# Machine Learning 1

Zaida Rodriguez (PID:A59010549)

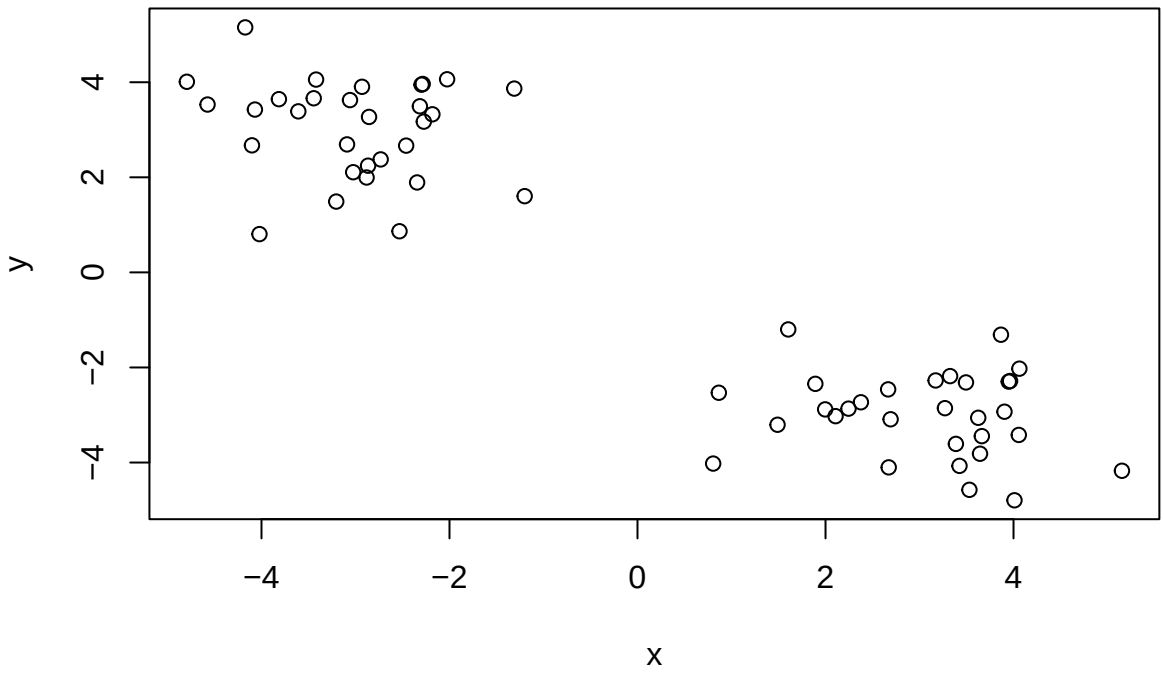
10/22/2021

#Clustering methods

Kmeans clustering in R is done with the `kmeans()` function Here we make up some data to test and learn with.

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

#the goal of this is to make a data set that has -3 and +3 values in x and y
# x:(-3, +3) and y:(+3, -3)
plot(data)
```




Run `kmeans()` set k(centers) to 2 and nstart to 20. The thing with Kmeans is you have to tell it how many clusters you want.

```
km <- kmeans(data, centers=2, nstart=2)
km
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.030438 -2.996526
## 2 -2.996526  3.030438
##
## 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 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] 54.21005 54.21005
##  (between_SS / total_SS =  91.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Clustering vector is telling you for which cluster your element belongs to.

