Machine Learning 1

Nicole Manuguid

10/21/2021

First up is clustering methods

Kmeans clustering

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

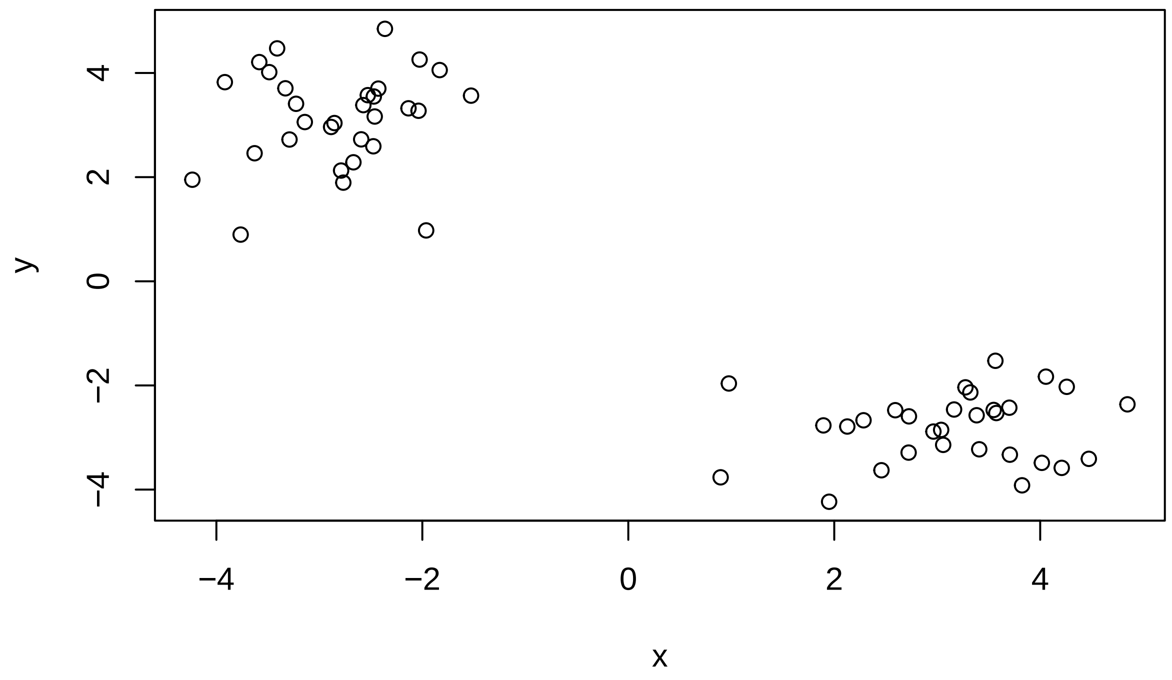
First make up some data where we know what the answer should be:

rnorm gives you 30 values around -/+3

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

```
##                x         y
##  [1,] -3.1412044  3.0578624
##  [2,] -3.4865367  4.0159913
##  [3,] -2.5297047  3.5739667
##  [4,] -3.7643103  0.8965555
##  [5,] -2.5936659  2.7252673
##  [6,] -3.2262401  3.4080059
##  [7,] -4.2333415  1.9504400
##  [8,] -1.9626672  0.9767131
##  [9,] -2.4623646  3.1637550
## [10,] -2.6691961  2.2844080
## [11,] -2.5729675  3.3828995
## [12,] -2.4710376  3.5483131
## [13,] -2.4272385  3.7001909
## [14,] -2.4757287  2.5924046
## [15,] -2.7673957  1.8946373
## [16,] -3.2898923  2.7222560
## [17,] -2.0265646  4.2577452
## [18,] -3.6287904  2.4584898
## [19,] -3.4097228  4.4723335
## [20,] -2.1343836  3.3215864
## [21,] -1.5268920  3.5648982
## [22,] -3.9178263  3.8238795
## [23,] -1.8314844  4.0556140
## [24,] -2.0368922  3.2741820
## [25,] -3.5831635  4.2090731
## [26,] -2.8534937  3.0382699
## [27,] -2.8858429  2.9630653
## [28,] -3.3296444  3.7055410
## [29,] -2.7888602  2.1265879
## [30,] -2.3630077  4.8473191
## [31,]  4.8473191 -2.3630077
## [32,]  2.1265879 -2.7888602
## [33,]  3.7055410 -3.3296444
## [34,]  2.9630653 -2.8858429
## [35,]  3.0382699 -2.8534937
## [36,]  4.2090731 -3.5831635
## [37,]  3.2741820 -2.0368922
## [38,]  4.0556140 -1.8314844
## [39,]  3.8238795 -3.9178263
## [40,]  3.5648982 -1.5268920
## [41,]  3.3215864 -2.1343836
## [42,]  4.4723335 -3.4097228
## [43,]  2.4584898 -3.6287904
## [44,]  4.2577452 -2.0265646
## [45,]  2.7222560 -3.2898923
## [46,]  1.8946373 -2.7673957
## [47,]  2.5924046 -2.4757287
## [48,]  3.7001909 -2.4272385
## [49,]  3.5483131 -2.4710376
## [50,]  3.3828995 -2.5729675
## [51,]  2.2844080 -2.6691961
## [52,]  3.1637550 -2.4623646
## [53,]  0.9767131 -1.9626672
## [54,]  1.9504400 -4.2333415
## [55,]  3.4080059 -3.2262401
## [56,]  2.7252673 -2.5936659
## [57,]  0.8965555 -3.7643103
## [58,]  3.5739667 -2.5297047
## [59,]  4.0159913 -3.4865367
## [60,]  3.0578624 -3.1412044
```

```
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 -2.813002  3.133742
## 2  3.133742 -2.813002
##
## 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] 39.03056 39.03056
##  (between_SS / total_SS =  93.1 %)
##
## 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 assignmnet/membership?

```
km$clusters
```

```
## NULL
```

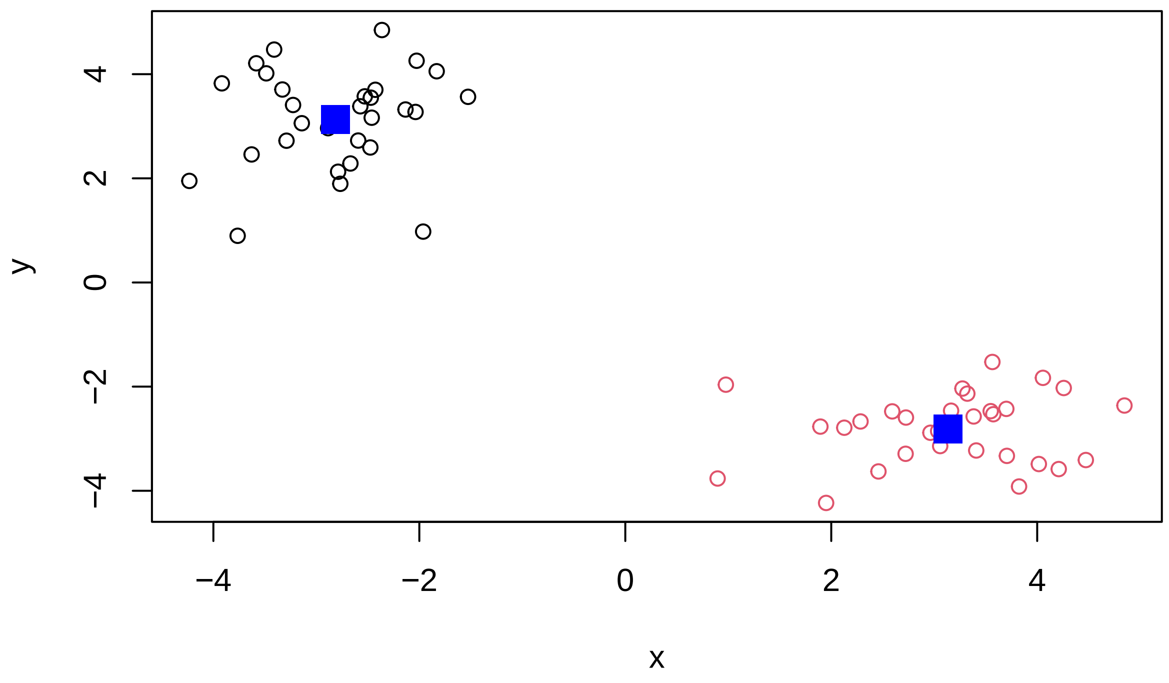
Q. What ‘component’ of your result object details clsuter center?

