Class 7: Machine Learning 1

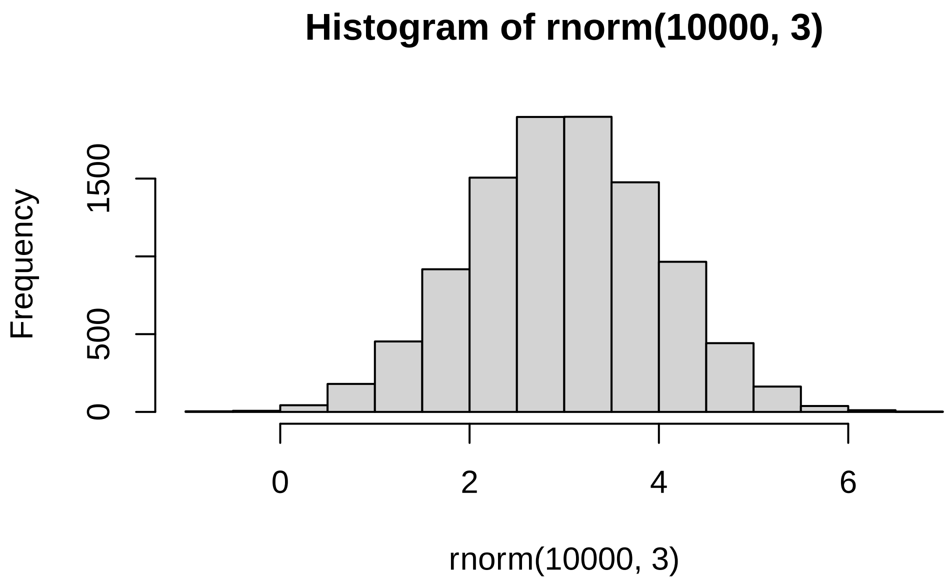
Alexander Liu (PID: 69026918)

Clustering

We will start with k-mneans clustering, one of the most prevelent of all clustering methods.

To get started let's make some data up:

```r
hist(rnorm(10000, 3))
```



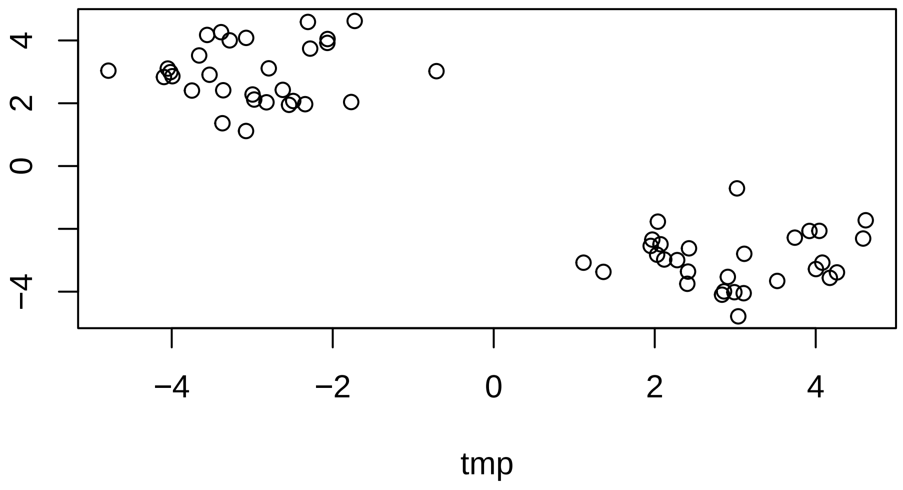
```r
tmp <- c(rnorm(30,3), rnorm(30, -3))
x <- cbind(tmp, rev(tmp))
x
```

```
            tmp
```

```
 [1,]  3.5216401 -3.6592951
 [2,]  2.4133656 -3.3605628
 [3,]  2.8358769 -4.0969981
 [4,]  2.8609877 -3.9922001
 [5,]  1.9699756 -2.3427514
 [6,]  3.1121443 -2.7947423
 [7,]  2.0715735 -2.4909884
 [8,]  3.0217256 -0.7112393
 [9,]  3.9218650 -2.0669755
[10,]  4.5889129 -2.3083050
[11,]  4.2635915 -3.3872075
[12,]  2.4253799 -2.6208772
[13,]  4.1752198 -3.5603367
[14,]  4.0816219 -3.0756043
[15,]  2.0386815 -1.7707004
[16,]  3.7402409 -2.2805179
[17,]  2.4043948 -3.7490486
[18,]  1.9497388 -2.5429718
[19,]  4.0450115 -2.0644694
[20,]  2.9891540 -4.0173051
[21,]  1.3627591 -3.3703716
[22,]  3.1047655 -4.0492415
[23,]  2.0295845 -2.8232469
[24,]  4.0025919 -3.2799778
[25,]  2.2786689 -2.9960491
[26,]  3.0375333 -4.7868841
[27,]  2.1179152 -2.9755287
[28,]  4.6209648 -1.7272890
[29,]  1.1153581 -3.0771897
[30,]  2.9077040 -3.5304005
[31,] -3.5304005  2.9077040
[32,] -3.0771897  1.1153581
[33,] -1.7272890  4.6209648
[34,] -2.9755287  2.1179152
[35,] -4.7868841  3.0375333
[36,] -2.9960491  2.2786689
[37,] -3.2799778  4.0025919
[38,] -2.8232469  2.0295845
[39,] -4.0492415  3.1047655
[40,] -3.3703716  1.3627591
[41,] -4.0173051  2.9891540
[42,] -2.0644694  4.0450115
[43,] -2.5429718  1.9497388
```

```
[44,] -3.7490486  2.4043948
[45,] -2.2805179  3.7402409
[46,] -1.7707004  2.0386815
[47,] -3.0756043  4.0816219
[48,] -3.5603367  4.1752198
[49,] -2.6208772  2.4253799
[50,] -3.3872075  4.2635915
[51,] -2.3083050  4.5889129
[52,] -2.0669755  3.9218650
[53,] -0.7112393  3.0217256
[54,] -2.4909884  2.0715735
[55,] -2.7947423  3.1121443
[56,] -2.3427514  1.9699756
[57,] -3.9922001  2.8609877
[58,] -4.0969981  2.8358769
[59,] -3.3605628  2.4133656
[60,] -3.6592951  3.5216401
```

```r
plot(x)
```



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

```r
k <-  kmeans(x, centers=2, nstart=20)
k
```

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

```
Cluster means:
        tmp
1 -2.983643  2.966965
2  2.966965 -2.983643

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 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] 48.40206 48.40206
 (between_SS / total_SS =  91.6 %)

Available components:

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

> Q1. How many points are in each cluster

```r
k$size
```

```
[1] 30 30
```

> Q2. The clustering result i.e. membership vector?

