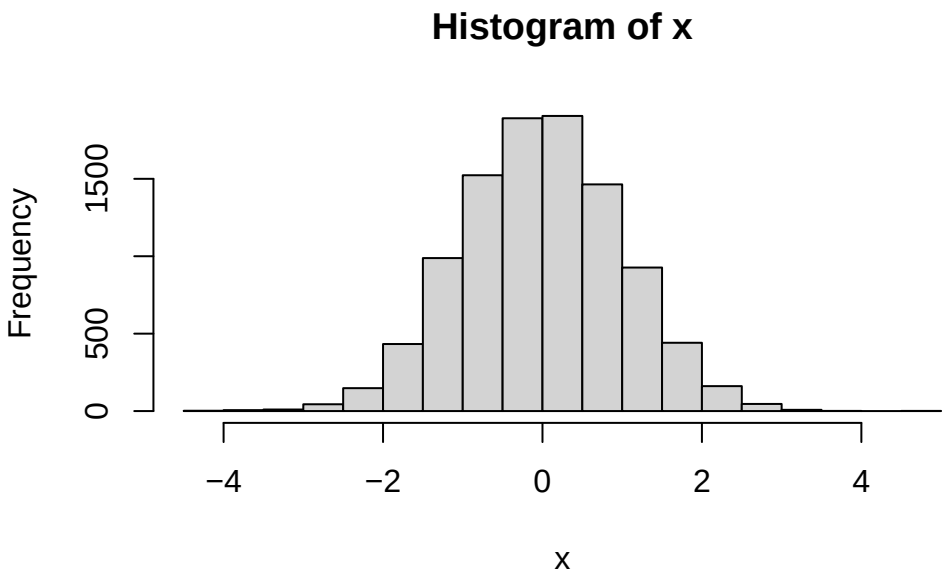
Class 07

Varun Durai

K-means clustering

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

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

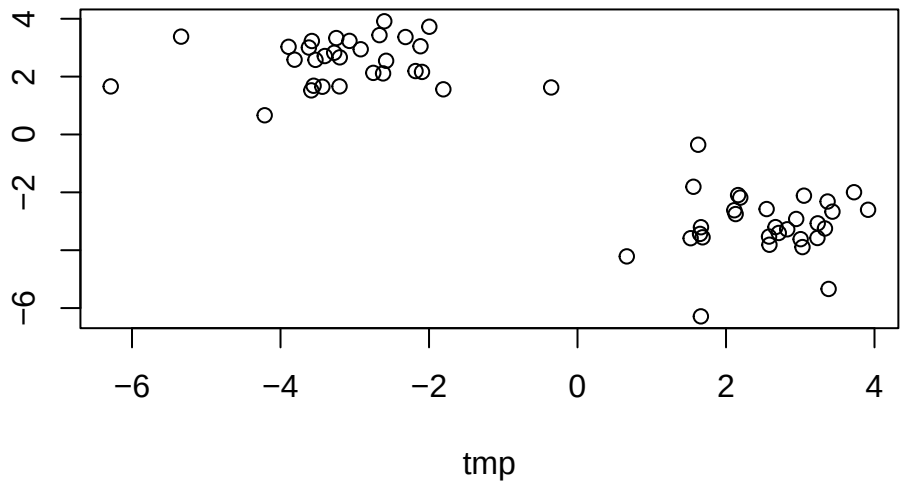


Let's make some numbers centered on -3

```
m <- c("a","b","c")
rev(m)
```

```
[1] "c" "b" "a"
```

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



Now let's see how `kmeans()` 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:
        tmp
1 -3.108403  2.539625
2  2.539625 -3.108403

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 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] 52.01096 52.01096
 (between_SS / total_SS =  90.2 %)

Available components:

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

```
km$centers
```

```
        tmp
1 -3.108403  2.539625
2  2.539625 -3.108403
```

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?