```
km$centers
```

```
##           x         y
## 1 -2.813002  3.133742
## 2  3.133742 -2.813002
```

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

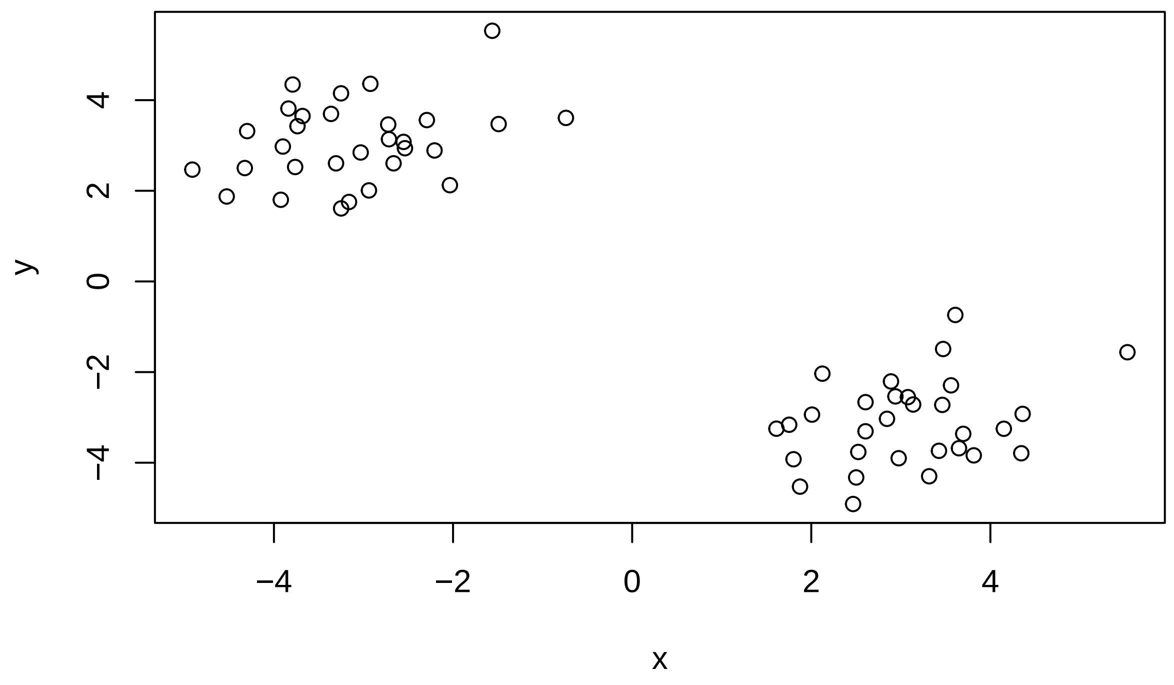
```
plot(x, col=km$cluster)
points(km$centers, col = "blue", pch = 15, cex = 2)
```



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

```
##                x          y
##  [1,] -2.6640839  2.6059780
##  [2,] -2.9390800  2.0082212
##  [3,] -1.5623976  5.5306240
##  [4,] -3.7372025  3.4258740
##  [5,] -3.3626717  3.6969259
##  [6,] -2.7239710  3.4631122
##  [7,] -3.7630841  2.5250689
##  [8,] -3.6806079  3.6495002
##  [9,] -3.0315489  2.8454496
## [10,] -0.7395924  3.6083428
## [11,] -3.2493730  1.6109370
## [12,] -4.9111490  2.4671732
## [13,] -3.9006485  2.9763371
## [14,] -2.2065966  2.8901086
## [15,] -2.9231745  4.3602302
## [16,] -4.3253700  2.5016956
## [17,] -4.5275763  1.8744552
## [18,] -2.7155145  3.1375565
## [19,] -3.8382030  3.8144844
## [20,] -4.2992578  3.3167316
## [21,] -2.0352245  2.1241205
## [22,] -2.5528483  3.0789149
## [23,] -1.4907307  3.4723631
## [24,] -3.7912991  4.3452170
## [25,] -3.2508148  4.1502938
## [26,] -2.2925818  3.5603762
## [27,] -3.3067187  2.6057904
## [28,] -3.9234862  1.8019638
## [29,] -3.1616756  1.7532017
## [30,] -2.5367984  2.9396486
## [31,]  2.9396486 -2.5367984
## [32,]  1.7532017 -3.1616756
## [33,]  1.8019638 -3.9234862
## [34,]  2.6057904 -3.3067187
## [35,]  3.5603762 -2.2925818
## [36,]  4.1502938 -3.2508148
## [37,]  4.3452170 -3.7912991
## [38,]  3.4723631 -1.4907307
## [39,]  3.0789149 -2.5528483
## [40,]  2.1241205 -2.0352245
## [41,]  3.3167316 -4.2992578
## [42,]  3.8144844 -3.8382030
## [43,]  3.1375565 -2.7155145
## [44,]  1.8744552 -4.5275763
## [45,]  2.5016956 -4.3253700
## [46,]  4.3602302 -2.9231745
## [47,]  2.8901086 -2.2065966
## [48,]  2.9763371 -3.9006485
## [49,]  2.4671732 -4.9111490
## [50,]  1.6109370 -3.2493730
## [51,]  3.6083428 -0.7395924
## [52,]  2.8454496 -3.0315489
## [53,]  3.6495002 -3.6806079
## [54,]  2.5250689 -3.7630841
## [55,]  3.4631122 -2.7239710
## [56,]  3.6969259 -3.3626717
## [57,]  3.4258740 -3.7372025
## [58,]  5.5306240 -1.5623976
## [59,]  2.0082212 -2.9390800
## [60,]  2.6059780 -2.6640839
```

```
plot(x)
```



Hierarachical Clustering

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()

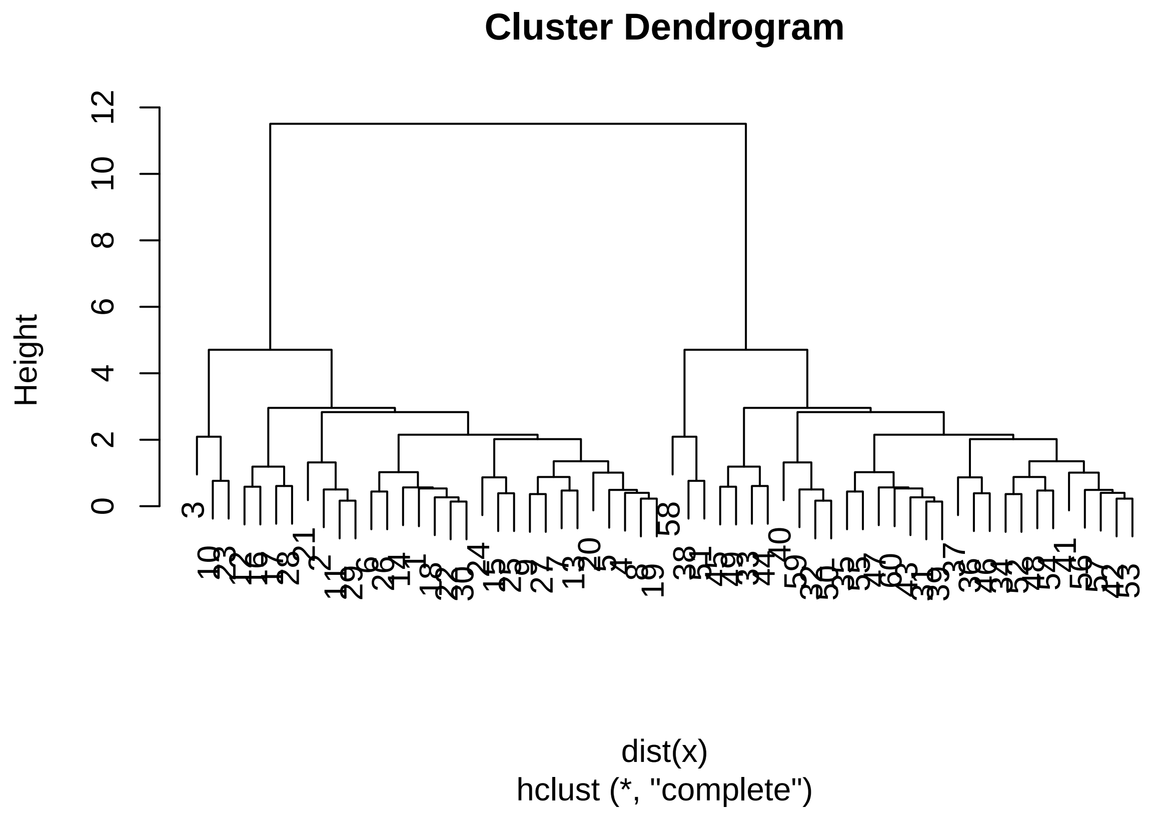
Demonstrate the use of dist(), hclust(), plot (), and cutree() functions to do clustering, Generate dendrograms and return cluster assingment membership vector…

