Machine Learning 1

Ethan Harding (PID A15468670)

10/21/2021

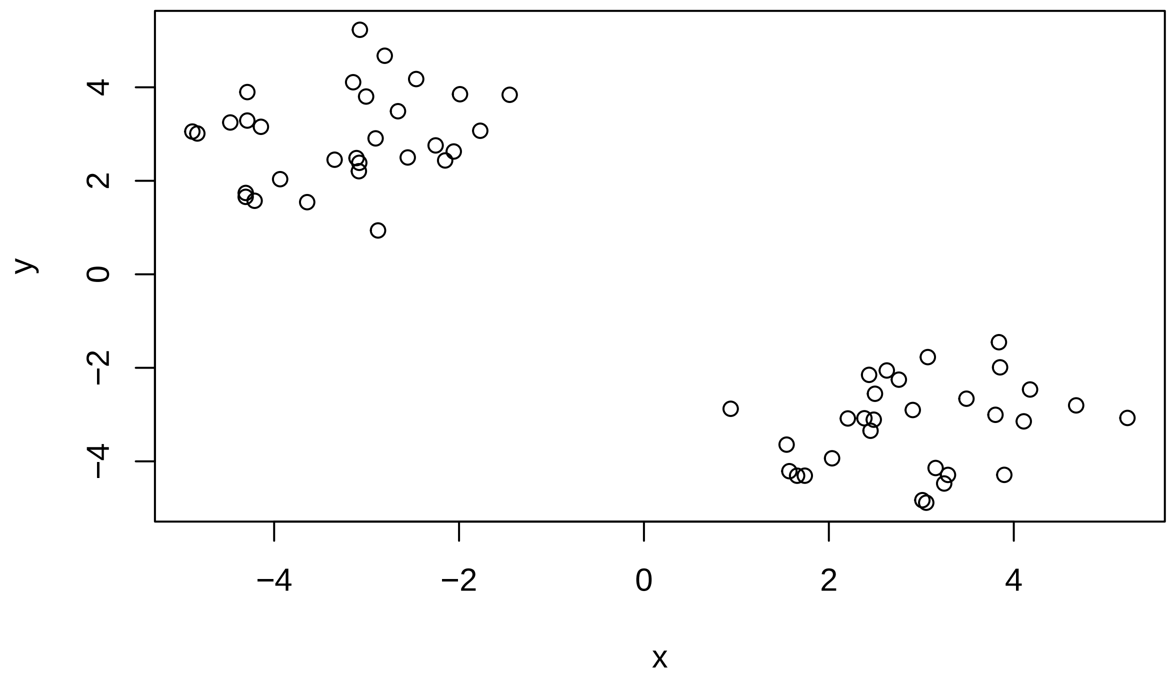
First up is clustering methods

K means clustering

The function ins base R to do Kmeans clustering is called 'kmeans()'.

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

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



Q. Can we use kmeans() to cluster this data setting k to 2 and nstart to 20?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.236323  2.937535
## 2  2.937535 -3.236323
##
## 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 2 2
##
## Within cluster sum of squares by cluster:
## [1] 54.79335 54.79335
##  (between_SS / total_SS =  91.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```r
km$size
```

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

There are 30 points in each cluster.

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

```r
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 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
```

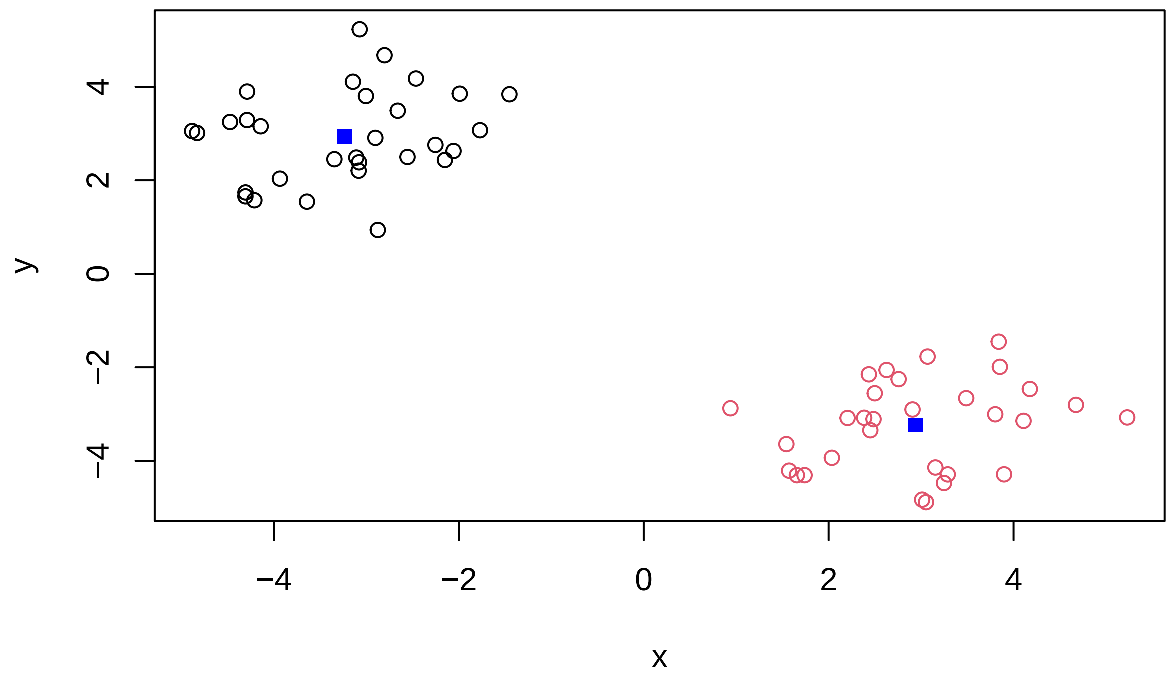
Q. What 'component' of your result object details cluster center?

