Machine Learning 1

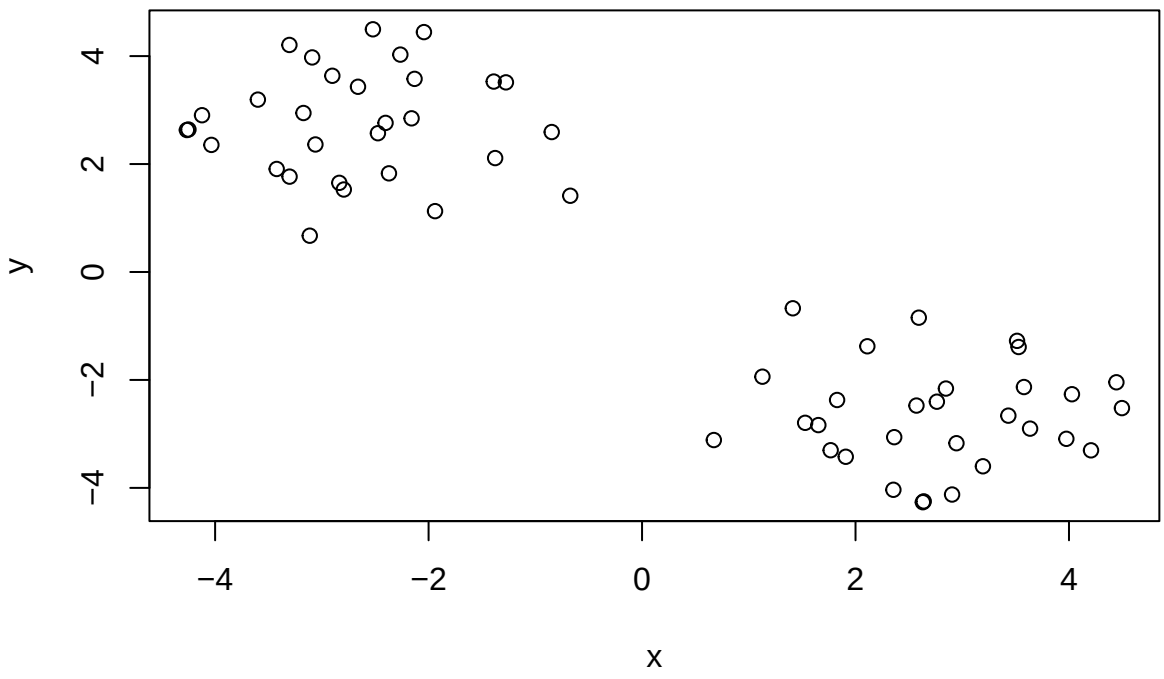
Stefanie Hodapp (PID: A53300084)

10/22/2021

Clustering methods

Kmeans clustering in R is done with the 'Kmeans()' function

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



Run 'kmeans()' and set k (centers) to 2 (i.e. the number of clusters we want), nstart to 20 (to run multiple times). You have to tell it how many clusters you want. Clustering vector tells you which cluster each element in your data set is in.

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  2.754913 -2.660100
## 2 -2.660100  2.754913
##
## 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 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] 55.88031 55.88031
##  (between_SS / total_SS =  88.7 %)
##
## 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/membership?

```
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 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
```

