

Class 08

Claire Chapman

10/24/2021

Clustering methods

kmeans()

First lets make up some data to cluster

```
tmp <- c(rnorm(30, 3), rnorm(30, -3))
tmp
```

```
## [1] 3.9241878 1.3206100 4.0999711 2.2322914 2.1214606 3.2730149
## [7] 2.8115440 1.3685479 0.7982468 2.5518494 1.0092300 2.8150751
## [13] 4.3688629 2.8202184 3.5779398 3.6754661 4.0324214 3.9107860
## [19] 3.3741663 3.9741656 2.0368022 4.2936020 1.4406627 4.4910550
## [25] 4.6004778 2.5265165 1.8524990 2.8998130 3.8515308 4.2320520
## [31] -3.2671142 -2.5143950 -4.0083676 -4.6899541 -3.3736933 -2.7754659
## [37] -4.0563220 -2.7541616 -3.2248000 -3.0113572 -4.7165608 -2.6165462
## [43] -2.6650563 -4.5773375 -3.0944488 -4.2278162 -3.3683348 -4.0514618
## [49] -2.8208647 -2.8658525 -0.4831573 -3.1373335 -1.1486991 -2.4184836
## [55] -2.4883612 -3.1823817 -2.0083242 -3.0879145 -4.6575562 -3.6366241
```

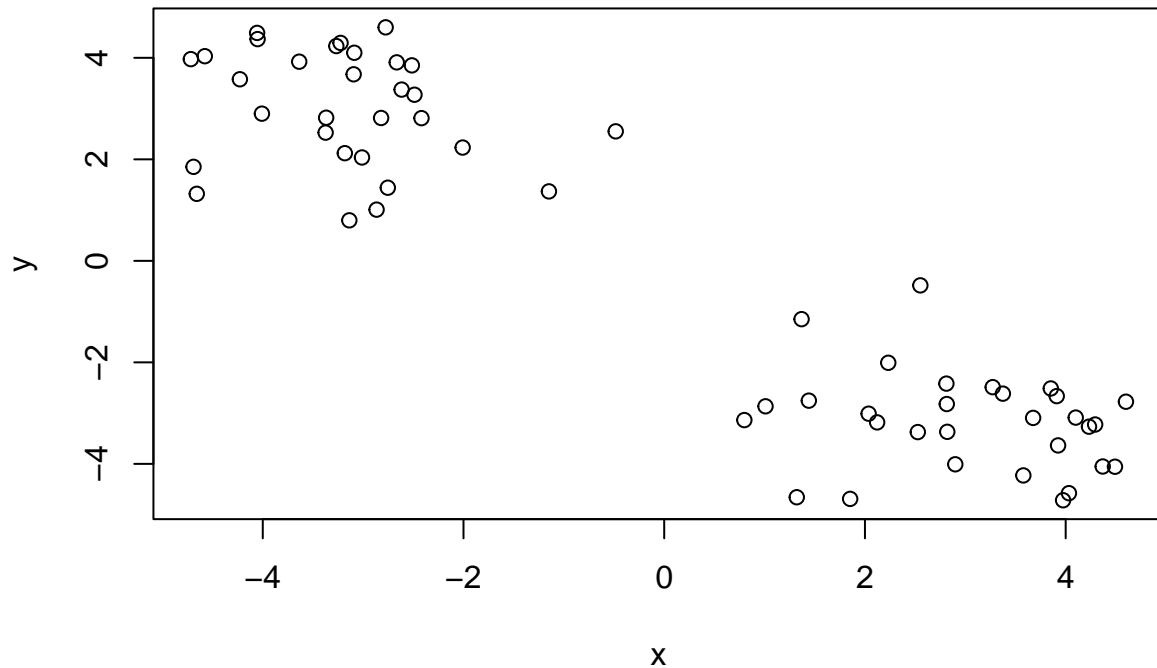
```
data <- cbind(x = tmp, y = rev(tmp))
data
```

```
##           x           y
## [1,] 3.9241878 -3.6366241
## [2,] 1.3206100 -4.6575562
## [3,] 4.0999711 -3.0879145
## [4,] 2.2322914 -2.0083242
## [5,] 2.1214606 -3.1823817
## [6,] 3.2730149 -2.4883612
## [7,] 2.8115440 -2.4184836
## [8,] 1.3685479 -1.1486991
## [9,] 0.7982468 -3.1373335
## [10,] 2.5518494 -0.4831573
## [11,] 1.0092300 -2.8658525
## [12,] 2.8150751 -2.8208647
## [13,] 4.3688629 -4.0514618
## [14,] 2.8202184 -3.3683348
```

```
## [15,] 3.5779398 -4.2278162
## [16,] 3.6754661 -3.0944488
## [17,] 4.0324214 -4.5773375
## [18,] 3.9107860 -2.6650563
## [19,] 3.3741663 -2.6165462
## [20,] 3.9741656 -4.7165608
## [21,] 2.0368022 -3.0113572
## [22,] 4.2936020 -3.2248000
## [23,] 1.4406627 -2.7541616
## [24,] 4.4910550 -4.0563220
## [25,] 4.6004778 -2.7754659
## [26,] 2.5265165 -3.3736933
## [27,] 1.8524990 -4.6899541
## [28,] 2.8998130 -4.0083676
## [29,] 3.8515308 -2.5143950
## [30,] 4.2320520 -3.2671142
## [31,] -3.2671142 4.2320520
## [32,] -2.5143950 3.8515308
## [33,] -4.0083676 2.8998130
## [34,] -4.6899541 1.8524990
## [35,] -3.3736933 2.5265165
## [36,] -2.7754659 4.6004778
## [37,] -4.0563220 4.4910550
## [38,] -2.7541616 1.4406627
## [39,] -3.2248000 4.2936020
## [40,] -3.0113572 2.0368022
## [41,] -4.7165608 3.9741656
## [42,] -2.6165462 3.3741663
## [43,] -2.6650563 3.9107860
## [44,] -4.5773375 4.0324214
## [45,] -3.0944488 3.6754661
## [46,] -4.2278162 3.5779398
## [47,] -3.3683348 2.8202184
## [48,] -4.0514618 4.3688629
## [49,] -2.8208647 2.8150751
## [50,] -2.8658525 1.0092300
## [51,] -0.4831573 2.5518494
## [52,] -3.1373335 0.7982468
## [53,] -1.1486991 1.3685479
## [54,] -2.4184836 2.8115440
## [55,] -2.4883612 3.2730149
## [56,] -3.1823817 2.1214606
## [57,] -2.0083242 2.2322914
## [58,] -3.0879145 4.0999711
## [59,] -4.6575562 1.3206100
## [60,] -3.6366241 3.9241878
```