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 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
```

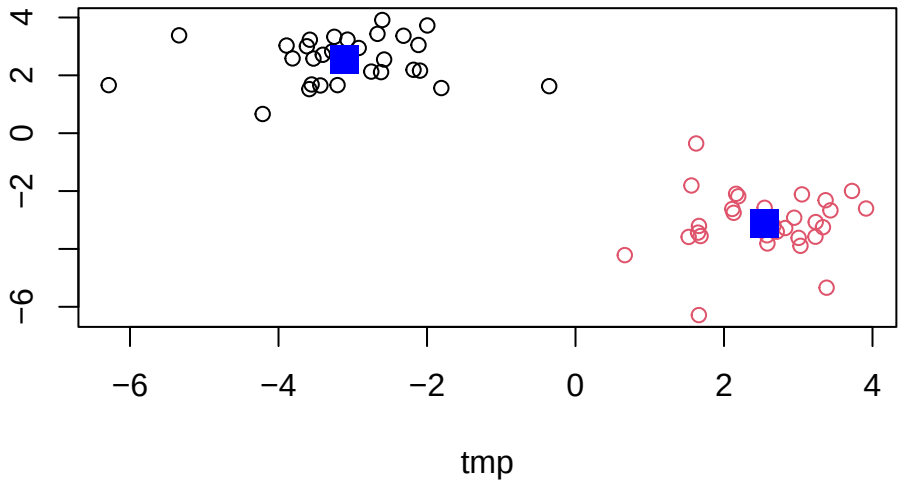
Cluster center:

```
km$centers
```

```
        tmp
1 -3.108403  2.539625
2  2.539625 -3.108403
```

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)
```



Hierarchichal Clustering

The `hclust()` function in R performs Hierarchical clustering.

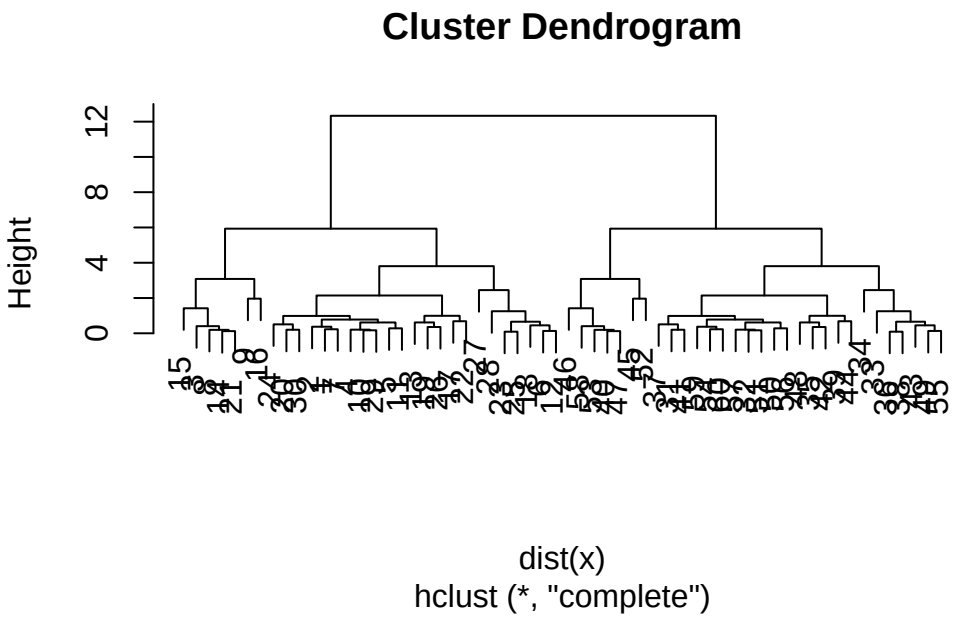
The `hclus()` function requires an input distance matrix, whcihc I can get from the `dist()` function.

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

```
Call:
hclust(d = dist(x))

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

Here is a plot() method for hclust objects...

