Class08: Machine Learning 1 & PCA

Monica Lin (PID: A15524235)

10/21/2021

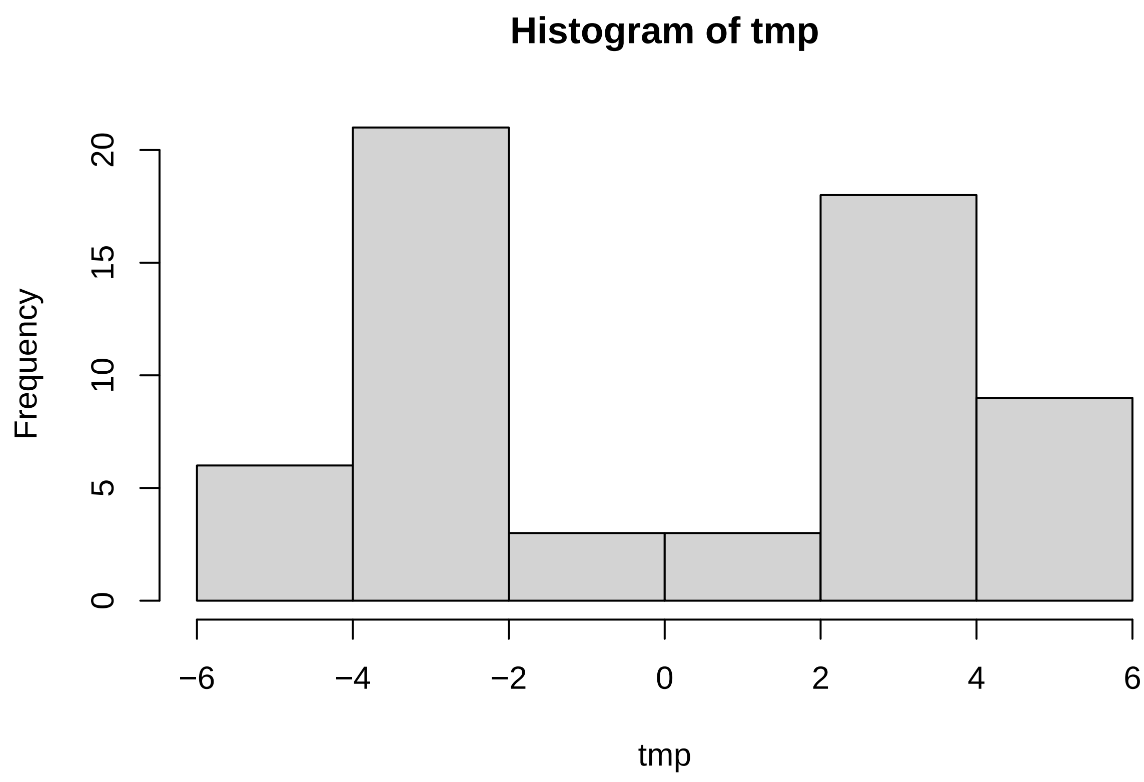
First up is clustering methods

Kmeans clustering

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

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

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



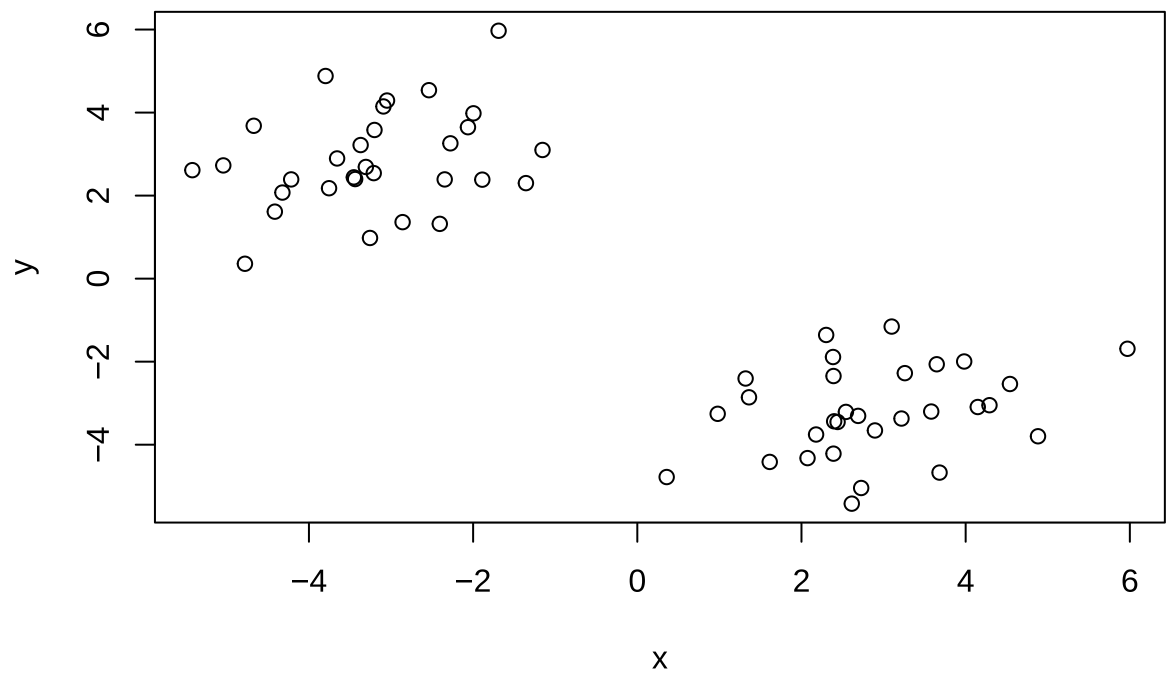
This gives two groups of vectors, centered around -3 and +3, with 60 values total.

Now reverse the order of the input. Center around +3 and -3.

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

```
##                x          y
##  [1,] -3.0928486  4.1475365
##  [2,] -3.3063550  2.6898744
##  [3,] -4.2164658  2.3896506
##  [4,] -3.0489580  4.2900816
##  [5,] -3.2562188  0.9793361
##  [6,] -5.4203664  2.6126142
##  [7,] -2.0628996  3.6475183
##  [8,] -1.9960504  3.9820419
##  [9,] -4.6724246  3.6820472
## [10,] -4.4153949  1.6132984
## [11,] -5.0434790  2.7272112
## [12,] -3.7975564  4.8813141
## [13,] -3.6569924  2.8947753
## [14,] -1.6897198  5.9709110
## [15,] -4.3233799  2.0735453
## [16,] -3.3704052  3.2179011
## [17,] -2.3463200  2.3899503
## [18,] -4.7797054  0.3578174
## [19,] -1.3567228  2.3006397
## [20,] -2.8591010  1.3606605
## [21,] -2.2770112  3.2589877
## [22,] -2.5383390  4.5393998
## [23,] -3.2016945  3.5805956
## [24,] -1.8882576  2.3842908
## [25,] -3.4533263  2.4397812
## [26,] -3.4370535  2.3982947
## [27,] -3.7552079  2.1778181
## [28,] -2.4061426  1.3193242
## [29,] -1.1547003  3.0987915
## [30,] -3.2107699  2.5414108
## [31,]  2.5414108 -3.2107699
## [32,]  3.0987915 -1.1547003
## [33,]  1.3193242 -2.4061426
## [34,]  2.1778181 -3.7552079
## [35,]  2.3982947 -3.4370535
## [36,]  2.4397812 -3.4533263
## [37,]  2.3842908 -1.8882576
## [38,]  3.5805956 -3.2016945
## [39,]  4.5393998 -2.5383390
## [40,]  3.2589877 -2.2770112
## [41,]  1.3606605 -2.8591010
## [42,]  2.3006397 -1.3567228
## [43,]  0.3578174 -4.7797054
## [44,]  2.3899503 -2.3463200
## [45,]  3.2179011 -3.3704052
## [46,]  2.0735453 -4.3233799
## [47,]  5.9709110 -1.6897198
## [48,]  2.8947753 -3.6569924
## [49,]  4.8813141 -3.7975564
## [50,]  2.7272112 -5.0434790
## [51,]  1.6132984 -4.4153949
## [52,]  3.6820472 -4.6724246
## [53,]  3.9820419 -1.9960504
## [54,]  3.6475183 -2.0628996
## [55,]  2.6126142 -5.4203664
## [56,]  0.9793361 -3.2562188
## [57,]  4.2900816 -3.0489580
## [58,]  2.3896506 -4.2164658
## [59,]  2.6898744 -3.3063550
## [60,]  4.1475365 -3.0928486
```

```
plot(x)
```



Q. Can we use kmeans() to cluster this data, setting k = 2 and nstart = 20?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.201129  2.864914
## 2  2.864914 -3.201129
##
## 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 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
##
## Within cluster sum of squares by cluster:
## [1] 77.27043 77.27043
##  (between_SS / total_SS =  87.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Clustering vector tells you the cluster to which each data point is allocated.

