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]  1.7497511  1.8082172  3.3273760  4.7986848  2.7836765  2.3894487
##  [7]  3.3663598  2.6700836  3.6999449  2.0594180  3.5175338  2.7780352
## [13]  2.1023812  3.6313188  2.5904721  1.6450529  3.8294294  2.1411001
## [19]  1.9452059  3.1755256  2.4640989  1.8798979  3.0080879  3.7096740
## [25]  1.8678499  2.8743725  3.5426674  2.3854772  2.2153416  2.3249671
## [31] -2.9265340 -2.1358771 -4.8591971 -3.0119060 -1.3121833 -3.3558764
## [37] -5.3322962 -2.8094544 -0.9467448 -1.7129933 -3.3559937 -1.9446448
## [43] -3.8791170 -3.0825804 -5.1509137 -1.8095680 -3.1033850 -3.5658274
## [49] -4.2431456 -2.5547385 -1.7783276 -2.1018411 -3.6365021 -2.9361731
## [55] -2.7886606 -4.6433911 -3.0466838 -4.7495074 -3.7453157 -3.3752116
```

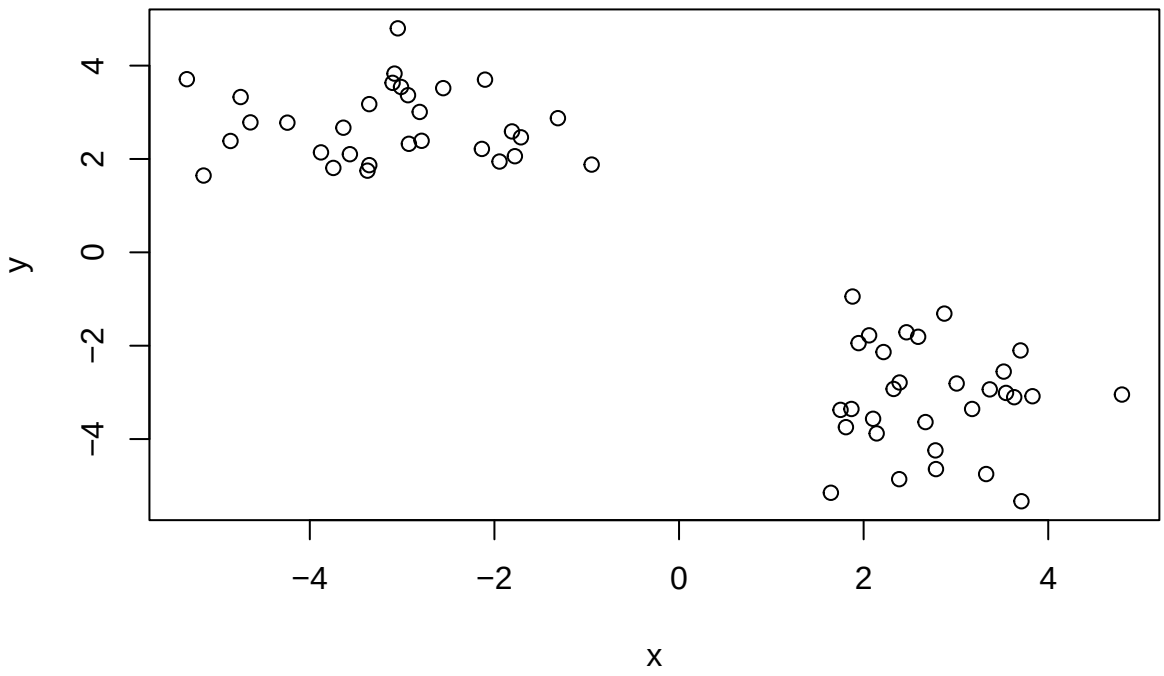
```
data <- cbind(x = tmp, y = rev(tmp))
data
```

```
##                x          y
##  [1,]  1.7497511 -3.3752116
##  [2,]  1.8082172 -3.7453157
##  [3,]  3.3273760 -4.7495074
##  [4,]  4.7986848 -3.0466838
##  [5,]  2.7836765 -4.6433911
##  [6,]  2.3894487 -2.7886606
##  [7,]  3.3663598 -2.9361731
##  [8,]  2.6700836 -3.6365021
##  [9,]  3.6999449 -2.1018411
## [10,]  2.0594180 -1.7783276
## [11,]  3.5175338 -2.5547385
## [12,]  2.7780352 -4.2431456
## [13,]  2.1023812 -3.5658274
## [14,]  3.6313188 -3.1033850
```

