Class 7: Machine Learning 1

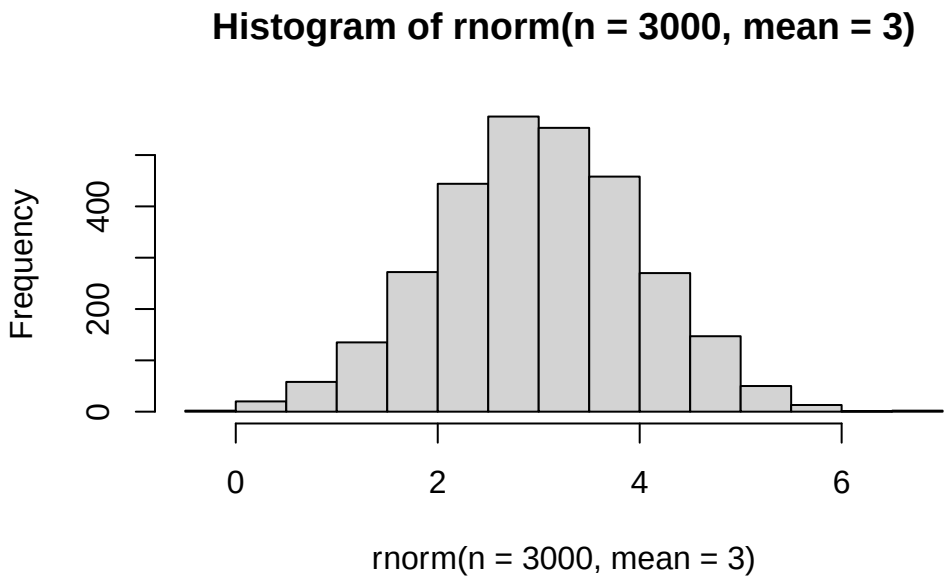
Ashley Allen (PID: A14633373)

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

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

We can use the `rnorm()` function to help us here: This function takes 3 input arguments (n, mean, sd) where mean and sd both have defaults.

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

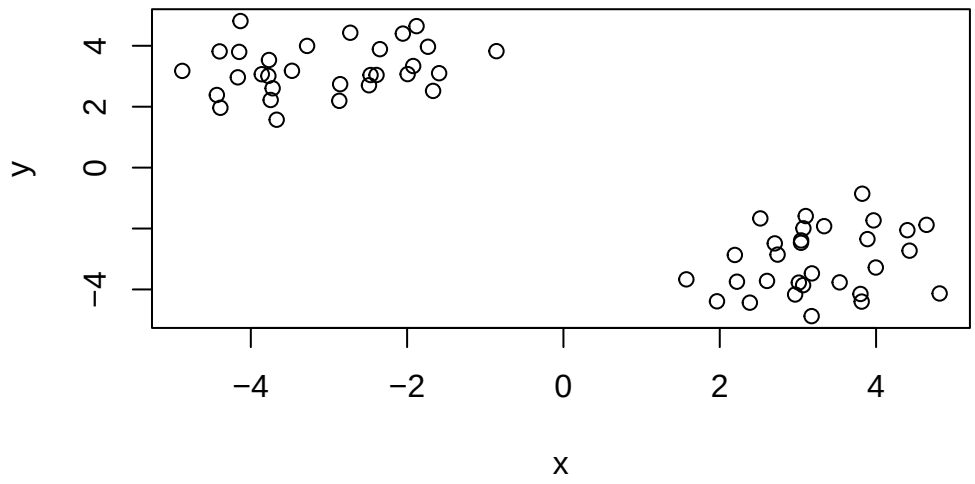


Make data `z` with two "cluster."

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

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

head(z)
```

```
             x        y
[1,] -2.391603 3.041459
[2,] -3.745637 2.220057
[3,] -3.279845 3.996215
[4,] -2.856739 2.739832
[5,] -4.148649 3.800204
[6,] -3.473237 3.178810
```

```r
plot(z)
```



How big is `z`

```r
nrow(z)
```

```
[1] 60
```

```
ncol(z)
```

```
[1] 2
```

K-means clustering

The main function in "base" R for K-means clustering is called `kmeans()`. It has 2 arguments that don't have defaults (x, centers).

-2 clusters because we set centers to 2. -The sizes are the number of data points in each cluster. -When we made z we gave it 30 points each. -Cluster means are the centers of each cluster. -Clustering vector tells us what cluster each point is in.

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

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

Cluster means:
          x         y
1  3.233283 -3.056133
2 -3.056133  3.233283

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 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

Within cluster sum of squares by cluster:
[1] 51.66208 51.66208
 (between_SS / total_SS =  92.0 %)

Available components:

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

```
attributes(z)
```

```
$dim
[1] 60  2
```

```
$dimnames
$dimnames[[1]]
NULL

$dimnames[[2]]
[1] "x" "y"
```

Q. How many points lie in each cluster?

```
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)

```
k$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 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
```

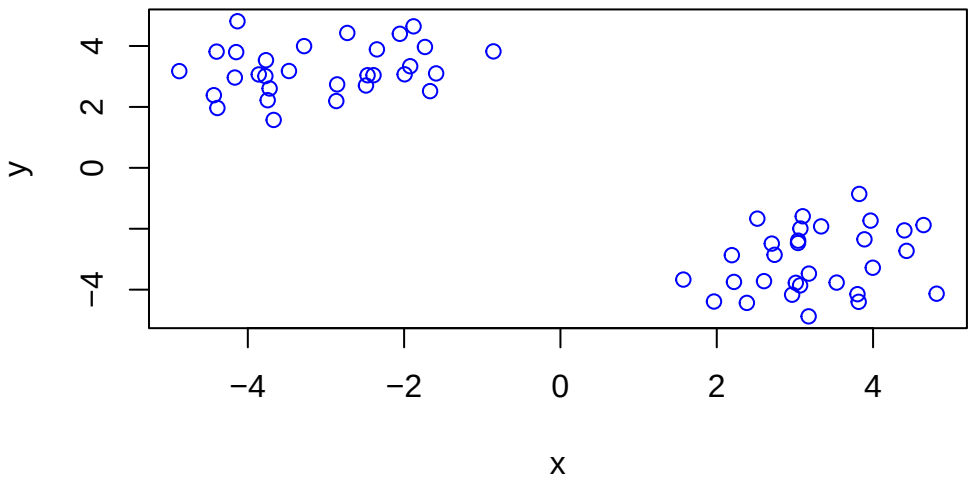
Q. Center of each cluster?

```
k$centers
```

