Machine Learning 1

Vina Nguyen

2023-05-01

kmeans()

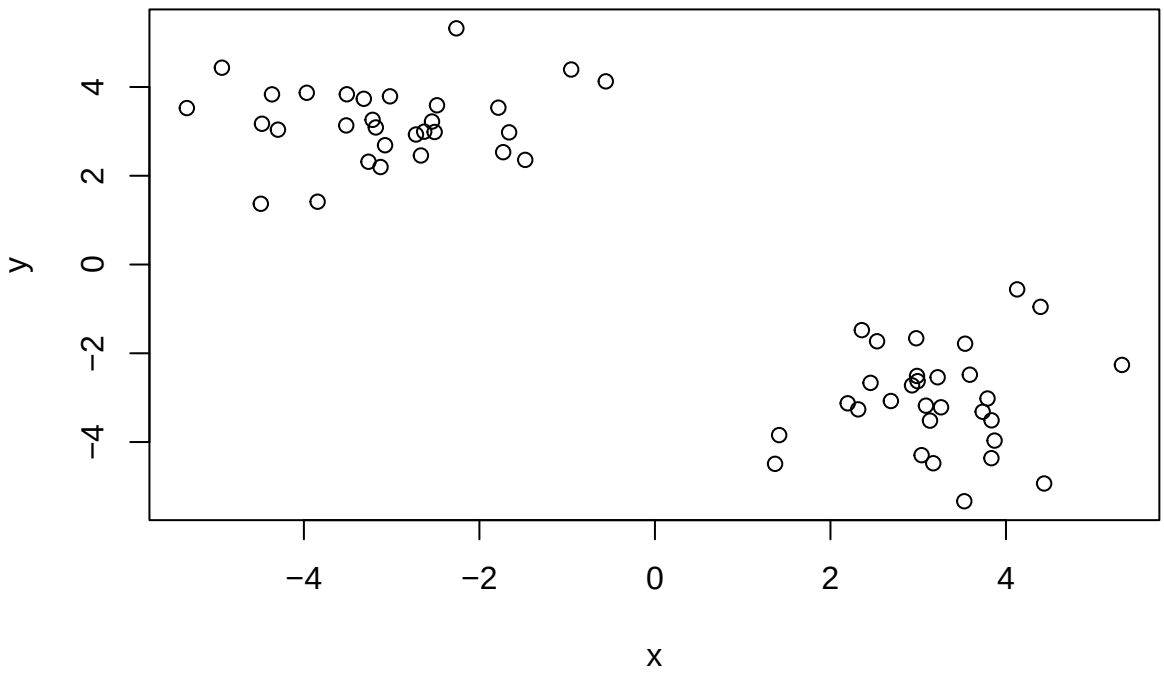
Demo of using kmeans() function in base R. First make up some data with a known structure.

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

```
##                x          y
##  [1,] -3.3173523  3.7341920
##  [2,] -5.3336907  3.5250975
##  [3,] -2.7228226  2.9293852
##  [4,] -3.5174919  3.1342142
##  [5,] -1.7838154  3.5342551
##  [6,] -3.2638389  2.3160530
##  [7,] -3.2170088  3.2599416
##  [8,] -1.6616059  2.9775092
##  [9,] -4.4908371  1.3688078
## [10,] -3.9670800  3.8699828
## [11,] -4.4776373  3.1715543
## [12,] -3.1266558  2.1965195
## [13,] -1.7291378  2.5311606
## [14,] -2.2625155  5.3214736
## [15,] -2.5102378  2.9863823
## [16,] -2.6289570  2.9932148
## [17,] -0.5599045  4.1271027
## [18,] -3.8433366  1.4155194
## [19,] -3.1804603  3.0880374
## [20,] -4.9338977  4.4347673
## [21,] -4.3635320  3.8341545
## [22,] -2.5403884  3.2215601
## [23,] -2.4830234  3.5880542
## [24,] -1.4780663  2.3577200
## [25,] -3.0186974  3.7895198
## [26,] -3.0751082  2.6884057
## [27,] -3.5106453  3.8355341
## [28,] -2.6666019  2.4567420
## [29,] -4.2955196  3.0384899
## [30,] -0.9540173  4.3939943
## [31,]  4.3939943 -0.9540173
## [32,]  3.0384899 -4.2955196
```

```
## [33,]  2.4567420 -2.6666019
## [34,]  3.8355341 -3.5106453
## [35,]  2.6884057 -3.0751082
## [36,]  3.7895198 -3.0186974
## [37,]  2.3577200 -1.4780663
## [38,]  3.5880542 -2.4830234
## [39,]  3.2215601 -2.5403884
## [40,]  3.8341545 -4.3635320
## [41,]  4.4347673 -4.9338977
## [42,]  3.0880374 -3.1804603
## [43,]  1.4155194 -3.8433366
## [44,]  4.1271027 -0.5599045
## [45,]  2.9932148 -2.6289570
## [46,]  2.9863823 -2.5102378
## [47,]  5.3214736 -2.2625155
## [48,]  2.5311606 -1.7291378
## [49,]  2.1965195 -3.1266558
## [50,]  3.1715543 -4.4776373
## [51,]  3.8699828 -3.9670800
## [52,]  1.3688078 -4.4908371
## [53,]  2.9775092 -1.6616059
## [54,]  3.2599416 -3.2170088
## [55,]  2.3160530 -3.2638389
## [56,]  3.5342551 -1.7838154
## [57,]  3.1342142 -3.5174919
## [58,]  2.9293852 -2.7228226
## [59,]  3.5250975 -5.3336907
## [60,]  3.7341920 -3.3173523
```

```
plot(x)
```



Now we have some made up data in 'x' let's see hpw kmeans works with this data.

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.030463  3.203978
## 2  3.203978 -3.030463
##
## 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 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 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 59.60032 59.60032
##  (between_SS / total_SS =  90.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
k$size
```

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

Q. How do we get to the cluster membership/assignment?

```
k$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 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
```

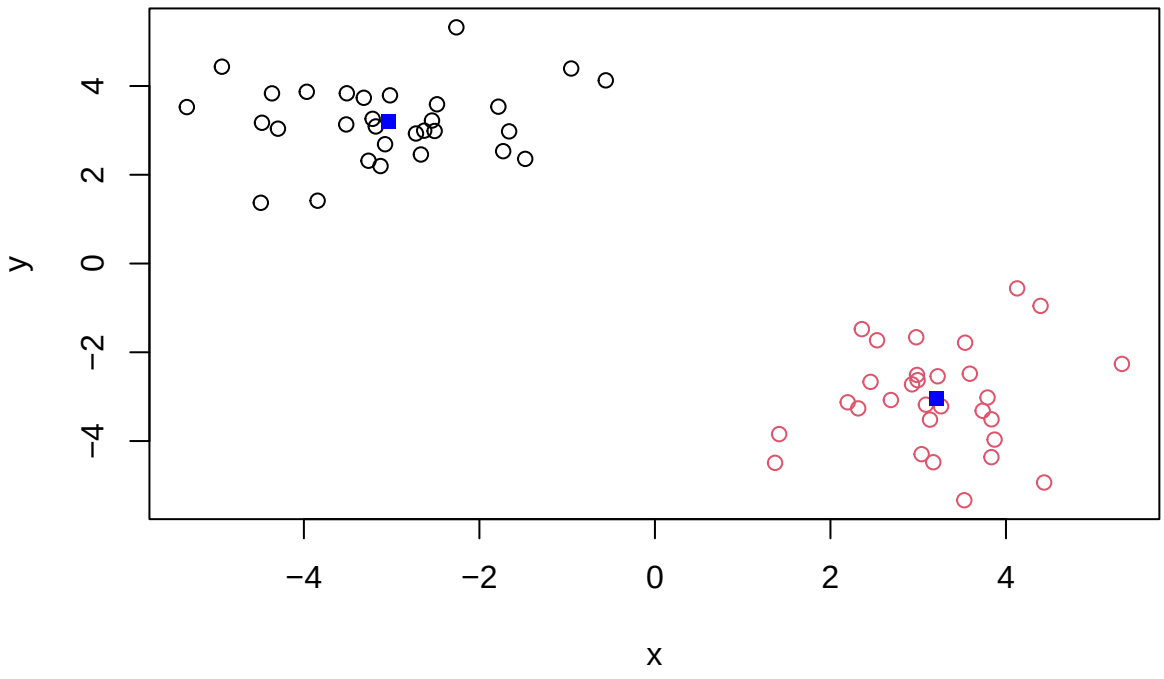
Q. What about cluster centers?