```
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)
```

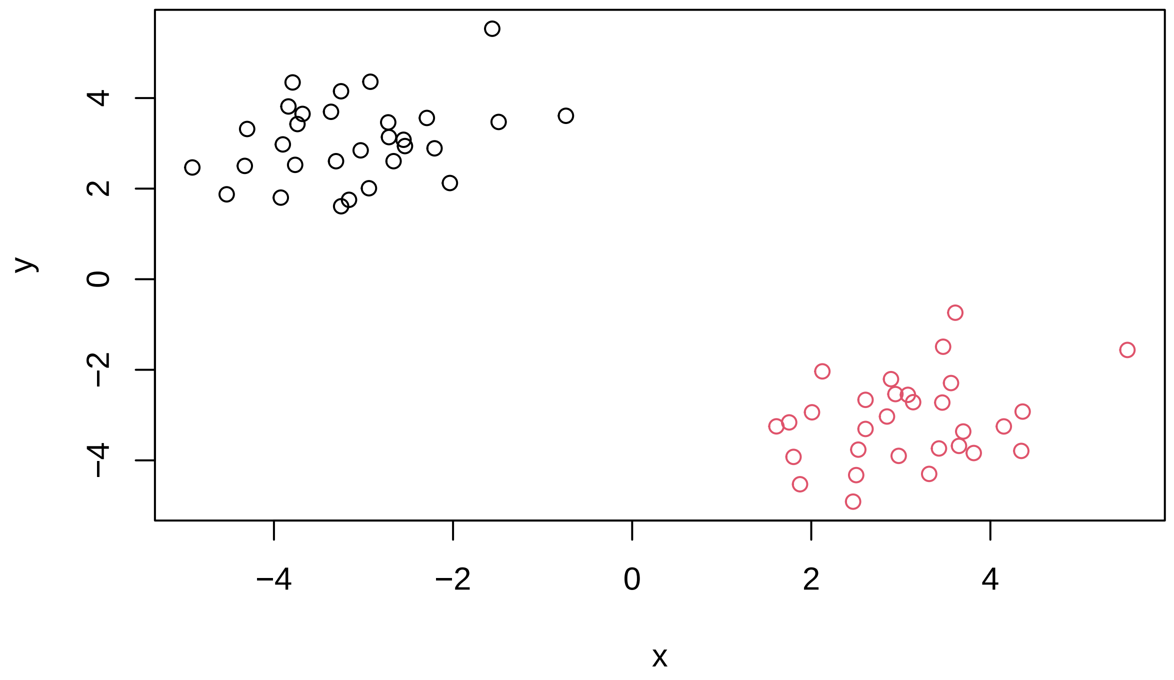
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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)
```



#Principal Component Analysis(PCA) of UK Food 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
```

Checking our data

```
View(x)
```

fix row names; this should be 17 x 4 but we have 17 x5 dimensions

```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
```

recheck dimensions

```r
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?

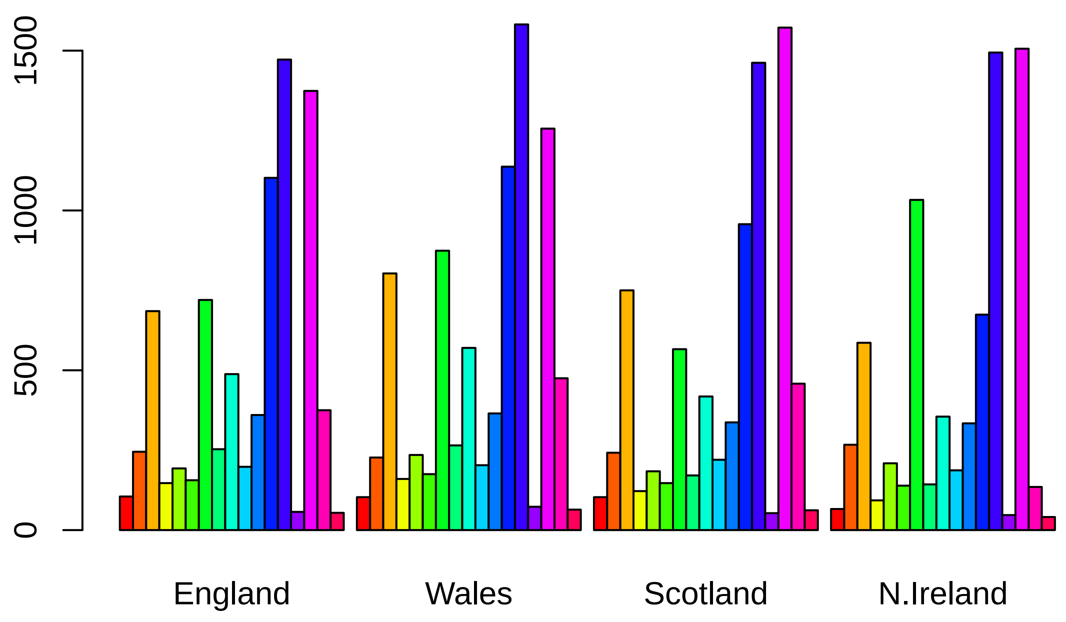
be careful with previous function. can use this function below to get the same results

```r
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
```