```
plot(hc)
```



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

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

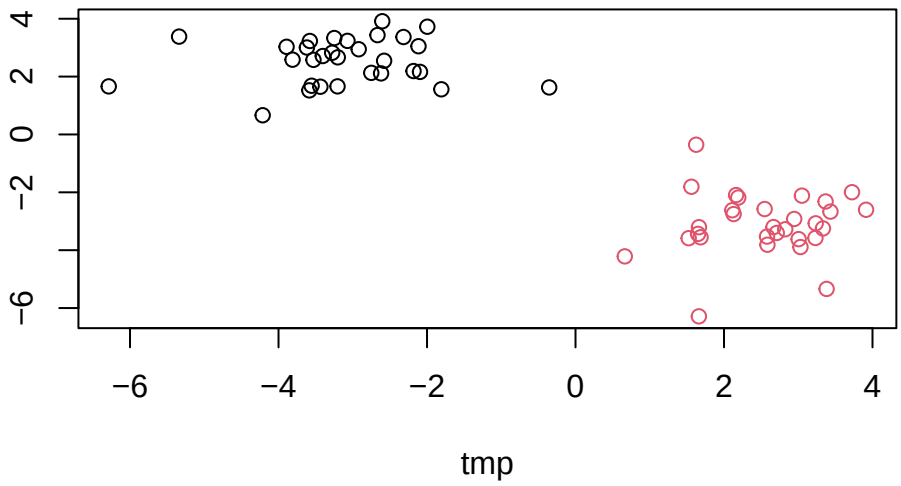
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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)

```
url <- "https://tinyurl.com/UK-foods"
ukFood <- read.csv(url)
ukFood
```

```
                    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?

```
dim(ukFood)
```

```
[1] 17  5
```

Running the code block above using the `dim()` function shows us that there are 17 rows and 5 columns

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?

```
rownames(ukFood) <- ukFood[,1]
ukFood
```

```
                                  X England Wales Scotland N.Ireland
Cheese                         Cheese     105   103      103        66
Carcass_meat             Carcass_meat     245   227      242       267
Other_meat                 Other_meat     685   803      750       586
Fish                             Fish     147   160      122        93
Fats_and_oils           Fats_and_oils     193   235      184       209
Sugars                         Sugars     156   175      147       139
Fresh_potatoes         Fresh_potatoes     720   874      566      1033
Fresh_Veg                   Fresh_Veg     253   265      171       143
Other_Veg                   Other_Veg     488   570      418       355
Processed_potatoes Processed_potatoes     198   203      220       187
Processed_Veg           Processed_Veg     360   365      337       334
Fresh_fruit               Fresh_fruit    1102  1137      957       674
Cereals                       Cereals    1472  1582     1462      1494
Beverages                   Beverages      57    73       53        47
Soft_drinks               Soft_drinks    1374  1256     1572      1506
Alcoholic_drinks     Alcoholic_drinks     375   475      458       135
Confectionery           Confectionery      54    64       62        41
```

The above method is one way of ensuring that the first column is set to the name of the rows of the dataframe. Another method is setting the row name while reading the csv file, shown below:

```
ukFood2 <- read.csv(url, row.names = 1)
ukFood2
```

```
                  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
```

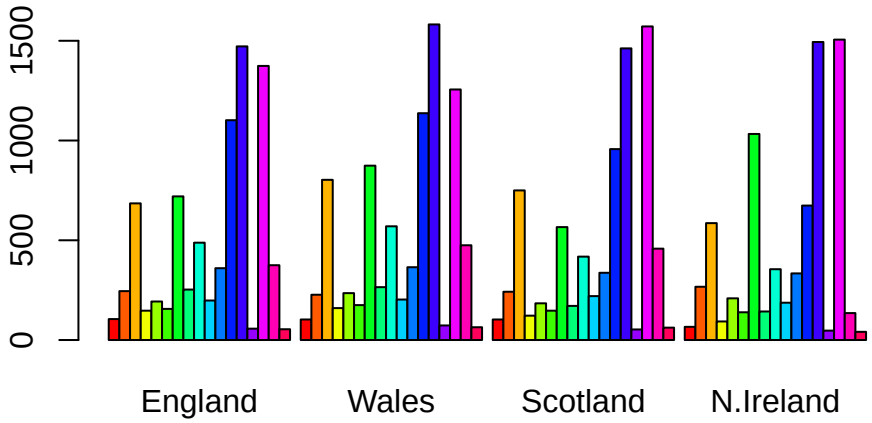
As we can see, the structure of ukFood and ukFood are the same, as both methods work to ensure that the first column is set to the name of the rows.

