

# Machine Learning 1

Leyna Nguyen (A15422197)

10/21/2021

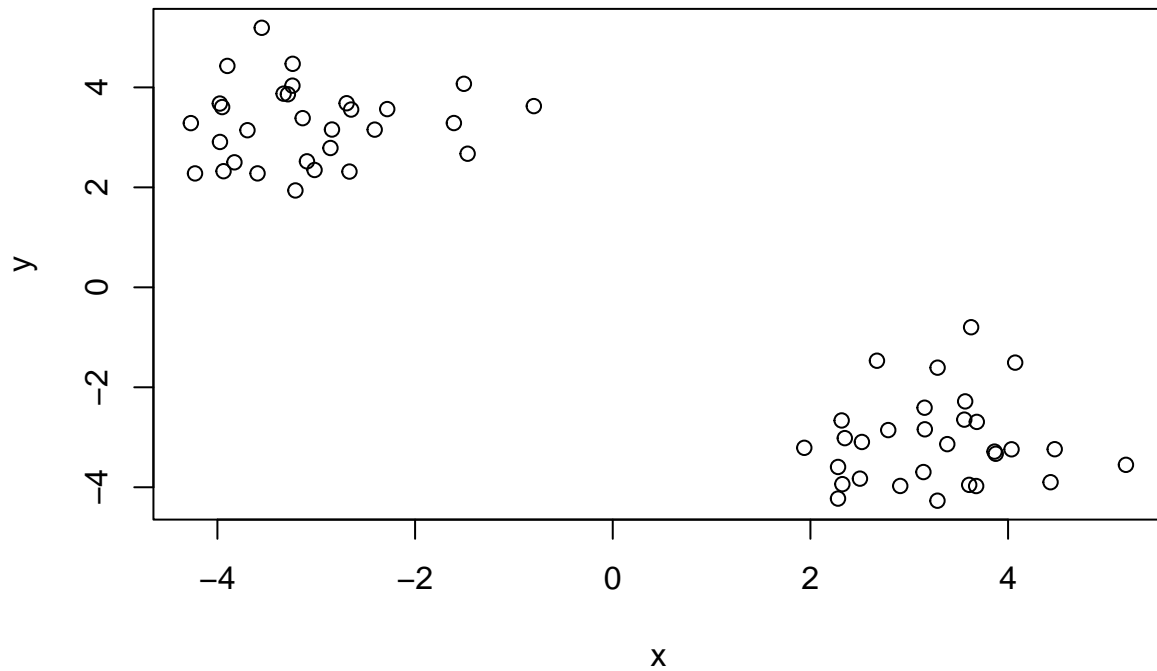
First up is clustering methods

## Kmeans clustering

The function in 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` 2 and `nstart` to 20?

[illegible]

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
```

[illegible]

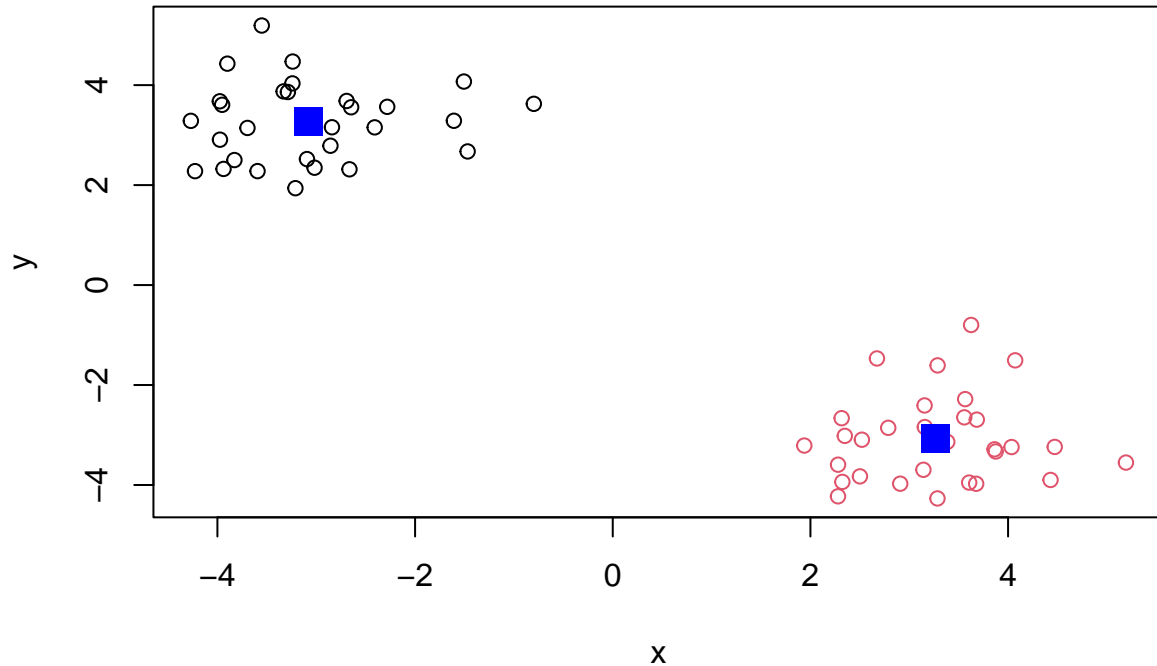
Q. What ‘component’ of your result object details cluster center?

km\$centers