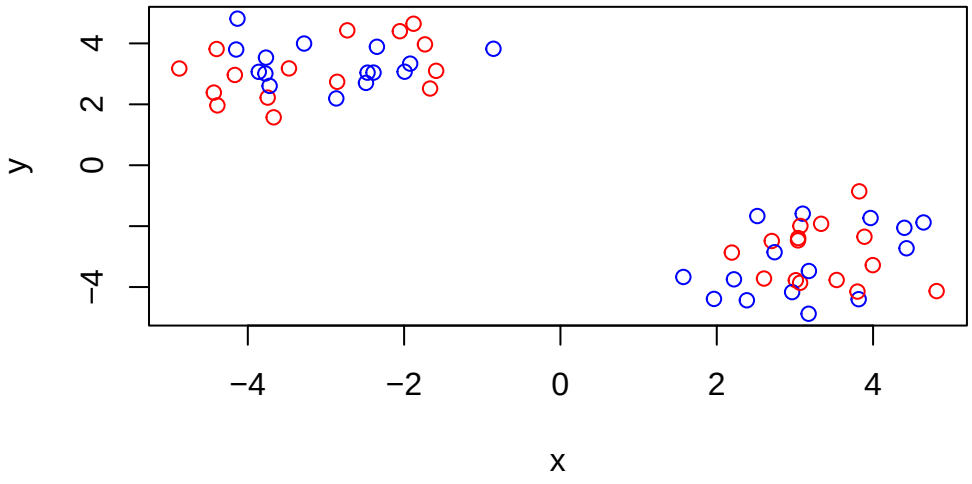
```
          x         y
1  3.233283 -3.056133
2 -3.056133  3.233283
```

Q. Put this result infor 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")
```

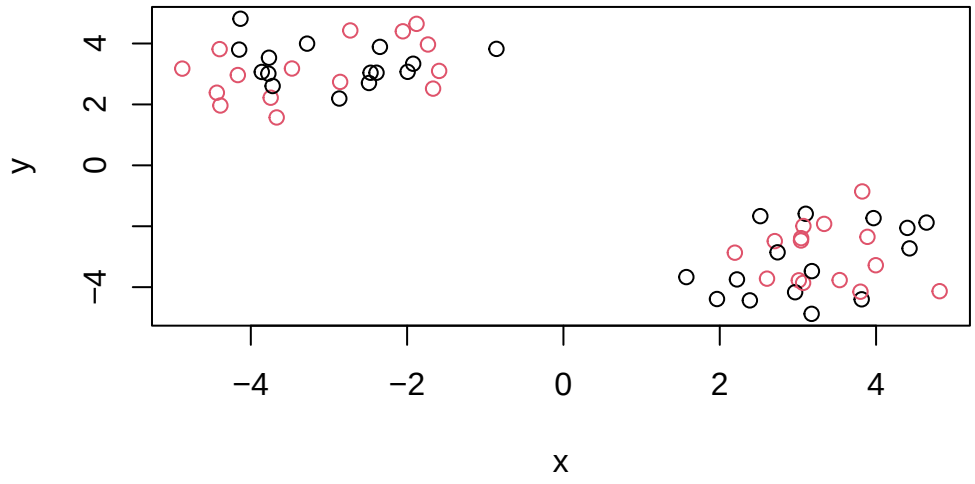



```
plot(z, col=c("blue", "red"))
```



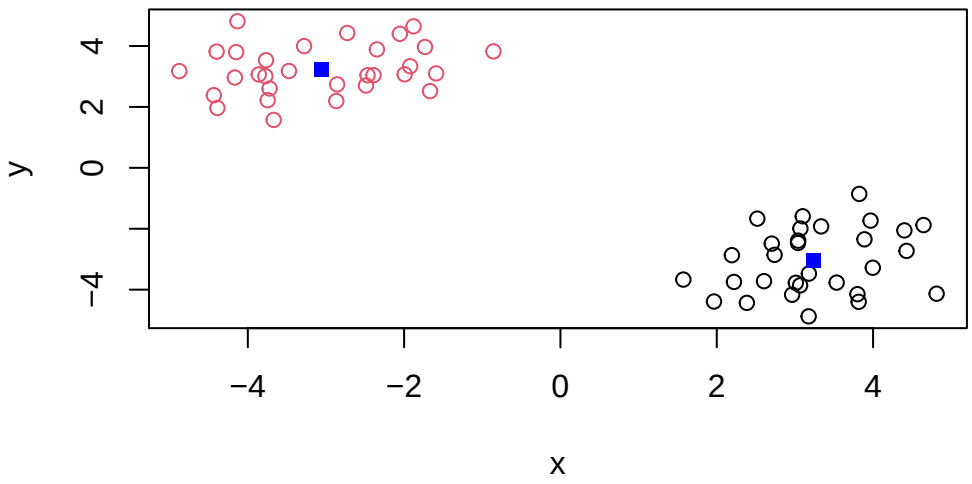
You can color by number (1st from the color palette). Like above the red and black are alternating.

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



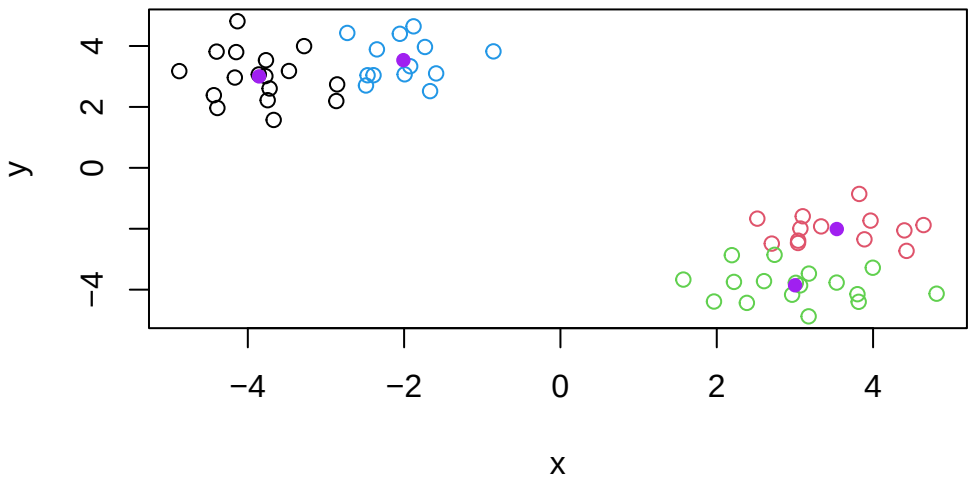
Plot colored by cluster membership:

```
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 visualization plot as above (plot of z colored by cluster membership).

```
k4 <- kmeans(z, centers=4)
plot(z, col=k4$cluster)
points(k4$centers, col="purple", pch=16)
```



Hieraarchical Clustering

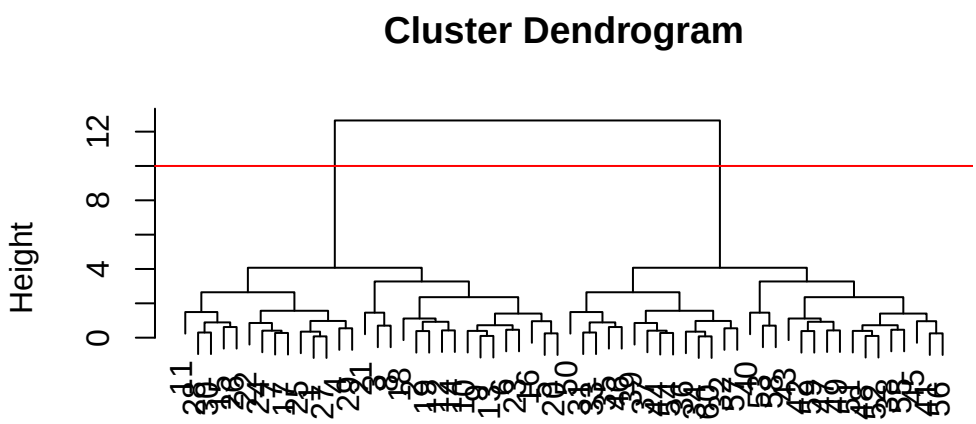
One advantage over K is that it reveals more of a structure in our data set because we set the clusters in K.