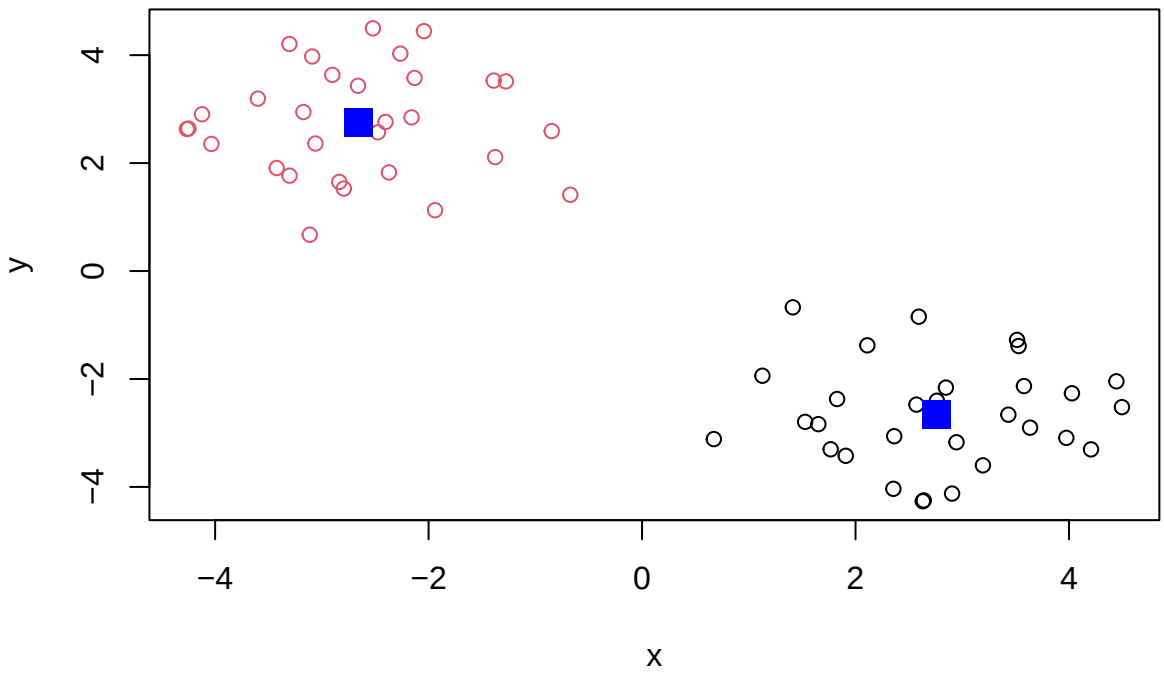
Q. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1  2.754913 -2.660100
## 2 -2.660100  2.754913
```

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

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



Hierarchical Clustering

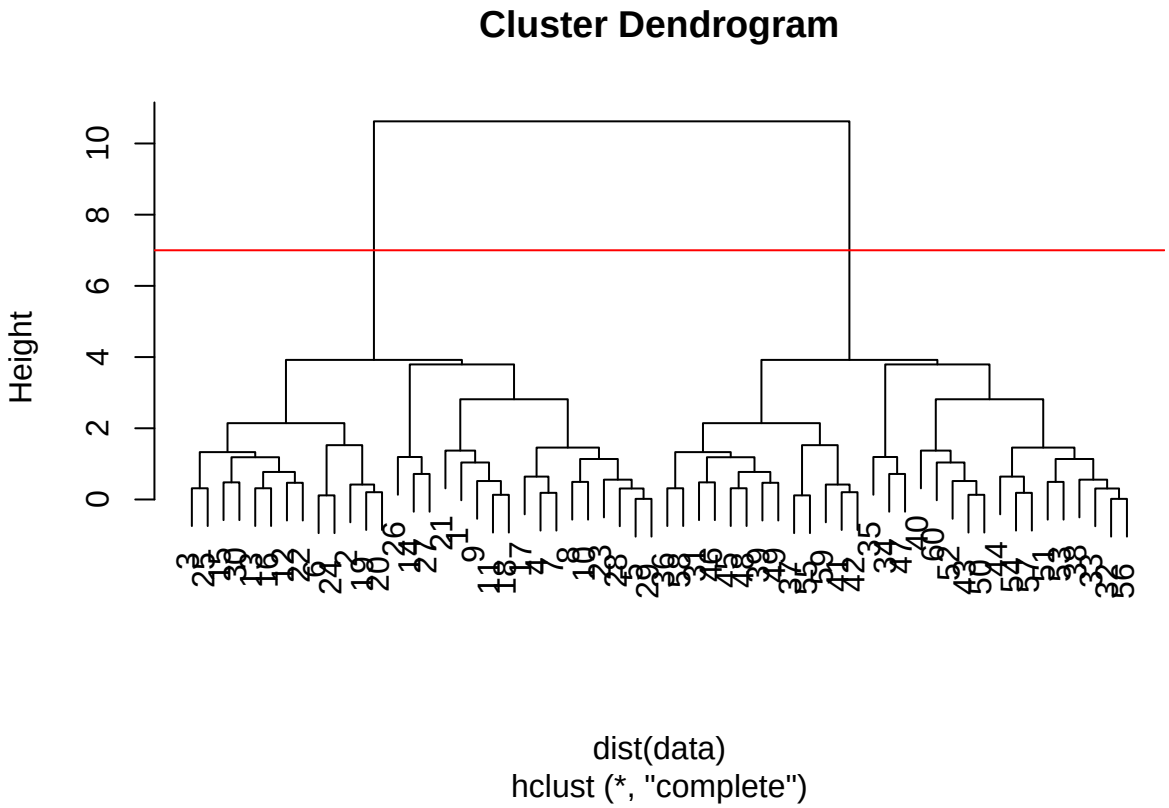
We will used the 'hclust()' function on the same data as before and see how this method works.

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

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

hclust has a plot method