Q1. How manyu points are in each cluster?

```
km$size
```

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

Q2. What 'component of your result object details 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 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
```

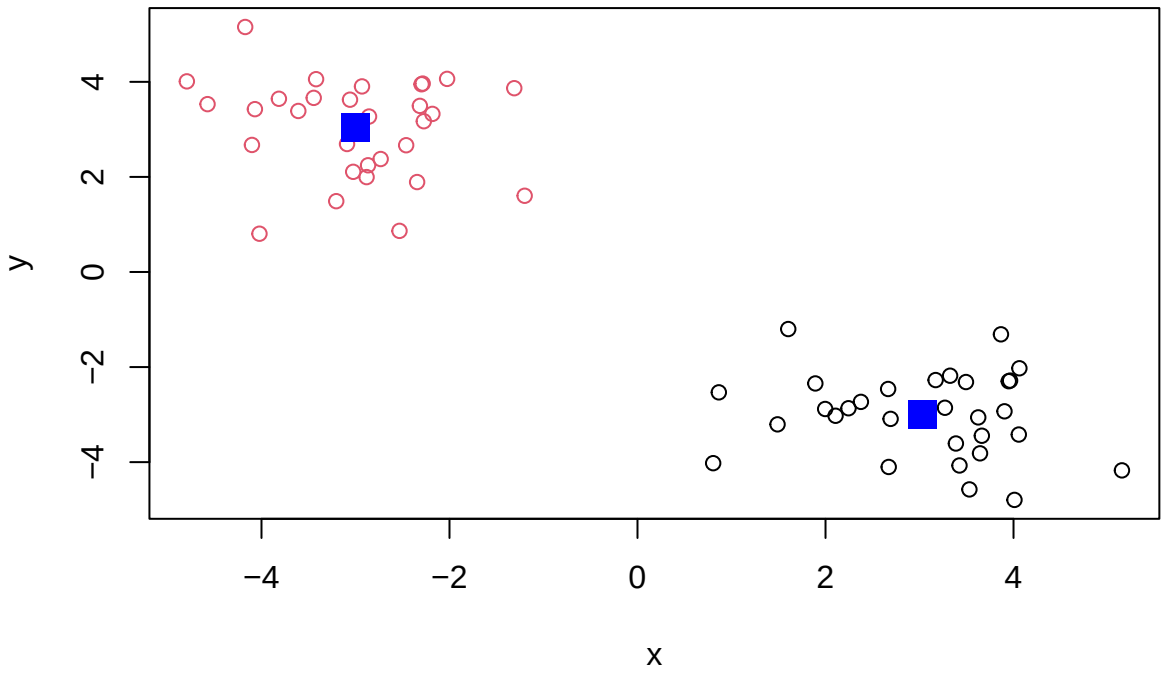
Q3. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1  3.030438 -2.996526
## 2 -2.996526  3.030438
```

Q4. Plot x colored by the kmeans cluster assignment and add clsuters as blue points (or by clusters)

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




#Hierarchical clustering

We will use the `hclust()` function on the same data as before and see how this method works.

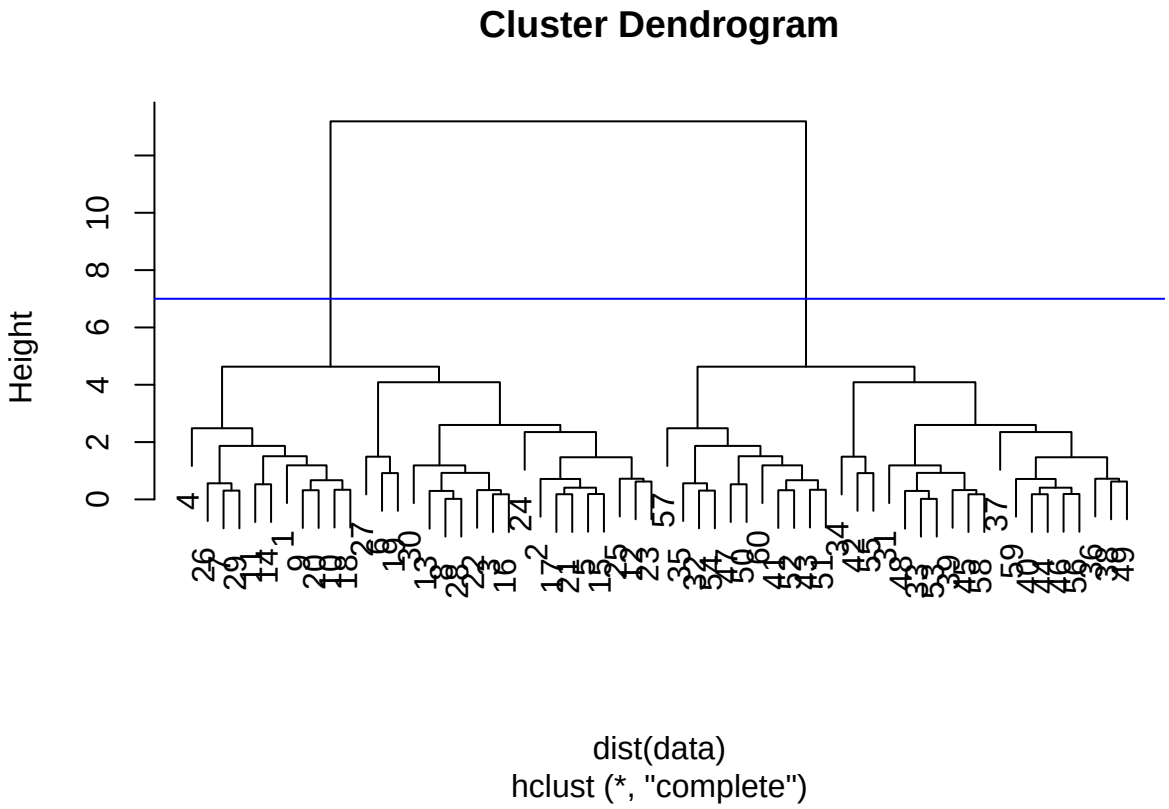
```
hc <- hclust(dist(data))
hc
```

```
##
## Call:
## hclust(d = dist(data))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