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 "raw" data as input - you have to first calculate a distance matrix from your data).

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

```
Call:
hclust(d = d)

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

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



d
hclust (*, "complete")

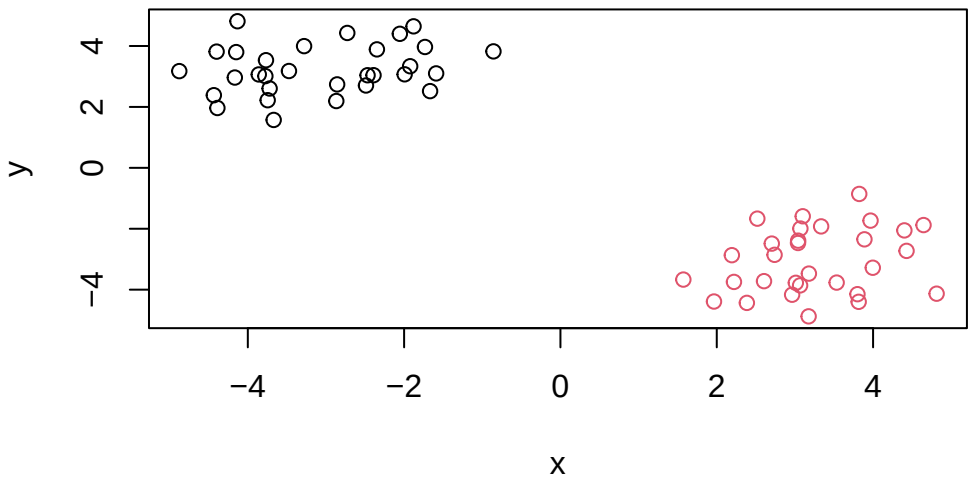
Once I inspect the dendrogram ("tree") I can cut it to yield my grouping or clusters. The function to do this is called `cutree()`. Above we used `abline()` to visualize where we'll cut.

```
cutree(hc, h=10)
```

```
 [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
```

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

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



Hands on with Principal Component Analysis (PCA)

Let's examine some silly 17-dimensional data detailing food consumption in the UK (England, Scotland, Wales, and N. Ireland). Are these countries eating habits different or similar and if so how?

Data import

```
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
```

> Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

The function `nrow()` returns the number or rows, `ncol()` the number of columns, and `dim()` returns both.

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

```
dim(x)
```

```
[1] 17  4
```

To preview the first 6 rows of our data:

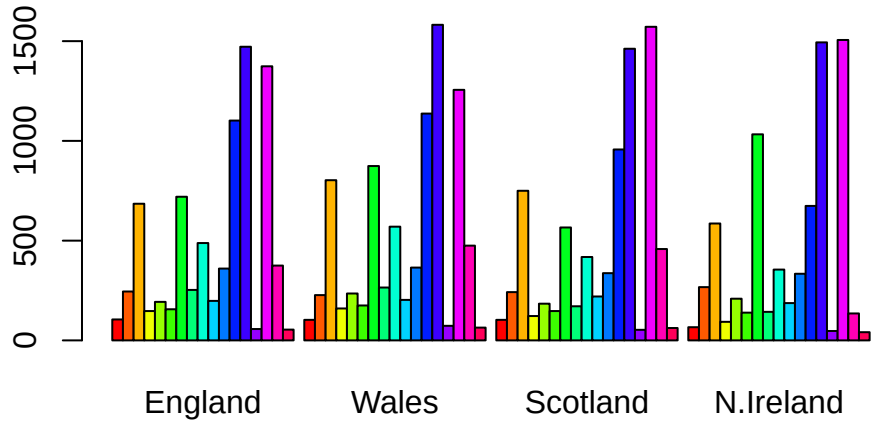
```
head(x)
```

```
              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?

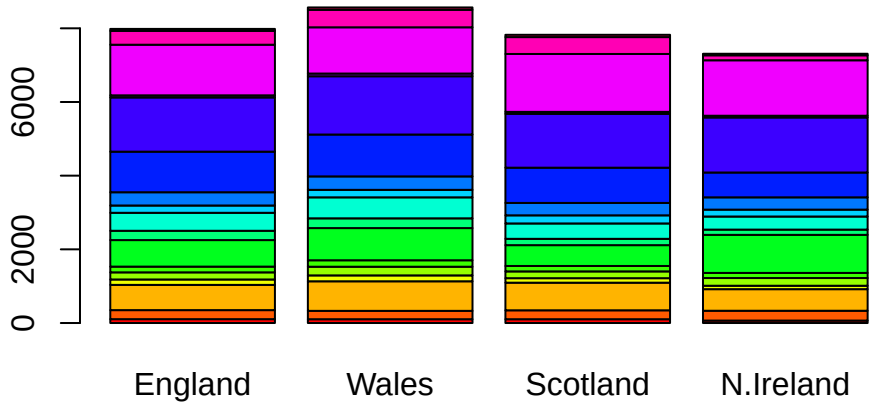
When we first imported our data we used `row.names()` in order to fix our row names problem rather than minus indexing. I prefer row.names() in our first example because it utilizes less code. Running a minus index multiple times would continue to remove row after row and you would eventually have an empty data set.

```
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?

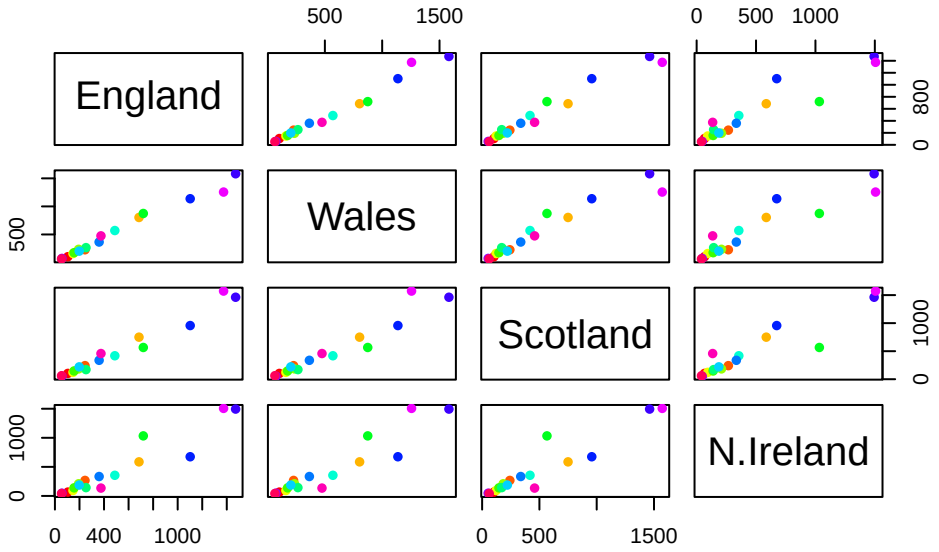
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

This plot compares each country to one another. In the first row for example, England is on the Y axis while the x axis is the following country on the diagonal. If a given point lies on the diagonal it indicates the similarity among both of the countries. If they're not on the diagonal then they're is a difference among the food groups, one value is more in one country than another.

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



Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

It's really difficult to differentiate the differences between data sets, but visually we can see that N. Ireland has multiple food groups that aren't similar to other countries. We can see this in the last plot as less points are aligned on the diagonal.

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

PCA to the rescue!

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

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-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`

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

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" (a.k.a. "PCs", "eigenvectors", etc.)

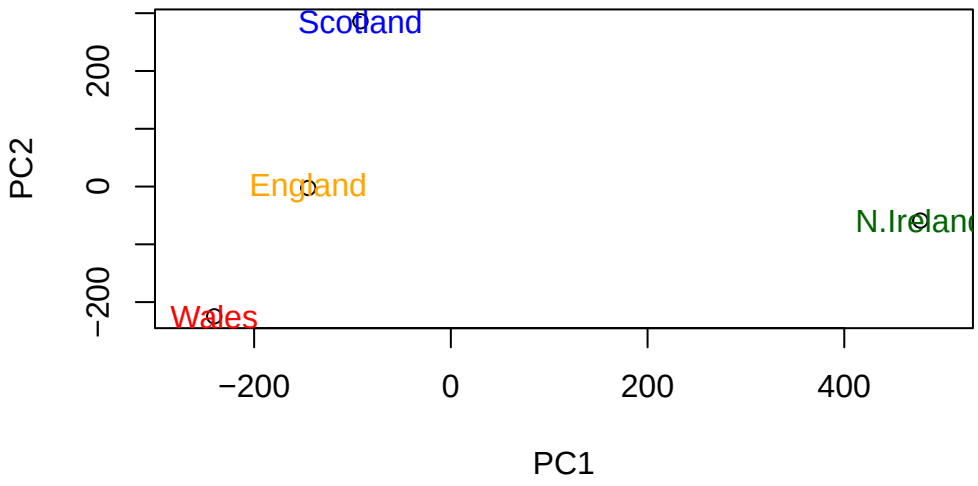
```
head(pca$x)
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