Specifying an argument in the `read.csv()` function saves time while trying to set a certain column as the names column, and so it is generally preferred. However, if there were a scenario where we would want a numbered column for our data, the first method would be useful for removing it later.

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

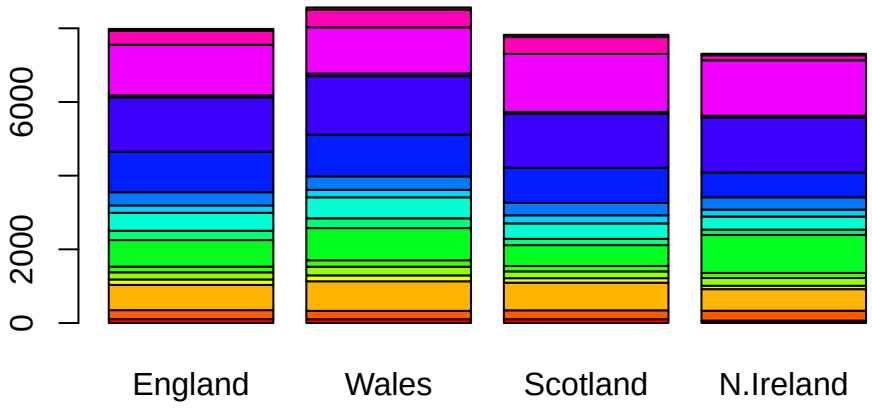
The code below is the original `barplot()` function:

```
barplot(as.matrix(ukFood2), beside=T, col=rainbow(nrow(ukFood2)))
```



Setting the `beside` argument to false or leaving it out will result in our required stacked plot.

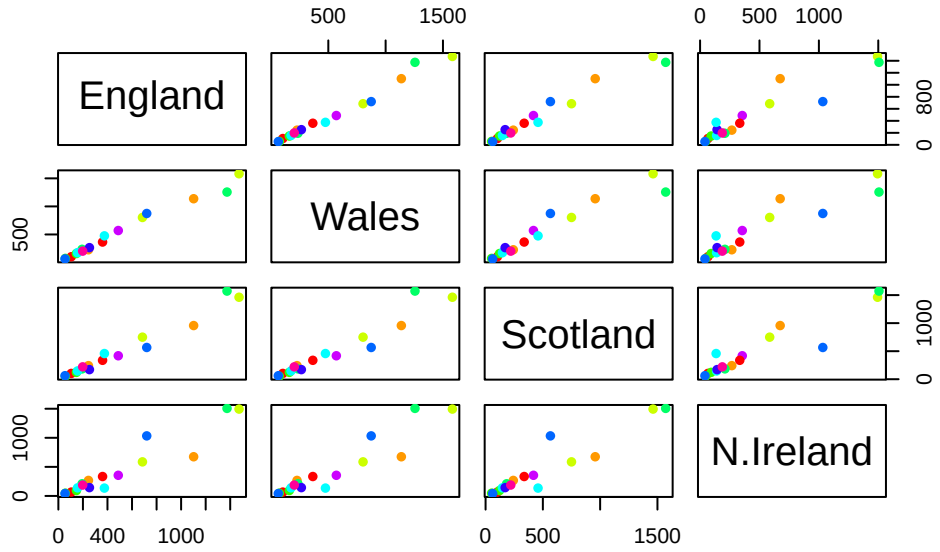
```
barplot(as.matrix(ukFood2), col=rainbow(nrow(ukFood2)))
```



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 following code generates a pairwise plot for our data:

```
pairs(ukFood2, col=rainbow(10), pch=16)
```



PCA to the rescue:

The following is the main PCA function in base R, called `prcomp()`. `t()` creates the transpose of the data frame.

```
pca <- prcomp( t(ukFood2) )
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 that PCA captures 67% of total variance in the original data in one PC and 96.5% in two PCs.