```
##          x          y
## 1 -3.074476  3.264673
## 2  3.264673 -3.074476
```

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

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



## Hierarchical Clustering

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()`

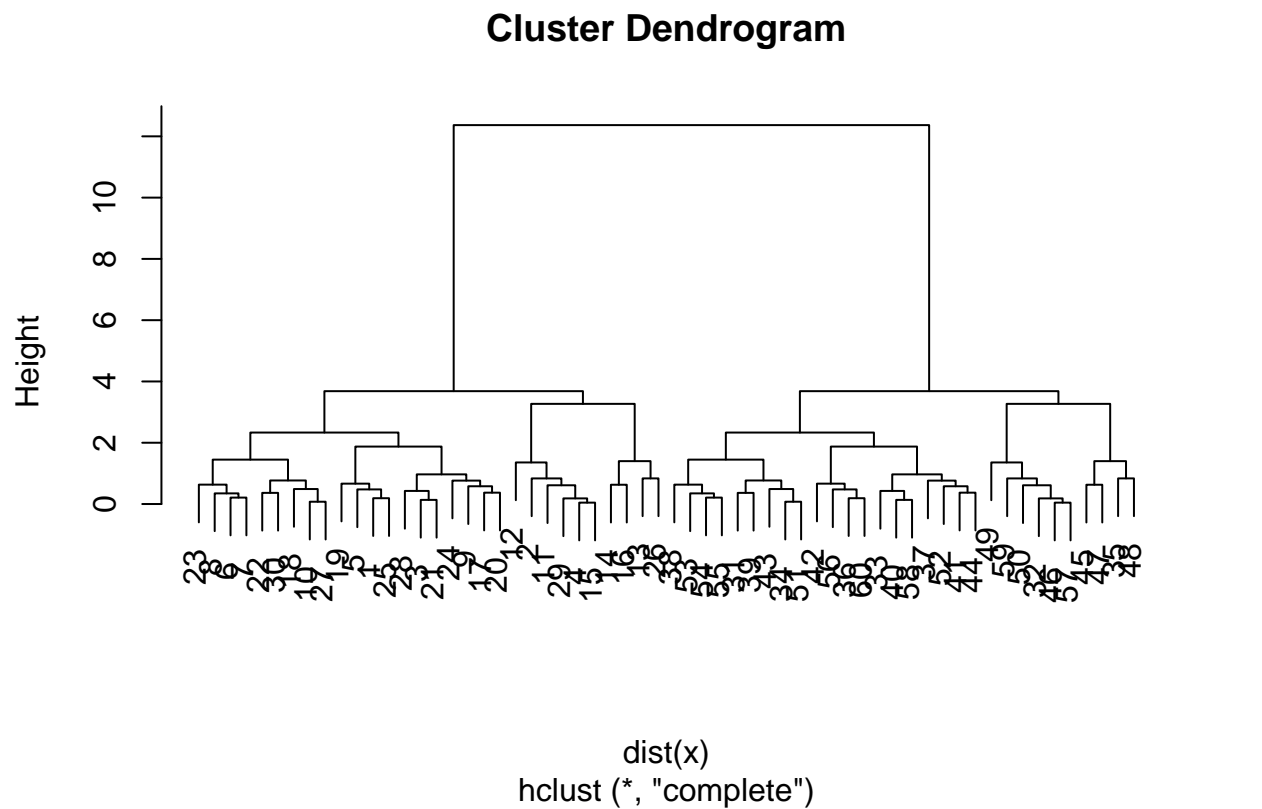
Demonstrate the use of `dist()`, `hclust()`, `plot()`, and `cutree()` functions to do clustering. Generate dendrograms and return cluster assignment/membership vector...

```
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 objects. Let's see it

```
plot(hc)
```



To get our cluster membership vector we have to do a wee bit more work. We have to “cut” the tree where we think it makes sense. For this we use the `cutree()` function.

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

[illegible]

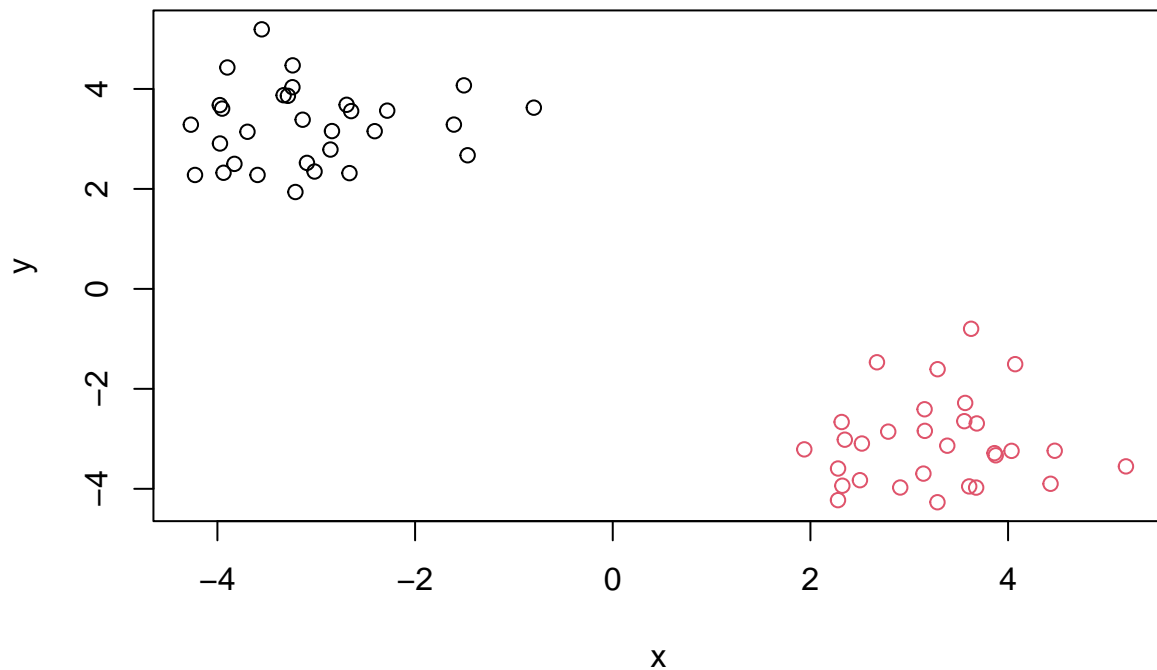
You can also call `cutree()` setting `k`=the number of groups/clusters you want

```
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 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
```