```
## [15,]  2.5904721 -1.8095680
## [16,]  1.6450529 -5.1509137
## [17,]  3.8294294 -3.0825804
## [18,]  2.1411001 -3.8791170
## [19,]  1.9452059 -1.9446448
## [20,]  3.1755256 -3.3559937
## [21,]  2.4640989 -1.7129933
## [22,]  1.8798979 -0.9467448
## [23,]  3.0080879 -2.8094544
## [24,]  3.7096740 -5.3322962
## [25,]  1.8678499 -3.3558764
## [26,]  2.8743725 -1.3121833
## [27,]  3.5426674 -3.0119060
## [28,]  2.3854772 -4.8591971
## [29,]  2.2153416 -2.1358771
## [30,]  2.3249671 -2.9265340
## [31,] -2.9265340  2.3249671
## [32,] -2.1358771  2.2153416
## [33,] -4.8591971  2.3854772
## [34,] -3.0119060  3.5426674
## [35,] -1.3121833  2.8743725
## [36,] -3.3558764  1.8678499
## [37,] -5.3322962  3.7096740
## [38,] -2.8094544  3.0080879
## [39,] -0.9467448  1.8798979
## [40,] -1.7129933  2.4640989
## [41,] -3.3559937  3.1755256
## [42,] -1.9446448  1.9452059
## [43,] -3.8791170  2.1411001
## [44,] -3.0825804  3.8294294
## [45,] -5.1509137  1.6450529
## [46,] -1.8095680  2.5904721
## [47,] -3.1033850  3.6313188
## [48,] -3.5658274  2.1023812
## [49,] -4.2431456  2.7780352
## [50,] -2.5547385  3.5175338
## [51,] -1.7783276  2.0594180
## [52,] -2.1018411  3.6999449
## [53,] -3.6365021  2.6700836
## [54,] -2.9361731  3.3663598
## [55,] -2.7886606  2.3894487
## [56,] -4.6433911  2.7836765
## [57,] -3.0466838  4.7986848
## [58,] -4.7495074  3.3273760
## [59,] -3.7453157  1.8082172
## [60,] -3.3752116  1.7497511
```

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  2.742715 -3.129820
## 2 -3.129820  2.742715
##
## 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] 54.93596 54.93596
##  (between_SS / total_SS =  90.4 %)
##
## 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
```

Thirty points in each cluster

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 2 2
```

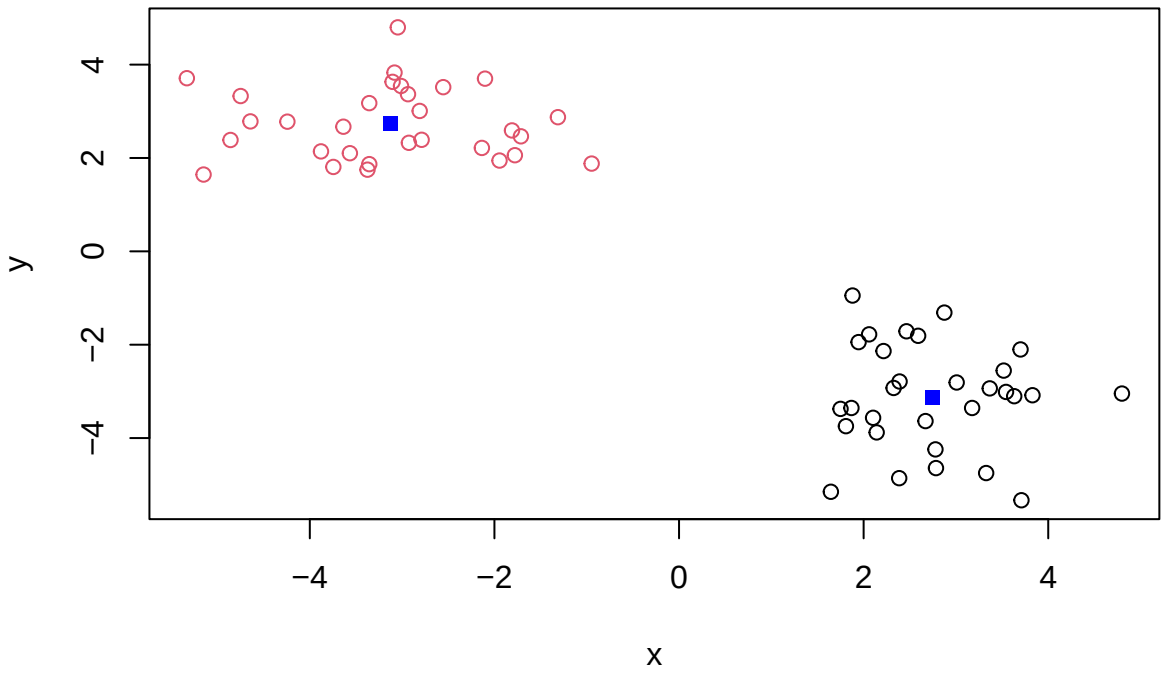
Q. What "component" of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1  2.742715 -3.129820
## 2 -3.129820  2.742715
```

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()