```r
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 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
```

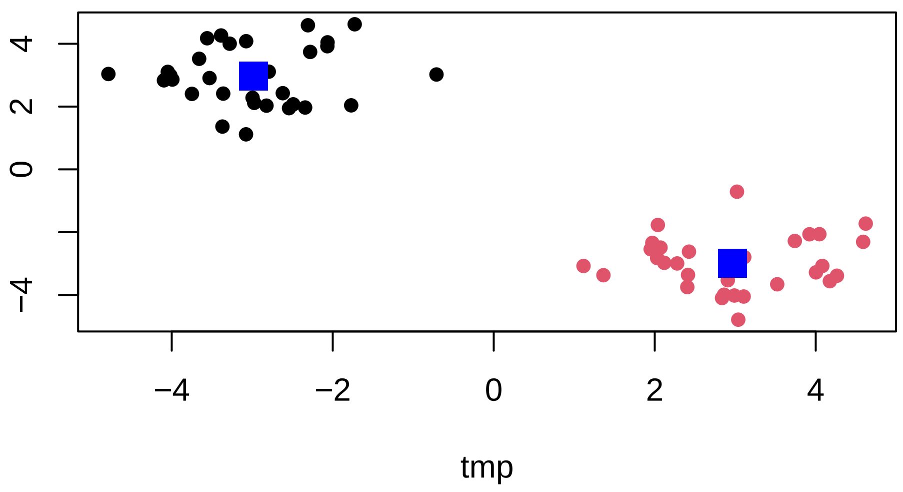
> Q3. Cluster centers

```r
k$centers
```

```
        tmp
1 -2.983643  2.966965
2  2.966965 -2.983643
```

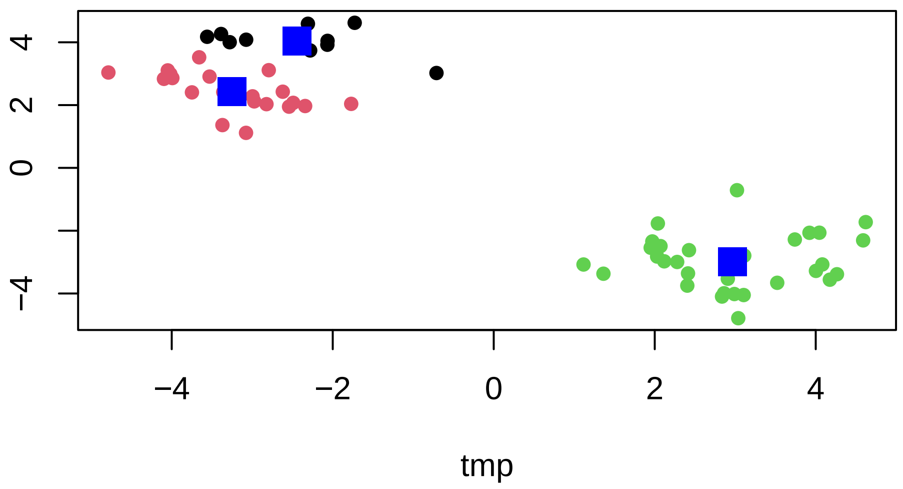
> Q4. Make a plot of our data colored by clustering results with optionally the cluster centers shown.

```r
plot(x, col=k$cluster, pch=16)
points(k$centers, col="blue", pch=15, cex=2)
```



Q5. Run kmeans again but cluter into 3 groups and plot the results like we did above.

```r
l <-  kmeans(x, centers=3, nstart=20)
plot(x, col=l$cluster, pch=16)
points(l$centers, col="blue", pch=15, cex=2)
```



K-means will always return a clustering result - even if there is no clear groupings.

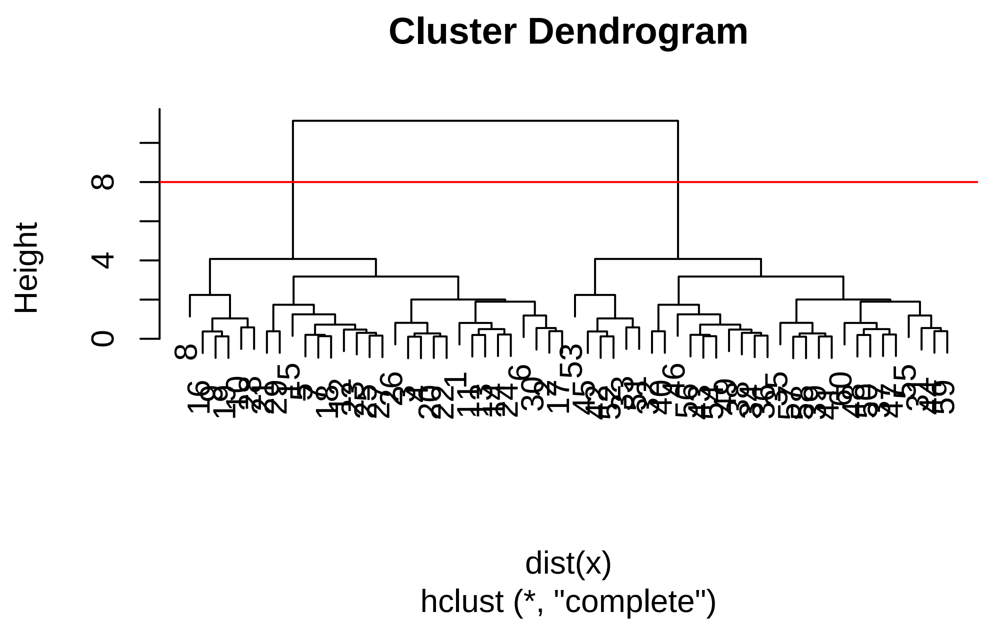
#Hierarchical Clustering

main function: `hclust()`

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

```

Call:
hclust(d = dist(x))

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

```
plot(hc)
abline(h=8, col="red")
```



