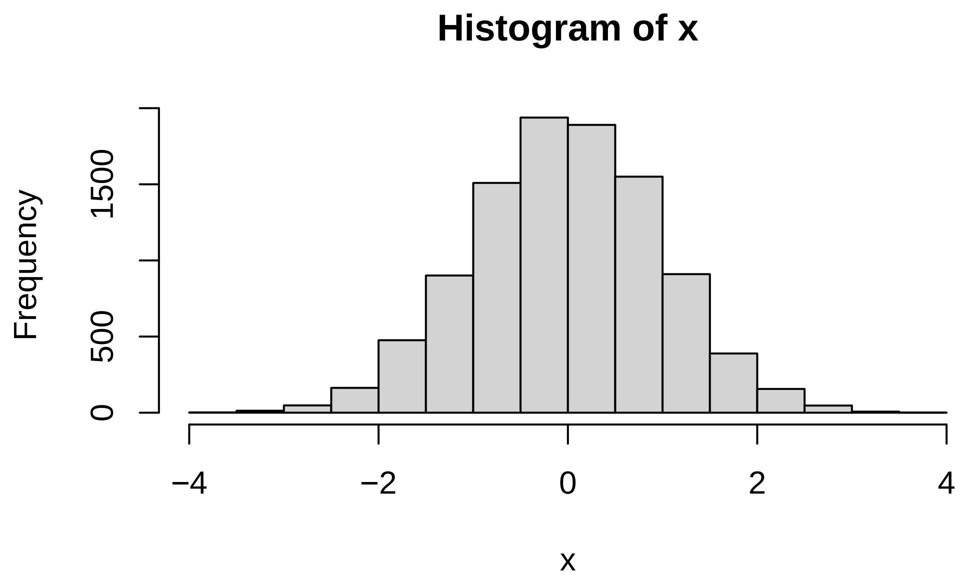
Class 7 Machine Learning 1

Chloe Do

K-means clustering

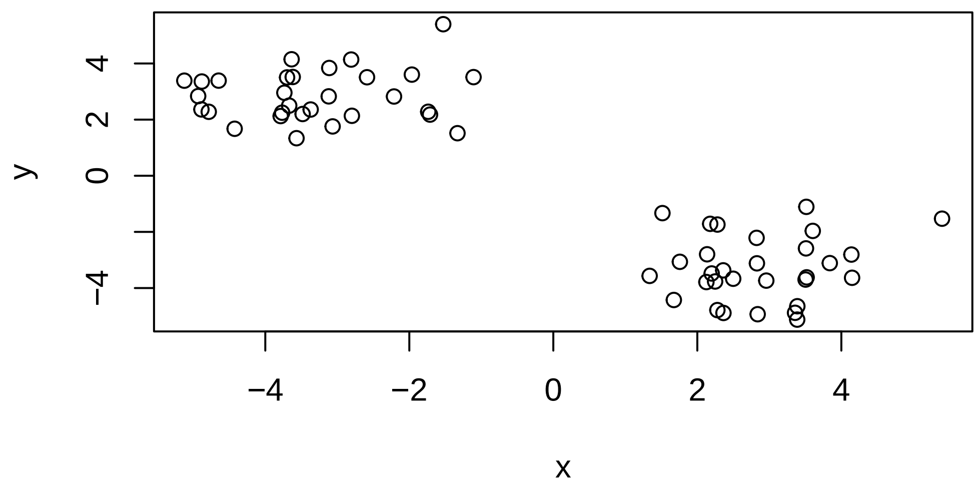
First we will test how this method works in R with some made up data

```r
x <- rnorm(10000)
hist(x)
```



Let's make some numbers centered on -3

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

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



Now let's see how `kmean()` works with this data…

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

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1  2.856846 -3.303110
2 -3.303110  2.856846

Clustering vector:
 [1] 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 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
Within cluster sum of squares by cluster:
[1] 62.604 62.604
 (between_SS / total_SS =  90.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
km$centers
```

```
          x         y
1  2.856846 -3.303110
2 -3.303110  2.856846
```

> Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

> Q. What ‘component’ of your result object details - cluster assignment/membership - cluster center

```
km$cluster
```

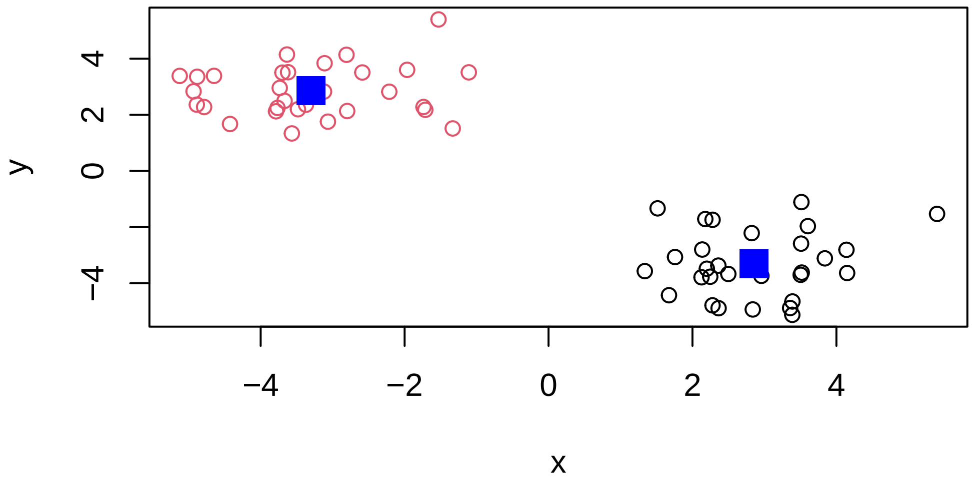
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
km$centers
```

```
          x         y
1  2.856846 -3.303110
2 -3.303110  2.856846
```