Make our results plot

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



## Principal Component Analysis (PCA)

### PCA of UK food 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
```

Checking the data

```
View(x)
```

### 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
```

Check the dimensions

```
dim(x)
```

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

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

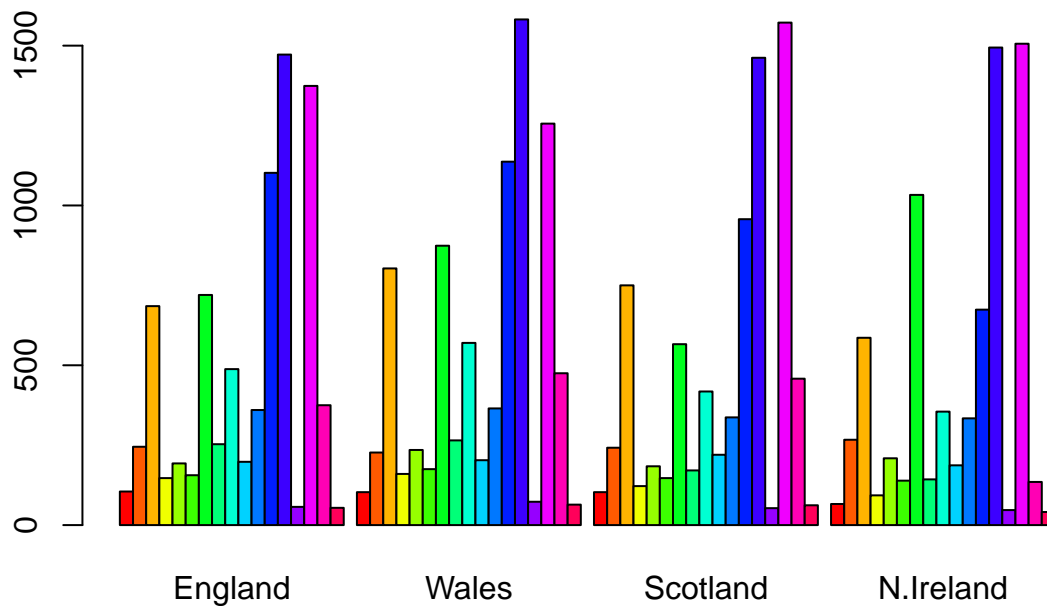
```
##           England Wales Scotland N.Ireland
## Cheese           105   103       103        66
## Carcass_meat      245   227       242       267
## Other_meat        685   803       750       586
## Fish              147   160       122        93
## Fats_and_oils      193   235       184       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 second approach because as you run the code block `x <- x[, -1]` over and over again, the next column on the right will become the first column and the left-most column will be removed. Once you run the code block enough times, you’ll receive an error message because ... Yes, the second approach is more robust because it resets the first column to be the rows and once it does that, running the code block multiple times won’t change that.

Now we have the data looking good, we want to explore it. We will use some conventional plots (barplots and pair plots).

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



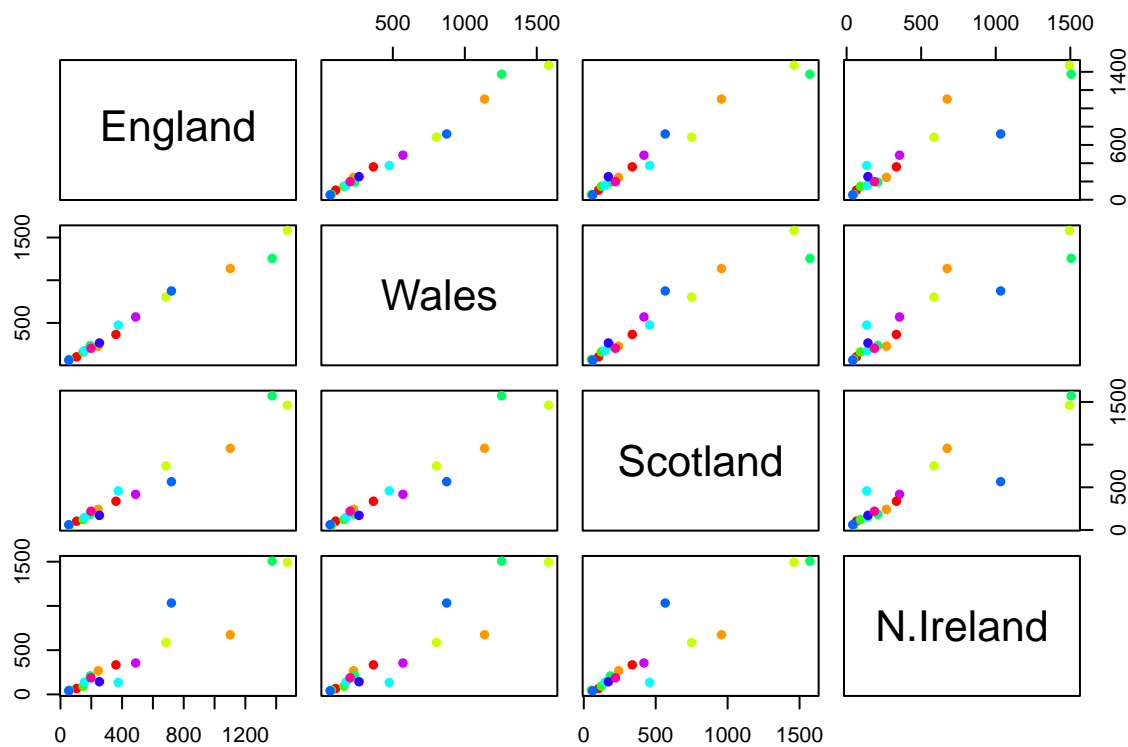
Q3: Changing what optional argument in the above `barplot()` function results in the following plot?

Changing the argument `beside=FALSE` in the `barplot()` function will result in a barplot with stacked bars instead of juxtaposed bars.

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?

If a point lies on the diagonal, this indicates that the quantity of food eaten, whichever one it may be, is the same in the two countries being measured on the x-axis and y-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?

One of the main differences is that N. Ireland consumes less fresh fruit, alcoholic drinks, other meat, and other vegetables, but more fresh potatoes than the other countries in the data-set.

## PCA to the rescue!

The main function in base R for PCA is `prcomp()` This wants the transpose of our data

## 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
```



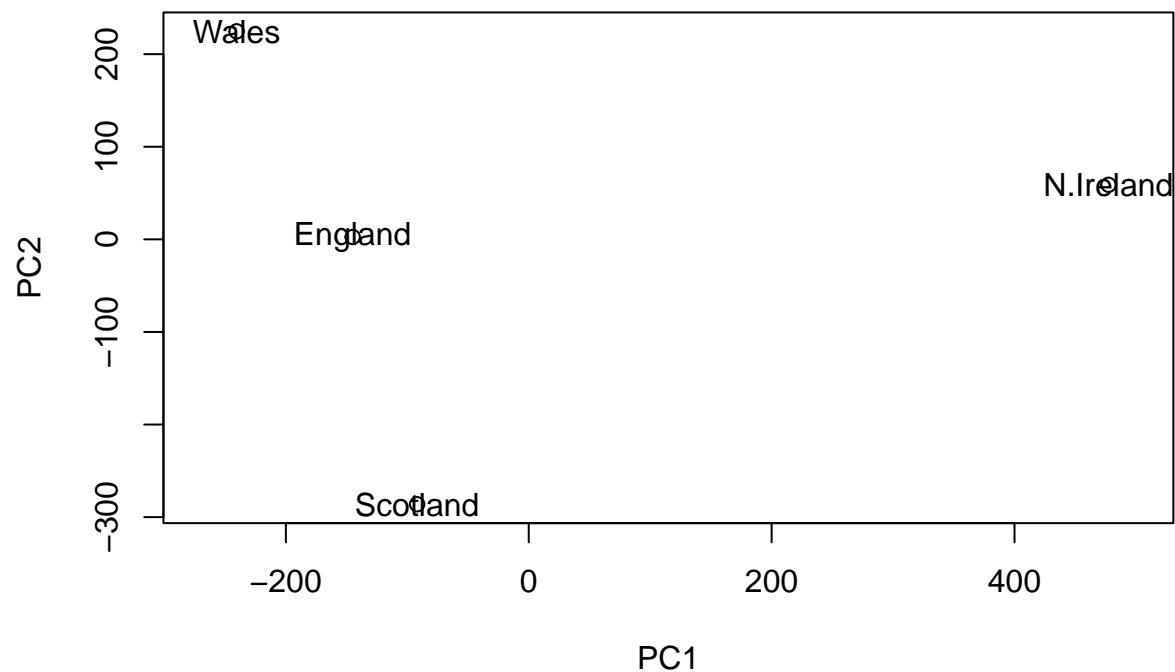
```
pca
```

```
## 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
```