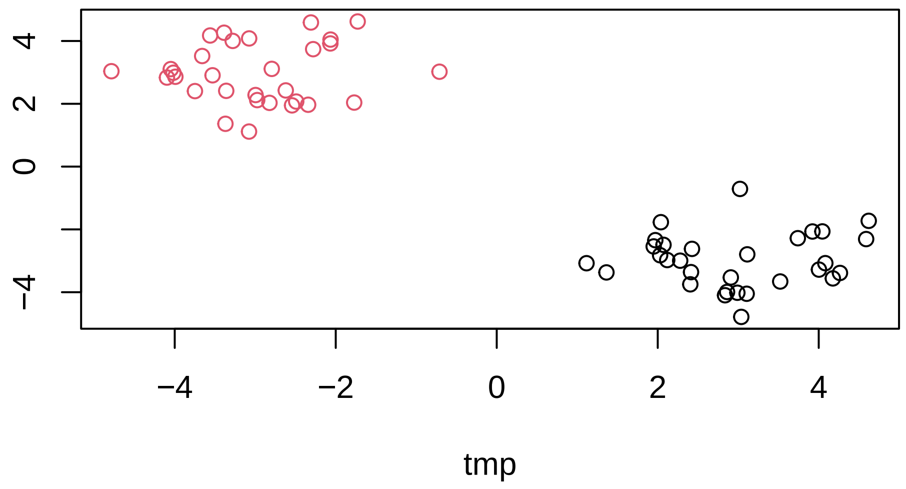
The function to get our clusters/groups from a hclust object is called `cutree()`

```
grps <- cutree(hc, h=8)
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 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. Plot our hclust results in terms of our data colored by cluster membership.

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



Principal Component Analysis (PCA)

Class 7 Lab

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

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

dim(x)
```

```
[1] 17  5
```

```r
## 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
```

```r
# 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
```

```r
dim(x)
```

```
[1] 17  4
```

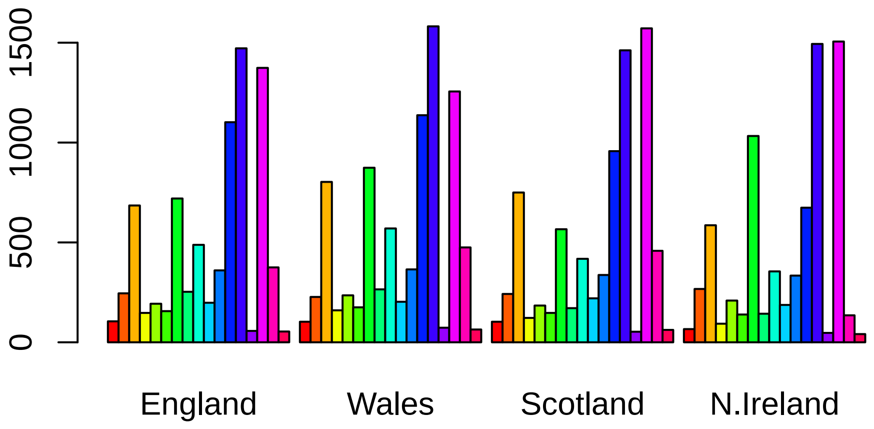
```r
x <- read.csv(url, row.names=1)
head(x)
```

```
               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?

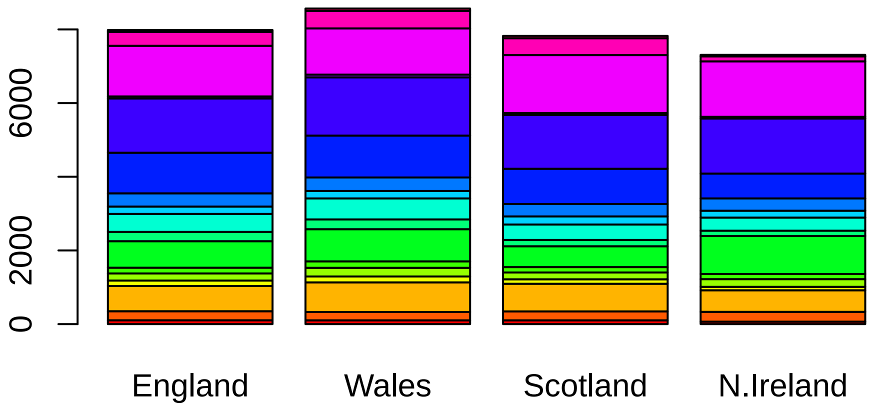
I prefer the latter (row.names=1). If you accidentally run the former code, it could overwrite your processed dataset.

```
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?

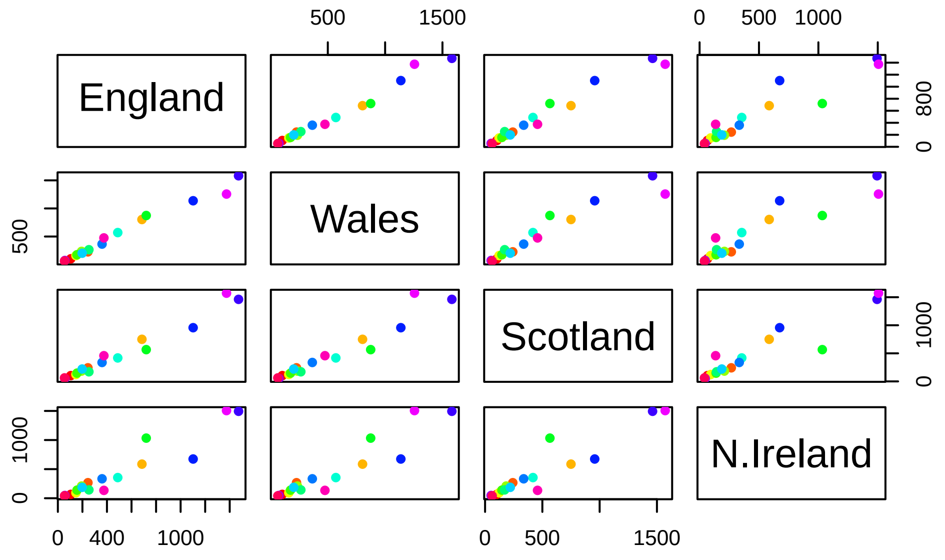
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)),)
```



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?

Lying on the diagonal means the value between two countries are similalr.