```
plot(hc)
abline(h=7, col="red")
```



To find our membership vector we need to "cut" the tree and for this we use the 'cutree()' function and tell it the height to cut at.

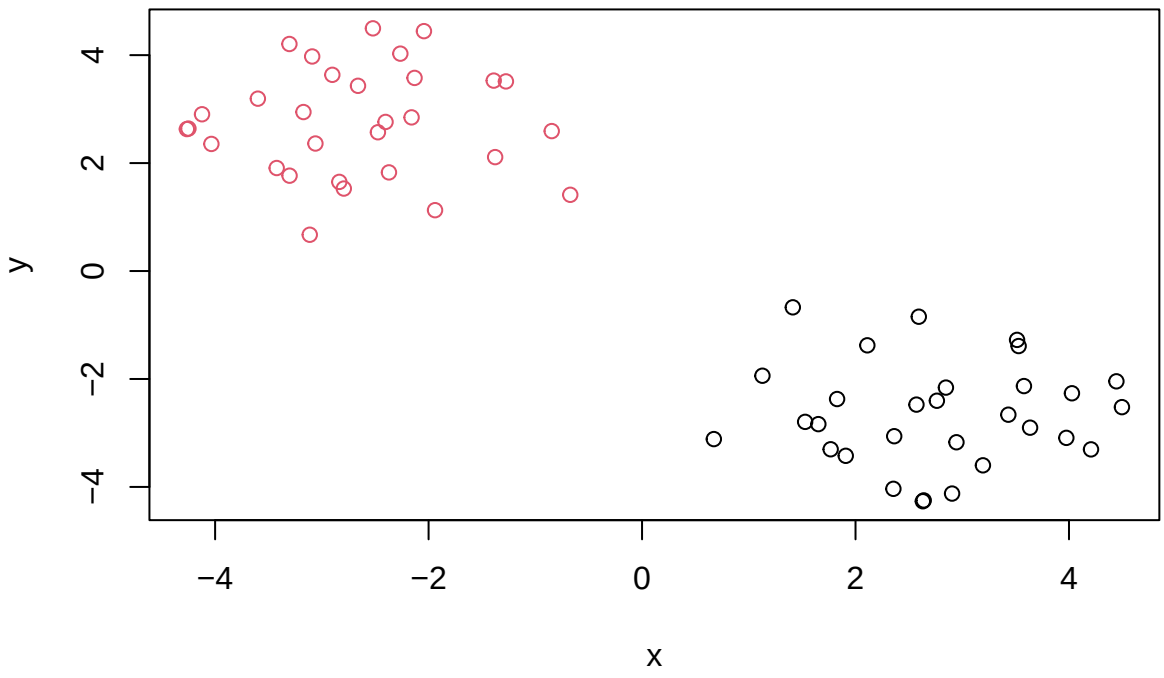
```
cutree(hc, h=7)
```

```
##  [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
```

We can also use 'cutree()' and state the number of clusters we want...

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

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



Principal Component Analysis (PCA)

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

```
nrow(x)
```

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

```
ncol(x)
```

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

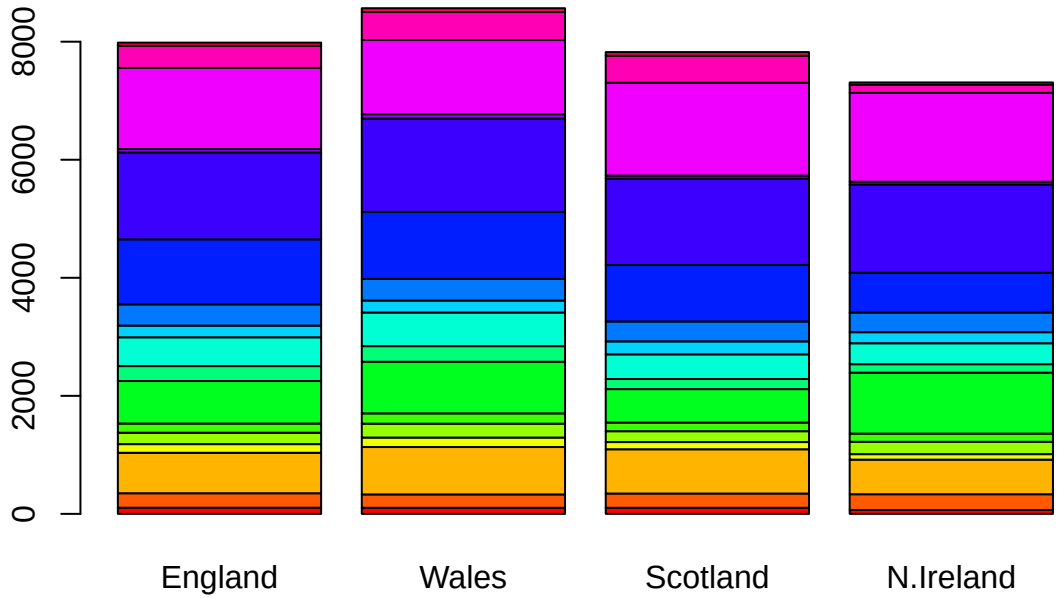
```
head(x)
```