> Q. How many points are in each cluster?

```
km$size
```

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

km$sizereturnsthenumberofpointsineachcluster, i.e.theclustersize.'' allows you to see the input functions that are available with 'km'. km$totss gives the total sum of squares.

> Q. 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 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
```

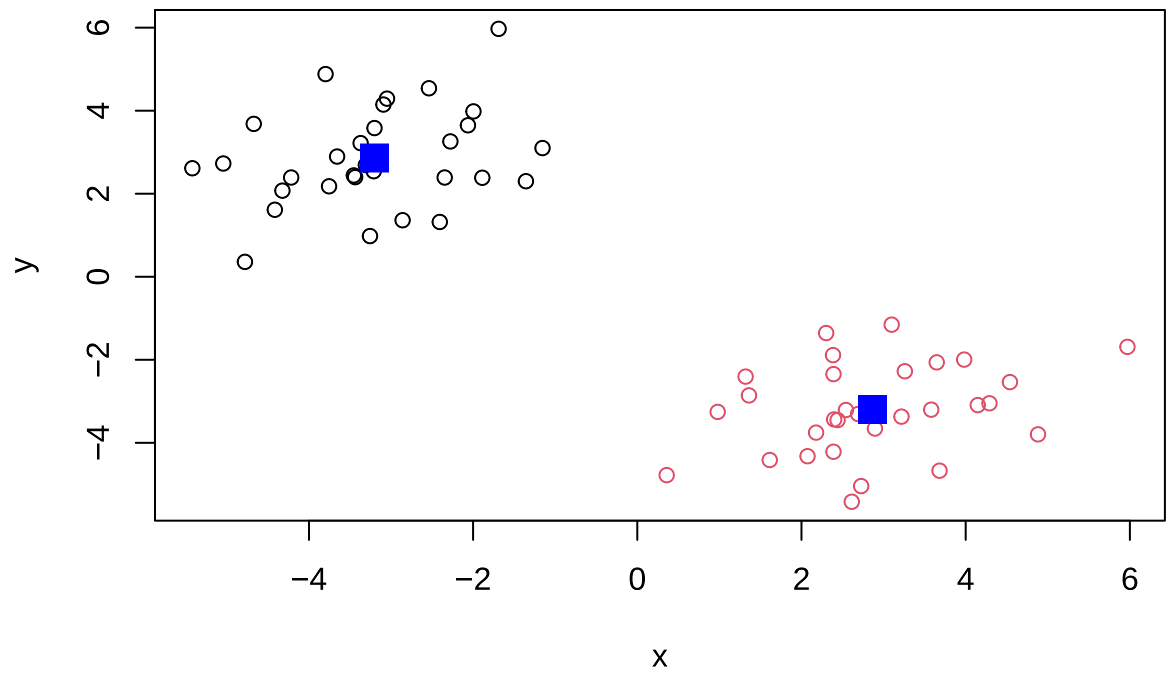
> Q. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1 -3.201129  2.864914
## 2  2.864914 -3.201129
```

> 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

A big limitation with k-means is that we have to tell it K (the number of clusters we want). Hierarchical clustering bypasses that by generating the ‘d’ via the distribution. Analyze this same data with hclust().

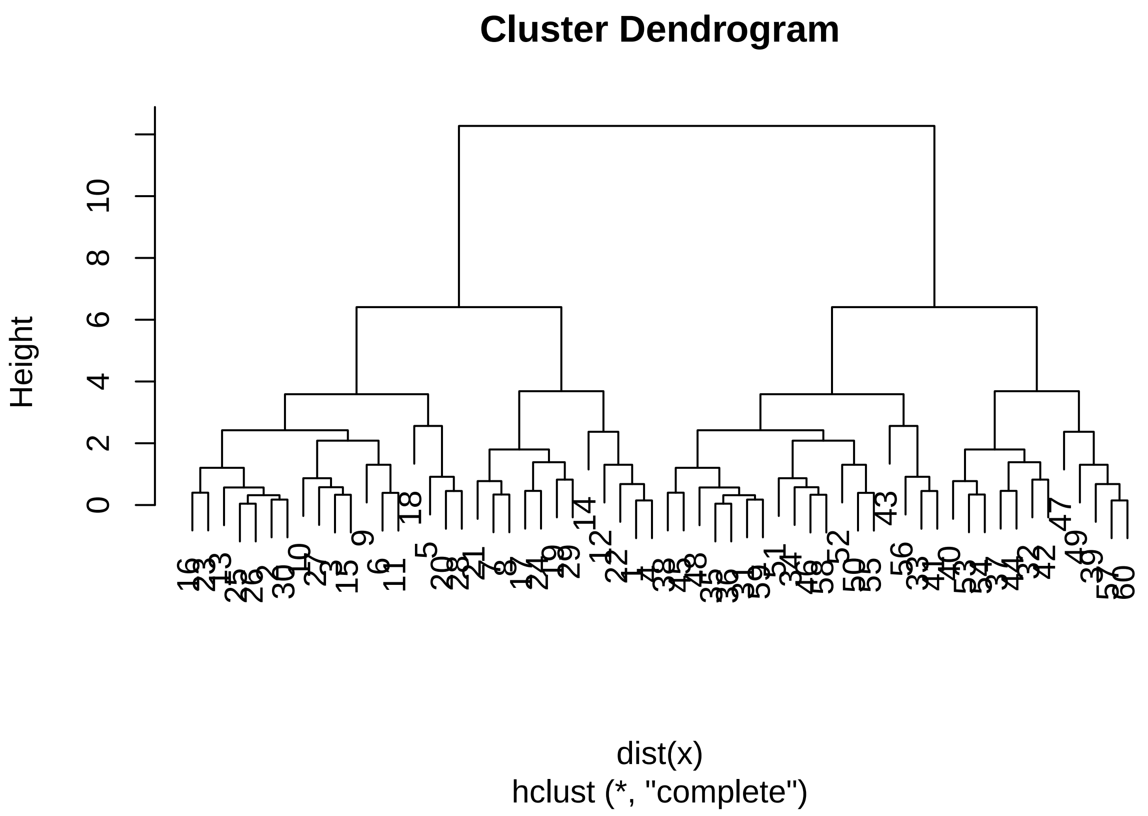
Demonstrate the use of dist(), hclust(), plot() and cutree() functions to do clustering. Generate dendrograms and return cluster assignment/membership vector.

```
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.

```
plot(hc)
```



Tree reveals structure of clusters. Taller goalposts = larger distance apart. Clusters 1 and 2 are separated and grouped.

To get our cluster membership vector, we have to do a wee bit more work. We have to “cut” the dendrogram where we think it makes sense. Visually analyze the shape of the tree to determine where to cut. For this we use the `cutree()` function to cut into k grps.