Using just our eyes, we can assume that we should get two clusters

```
plot(data)
```



Run `kmeans()` The argument `k` tells how many clusters you are asking for. The argument `nstarts` tells how many iterations

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

```
## K-means clustering with 2 clusters of sizes 30, 30  
##  
## Cluster means:  
##           x          y  
## 1   3.009502 -3.164292  
## 2  -3.164292   3.009502  
##  
## 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  
##  
## Within cluster sum of squares by cluster:  
## [1] 65.19934 65.19934  
## (between_SS / total_SS = 89.8 %)  
##  
## Available components:  
##  
## [1] "cluster"      "centers"       "totss"         "withinss"      "tot.withinss"  
## [6] "betweenss"    "size"          "iter"          "ifault"
```

Q. How many points are in each cluster?

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

Thirty points in each cluster

Q. What “component” of your result object details cluster assignment/membership?

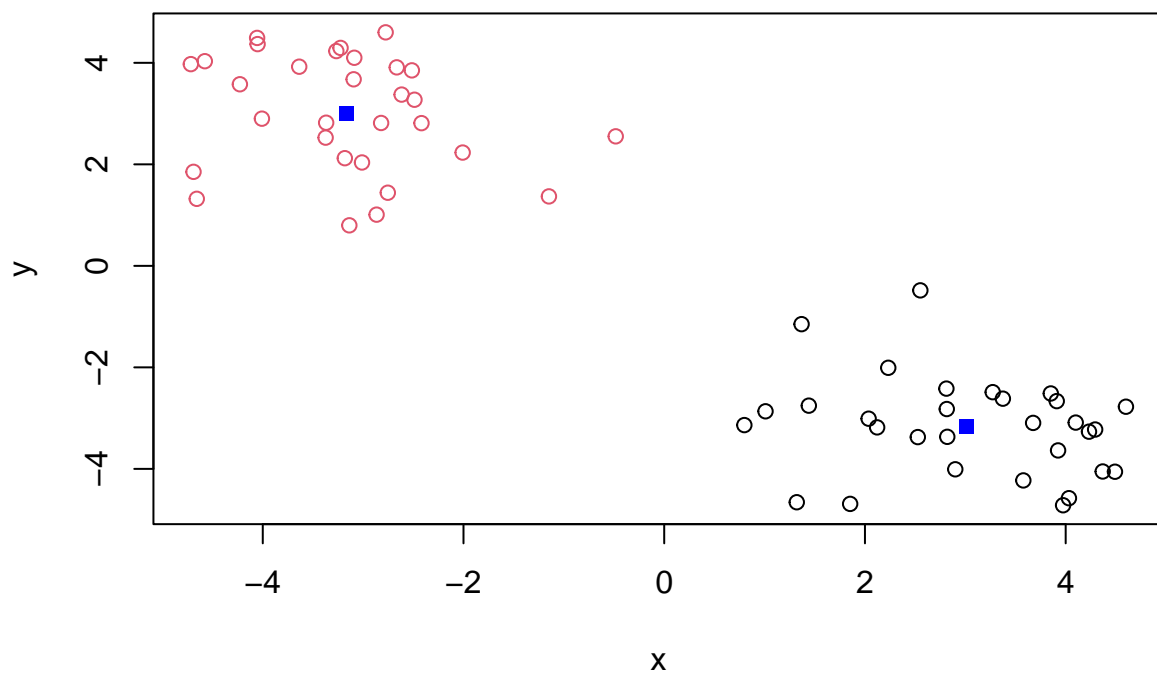
```
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2  
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

```
##           x           y
## 1  3.009502 -3.164292
## 2 -3.164292  3.009502
```

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)
```



Hierarchical Clustering, hclust()