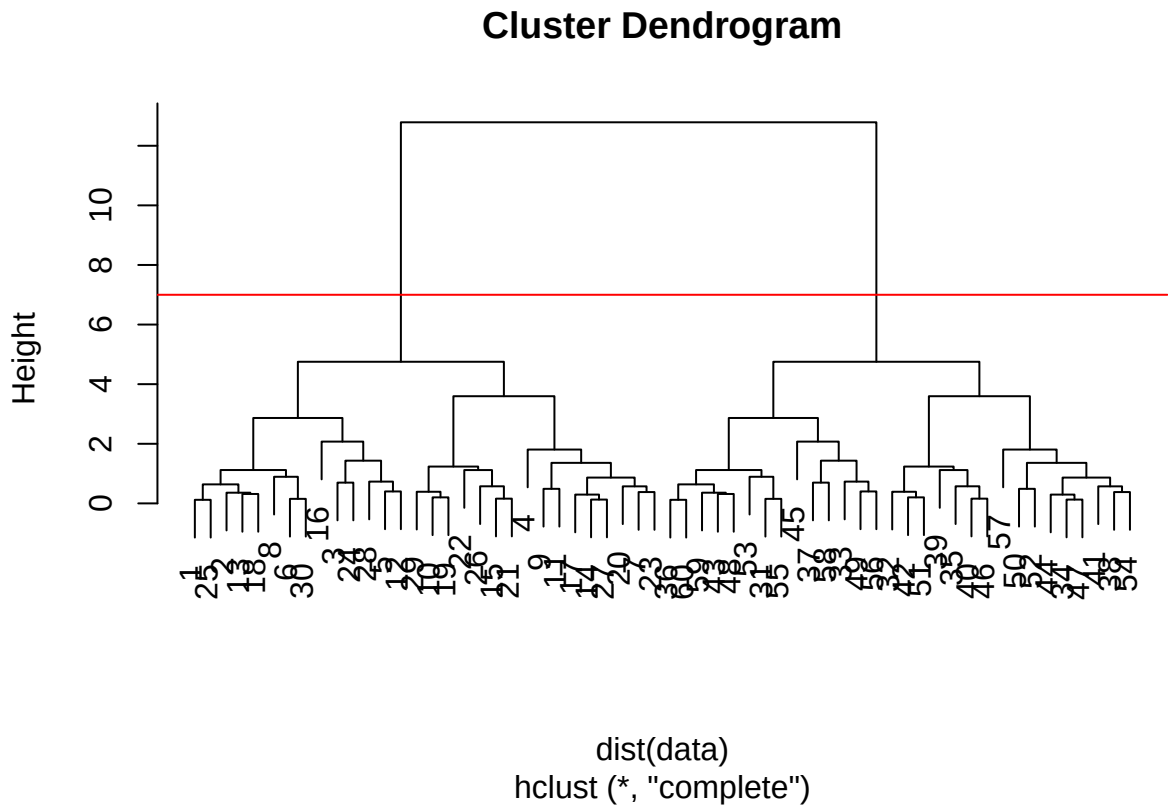
A little bit more work required than in kmeans: can't just use x, need a distance matrix using function **dist()**

```
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)
```

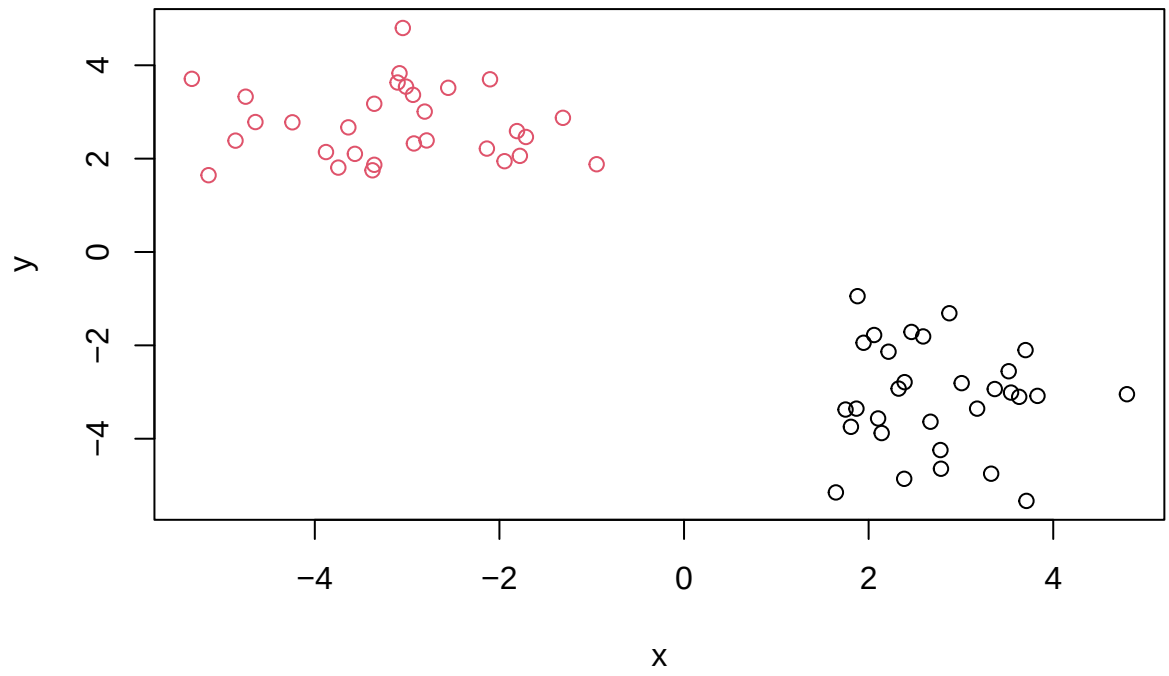
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

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

```
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)
```