```
#this function requires you to tell it the distance from each point
```

hclust has a plot method

```
plot(hc)
abline(h=7, col="blue")
```




```
# the bottom (leaves) are your row names
# it puts what is closest to each other together
# 2 main groups; you can "cut it"
```

To find our membership vector we need to "cut" the tree (dendrogram) and for this we use the `cutree()` fucntion and tell it the height to cut at.

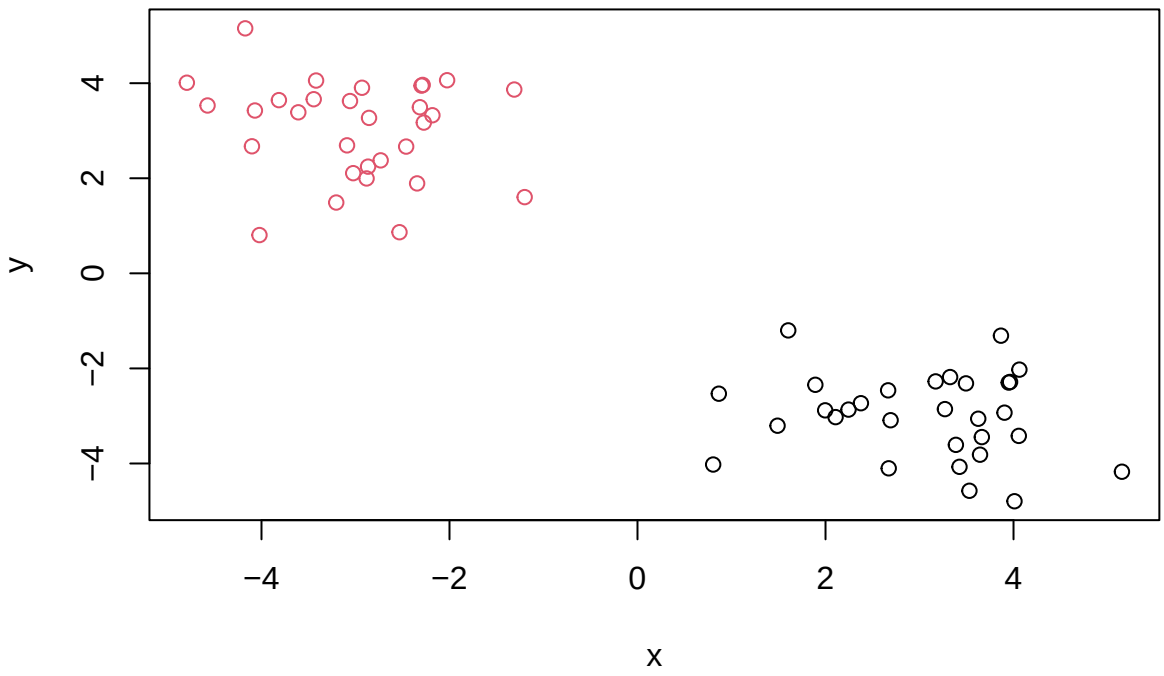
```
cutree(hc,h=7)
```

```
##  [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 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 2 2 2
```

We can also use t we want ...

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

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




# Principal Component Analysis (PCA)

## PCA OF UK Food data

read the file ine

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

How many rows and cols?