```
pairs(x, col=rainbow(17), pch=16)
```



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

There are more plots that are not on the diagonal, which indicates N.Ireland is more dissimilar to other countries.

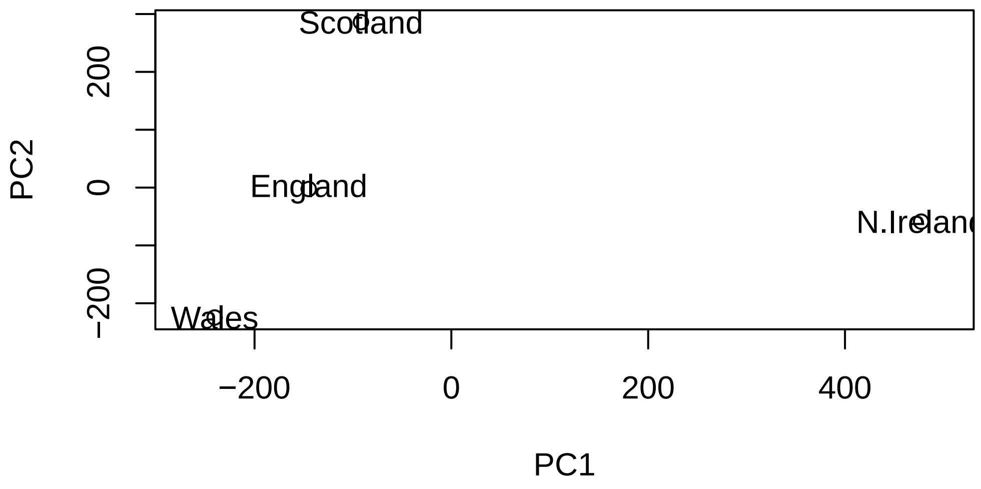
PCA to the rescue

```
# Use the prcomp() PCA function
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
```

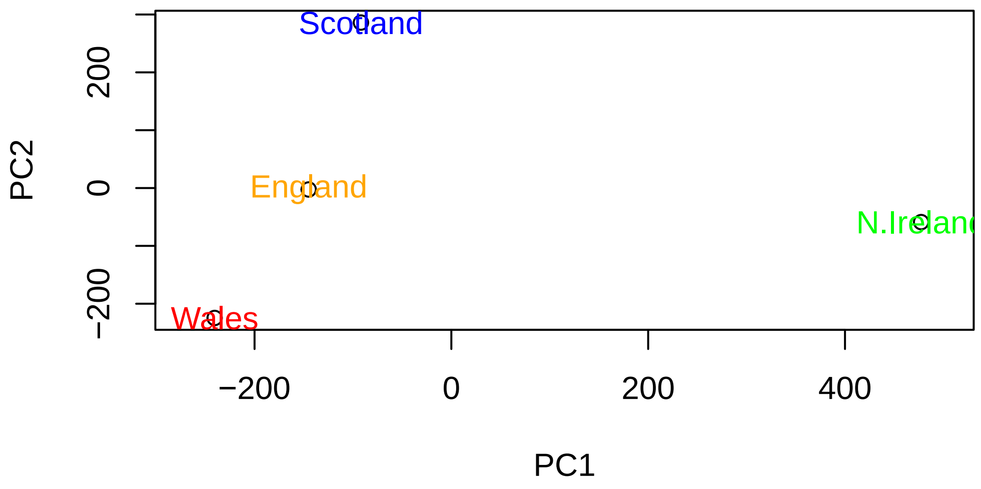
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], 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.

```
colors=c("orange", "red", "blue", "green")
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=colors)
```



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

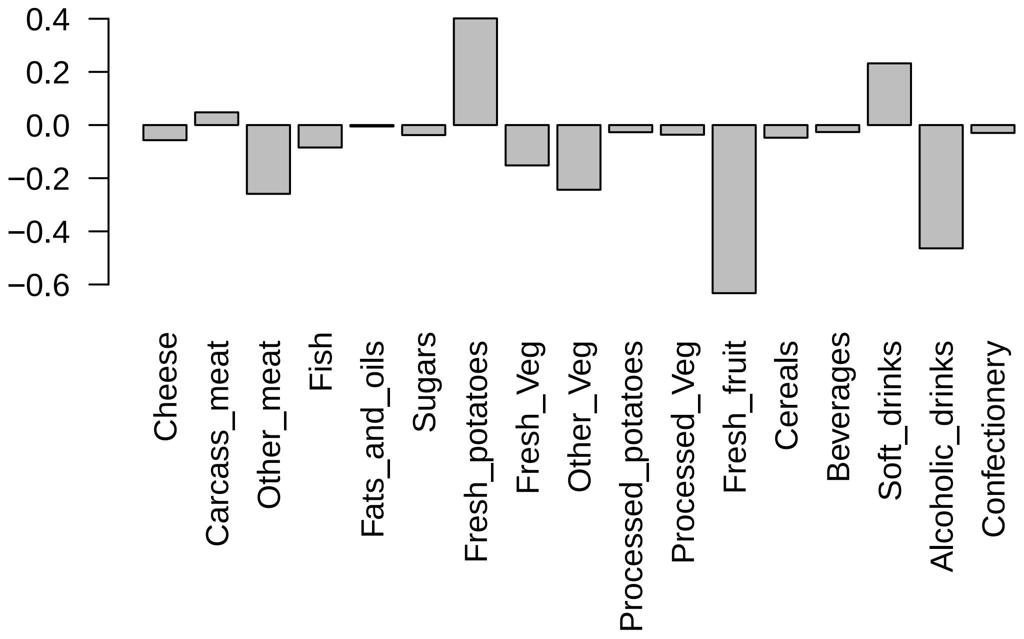
```
[1] 67 29  4  0
```

```
## 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
```

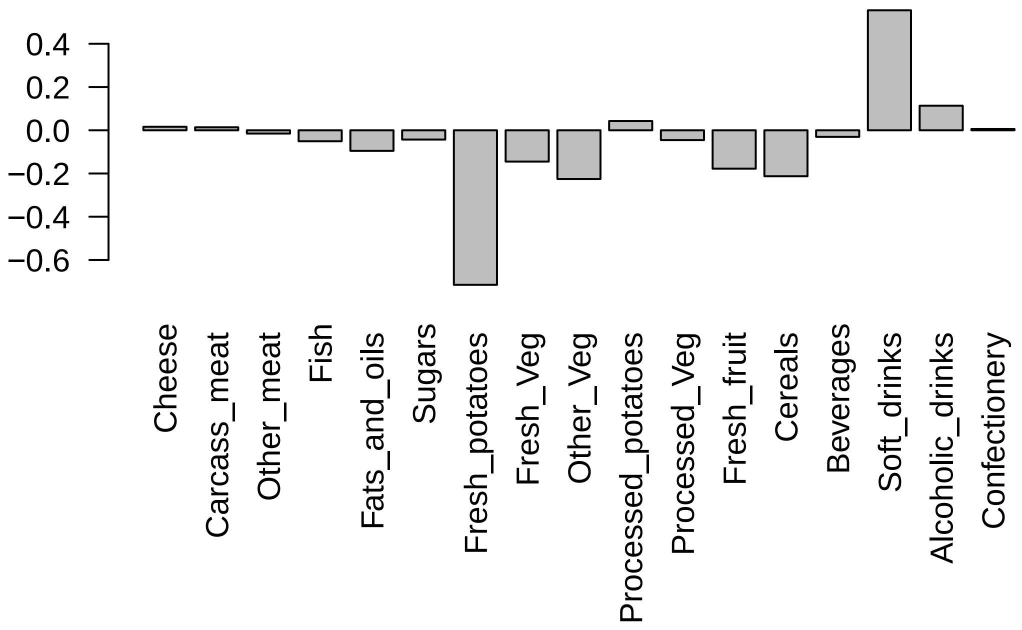
Digging deeper (variable loadings)

```
## 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?

