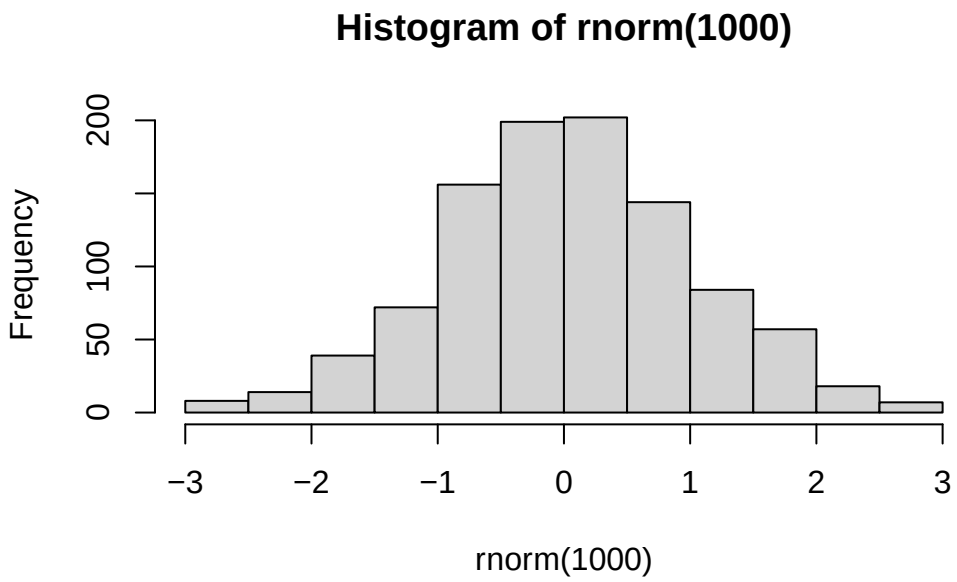
class07: Machine Learning 1

Rachel (PID:A16037641)

Today we will begin our exploration of some "classical" machine learnign approaches. We will start with clustering:

Let's first make up some data to cluster where we know what the answer should be.

```
hist(rnorm(1000))
```

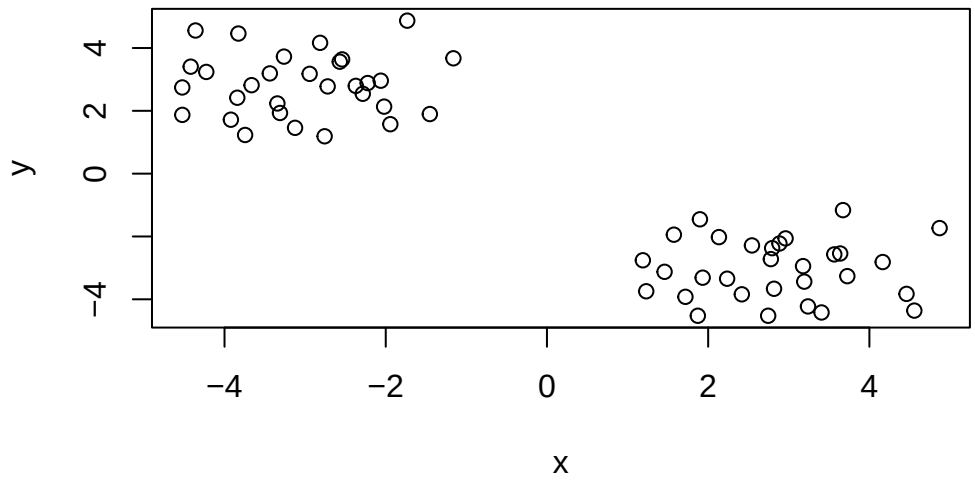



```
x <- c( rnorm(30, mean=-3), rnorm(30, mean=+3))
y<- rev(x)
x <- cbind(x,y)
head(x)
```

```
            x        y
[1,] -3.311719 1.931786
[2,] -4.419169 3.406210
[3,] -2.814196 4.164933
[4,] -3.262510 3.726027
[5,] -2.720162 2.777510
[6,] -4.359768 4.557326
```

Peek at x with `plot()`

```
plot(x)
```



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

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

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

Cluster means:
          x         y
1  2.828172 -3.037670
```

```
2 -3.037670  2.828172

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] 54.81431 54.81431
 (between_SS / total_SS =  90.4 %)

Available components:

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

Q. How big are the clusters (i.e their size)?

```
k$size
```

```
[1] 30 30
```

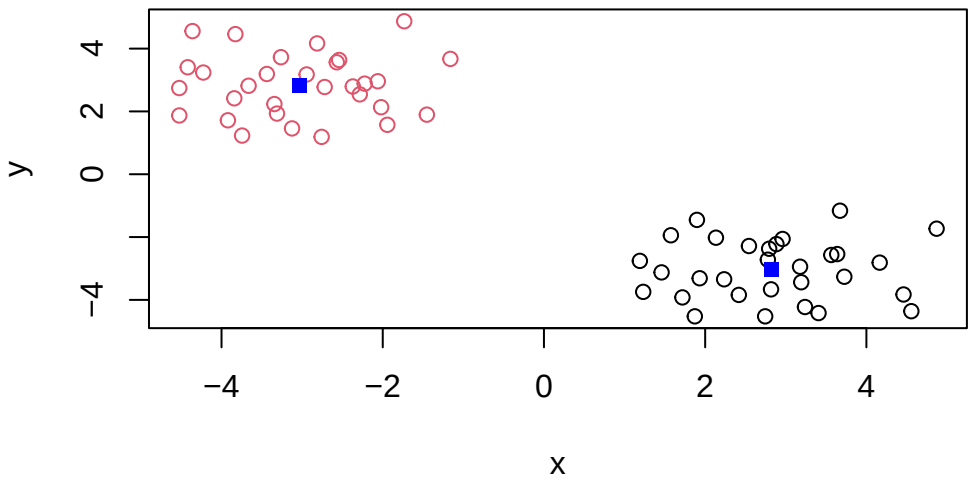
Q. What clusters do my data points reside in? What are my cluster results?

```
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 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
```

