Machine Learning 1

Anita Wang (PID: A15567878)

10/21/2021

First up is clustering methods

Kmeans clustering

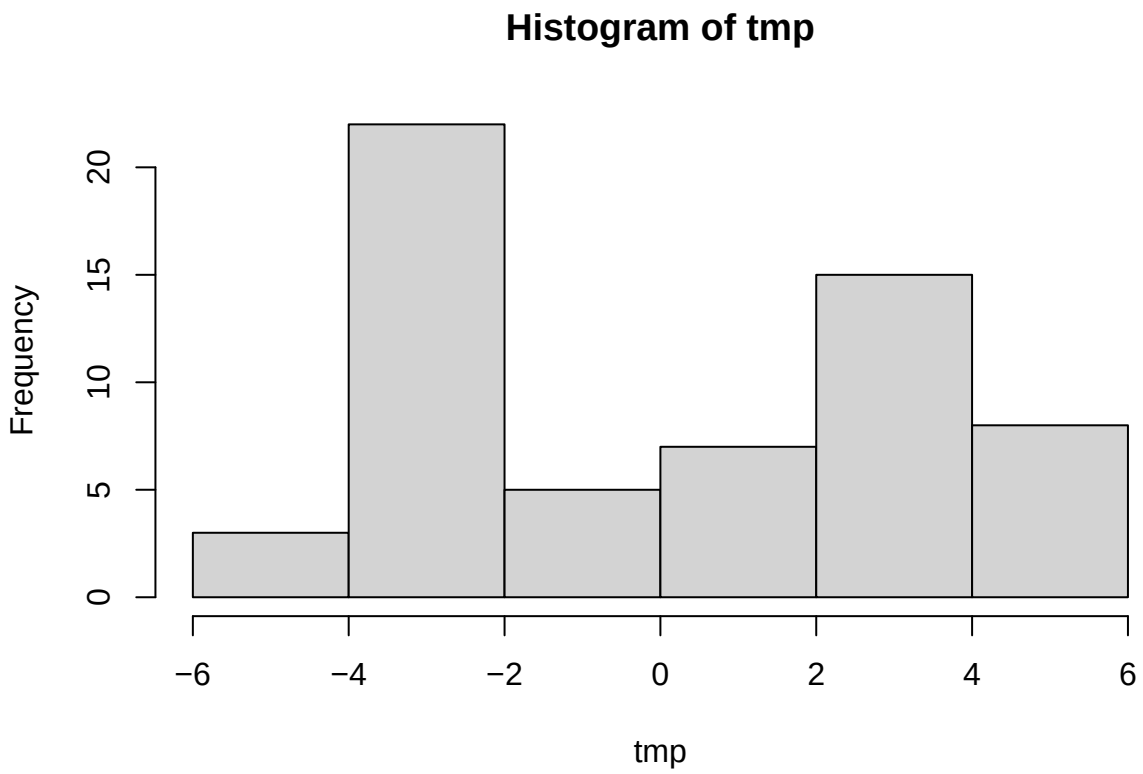
The funciton in base R to do Kmeans clustering is called `kmeans()`

First make up some data where we know what the answer should be:

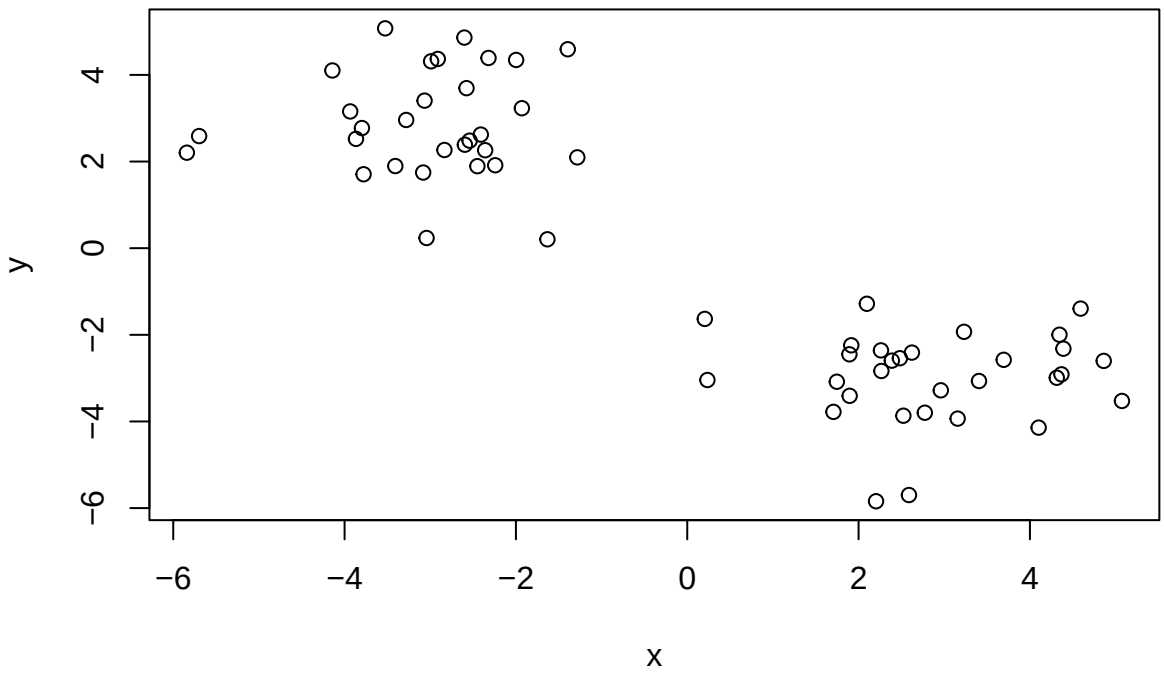
Use rnorm(): Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Using rbind() or cbind(): Combine R Objects by Rows or Columns

```
tmp <- c(rnorm(30,-3), rnorm(30,3))
hist(tmp)
```




```
x <- cbind(x=tmp, y=rev(tmp))
plot(x)
```



Q: Can we use kmeans() to cluster these data, setting k to 2 and nstart to 20? YEAH, we can.

K-Means Clustering: Perform k-means clustering on a data matrix. Usage kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy","MacQueen"), trace=FALSE)

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -2.984583  2.876489
## 2  2.876489 -2.984583
##
## 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] 77.65976 77.65976
##  (between_SS / total_SS =  86.9 %)
##
## Available components:
##
```

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

But we need to do something with these data now..

> Q: How many points are in each cluster? How do we extract this? (What component of your result object details size?)

Size:
The number of points in each cluster.

Look at the kmeans clustering function! There's an argument called "size"

```
km$size
```

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

> Q: What component of your result object details cluster assignmnet/membership?

Cluster: A vector of integers (from 1:k) indicating the cluster to which each point is allocated

```
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 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
```

> Q: What component of your result object details the cluster center?

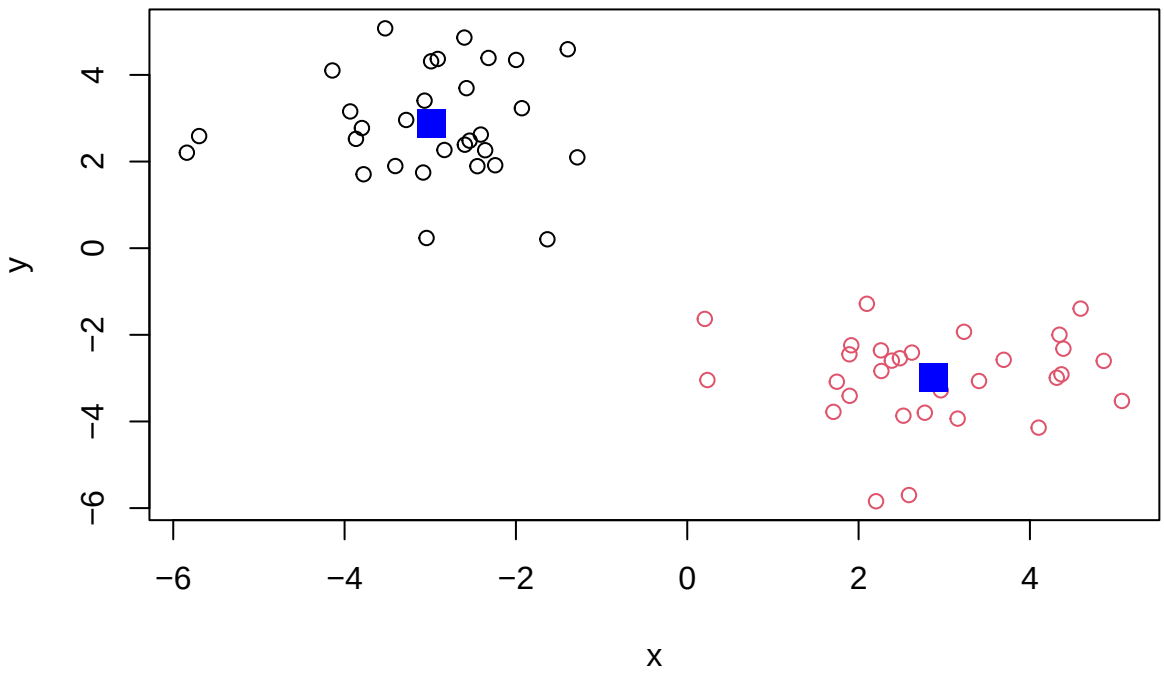
Centers: A matrix of cluster centres.