```
dim(x)
```

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

```
x[,-1]
```

```
##    England Wales Scotland N.Ireland
## 1      105   103      103        66
## 2      245   227      242       267
## 3      685   803      750       586
## 4      147   160      122        93
```

```
## 5      193   235      184       209
## 6      156   175      147       139
## 7      720   874      566      1033
## 8      253   265      171       143
## 9      488   570      418       355
## 10     198   203      220       187
## 11     360   365      337       334
## 12    1102  1137      957       674
## 13    1472  1582     1462      1494
## 14      57    73       53        47
## 15    1374  1256     1572      1506
## 16     375   475      458       135
## 17      54    64       62        41
```

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

Don't do things this way, you will overwrite your object. Instead, read it in with row names already. This is the preferred method

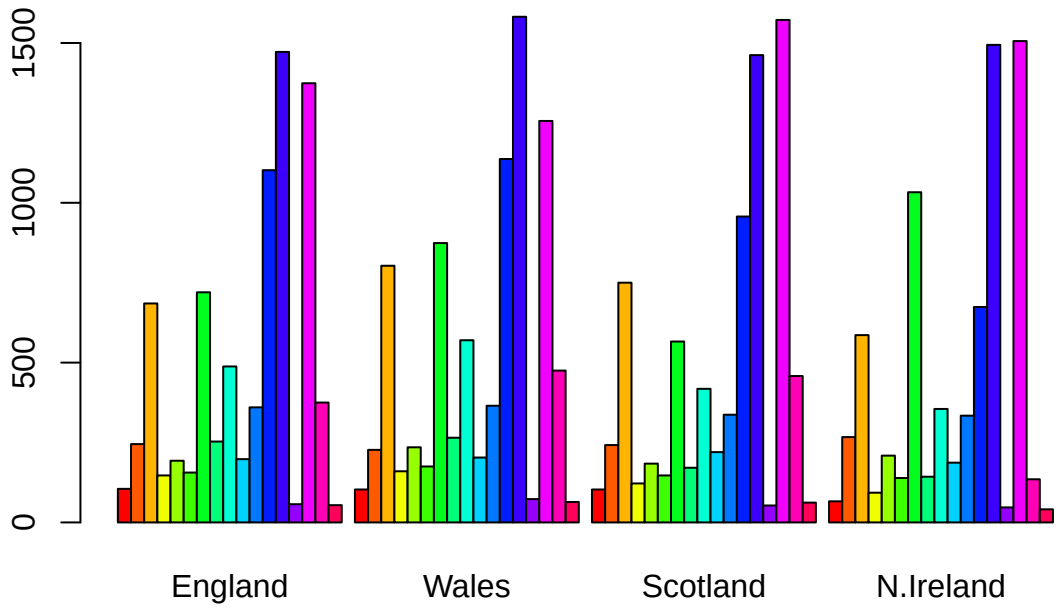
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names = 1)
x
```

```
##                    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
```

now create a barplot. By adding "col=rainbow" you can change the color of the bars