> Q. 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, cex=2)
```



Hierarchical Clustering

The **hclust()** function in R performs hierarchial clustering.

```r
#hclust()
```

The **hclust()** function requires an input distance matrix, which I can get from the **dist()** function.

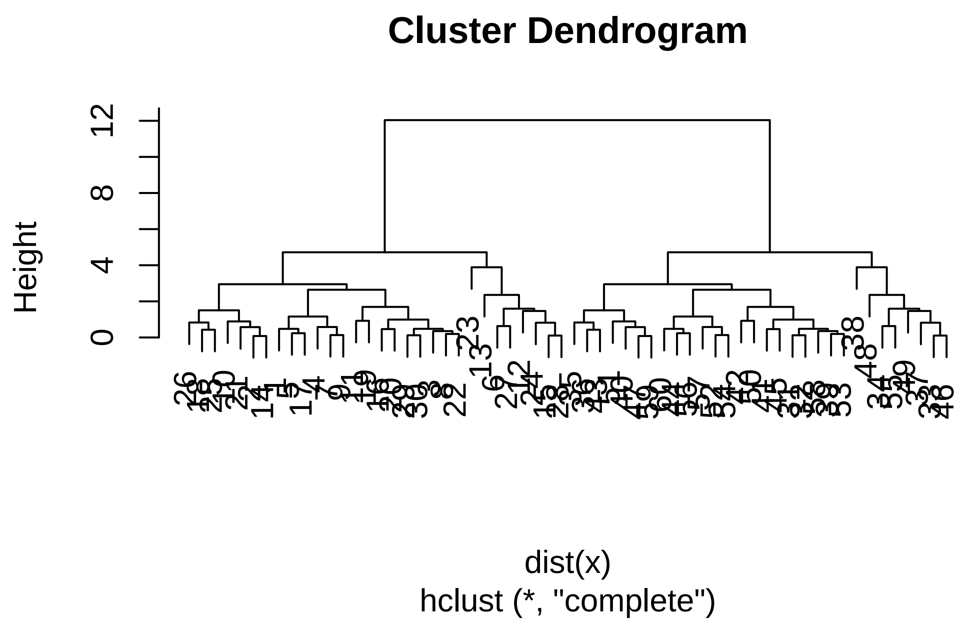
```r
hc <- hclust(dist(x))
hc
```

```

Call:
hclust(d = dist(x))

Cluster method   : complete
```

```
Distance         : euclidean
Number of objects: 60
```

There is a plot method for hclust objects…

```
plot(hc)
```



Now to get my cluster membership vector I need ro "cut" the tree to yield separate "branches" with the "leaves" on each branch being our clusters. To do this we use `cutree()`

```
cutree(hc, h=8)
```

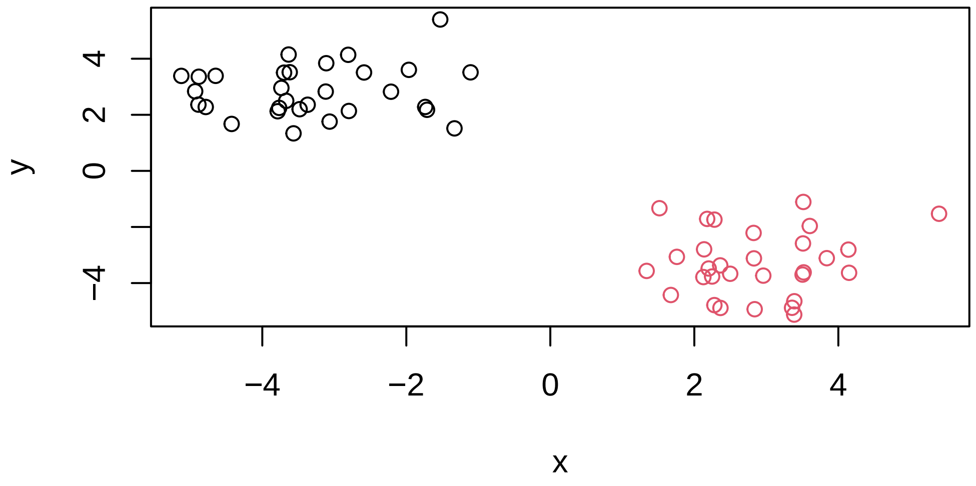
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Use `cutree()` with a k=2

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

A plot of our data colored by our hclust grps

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



Principal Component Analysis (PCA)

First we will read the provided UK_foods.csv input file (note we can read this directly from the following tinyurl short link:

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
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
7    Fresh_potatoes       720   874      566      1033
8          Fresh_Veg      253   265      171       143
```

```
9            Other_Veg      488   570       418       355
10 Processed_potatoes      198   203       220       187
11       Processed_Veg      360   365       337       334
12         Fresh_fruit     1102  1137       957       674
13             Cereals     1472  1582      1462      1494
14           Beverages       57    73        53        47
15         Soft_drinks     1374  1256      1572      1506
16    Alcoholic_drinks      375   475       458       135
17       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?