```r
km$centers
```

```
##           x         y
## 1 -3.236323  2.937535
## 2  2.937535 -3.236323
```

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

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



hclust

A big limitation with k-means is that we have to tell it K (the number of clusters we want)

Analyze this same data with hclust()

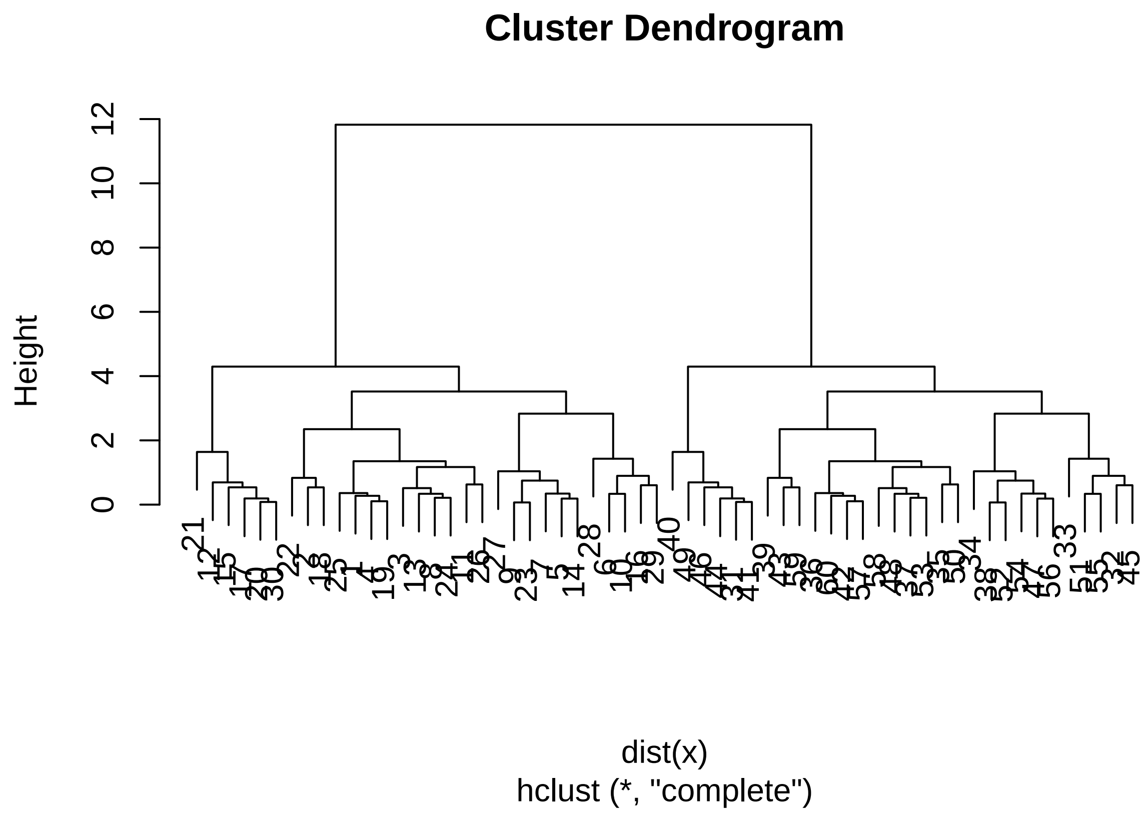
Demonstrate the use of dist(), hclust(), plot(), and cutree() functions to do clustering, generate dendrograms, and return cluster assignment / membership vectors.

```r
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 object. Let’s see it.

```
plot(hc)
```



To get out cluster membership vector, we have to do a wee bit more work. We have to "cut" the tree (dendrogram), where we think it makes sense. For this we use the 'cutree()' function.

```
cutree(hc, h=6)
```

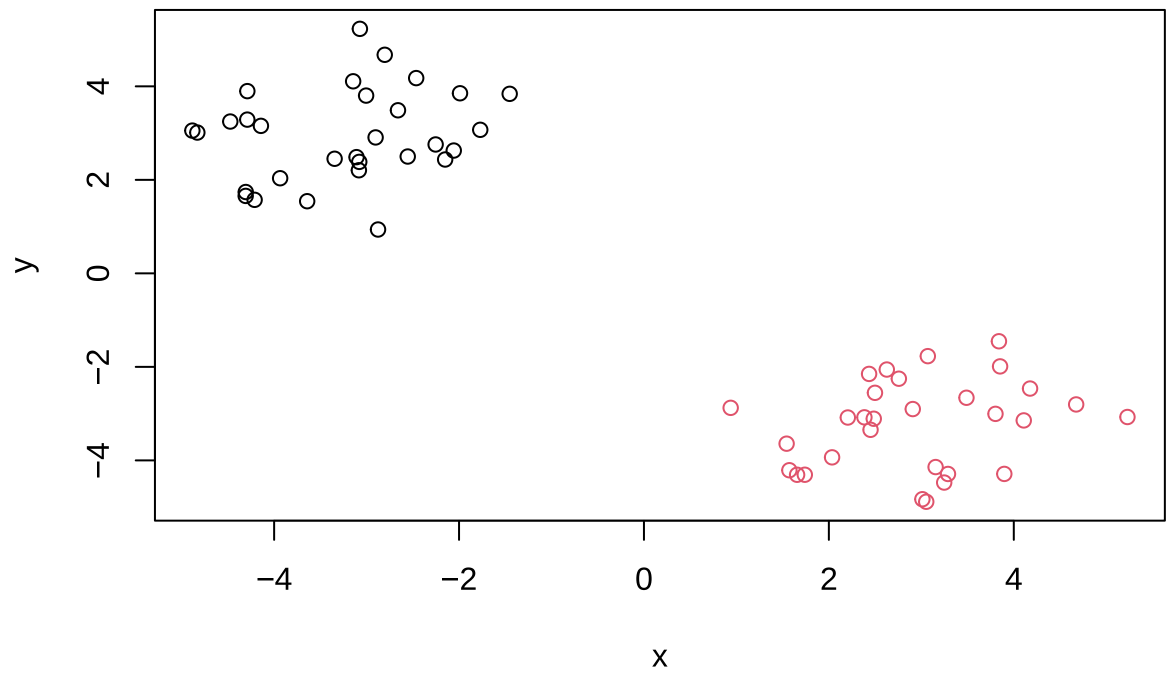
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also call 'cutree()', setting k=the number of groups / clusters you want.

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

Make our results plot

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



Principal Compnent Analysis

```
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 question?

We can use ‘dim()’ to find the dimensions of our object

```
dim(x)
```

```
## [1] 17  5
```

```
View(x)
```

Without any changes, we see there are 17 rows and 5 columns. We want it to display that there are 4 columns, as the first column is a header.

```
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
```

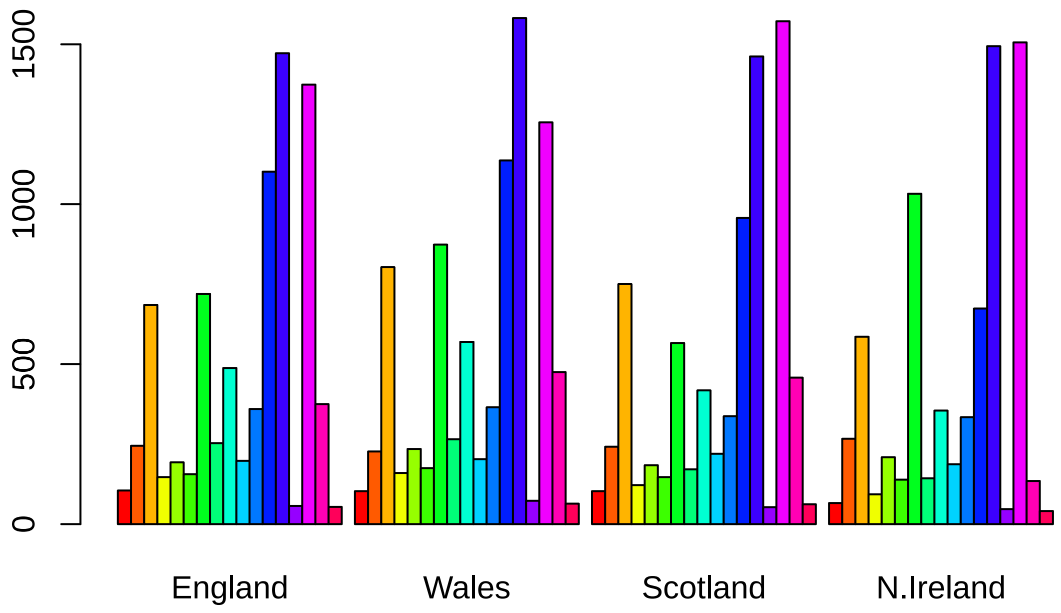
```
dim(x)
```

```
## [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?