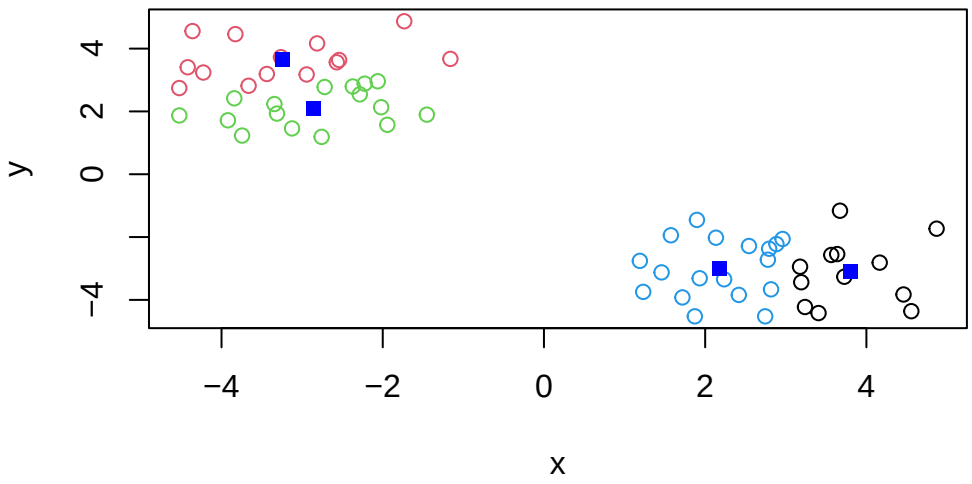
Q. Make a plot of our data colored by clsuter assignment - i.e. Make a rsult figure...

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



Q. Cluster with k-means into 4 clusters and plot your results as above.

```
k4 <- kmeans(x, centers=4)
plot(x, col=k4$cluster)
points(k4$centers,col="blue", pch=15)
```



Q. Run kmeans with centers (i.e. values of k) equal 1 to 6

```
k$tot.withinss
```

```
[1] 109.6286
```

```
k1 <-kmeans(x, centers=1)$tot.withinss
k2 <-kmeans(x, centers=1)$tot.withinss
k3 <-kmeans(x, centers=1)$tot.withinss
k4 <-kmeans(x, centers=1)$tot.withinss
k5 <-kmeans(x, centers=1)$tot.withinss
k6 <-kmeans(x, centers=1)$tot.withinss

ans <- c(k1,k2,k3,k4,k5,k6)
```

Or use a for loop

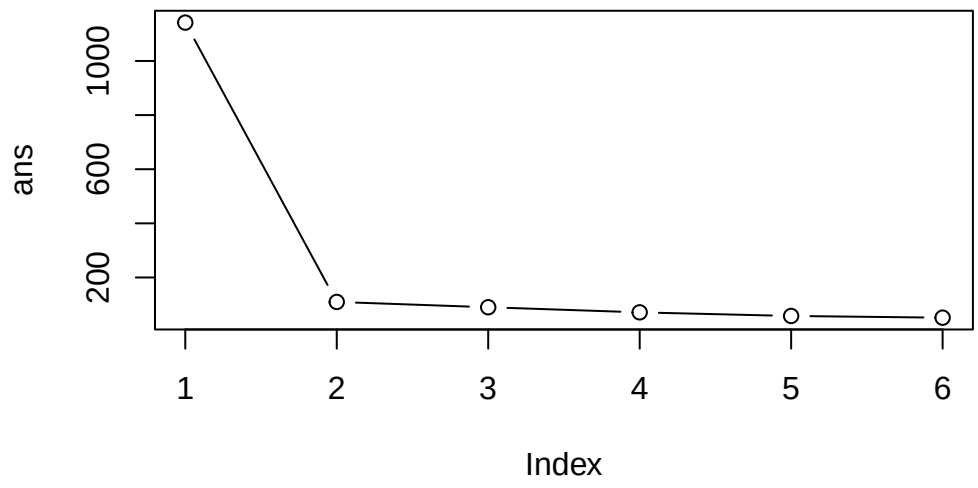
```
ans <- NULL
for(i in 1:6) {
  ans <- c(ans, kmeans(x, centers=i)$tot.withinss)
}
```

Make a "scree-plot'

```
ans
```

```
[1] 1141.87158  109.62862   90.02583   71.17855   57.82773   51.47343
```

```
plot(ans, typ="b")
```



Hierarchiacal Clustering

The main function in "base" R for this is called `hclust()`

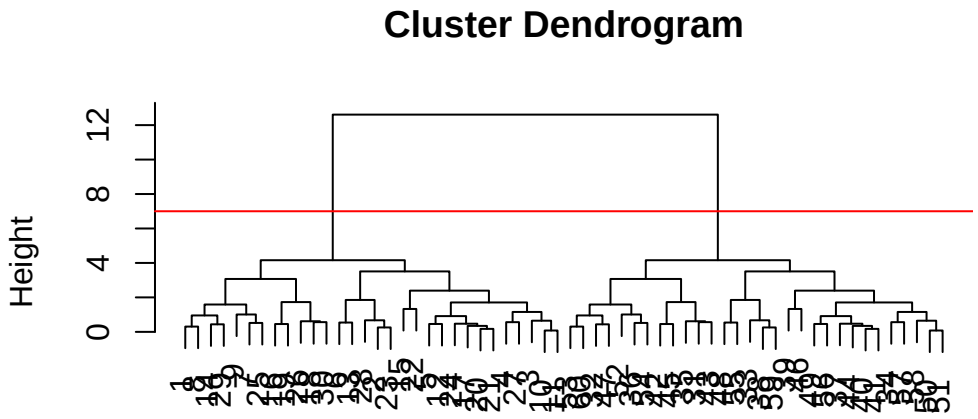
```
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

Cluster method   : complete
```

```
Distance         : euclidean
Number of objects: 60
```

```
plot(hc)
abline(h=7, col='red')
```