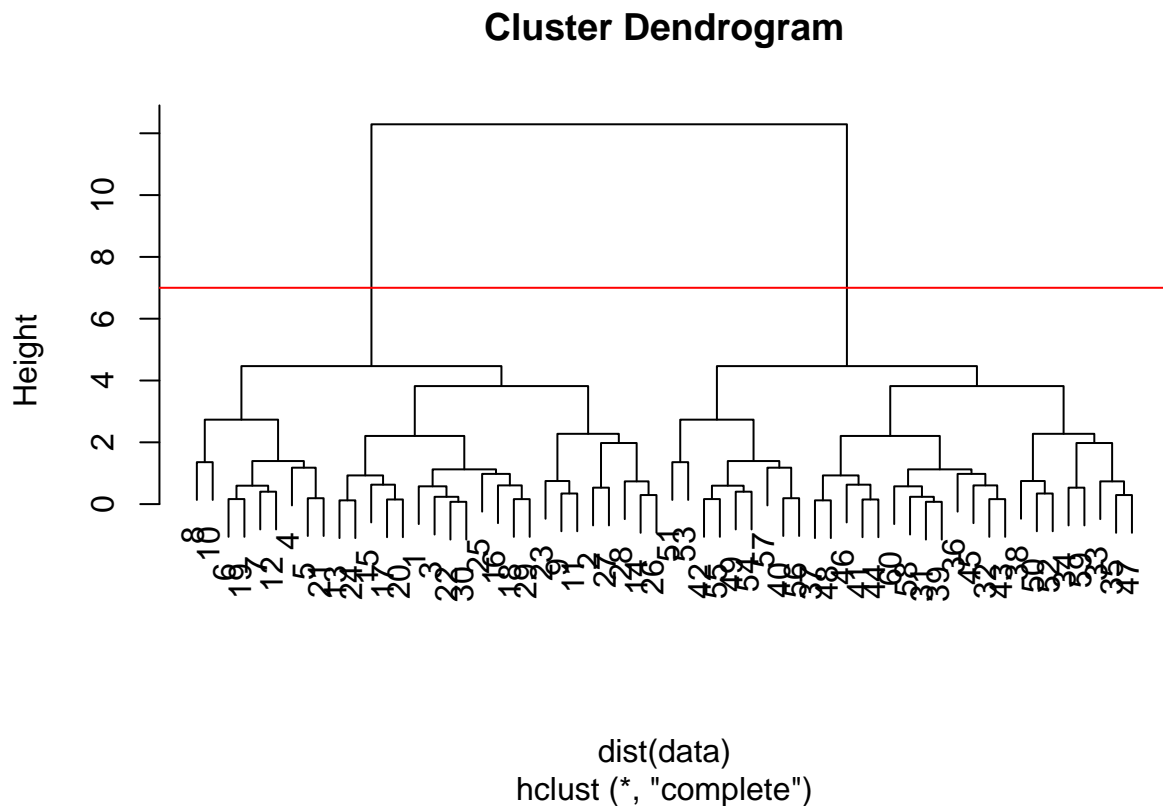
A little bit more work required than in kmeans: can't just use x, need a distance matrix using function `dist()`

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

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

This printout wasn't very helpful, use a plot. Hclust() has its own useful plot method

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



To find out membership vector, we need to “cut” the tree. Use `cutree()` function and tell it the height to cut at

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

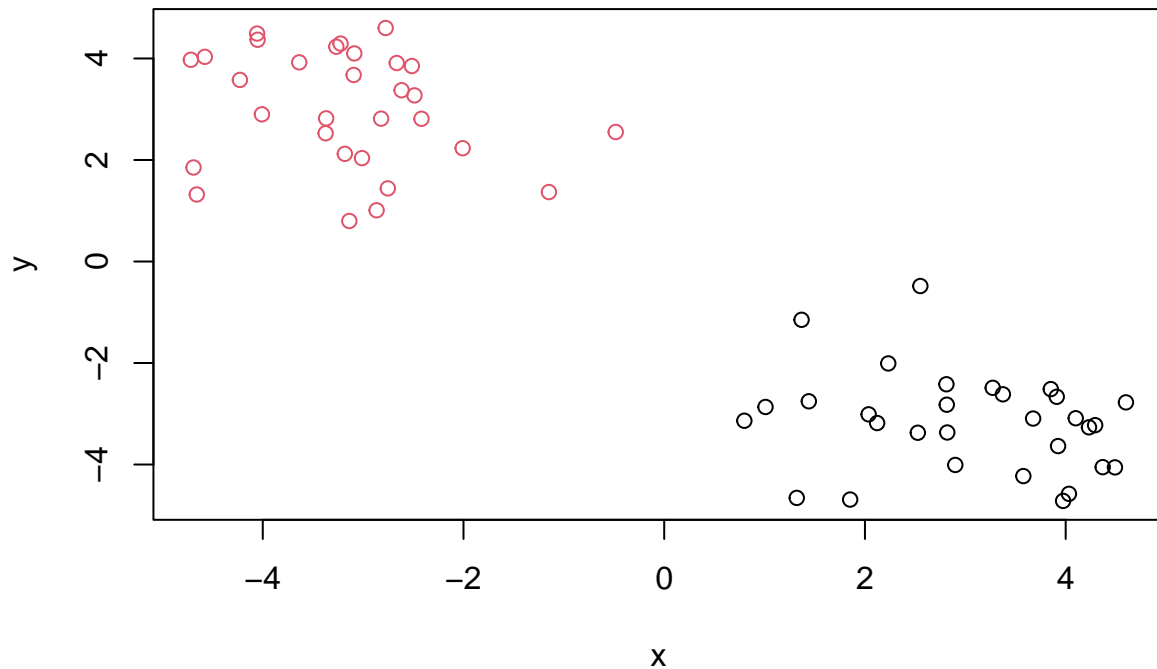
[illegible]