```r
## Complete the following code to find out how many rows and columns are in x?
dim(x)
```

```
[1] 17  5
```

```r
## Preview the first 6 rows
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
```

```r
# Note how the minus indexing works
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
```

This looks much better, now lets check the dimensions again:

```
dim(x)
```

```
[1] 17  4
```

An alternative approach:

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

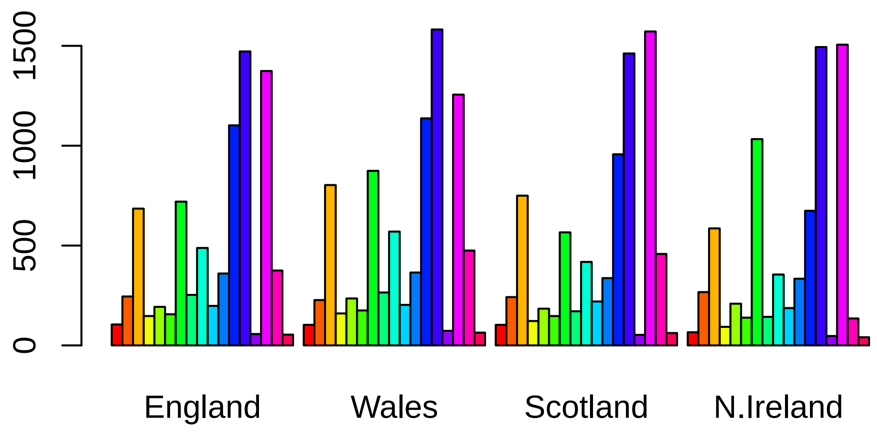
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             156   175      147       139
```

> 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 the alternative approach because it looks so much cleaner compared the first one. The first one seems more dangerous because we can mistakenly input the wrong values and if we keep running `x <- x[, -1]` it will keep deleting the column.

Spotting major differences and trends

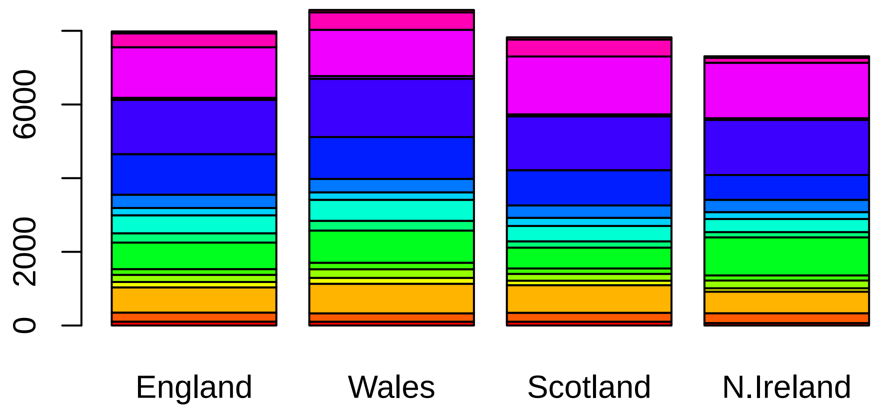
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

We can change `beside=T` to `beside=F`

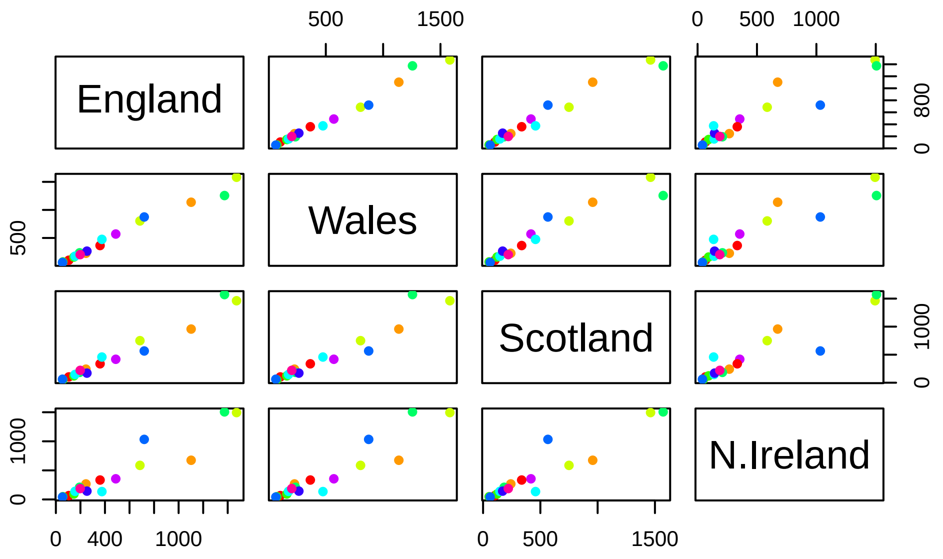
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



> 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?