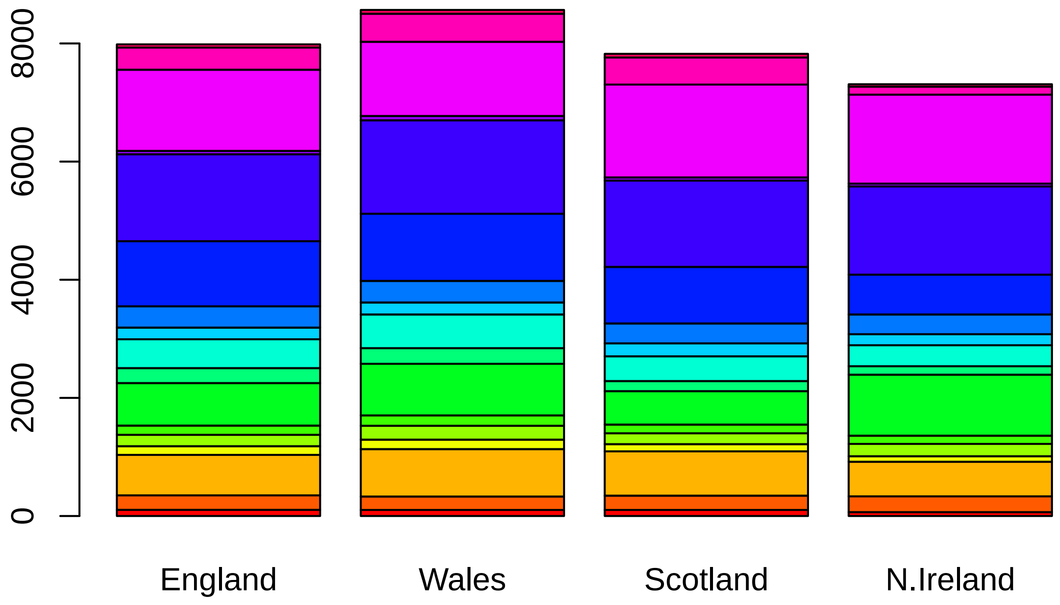
Spotting major differences and trends

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



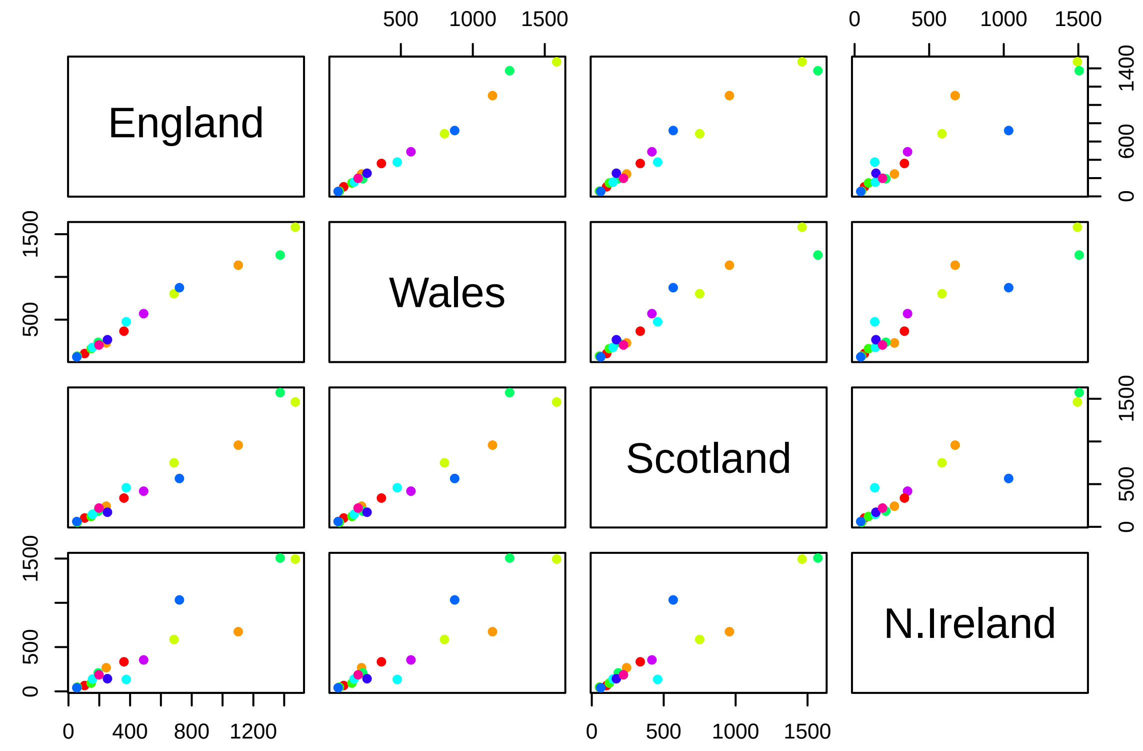
Q3: Changing what optional argument in the above barplot() function results in the following plot?

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



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 point lies in the diagonal between two countries it means they share similarities within the data.

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

For N.Ireland, the blue point is not following the general shape of the diagonal and is most different compared to other coutnries in the UK.

#PCA to the rescue

the main function is in base R

the main function in base R for PCA is 'prcomp()' this wants the transpose of our data

```
t(x)
```

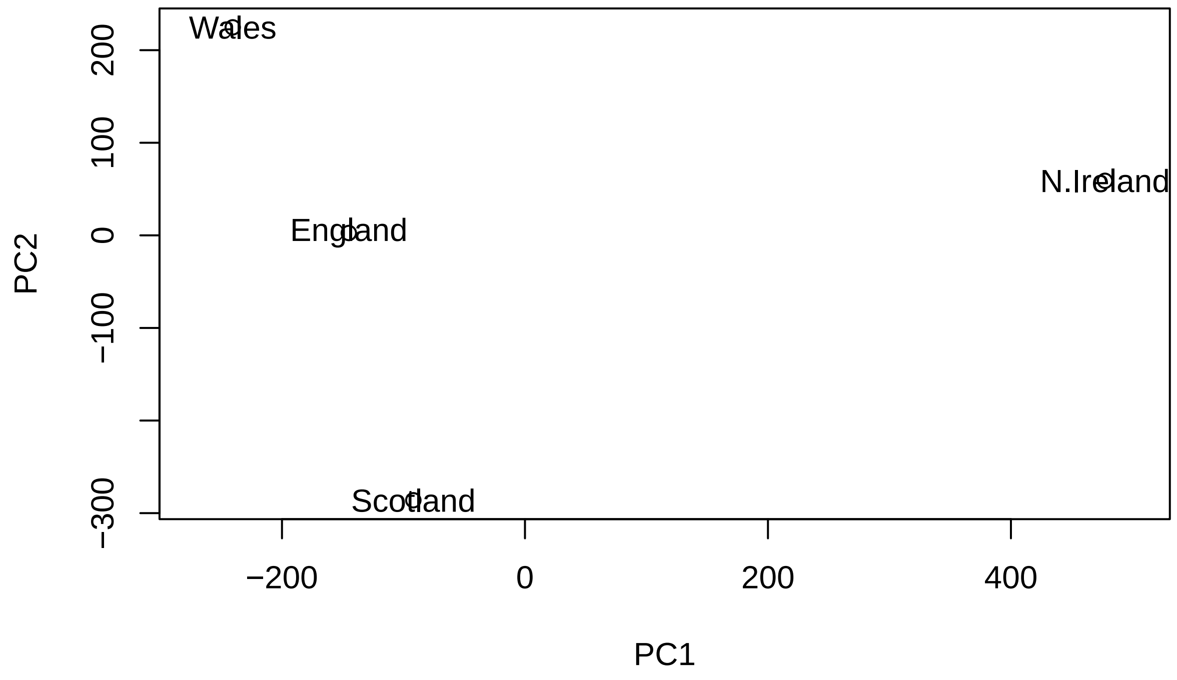
```
##             Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
## England        105           245         685   147           193     156
## Wales          103           227         803   160           235     175
## Scotland       103           242         750   122           184     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
```

```
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
```