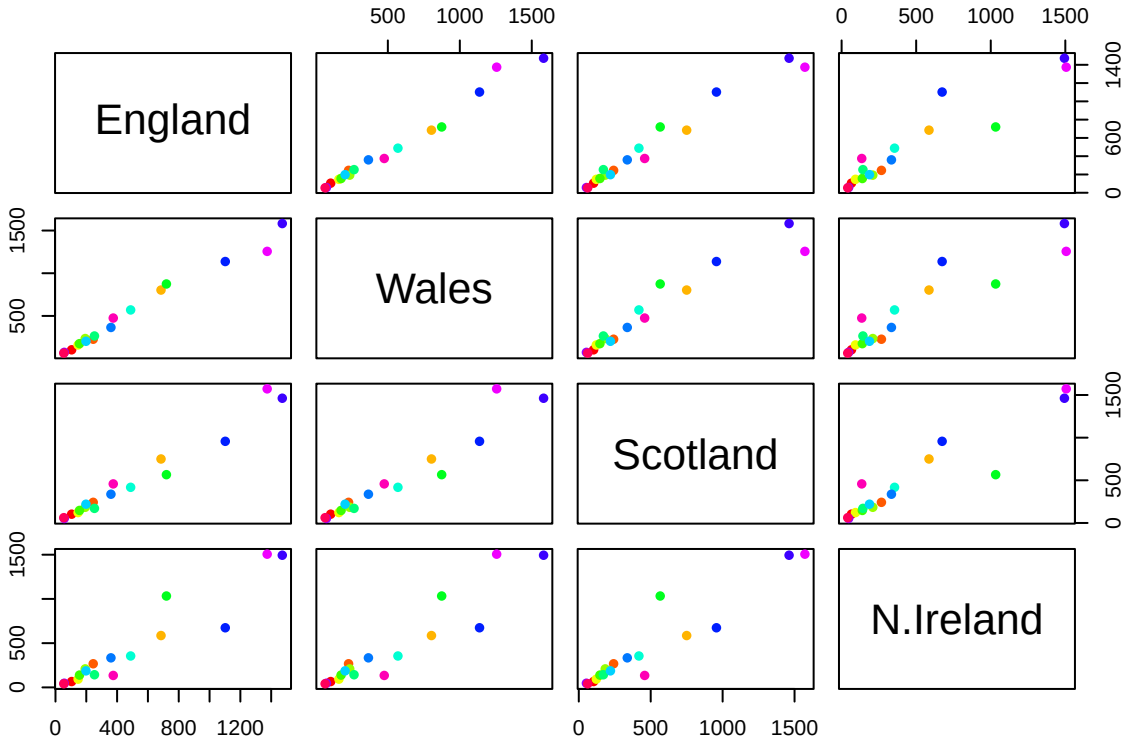
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```




```
# barplot need to be run as a matrix, since we didn't have a matrix, you can force it right into the fu
```

This pairwise plot allows you to compare the 4 countries against each other. The first row compares England (x) and Wales (y), then Scotland (y), then N. Ireland(y) The 2nd row compares Wales (x) and England(y), then N. Ireland (y); and so on and so on. The points that are outliers are visible and those indicate differences in the consumption of the particular food categories depending on the country.

```
mycols <- rainbow(nrow(x))
pairs(x, col=mycols, pch=16)
```




N, Ireland has a greater variation of the food consumption from England, Wales, and Scotland.

### PCA to the rescue!

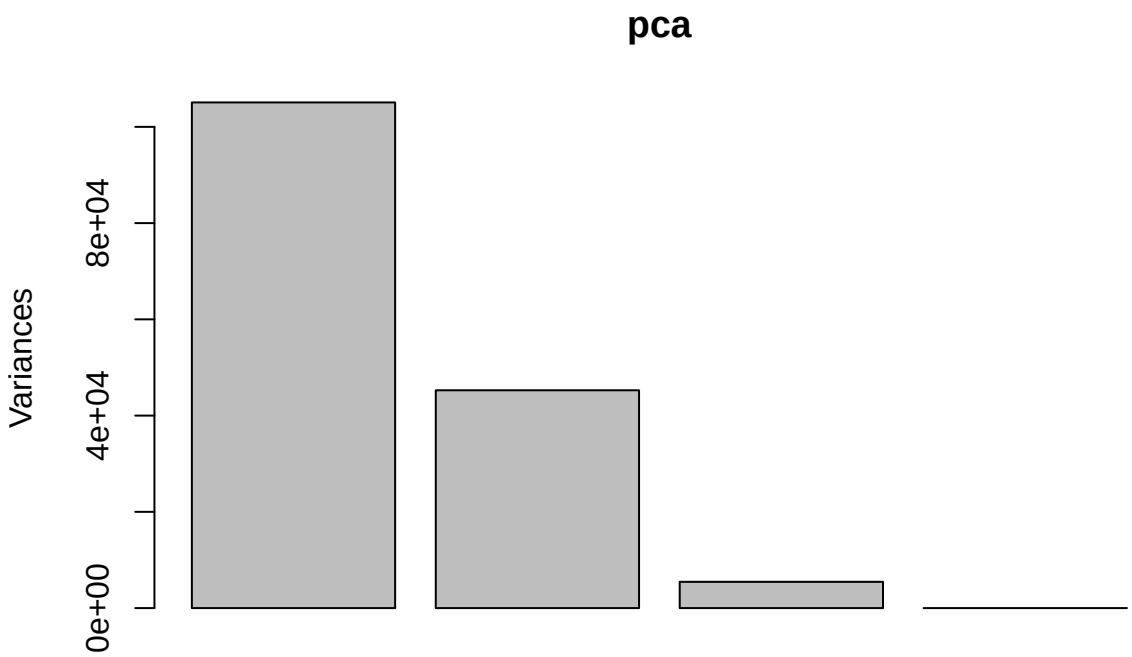
Not easy to interpret, takes time. Instead try PCA!