```
k$centers
```

```
##           x         y
## 1 -3.030463  3.203978
## 2  3.203978 -3.030463
```

Now we got to the main results. Let's use them to plot out data with the kmeans result.

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



hclust()

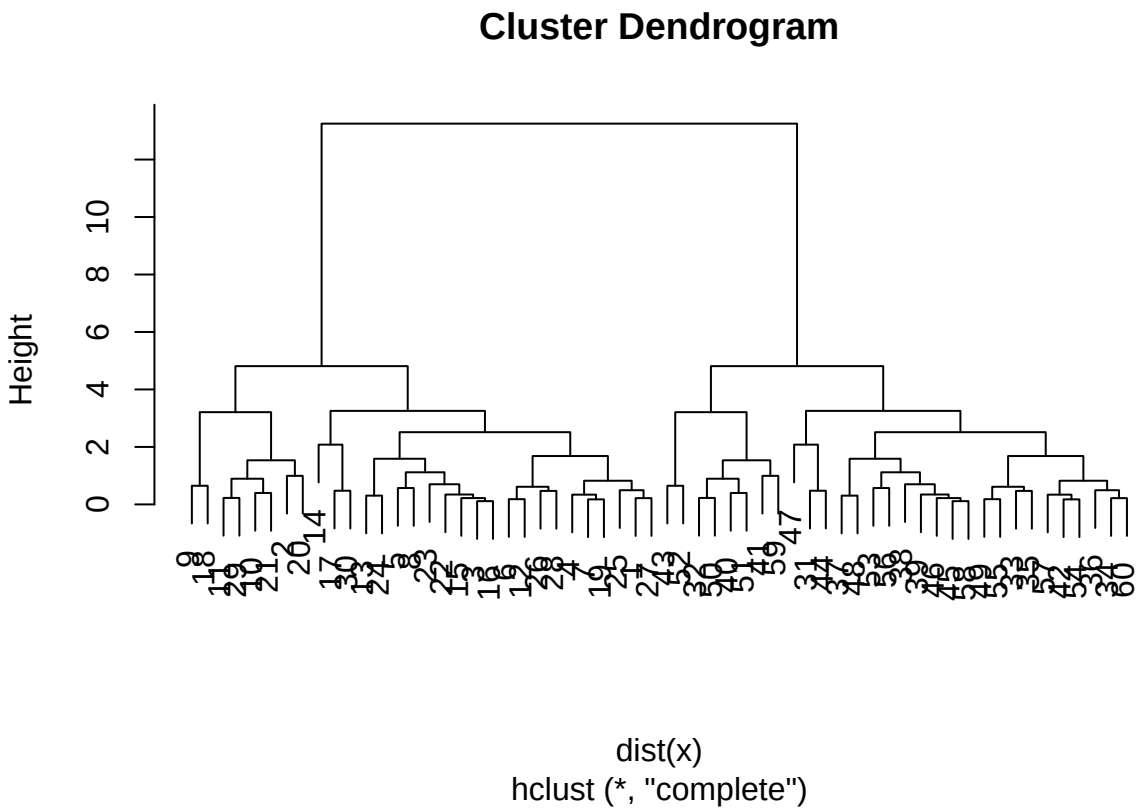
We will cluster the same data 'x' with the 'hclust()'. In this case, 'hclust()' requiresd a distaznce matrix as input.

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

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

Let's plot out hclust results.

```
plot(hc)
```



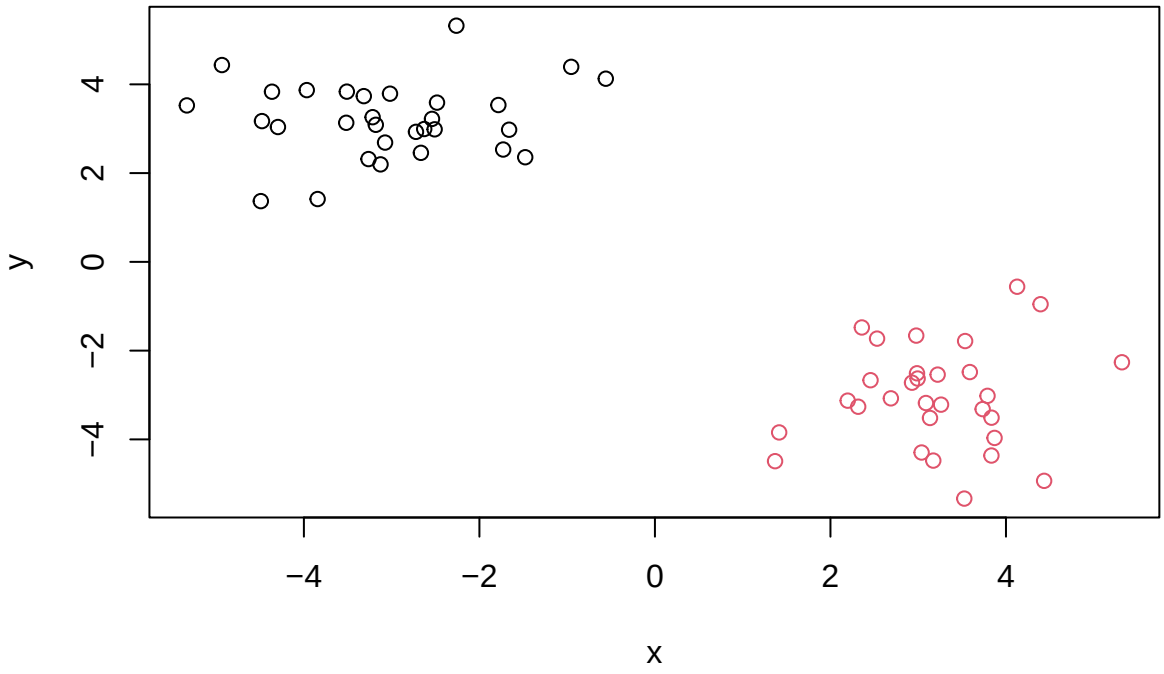
To get out cluster membership vector, we need to "cut' the tre with the 'cutree()'.