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

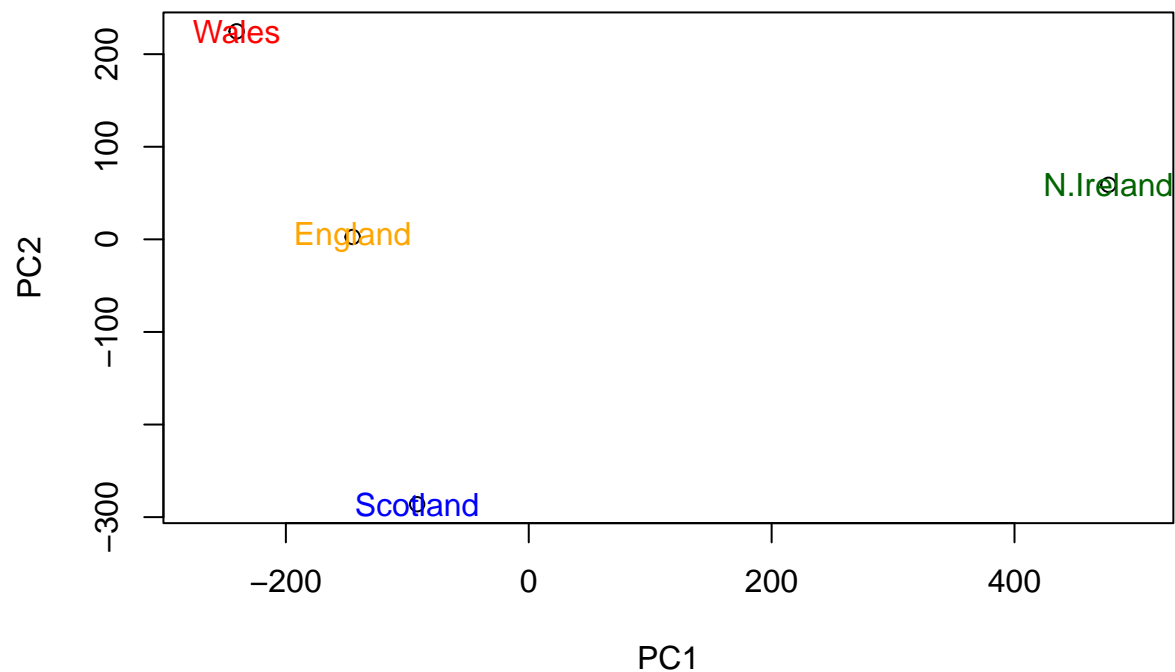
## Plot PC1 vs PC2

```
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.

```
country_cols <- c("orange", "red", "blue", "dark green")
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=country_cols)
```



How much variation in the original data does each PC account for?

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

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

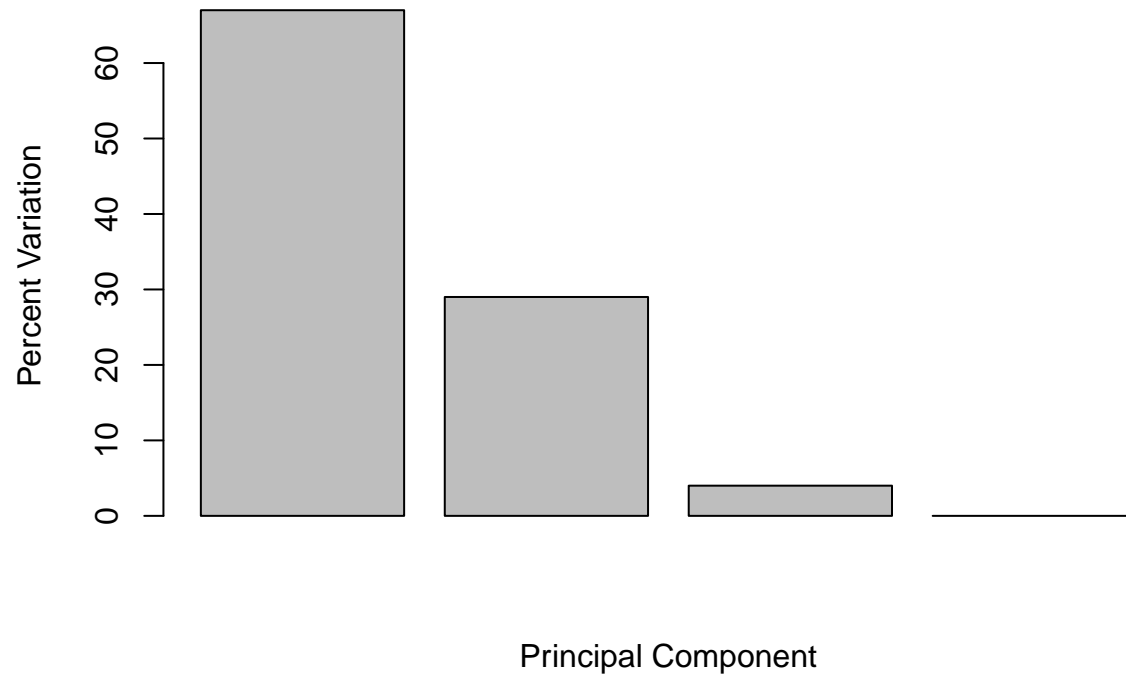
or the second row here...