Here we will use the base R function for PCA, which is called `prcomp()`. This function wants you to first transpose your data (ie: flip the columns with rows)

```
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
```

```
plot(pca)
```

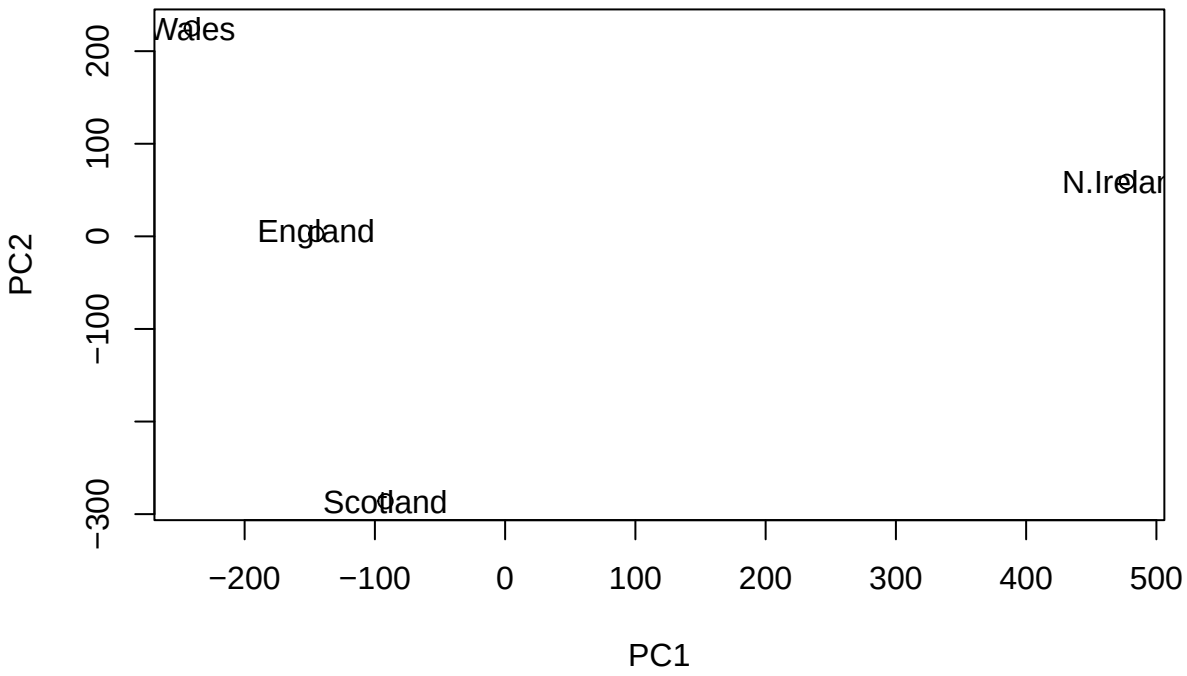



We want a score plot (aka: PCA plot). Basically plot of PC1 vs PC2

```
attributes(pca)
```

