Class 7: Intro to Machine Learning

Xinyu Wen (A17115443)

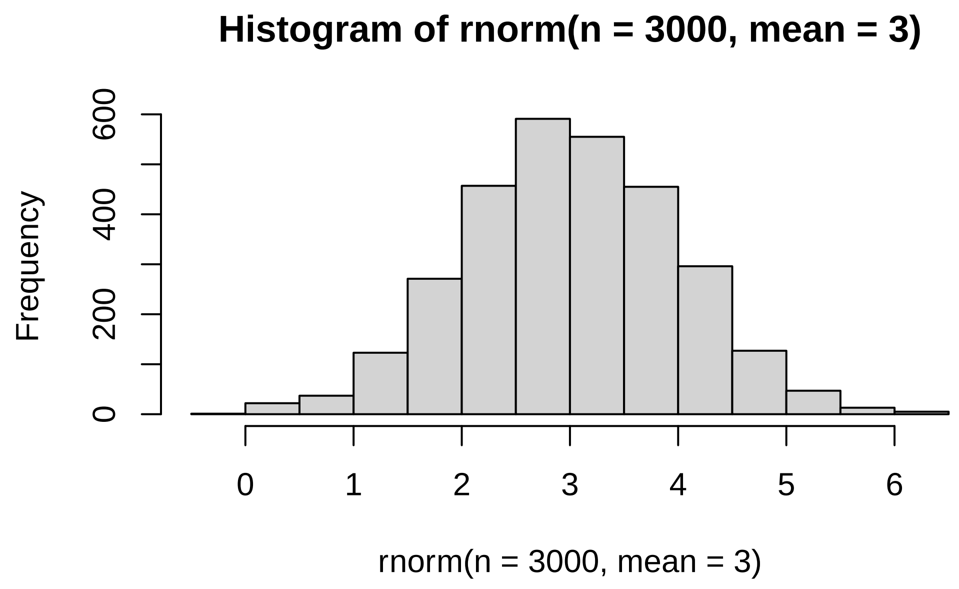
Table of contents

Today we will explore unsupervised machine learning methods including clustering and dimentionality reduction methods.

Let's start by making up some data (where we know there rae clear groups) that we can use to test out different clustering methods.

We can use the `rnorm()` for

```
hist(rnorm(n=3000, mean = 3))
```



Make data z with two "clusters"

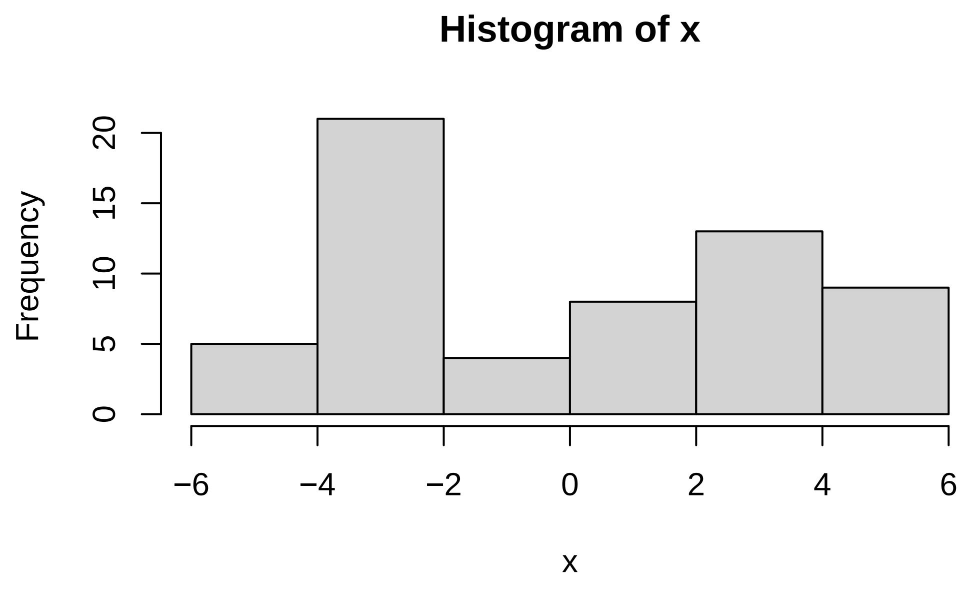
```r
rnorm(30, mean=-3)
```

```
 [1] -3.7433153 -4.3217792 -1.7935740 -4.0083938 -2.1423993 -4.8050819
 [7] -3.1227694 -3.8793063 -1.6400976 -2.4738805 -2.5137105 -3.2710059
[13] -4.3814283 -2.7573337 -3.8191915 -3.4313084 -3.9037321 -3.7413905
[19] -3.2917719 -2.1486427 -3.1573811 -3.2697133 -2.6017514 -4.7756139
[25] -1.5591695 -0.9643865 -2.1290850 -2.4372159 -2.9797937 -2.4223979
```

```r
rnorm(30, mean= +3)
```

```
 [1]  1.9656432  2.6165892  2.4576659  4.0839193  2.5460514  2.2198801
 [7]  2.6294766  3.9199817  2.4131573  1.7513112  3.7873296  1.8843090
[13]  3.7294562  2.7017269  3.1007470  4.8384023  3.7593797 -0.1232246
[19]  4.1205845  6.2365091  2.0469441  3.7537896  3.0279897  3.3793664
[25]  4.2383501  2.7450149  1.1914912  2.2843404  4.2715554  2.1194014
```