```
##                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
```

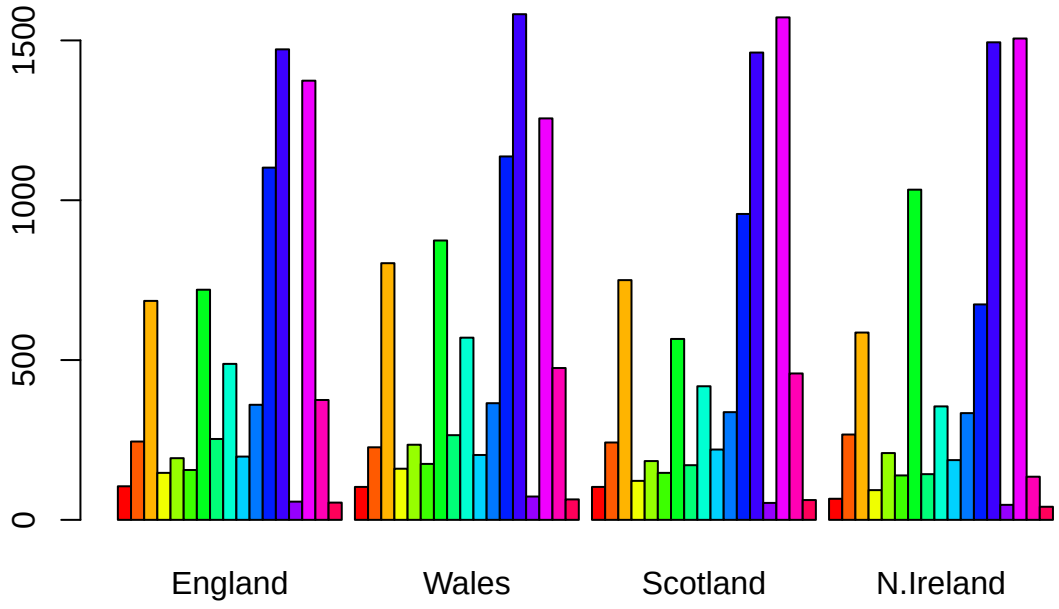
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
x
```

```
##                     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
```

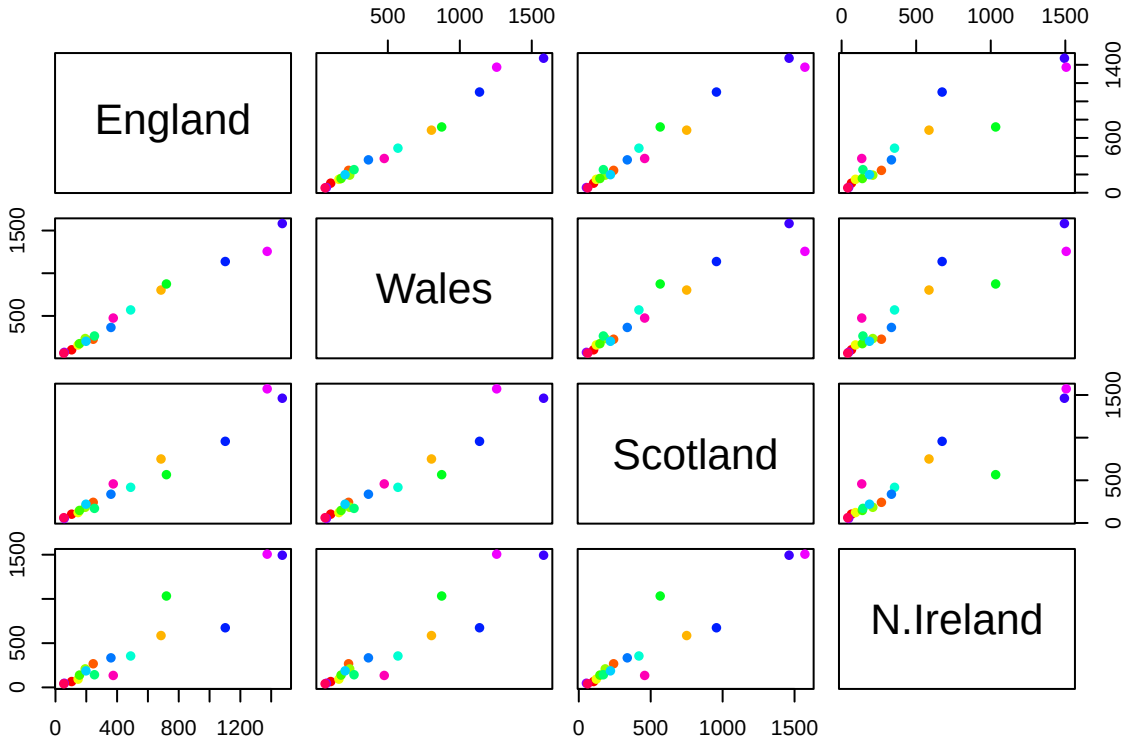
```
barplot(as.matrix(x), col=rainbow(17))
```