```
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
```

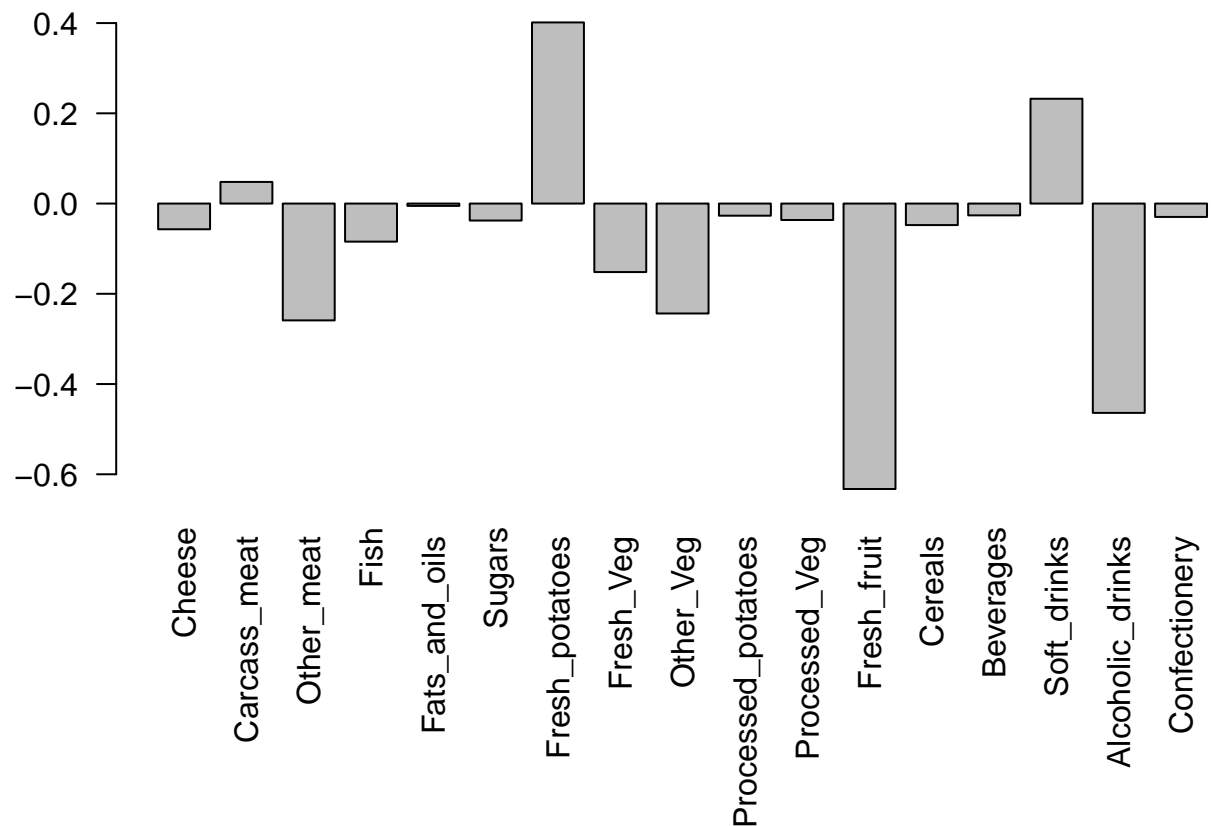
Plot of variances

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



Let's focus on PC1 as it accounts for >90% of variance

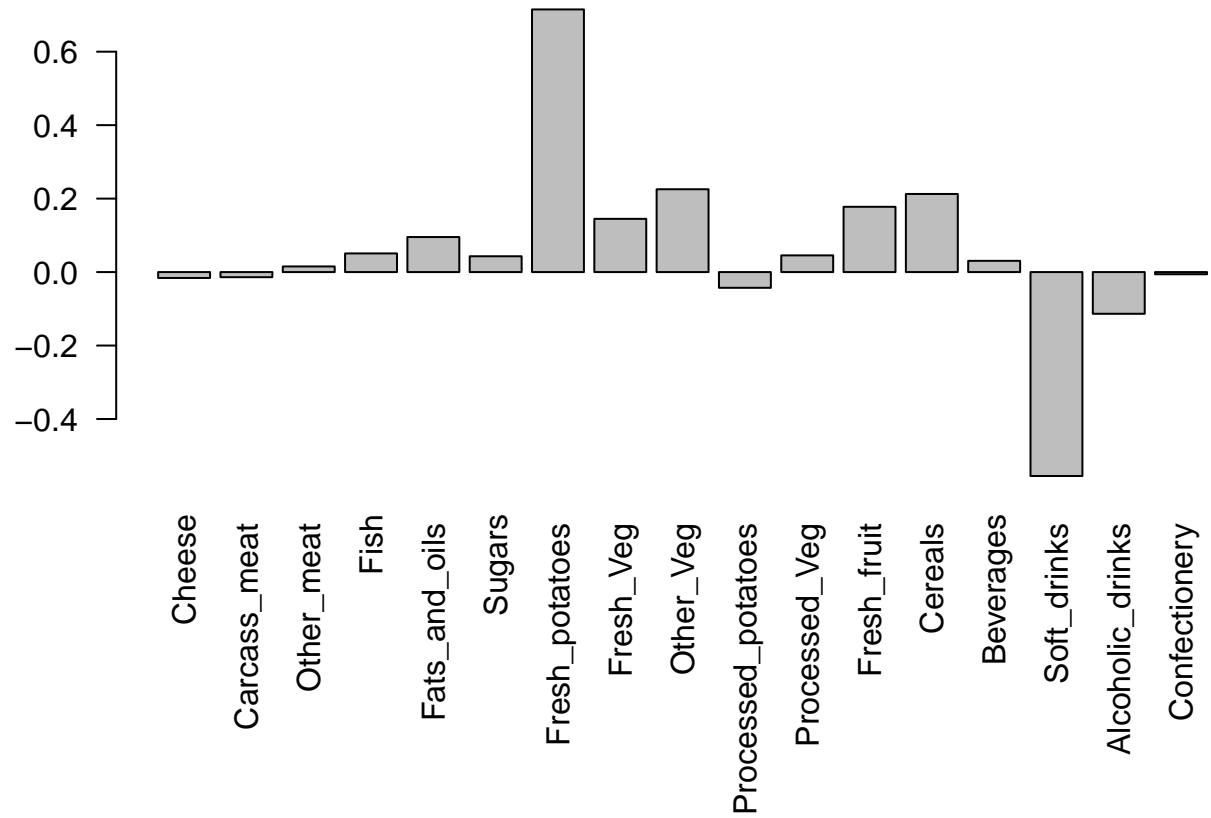
```
par(mar=c(10, 3, 0.35, 0))  
barplot(pca$rotation[, 1], las=2)
```



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

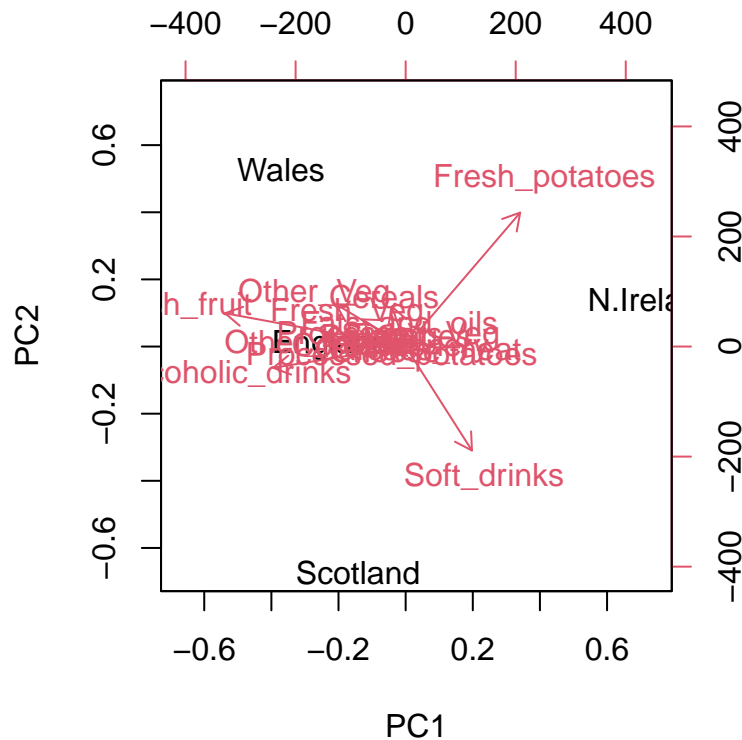
Fresh potatoes and soft drinks are the most prominent food groups that constitute variance in PC2. PC2 tells us that it captures approximately 29% of the variance in the original data-set and of that 29%, the majority of the variance comes from fresh potatoes and soft drinks.

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



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

```
biplot(pca)
```



## 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
```