Can also use **cutree()** and state the number of k clusters we want

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

[illegible]

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



Principal Component Analysis (PCA)

A useful analysis method when you have lots of dimensions in your data

PCA of UK food data

Read in 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
```

Why aren't there only 4 columns (one for each country)?

```
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
```

Row names were being assigned to the first column, incorrectly

Fix data

```
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
```

The better way to fix this is upon import

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

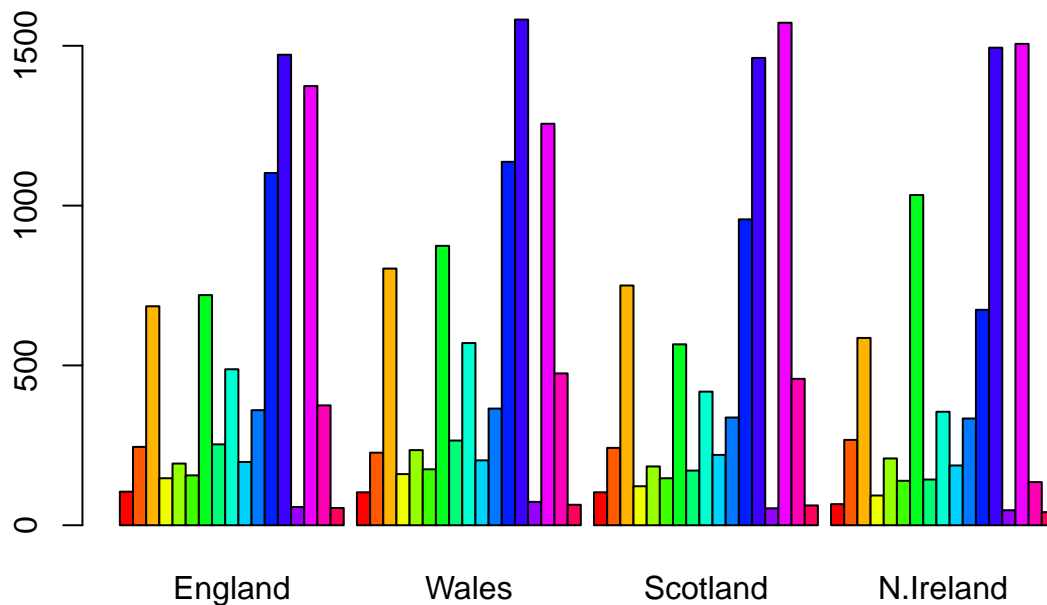
```
## [1] 17  4
```

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?

The second method is much better because it makes the correct change, permanently, to the dataset whereas the first method will keep taking away the first column.

Plotting the data

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