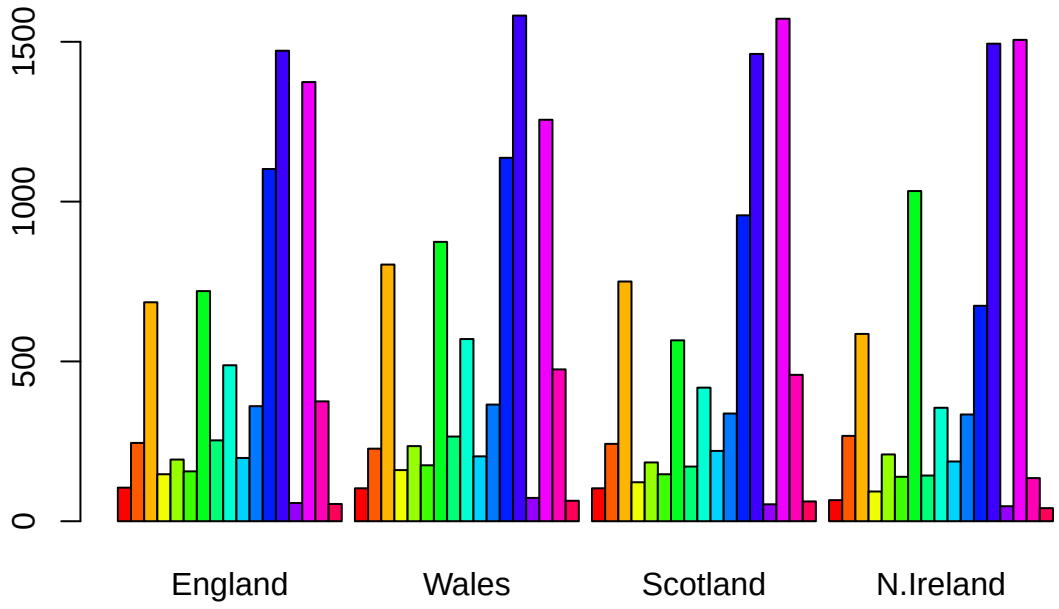
```
## [1] 17  4
```

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)))
```

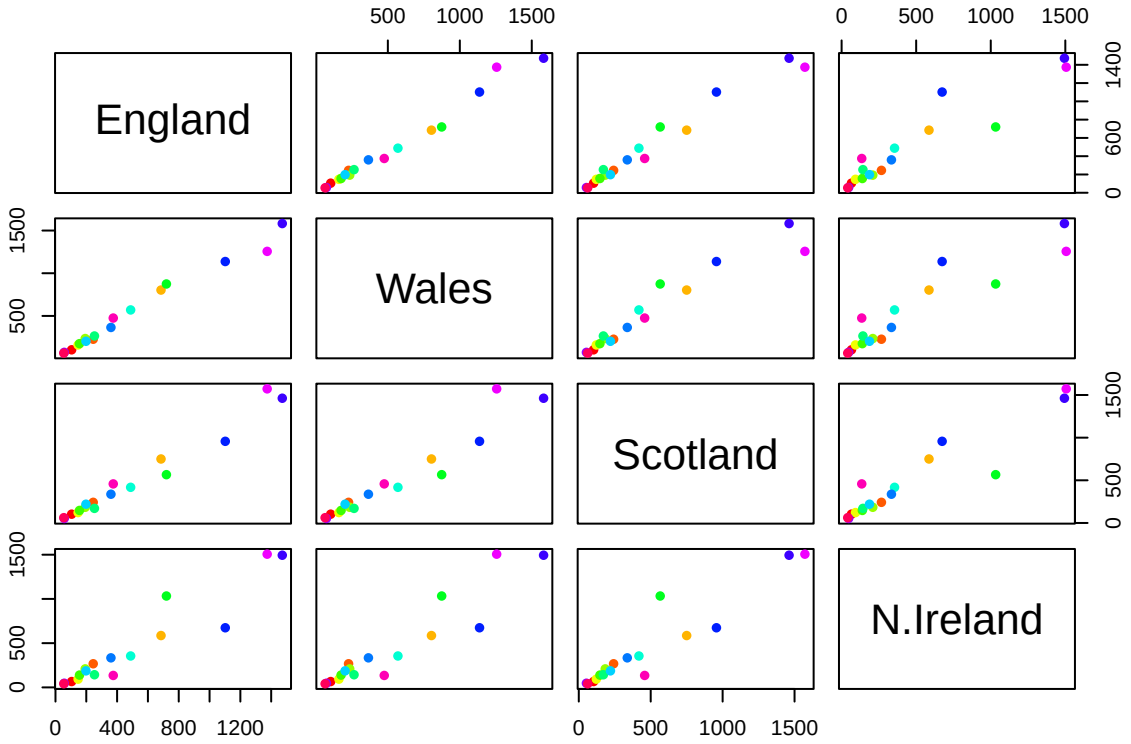


> Q3: Changing what optional argument in the above barplot() function results in the following plot?

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 tha 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 differences 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...

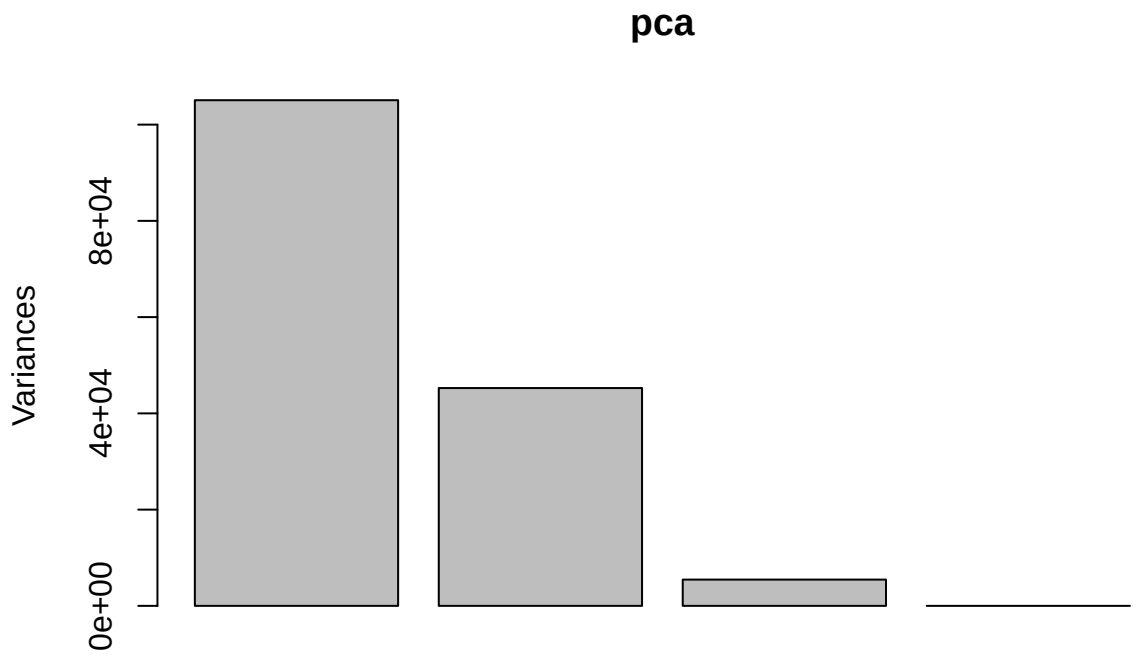
PCA is here to help

Base R function **prcomp()** to find PCA First need to tranpose 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)
```