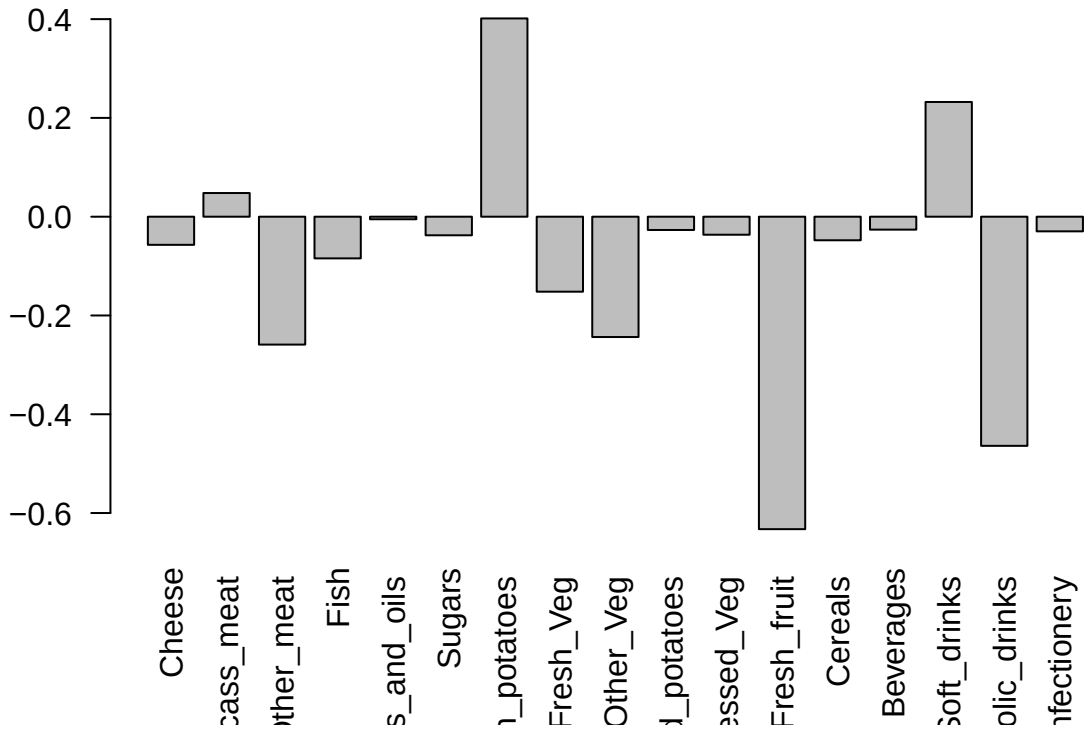
```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels=colnames(x))
```




We can also examine the PCA "loadings", which tells us how much teh original variables contribute to each new PC.

```
barplot(pca$rotation[,1], las=2)
```




### one more PCA

```
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
```

Q. How many genes are there in this data?

```
nrow(rna.data)
```

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

n

```
ncol(rna.data)
```

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

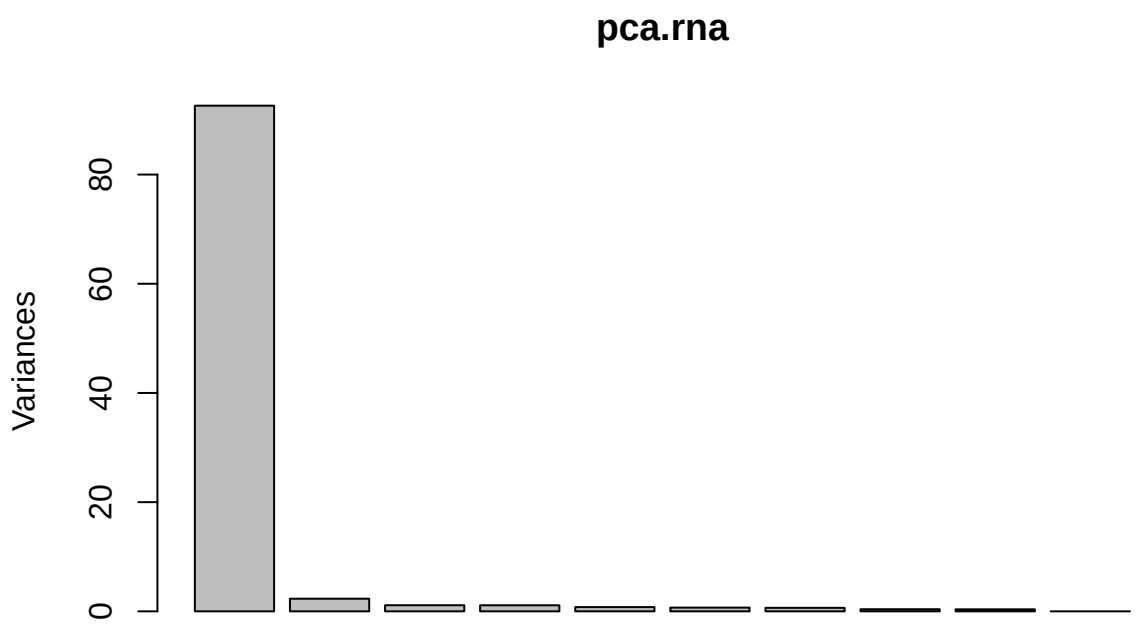
```
colnames(rna.data)
```

```
##  [1] "wt1" "wt2" "wt3" "wt4" "wt5" "ko1" "ko2" "ko3" "ko4" "ko5"
```