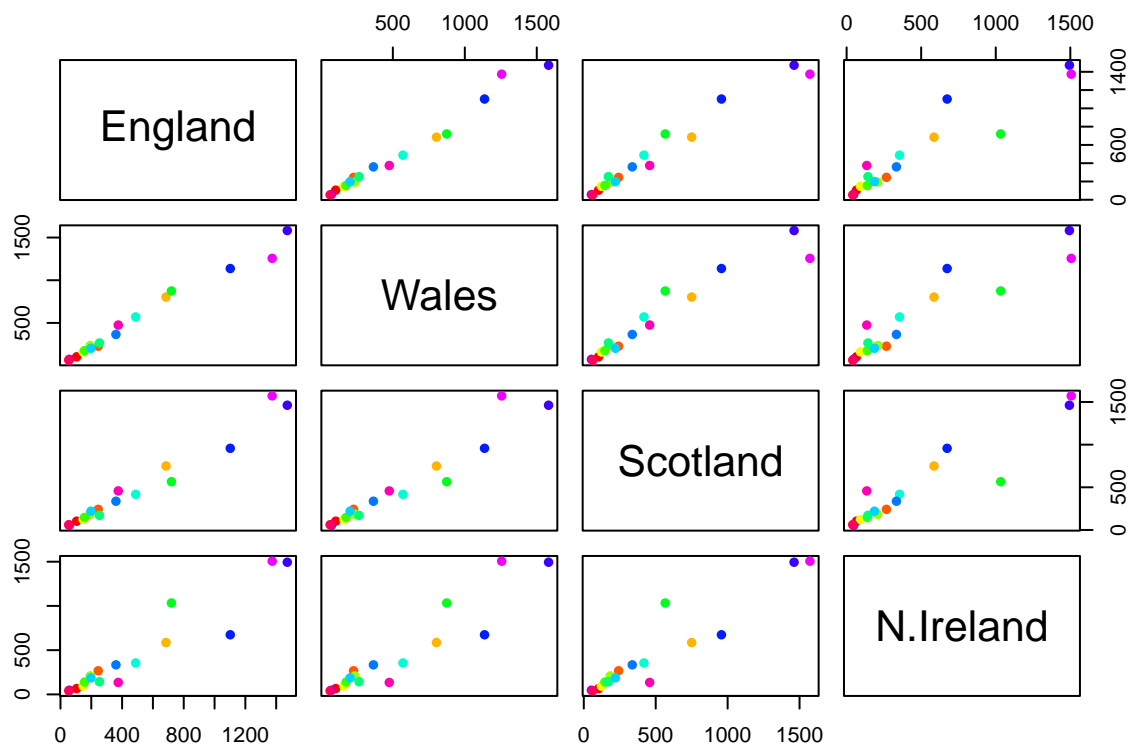


> Q3: Changing what optional argument in the above barplot() function results in the following plot?

The argument “beside” determines if values are stacked on top of each other or beside each other. Default is beside = F and values are stacked

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?

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

This plot compares all pairs of countries. Points on the diagonal show that the data is the same for the pair of countries. Points that are not on the diagonal signify differences between the two countries being compared.

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

The main difference between N. Ireland and the other countries is the green data point that was consumed at a higher quantity in N.Ireland than others. The results are still just qualitative though...

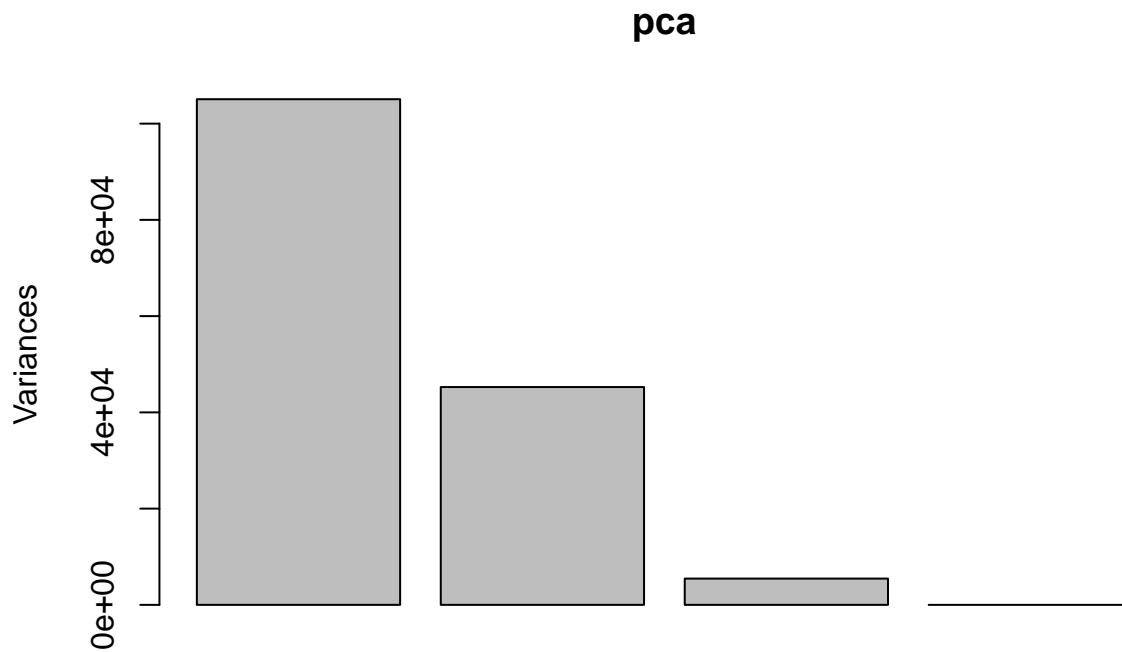
PCA is here to help

Base R function **prcomp()** to find PCA. First need to transpose the data, this function wants countries in rows and foods in columns.