The plots represent each country on x-axis and y-axis in different graph. Each point that lies on the diagonal on a given plot represent the same amount of food consumed by the two countries on the x and y axis. Any point that lies above the diagonal mean the food is consumed more by the country on the y-axis while any point that lies below the diagonal mean that the food is consumed more by the country on the x-axis.

```
pairs(x, col=rainbow(10), pch=16)
```



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 other countries is that N. Ireland always have one point lies above or beyond the diagonal compared to other countries which mean there is a certain food that N. Ireland consume more than any other countries.

While this is kind of useful it takes work to dig into details here to find out what is different in these countries.

PCA to the rescue

Principal Component Analysis (PCA) can be a big help in these cases where we have a lot of things that are being measured in a data set.

The main PCA function in base R is called `prcomp()`.

The `prcomp()` function wants as input the transpose of our food matrix/table/data.frame.

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 5.552e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

The above results show PCA captures 67% the total variance in the original data in one PC and 96.5 in two PCs.

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

```
attributes(pca)
```

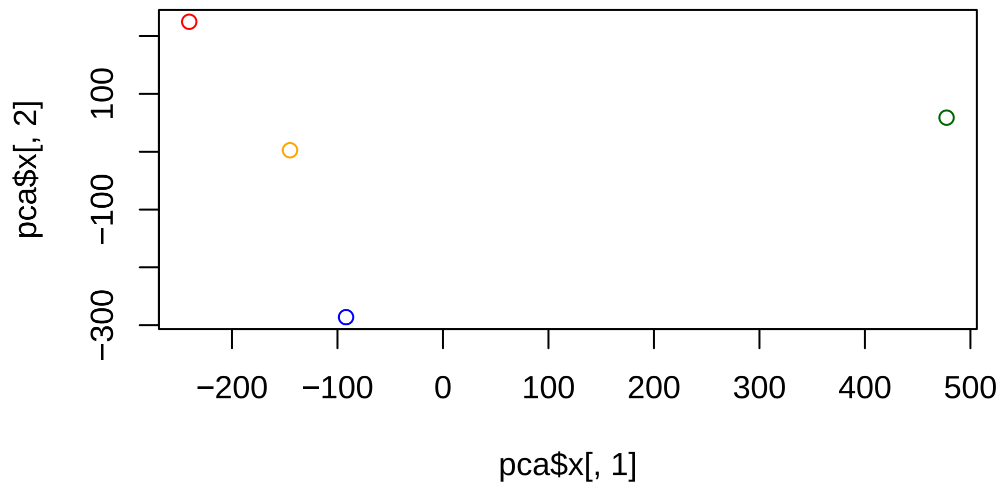
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
head(pca$x)
```

```
                 PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  1.042460e-14
Wales     -240.52915  224.646925   56.475555  9.556806e-13
Scotland   -91.86934 -286.081786   44.415495 -1.257152e-12
N.Ireland  477.39164   58.901862    4.877895  2.872787e-13
```

Let's plot our main results. > 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], col=c("orange", "red", "blue", "darkgreen"))
```



Below we can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for.

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

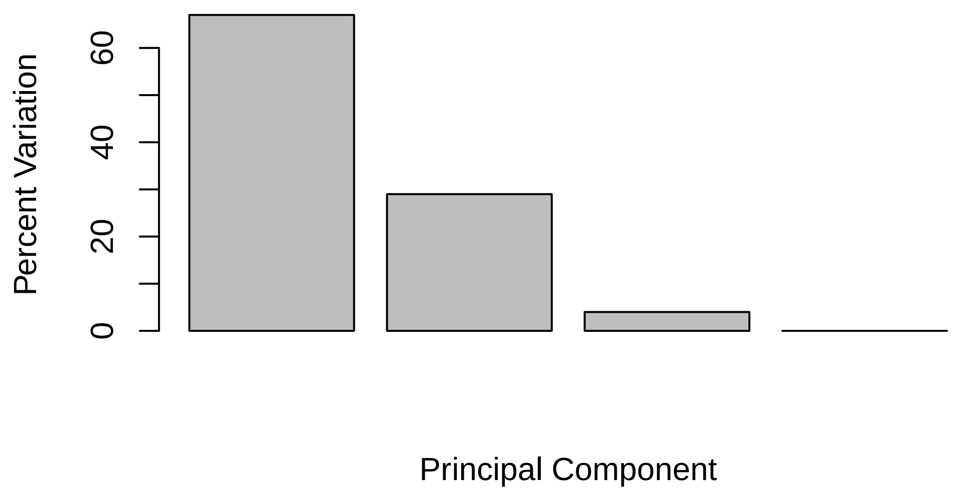
```
[1] 67 29  4  0
```

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

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 5.551558e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