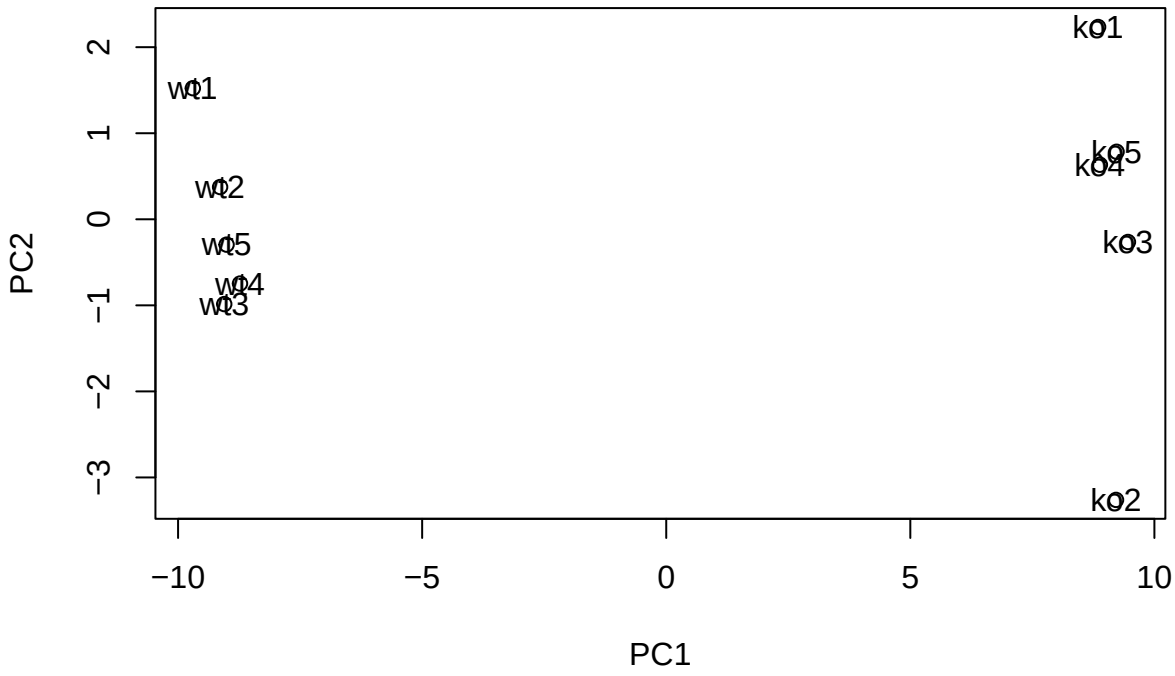
```
pca.rna = prcomp(t(rna.data), scale=TRUE)
summary(pca.rna)
```

```
## 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
```

```
plot(pca.rna)
```



```
plot(pca.rna$x[,1:2])
text(pca.rna$x[,1:2], labels=colnames(rna.data))
```




PC1 is telling us which genes are changing the most between KO to KO and WT to WT. PC2