```
cutree(hc, h=8)
```

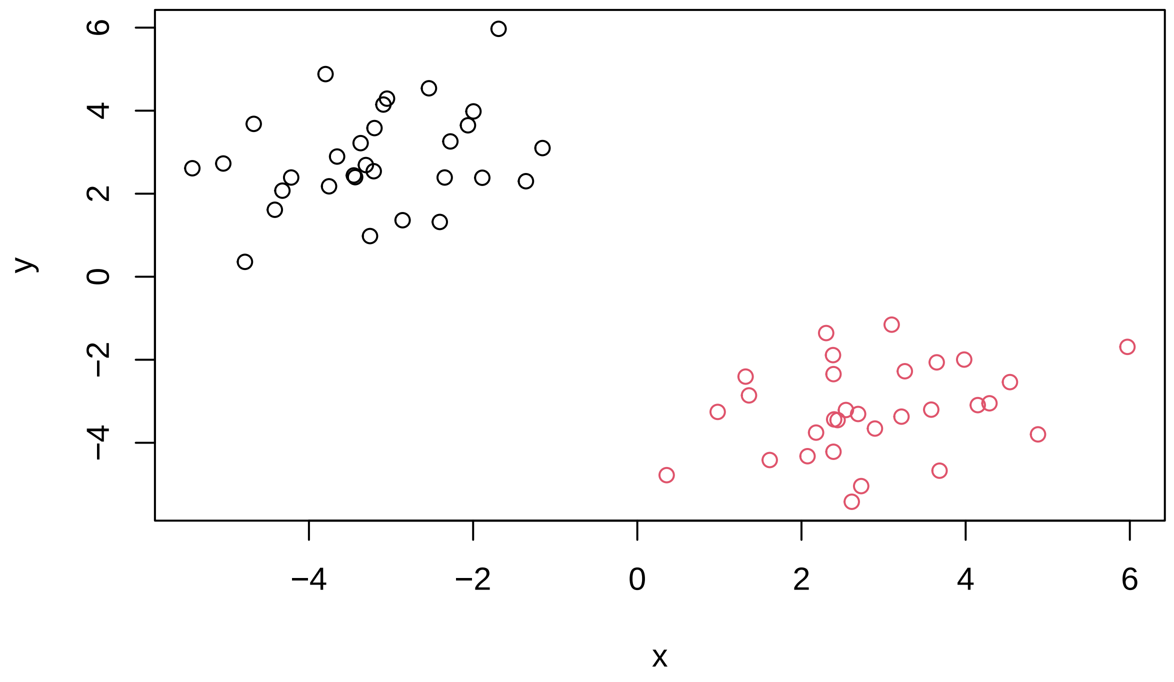
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

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

Make our results plot

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



1. Principal Component Analysis of UK food data

Data import

Read the provided `UK_foods.csv` input file.

```
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 these questions?

```
nrow(x)
```

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

```
ncol(x)
```

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

Examine the imported data

Use the `View()` function to display all the data, or the `head()` and `tail()` functions to preview the first 6 rows of the top and bottom of the dataset.

```
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
```

Rats! This should be 17 x 4 dimensions. The first column X should not be there. Get rid of the X column because they are not numerical.

One way:

```
# 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
```

This is dangerous! Every time you run the code chunk, a column is removed.

Better way:

```
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
```

This turns the first column X into row labels and does not interfere with the data.

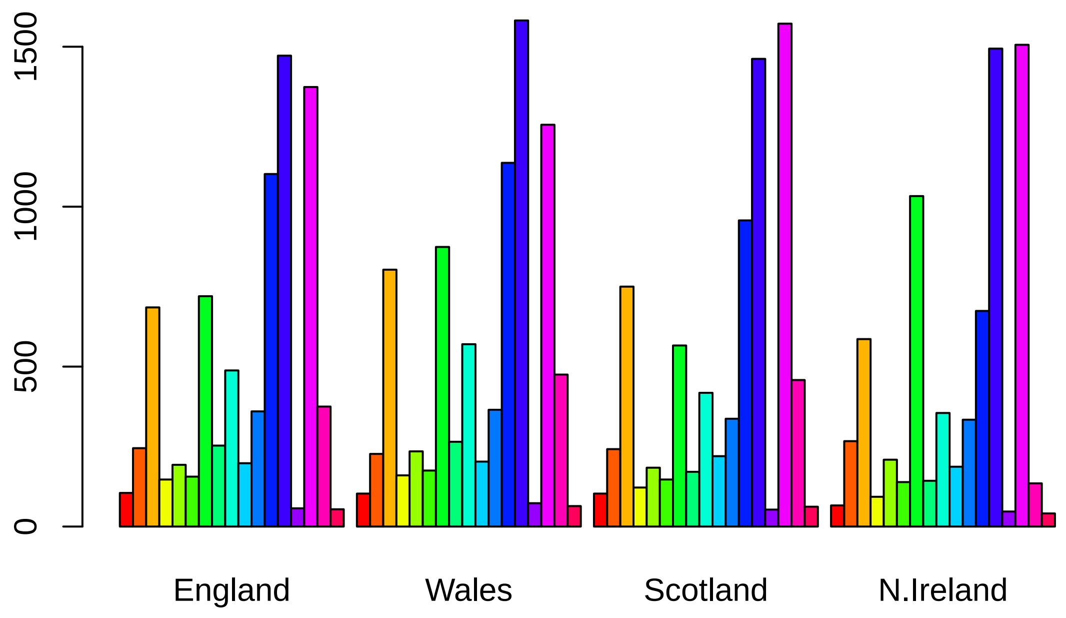
> **Q2.** Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

The second approach is preferable because it does not run the risk of overwriting the data by deleting a column every time the code chunk is run.

Spotting major differences and trends

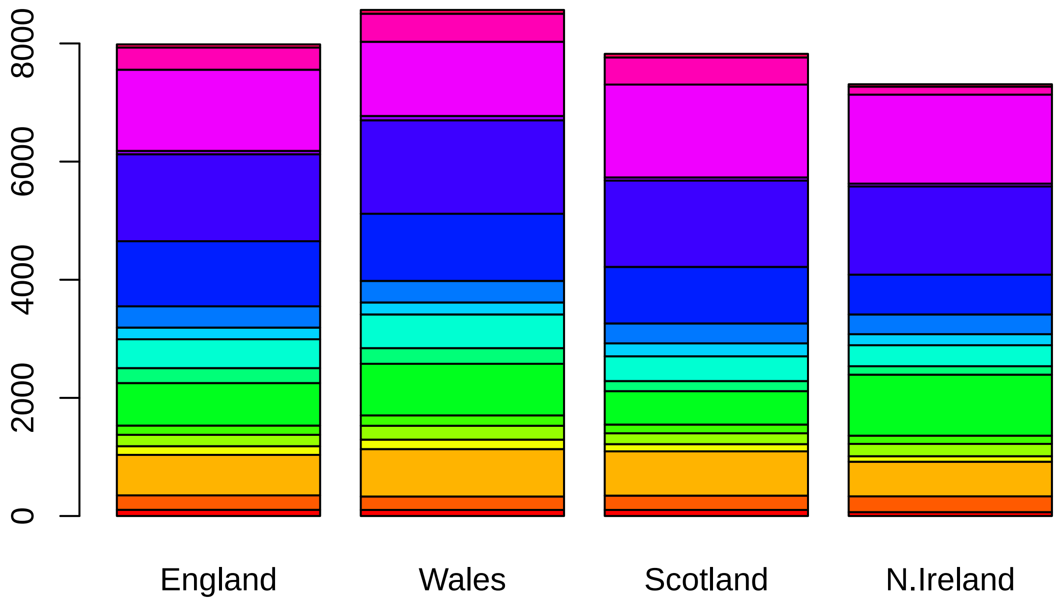
Generating regular bar plots and pairwise plots are not helpful.

```
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?

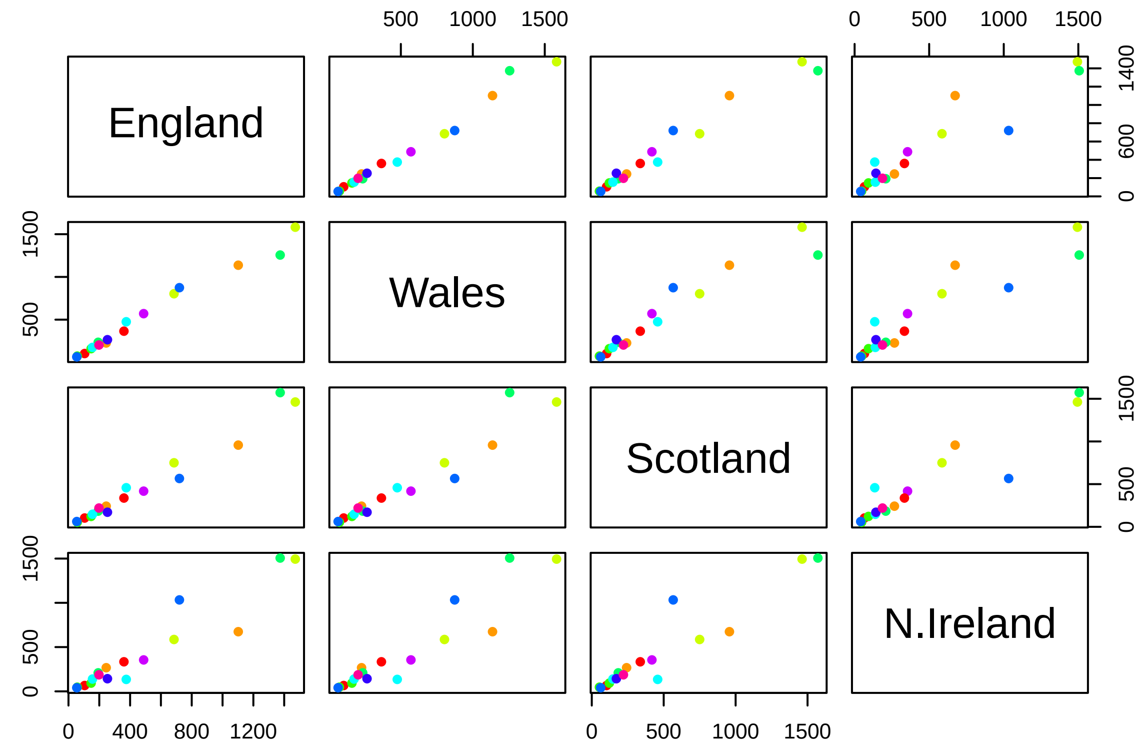
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Changing the beside argument from **TRUE** to **FALSE** results in the following plot.

> **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)
```