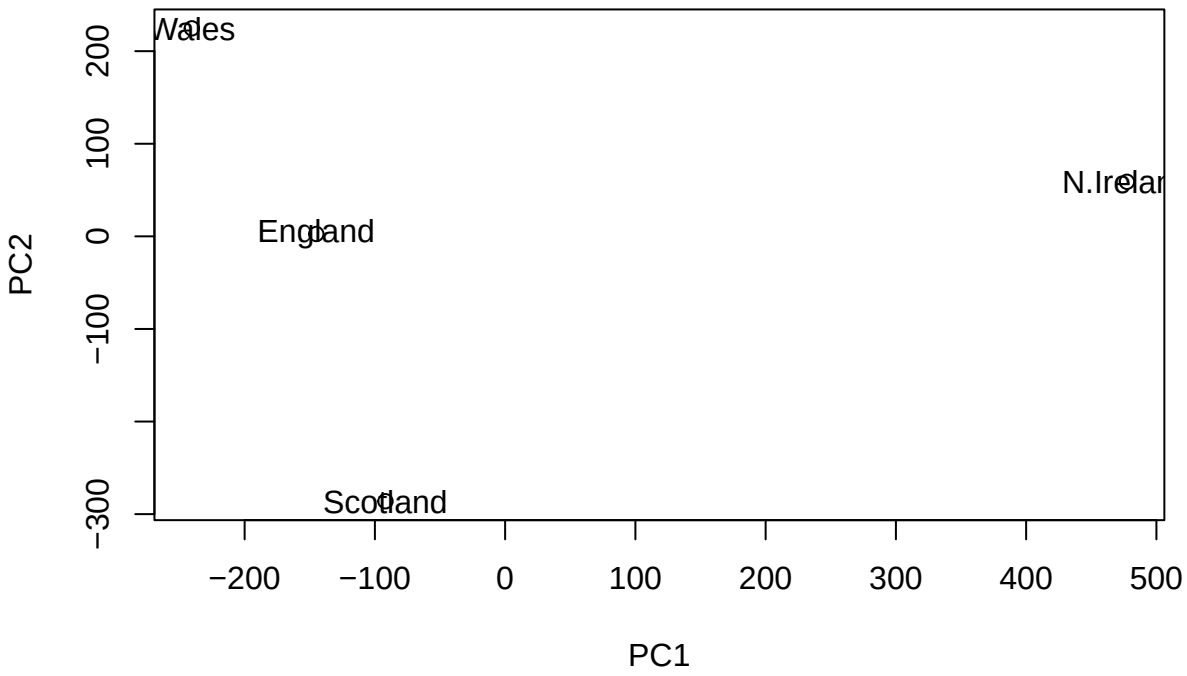
We want the **score plot** (aka PCA plot), the plot of PC1 vs. PC2 that we often see

```
attributes(pca)
```