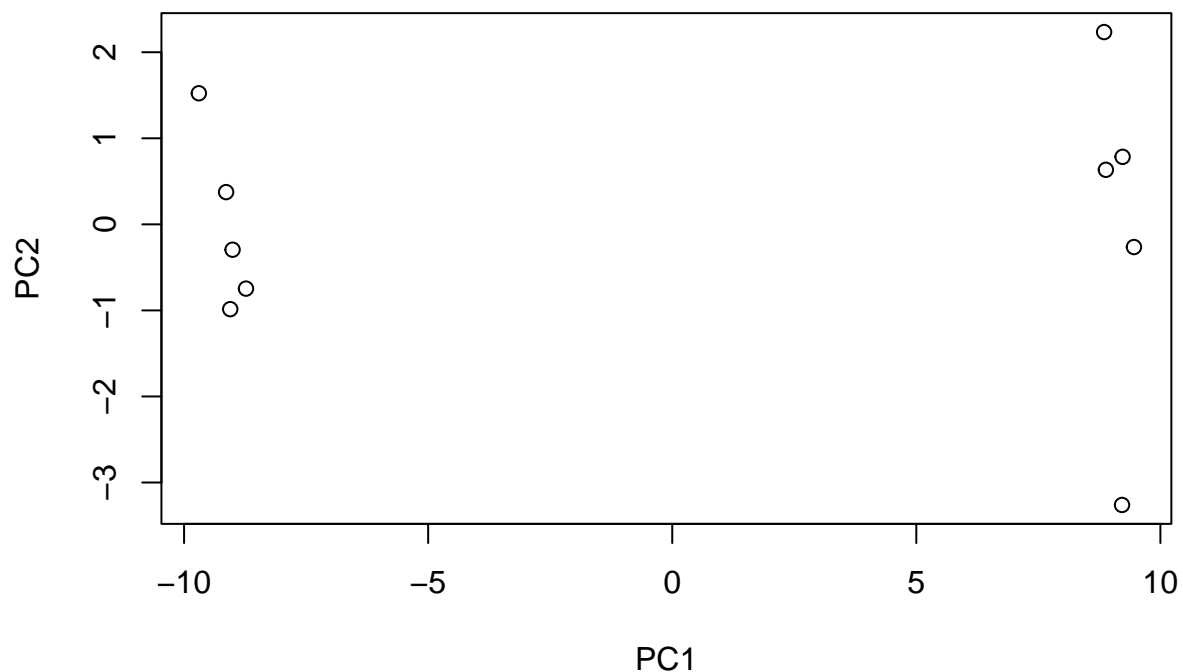
There are 100 genes and 10 samples.

Again we have to take the transpose of our data

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

Simple unpolished plot of pc1 and pc2

```
plot(pca$x[, 1], pca$x[, 2], xlab="PC1", ylab="PC2")
```



How much variation does each PC account for?

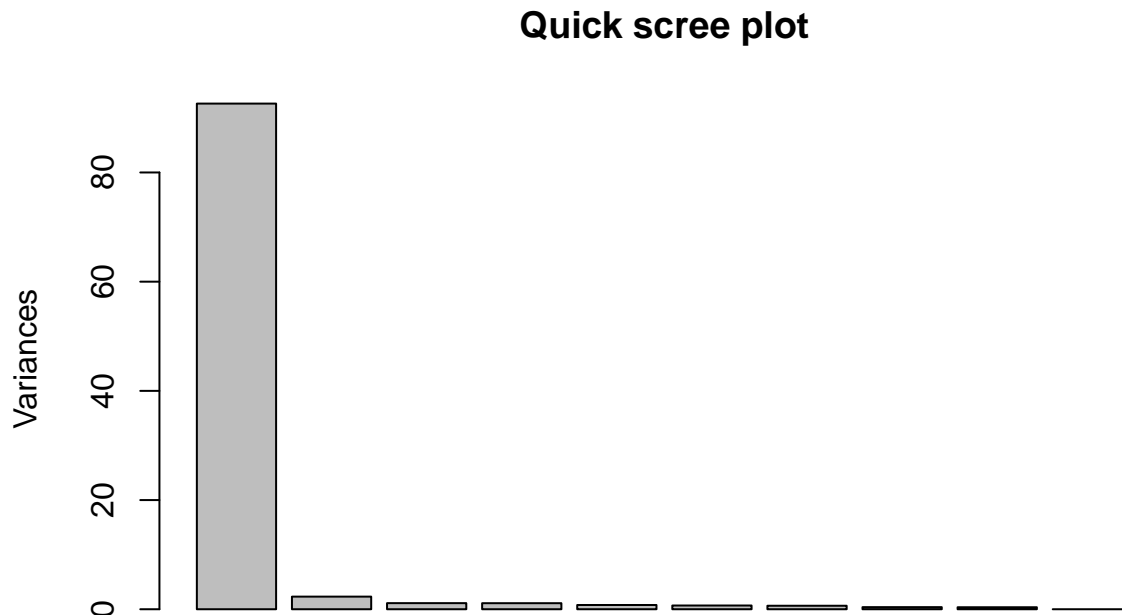
```
summary(pca)
```

```
## 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
```



Let's make a quick barplot summary

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



Variance captured per PC

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

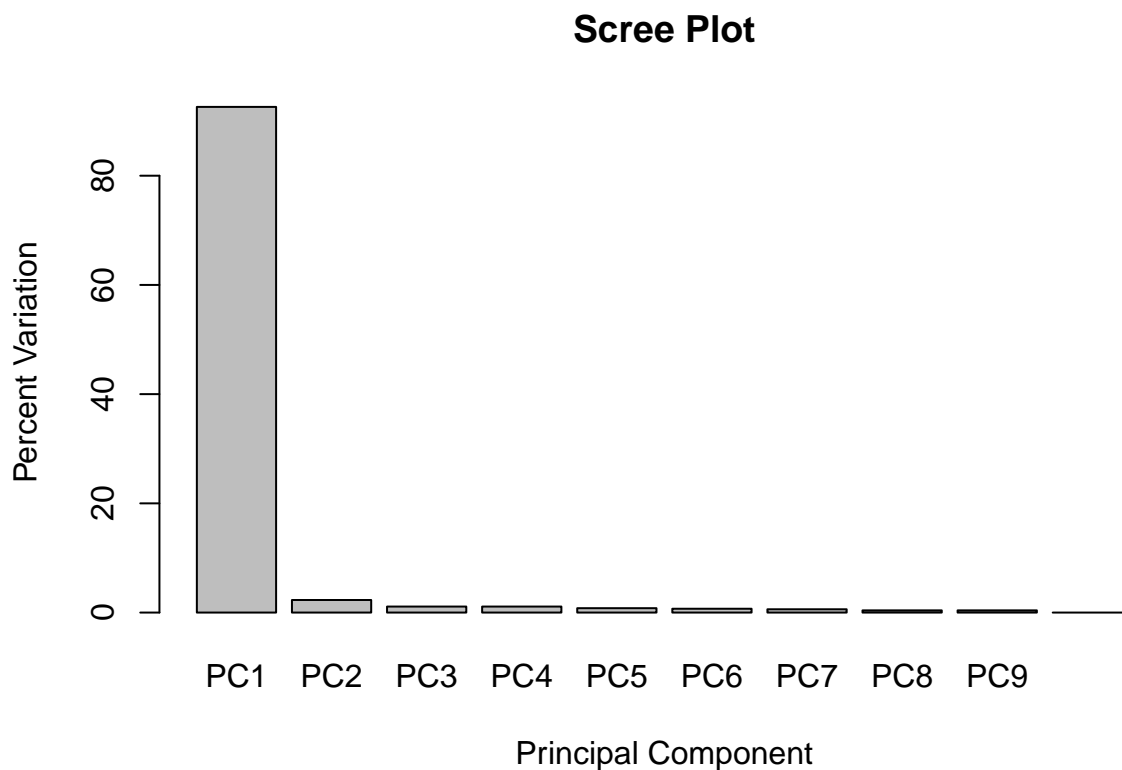
Percent variance is often more informative to look at

```
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
```