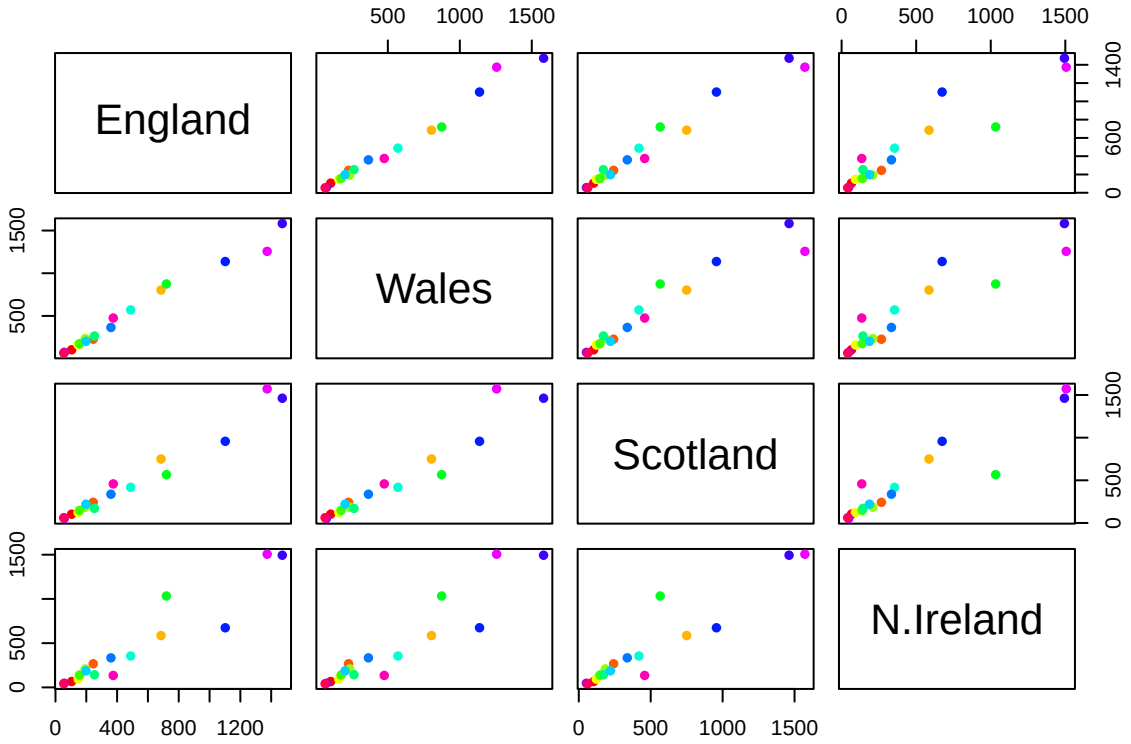
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```




```
mycols <- rainbow(nrow(x))
pairs(x, col=mycols, pch=16)
```




```
pairs(x, col=rainbow(nrow(x)), pch=16)
```



PCA to the rescue!

Here we will use the base R function for PCA, which is called 'prcomp()'. '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 't()' for this.

```
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
```

```
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
```