The 17 colors in each plot are for each of the different rows. The labeled countries correspond to their respective columns and rows, where each plot is a comparison of the two different countries in that column and row. Lines on the diagonal represent that the x and y variables are similar and follows the general trend, while points that lie off the diagonal are more dissimilar.

> **Q6.** What is the main difference between N. Ireland and the other countries of the UK in terms of this dataset?

N. Ireland deviates the most from that diagonal line, illustrating its dissimilarity from the other countries of the UK.

PCA to the rescue!

The main function in base R for PCA is `prcomp()`. `prcomp()` expects the *observations* as rows and the *variables* as columns. Thus, we want to transpose our data.frame matrix with the `t()` transpose 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
```

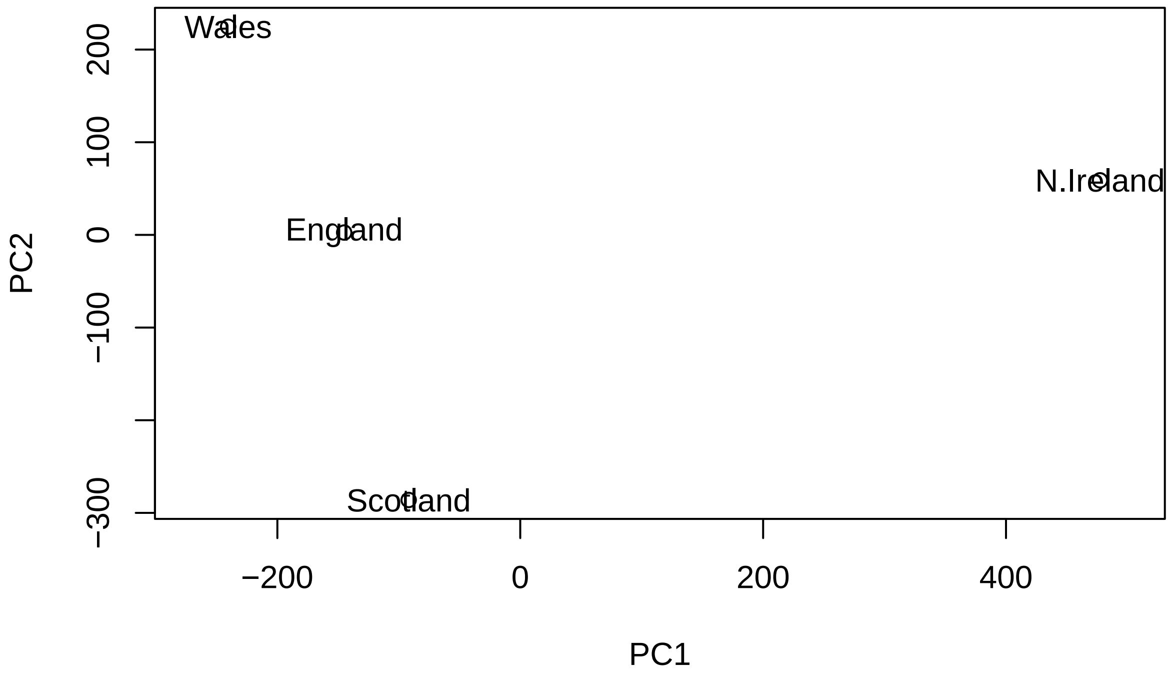
Look inside the PCA object

```
attributes(pca)
```

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

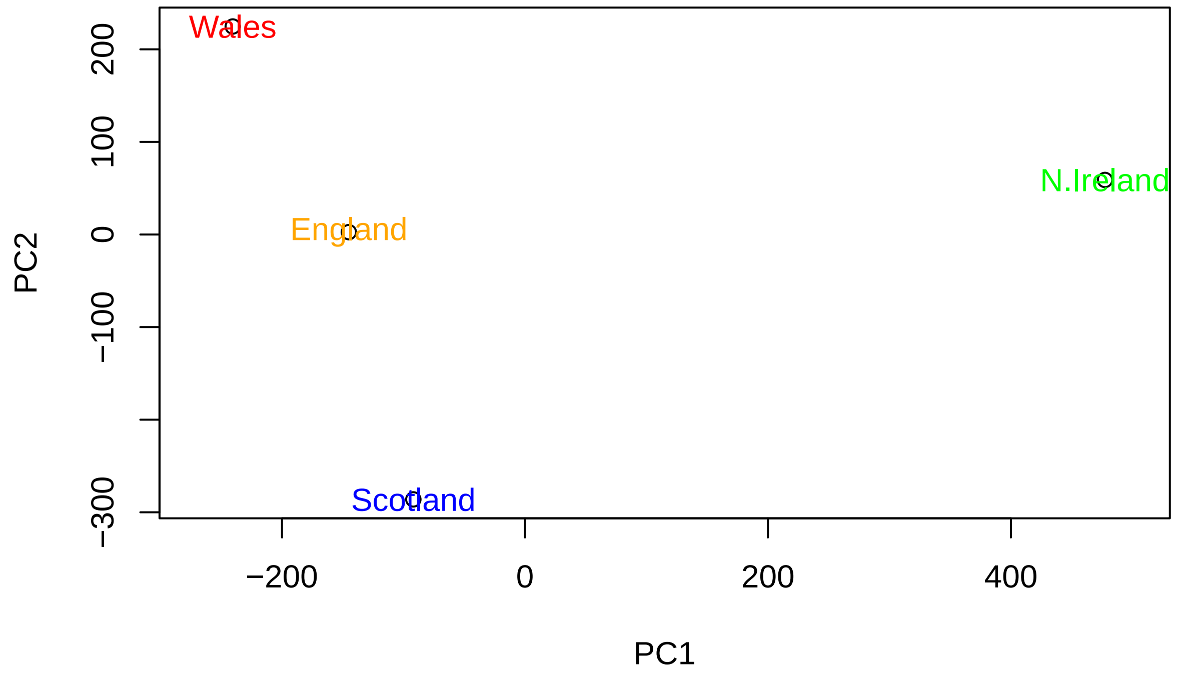
> **Q7.** Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))

# Add column names to the plot
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 the start of this document.

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
color <- c("orange", "red", "blue", "green")
text(pca$x[,1], pca$x[,2], colnames(x), col=color)
```



Now that the principal components are obtained, we can use them to map the relationship between variables (i.e. countries) in terms of these major PCs (i.e. new axis that maximally describe the original data variance).

Use the square of pca$sdev to calculate how much variation each PC accounts for in the original data.