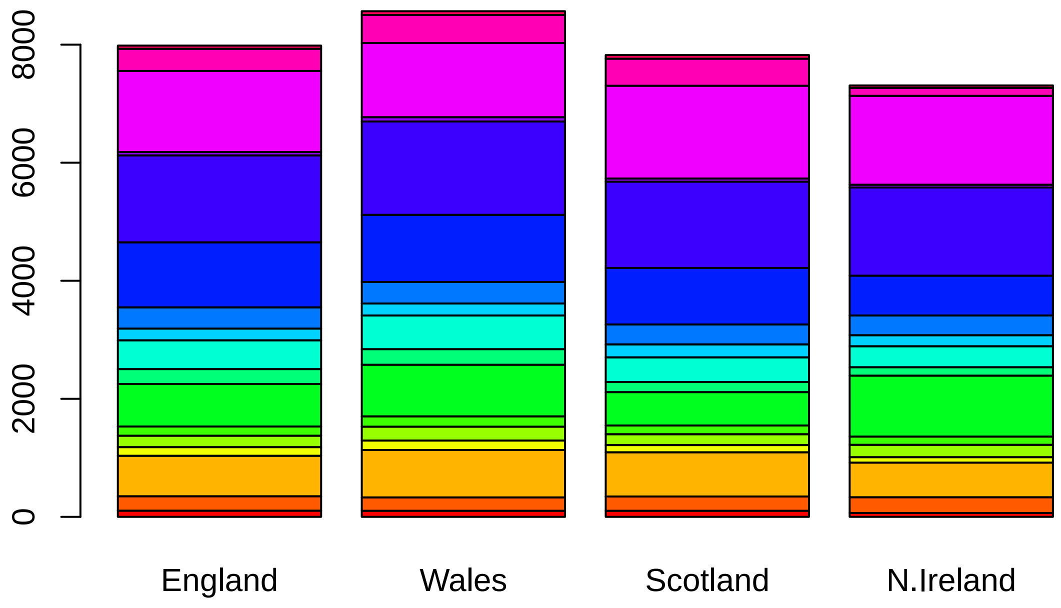
I prefer adding the "row.names" argument to the read function, as it does not delete any rows or columns everytime we run the code, but instead just lets R know that "n" number of row.names are the headers essentially.

```
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?

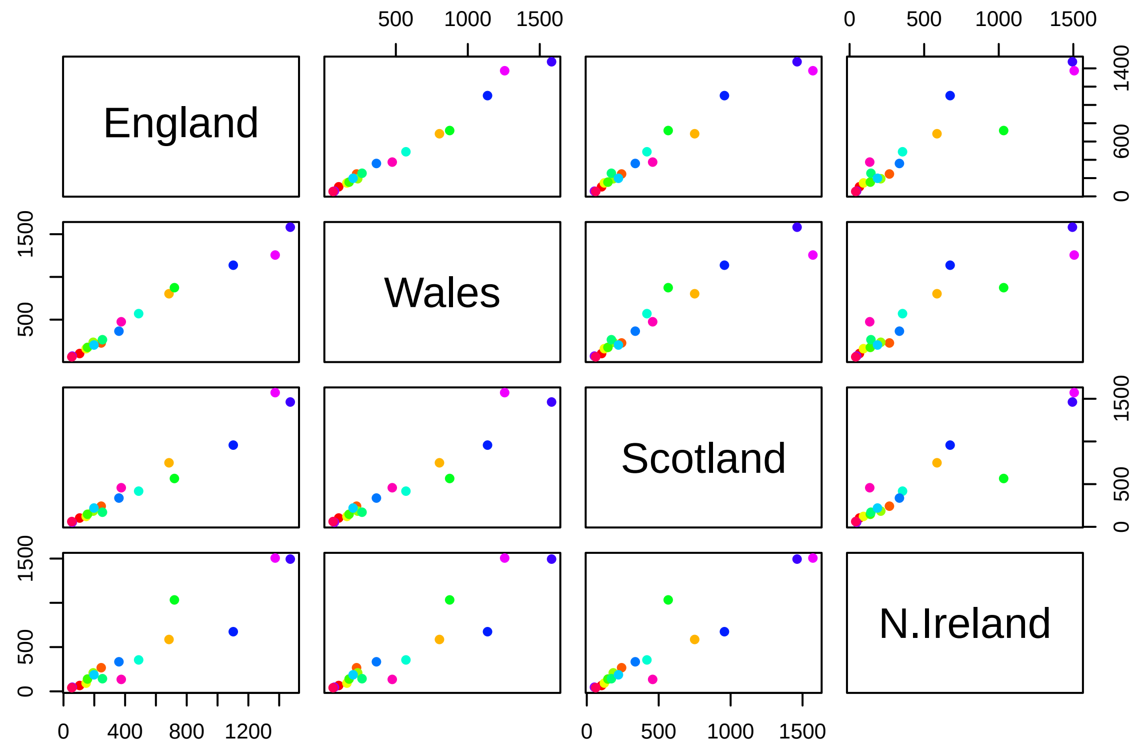
```r
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



By changing the 'beside' argument in the barplot function, we can choose if we want to stack the bar plots or not. 'beside' is automatically set to False, which stacks the bars, if we change 'beside' to True, then the bars are not stacked and instead are displayed "beside" one another.

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?

```r
pairs(x, col=rainbow(17), pch=16)
```



Yes, the function generates a plot among "pairs", in this instance the different countries would be a pair, and then If a given point lies on a diagonal for a given plot, it means that the data is the same, there is no variation within the data for that dataset.

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

We can find the main differences between N. Ireland and the other countries of the UK by looking at which data points in the pairs plots from N. Ireland are not along the diagonal.

PCA to the rescue!

The main function in base R for PCA is 'prcomp()'. However, 'prcomp()' expects the observations to be rows and the variables to be columns, thus we need to switch the two by using the transpose 't()' function.

```
# Use the prcomp() PCA function
pca <- prcomp( t(x) )
summary(pca)
```

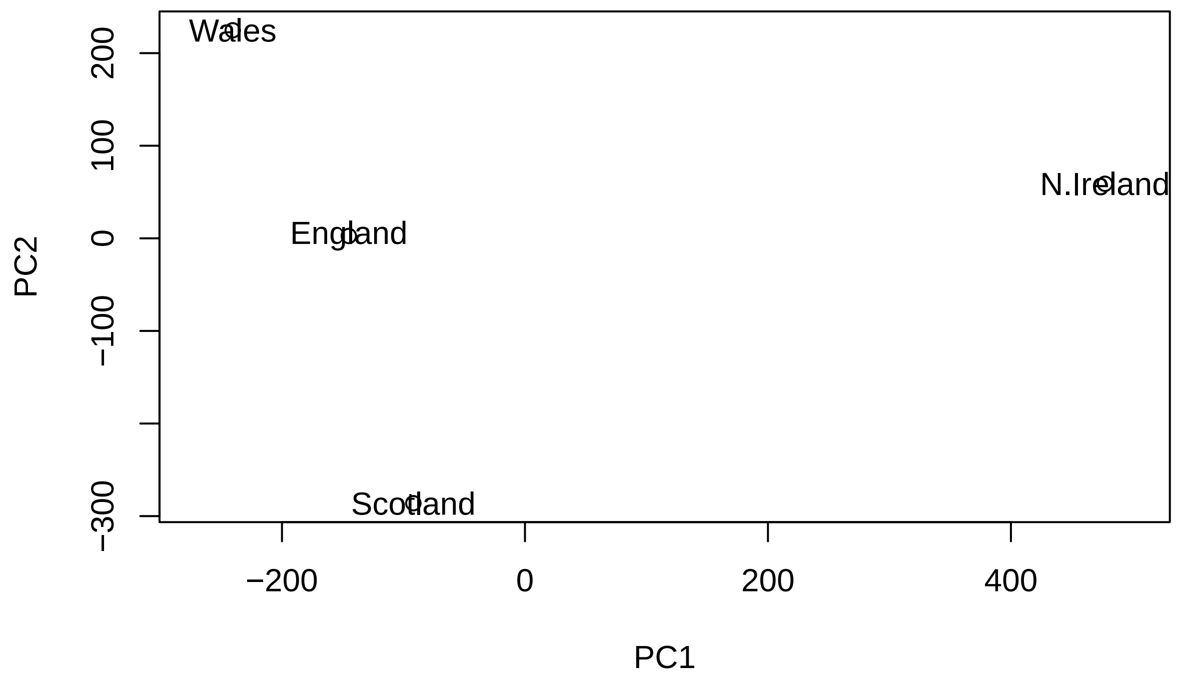
```
## Importance of components:
##                             PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
attributes(pca)
```