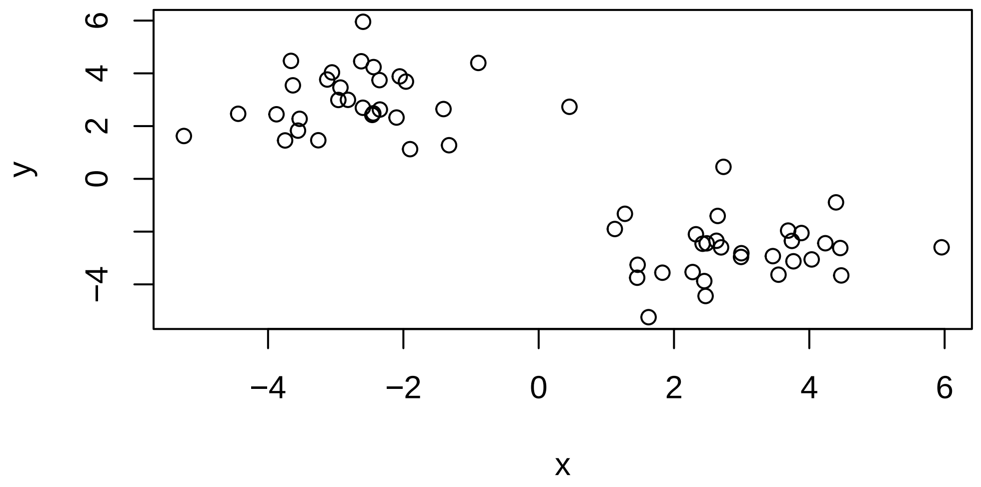
```r
x <- c(rnorm(30, mean=-3),
rnorm(30, mean= +3))

hist(x)
```



```r
x <- c(rnorm(30, mean=-3),
rnorm(30, mean= +3))

z <- cbind(x=x, y=rev(x))
head(z)
```

```
              x        y
[1,] -2.9295538 3.460984
[2,] -0.8919533 4.394508
[3,] -3.8755243 2.447696
[4,] -2.6230031 4.457529
[5,]  0.4548172 2.731386
[6,] -2.0539523 3.883956
```

```r
plot(z)
```



How big is `z`

```
nrow(z)
```

```
[1] 60
```

```
ncol(z)
```

```
[1] 2
```

K-means clustering

The main function in "base" R for K-means clustering is called `kmeans()`.

```
k <- kmeans(z, centers = 2)
k
```

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

Cluster means:
          x         y
```

```
1 -2.696343  2.983828
2  2.983828 -2.696343

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

Within cluster sum of squares by cluster:
[1] 72.73951 72.73951
 (between_SS / total_SS =  86.9 %)

Available components:

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

```r
attributes(k)
```

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

$class
[1] "kmeans"
```

> Q. How many points lie in each cluster? (aka. size of cluster?)

```r
k$size
```

```
[1] 30 30
```

> Q. What component of our results tells us about the cluster membership (i.e. which point likes in which cluster)?

```r
k$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
```

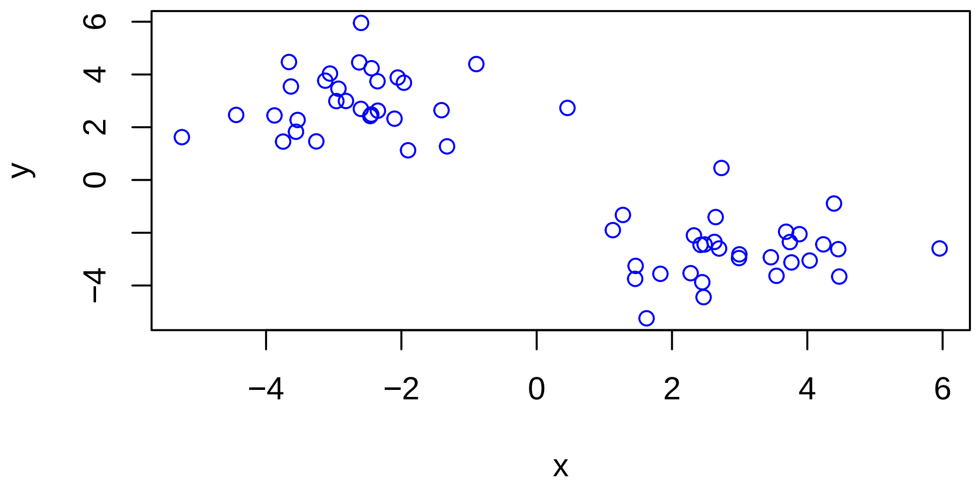
> Q. Center of each cluster?

```
k$centers
```

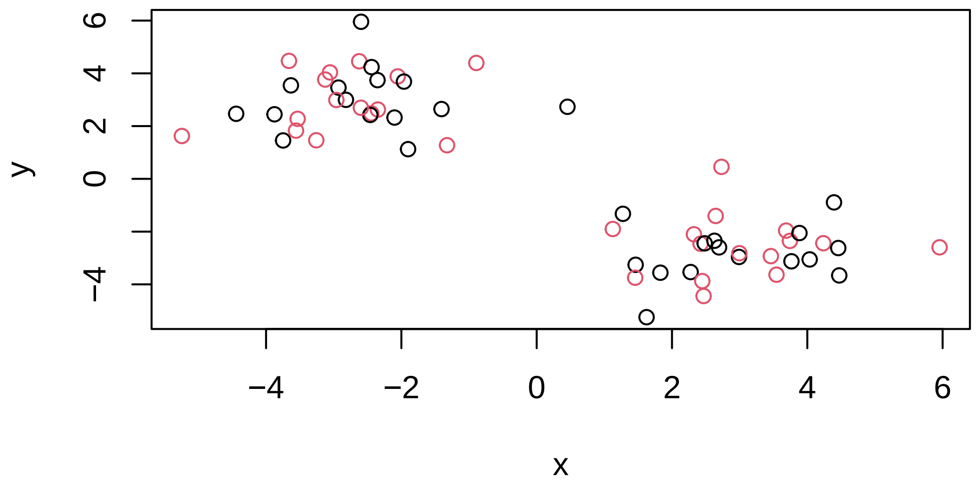
```
          x         y
1 -2.696343  2.983828
2  2.983828 -2.696343
```

> Q. Put this result info together and make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