```
grps <- cutree(hc, h=8)
grps
```

```
##  [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
```

Now plot out data with the hclust() results.

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



Principal Components Analysis (PCA)

PCA of UK food Data

Read data from website and try a few visualizations.

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
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
## 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
```

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

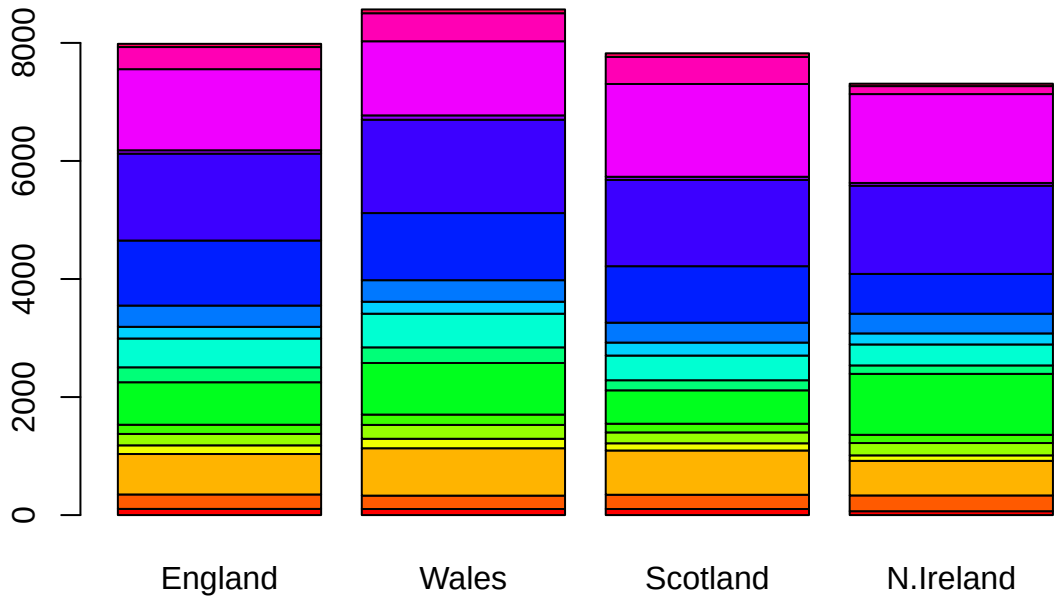
```
dim(x)
```

```
## [1] 17  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?

use argument setting row.names=1.

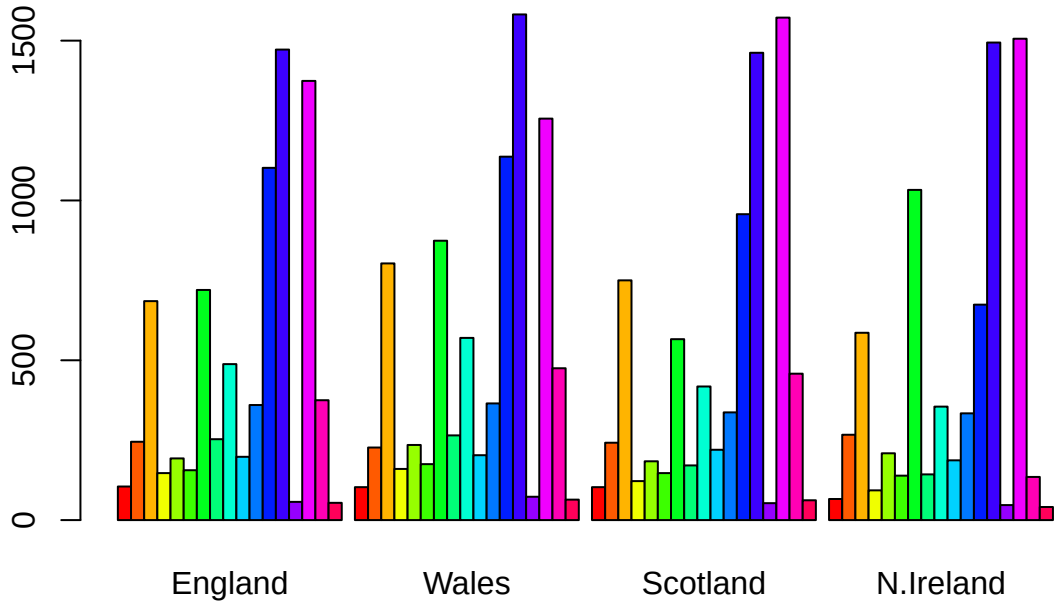
```
cols <- rainbow(nrow(x))
barplot(as.matrix(x), col=cols)
```



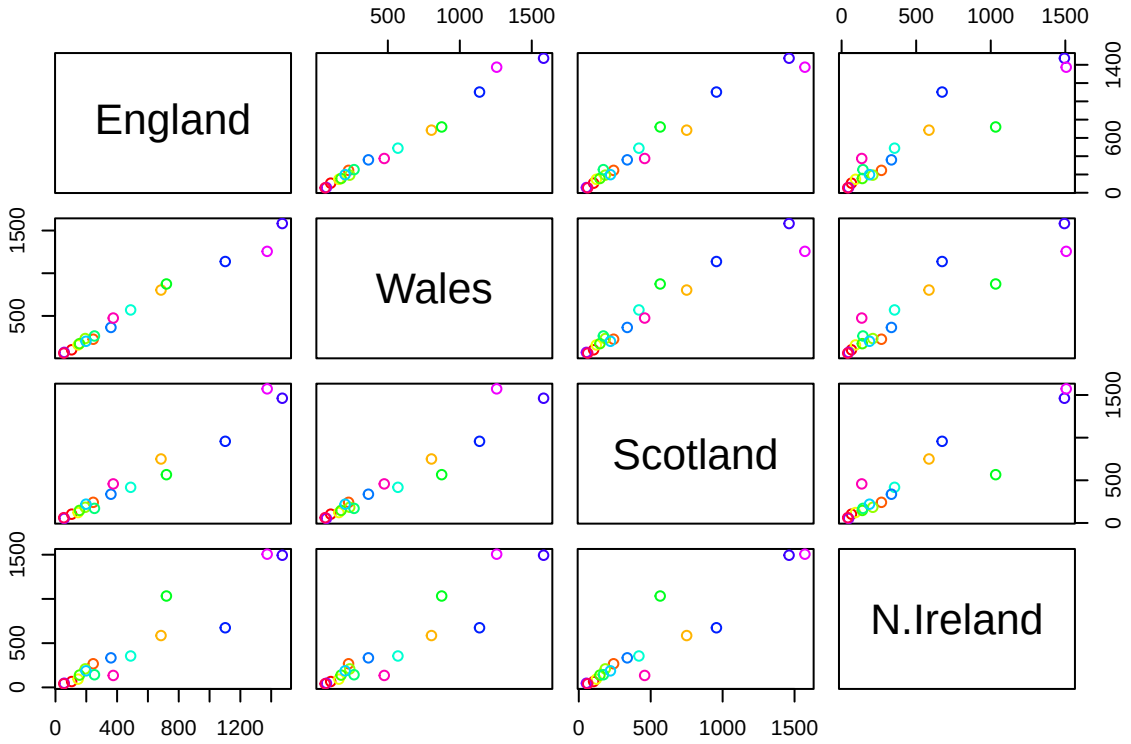
Q3: Changing what optional argument in the above barplot() function results in the following plot?

use beside=FALSE in barplot() code.