```
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
```

```
plot(pca)
```



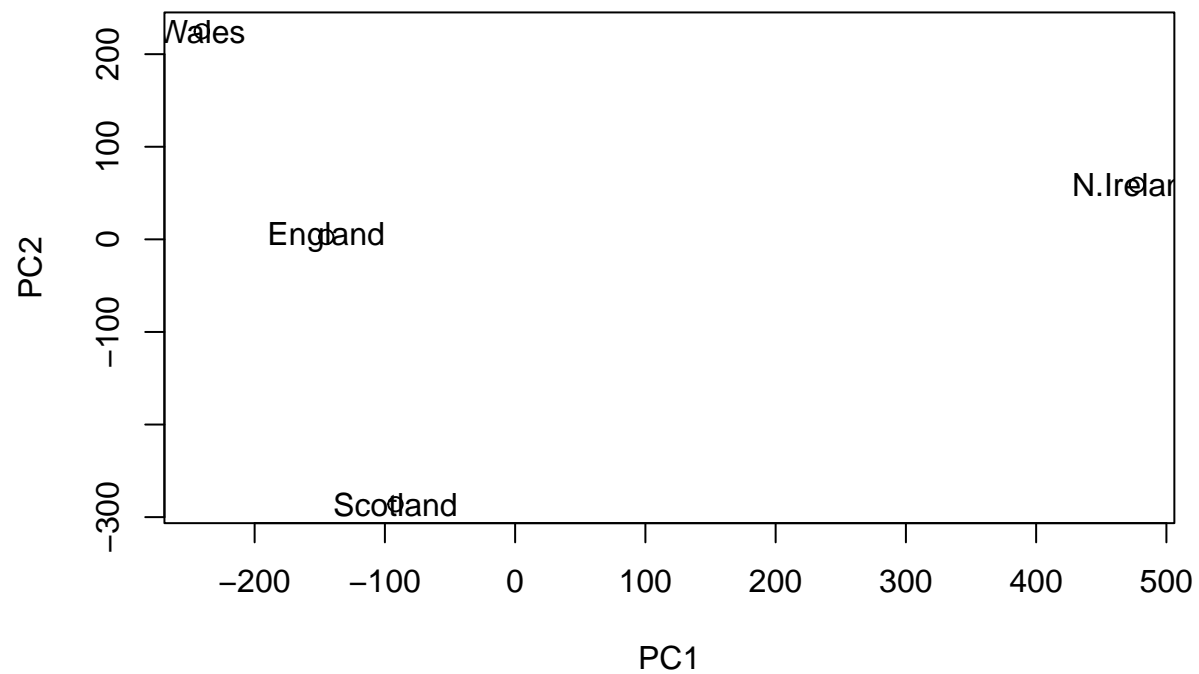
We want the **score plot** (aka PCA plot), the plot of PC1 vs. PC2 that we often see

```
attributes(pca)
```

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

We want `pca$x` component for this plot

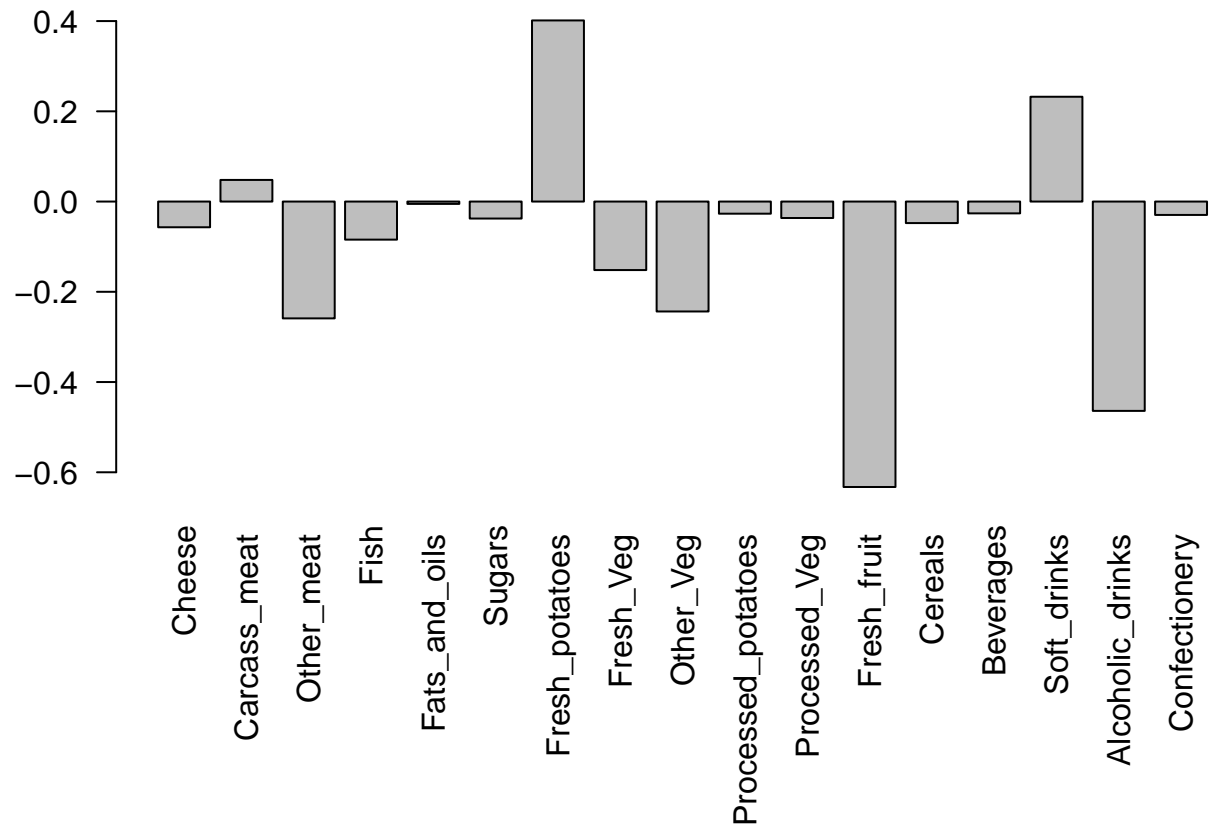
```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels = colnames(x))
```



PCA “loadings”

Loadings tell us how much each variable contributes to each PC

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



One more PCA for the day

```
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 the data set?