```
plot(z, col = "blue")
```

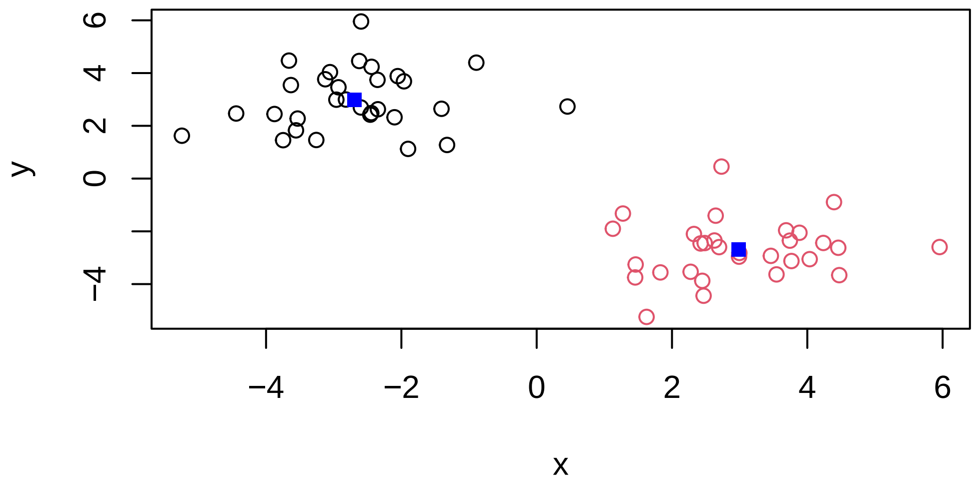


```
plot(z, col=c(1, 2))
```



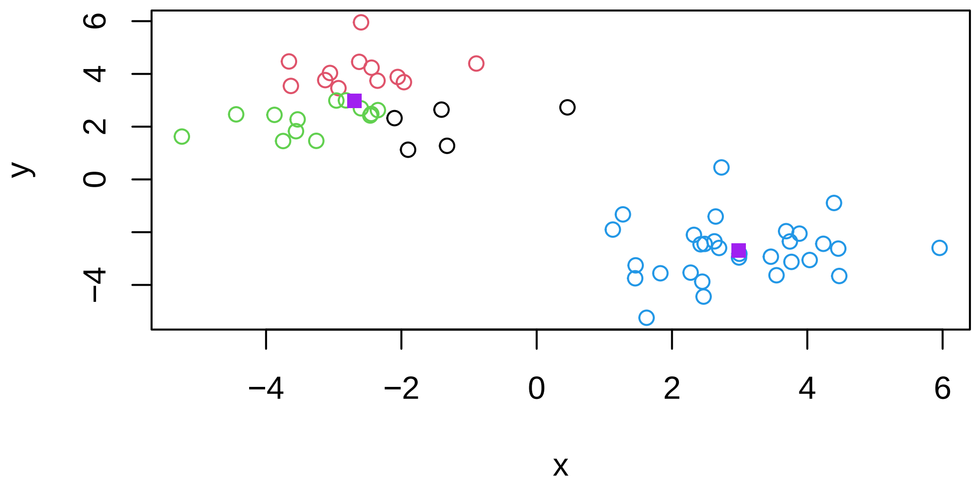
Plot colored by cluster membership:

```
plot(z, col=k$cluster)
points(k$centers, col="blue", pch = 15)
```



Q. Run kmeans on our input `z` and define 4 clusters making the same result vizualization plot as above (plot of z colored by cluster membership).

```r
k4 <- kmeans(z, center = 4)

plot(z, col=k4$cluster)
points(k$centers, col="purple", pch = 15)
```



Hierarchical Clustering

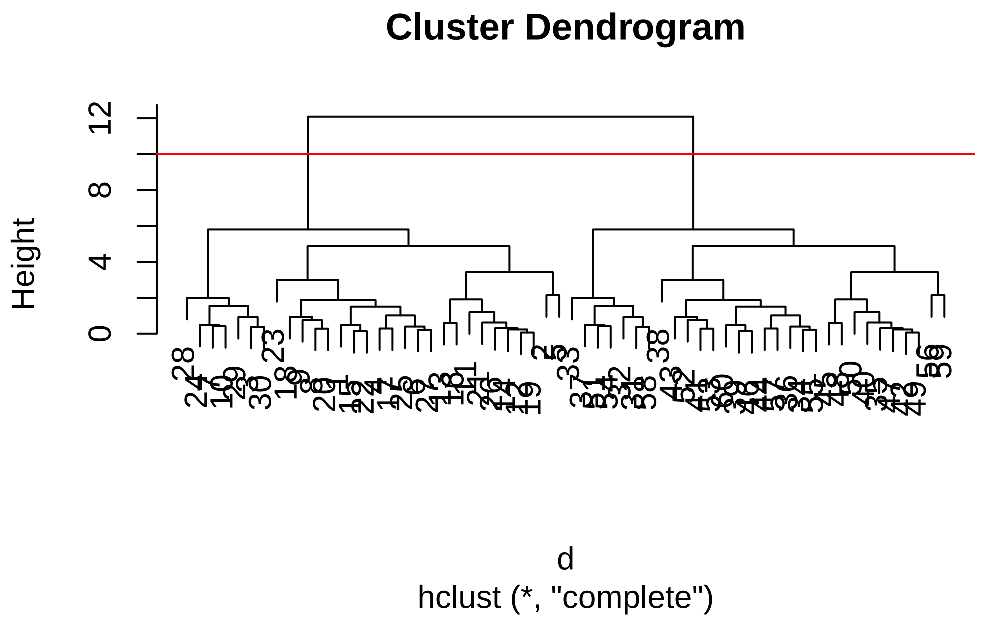
The main function in base R for this is called `hclust()`. It will take as input a distance matrix (key point is that you can't just give your "row" data as input. You have to first calculate a distance matrix from your data).

```r
d <- dist(z)
hc <- hclust(d)
hc
```

```

Call:
hclust(d = d)
```

```
Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

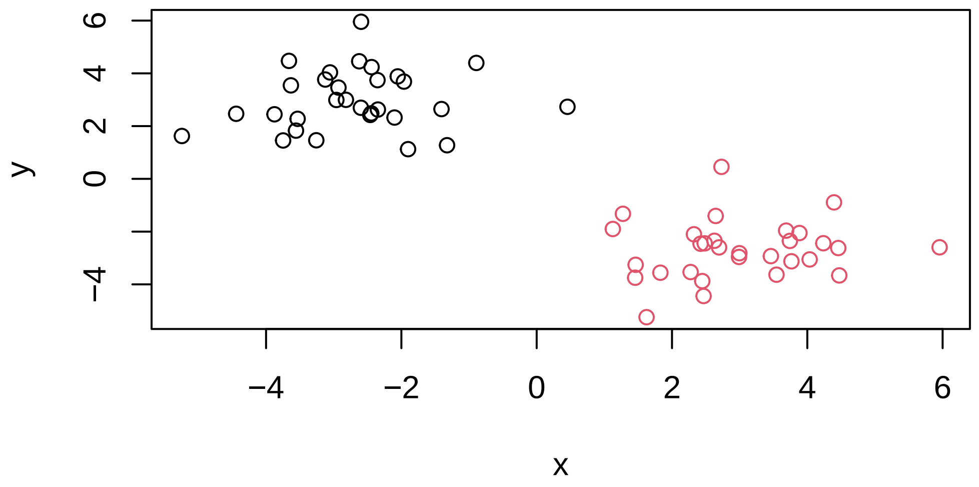
```r
plot(hc)
abline(h=10, col = "red")
```



Once I inspect the "tree" of dendrogram, I can "cut" the tree to yield my groupings or clusters. The function to this is called `cutree()`.

```r
grps <- cutree(hc, h=10)
```

```r
plot(z, col = grps)
```



1. PCA of UK food data

Hands on with Principal Component Analysis (PCA)

Let's examine a 17-dimensional data containing details of food consumption in the UK. Are these countries different? How?