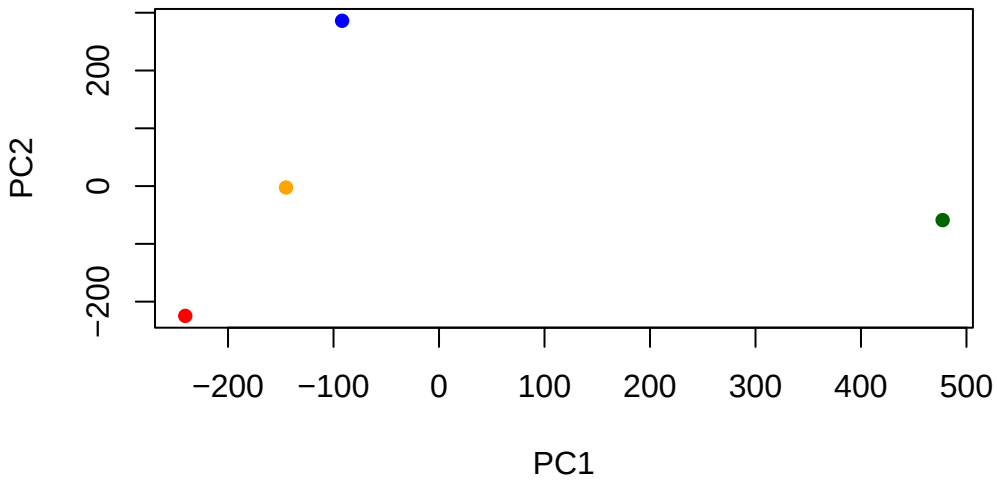
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

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 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), col=c("orange", "red", "blue", "darkgreen"))
```




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



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

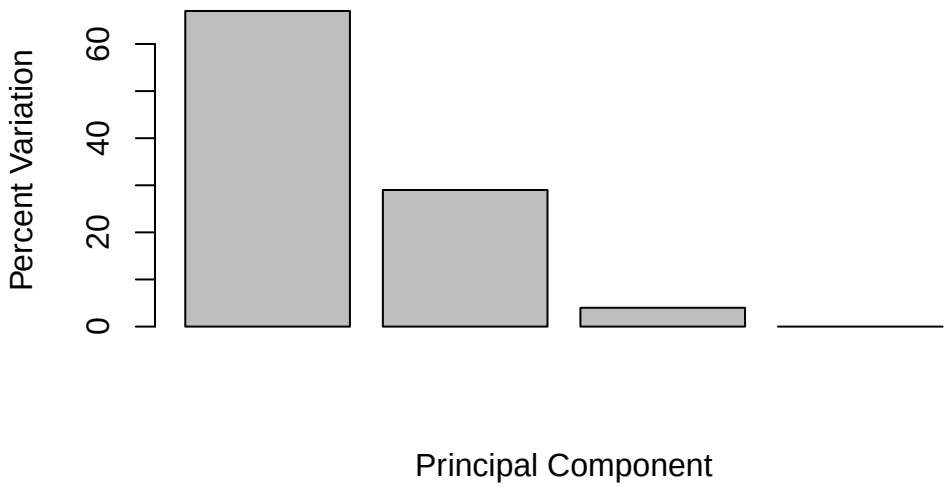
```
[1] 67 29  4  0
```

```
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 3.175833e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

To help visualize the variation in each PC

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



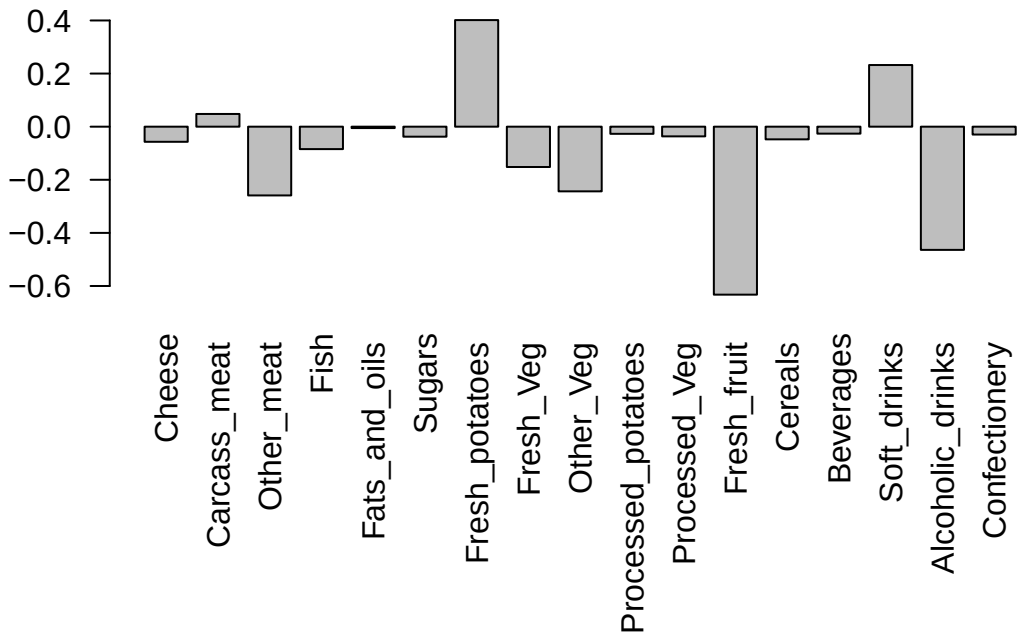
We can look at the so-called PC "loadings" result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better PC variables).

```
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
```