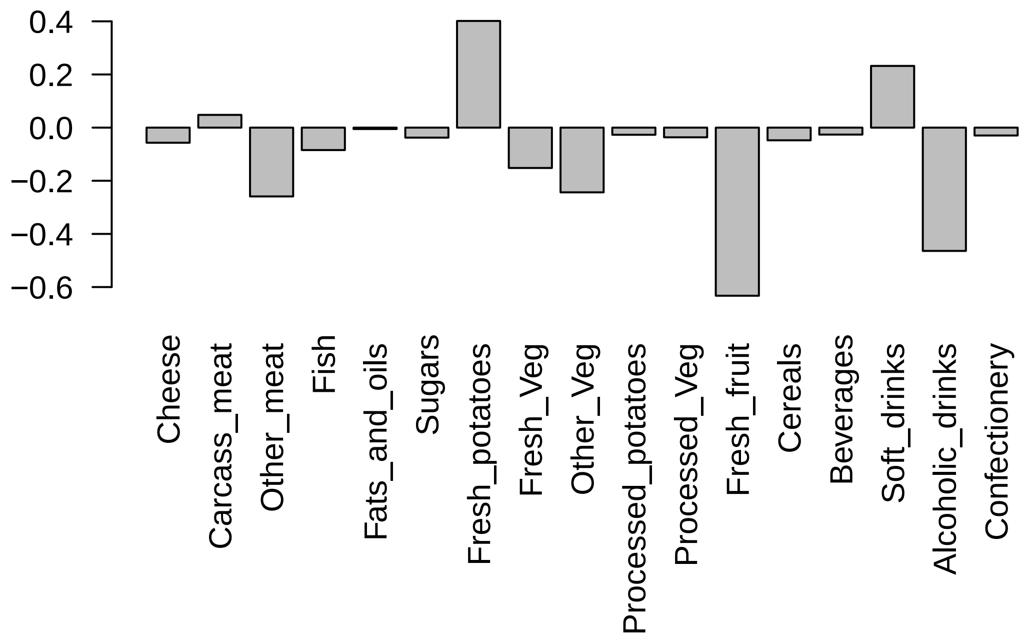
Convert this information into plot.

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



Digging deeper (variable loadings)

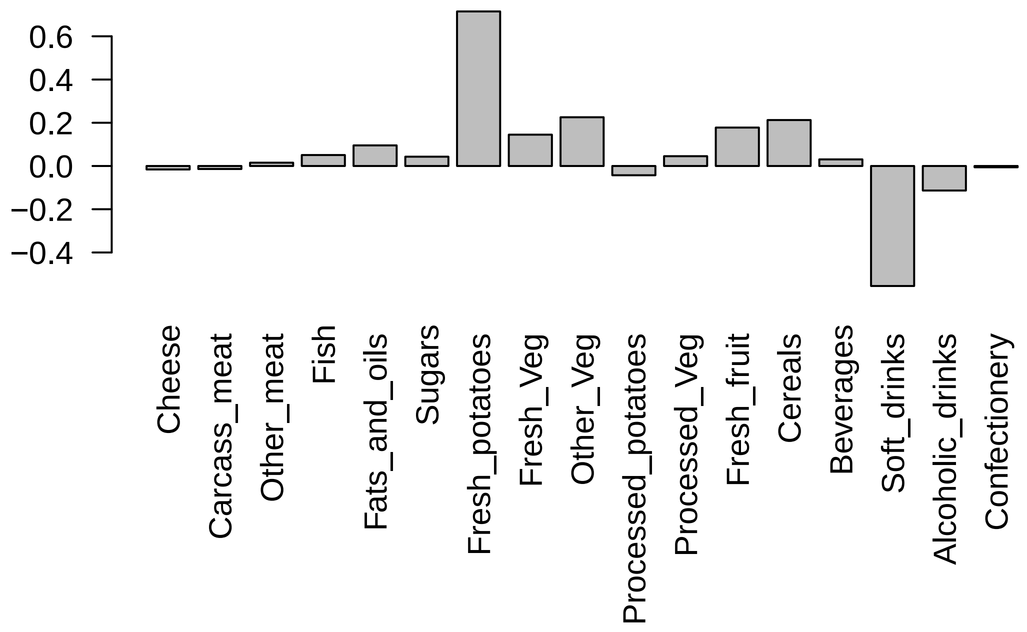
```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

The two prominantely groups are soft drinks and alcoholic drinks

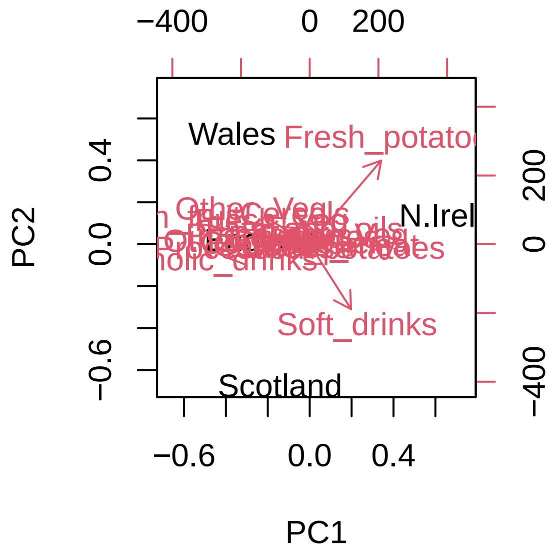
```
## Looking at PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



Biplots

Another way to see this information together with the main PCA plot is in a so-called biplot:

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



2. PCA of RNA-seq data

Here we apply PCA to some example RNA-Seq a know-out experiment.