Data import

```r
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names =1) #Q2, first way
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
Fresh_potatoes          720   874       566       1033
Fresh_Veg               253   265       171        143
Other_Veg               488   570       418        355
Processed_potatoes      198   203       220        187
Processed_Veg           360   365       337        334
Fresh_fruit            1102  1137       957        674
Cereals                1472  1582      1462       1494
Beverages                57    73        53         47
Soft_drinks            1374  1256      1572       1506
Alcoholic_drinks        375   475       458        135
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 examine this?

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 4
```

```r
dim(x)
```

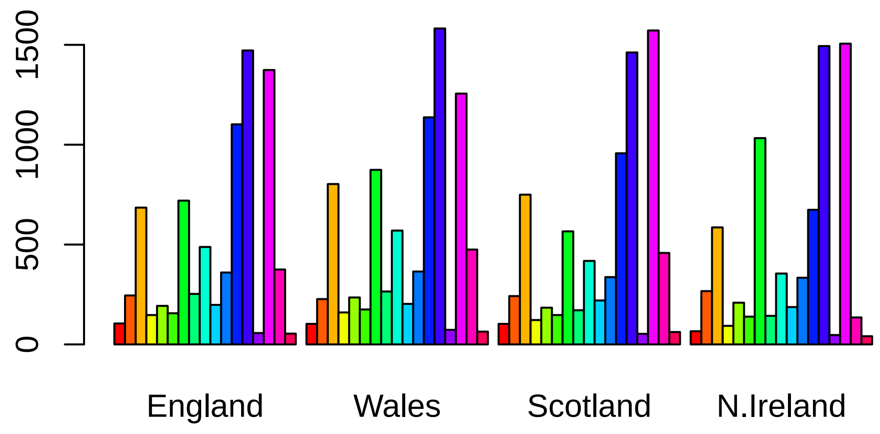
```
[1] 17  4
```

> 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?

```r
#The second way
#rownames(x) <- x[,1]
#x <- x[,-1]
#head(x)
```

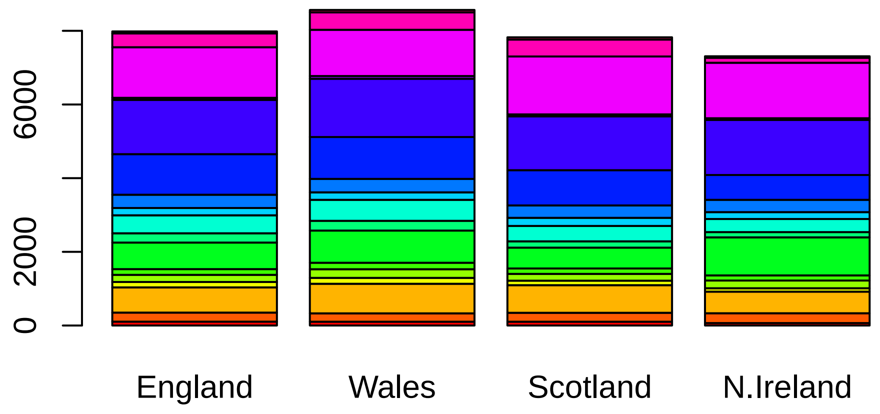
I like the first way better, because it eliminates first column and read the file in one line. However, if I do not know which column of the file I should eliminate, it would be better to print the original table first, then use the second way to eliminate the column.

```r
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?

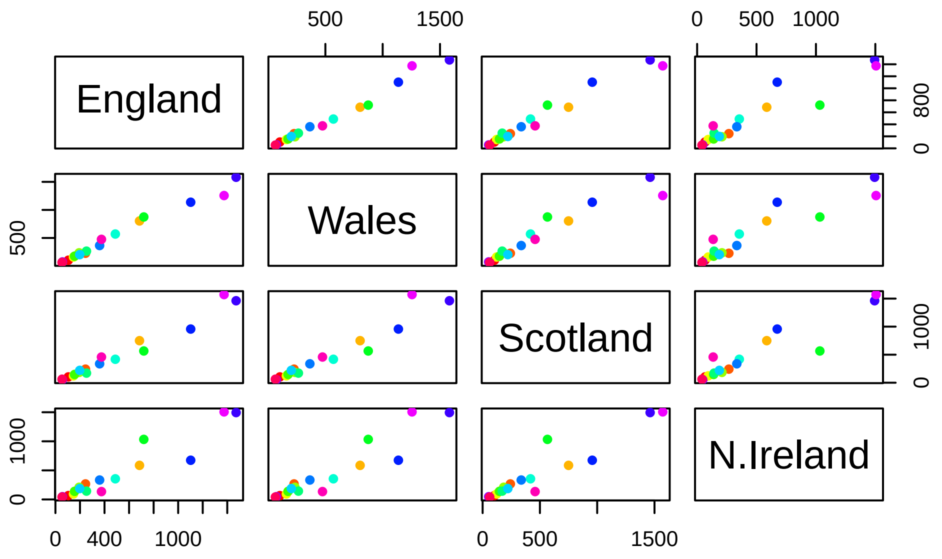
```r
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Set beside as `False`.