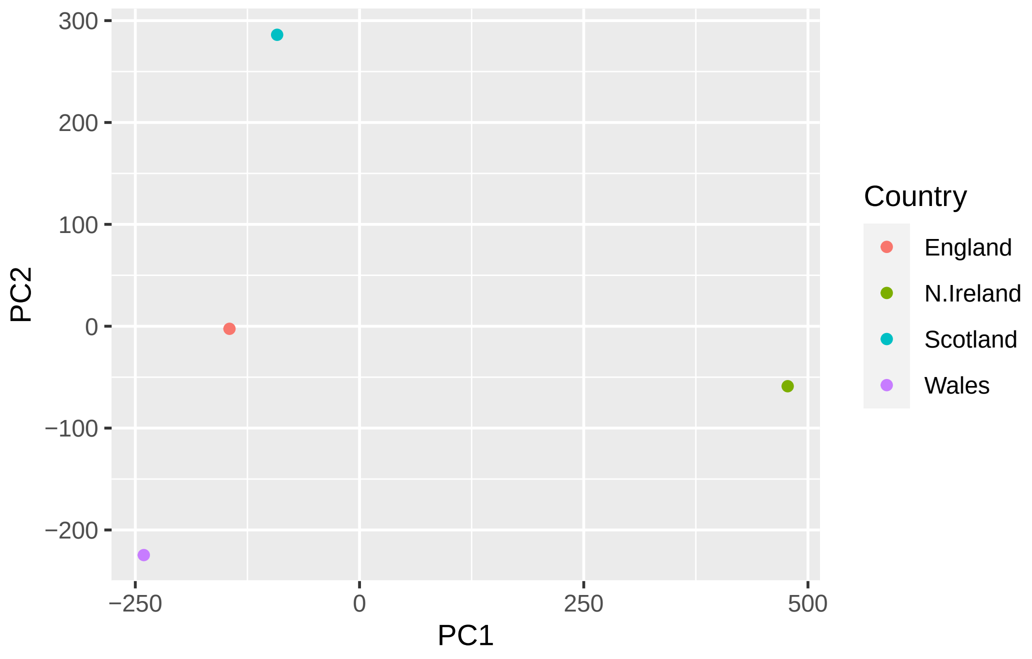
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



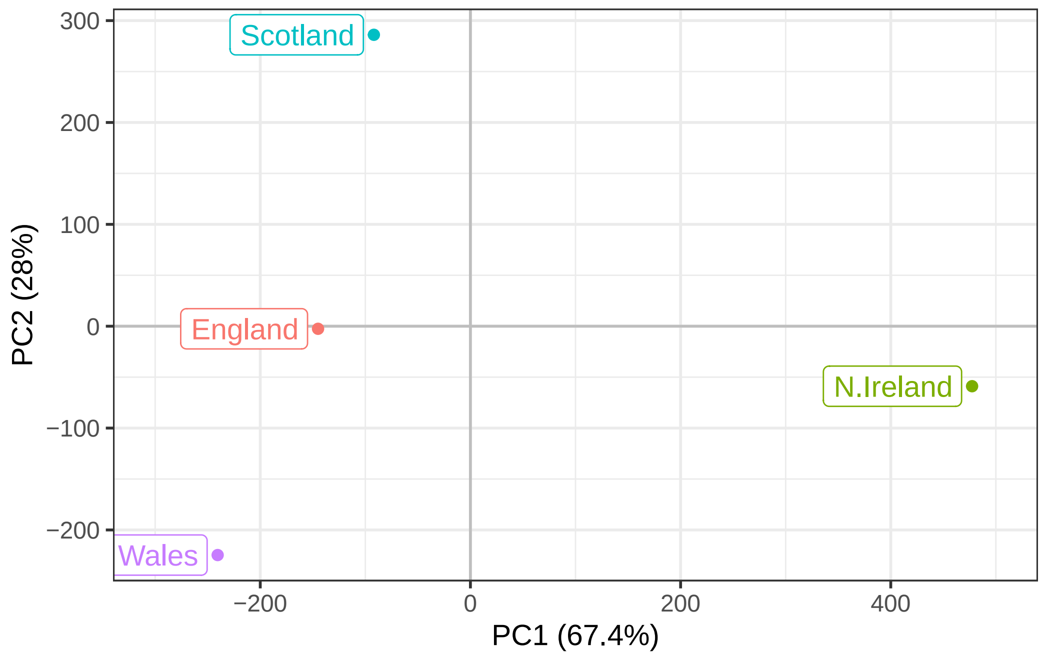
Fresh_potatoes and Soft_drinks. They are the main drivers to "push" Walse and Scotland to negative or positive side, respectively.

Using ggplot for these figures

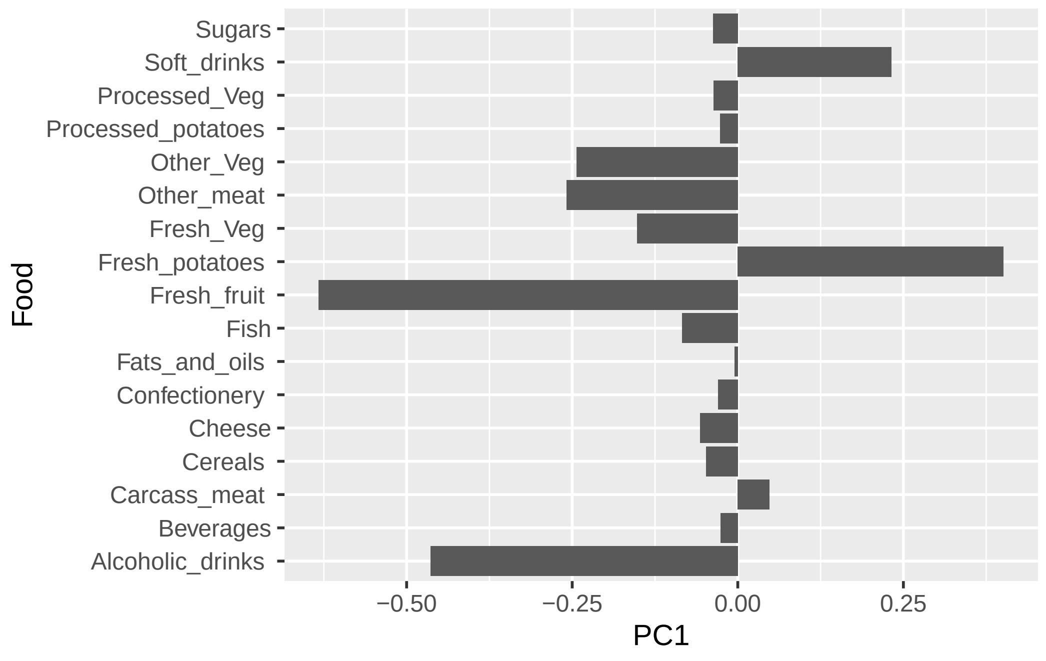
```
library(ggplot2)

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

# Our first basic plot
ggplot(df_lab) +
  aes(PC1, PC2, col=Country) +
  geom_point()
```