PC1

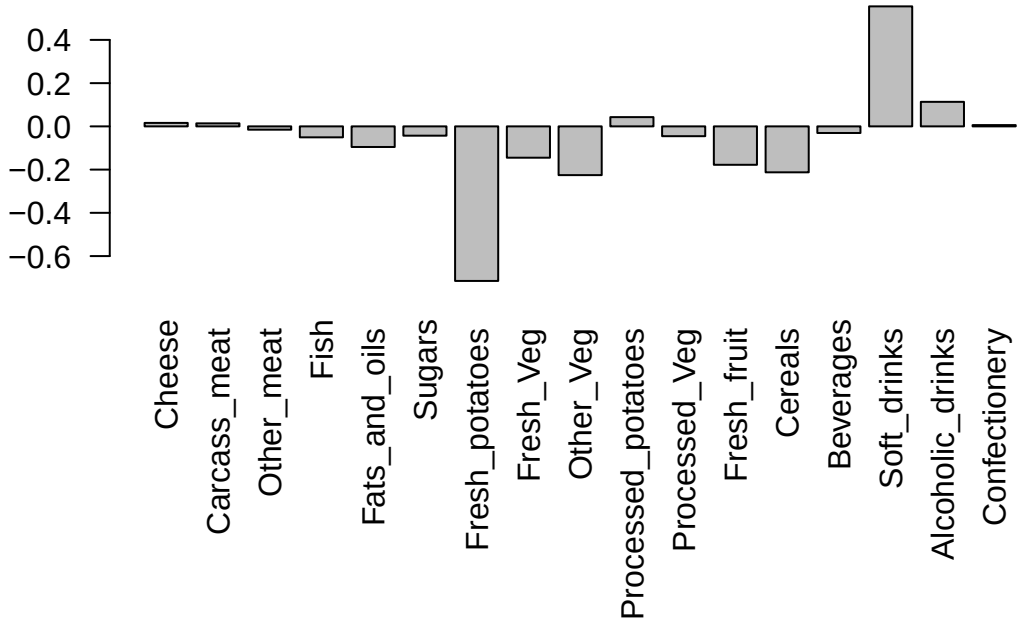
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

Fresh potatoes and soft drinks are prominent, but fresh potatoes is negative while soft drinks are positive. PC2 tells us there is a larger variance in these two food groups between N. Ireland and other countries.

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

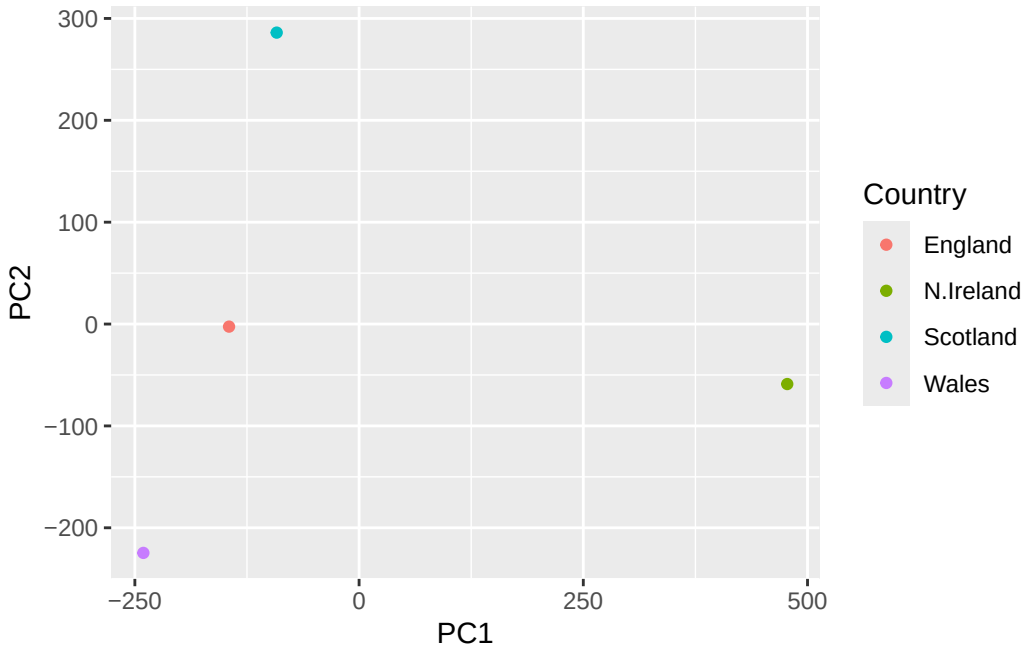


Ggplot

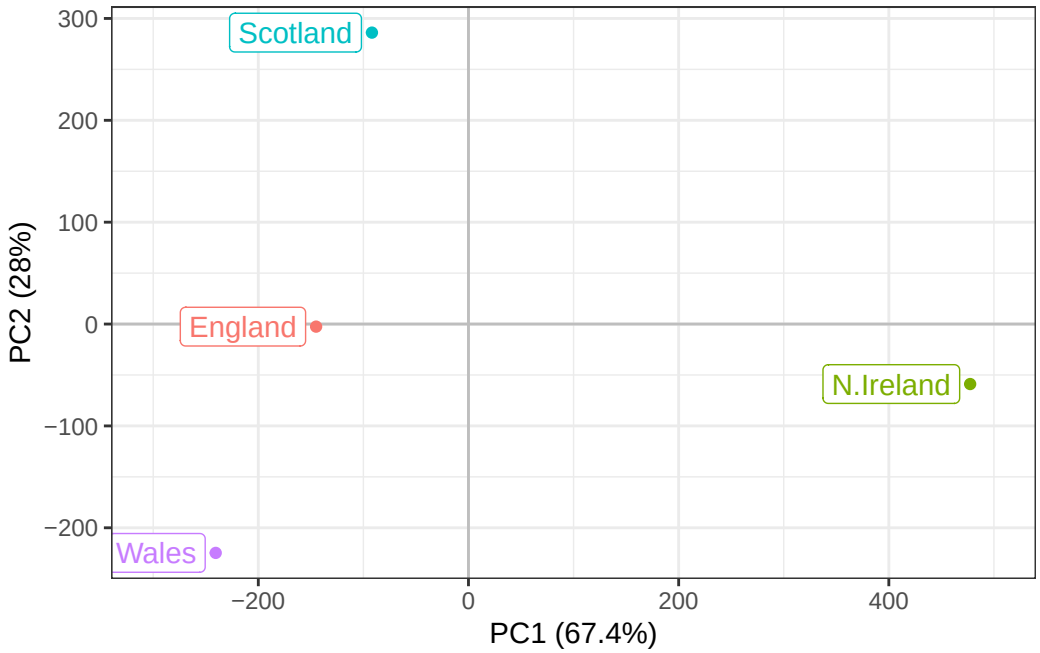
```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.3.3
```