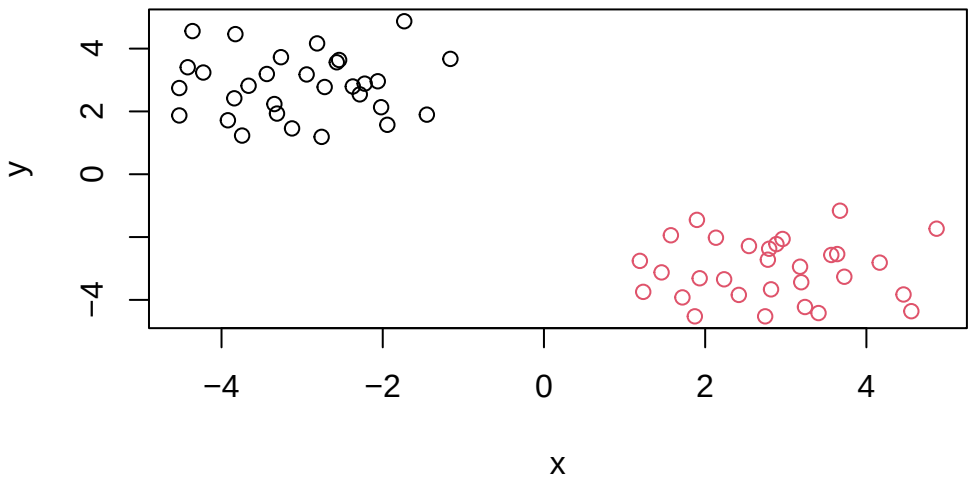
d
hclust (*, "complete")

To obtain clusters from our `hclust` result object **hc** we "cut" the tree fo yield different sub branches. For this we use the `cutree()` function.

```
grps <- cutree(hc, h=7)
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 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
```

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



Principal Componenet Analysis (PCA)

```
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
```

Preview the first 6 rows

```
head(x)
```

```
              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
```

Fix the amount of rows

```
# 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
```

Now lets check the dimensions again:

```
dim(x)
```

```
[1] 17  4
```

Side-note: An alternative approach to setting the correct row-names in this case would be to read the data file again and this time set the row.names argument of read.csv() to be the first column (i.e. use argument setting row.names=1), see below:

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

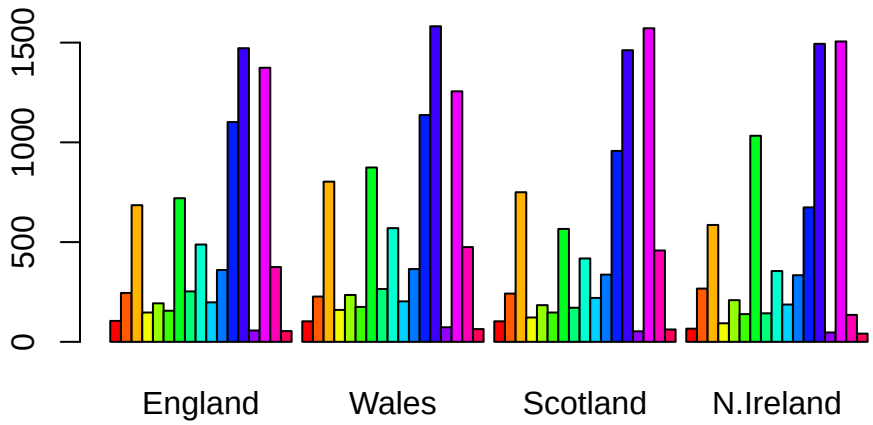
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars            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?

If you run `x <- x[,-1]` multiple times, you remove a column every time you run a code chunk. So the best way to run this is reading the data the correct way using `x <- read.csv(url, row.names=1)`

Spotting major differences and trends

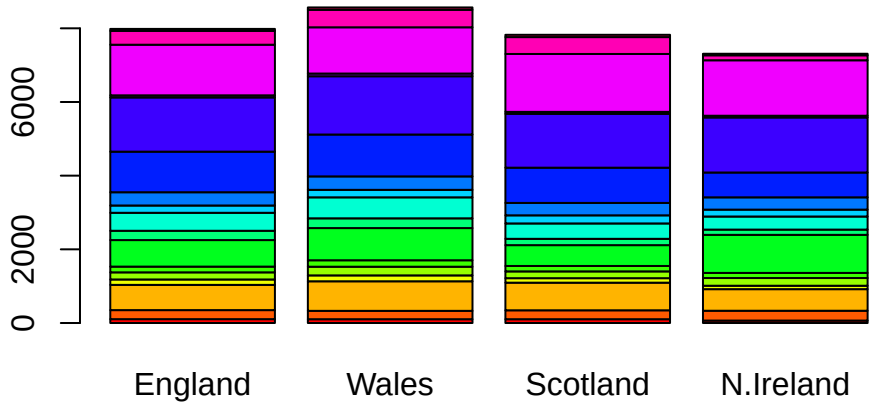
```
# Using base R
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

If you change the `beside=T` to `beside=F`

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



Currently we have wide format

Data organized with one row per Food (17 observation) and multiple columns for Country (4 different measurements across a row):

```
dim(x)
```

```
[1] 17  4
```

To convert this to "long" format we want one row per measurement - maximizes rows (17x4=68), minimizes columns (with a singe Consumption measurement value per Country). We will do tidying with the pivot_longer() function from the tidyr package:

```
library(tidyr)

# Convert data to long format for ggplot with `pivot_longer()`
x_long <- x |>
          tibble::rownames_to_column("Food") |>
          pivot_longer(cols = -Food,
                       names_to = "Country",
```

```
                        values_to = "Consumption")