```
km$centers
```

```
##           x         y
## 1 -2.984583  2.876489
## 2  2.876489 -2.984583
```

> Q: Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

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



#hclust(): Hierarchical Clustering Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

A big limitation with kmeans is that we have to tell it (must specify) K (the number of clusters we want)

It doesn't reveal the natural groupings of data! Since K is specified, the visualization may not be accurate

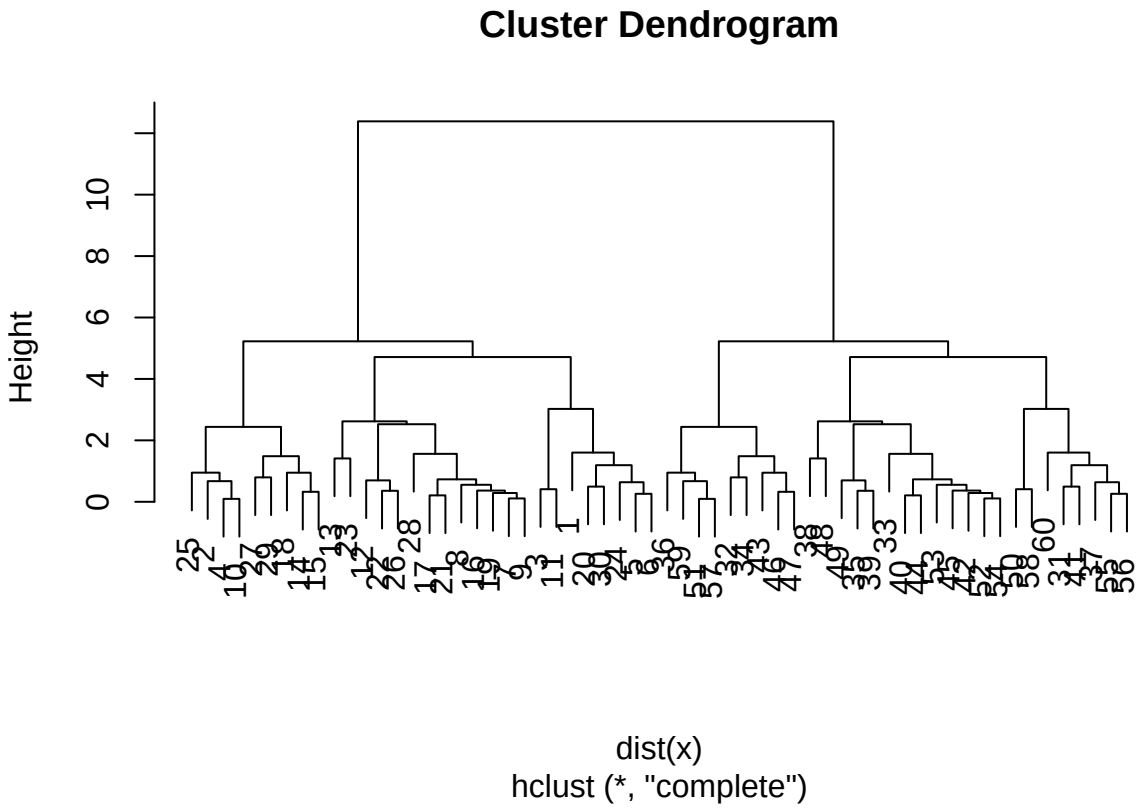
> Q: Analyze this same data with hclust() Demonstrate the use of dist(), hclust(), plot() and cutree() functions to do clustering. Generate dendrograms and return cluster assignment/membership vector

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

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

There is a plot method for hclust result objects. Let's see it. –> a Cluster Dendrogram

```
plot(hc)
```



To get our cluster membership vector, we have to do a bit more work. We have to "cut" the tree where we think it makes best sense (Pick a height to cut the tree at). For this, we use the `cutree()` function.

```
cutree(hc, h=6)
```

```
##  [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
```

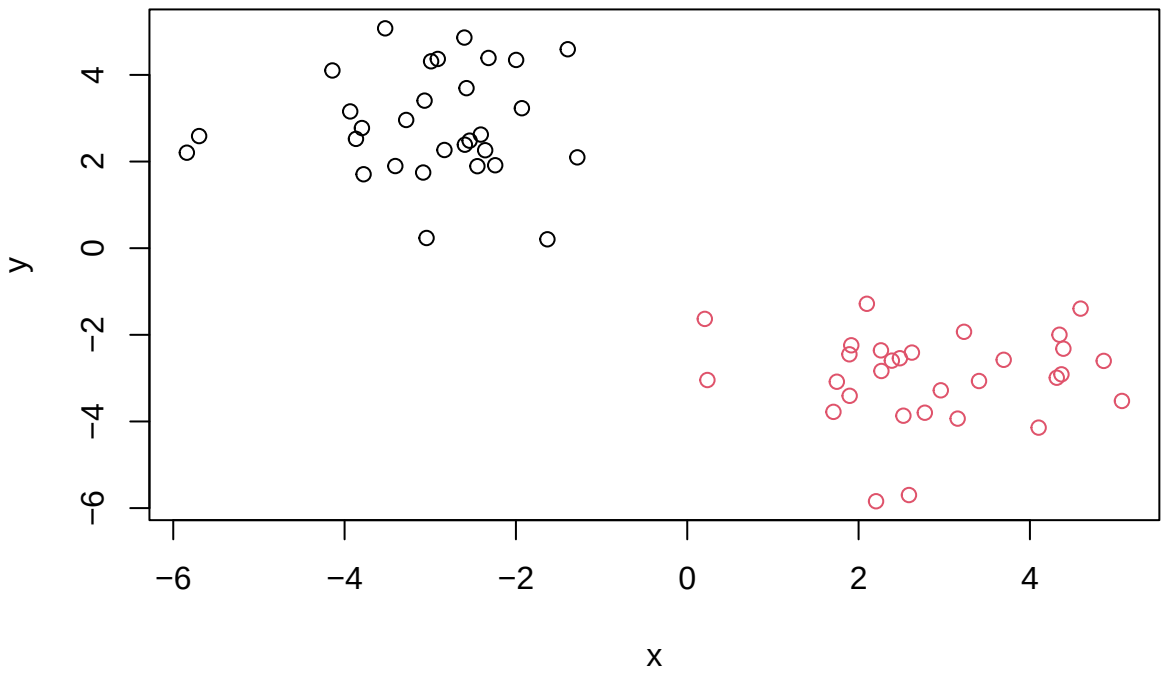
You can also call `cutree()` by setting k=the number of groups/clusters you want.

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

To make plot, save to a variable.

Now make our results plot

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



#Principal Component Analysis (PCA) - Dimensionality reduction, visualization and 'structure' analysis - PCA projects the features onto the principle components - Motivation is to reduce the features demensionality while only losing a small amount of information _ Can make left/right/above/below variations easier to see

Goals of PCA: - To reduce dimensionality • To visualize multidimensional data • To choose the most useful variables (features) • To identify groupings of objects (e.g. genes/samples) • To identify outliers

Class 8 Lab: Hands on with Principal Component Analysis (PCA)

1. PCA of UK food data

Data import

```
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?