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

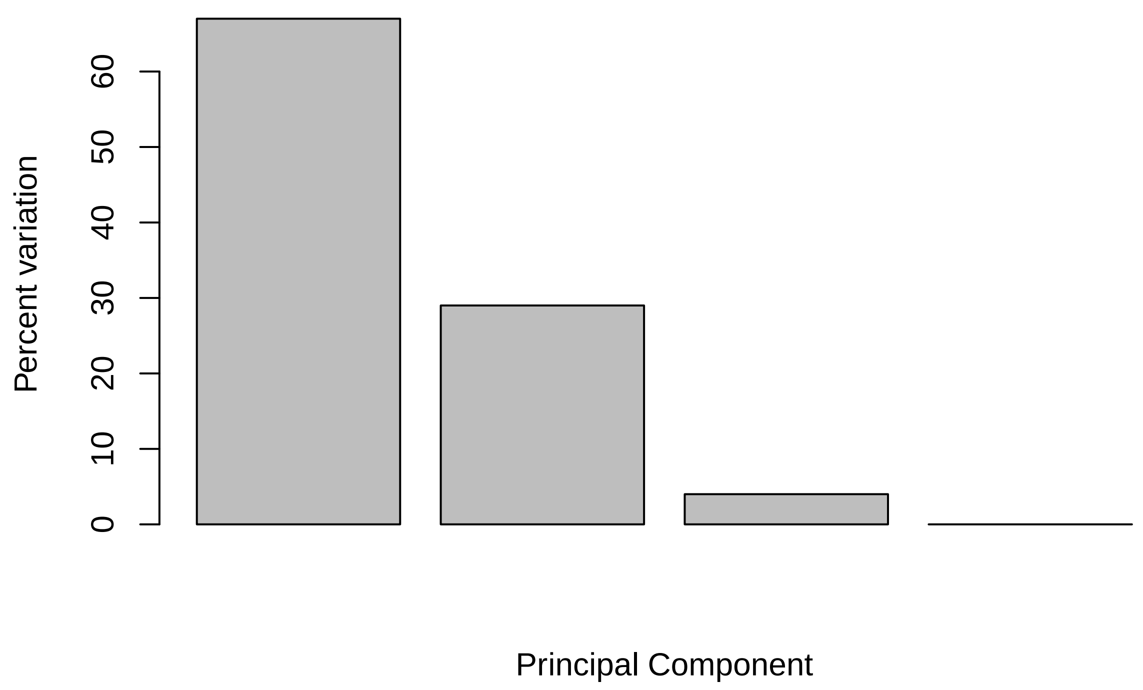
```
## [1] 67 29  4  0
```

```
# 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 be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number), which is given below.

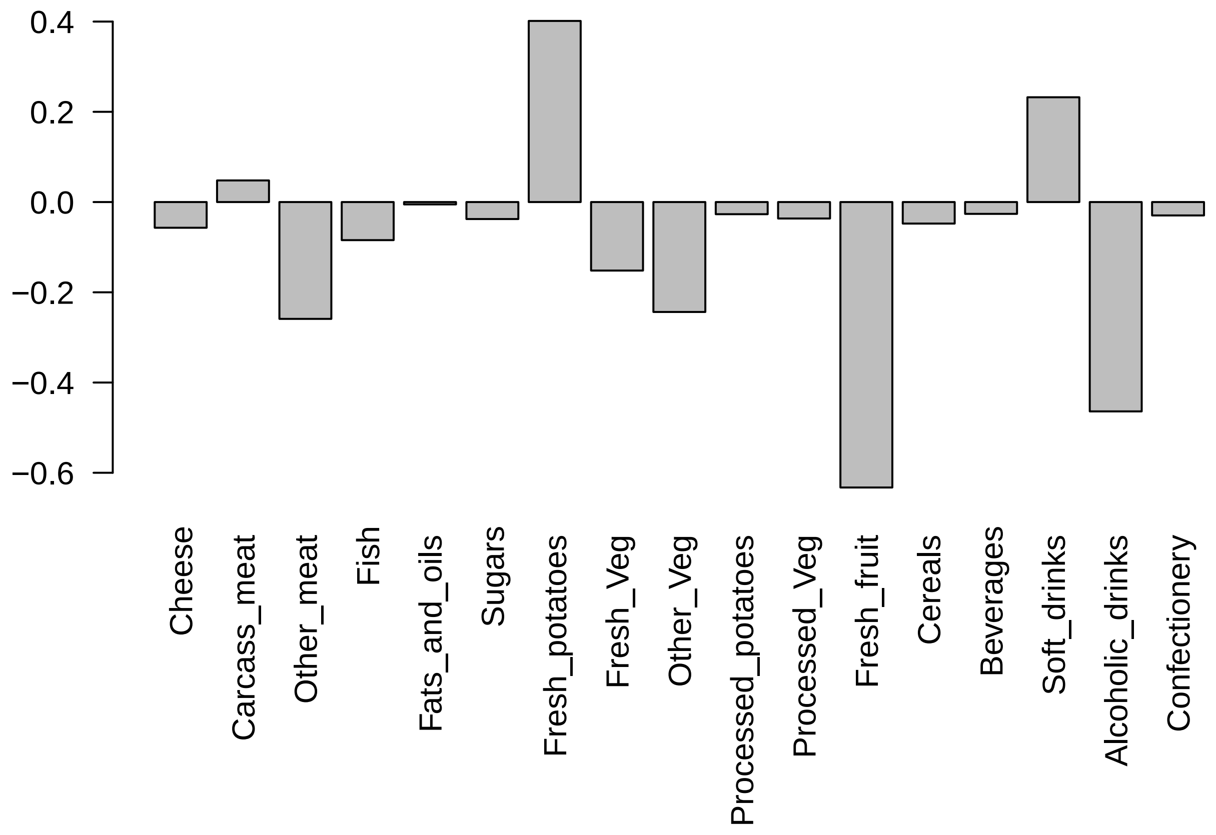
```r
barplot(v, xlab="Principal Component", ylab="Percent variation")
```



Digging deeper (variable loadings)

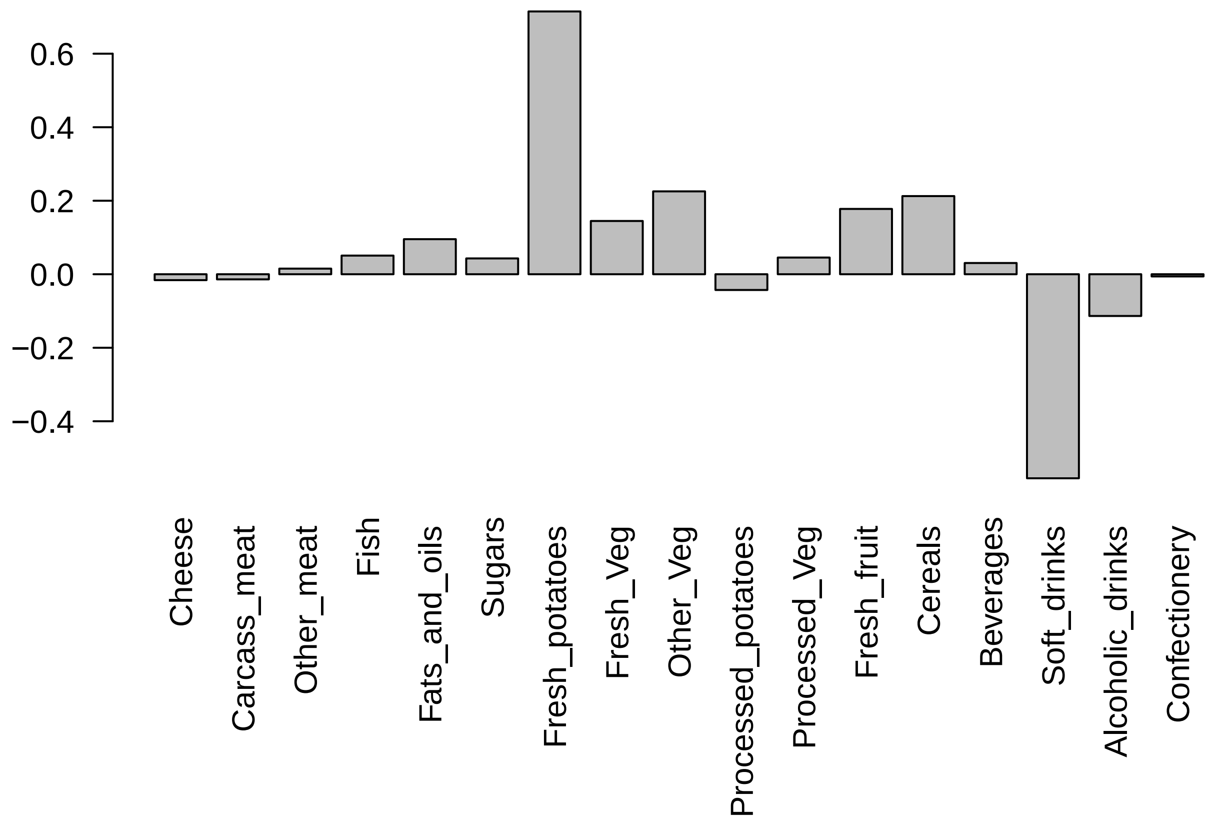
We can also consider the influence of each of the original variables upon the principal components (typically known as **loading scores**). This information can be obtained from the **prcomp()** returned `$rotation` component. It can also be summarized with a call to **biplot()**, see below:

```r
# Let's focus on PC1 as it accounts for almost 70% of the 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 mainly tell us about?