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

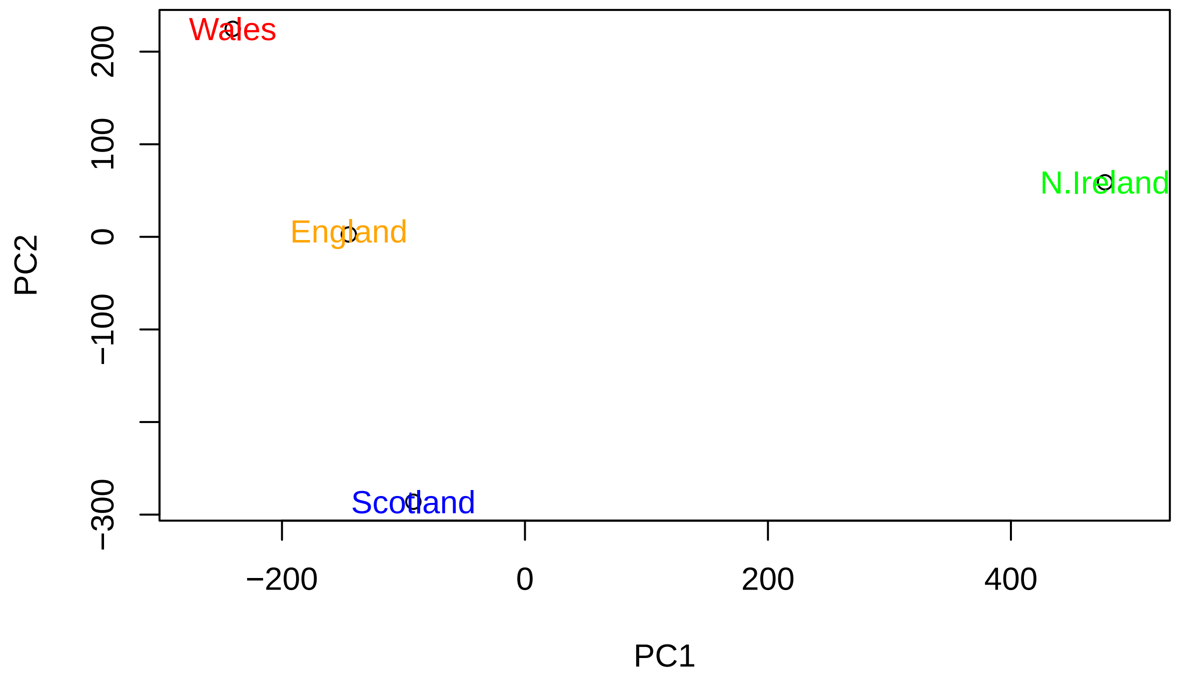
> Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
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("orange", "red", "blue", "green"))
```



We can use the square of pca$sdev, which stands for “standard deviation” to calculate how much deviation is in the original data each PC accounts for.

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

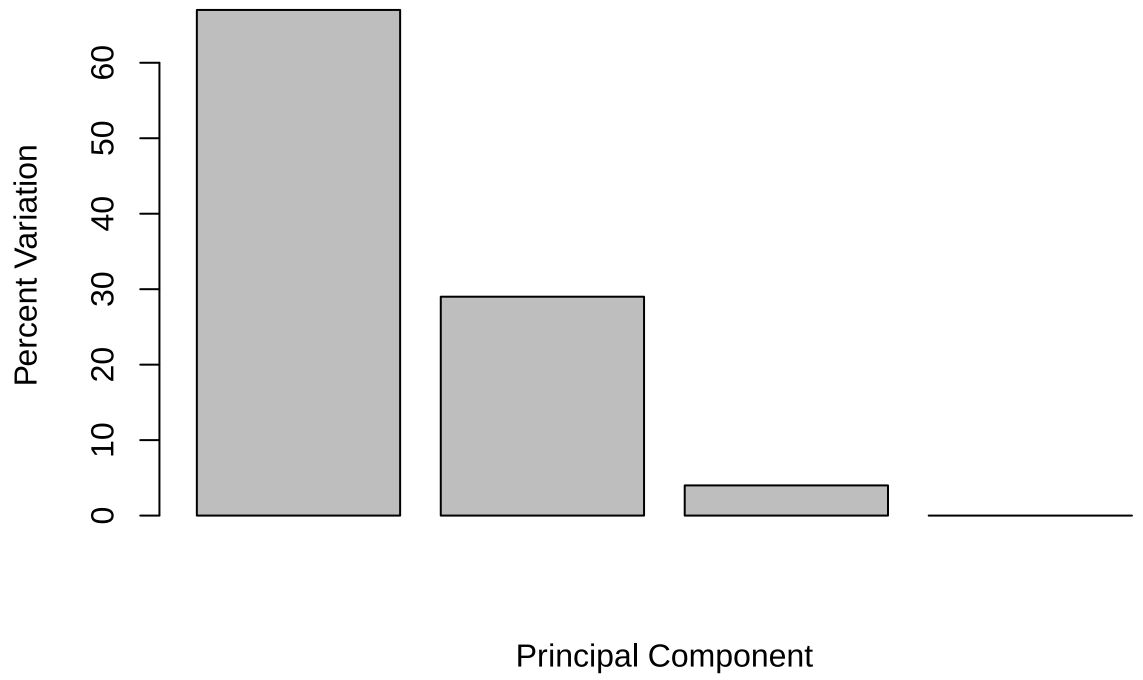
```
## [1] 67 29  4  0
```