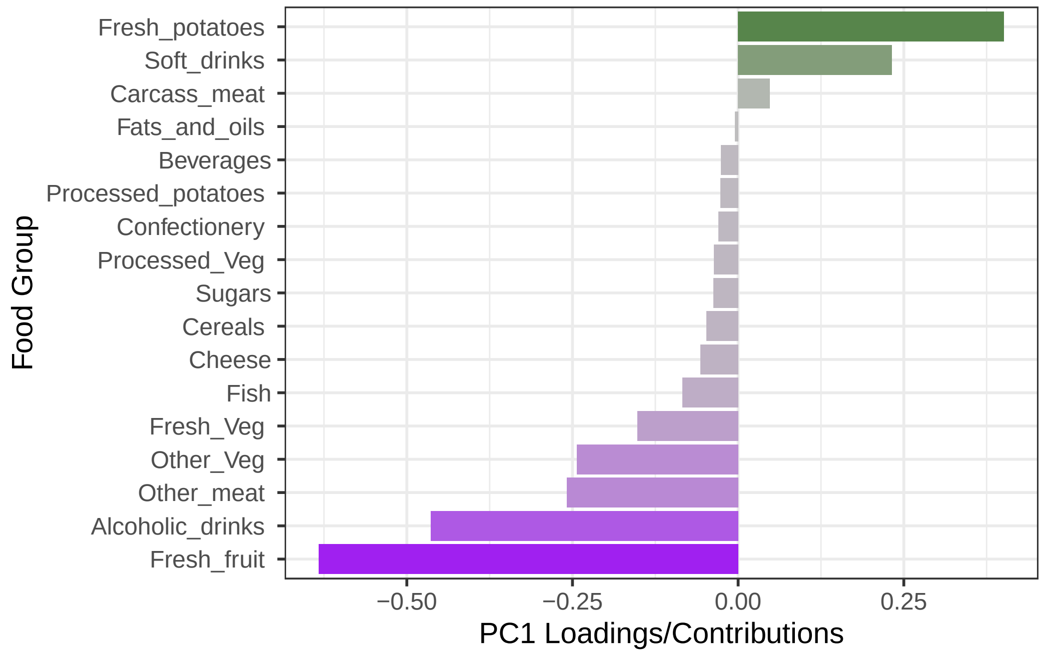
```r
ggplot(df_lab) +
  aes(PC1, PC2, col=Country, label=Country) +
  geom_hline(yintercept = 0, col="gray") +
  geom_vline(xintercept = 0, col="gray") +
  geom_point(show.legend = FALSE) +
  geom_label(hjust=1, nudge_x = -10, show.legend = FALSE) +
  expand_limits(x = c(-300,500)) +
  xlab("PC1 (67.4%)")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



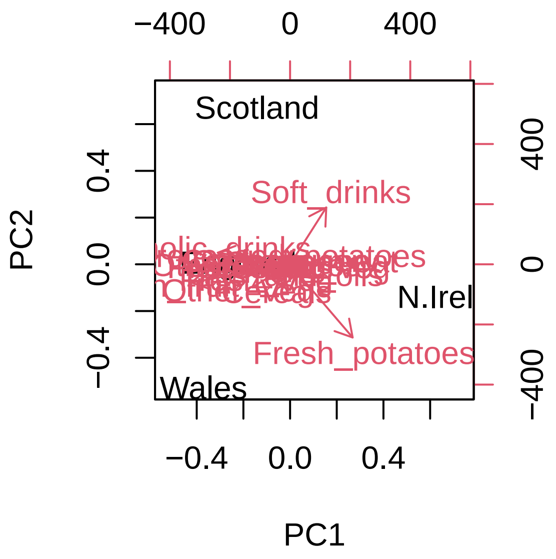
```
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

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()
```



```r
## The inbuilt biplot() can be useful for small datasets
biplot(pca)
```



2. PCA of RNA-seq data

```r
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
```

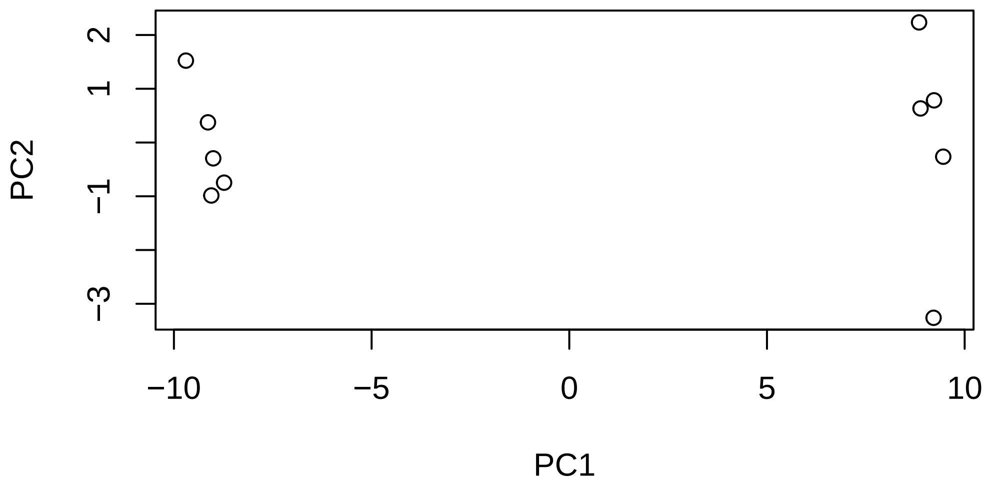
Q10: How many genes and samples are in this data set?

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