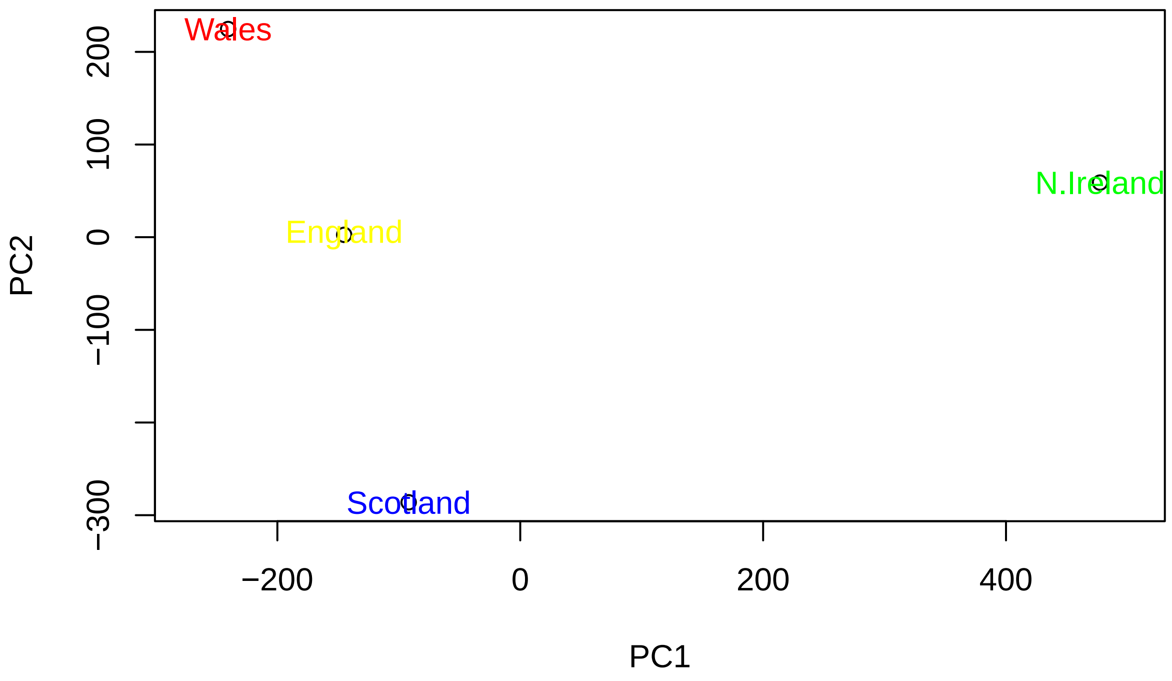
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
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("yellow", "red", "blue", "green"))
```



calculate how much variation in the original data each PC accounts for

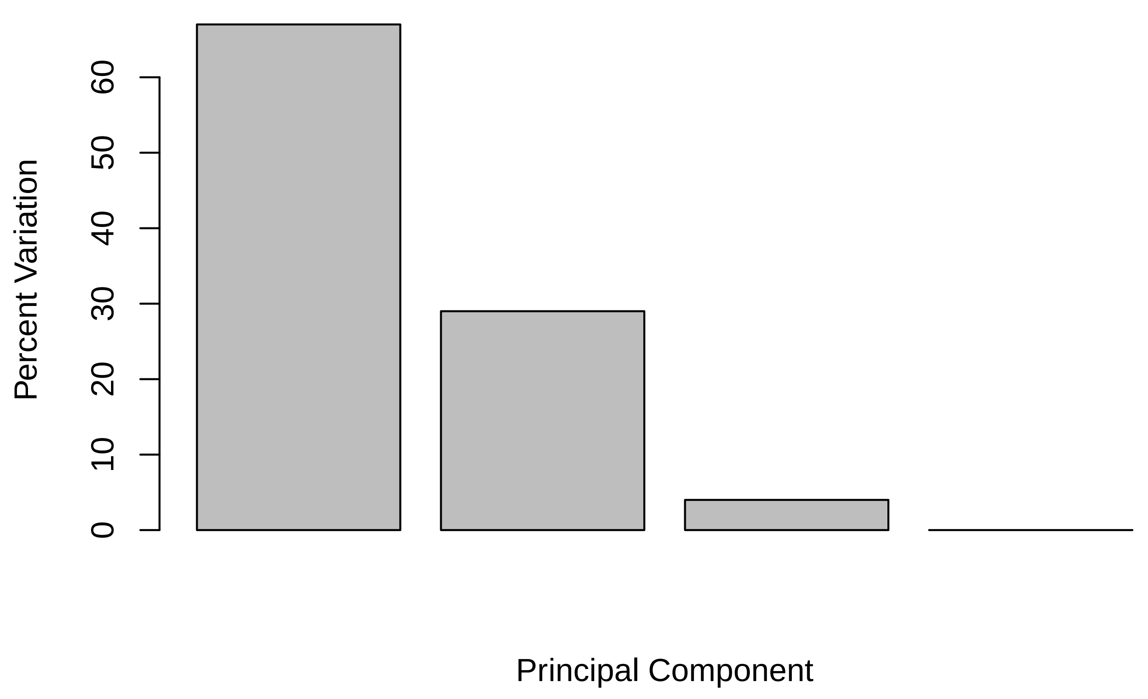
```
v <- round(pca$sdev^2/sum(pca$sdev^2)*100)
v
```

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

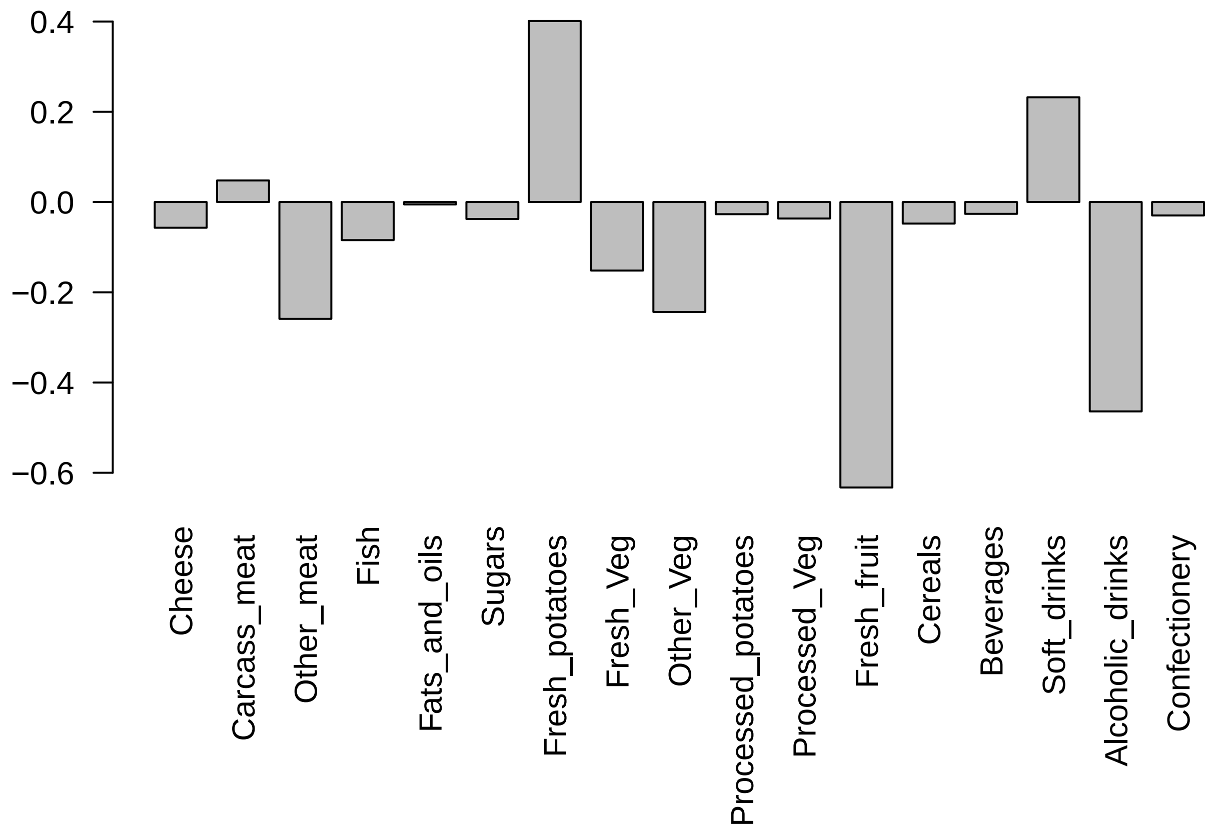
```
z <- summary(pca)
z$importance
```

```
##                              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
```

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

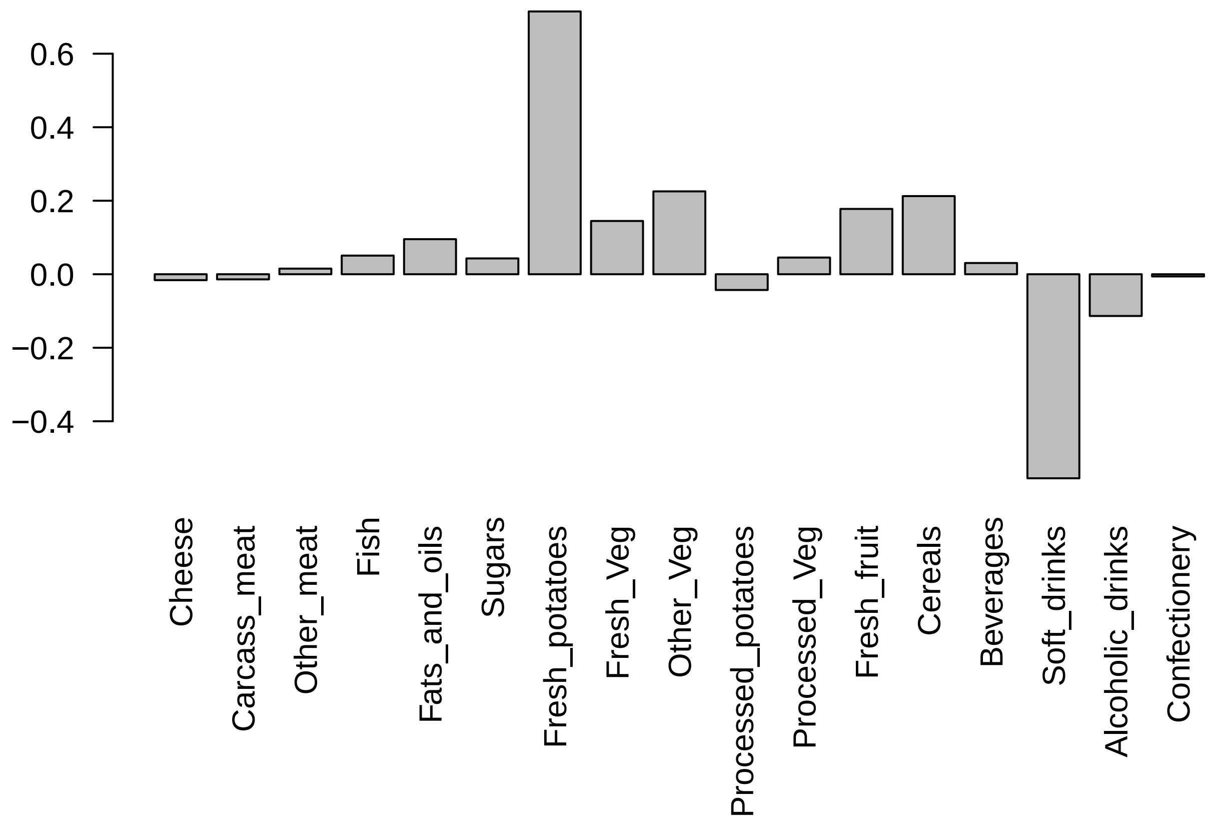