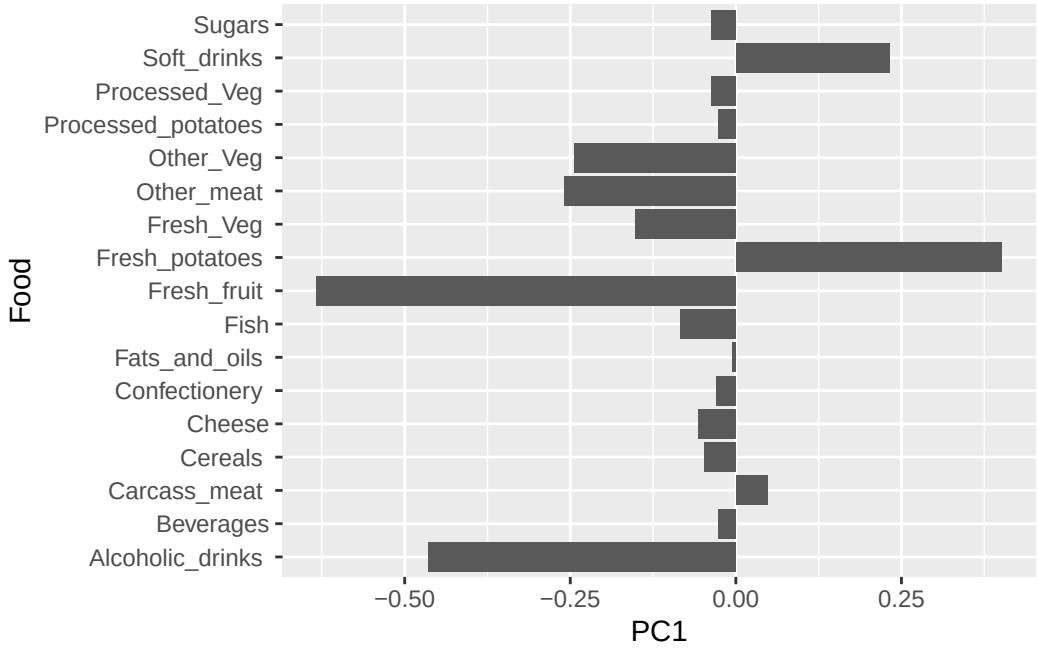
```
df <- as.data.frame(pca$x)
df_lab <- tibble::rownames_to_column(df, "Country")

ggplot(df_lab) +
  aes(PC1, PC2, col=Country) +
  geom_point()
```



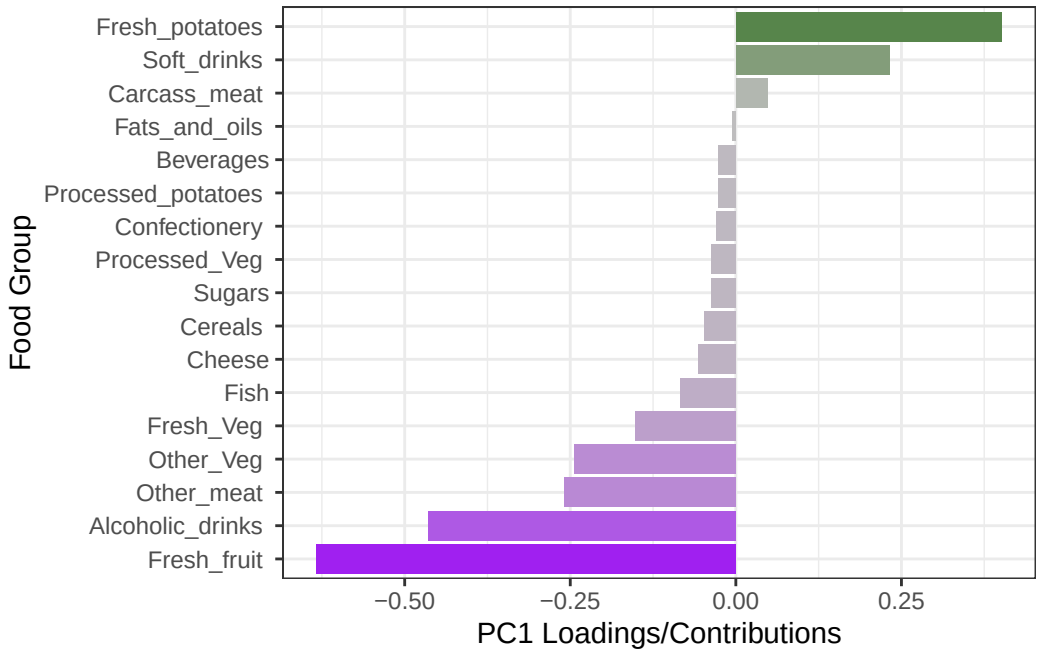

```
ggplot(df_lab) +
  aes(PC1, PC2, col=Country, label=Country) +
  geom_hline(yintercept = 0, col="gray") +
  geom_vline(xintercept = 0, col="gray") +
  geom_point(show.legend = FALSE) +
  geom_label(hjust=1, nudge_x = -10, show.legend = FALSE) +
  expand_limits(x = c(-300,500)) +
  xlab("PC1 (67.4%)") +
  ylab("PC2 (28%)") +
  theme_bw()
```




```
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

ggplot(ld_lab) +
  aes(PC1, Food) +
  geom_col()
```

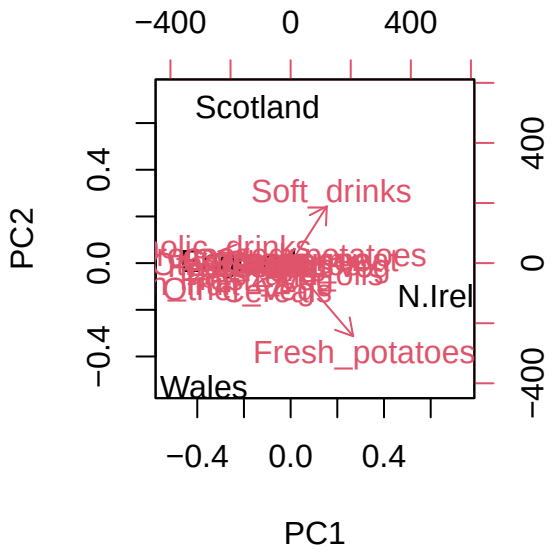



```
ggplot(ld_lab) +
  aes(PC1, reorder(Food, PC1), bg=PC1) +
  geom_col() +
  xlab("PC1 Loadings/Contributions") +
  ylab("Food Group") +
  scale_fill_gradient2(low="purple", mid="gray", high="darkgreen", guide=NULL) +
  theme_bw()
```