dim(x_long)
```

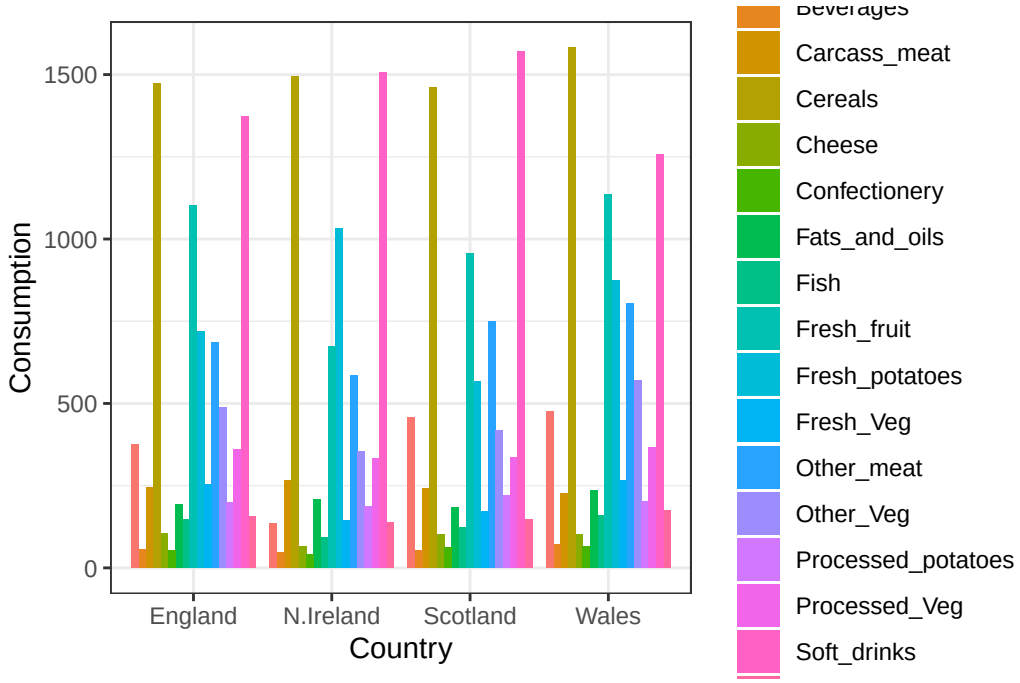
```
[1] 68  3
```

```
head(x_long)
```

```
# A tibble: 6 x 3
  Food            Country   Consumption
  <chr>           <chr>           <int>
1 "Cheese"        England           105
2 "Cheese"        Wales             103
3 "Cheese"        Scotland          103
4 "Cheese"        N.Ireland          66
5 "Carcass_meat " England           245
6 "Carcass_meat " Wales             227
```

```
# Create grouped bar plot
library(ggplot2)

ggplot(x_long) +
  aes(x = Country, y = Consumption, fill = Food) +
  geom_col(position = "dodge") +
  theme_bw()
```

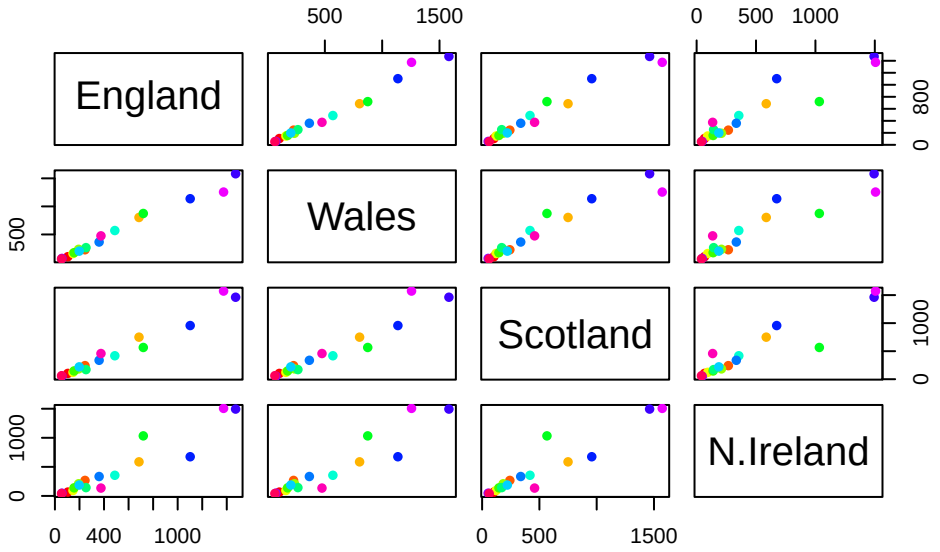


Q4: Changing what optional argument in the above ggplot() code results in a stacked barplot figure?

Q5: We can use the pairs() function to generate all pairwise plots for our countries. 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?

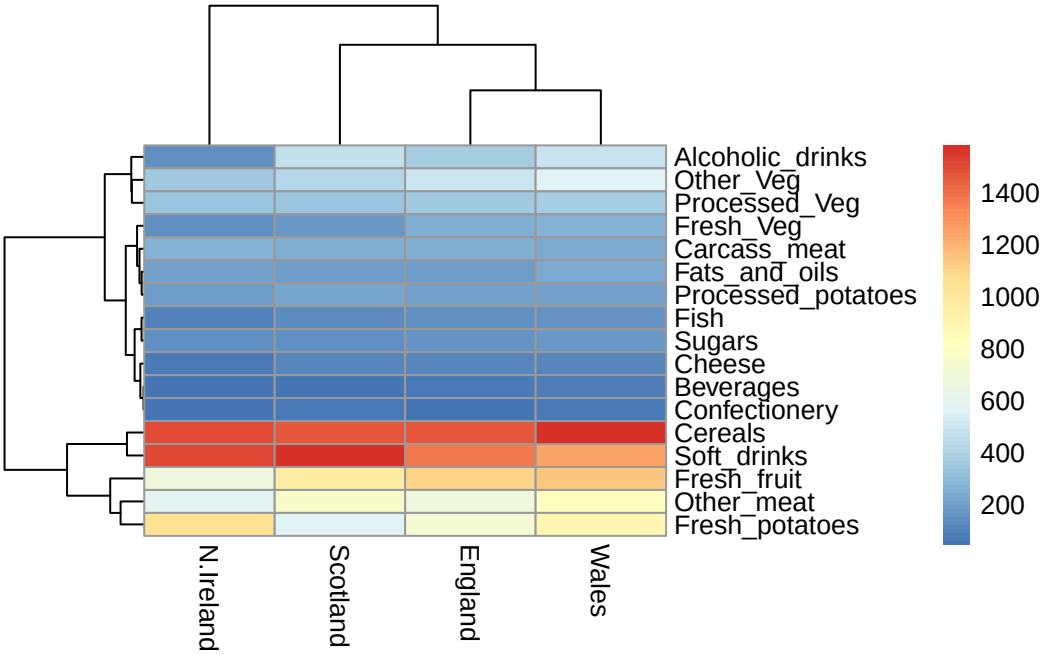
In the y axis of every plot in the first row, England is on the y-axis and Wales on the x-axis, the same pattern goes for all the rest of the rows. The points represent all the different foods. Points on the diagonal mean these foods are consumed similar between countries.

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