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

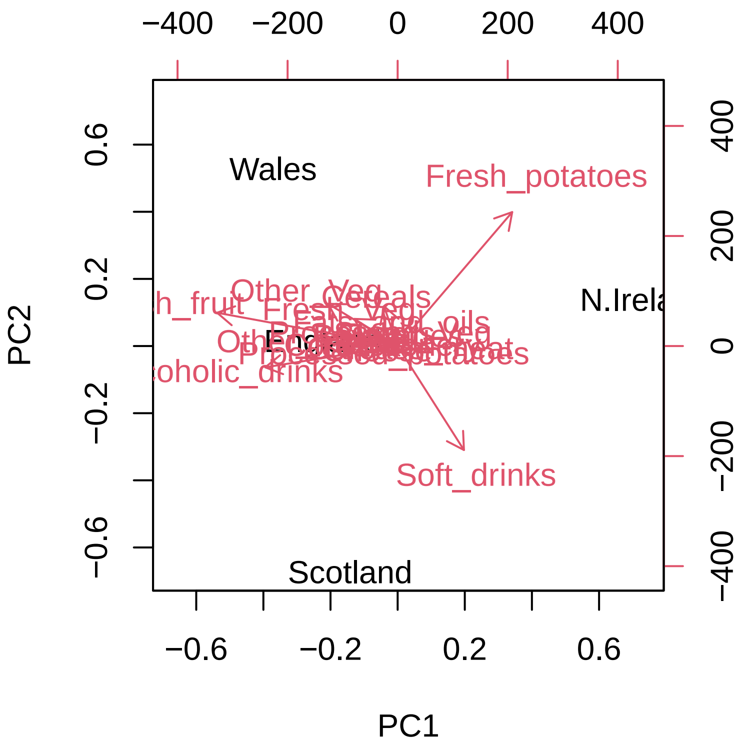


The two food that feature prominently are `Fresh_potatoes` and `Soft_drinks`, telling us that the former pushes Ireland to the right side of the plot while the latter pushes all the other countries to the left. PC2 mainly tells us that there is lower variance in the other food groups. This is illustrated by the similar distributions and loading scores closer to 0.

Biplots

This is another way to see the information together with the main PCA plot.

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



The two food groups `Fresh_potatoes` and `Soft_drinks` are apparent and visibly different. `Alcoholic_drinks` and `Fresh_fruit` are also noticeably different.

2. PCA of RNA-seq data

Input the 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
```

***Note**: The samples are columns, and the genes are rows!*

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

```
nrow(x)
```

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

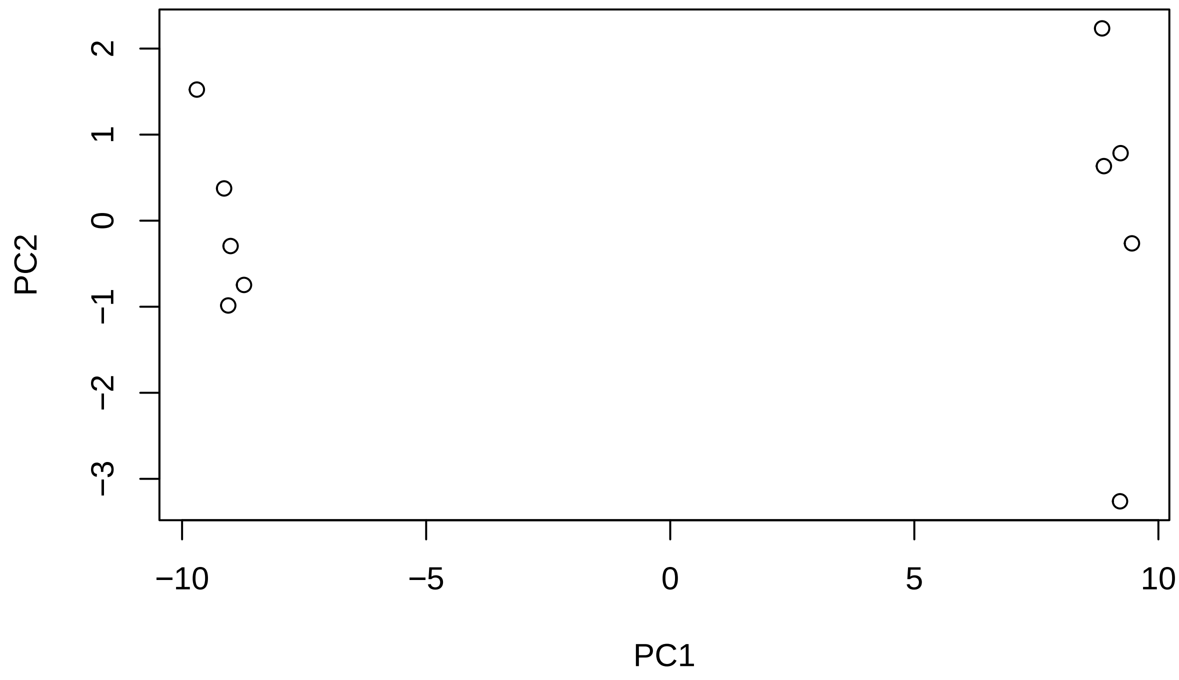
```
ncol(x)
```

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

17 rows, 4 columns

Generating barplots etc. to make sense of this data is really not an exciting or worthwhile option to consider. So lets do PCA and plot the results:

```
## 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")
```



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

```
summary(pca)
```

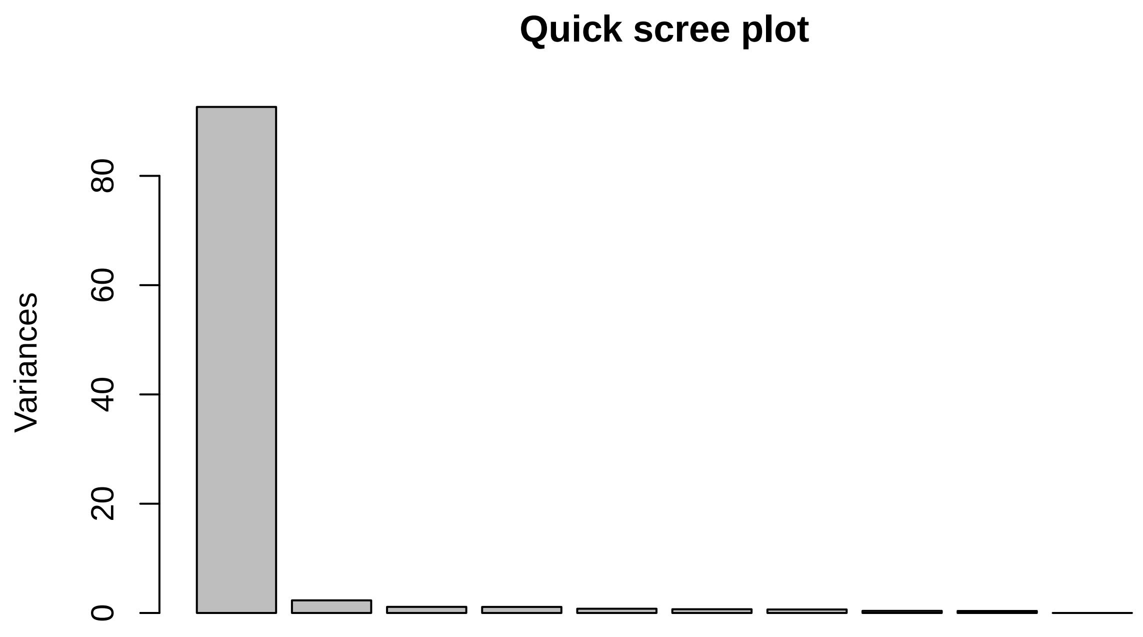
```
## Importance of components:
```