Generating our own scree plot

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

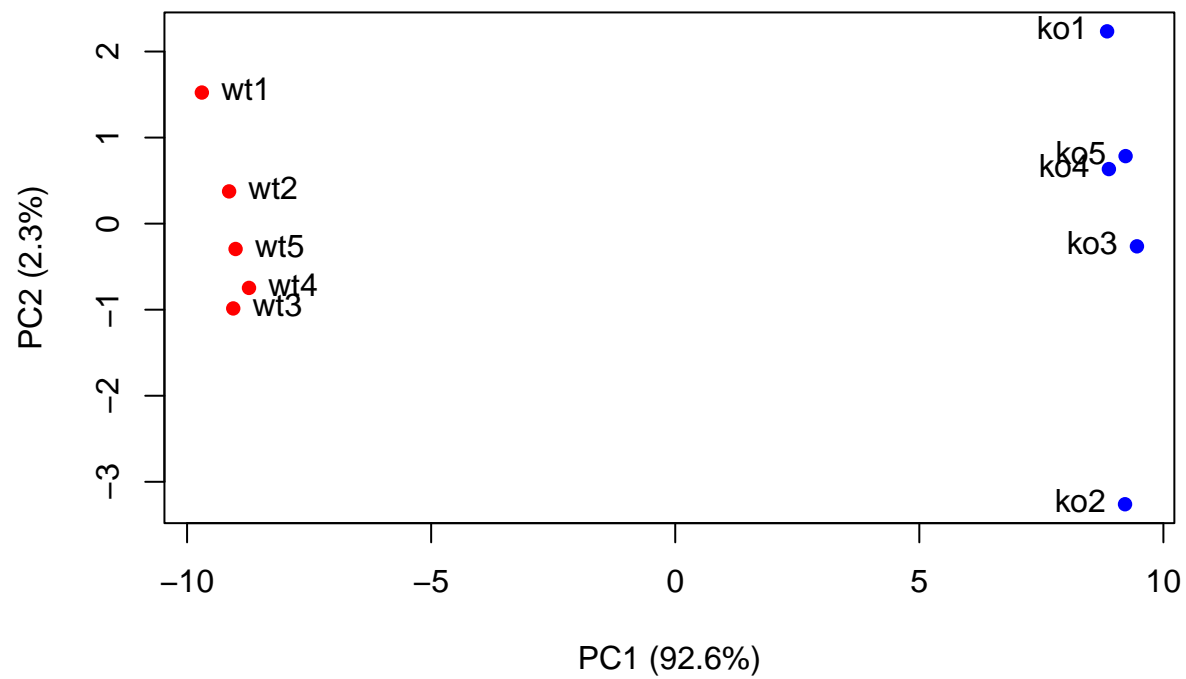


A vector of colors for wt and ko samples

```
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], "%)"))

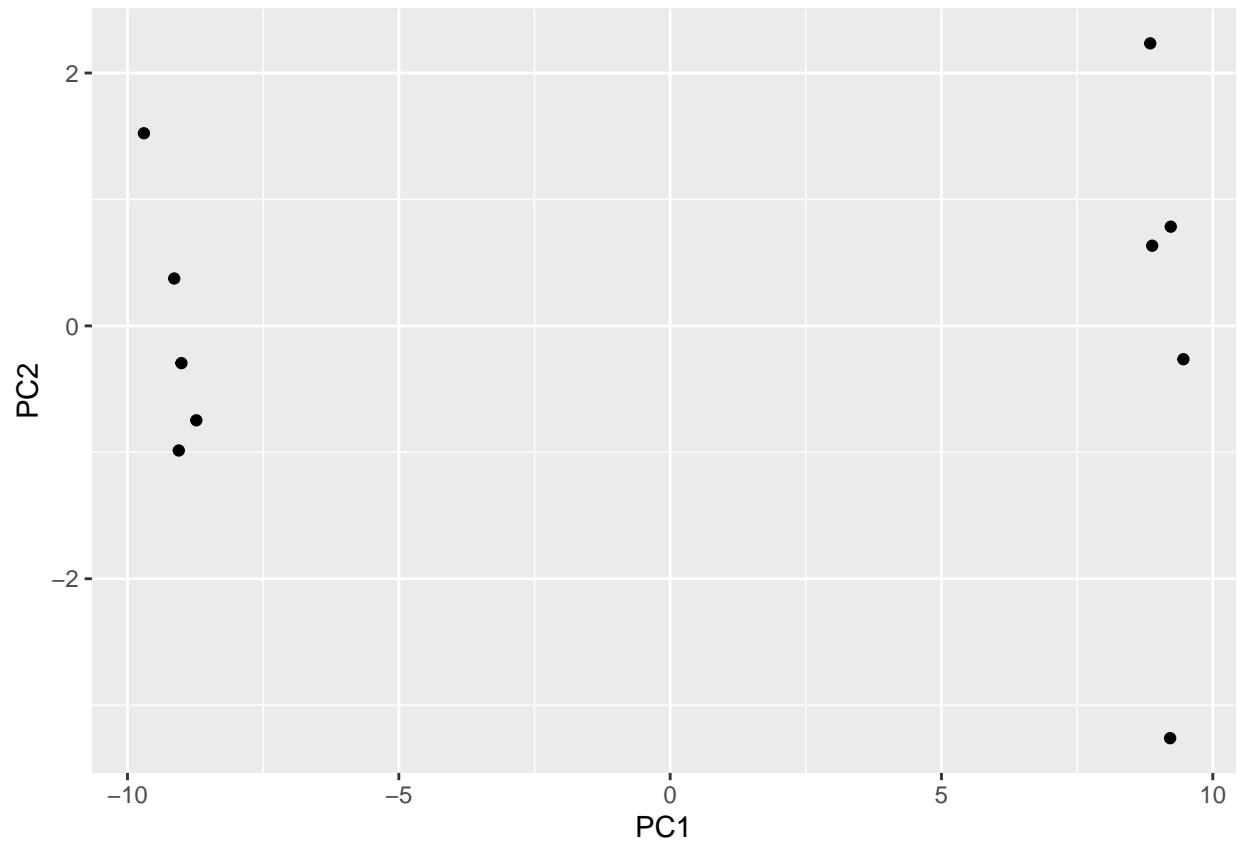
text(pca$x[,1], pca$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```



