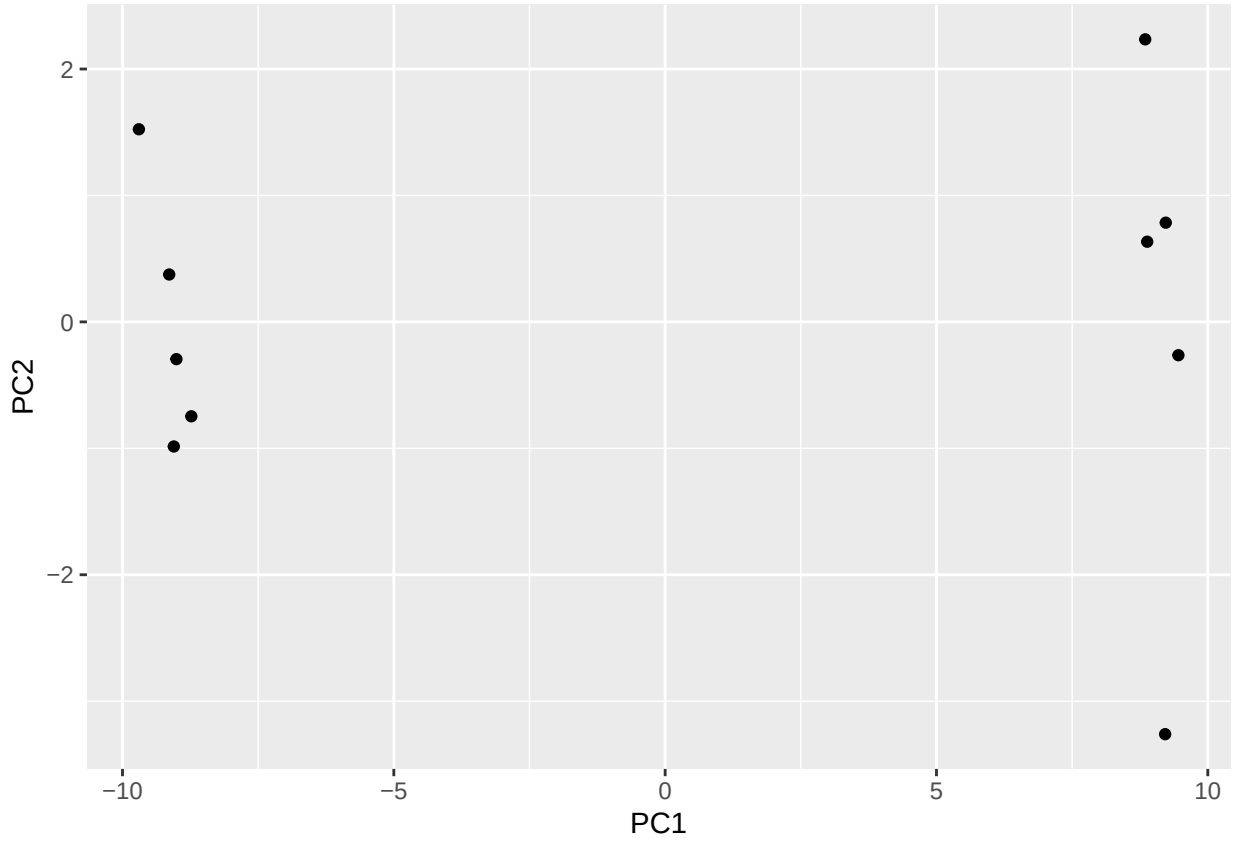
Let's add details to make it more useful

```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

plot(pca2$x[,1], pca2$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%"))

text(pca2$x[,1], pca2$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```



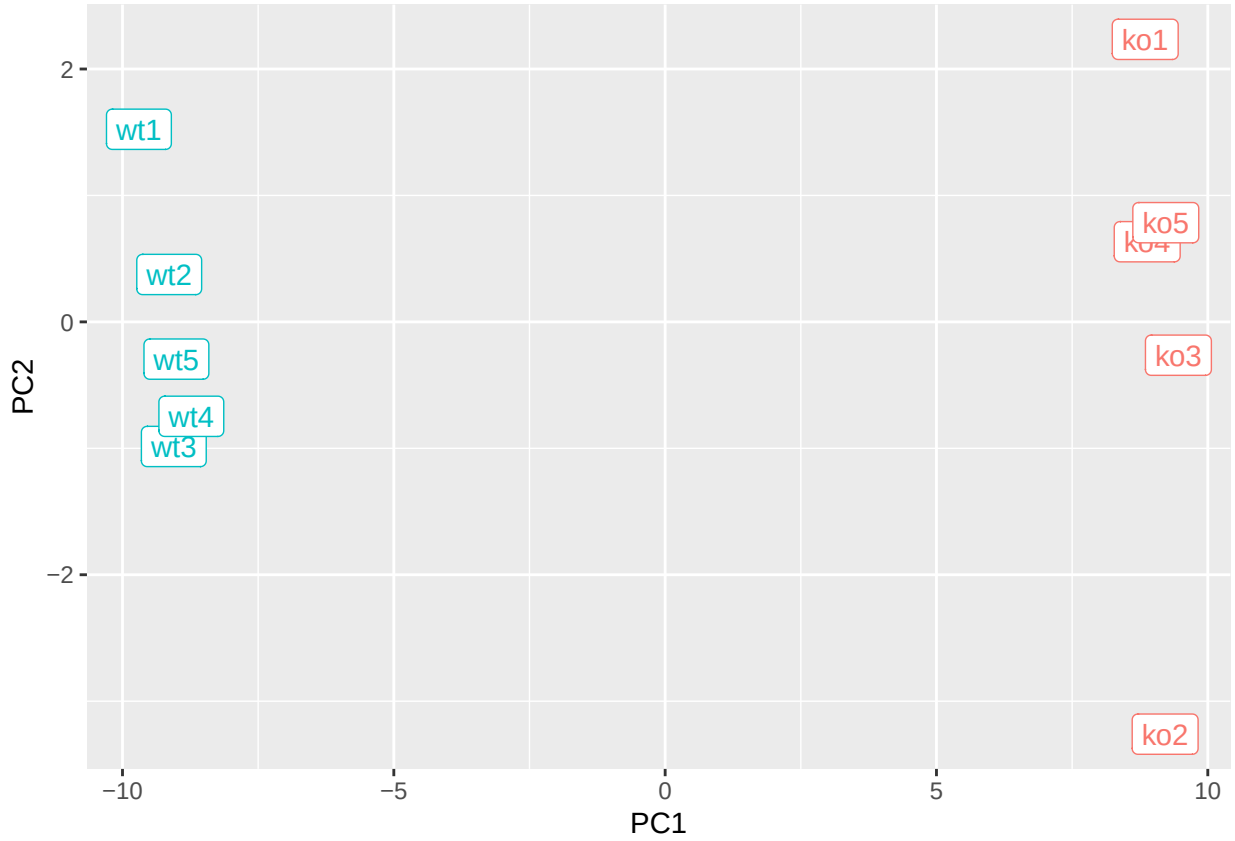
Let's also use our ggplot that we learned priorly!