Biplots

```
biplot(pca)
```



PCA of RNA-seq data

The samples are columns and the genes are rows.

```
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?

Number of genes:

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

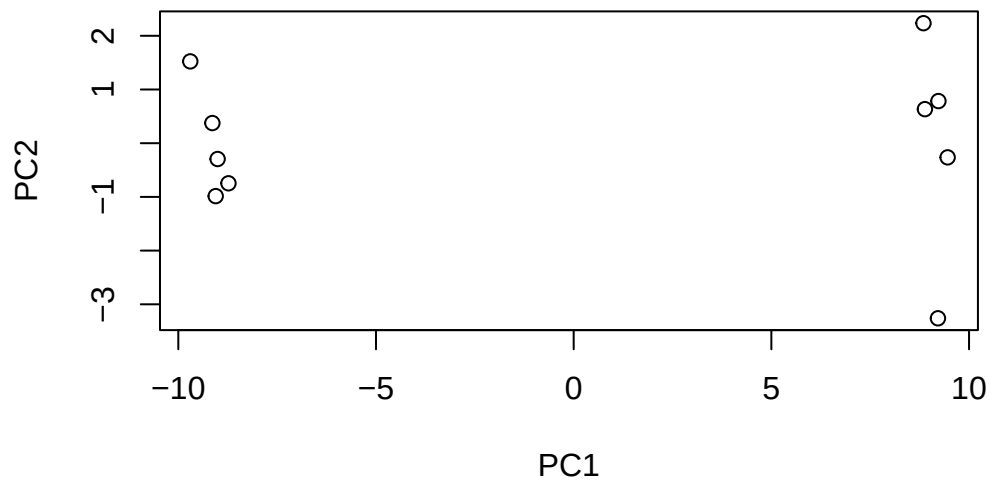
```
[1] 100
```

Number of samples:

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

```
[1] 10
```

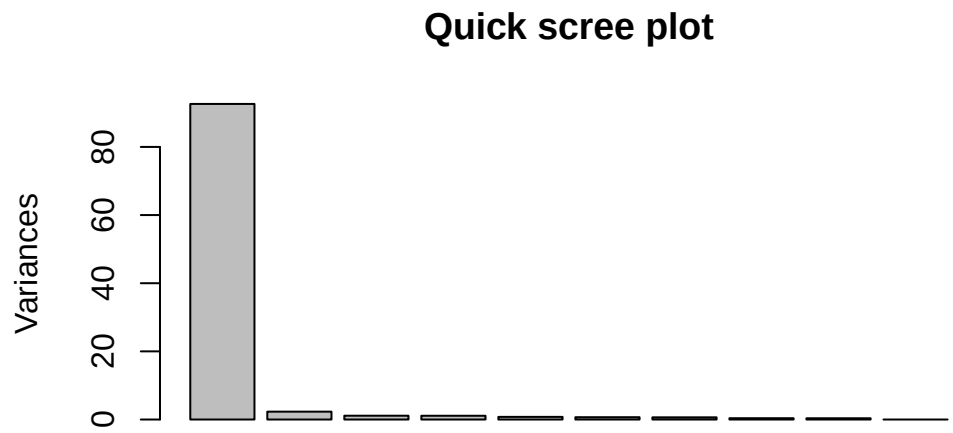
```
pca <- prcomp(t(rna.data), scale=TRUE)
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```




```
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 3.457e-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")
```

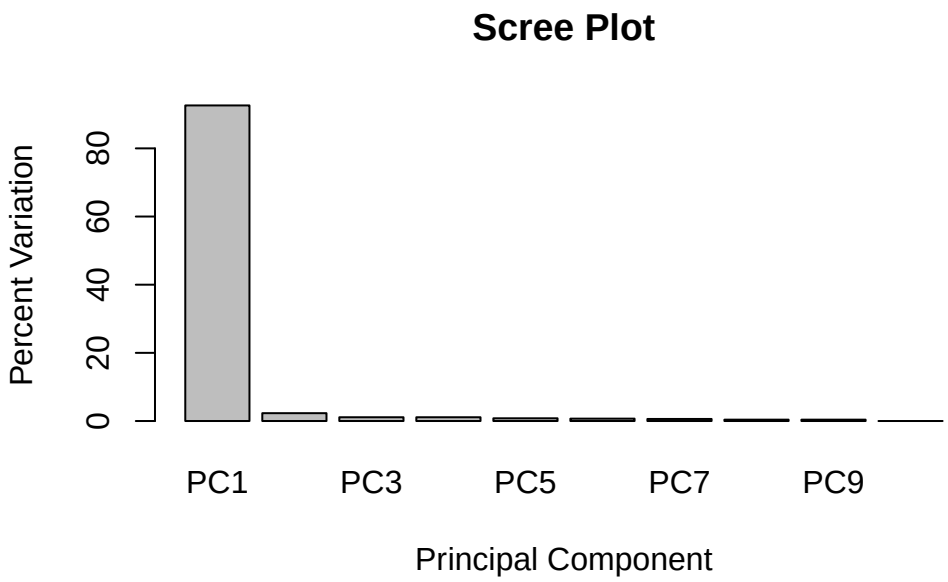



```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

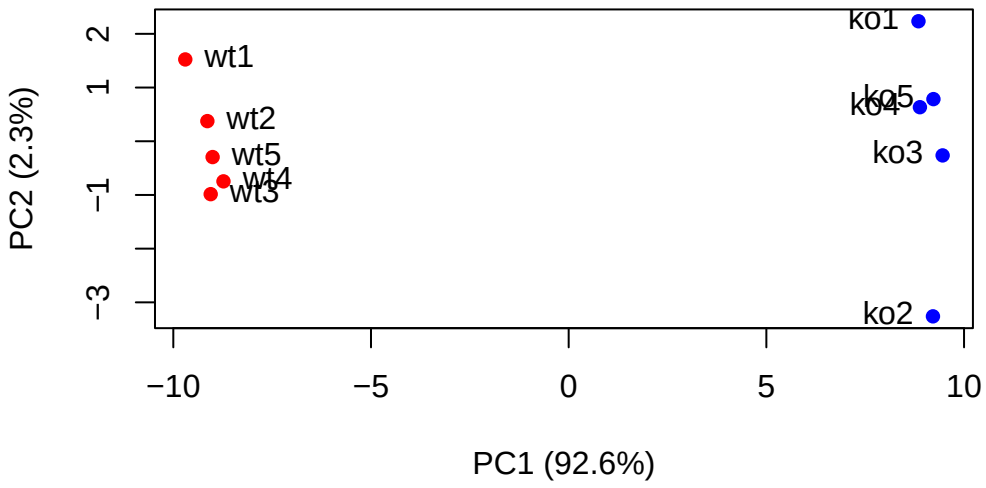
```
 [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

We see in our scree plot that PC1 has captured most of the variance.