> 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(nrow(x)), pch =16)
```



If a point lies on the diagonal of a plot, it means the two countries have similar comsumption of that food.

> 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 is that N. Ireland consumes much more fresh potatoes.

Looking at these types of "pairwise plots" can be helpful but it does not scale well. There must be a better way……

PCA to the rescue!

The main function for PCA in base R is called `prcomp()`. This function wants the transpose of our input data - ie the important foods in as columns and the countries as rows.

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 2.921e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

Let's see what is in our PCA result object `pca`.

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

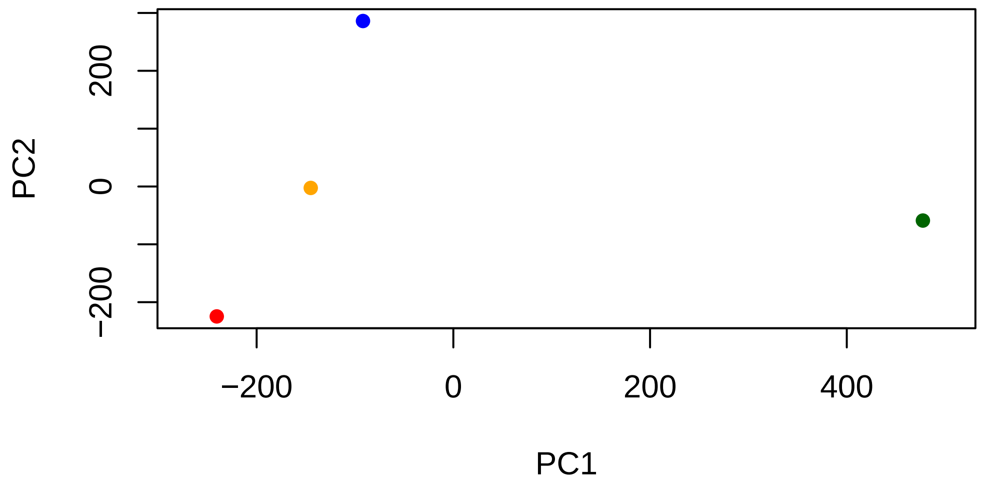
```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -9.152022e-15
Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

The `pca$x` result object is where we will focus first as this details how the countries are. related to each other in terms of our new "axis" (aka. "PCs", "eigenvectors", etc.)

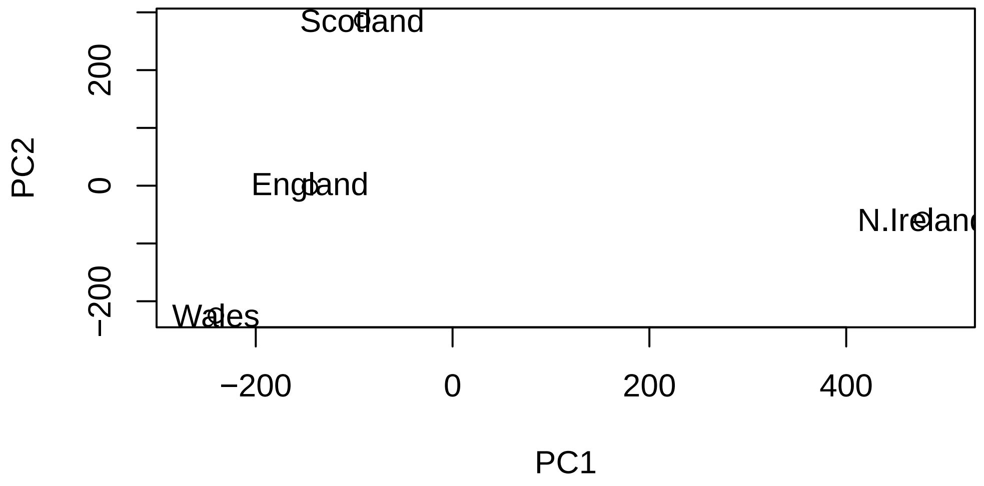
> Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

Plot PC1 vs PC2

```
plot(pca$x[,1], pca$x[,2], col=c("orange", "red", "blue", "darkgreen"), pch =16, xlab="PC1",
```

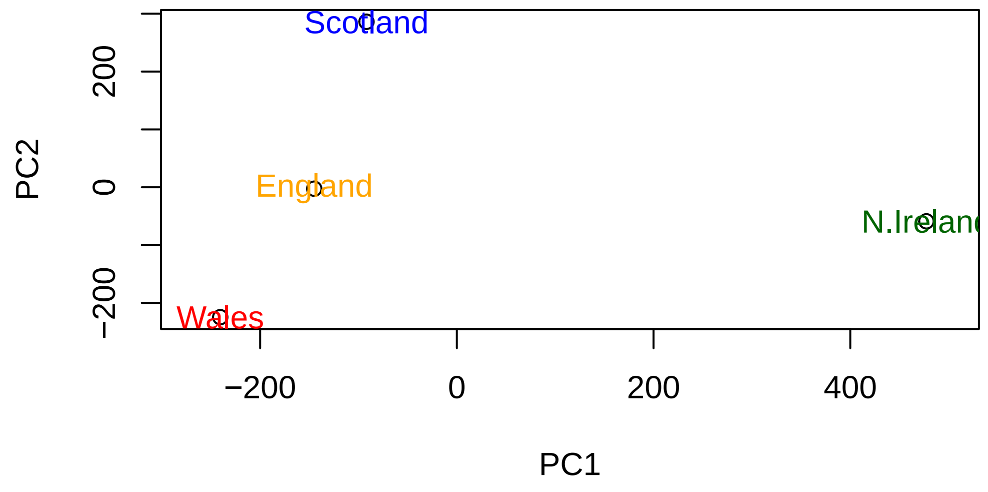


```r
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```



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.

```r
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", "darkgreen"))
```



We can look at the so called PC "loadings" result object to see how the original foods contribute to our new PCs. (ie, how the original variables contribute to our new better PC variable)

```r
pca$rotation[,1]
```

```
             Cheese        Carcass_meat          Other_meat                Fish 
       -0.056955380         0.047927628        -0.258916658        -0.084414983 
     Fats_and_oils              Sugars     Fresh_potatoes          Fresh_Veg 
       -0.005193623        -0.037620983         0.401402060        -0.151849942 
         Other_Veg  Processed_potatoes      Processed_Veg         Fresh_fruit 
       -0.243593729        -0.026886233        -0.036488269        -0.632640898 
           Cereals           Beverages         Soft_drinks   Alcoholic_drinks 
       -0.047702858        -0.026187756         0.232244140        -0.463968168 
     Confectionery 
       -0.029650201 
```

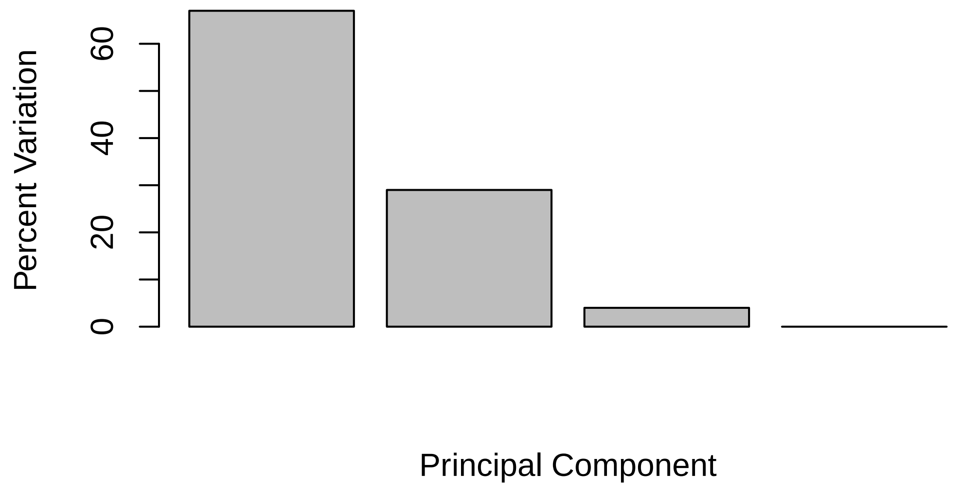
```r
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