- You can use the dim() function, which returns the number of rows and columns or the nrow() and ncol() functions to return each separately, i.e. dim(x); ncol(x); nrow(x)

```
dim(x)
```

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

Checking your data

Use the View() function to display all the data (in a new tab in RStudio) or the head() and tail() functions to print only a portion of the data (by default 6 rows from either the top or bottom of the dataset respectively

Also, never leave a View() function call uncommented in your Rmarkdown document as this is intended for interactive use and will stop the Knit rendering process when you go to Knit and generate HTML, PDF, MD etc. reports.

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
```

We don't have proper row names! Use rownames() to set rownames to first column and then removes the troublesome first column (with the -1 column index)

```
#Note how 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
```

ALTERNATIVELY, we could have done intitially:

```
url <- "https://tinyurl.com/UK-foods"
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
```

Check dimensions again:

```
dim(x)
```

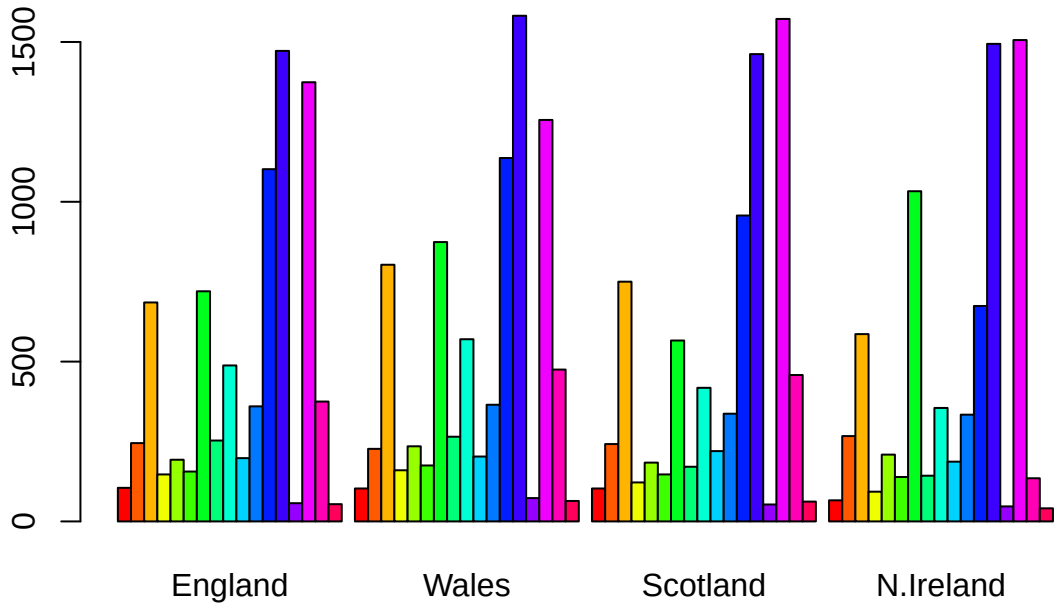
```
## [1] 17  4
```

> 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 would prefer the alternative method as it utilizes less lines of code. It is also more robust under certain circumstances as the first approach falls apart if the code is run multiple times! The row names begin disappearing one bye one after each run.

Spotting major differences and trends

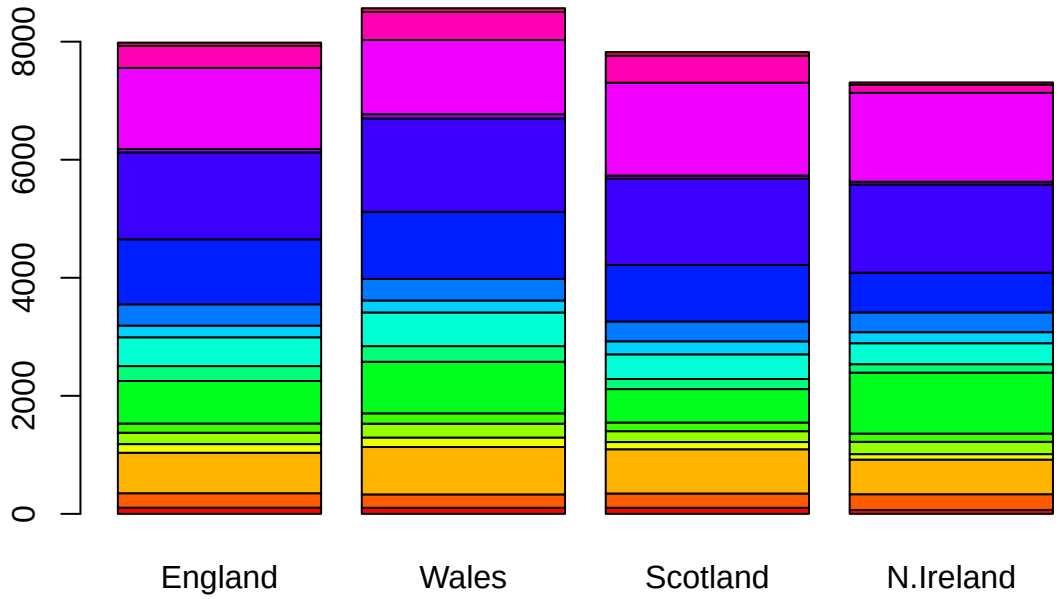
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

You can change the beside argument – beside is a logical value.If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.

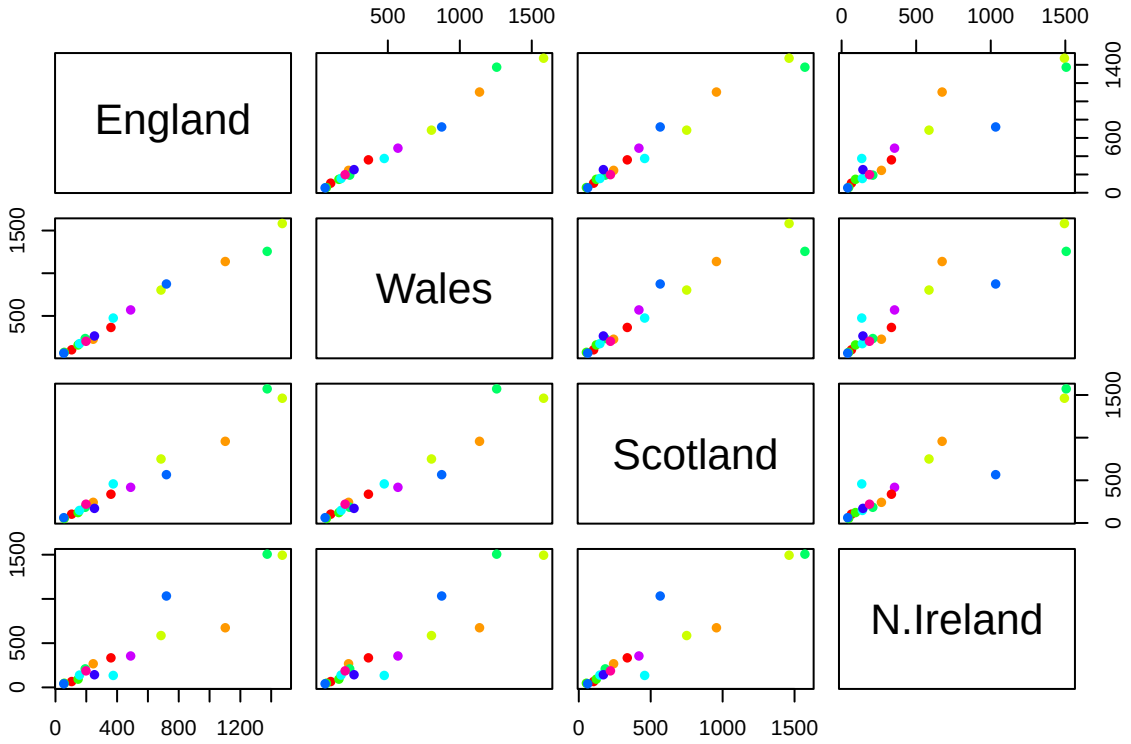
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Leaving this argument out has the same effect as setting it to FALSE because the default of the barplot function is to have beside = FALSE.

> 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?

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



COMPLETE THIS SECTION

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

#and this section

PCA to the rescue

The main function in base R for PCA is `prcomp()`

This wants the transpose of our data.

- prcomp() expects the observations to be rows and the variables to be columns therefore we need to first transpose our data.frame matrix with the t() transpose function.