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

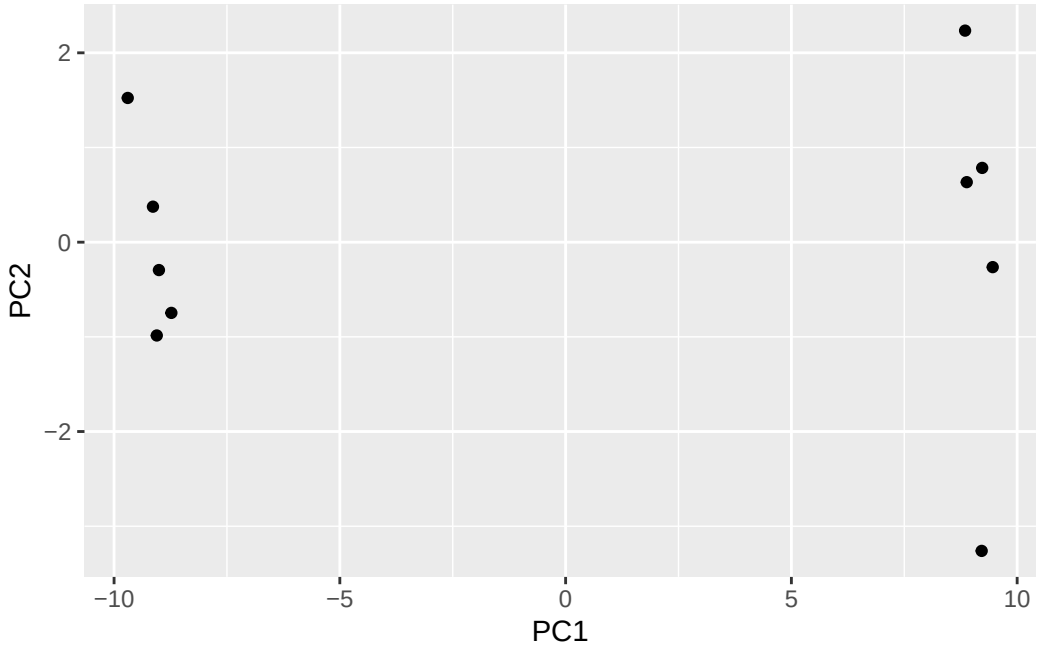



```
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```



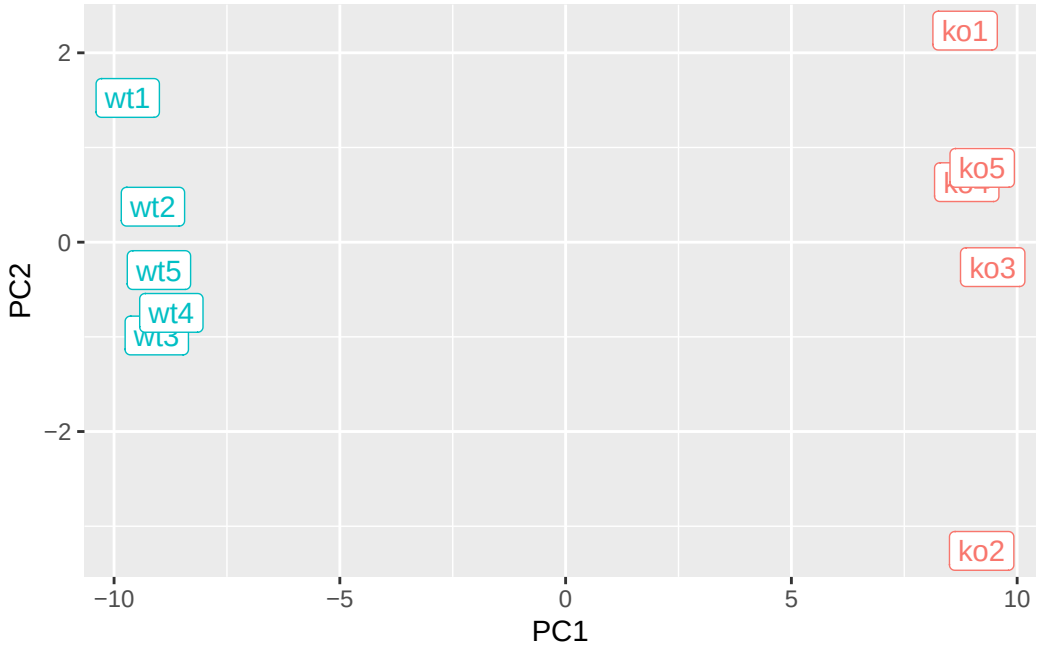
Using ggplot to make our first basic graph:

```
library(ggplot2)

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

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

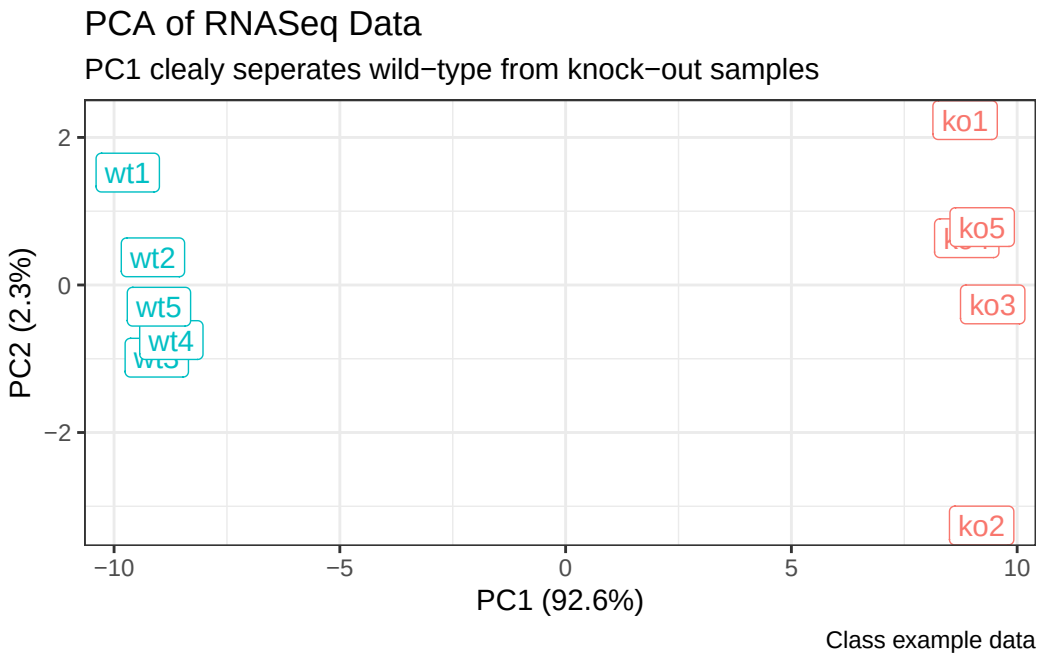


To add wt and ko conditions:

```
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()
```



The top 10 genes that contribute the most to PC1 in either the positive or negative direction:

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

gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes
```

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