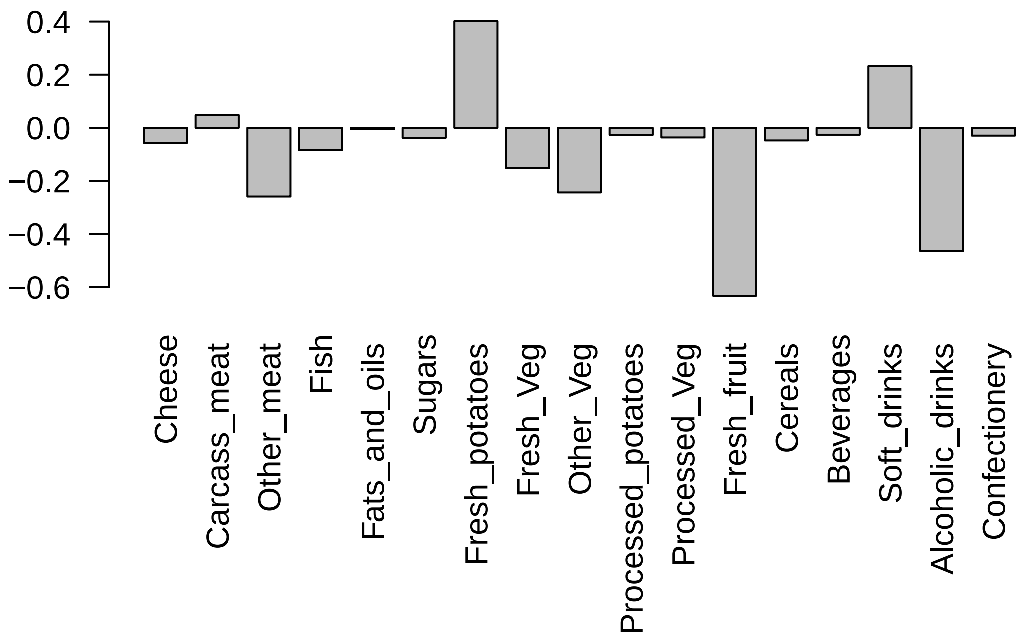
```
## or the second row here...
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 2.921348e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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

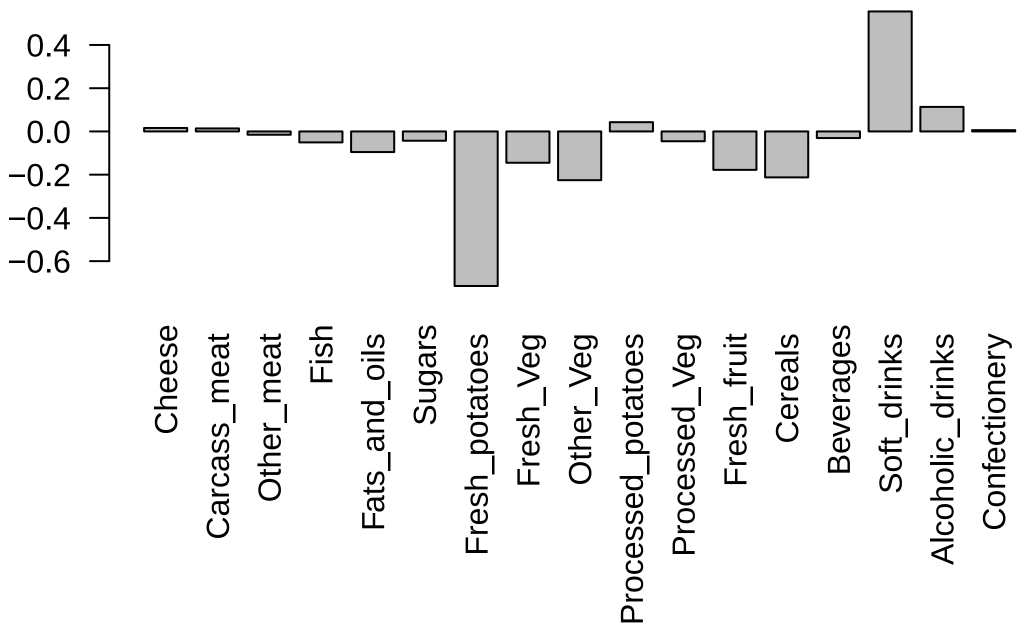


```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

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



Soft drinks and fresh potatoes are the two prominent food groups. PC2 tells us that the consumption of fresh potatoes and soft drinks have opposite trends. If fresh potatoes are larges consumed, soft drinks are not. And vise versa.

2. 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?

```r
genes <- nrow(rna.data)
genes
```

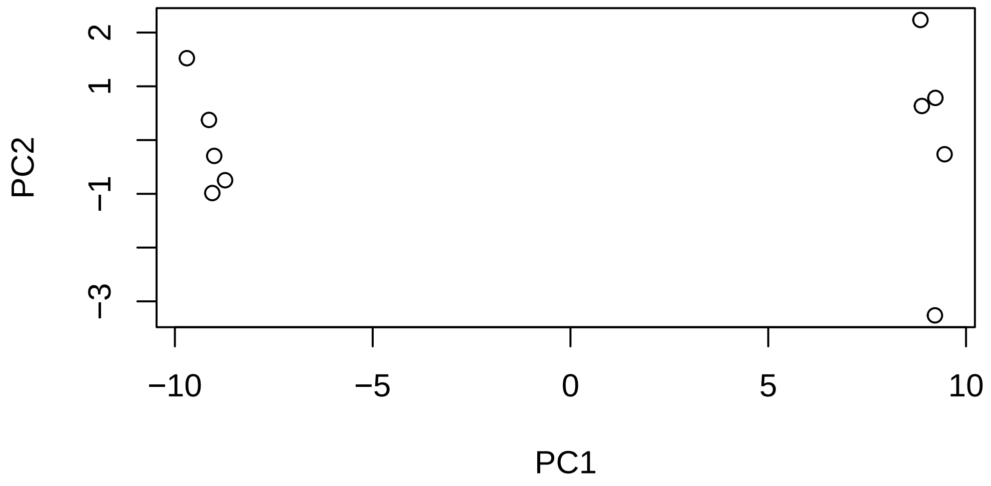
```
[1] 100
```

```r
samples <- ncol(rna.data)
samples
```

```
[1] 10
```

There are 100 genes and 10 samples.