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

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

100 genes

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

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

```
colnames(rna.data)
```

```
## [1] "wt1" "wt2" "wt3" "wt4" "wt5" "ko1" "ko2" "ko3" "ko4" "ko5"
```

10 samples

PCA

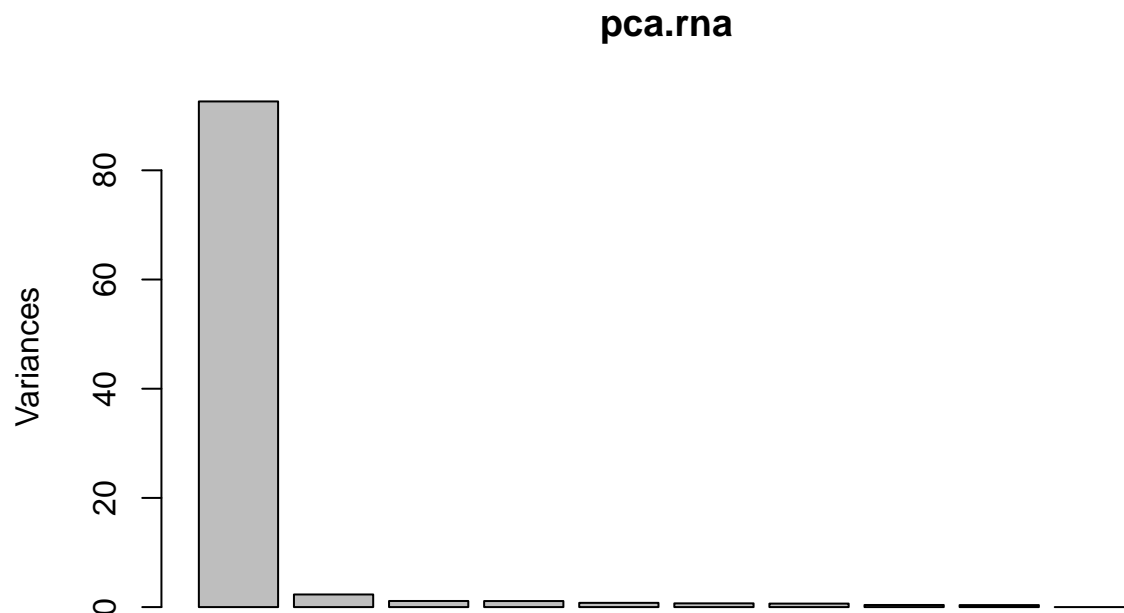
The argument `scale` normalizes the ranges in observations

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

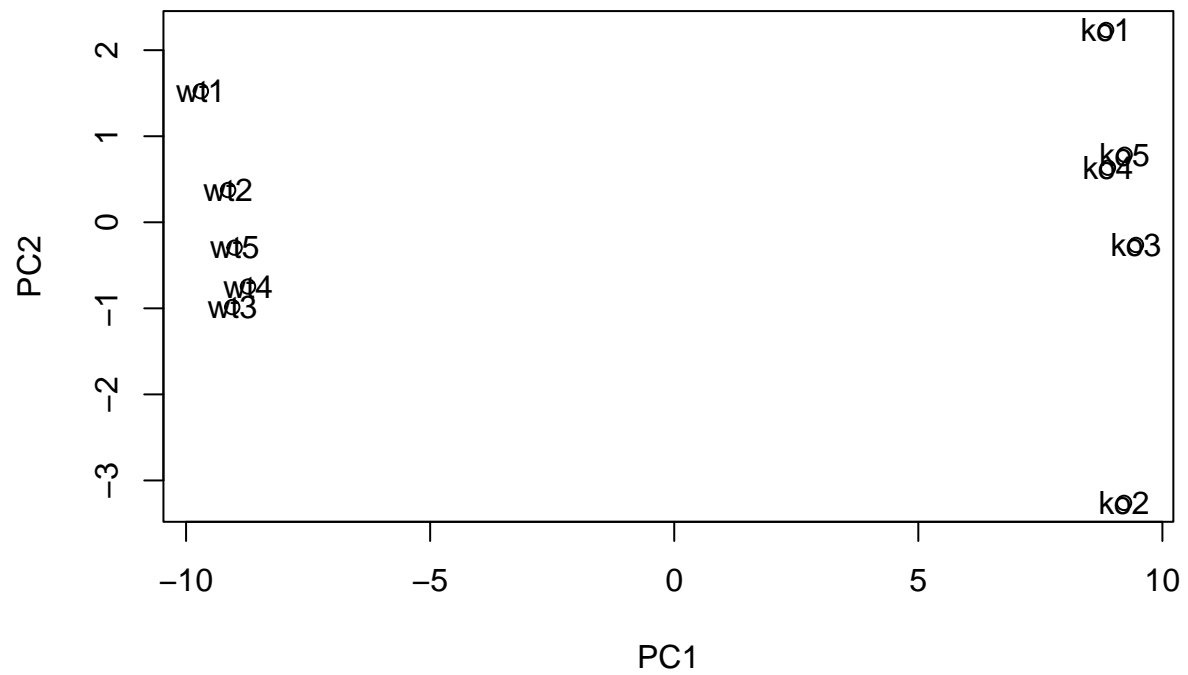
```
## Importance of components:
```

```
##              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
```

```
plot(pca.rna)
```