```
##                             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
```

PC1 is where all the action is (accounts for 92.6% of the variation!)

A quick barplot summary of this Proportion of Variance for each PC can be obtained by calling the `plot()` function directly on our prcomp result object.

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



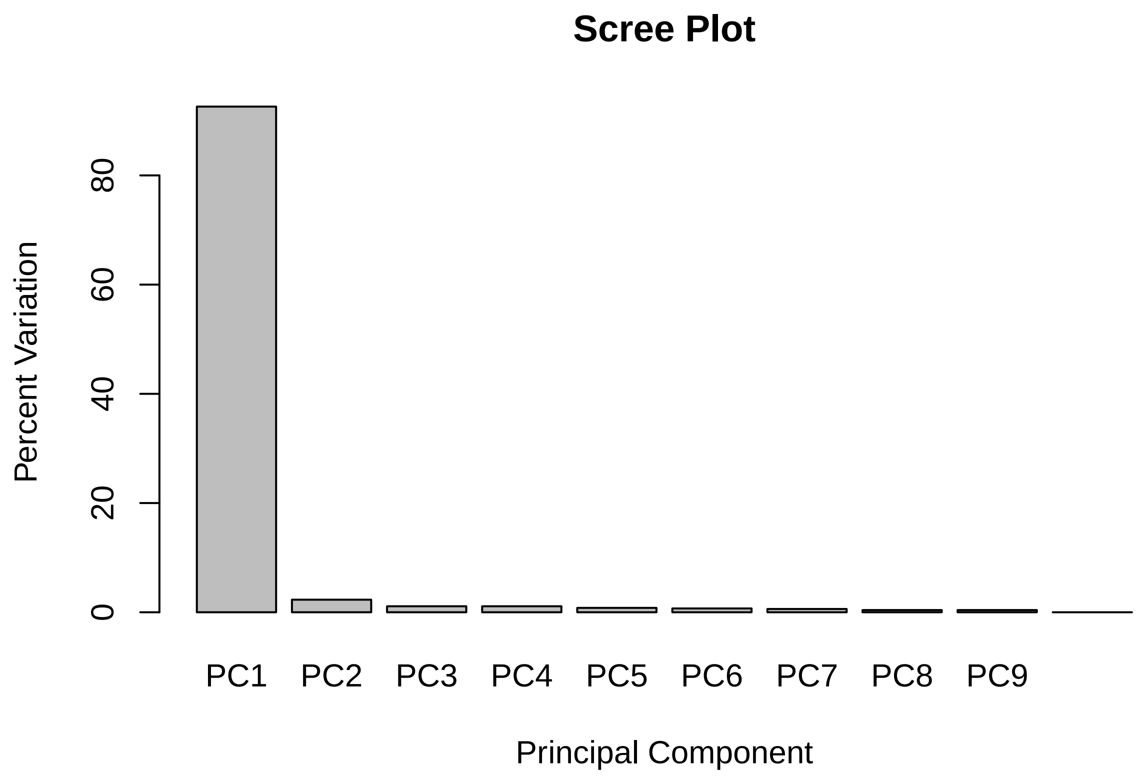
We can use the square of `pra$sdev` to calculate how much variation in the original data each PC accounts for:

```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
```

Generate a scree-plot:

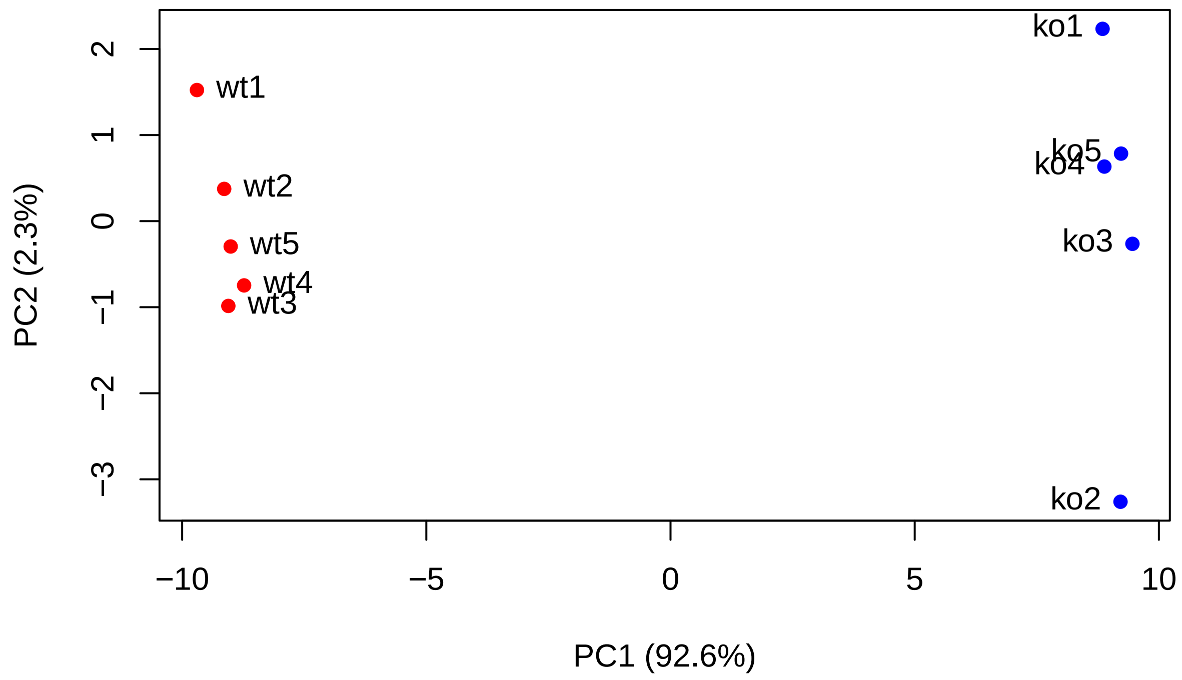
```r
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



Again, this tells us that PC1 accounts for almost all the variation.

Now make the main PCA plot more aesthetic.