```
# Use the prcomp() PCA function
pca <- prcomp( t(x) )
summary(pca)
```

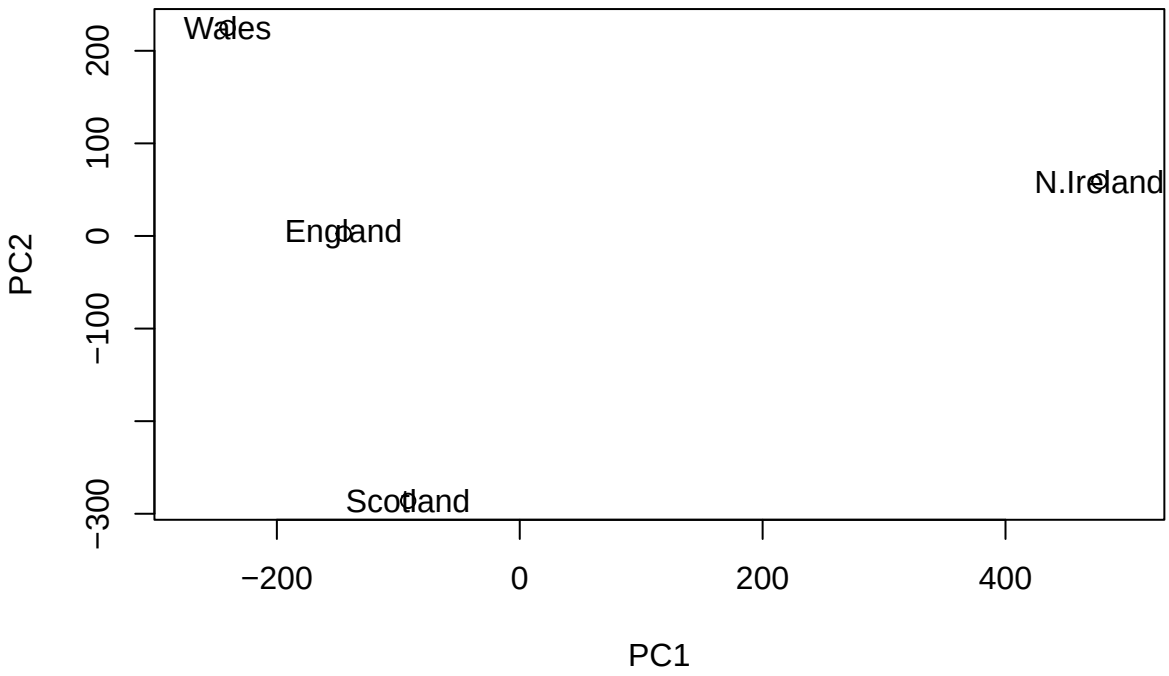
```
## Importance of components:
##                             PC1      PC2      PC3        PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
```

```
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

The summary print-out above indicates that PC1 accounts for more than 67% of the sample variance, PC2 29% and PC3 3%. Collectively PC1 and PC2 together capture 96% of the original 17 dimensional variance. Thus these first two new principal axis (PC1 and PC2) represent useful ways to view and further investigate our data set. Lets start with a simple plot of PC1 vs PC2.

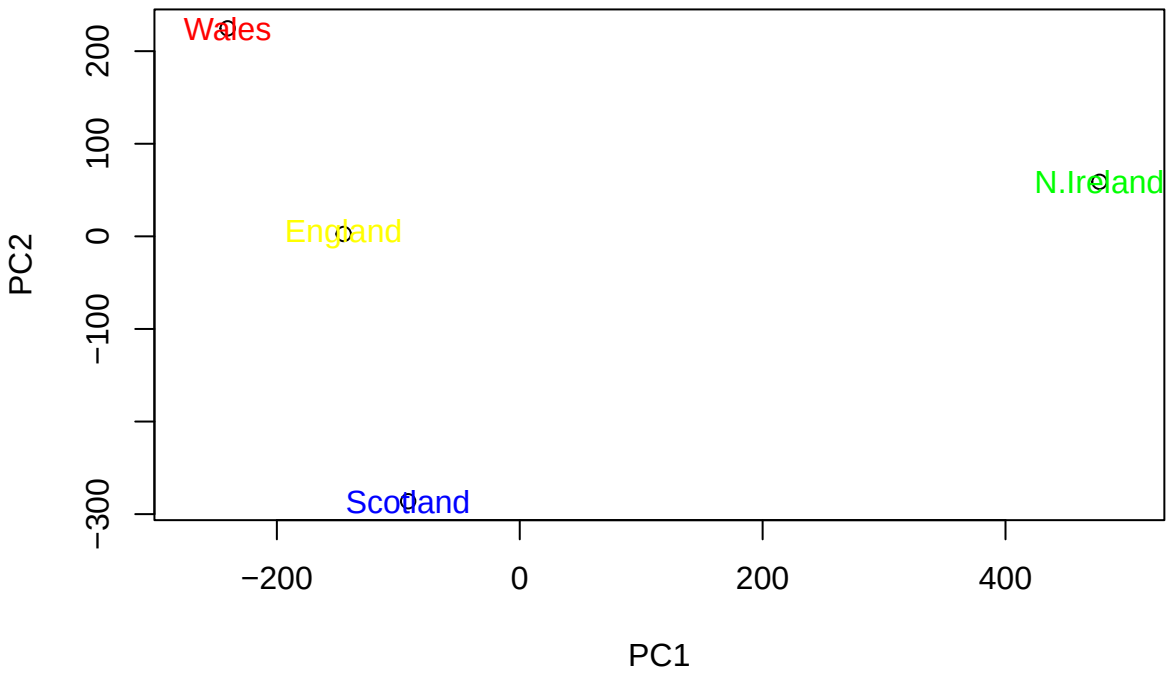
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# 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.

```
#Customize colors
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("yellow","red","blue","green"))
```



Now we can reduce demensionality from 17 to 2

Fist, calculate how much variation in the original data each PC accounts for using the square of pca$sde:

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

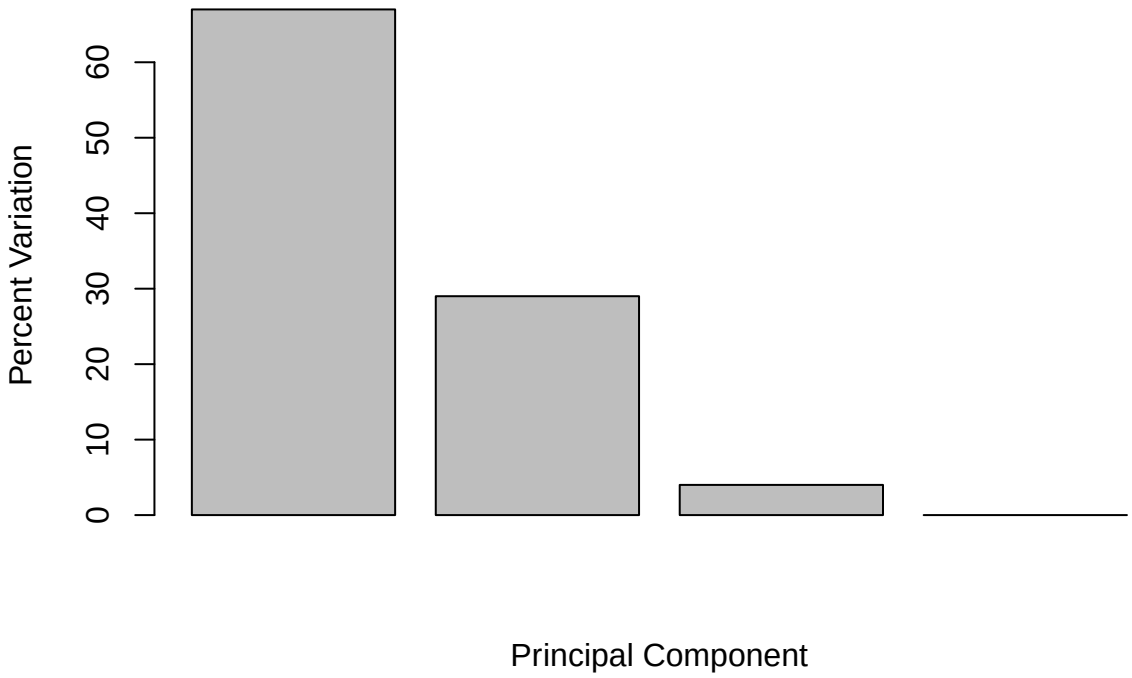
```
## [1] 67 29  4  0
```

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

```
##                             PC1       PC2      PC3          PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

- This information can then be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number):

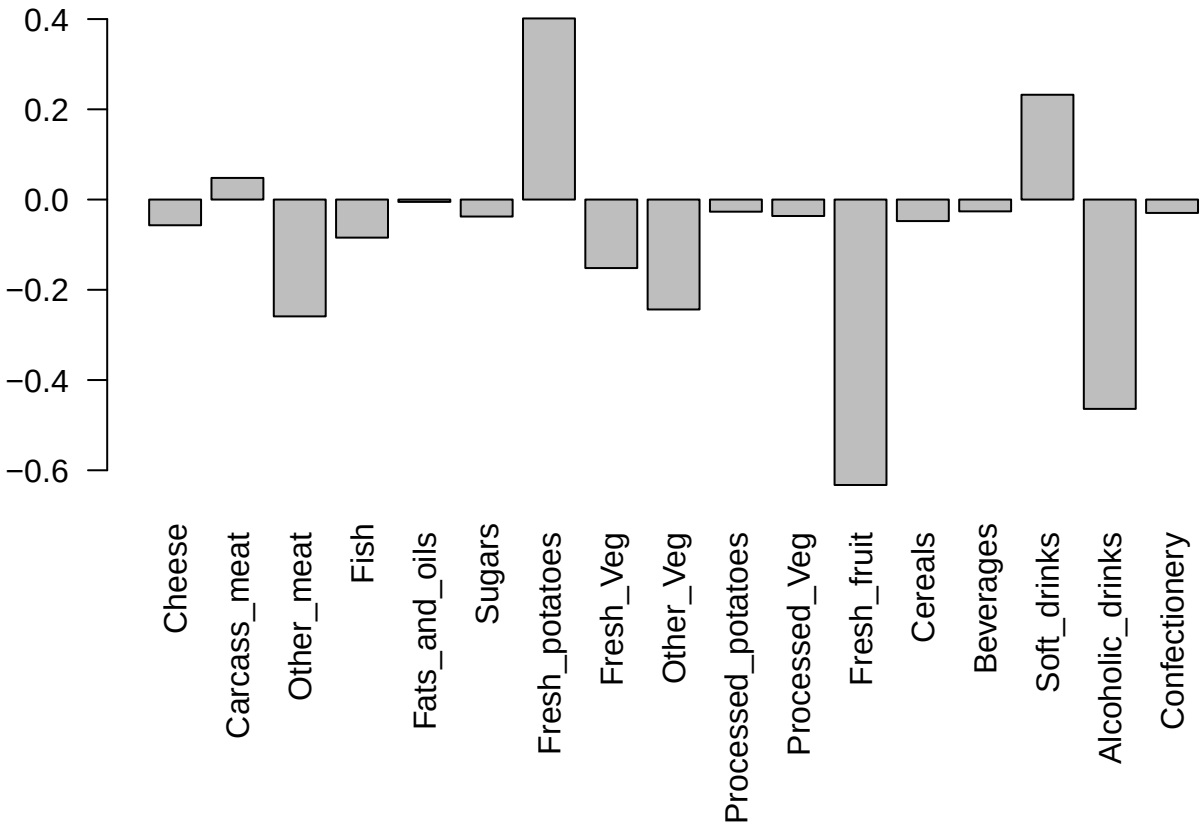
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



Digging deeper (variable loadings)

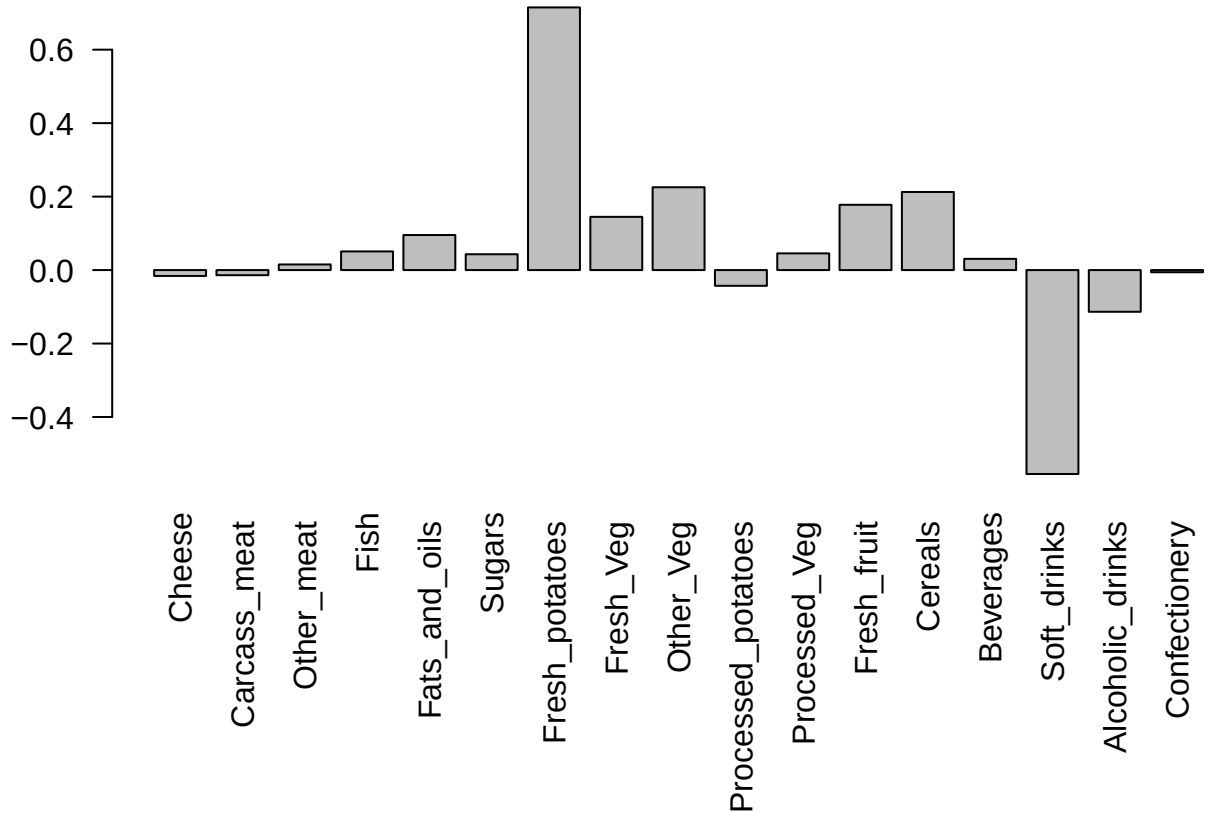
Now, consider the influence of each of the original variables upon the principal components (typically known as loading scores). This information can be obtained from the prcomp() returned $rotation component. It can also be summarized with a call to biplot().