```
attributes(pca)
```

```
$names
```

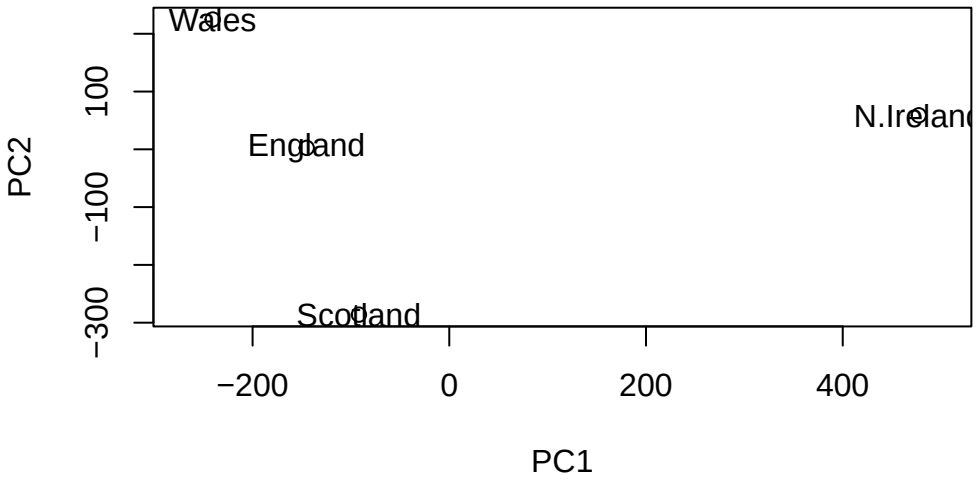
```
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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
```

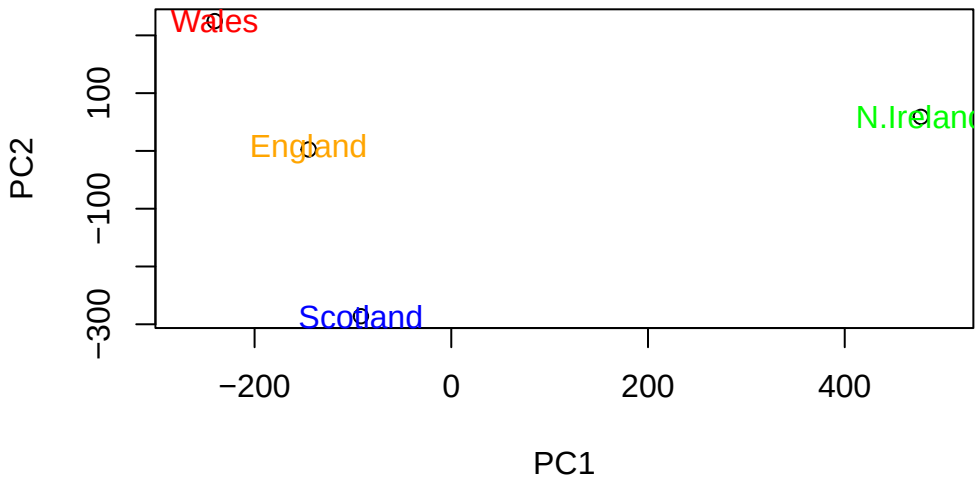
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(ukFood2))
```



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(ukFood2), col = c("orange", "red", "blue", "green"))
```



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
```

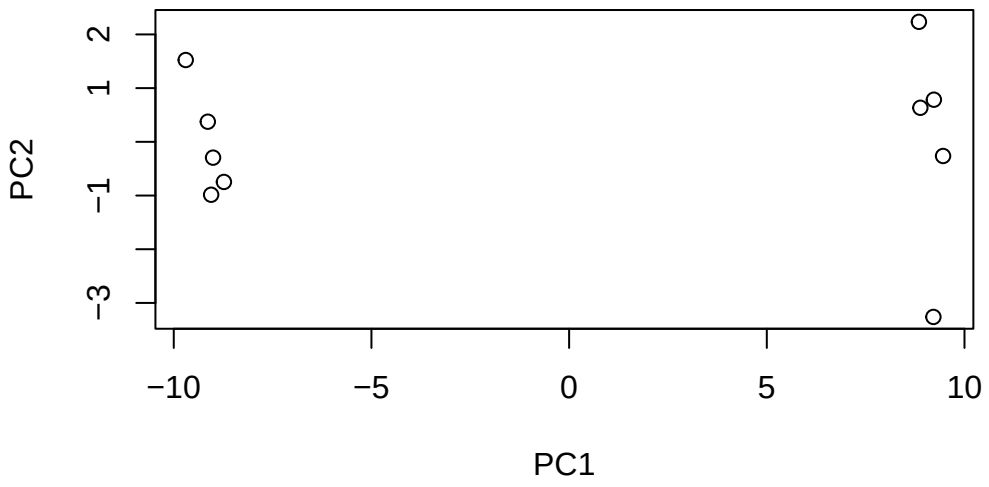
```
dim(rna.data)
```

```
[1] 100  10
```

Q10: How many genes and samples are in this data set?