```
library(ggplot2)

df <- as.data.frame(pca2$x)

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



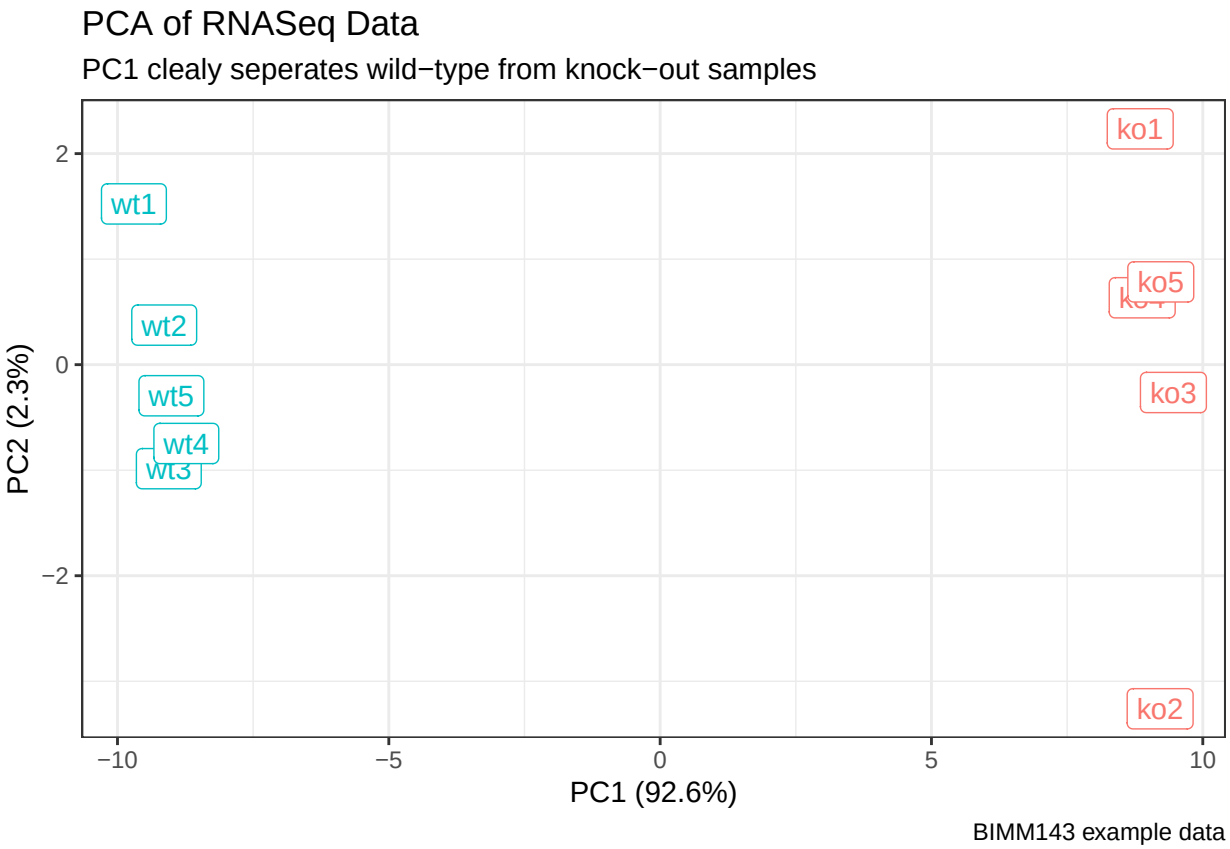
Then we can add specific labels

```
# 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
```



Finally add some polish!

```
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