```
library(pheatmap)

pheatmap( as.matrix(x) )
```



> Q6. Based on the pairs and heatmap figures, which countries cluster together and what does this suggest about their food consumption patterns? Can you easily tell what the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

It looks like Wales and England are quite similar in their consumption of these foods. it is sill quite difficult to tell what is going on in the dataset.

PCA to the rescue

The main funciton in "base" R for PCA is called `prcomp()`.

As we want to do PCA on the food data for the different countries we will want the foods in the columns.

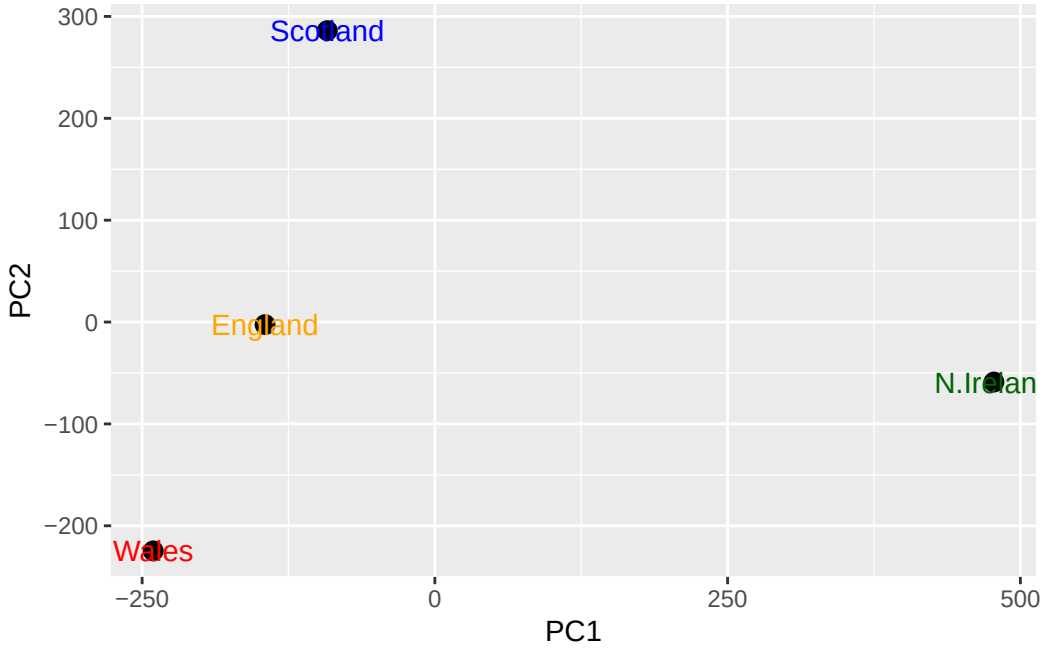
```
pca <- prcomp( t(x) )
summary(pca)
```

```
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
```

Our result object is called `pca` and it has a `$x` component that we will look at first

> 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.

```
library(ggplot2)
cols <- c("orange", "red", "blue", "darkgreen")
ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point(size = 3) +
  geom_text(col=cols)
```



We can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for.

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

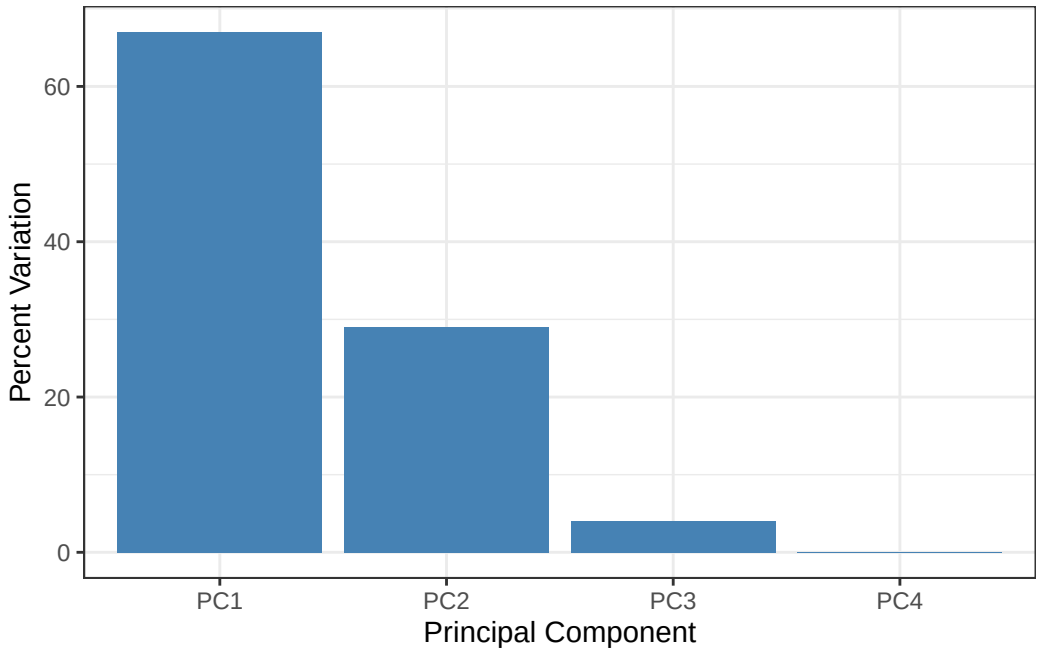
```
[1] 67 29  4  0
```

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

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 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
```

```
# Create scree plot with ggplot
variance_df <- data.frame(
  PC = factor(paste0("PC", 1:length(v)), levels = paste0("PC", 1:length(v))),
  Variance = v
)
```