```
[1] 100
```

100 genes

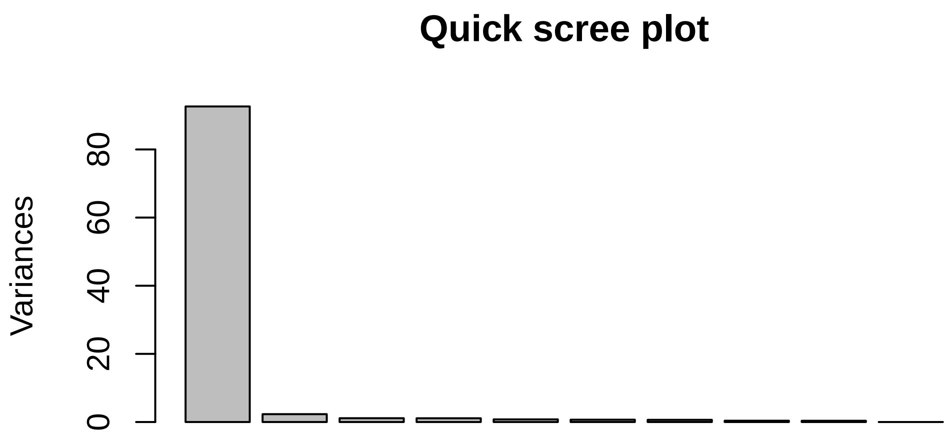
```r
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

## 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
```

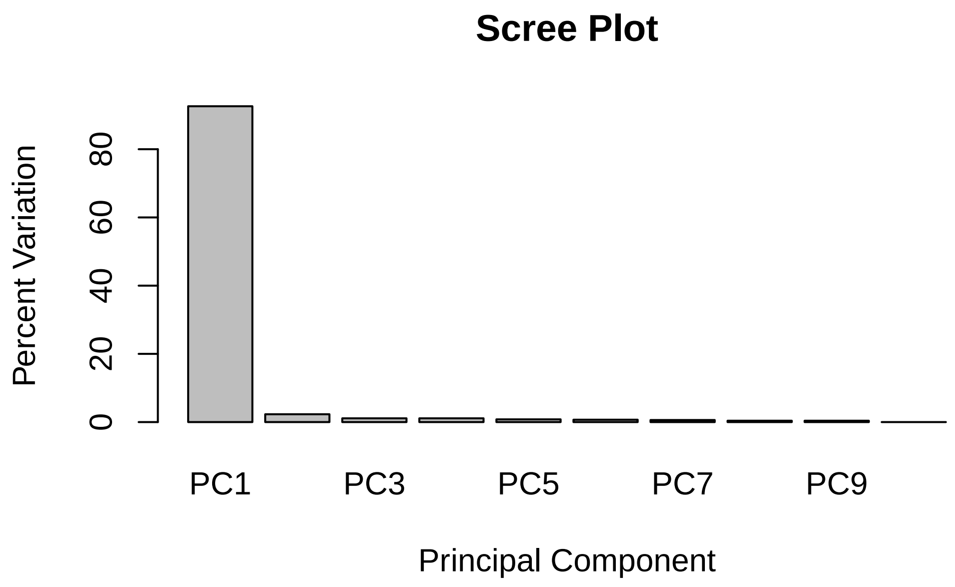
```r
plot(pca, main="Quick scree plot")
```



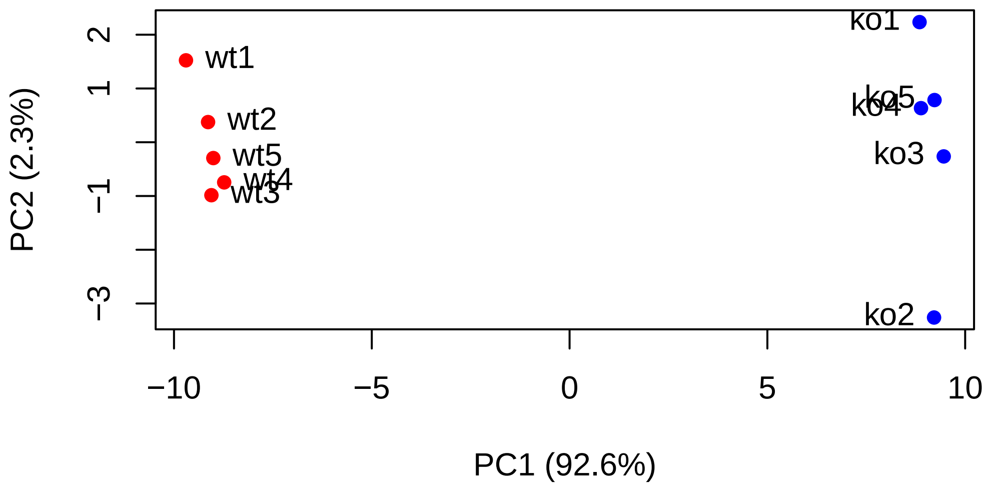
```r
## Variance captured per PC
pca.var <- pca$sdev^2