```
barplot(as.matrix(x), col=cols, beside=TRUE)
```




```
pairs(x, col=cols)
```



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?

Looking at the top left plots, everything on the x-axis represents England while everything on the y-axis represents Wales. If the value of the given point lies on the diagonal, it means that the values are the same for the x and y-axis.

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

N. Ireland points are more off (below) the diagonal.

The main base R PCA function is called 'prcomp()' and we will need to give it the transpose of our input data.

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

```
## Standard deviations (1, .., p=4):
## [1] 3.241502e+02 2.127478e+02 7.387622e+01 4.188568e-14
##
## Rotation (n x k) = (17 x 4):
##                              PC1          PC2         PC3          PC4
## Cheese               -0.056955380 -0.016012850 -0.02394295 -0.691718038
## Carcass_meat          0.047927628 -0.013915823 -0.06367111  0.635384915
```

```
## Other_meat          -0.258916658  0.015331138  0.55384854  0.198175921
## Fish                -0.084414983  0.050754947 -0.03906481 -0.015824630
## Fats_and_oils       -0.005193623  0.095388656  0.12522257  0.052347444
## Sugars              -0.037620983  0.043021699  0.03605745  0.014481347
## Fresh_potatoes       0.401402060  0.715017078  0.20668248 -0.151706089
## Fresh_Veg           -0.151849942  0.144900268 -0.21382237  0.056182433
## Other_Veg           -0.243593729  0.225450923  0.05332841 -0.080722623
## Processed_potatoes  -0.026886233 -0.042850761  0.07364902 -0.022618707
## Processed_Veg       -0.036488269  0.045451802 -0.05289191  0.009235001
## Fresh_fruit         -0.632640898  0.177740743 -0.40012865 -0.021899087
## Cereals             -0.047702858  0.212599678  0.35884921  0.084667257
## Beverages           -0.026187756  0.030560542  0.04135860 -0.011880823
## Soft_drinks          0.232244140 -0.555124311  0.16942648 -0.144367046
## Alcoholic_drinks    -0.463968168 -0.113536523  0.49858320 -0.115797605
## Confectionery       -0.029650201 -0.005949921  0.05232164 -0.003695024
```

There is a nice summary of how well PCA is doing.

```
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
```

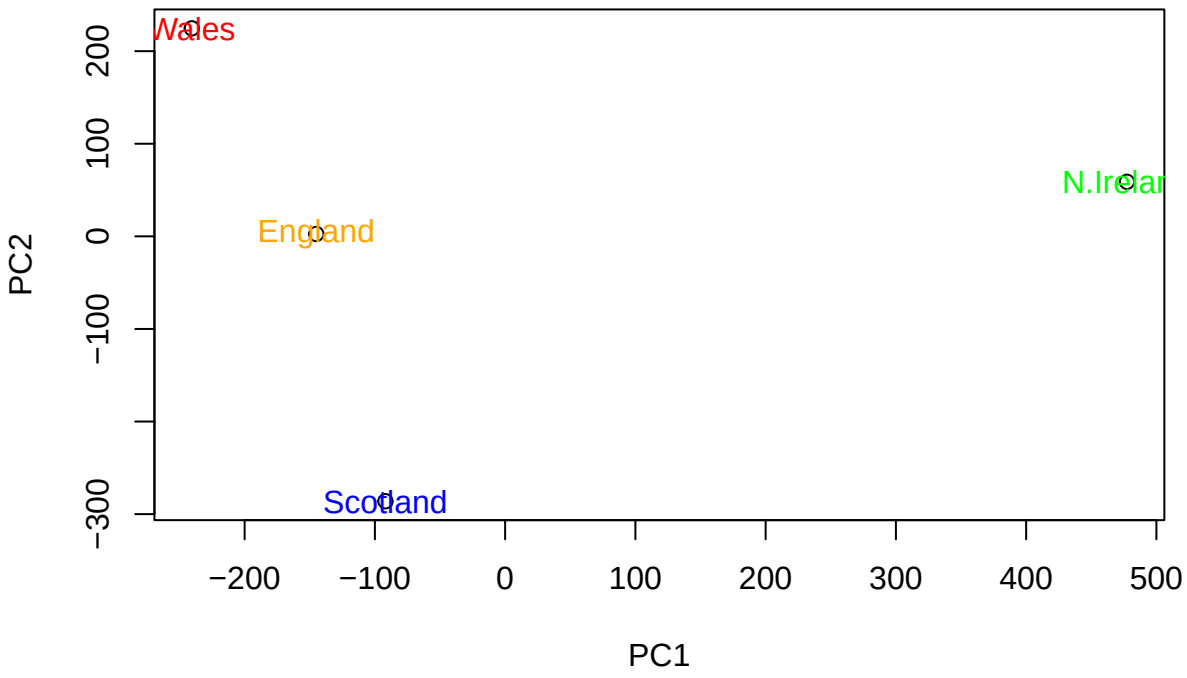
```
attributes(pca)
```

```
## $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. 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.

To make our new PCA plot (aka PCA score plot) we access 'pca$x'.

```
country_cols <- c("orange", "red", "blue", "green")
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
text(pca$x[,1], pca$x[,2], colnames(x), col=country_cols)
```