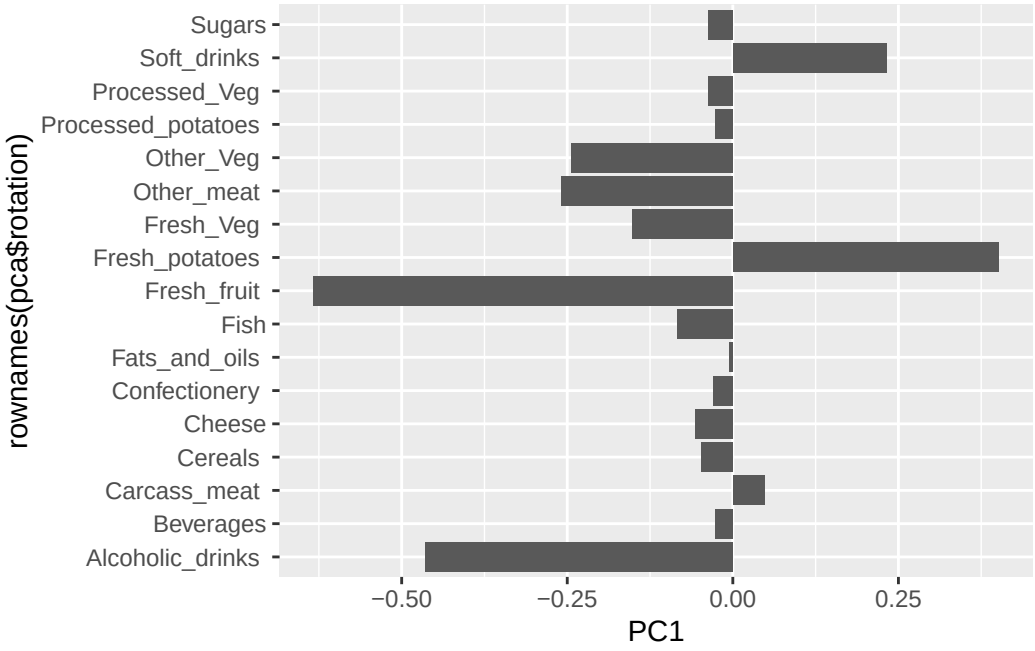
```
ggplot(variance_df) +
  aes(x = PC, y = Variance) +
  geom_col(fill = "steelblue") +
  xlab("Principal Component") +
  ylab("Percent Variation") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 0))
```



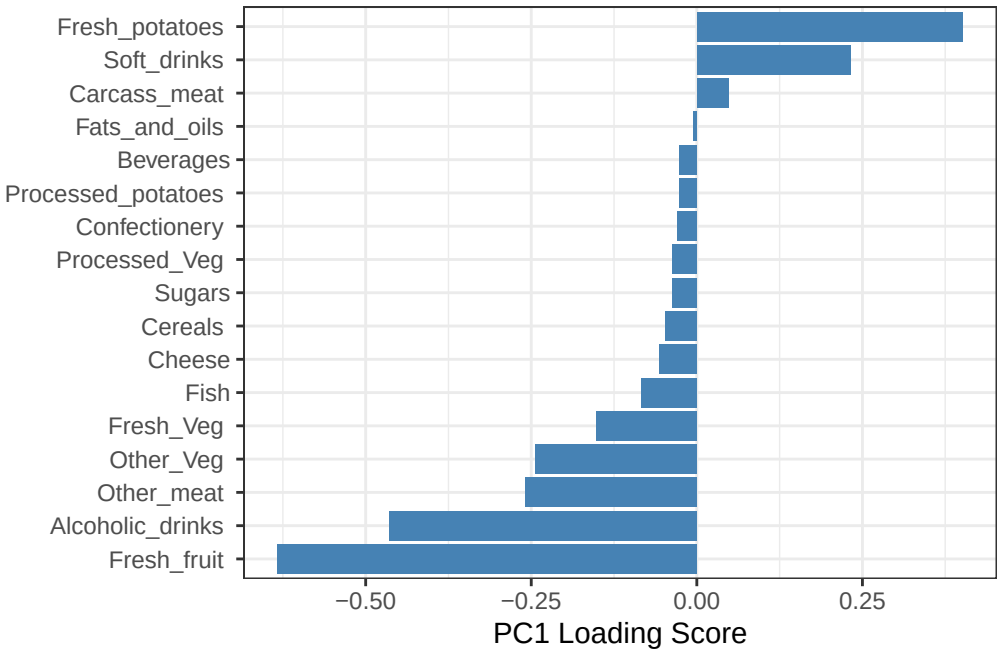
Digging deeper (variable loadings)

Another major result out of PCA is the so-called "variable loadings" or `$rotation` that tells us how the original variables (foods) contribute to PCs (i.e. our new axis)

```
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
  geom_col()
```



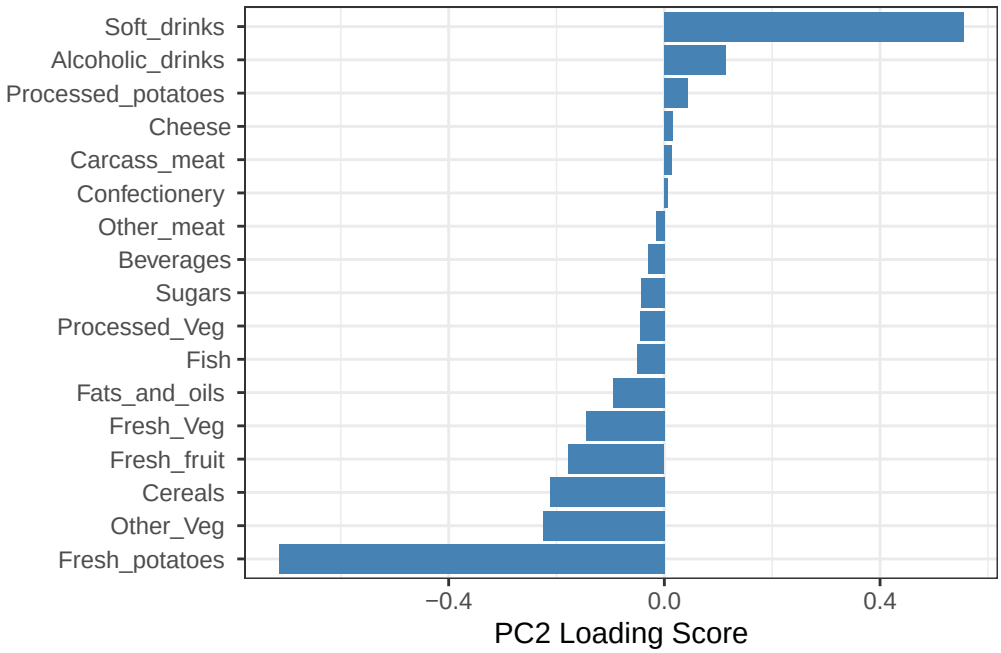

```
## Lets focus on PC1 as it accounts for > 90% of variance
ggplot(pca$rotation) +
  aes(x = PC1,
      y = reorder(rownames(pca$rotation), PC1)) +
  geom_col(fill = "steelblue") +
  xlab("PC1 Loading Score") +
  ylab("") +
  theme_bw() +
  theme(axis.text.y = element_text(size = 9))
```