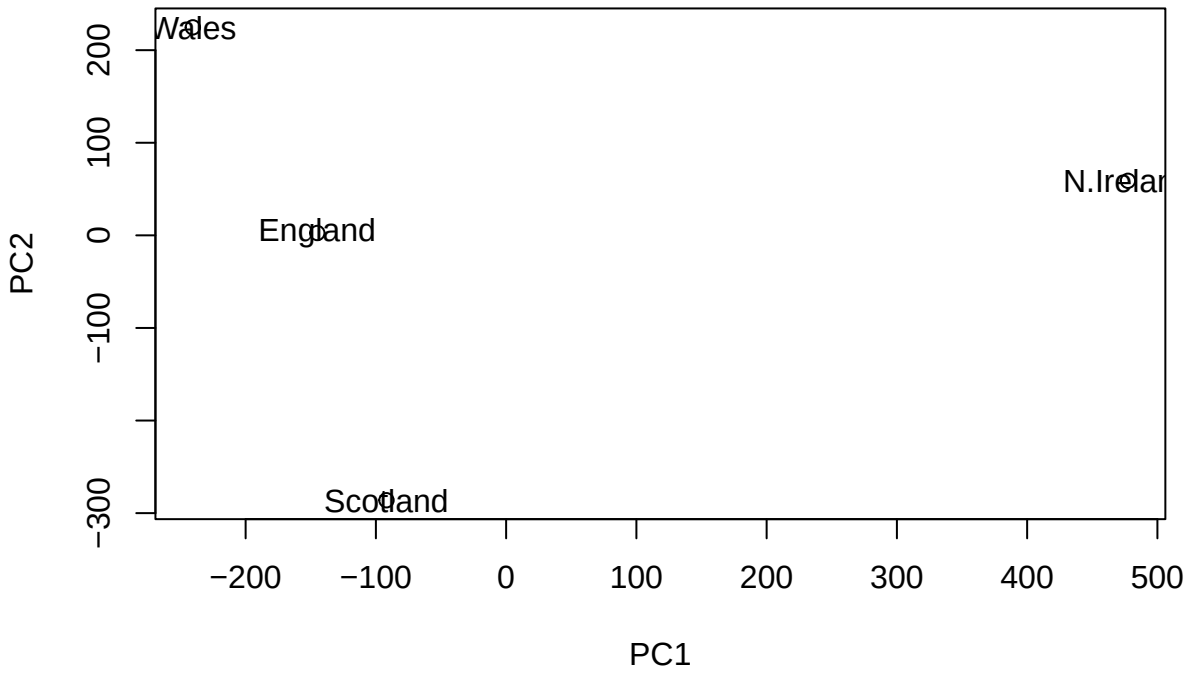
```
## Fats_and_oils       -0.005193623  0.095388656  0.12522257  0.052347444
## Sugars              -0.037620983  0.043021699  0.03605745  0.014481347
## Fresh_potatoes       0.401402060  0.715017078  0.20668248 -0.151706089
## Fresh_Veg           -0.151849942  0.144900268 -0.21382237  0.056182433
## 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
```

```
attributes(pca)
```

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

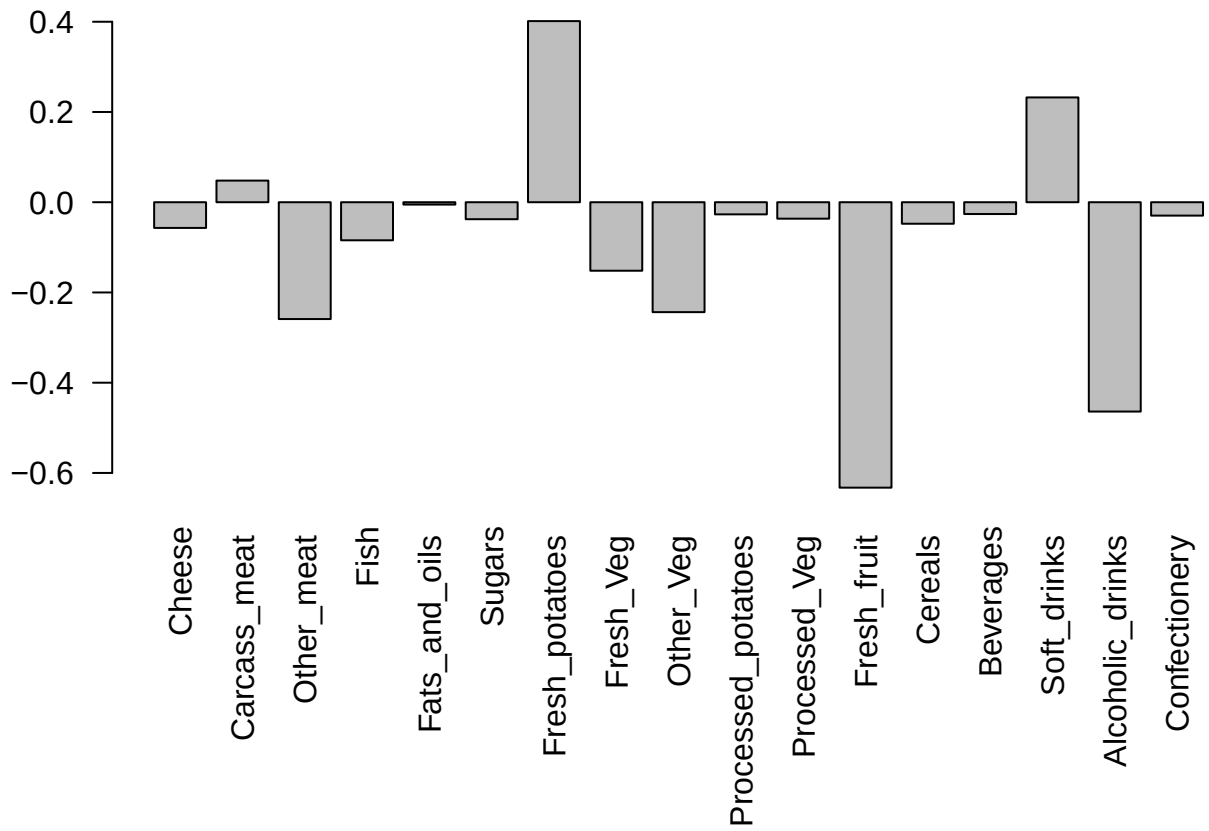
Plot PCA 1 vs PCA 2

```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels=colnames(x))
```