First, read the dataset:

```r
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?

There are 100 genes and 10 samples

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

```
[1] 100  10
```

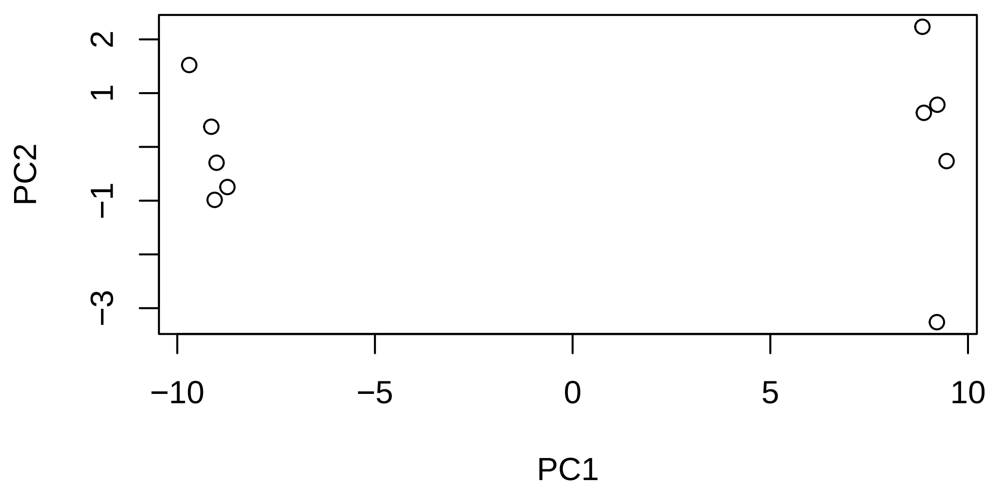
Now PCA

```
## Again we have to take the transpose of our data
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 3.327e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

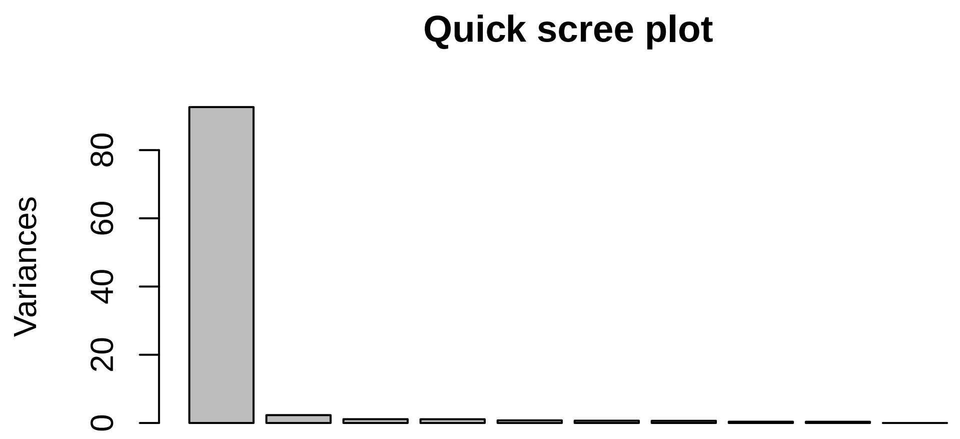
Now plot

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



A quick barplot summary of this Proportion of Variance for each PC can be obtained by calling the plot() function directly on our prcomp result object.

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



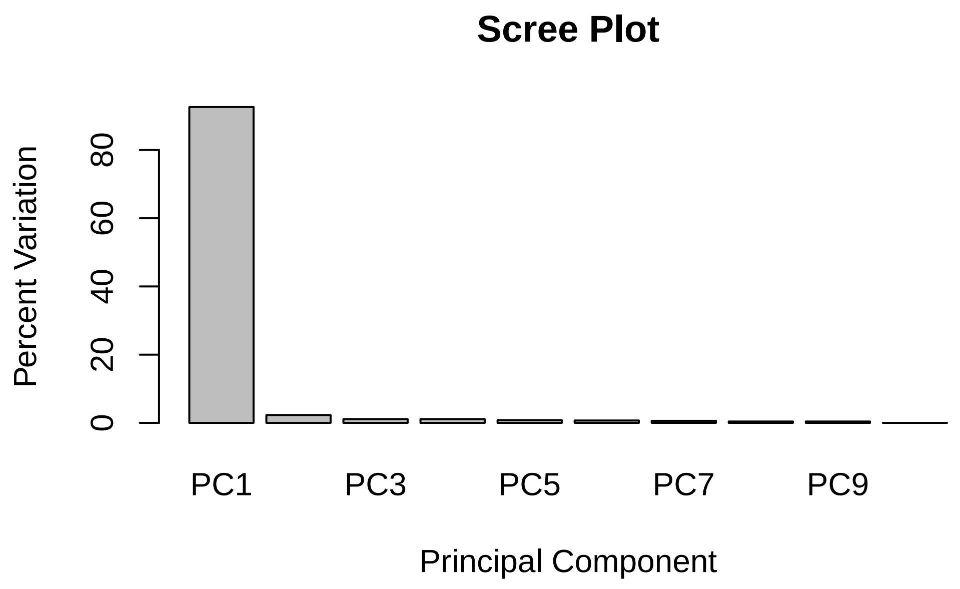
Let's make the above scree plot

```
## 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
```