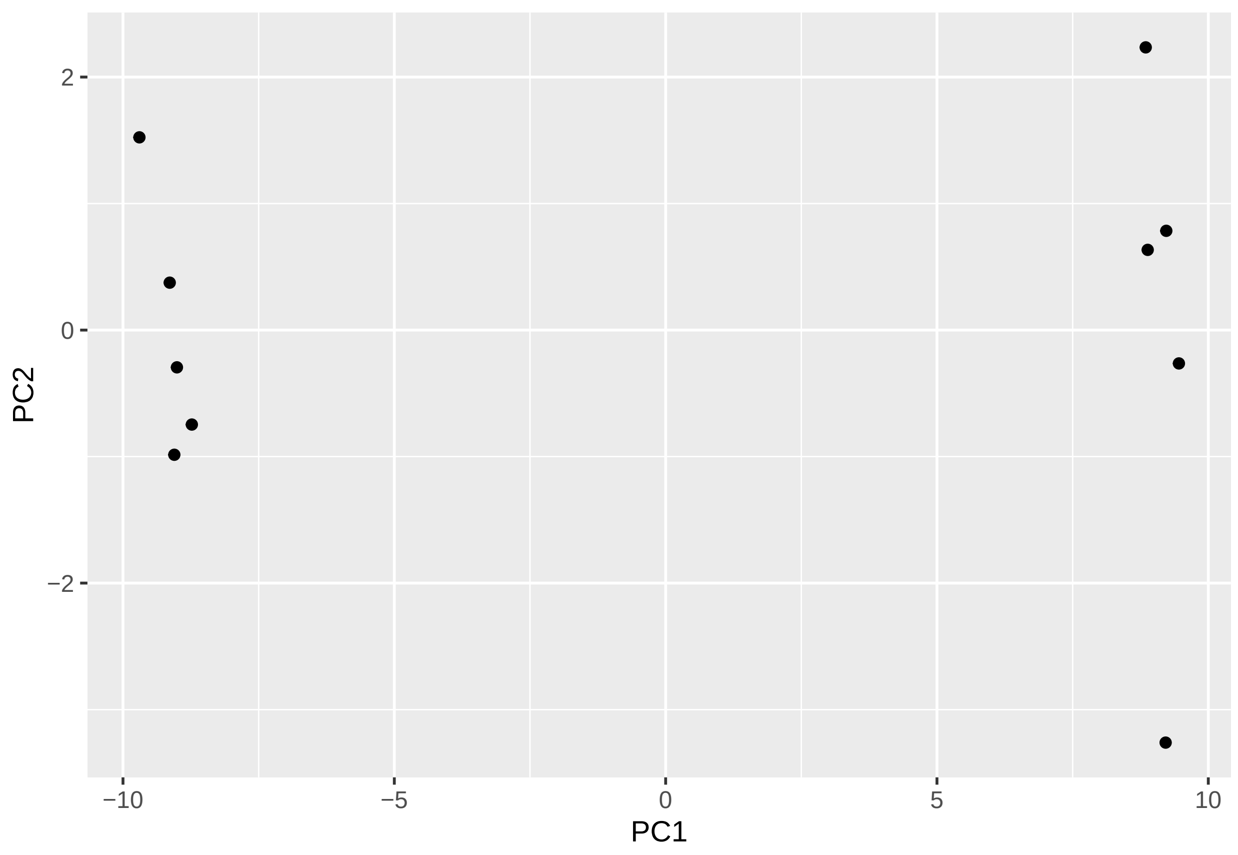
```r
# A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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

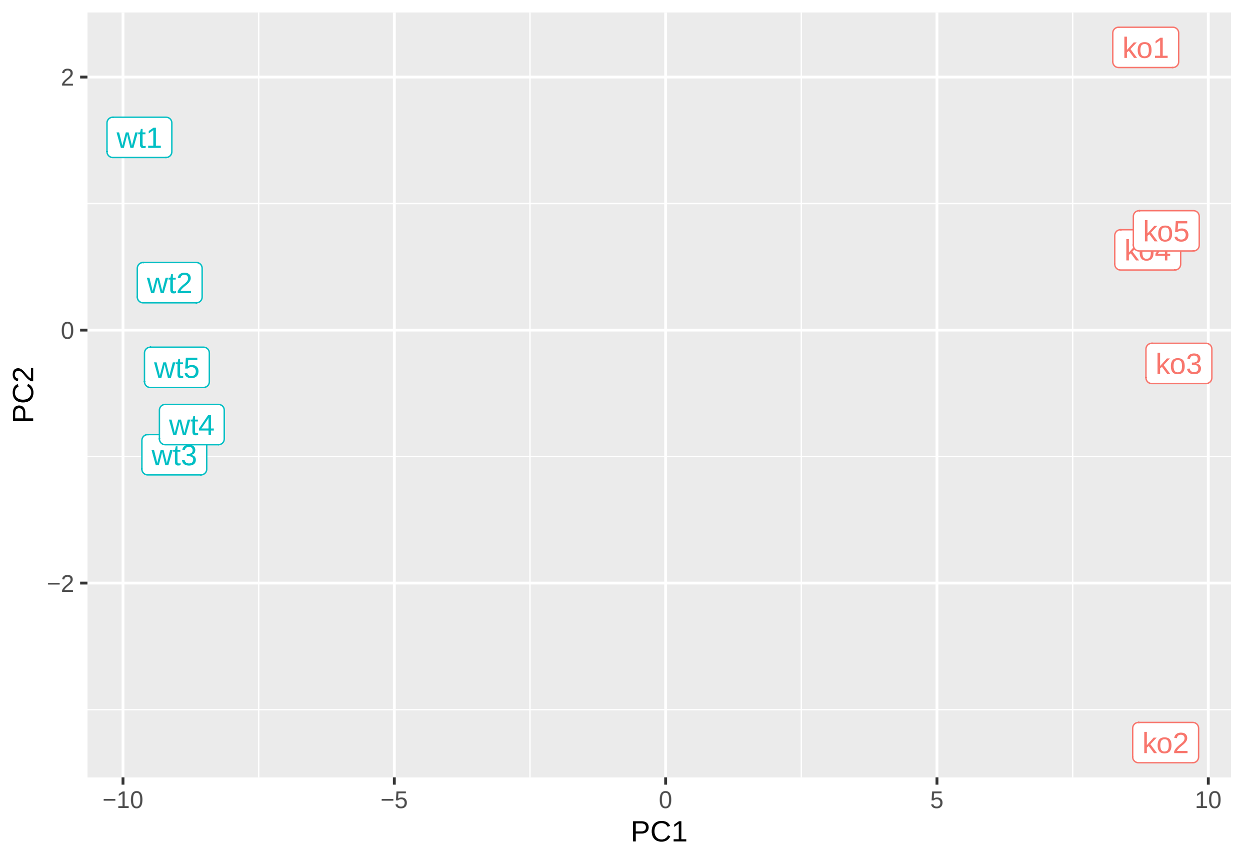
Visualize with ggplot2

```
library(ggplot2)

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

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



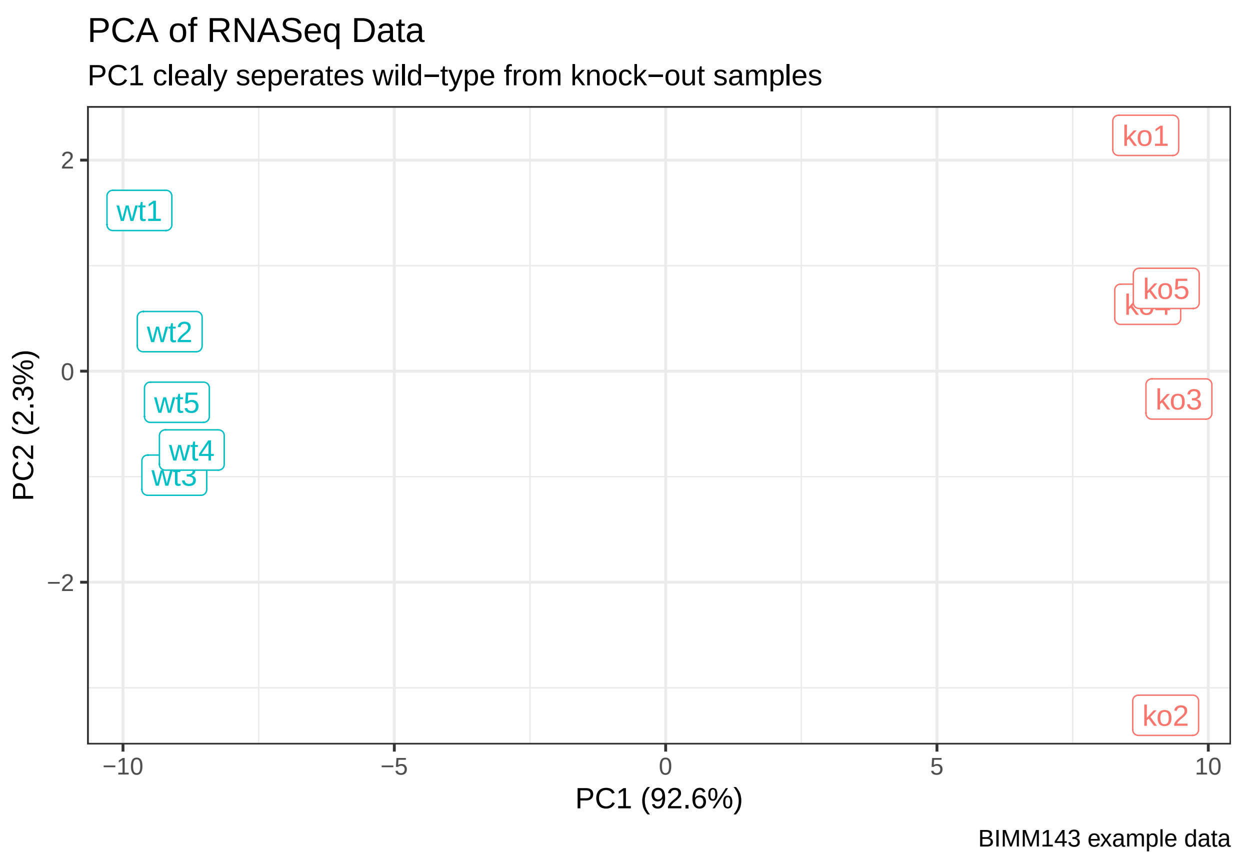
Add a condition-specific color and label aesthetics for wild-type and knock-out samples:

```
# 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
```



Finally, polish the plot:

```
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

For demonstration purposes, let's find 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"
```

These may be the genes we would like to focus on for further analysis – if their expression changes are significant.