```
## 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
```

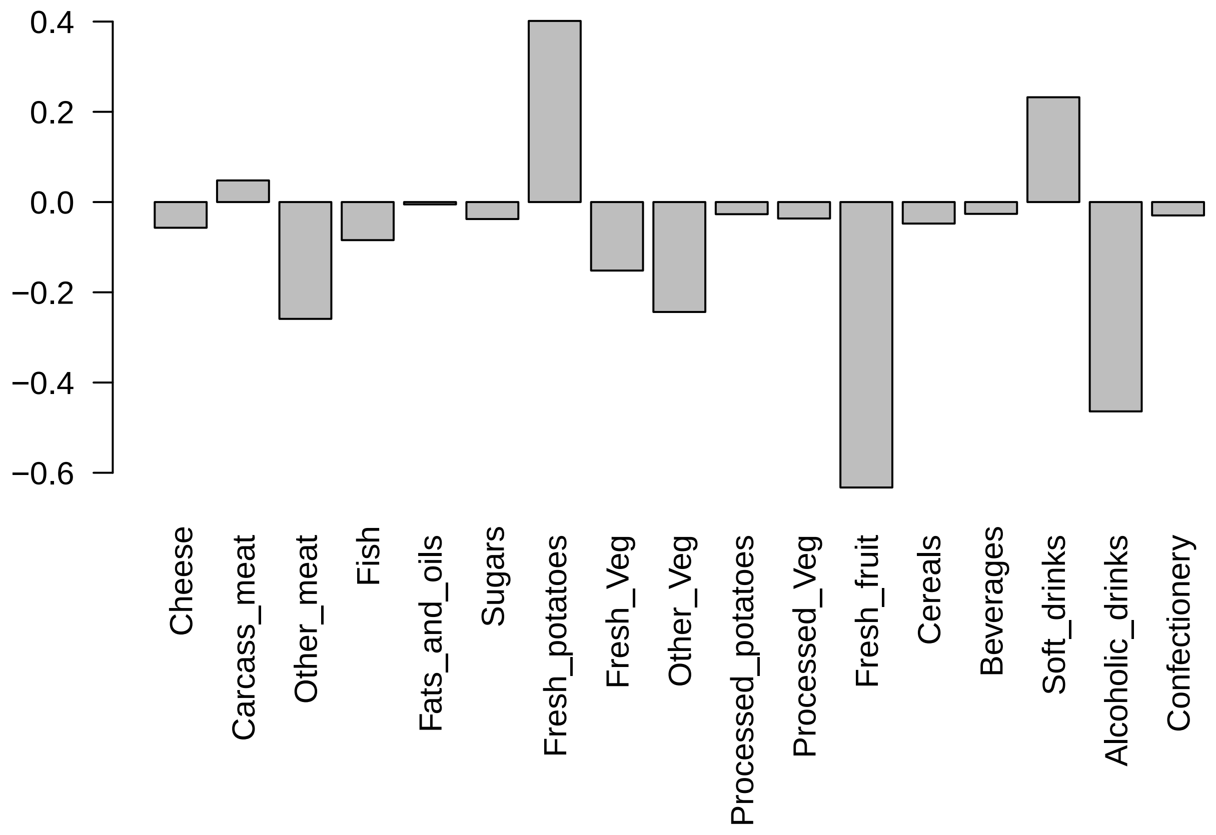
We can plot the variance with respect to PC number

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



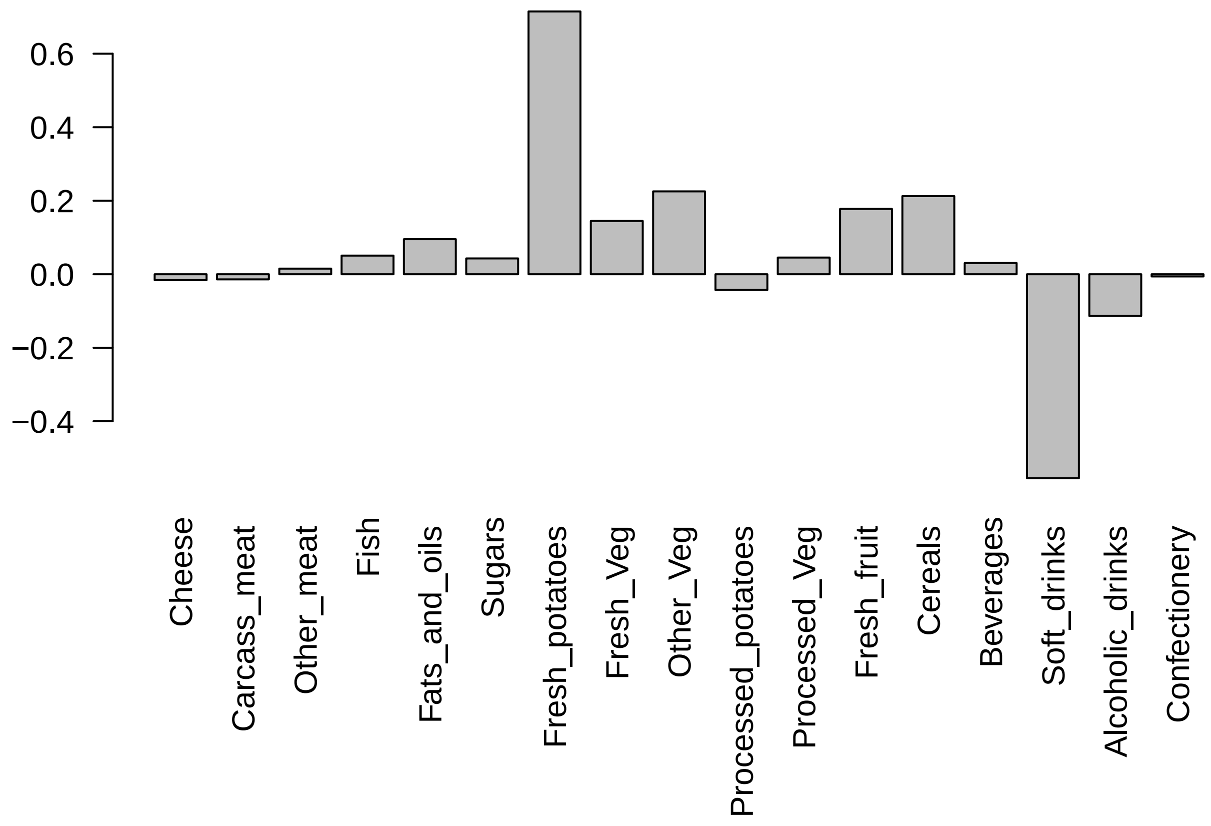
We can also consider the influence of each of the original variables on the PC's (loading scores).

```r
## 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 )
```



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

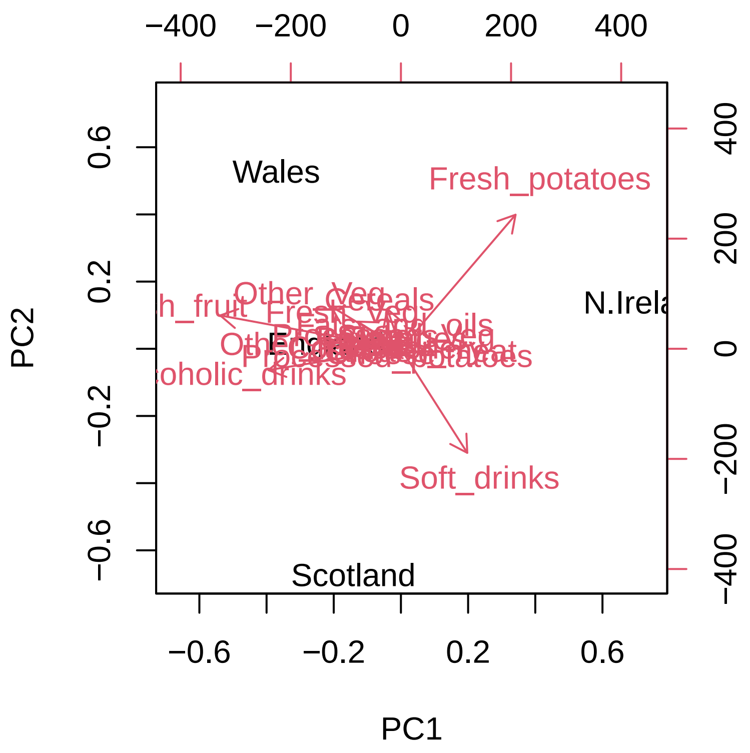
```
## Lets focus on PC2 now
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



The two prominently featured food groups in the loadings plot for PC2 are Fresh_potatoes and Soft_drinks. This tells us that the main difference between N. Ireland and the rest of the UK lies mainly within their consumption of Fresh_potatoes and Soft_drunks.