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

We want pca$x component for this plot

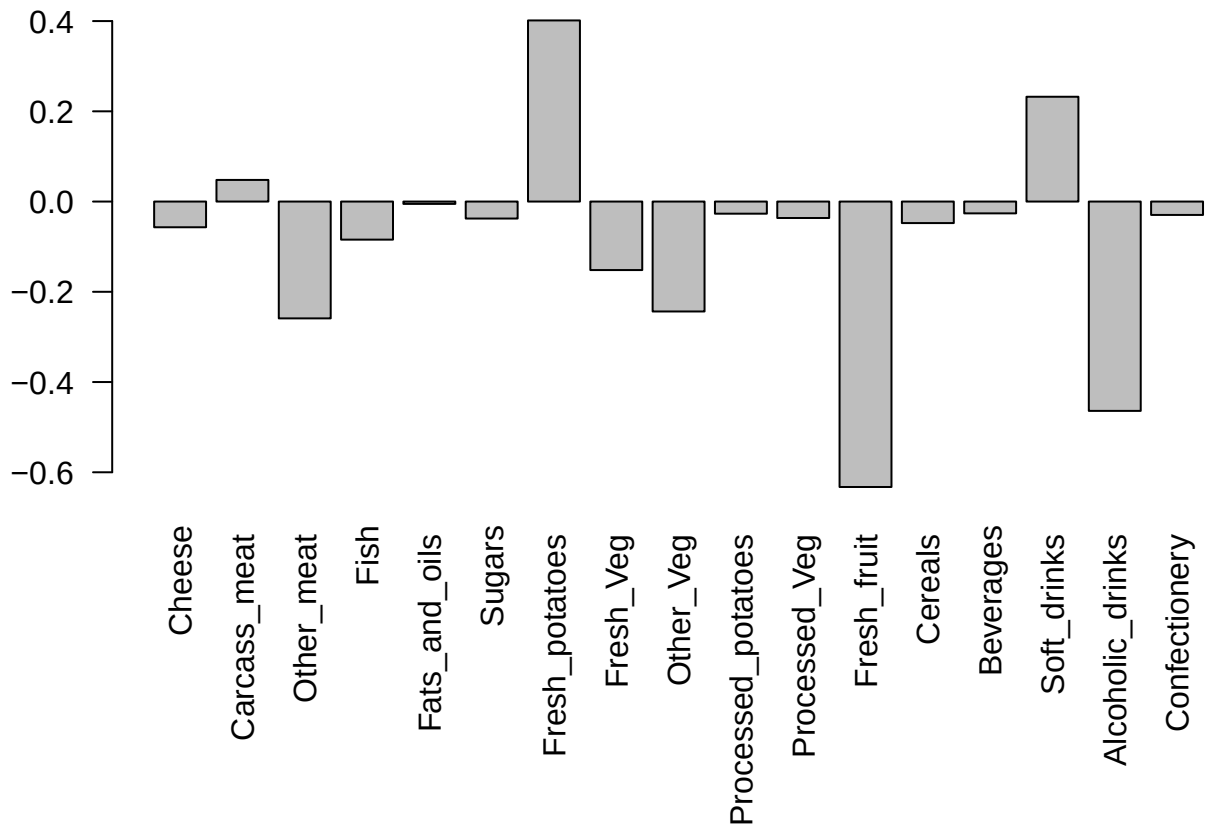
```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels = colnames(x))
```