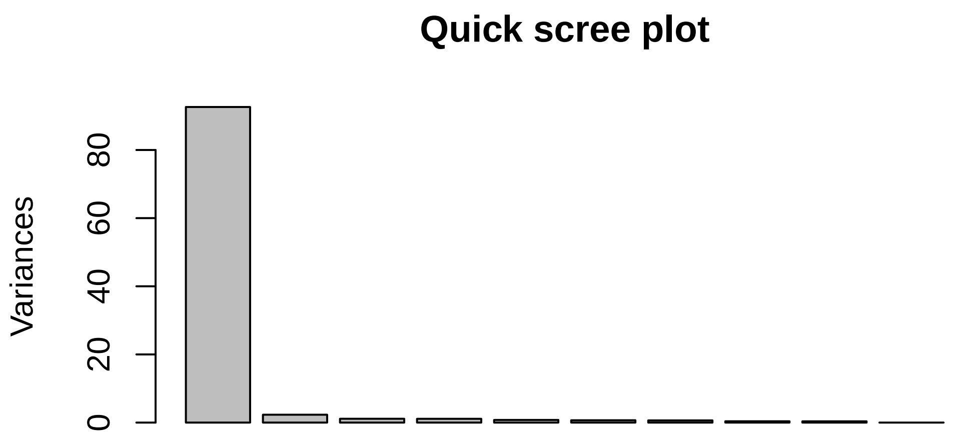
```r
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

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



```r
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 3.345e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

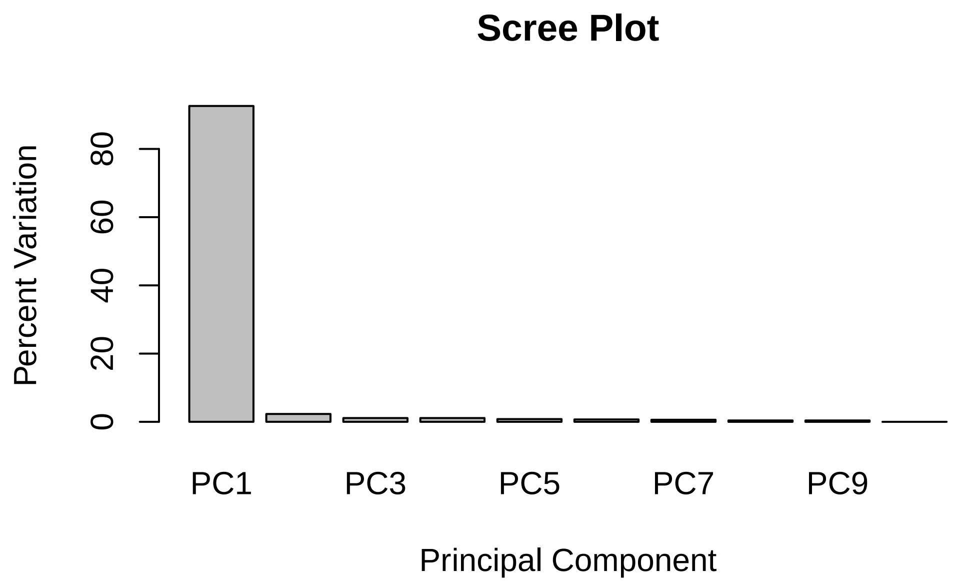
```
plot(pca, main="Quick scree plot")
```



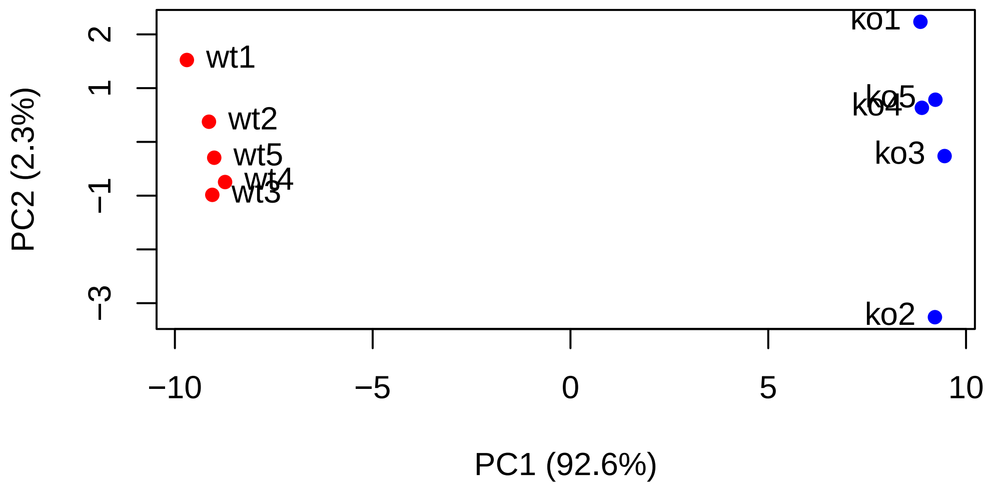
```
## Variance captured per PC
pca.var <- pca$sdev^2

## 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
```