There are 100 rows (genes) and 10 columns (samples) in the data set.

```
## 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")
```

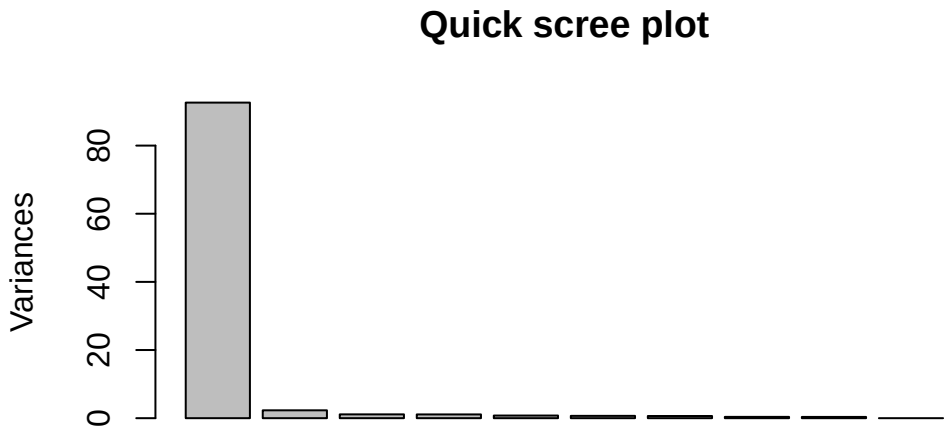



```
summary(pca)
```

```
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 3.327e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
```

```
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

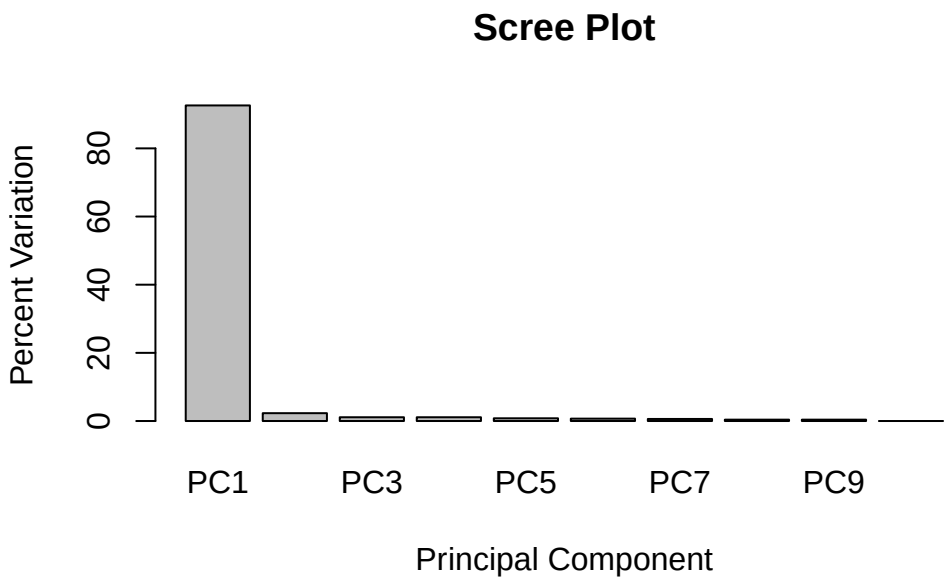
```
plot(pca, main="Quick scree plot")
```



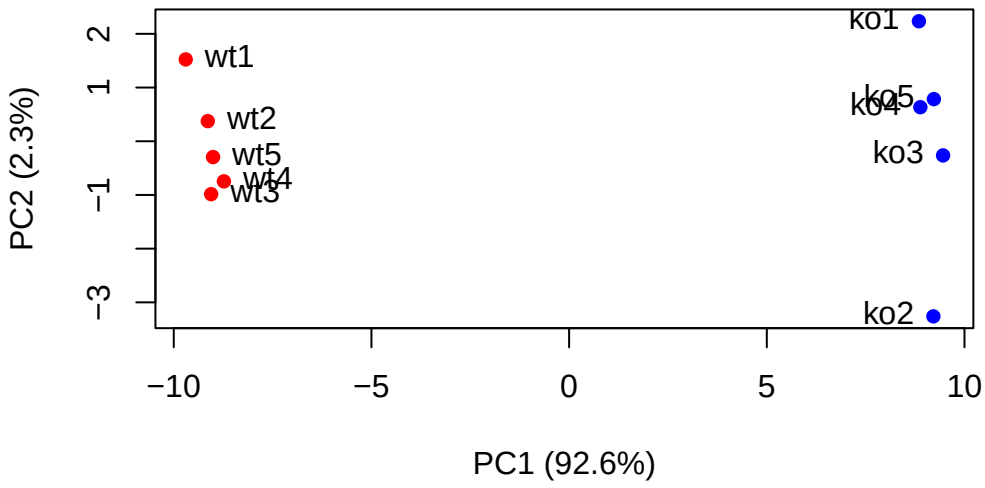