Another way to see these differences would be to generate a biplot.

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



Here we can see that Fresh_potatoes and Soft_drinks are the most outstanding, pointing towards N. Ireland, meaning that N. Ireland consumes the most of these two food choices compared to the rest of the UK.

PCA of RNA-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?

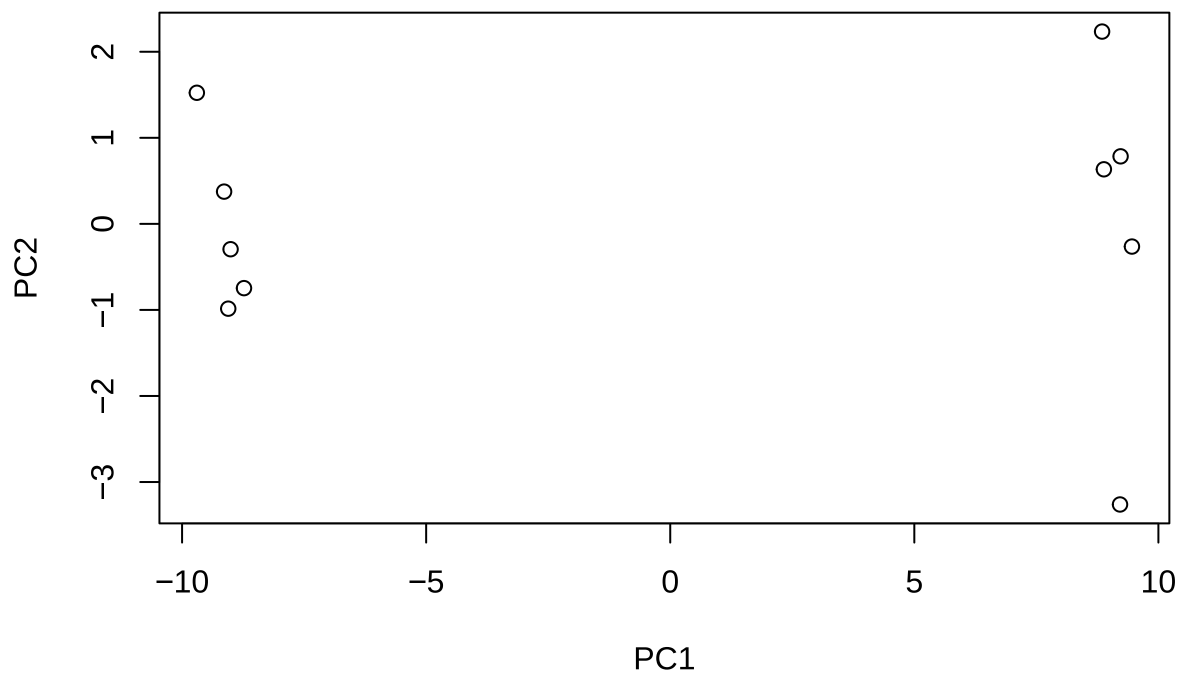
```
dim(rna.data)
```

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

There are 100 genes and 10 samples in this data set.

Let's do PCA to plot the results:

```
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

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

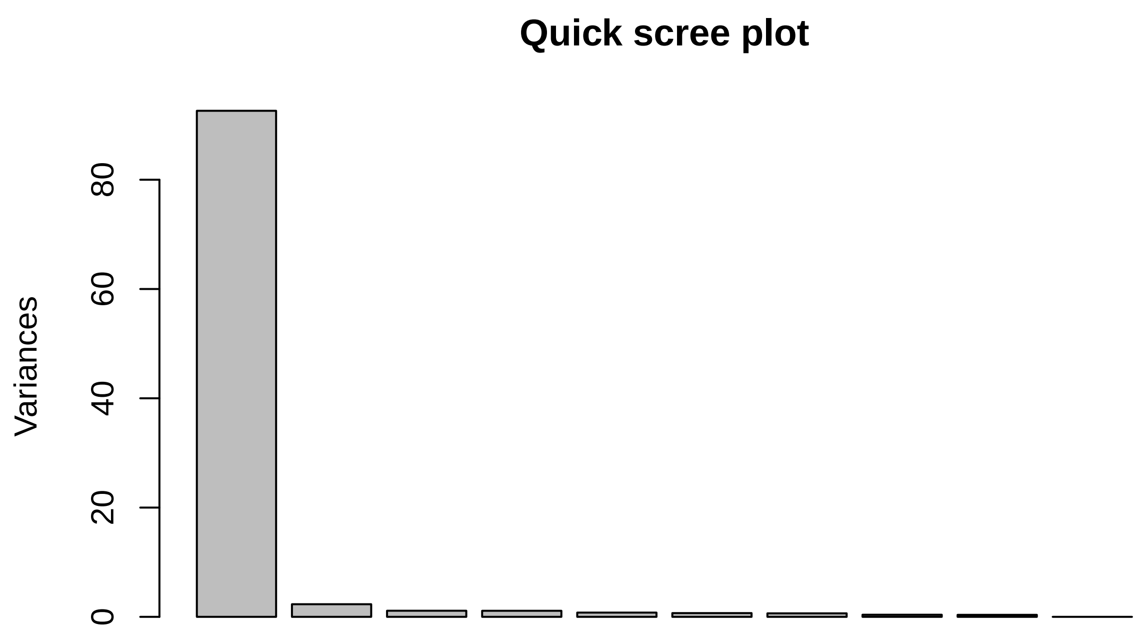


```
summary(pca)
```

```
## 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
```

PC1 is where most of the variation within the data is.

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

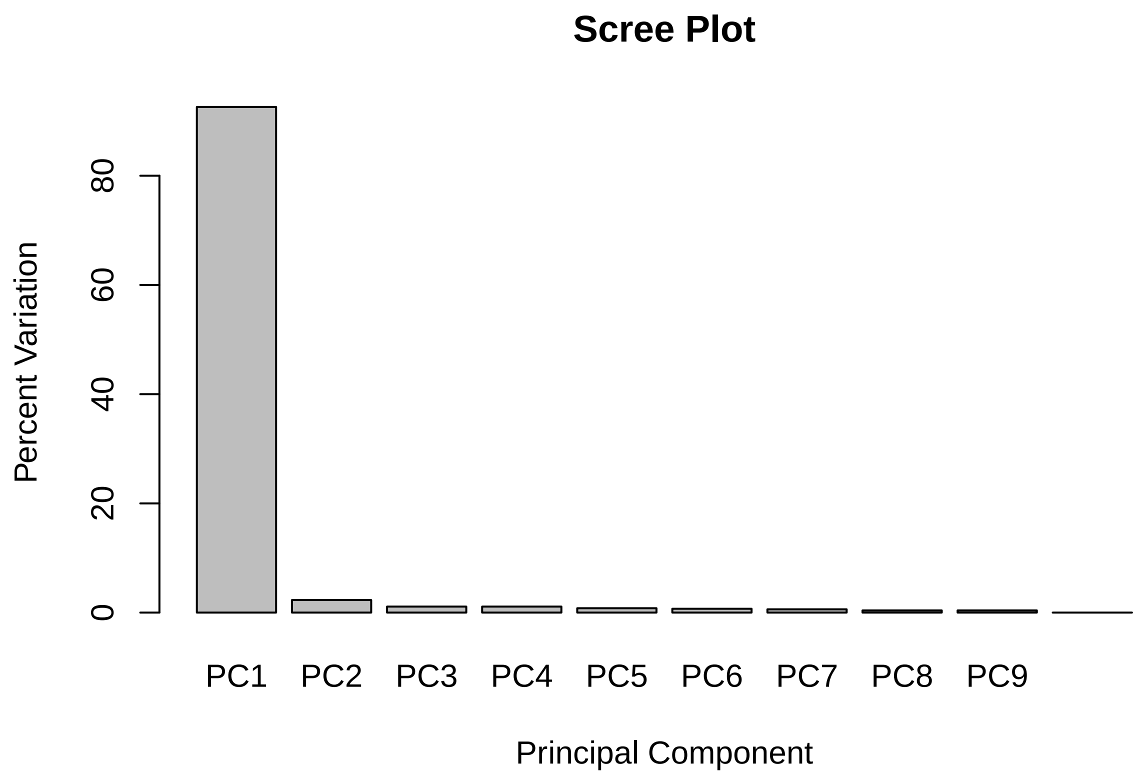


We can use pca$sdev to calculate the amount of variation in the data set that PC accounts for.