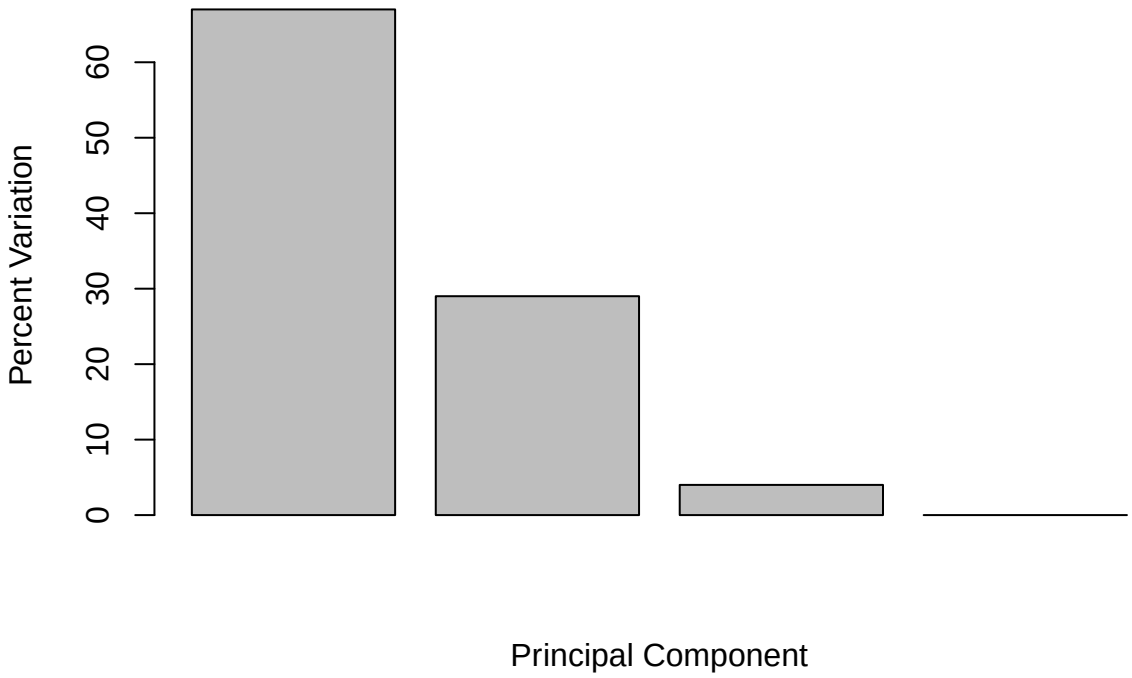

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

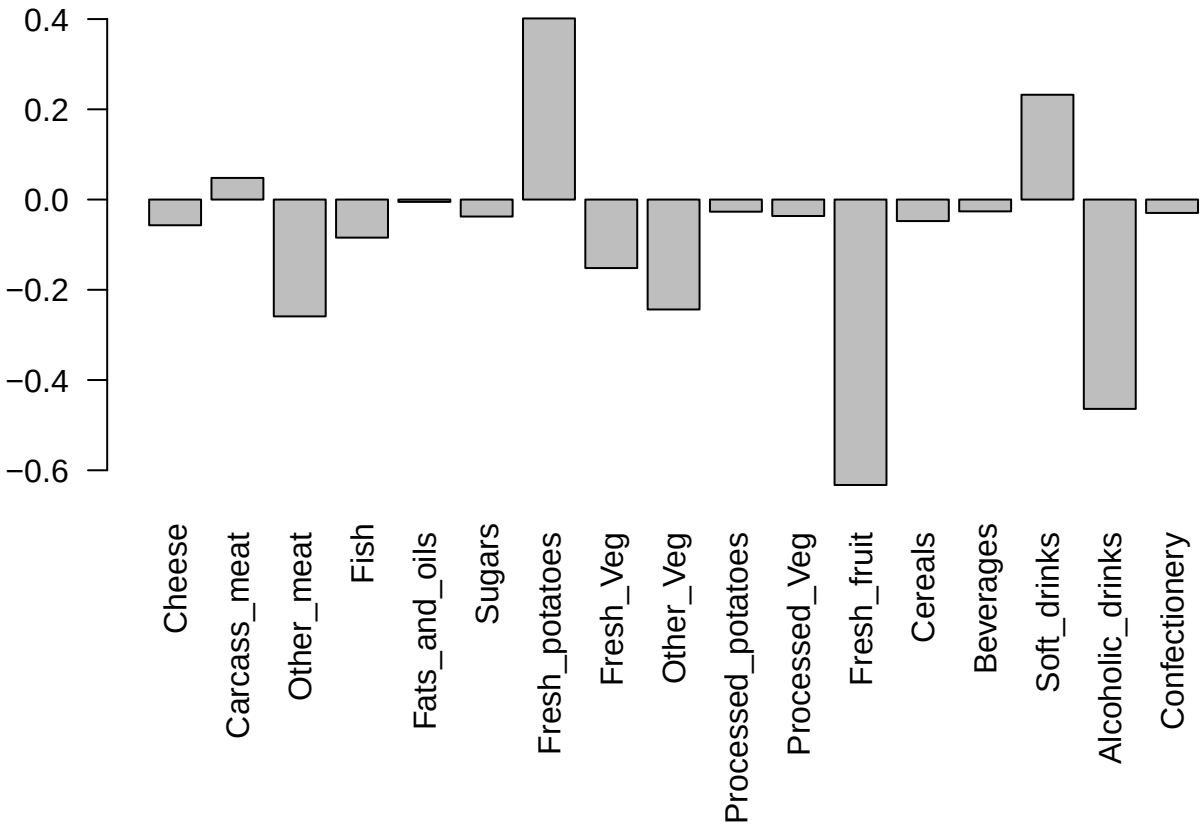
```
## or the second row here...
z <- summary(pca)
z$importance
```

```
##                             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
```

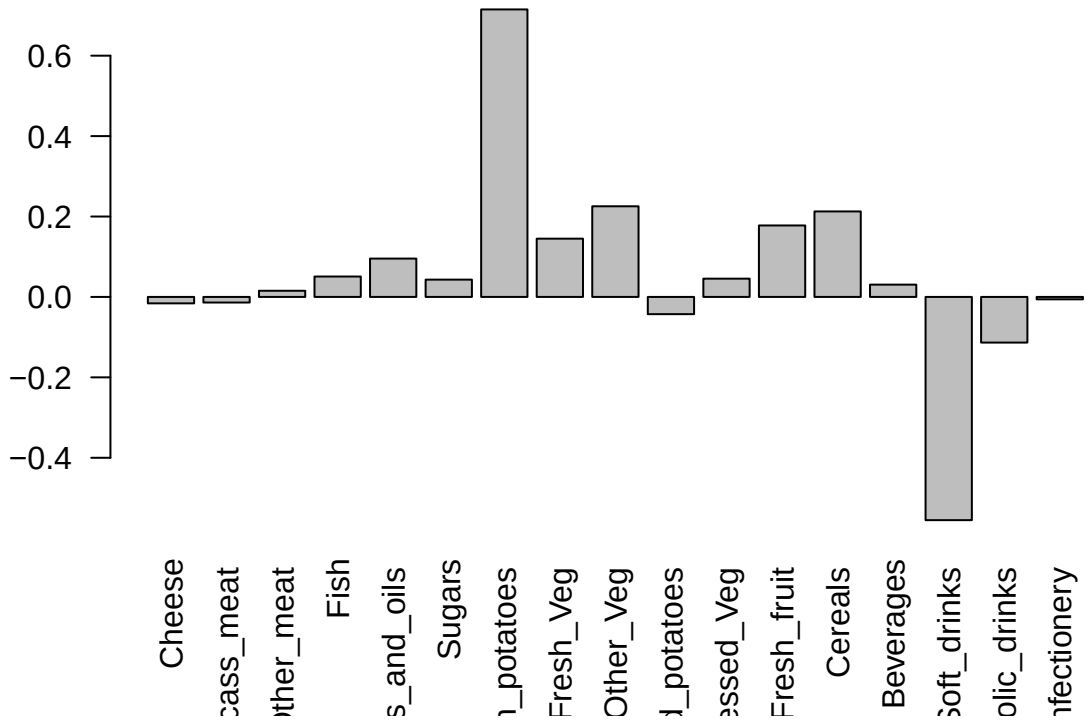
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```




```
## 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 )
```



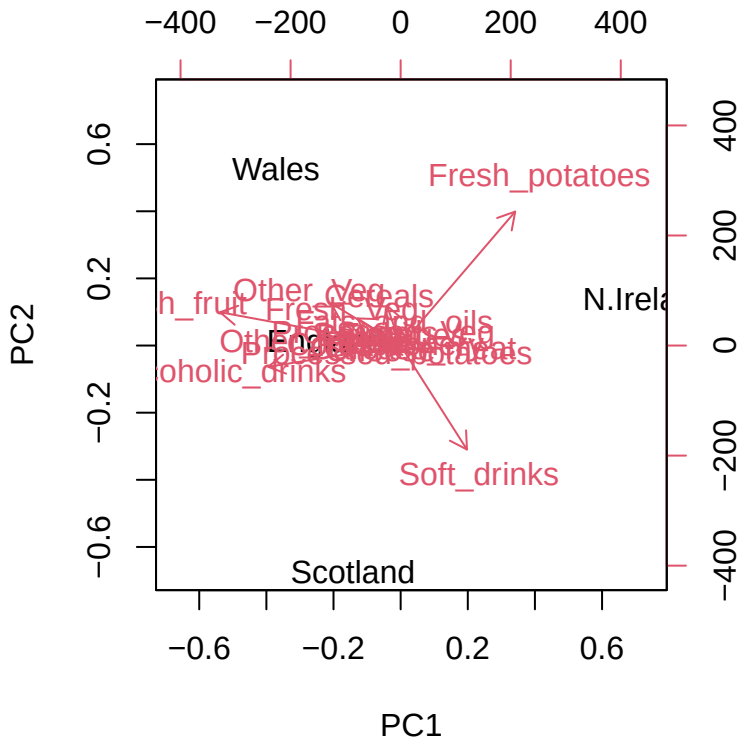

```
barplot( pca$rotation[,2], las=2 )
```



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

Fresh_potatoes and Other_Veg are the largest positive loading scores.