```
par(mar=c(10, 3,0.35, 0))
barplot (pca$rotation[,1], las=2)
```



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

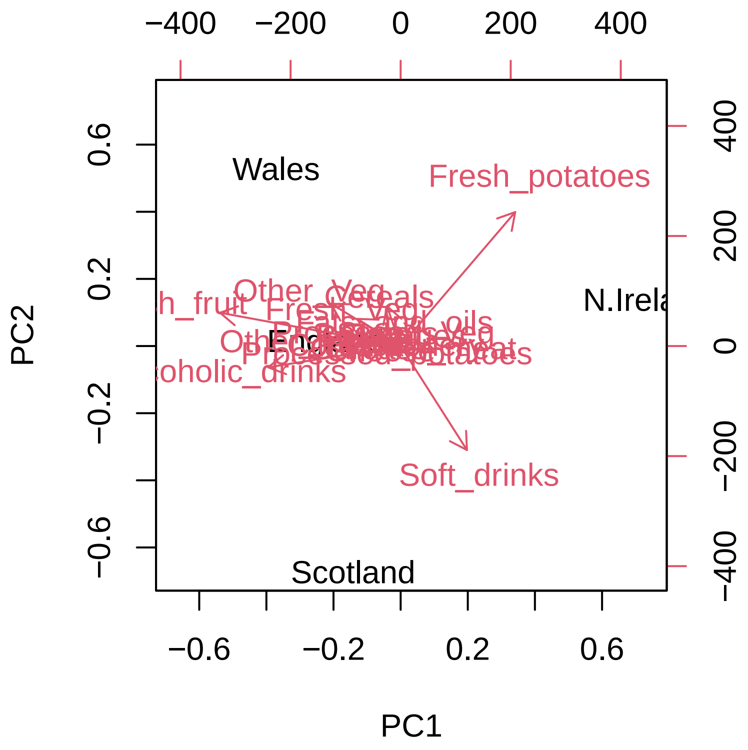
```
par(mar=c(10, 3,0.35, 0))
barplot (pca$rotation[,2], las=2)
```



The two main food groups are fresh potatoes and soft drinks. PC2 shows us how much variation there is in food in N.Ireland compared to the rest of the countries in the UK.

#biplots

```
biplot(pca)
```



#pca of r-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
```

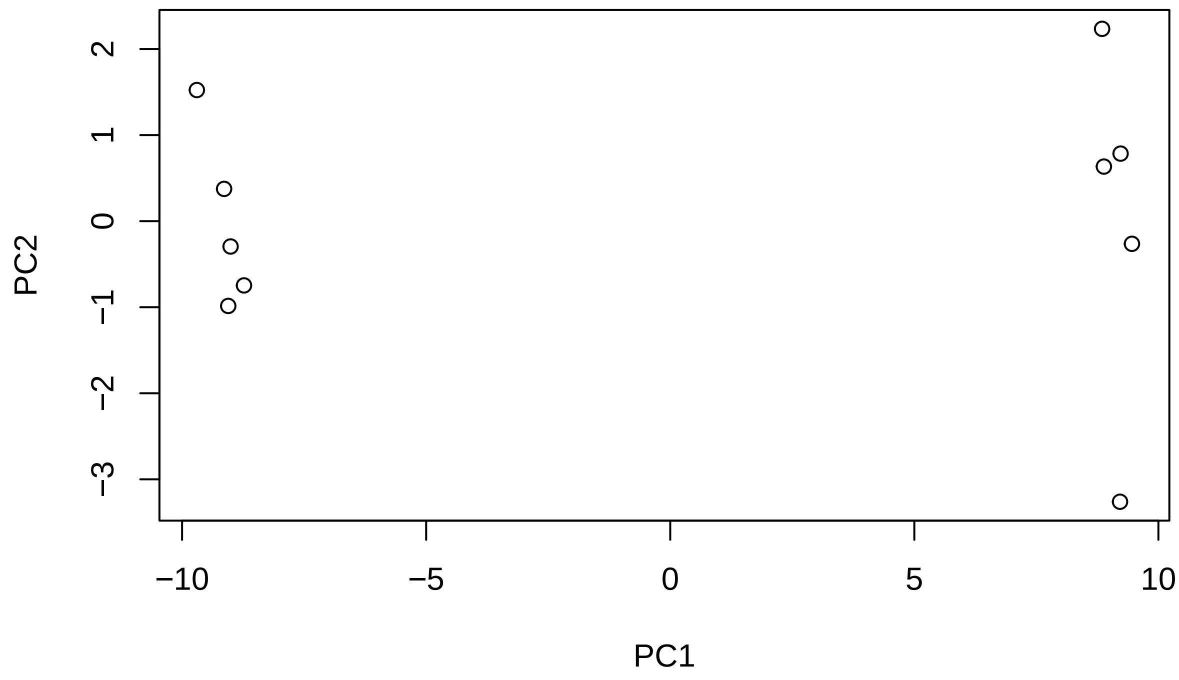
Q10: How many genes and samples are in this data set?

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

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

There are 100 genes and 10 samples

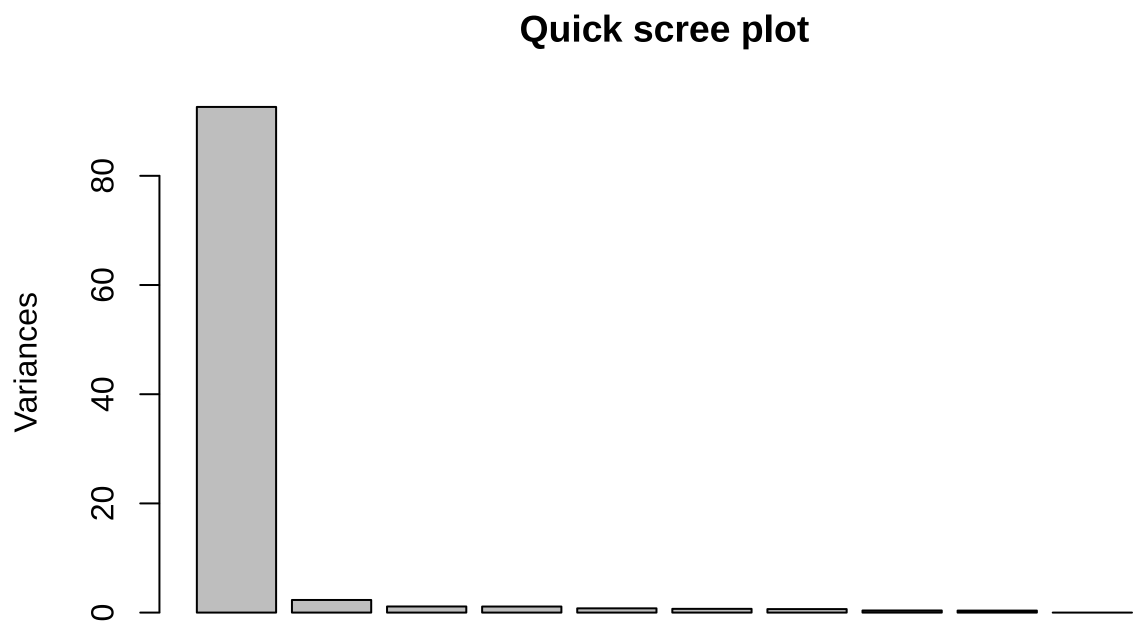
```
pca <- prcomp(t(rna.data), scale = TRUE)
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2")
```



```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")
```