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

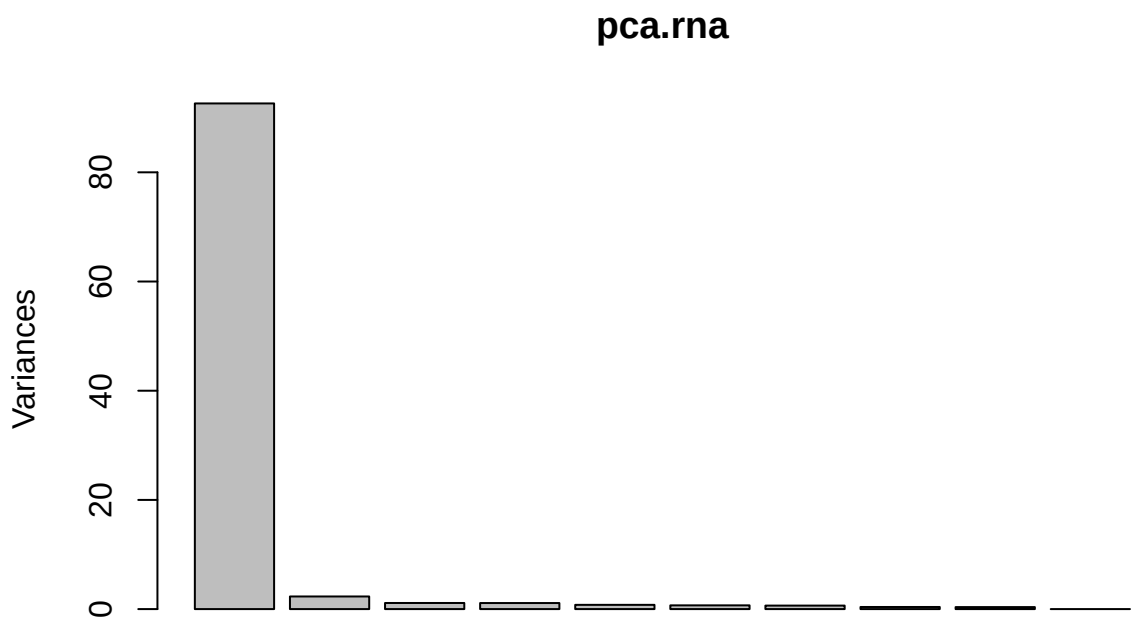
PCA

The argument scale normalizes the ranges in observations

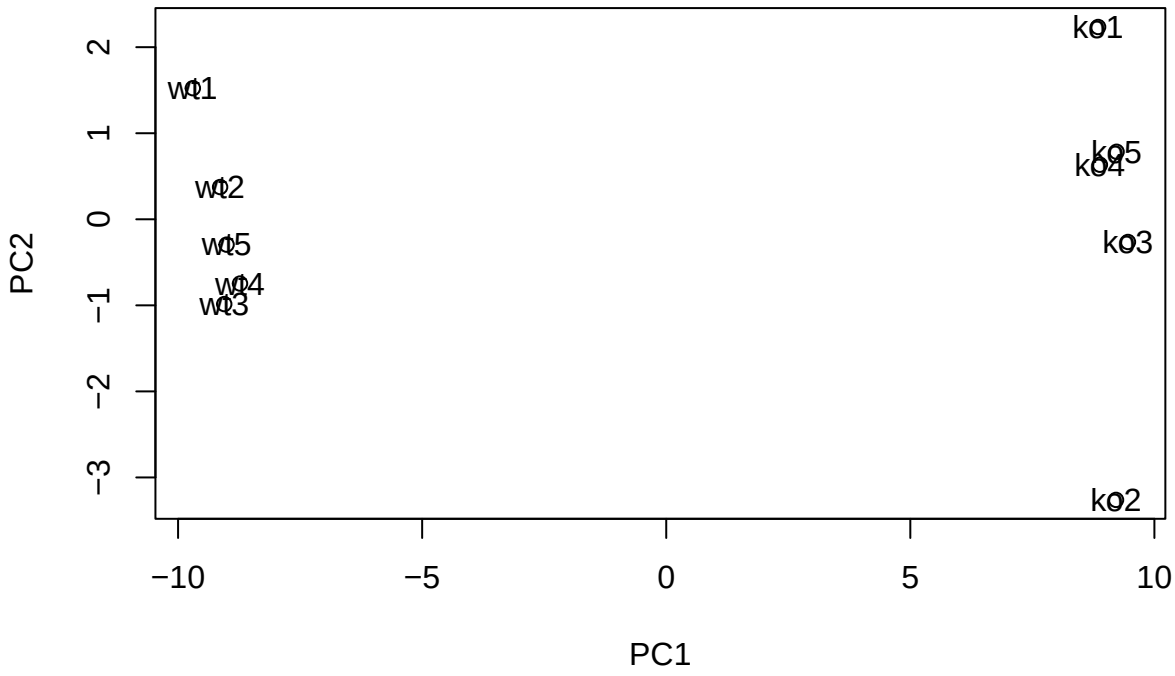
```
pca.rna <- prcomp(t(rna.data), scale = T)
summary(pca.rna)
```

```
## 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)
```

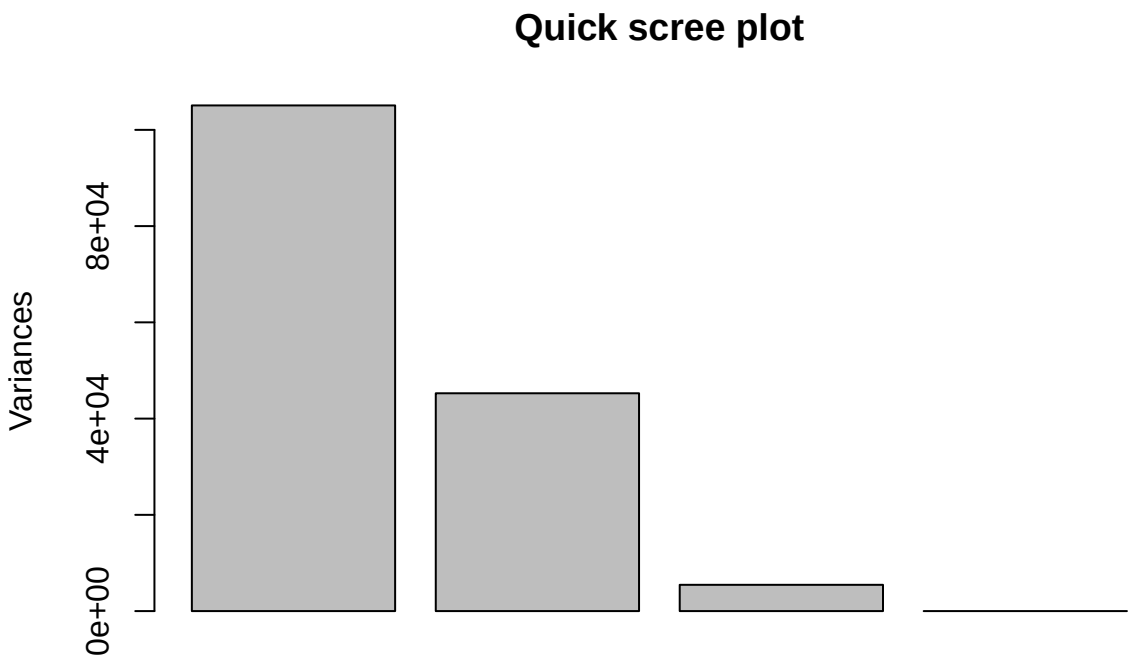


```
plot(pca.rna$x[,1:2])
text(pca.rna$x[,1:2], labels = colnames(rna.data))
```