We can also examine the PCA "loadings", which tell us how much the original variables contribute to each new PC...

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

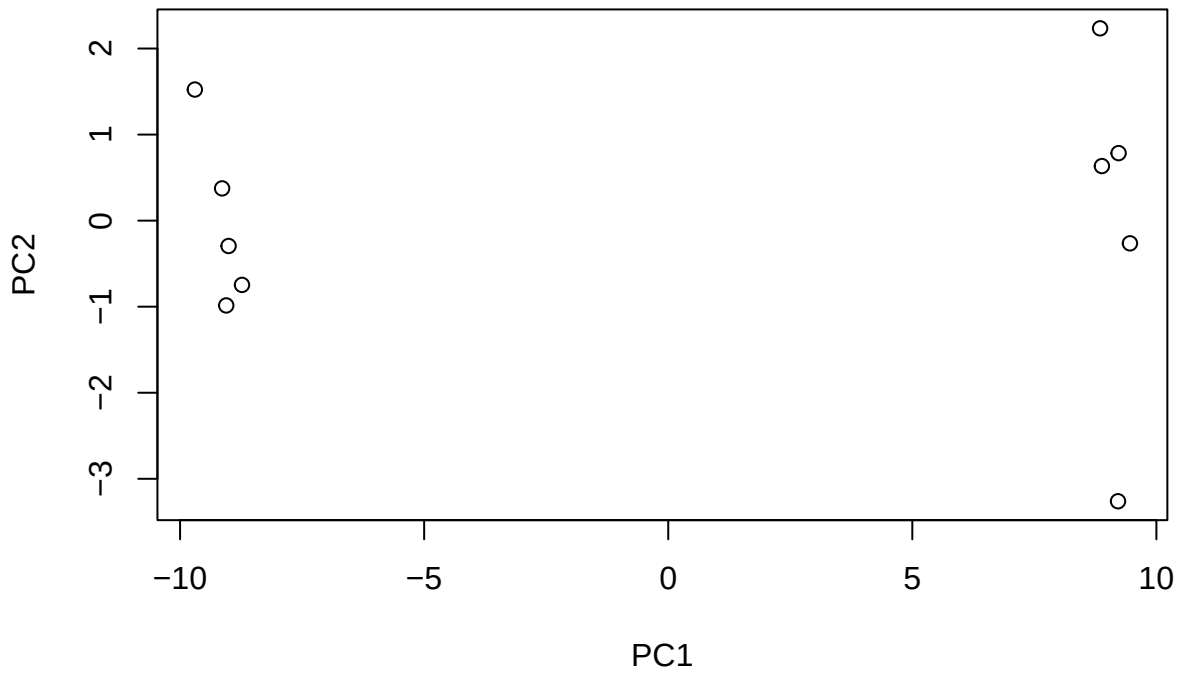


PCA of RNA sequencing data

```
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
```