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

We can see when we plot PC2 there are two outliers, one being for fresh potatoes and the other soft drinks

```
ggplot(pca$rotation) +
  aes(x = PC2,
      y = reorder(rownames(pca$rotation), PC2)) +
  geom_col(fill = "steelblue") +
  xlab("PC2 Loading Score") +
  ylab("") +
  theme_bw() +
  theme(axis.text.y = element_text(size = 9))
```



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?

There are 100 rows, and 10 columns

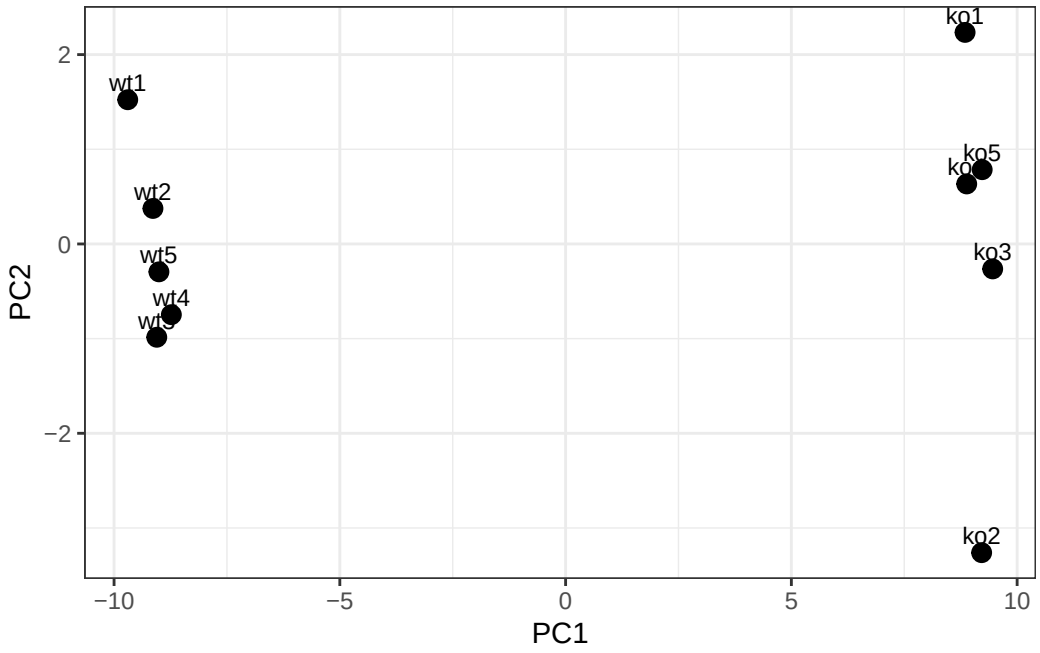
```
dim(rna.data)
```

```
[1] 100  10
```

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

# Create data frame for plotting
df <- as.data.frame(pca$x)
df$Sample <- rownames(df)

## Plot with ggplot
ggplot(df) +
  aes(x = PC1, y = PC2, label = Sample) +
  geom_point(size = 3) +
  geom_text(vjust = -0.5, size = 3) +
  xlab("PC1") +
  ylab("PC2") +
  theme_bw()
```




```
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
```

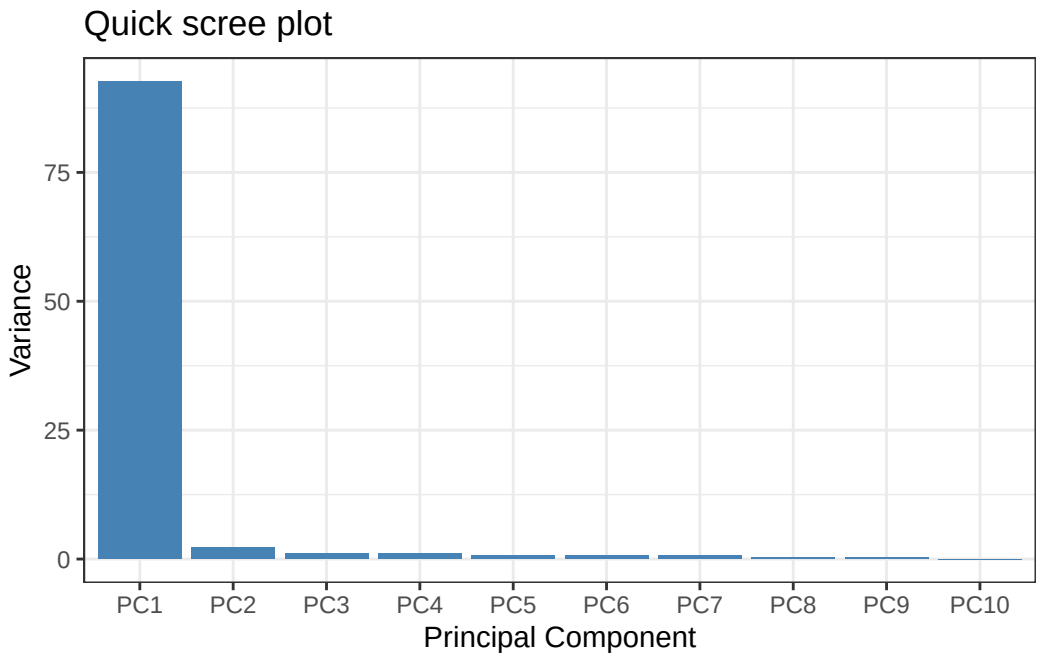
```
Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
                           PC8     PC9    PC10
Standard deviation     0.62065 0.60342 3.3e-15
Proportion of Variance 0.00385 0.00364 0.0e+00
Cumulative Proportion  0.99636 1.00000 1.0e+00
```

A quick scree plot summary of this Proportion of Variance for each PC can be obtained using ggplot:

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

# Create scree plot data
scree_df <- data.frame(
  PC = factor(paste0("PC", 1:10), levels = paste0("PC", 1:10)),
  Variance = pca.var[1:10]
)

ggplot(scree_df) +
  aes(x = PC, y = Variance) +
  geom_col(fill = "steelblue") +
  ggtitle("Quick scree plot") +
  xlab("Principal Component") +
  ylab("Variance") +
  theme_bw()
```