```
## Variance captured per PC
pca.var <- pca$sdev^2

## 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
```

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

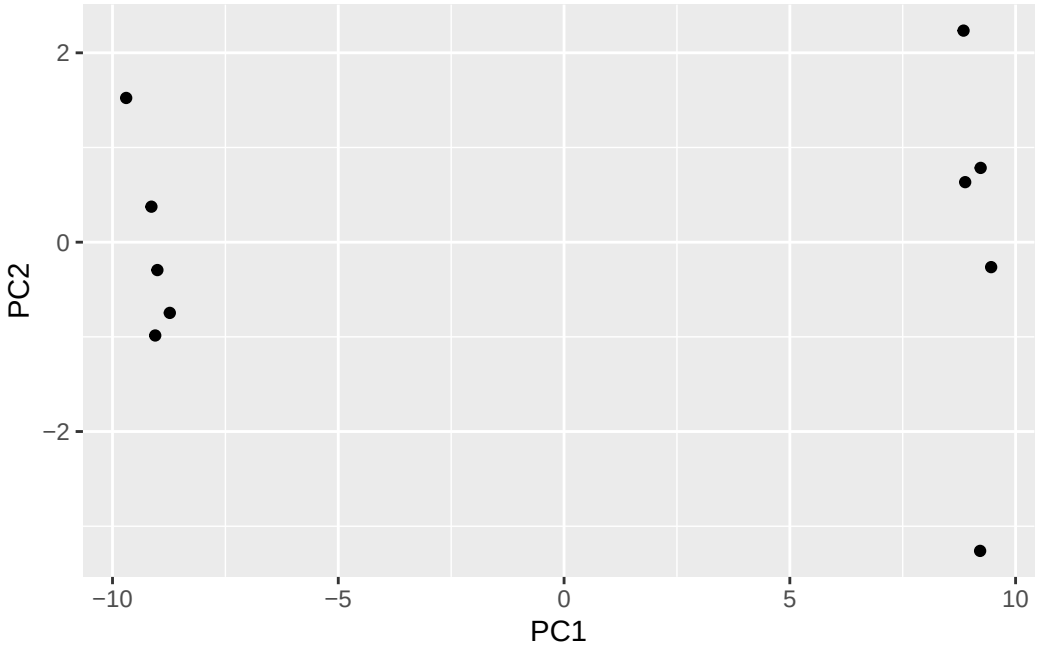