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



PCA of RNA-Seq Data

Read in data from website.

```
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
```

```
nrow(rna.data) #genes
```

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

```
ncol(rna.data) #samples
```

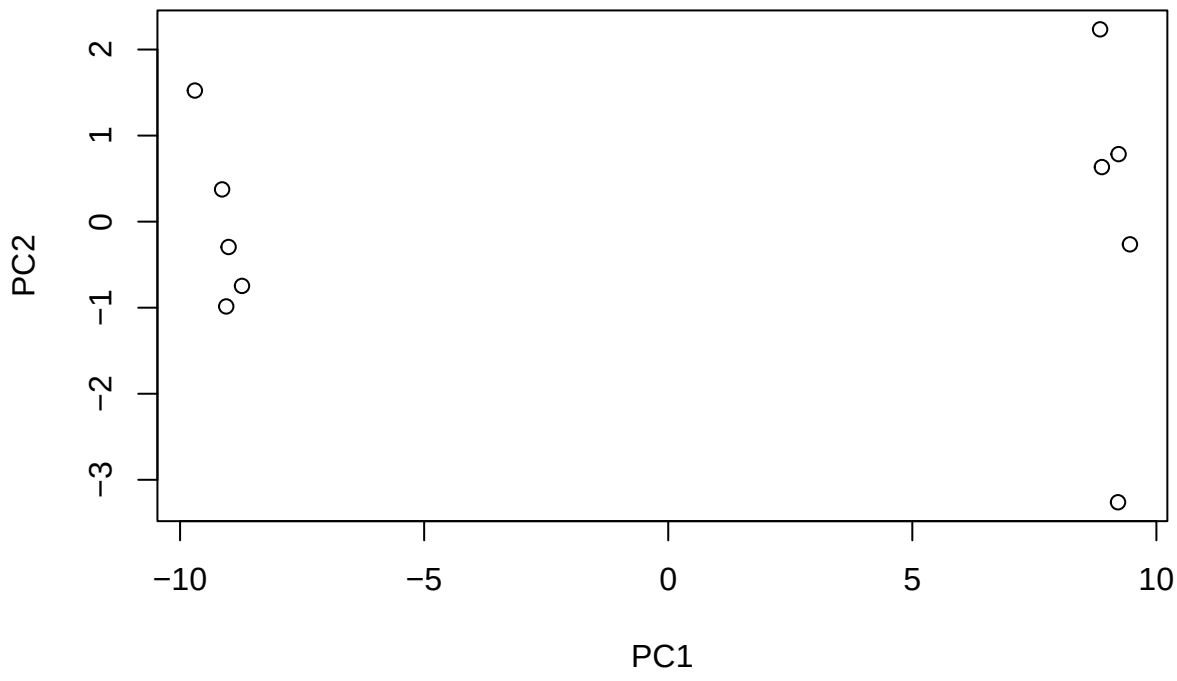
```
## [1] 10
```

```
pca <- prcomp( t(rna.data), scale=TRUE)
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
```

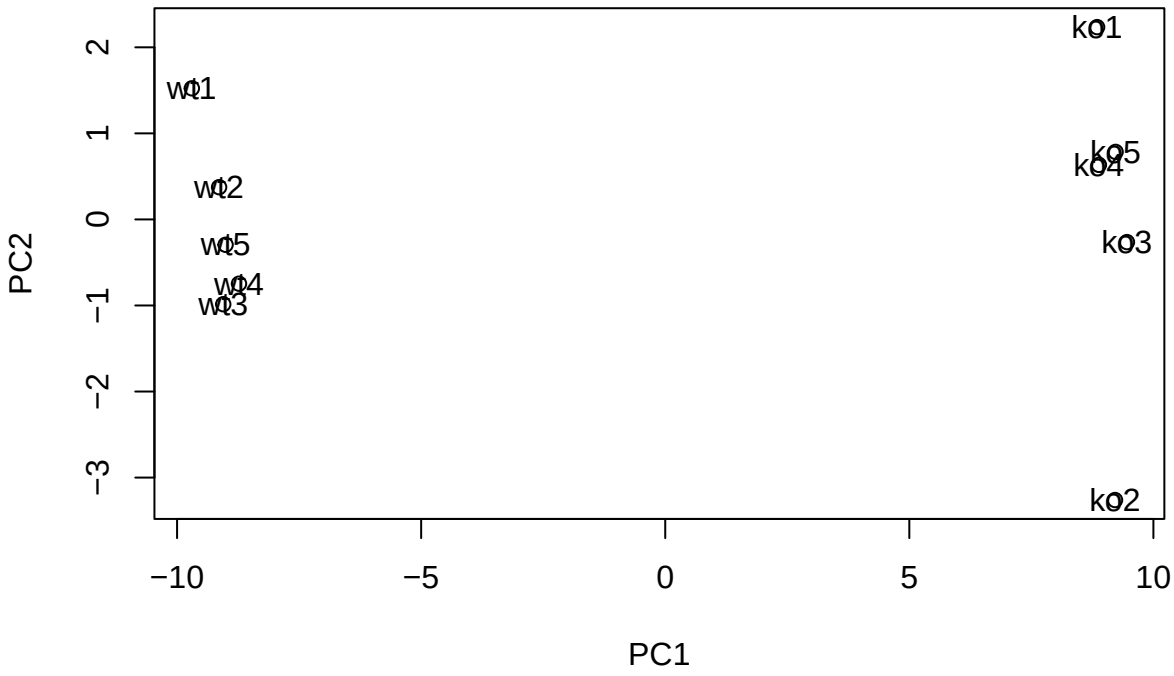
Do our PCA plot of this RNA-Seq.