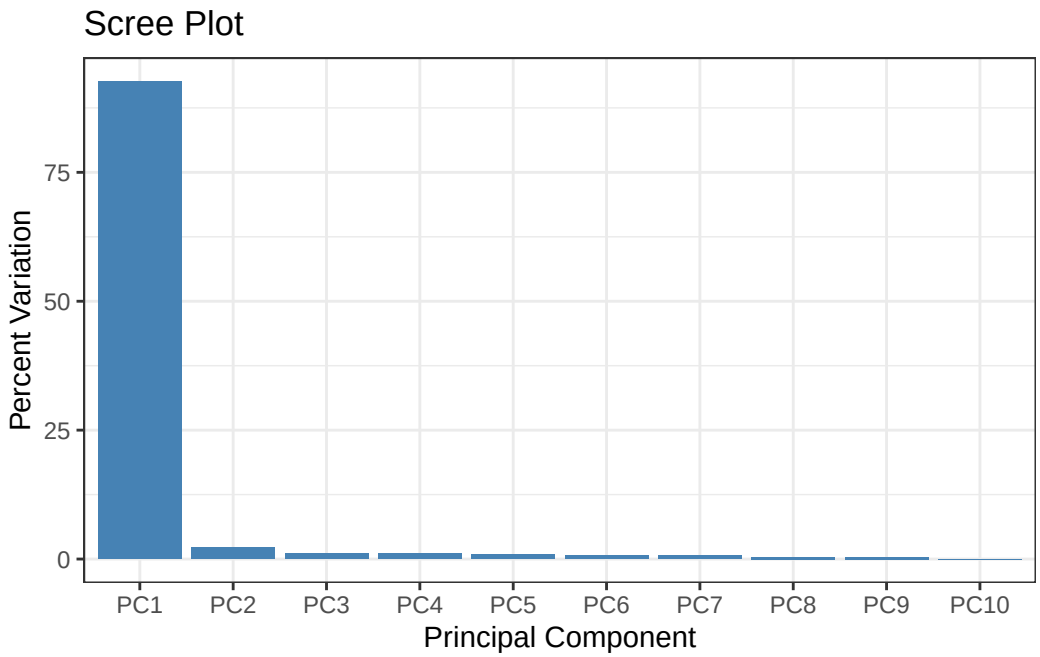



```
## Percent variance is often more informative to look at
pca.var.per
```

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

```
# Create percent variance scree plot
scree_pct_df <- data.frame(
  PC = factor(paste0("PC", 1:10), levels = paste0("PC", 1:10)),
  PercentVariation = pca.var.per[1:10]
)

ggplot(scree_pct_df) +
  aes(x = PC, y = PercentVariation) +
  geom_col(fill = "steelblue") +
  ggtitle("Scree Plot") +
  xlab("Principal Component") +
  ylab("Percent Variation") +
  theme_bw()
```

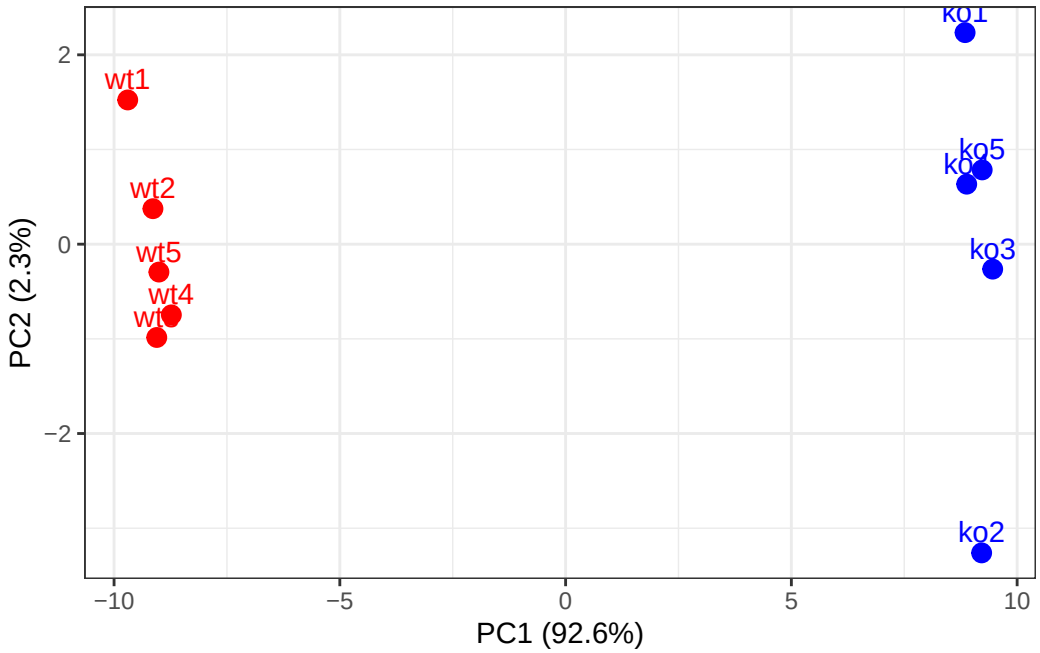


A better looking graph:

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

# Add condition to data frame
df$condition <- substr(df$Sample, 1, 2)
df$color <- colvec

ggplot(df) +
  aes(x = PC1, y = PC2, color = color, label = Sample) +
  geom_point(size = 3) +
  geom_text(vjust = -0.5, hjust = 0.5, show.legend = FALSE) +
  scale_color_identity() +
  xlab(paste0("PC1 (", pca.var.per[1], "%)")) +
  ylab(paste0("PC2 (", pca.var.per[2], "%)")) +
  theme_bw()
```



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"
```