Quick scree plot

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



our own scree plots

Variance captured per PC

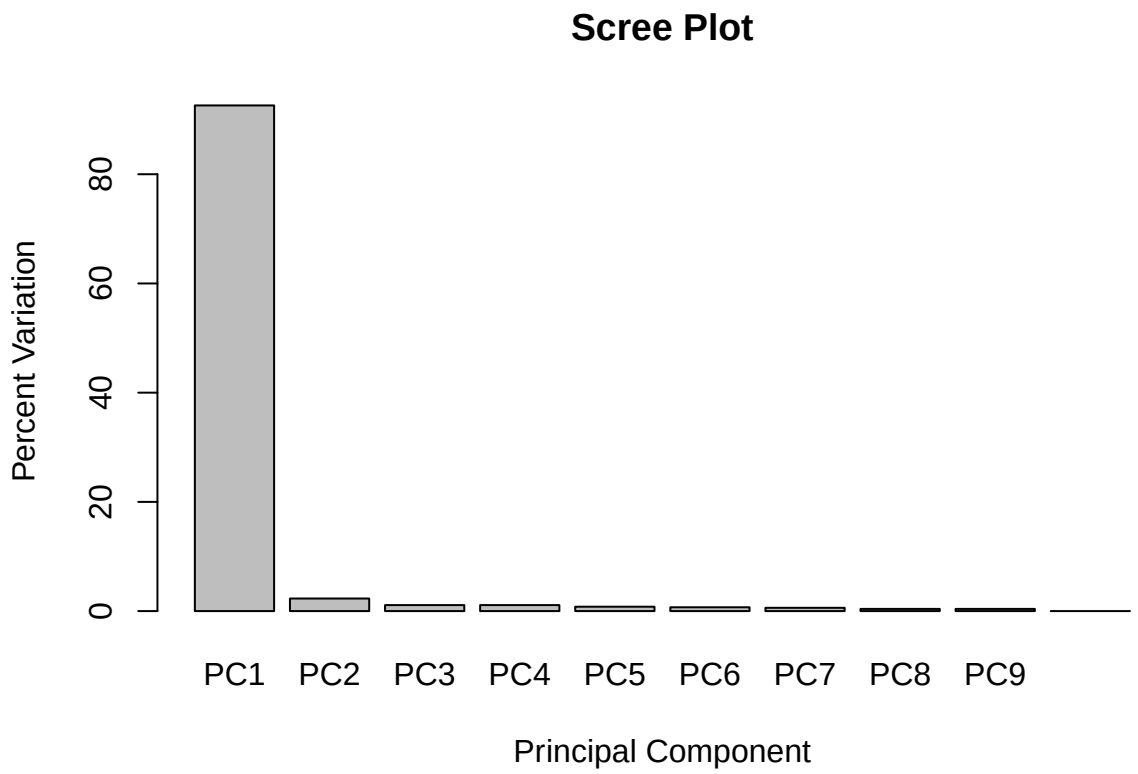
```
pca.var <- pca.rna$sdev^2
```

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")
```

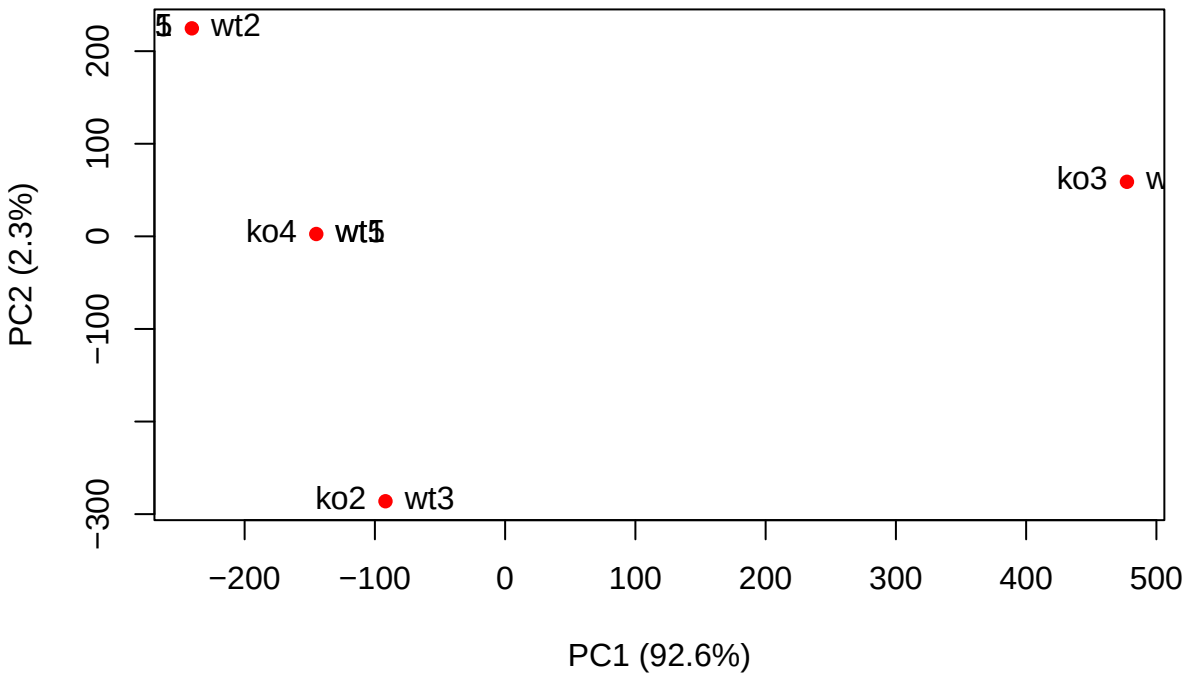


Adding colors

```
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)))
```


wt2
ko3
ko4
wt1
ko2
wt3
PC2 (2.3%)
PC1 (92.6%)
200
100
0
−100
−300
−200
−100
0
100
200
300
400
500