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



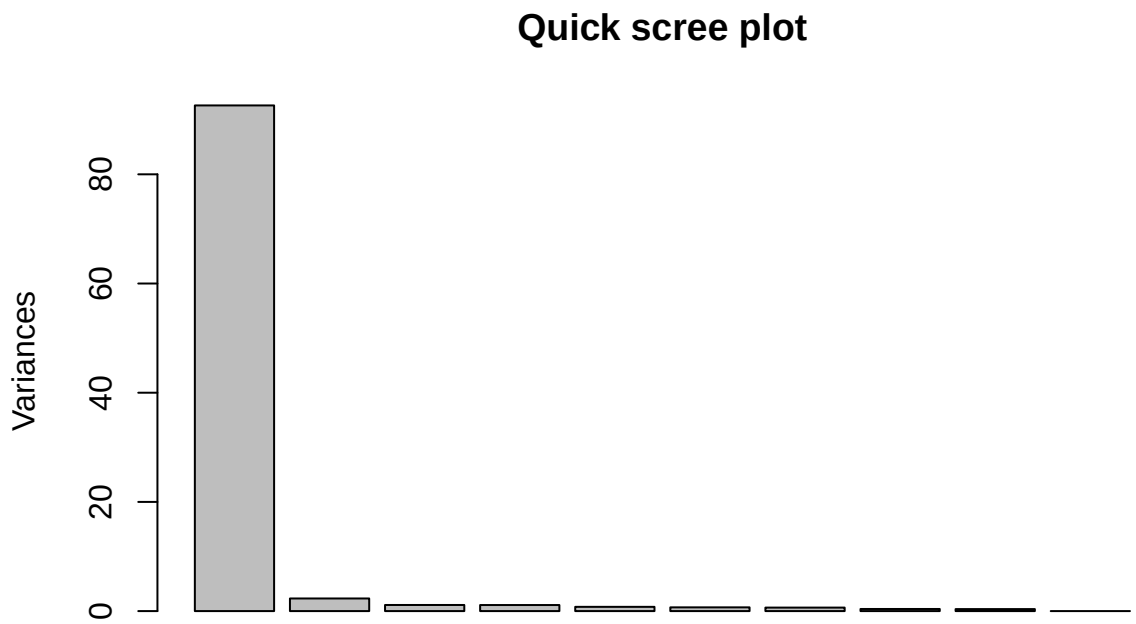
Let label the plot!

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
text(pca$x[,1], pca$x[,2], colnames(rna.data))
```



Scree-plot:

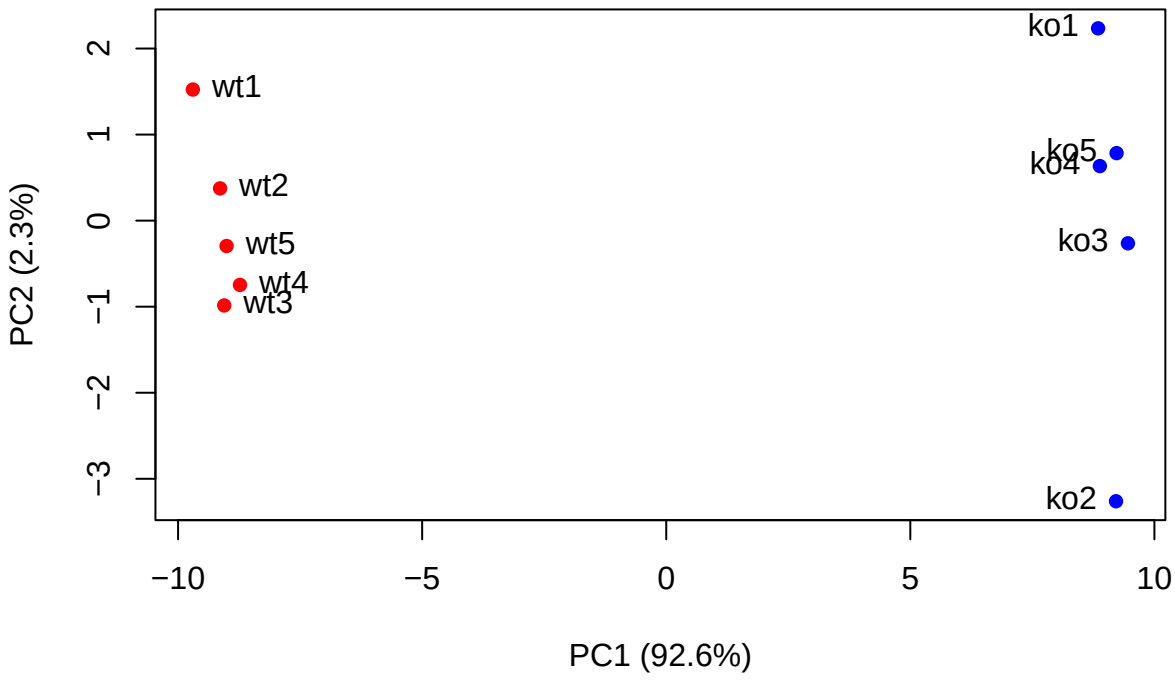
```
plot(pca, main="Quick scree plot")
```




```
## 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
```

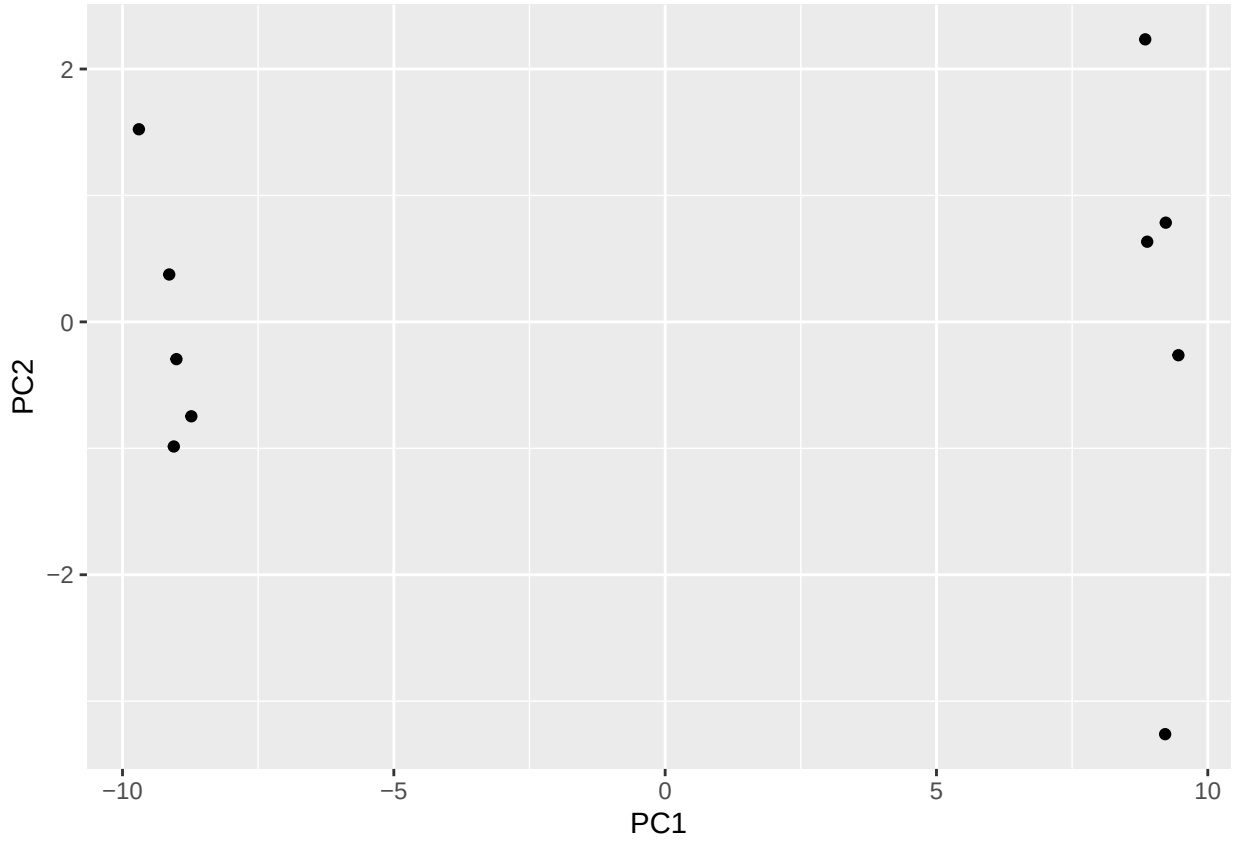
```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

```
## 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)))
```