```
library(ggplot2)
df <- as.data.frame(pca$x)
```

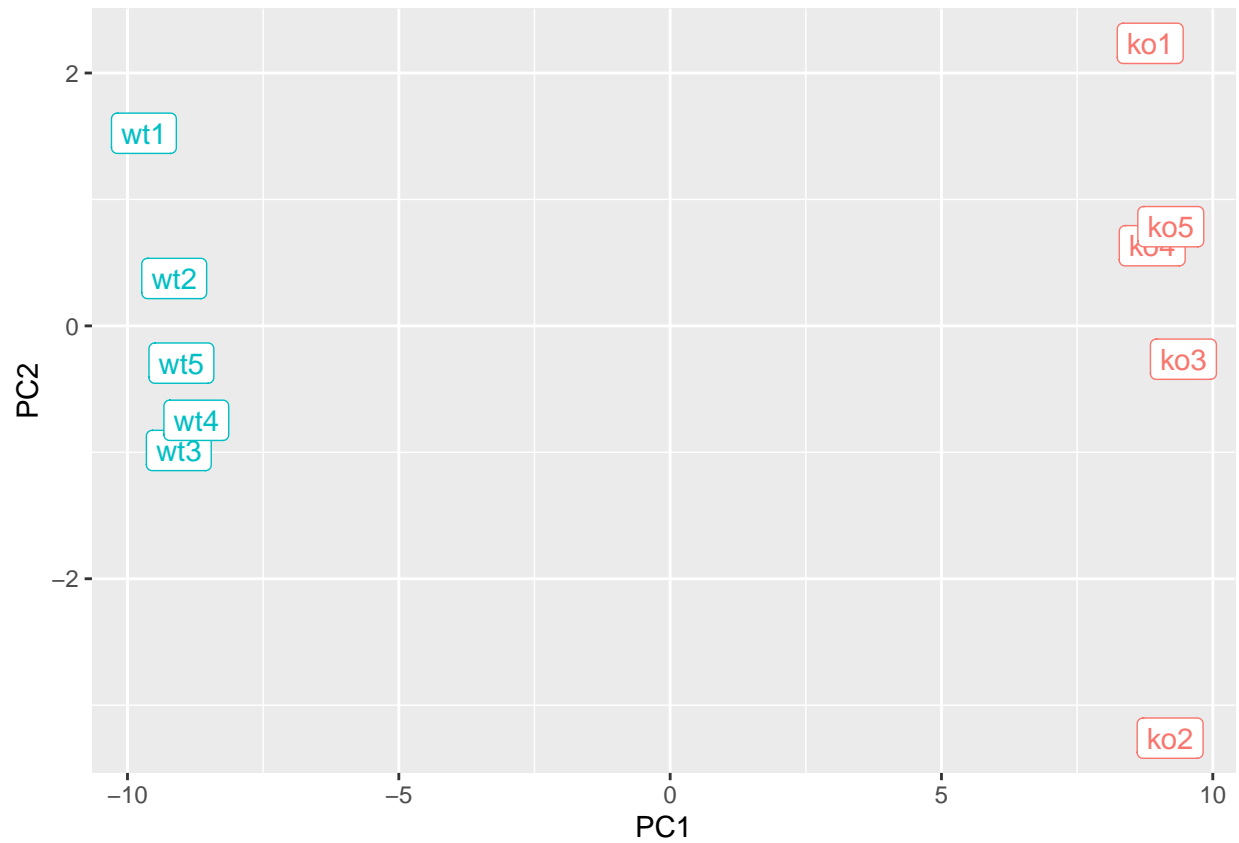
## Our first basic plot

```
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



Add a 'wt' and 'ko' "condition" column

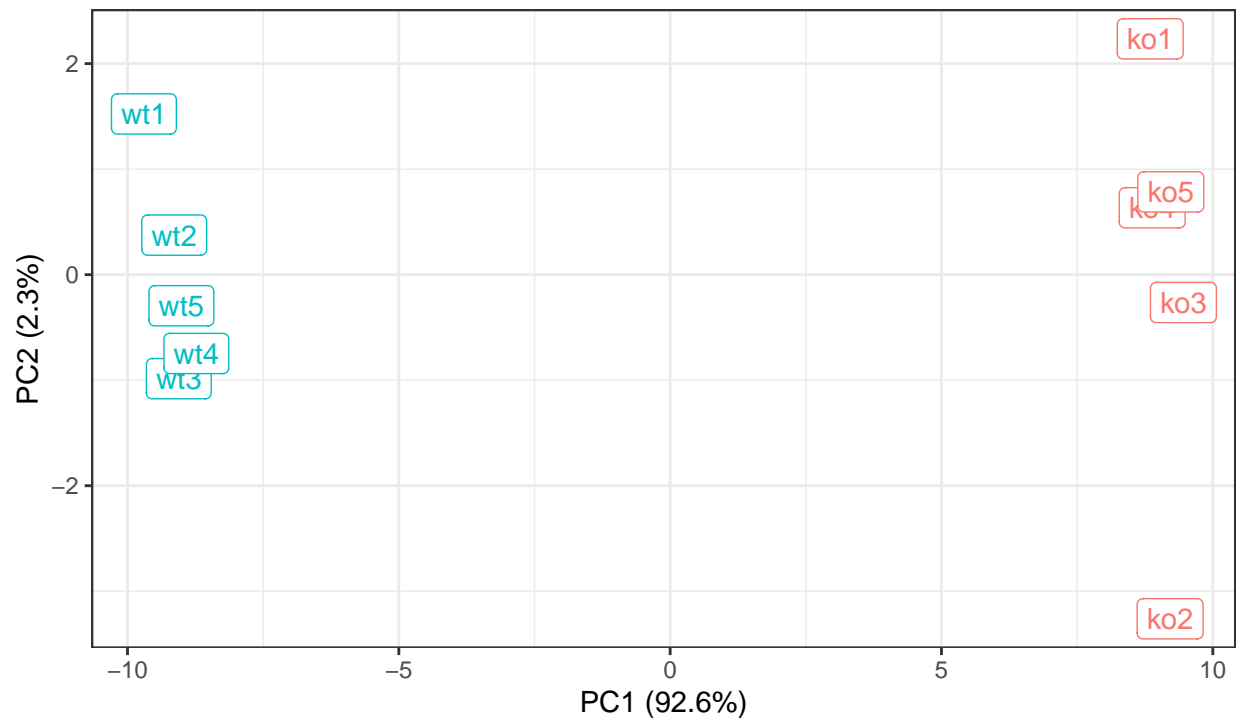
```
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)
p <- ggplot(df) +
  aes(PC1, PC2, label=samples, col=condition) +
  geom_label(show.legend = FALSE)
p
```



```
p + labs(title="PCA of RNASeq Data",
  subtitle = "PC1 clearly separates 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 clearly separates wild-type from knock-out samples



BIMM143 example data

```
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"
```