```
## 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?

```
#Generating a similar loading plot for PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```

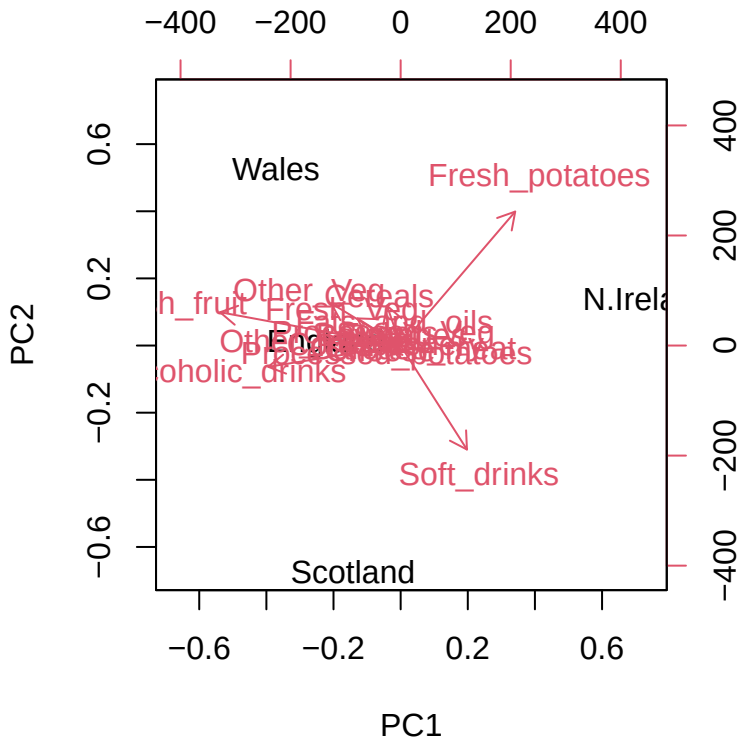


Fresh potatoes and soft drinks feature most predominantly.PC2 mainly tells us that there is a general increase in consumption of fresh potatoes along with a decrease in consumption of soft drinks in Northern Ireland.

Biplots

Another way to see this information together with the main PCA plot is in a so-called biplot:

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



Overall,PCA has the awesome ability to be able to make these associations for us. It has also successfully managed to reduce the dimensionality of our data set down from 17 to 2, allowing us to assert (using our figures above) that *countries England, Wales and Scotland are 'similar' with Northern Ireland being different in some way.* Furthermore, digging deeper into the loadings we were able to associate certain food types with each cluster of countries.

#2. PCA of RNA-seq data

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

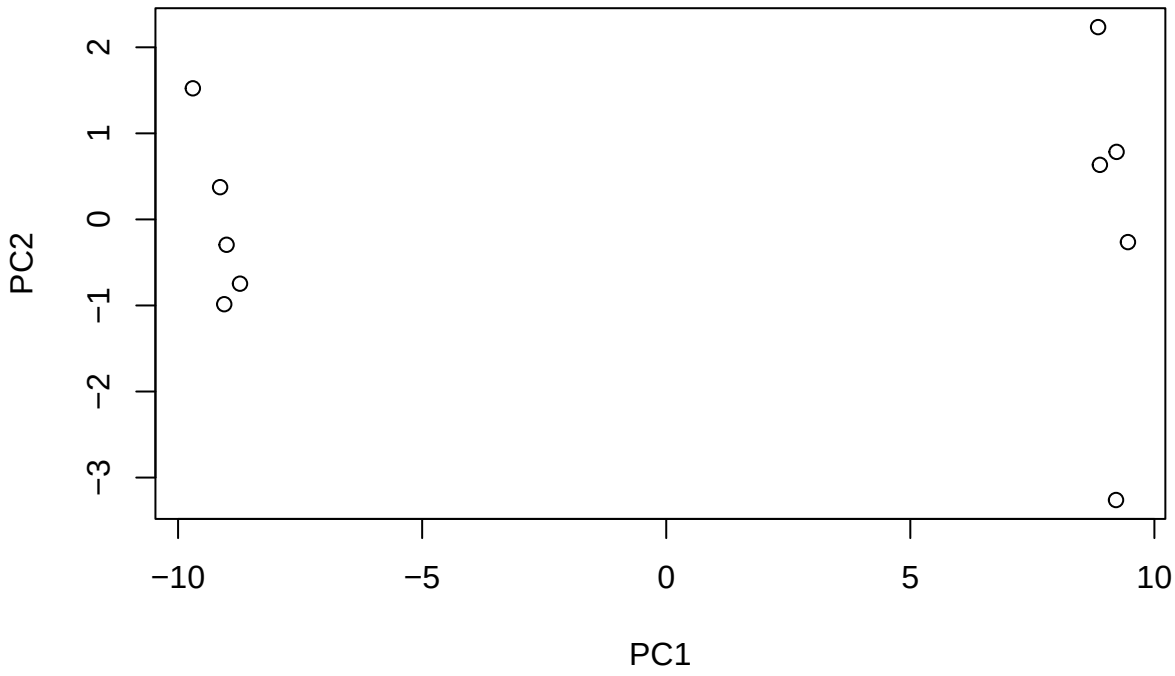
```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

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

- There are 6 genes and 10 samples in this data set

Doing PCA and plotting the results is more worthwhile and exciting than simply plotting on a barplot,etc

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

## This is a simple, un-polished plot of pc1 and pc2: samples are seperated into 2 groups with 5 sample
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```



Let's also examine a summary of how much variation in the original data each PC accounts for:

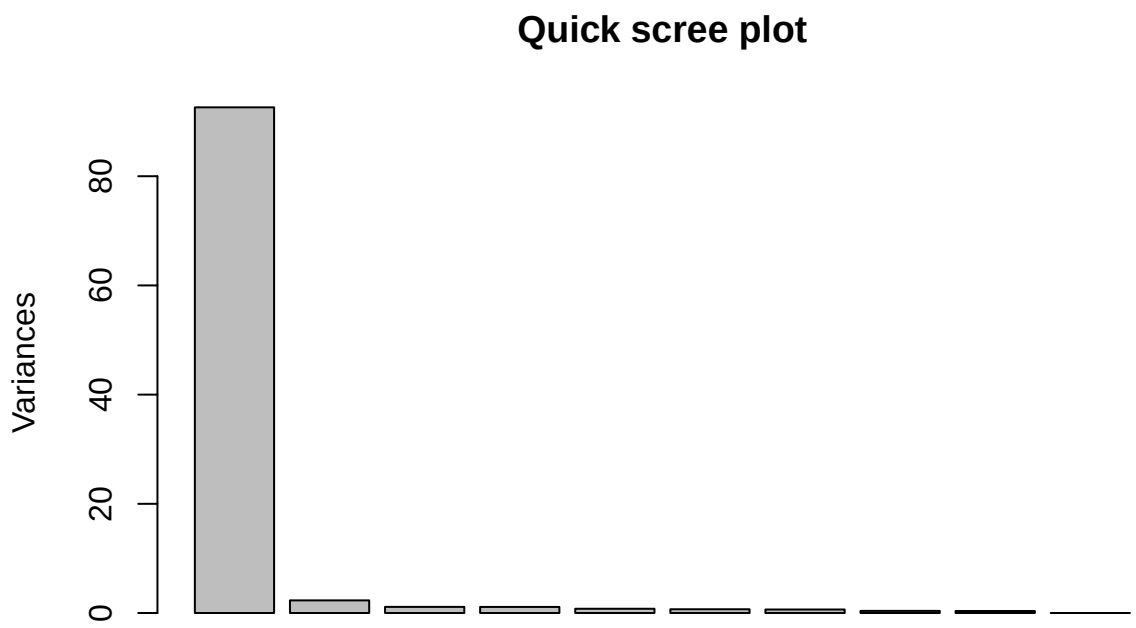
```
summary(pca)
```

```
## 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 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

- We can see from these results that PC1 is where all the action is (92.6% of it in fact!). This indicates that we have successfully reduced a 100 dimensional data set down to only one dimension that retains the main essential (or principal) features of the original data. PC1 captures 92.6% of the original variance with the first two PCs capturing 94.9%.

Let summarize this Proportion of Variance for each PC in a barplot:

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



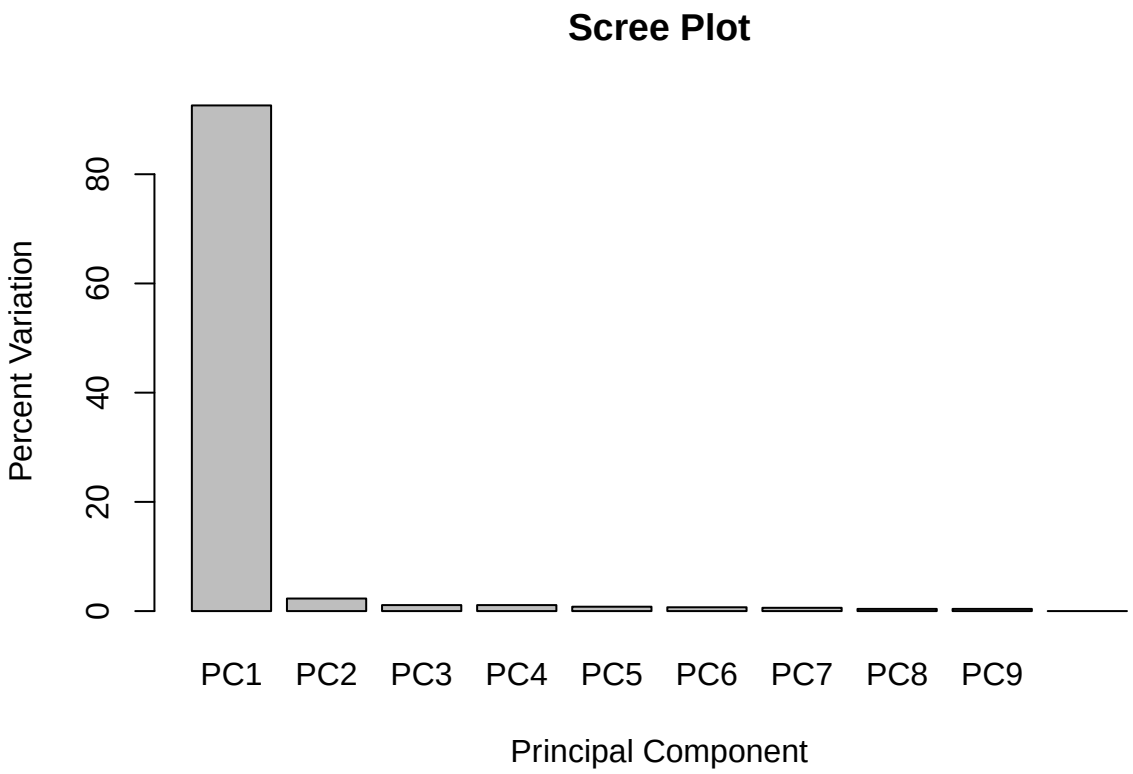
Now, let's explore by calculating how much variation in the original data each PC accounts for manually using the square of the standard deviation:

```
## 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
```

Generating our own scree plot too:

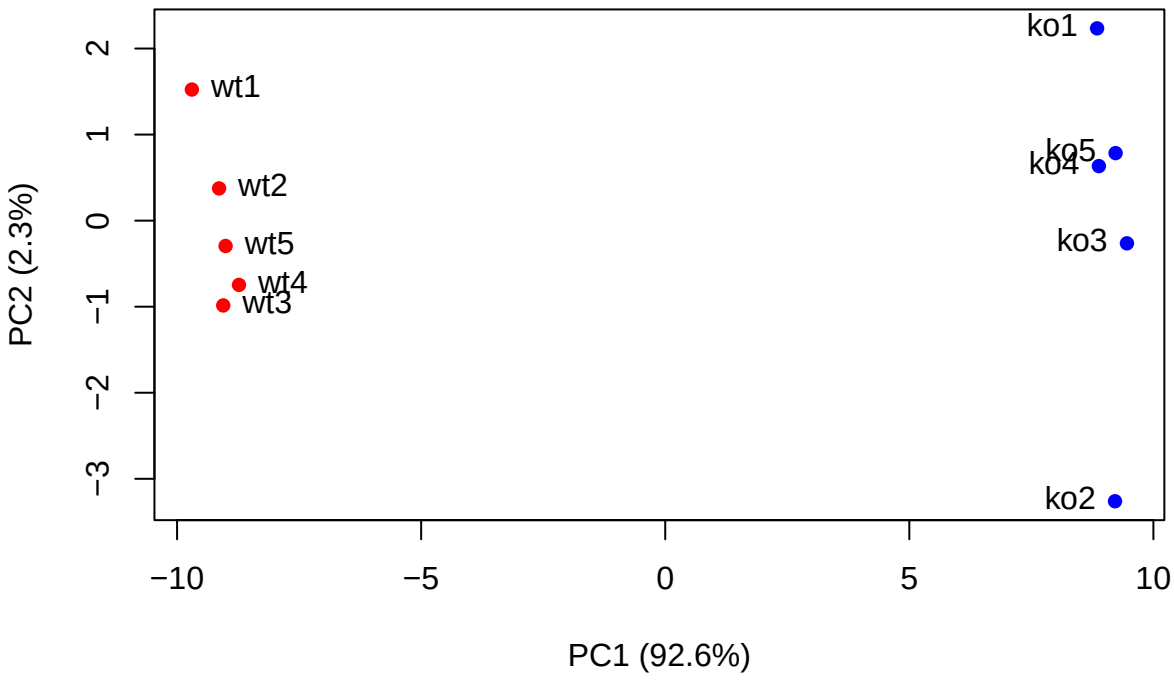
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



- PC1 is where it's at

Further customizing the plot:

```
## 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)))
```

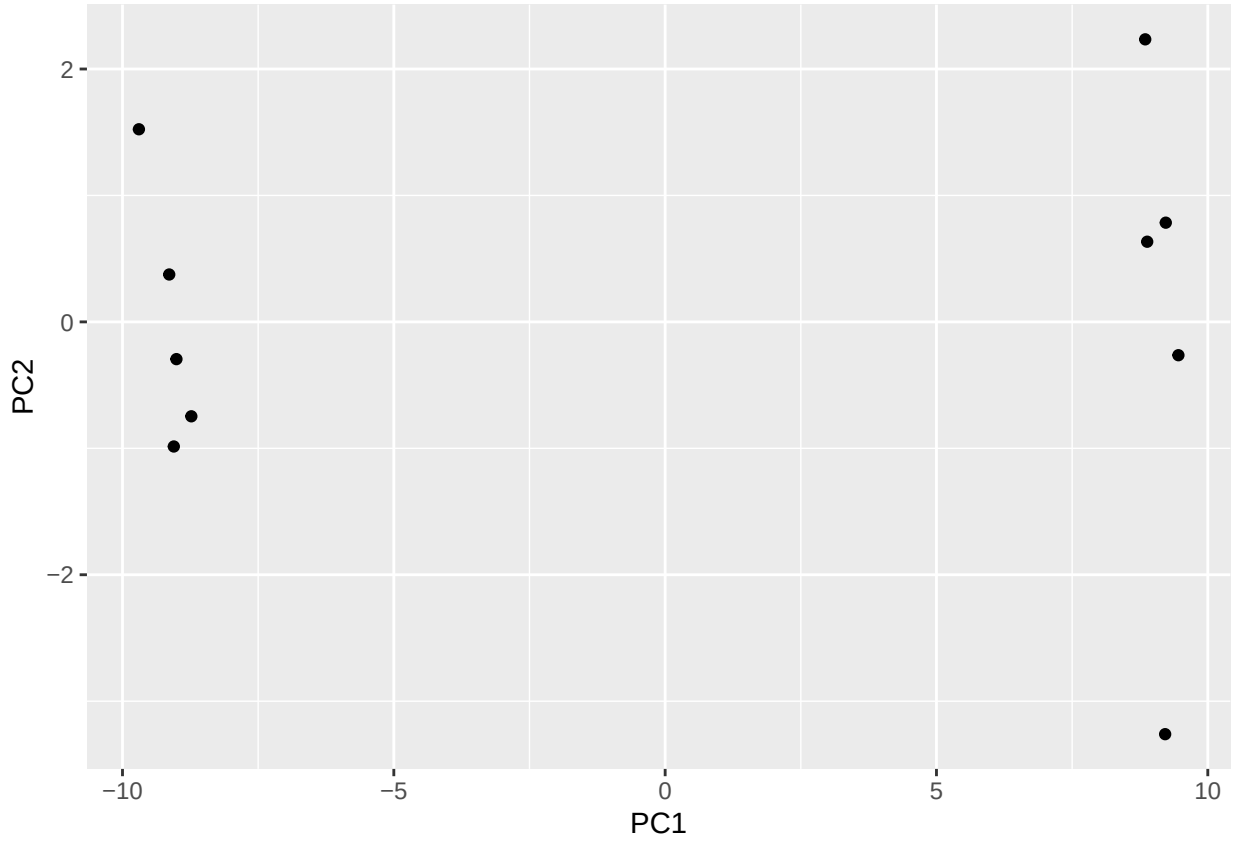


Using ggplot

What if we used ggplot2?

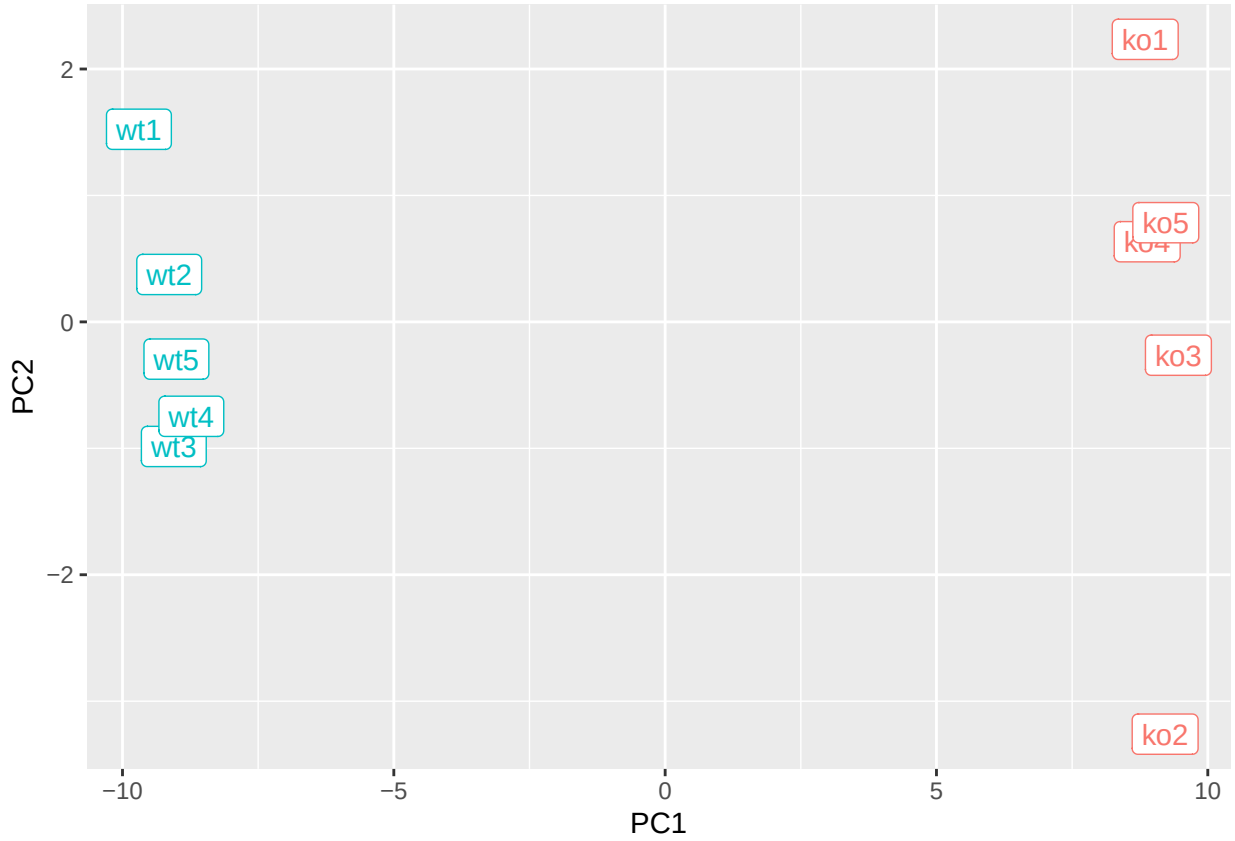
If using ggplot, we will first need a data.frame as input for the main ggplot() function. This data.frame will need to contain our PCA results (specifically pca$x) and additional columns for any other aesthetic mappings we will want to display:

```
library(ggplot2)

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

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



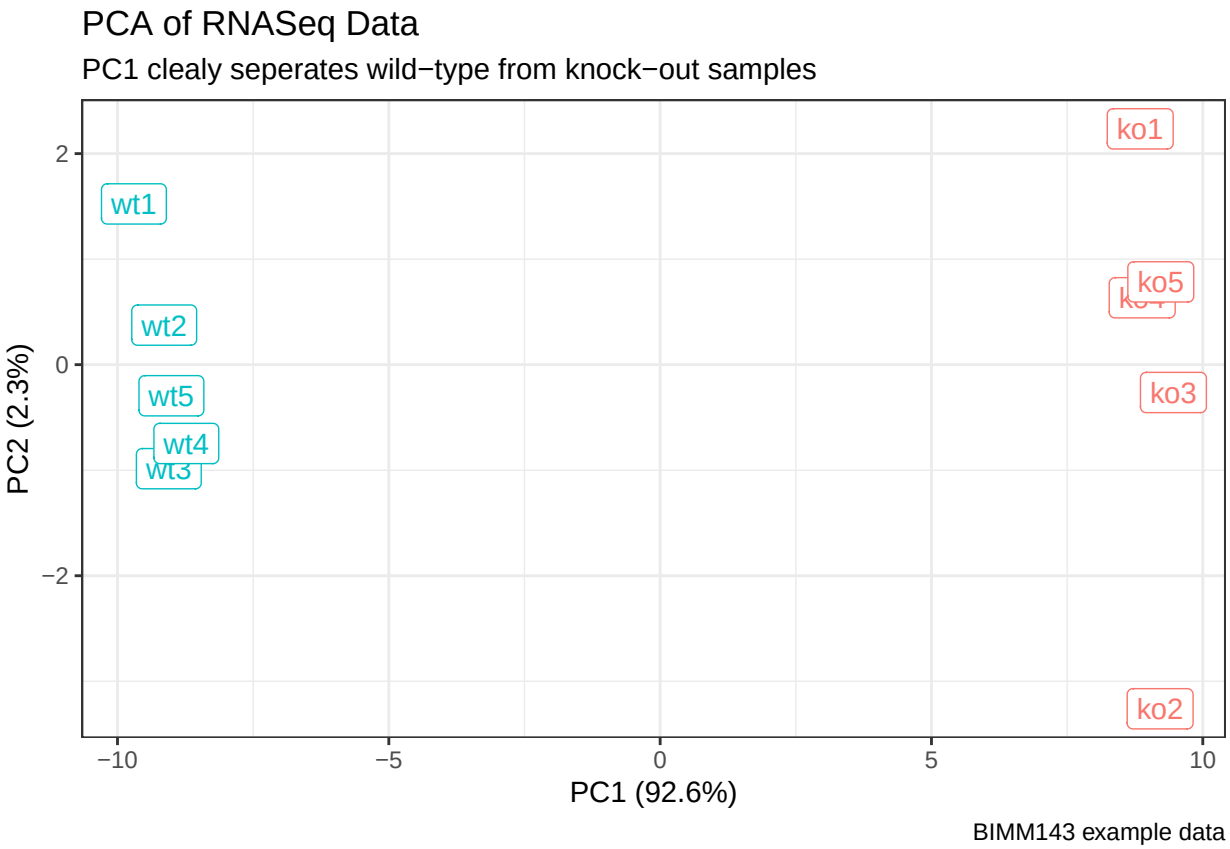
Adding condition specific colors and sample label aesthetics for wild-type and knock-out samples:

```
# 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
```



Making the plot more complete:

```
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="BIMM143 example data") +
     theme_bw()
```



Optional: Gene loadings

Finding the top 10 measurements (genes) that contribute most to pc1 in either direction (+ or -):

```
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"
```