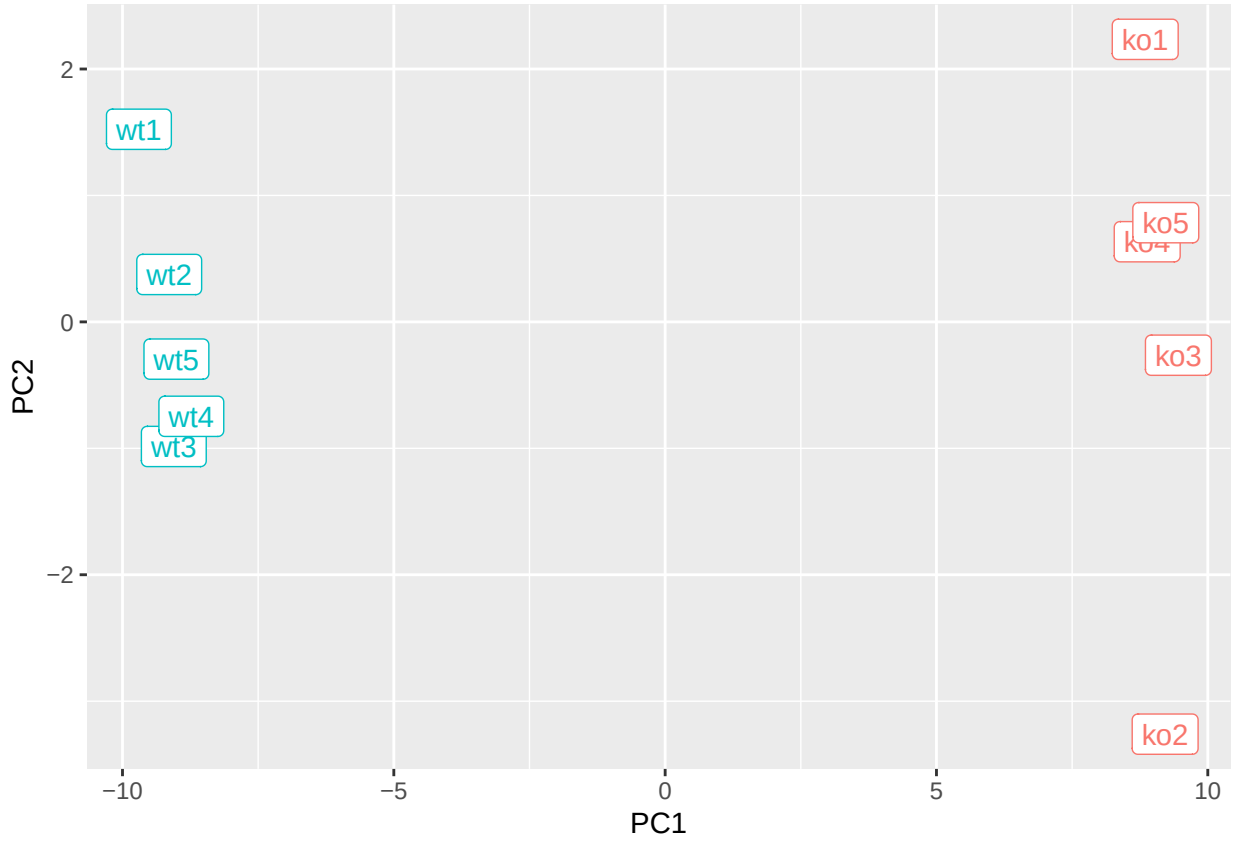


Making a ggplot using the ggplot2 package

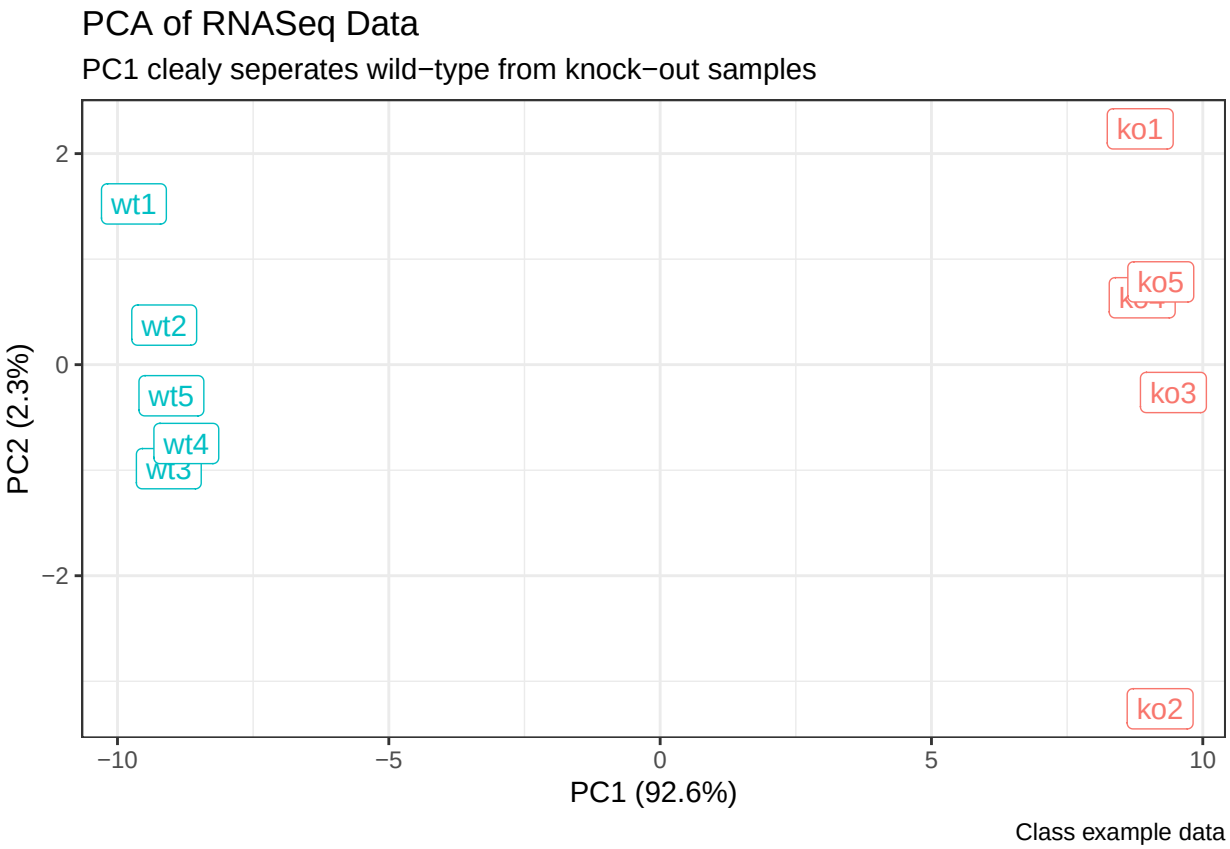
```
library(ggplot2)

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="Class 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%)
Class example data