Variance captured per PC

```
pca.var <- pca$sdev^2
```

Percent variance is more informative to look at

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

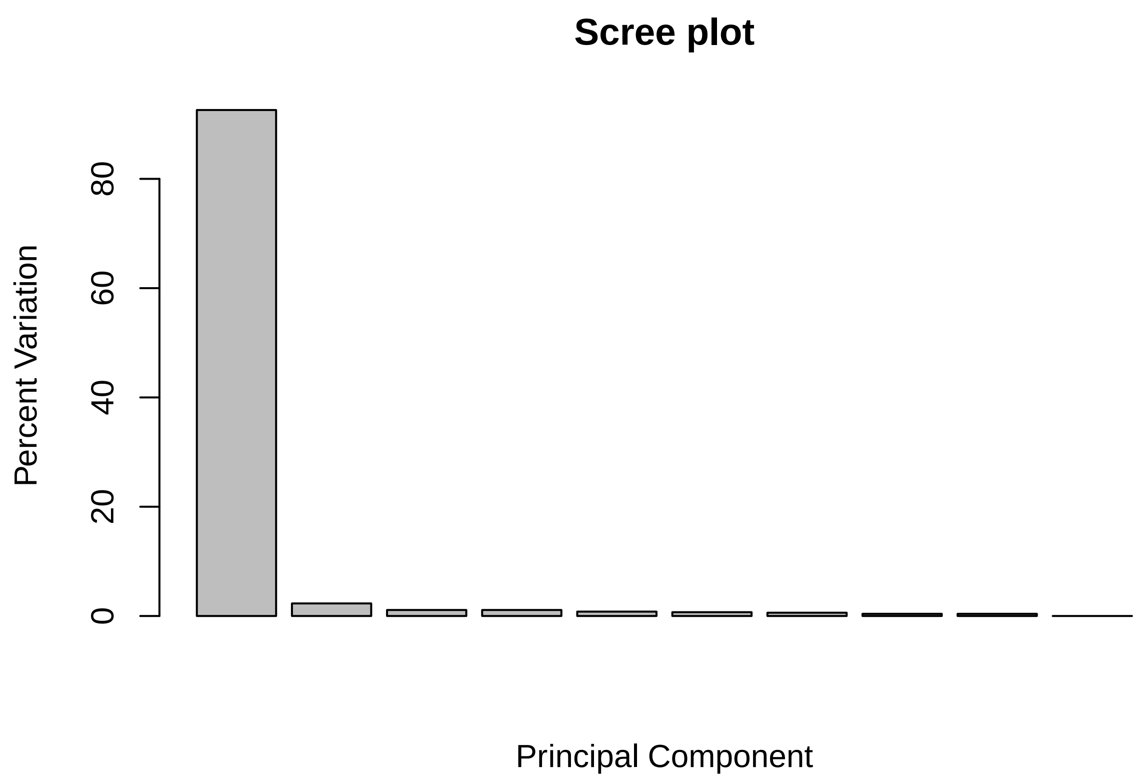
```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

generate scree-plot

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

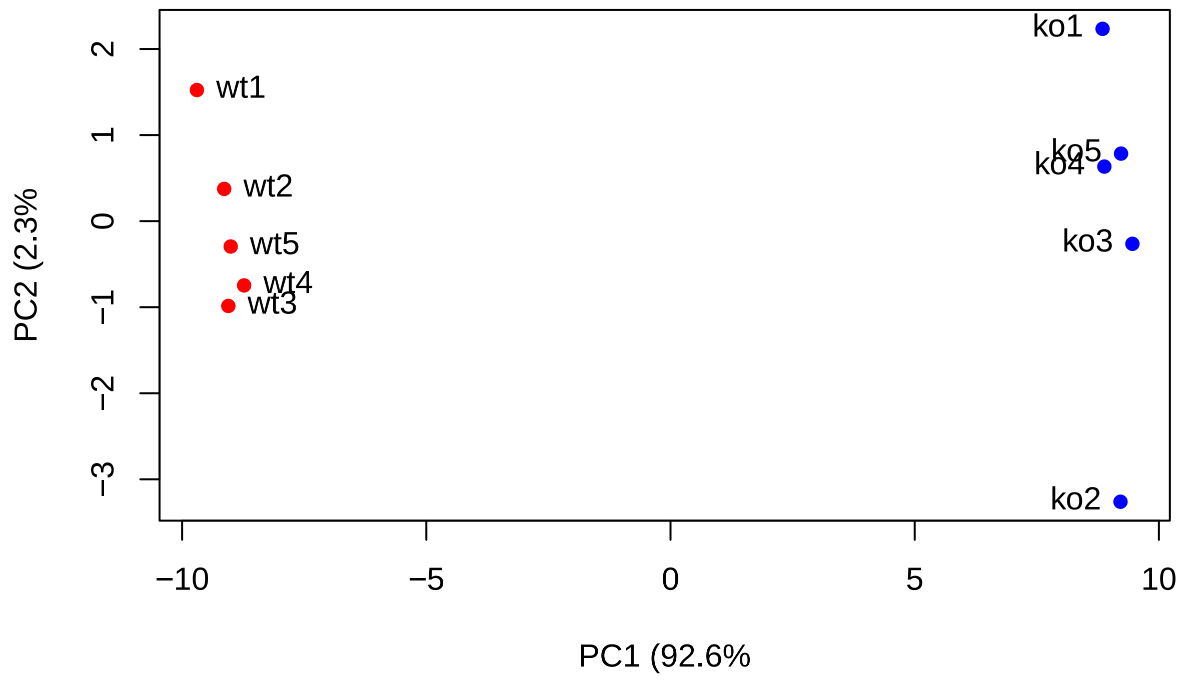
```
## Warning in plot.window(xlim, ylim, log = log, ...): "name.arg" is not a
## graphical parameter
```

```
## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...):
## "name.arg" is not a graphical parameter
## Warning in axis(if (horiz) 1 else 2, cex.axis = cex.axis, ...): "name.arg" is
## not a graphical parameter
```



A vector of colors for wt and ko samples

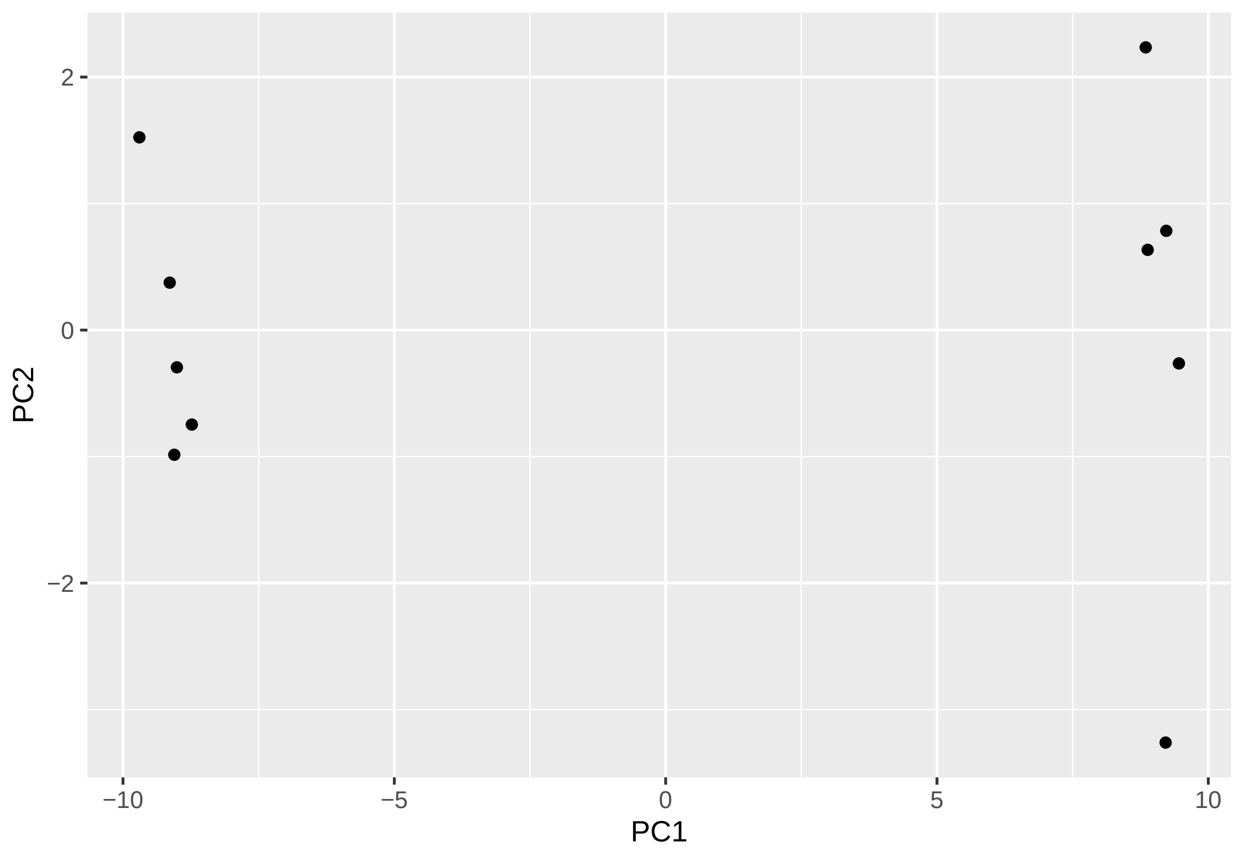
```
colvec <- colnames(rna.data)
colvec[grep ("wt", colvec)] <- "red"
colvec[grep ("ko", colvec)] <- "blue"