```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```

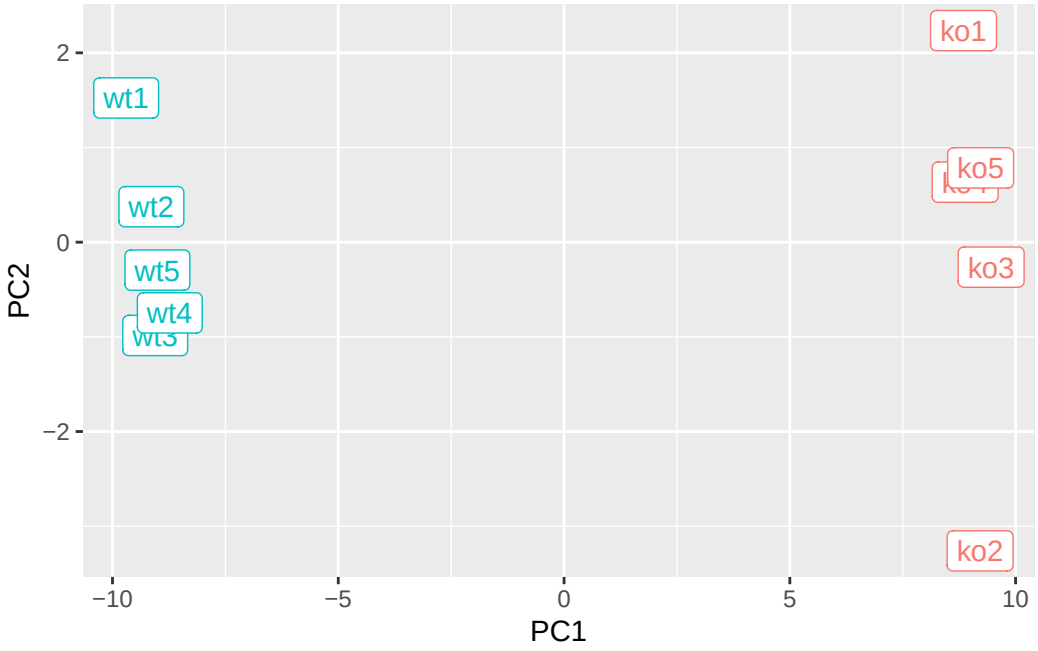


Using ggplot

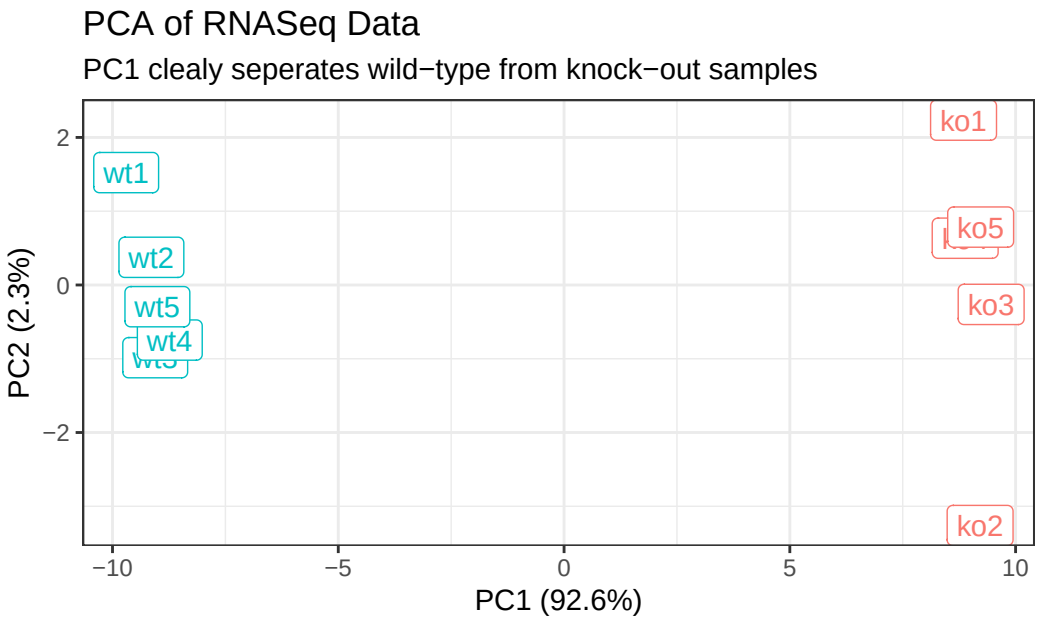
```
library(ggplot2)

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()
```


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