```r
## 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
```

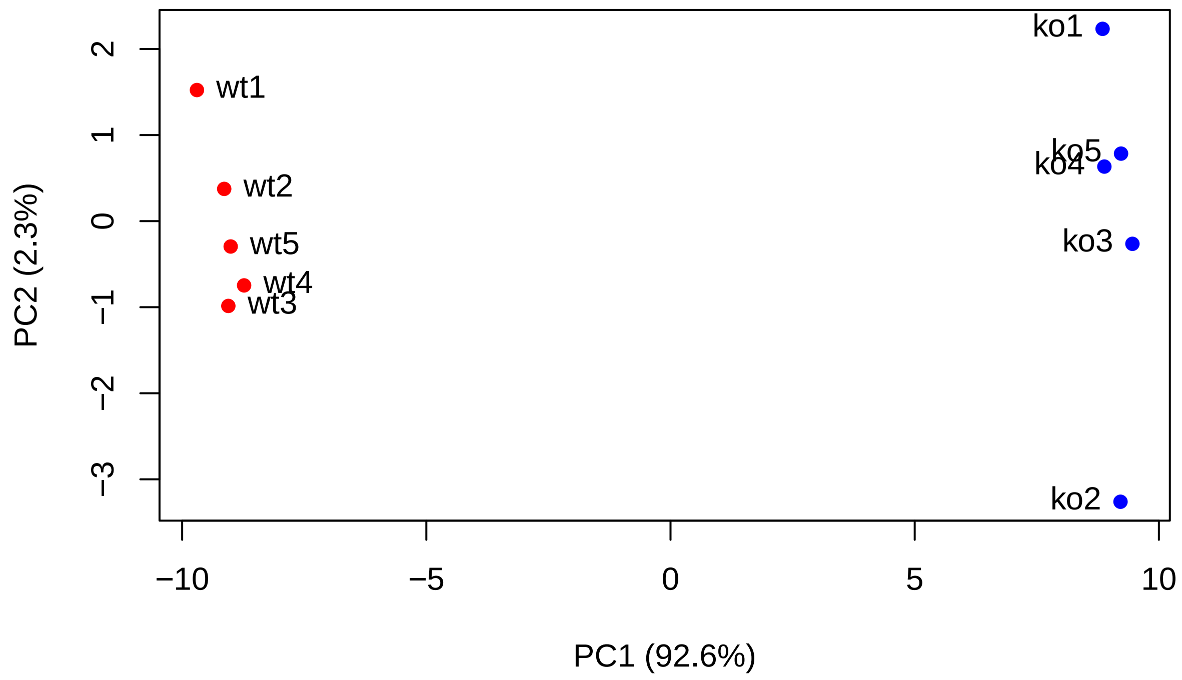
```r
## Generate our own scree-plot.
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



PC1 covers the most variation.

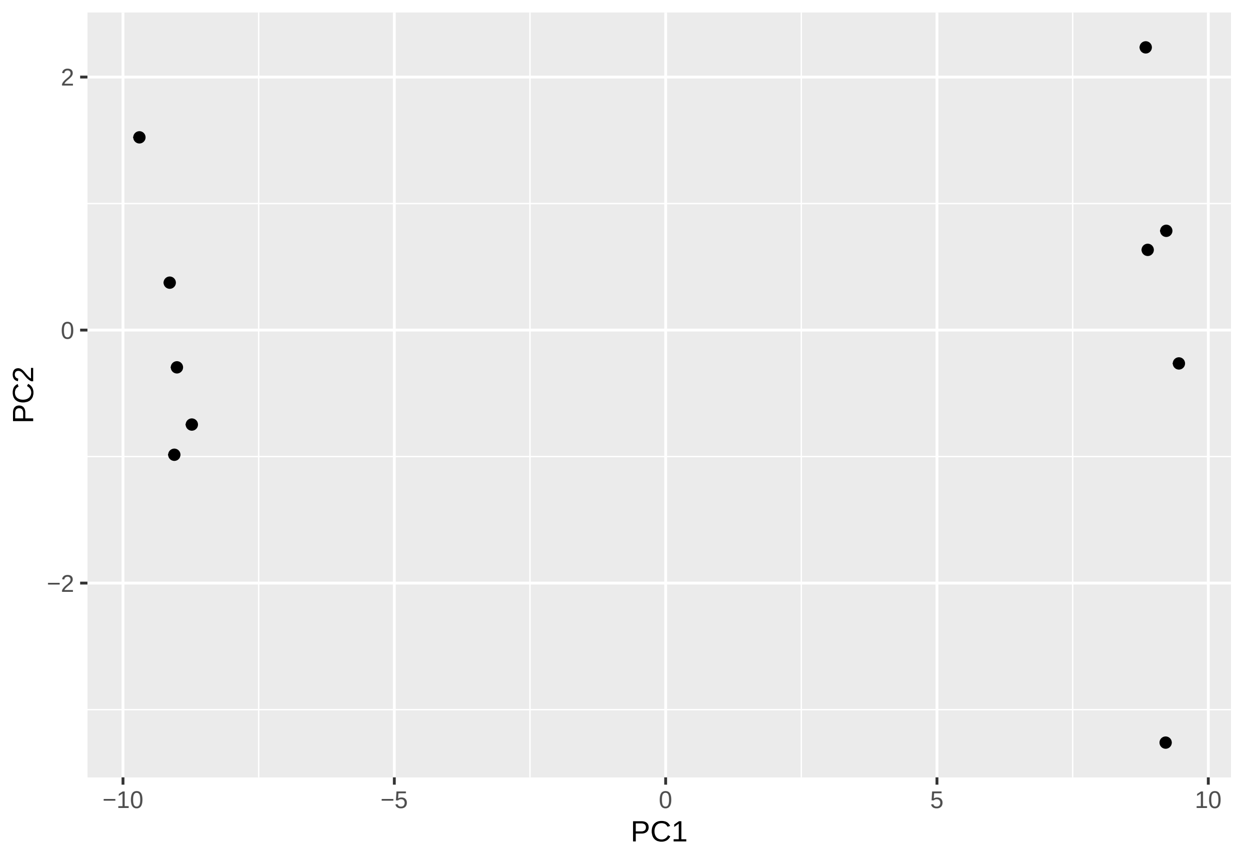
Let's make the PCA look more interesting.

```r
## 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)))
```