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

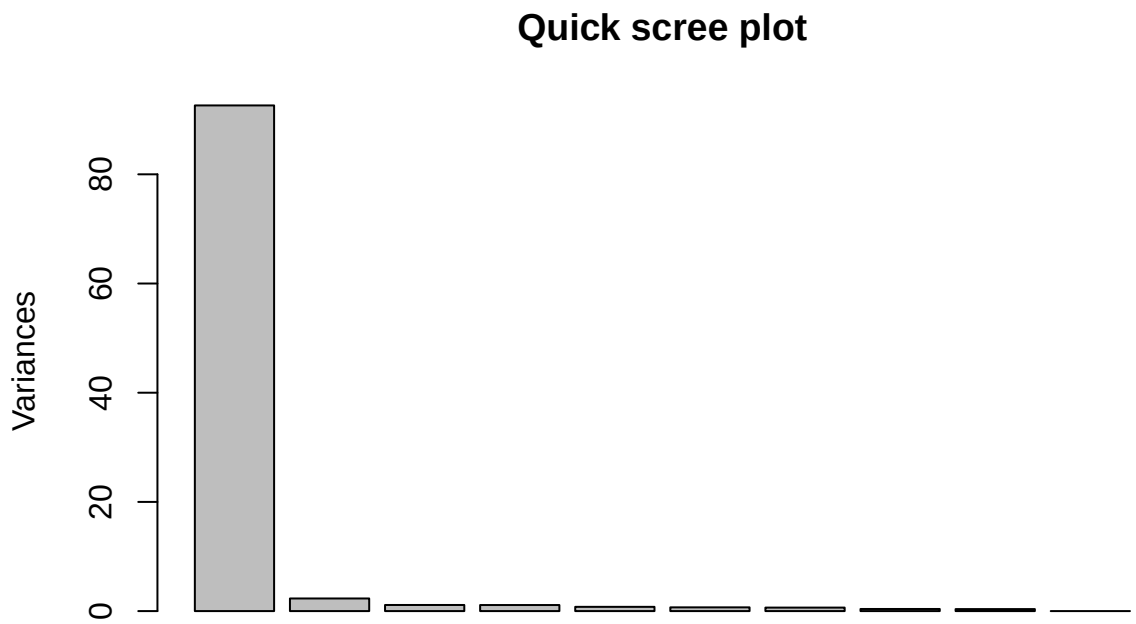



```
summary(pca_rna)
```

```
## 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
```

Calling the plot function on our prcomp data will show which PC captures the most variance in our data.

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

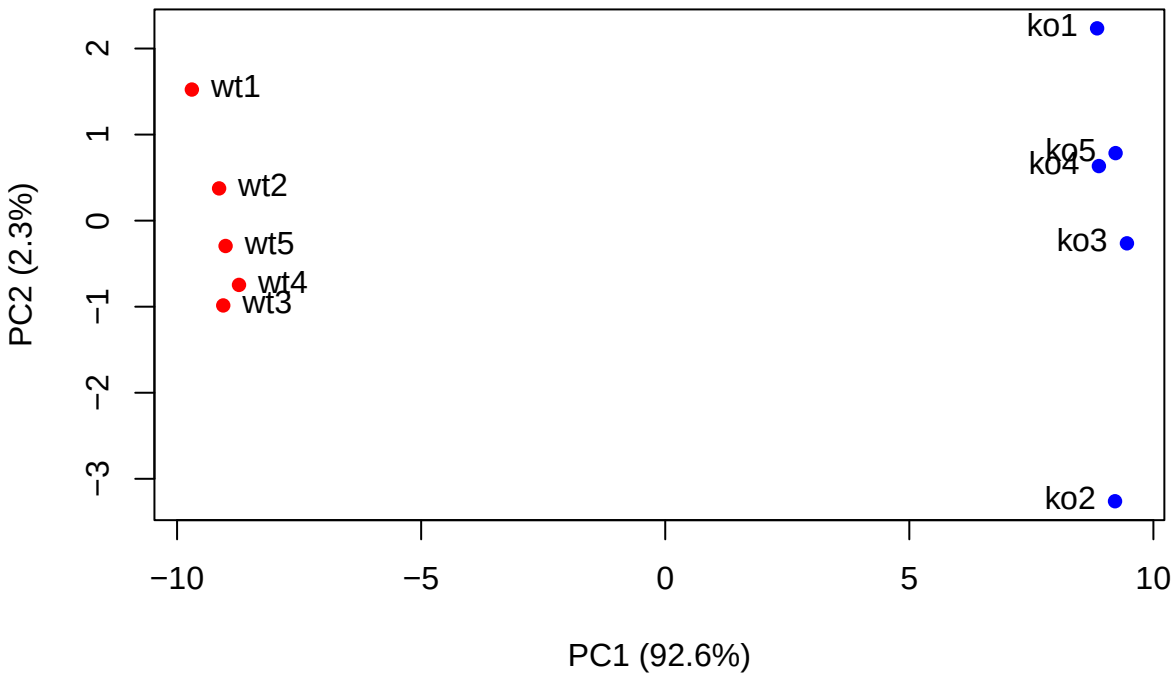



```
# Most of our variability is in PC 1
```

```
## Variance captured per PC
pca.var <- pca_rna$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
```

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

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

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


ko1
wt1
ko5
ko4
wt2
ko3
wt5
wt4
wt3
ko2
PC2 (2.3%)
2
1
0
−1
−2
−3
−10
−5
0
5
10
PC1 (92.6%)