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

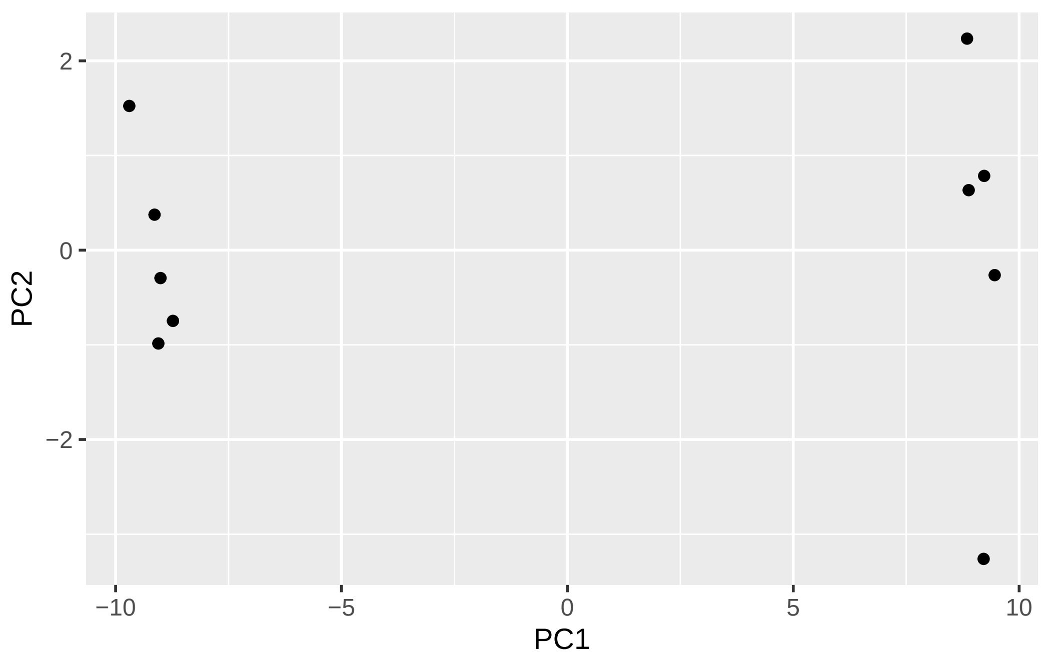


```r
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```

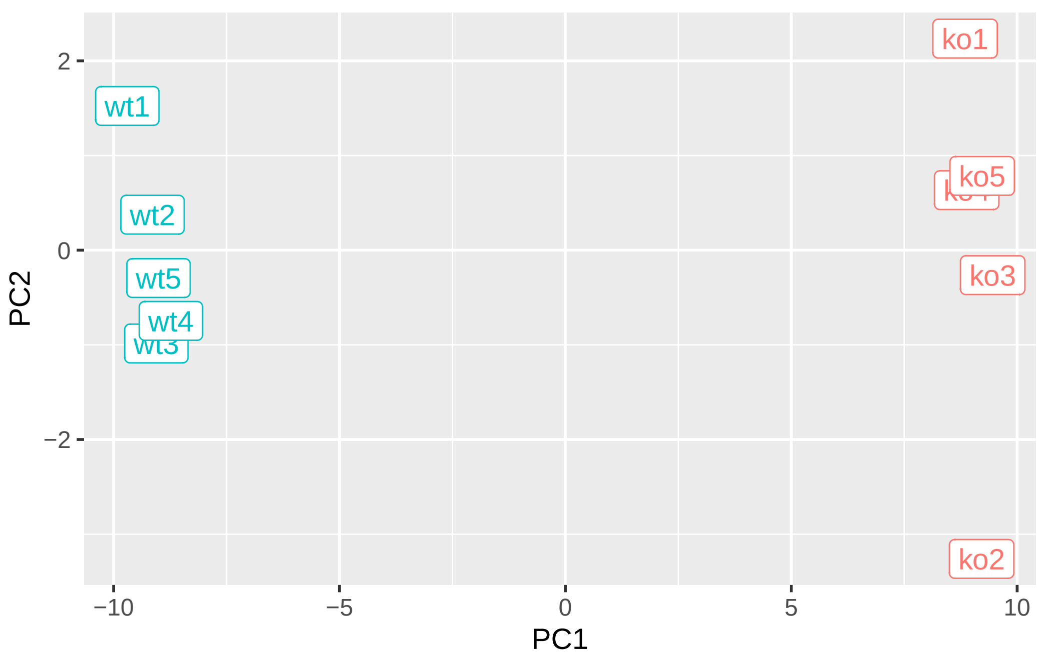


ggplot

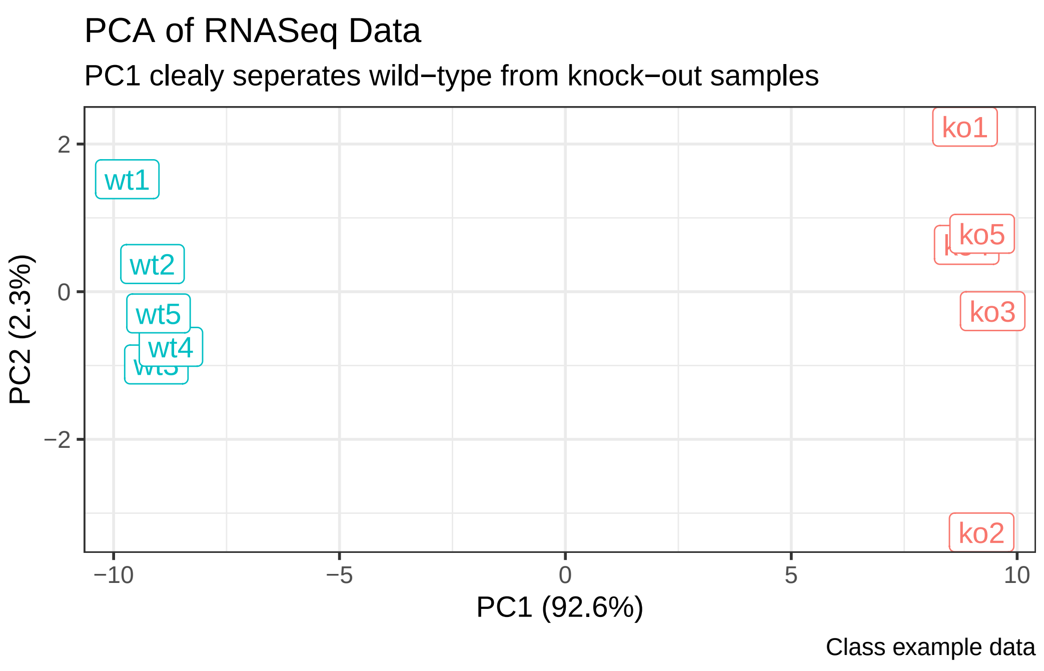
```
library(ggplot2)

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 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()
```



Optional: Gene loadings

```
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" 
```