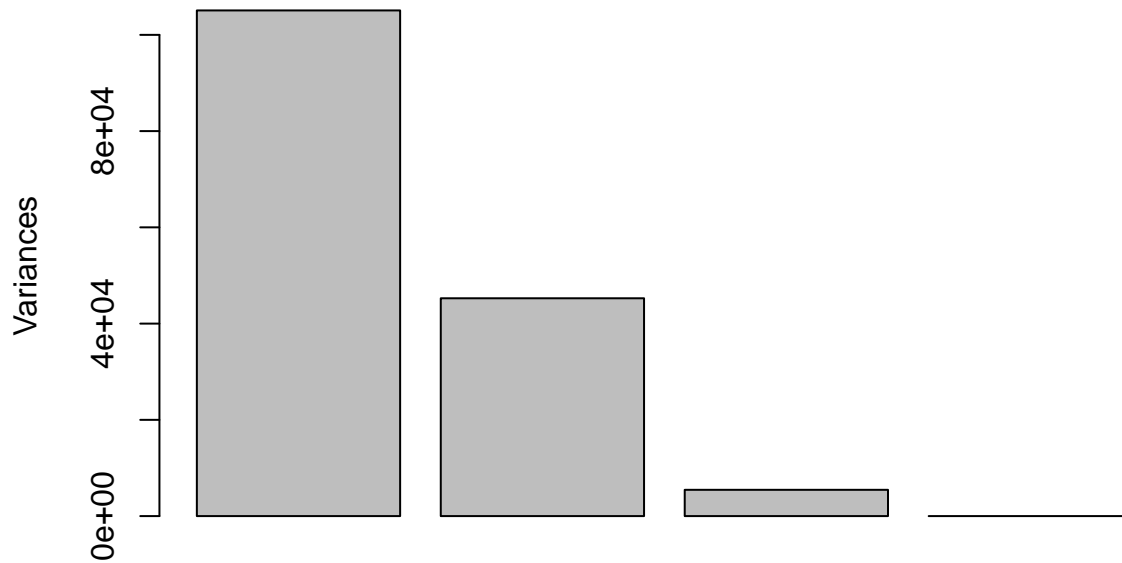
```
plot(pca.rna$x[,1:2])
text(pca.rna$x[,1:2], labels = colnames(rna.data))
```



Quick scree plot

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

Quick scree plot



our own scree plots

Variance captured per PC

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

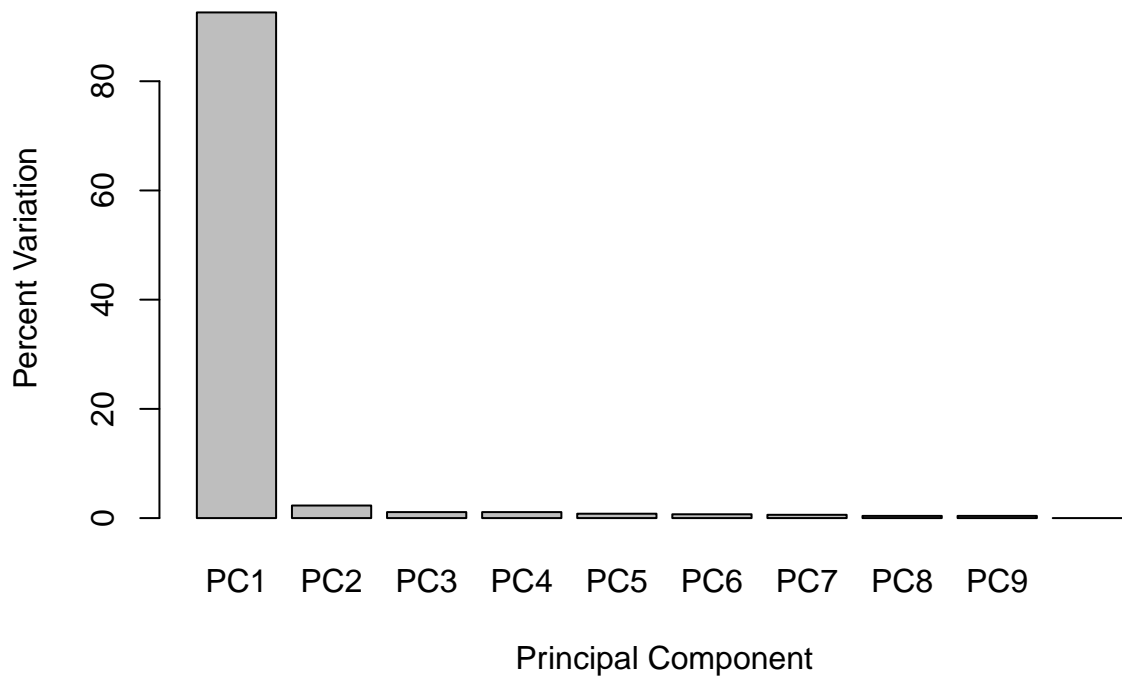
Percent variance is often more informative to look at

```
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
```

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

Scree Plot



Adding colors

```
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)))
```