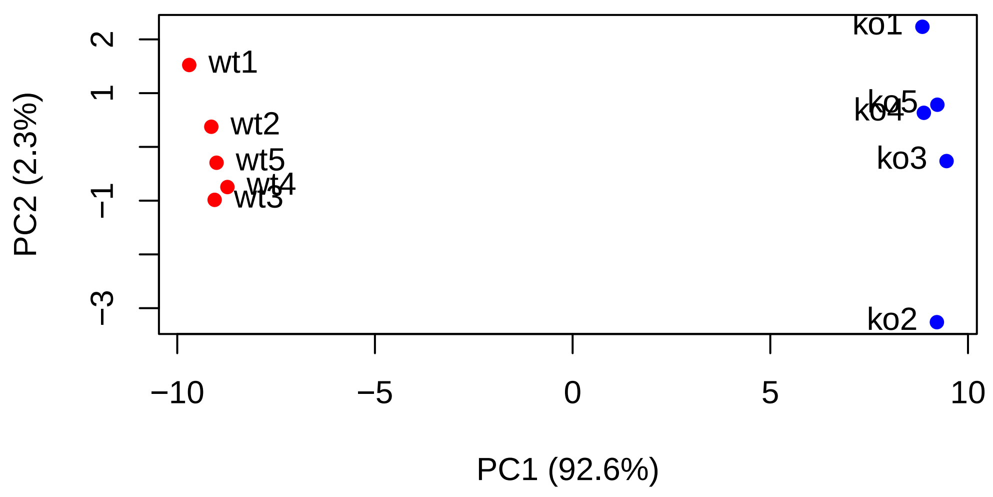
We can use this to generate our own scree-plot

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



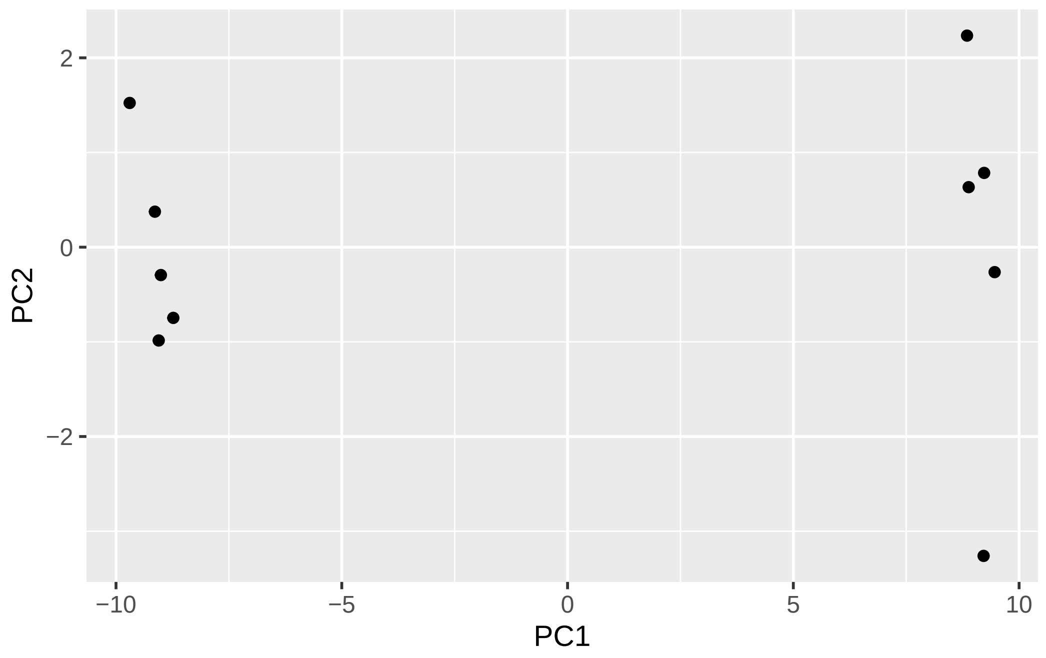
Now lets make our main PCA plot a bit more attractive and useful…

```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)))
```