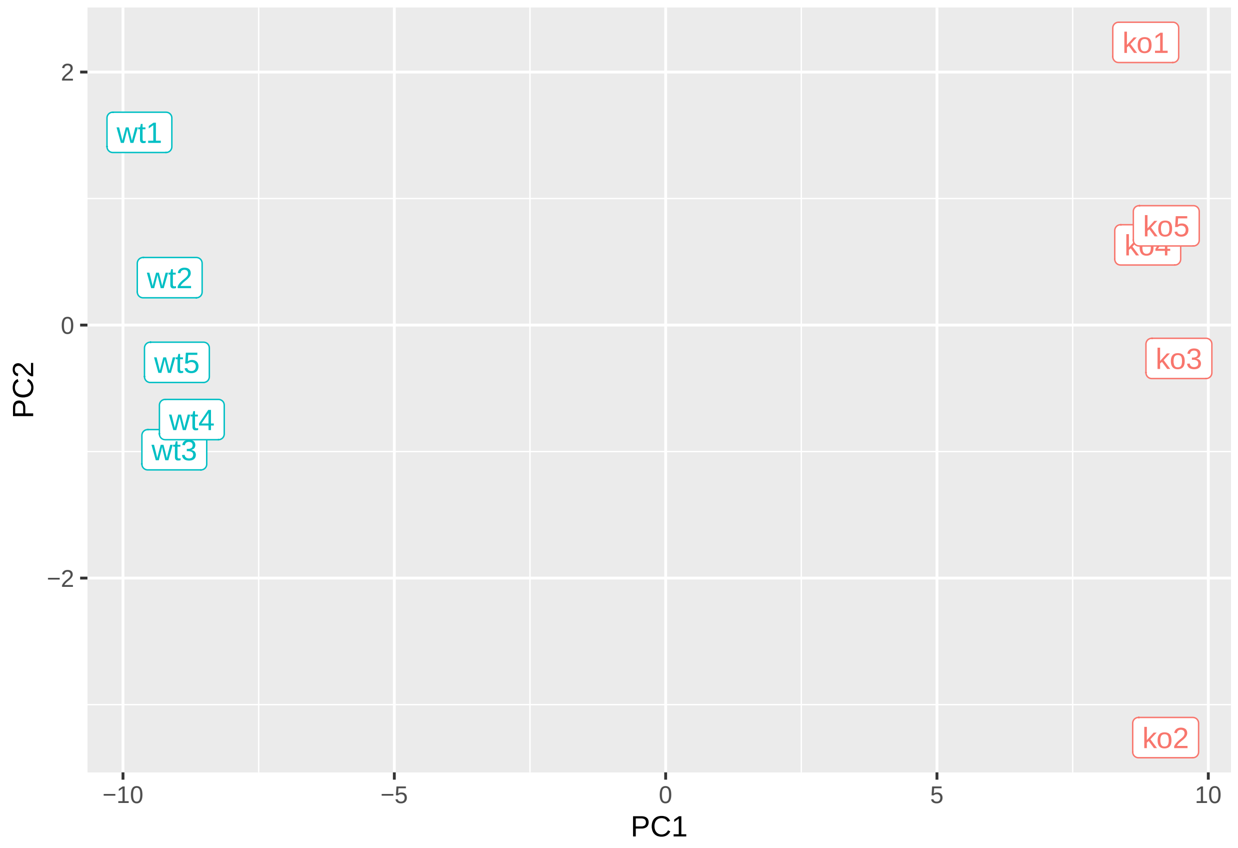
We can also use ggplot to make our PCA.

```
library(ggplot2)

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

# Our first basic plot
ggplot(df) + aes(PC1, PC2) + geom_point()
```



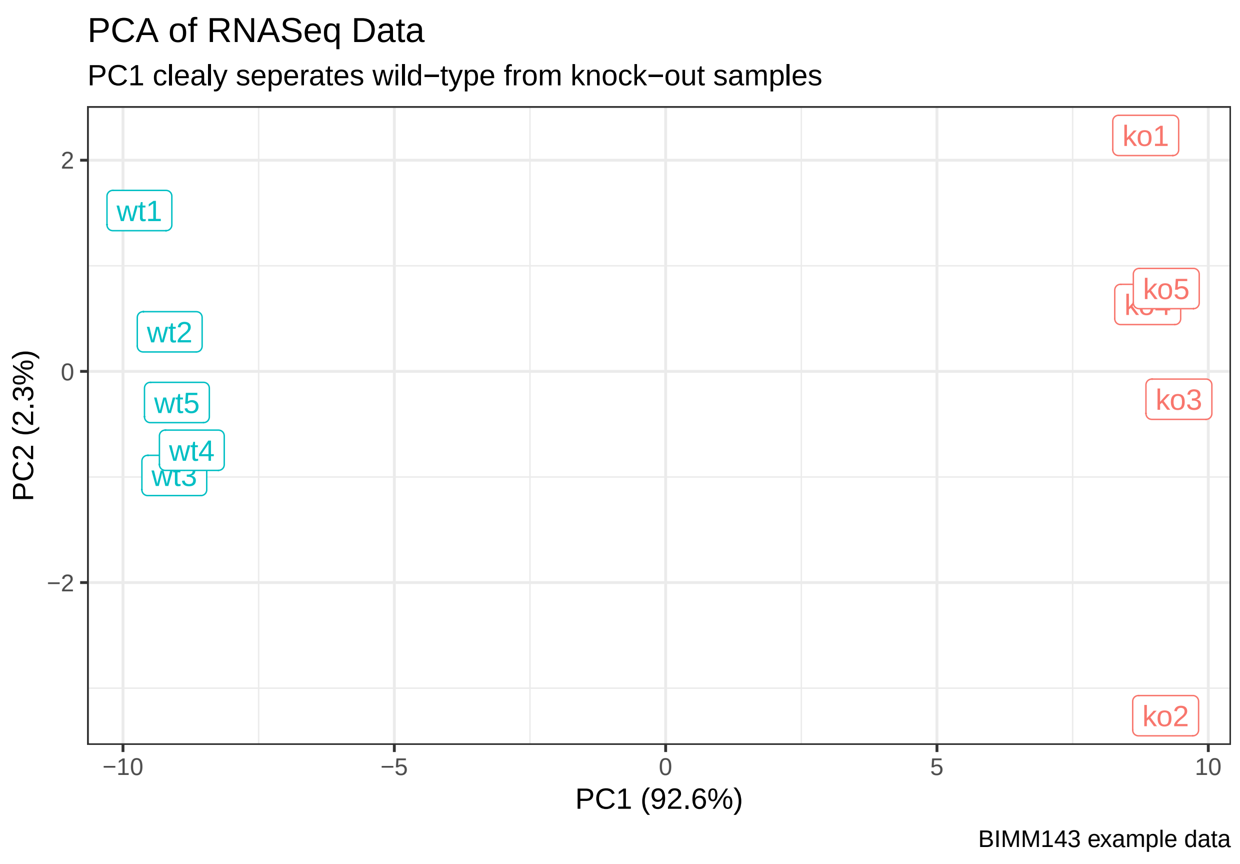
We can add more aesthetics to the WT and KO data points.

```
# 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
```



Now, add a title and caption to make the plot look nicer.

```
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="BIMM143 example data") +
     theme_bw()
```


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