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

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



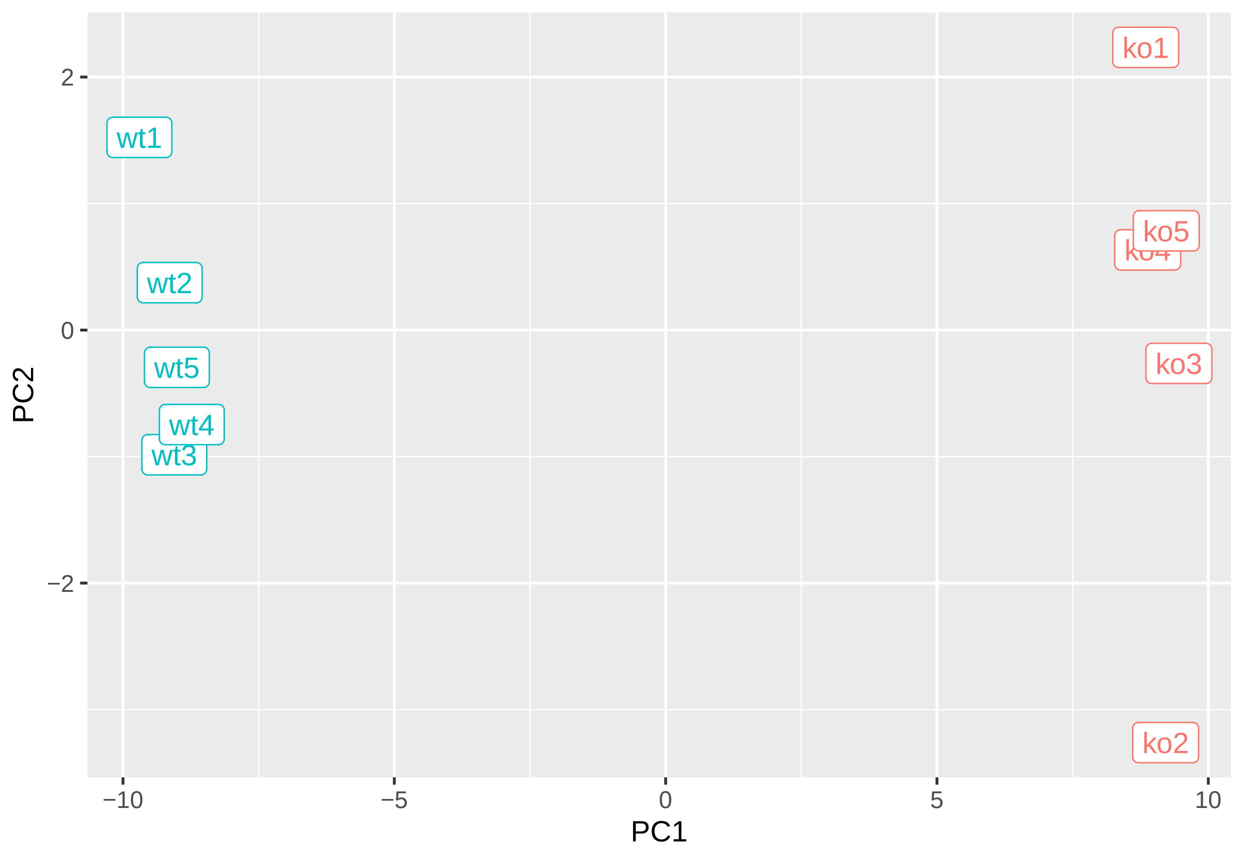
```
text
```

```
## function (x, ...)
## UseMethod("text")
## <bytecode: 0x000000000f9785a8>
## <environment: namespace:graphics>
```

```
library(ggplot2)
df <- as.data.frame(pca$x)

#make first plot

ggplot(df) + aes(PC1, PC2) + geom_point()
```

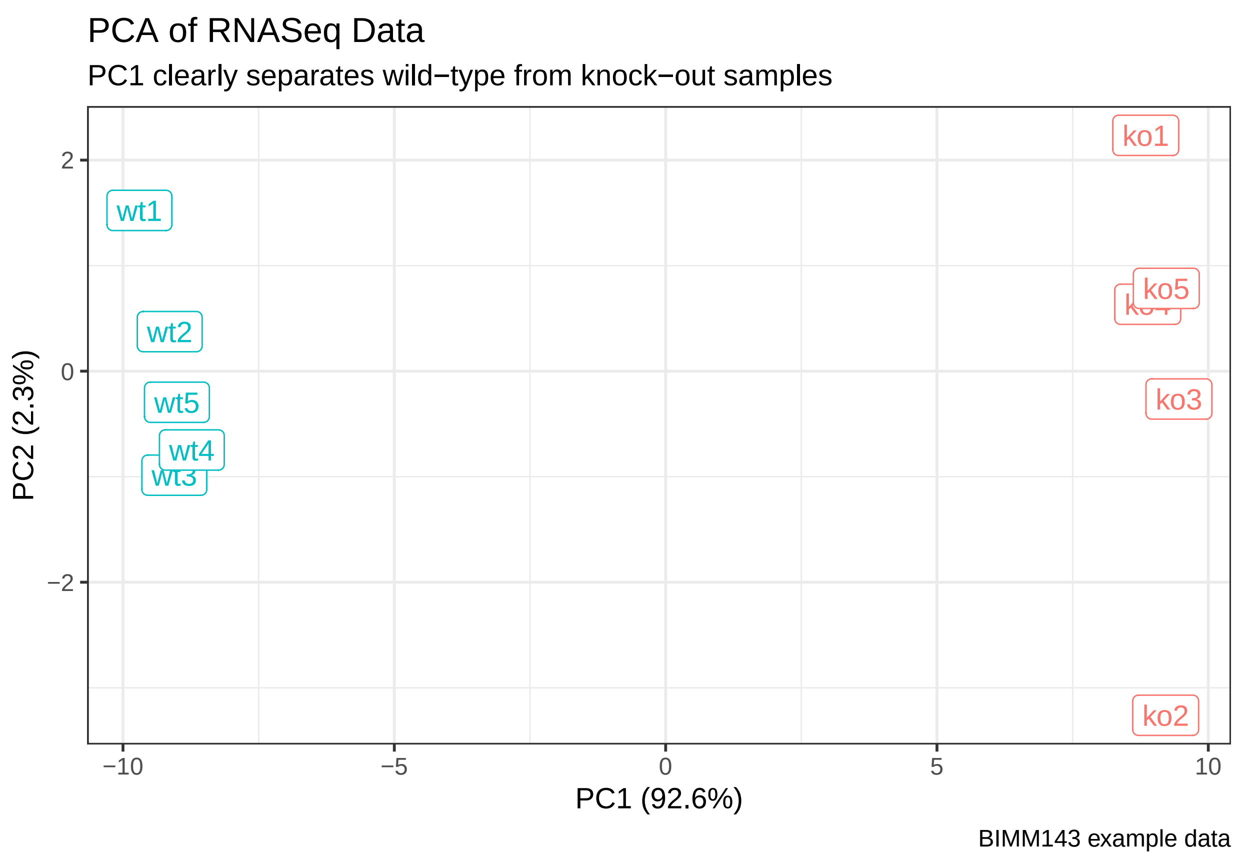


```
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data), 1,2)

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

p
```



polish plot

```
p + labs(title = "PCA of RNASeq Data", subtitle = "PC1 clearly separates wild-type from knock-out sampl
```


PCA of RNASeq Data
PC1 clearly separates wild−type from knock−out samples
ko1
wt1
ko5
ko4
wt2
ko3
wt5
wt4
wt3
ko2
2
0
−2
−10
−5
0
5
10
PC1 (92.6%)
PC2 (2.3%)
BIMM143 example data