## 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
```

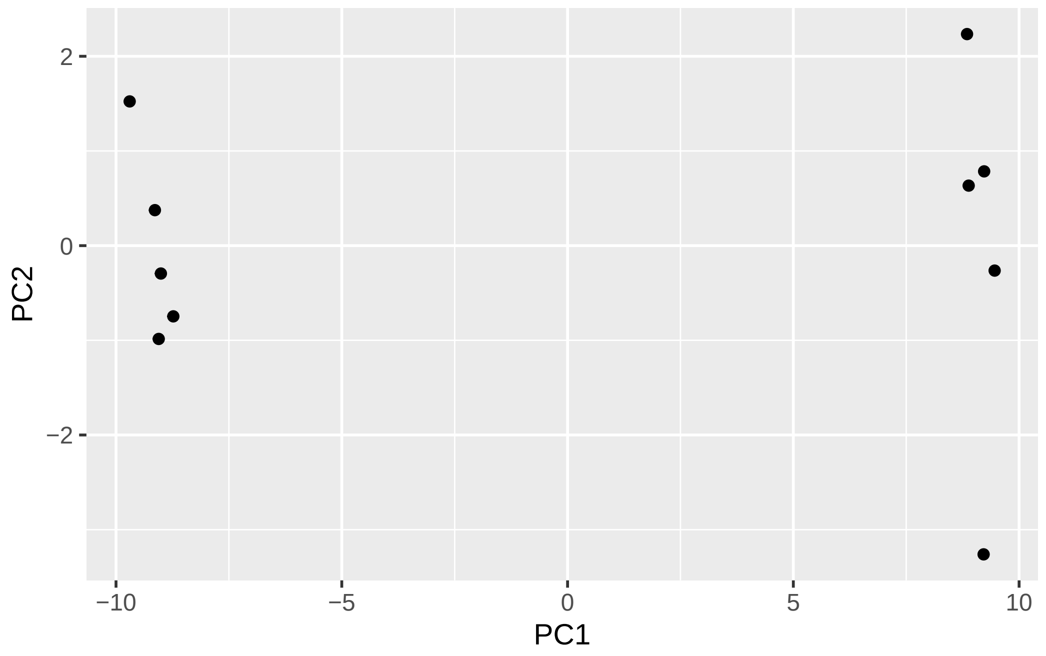
```r
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
          xlab="Principal Component", ylab="Percent Variation")
```



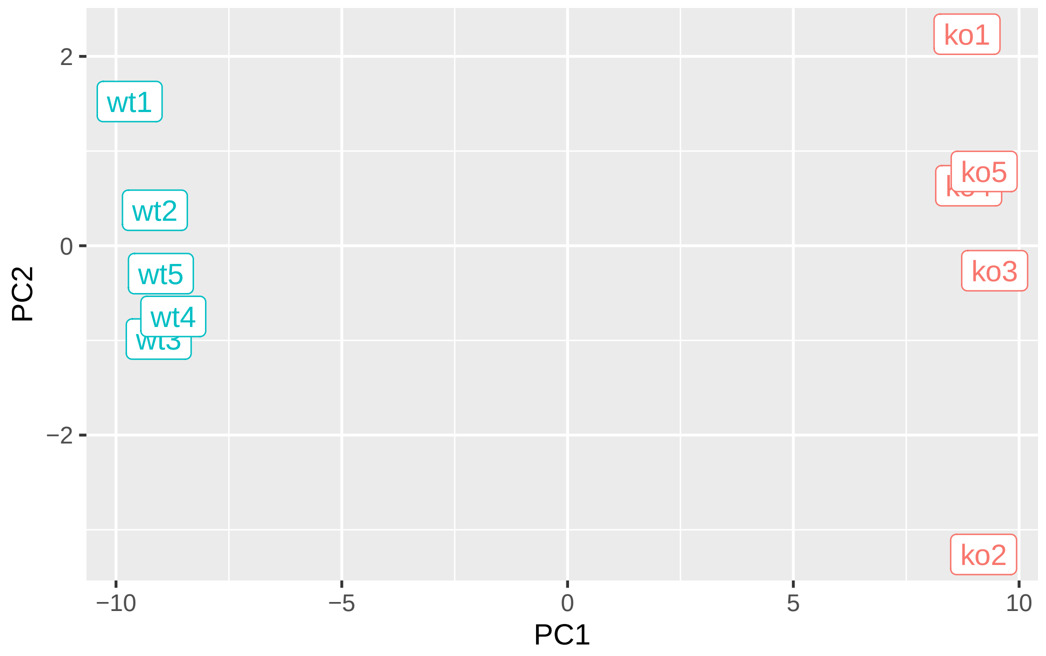
```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```



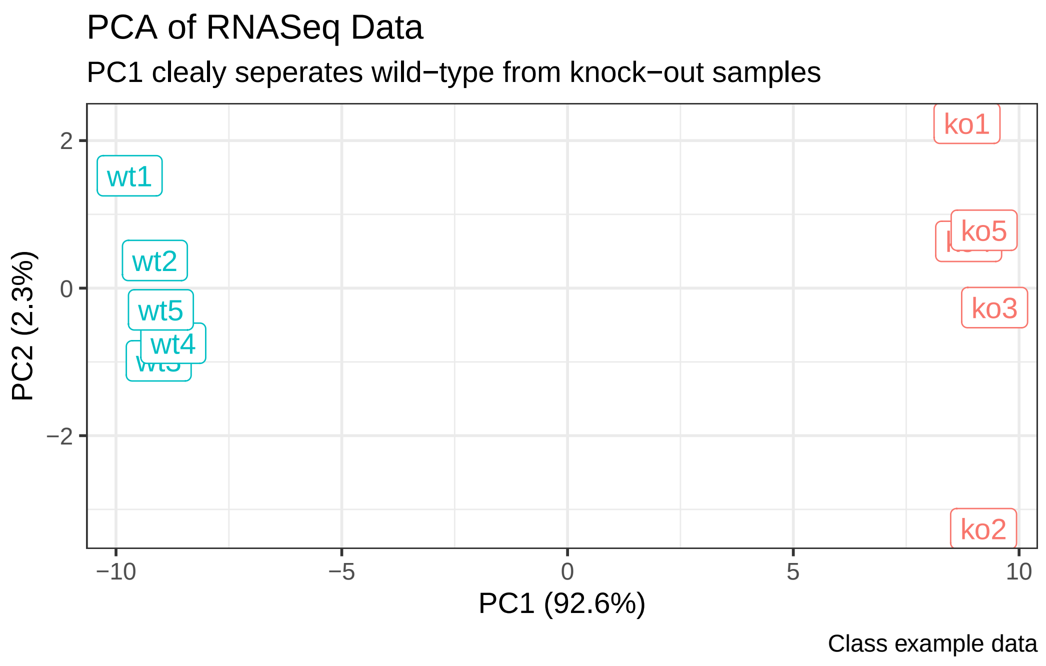
```r
library(ggplot2)

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

# Our first basic plot
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



```r
# 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
```



```r
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()
```



```r
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" 
```

```r
sessionInfo()
```

```
R version 4.3.1 (2023-06-16)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.5

Matrix products: default
```

```
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ggplot2_3.4.4

loaded via a namespace (and not attached):
 [1] vctrs_0.6.4       cli_3.6.1         knitr_1.44        rlang_1.1.1
 [5] xfun_0.40         generics_0.1.3    jsonlite_1.8.7    labeling_0.4.3
 [9] glue_1.6.2        colorspace_2.1-0  htmltools_0.5.6.1 scales_1.2.1
[13] fansi_1.0.5       rmarkdown_2.25    grid_4.3.1        evaluate_0.22
[17] munsell_0.5.0     tibble_3.2.1      fastmap_1.1.1     yaml_2.3.7
[21] lifecycle_1.0.3   compiler_4.3.1    dplyr_1.1.3       pkgconfig_2.0.3
[25] farver_2.1.1      digest_0.6.33     R6_2.5.1          tidyselect_1.2.0
[29] utf8_1.2.4        pillar_1.9.0      magrittr_2.0.3    withr_2.5.1
[33] tools_4.3.1       gtable_0.3.4
```