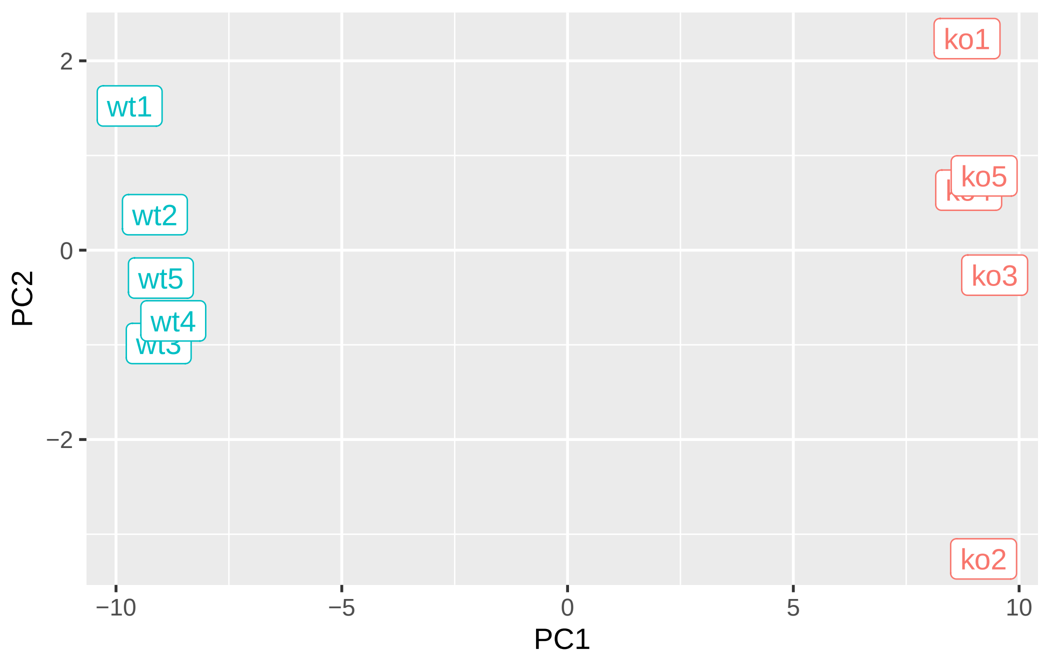
Using ggplot

```
library(ggplot2)

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

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



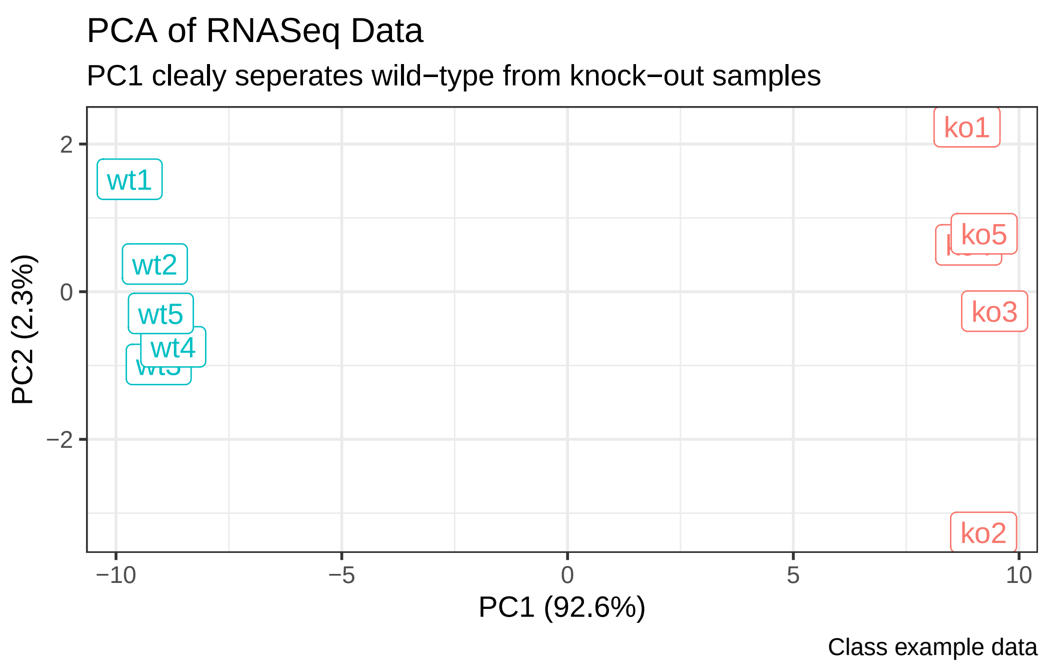
Add a condition specific color and for wild-type and knock-out samples, we need to have this information added to our data.frame:

```
# 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
```



And finally add some spit and polish

```
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()
```



Gene loadings

```r
loading_scores <- pca$rotation[,1]

## Find the top 10 measurements (genes) that contribute
## most to PC1 in either direction (+ or -)
gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## show the names of the top 10 genes
top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes
```

```
[1] "gene100" "gene66"  "gene45"  "gene68"  "gene98"  "gene60"  "gene21"
[8] "gene56"  "gene10"  "gene90"
```