class 7: Hands on with Principal Component Analysis (PCA)

Monika Ramos

1. PCA of UK food data Suppose that we are examining the following data, from the UK's 'Department for Environment, Food and Rural Affairs' (DEFRA), showing the consumption in grams (per person, per week) of 17 different types of food-stuff measured and averaged in the four countries of the United Kingdom in 1997. We shall say that the 17 food types are the variables and the 4 countries are the observations.

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 this questions?

```
dim(x)
```

```
[1] 17  5
```

There are 17 rows and 5 columns in the data frame 'x'.

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).

```
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
```

```
#View(x)
tail(x)
```

```
                   X England Wales Scotland N.Ireland
12      Fresh_fruit     1102  1137      957       674
13          Cereals     1472  1582     1462      1494
14         Beverages       57    73       53        47
15      Soft_drinks     1374  1256     1572      1506
16 Alcoholic_drinks      375   475      458       135
17    Confectionery       54    64       62        41
```

Here it appears that the row-names are incorrectly set as the first column of our x data frame (rather than set as proper row-names). This is very common error. Lets try to fix this up with the following code, which sets the rownames() to the first column and then removes the troublesome first column (with the -1 column index):

```
# 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 check 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 filie 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)

```
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
```

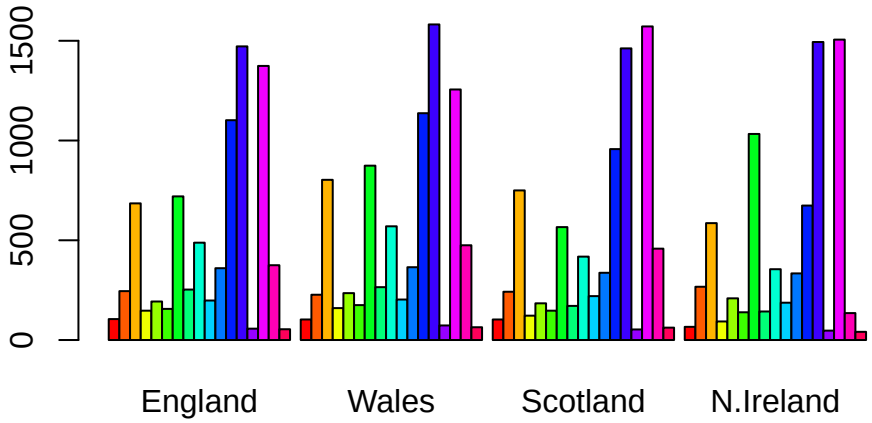
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

I prefer the second approach because it performs the necessary actions in one line of code rather than a couple.

Spotting major differences and trends.

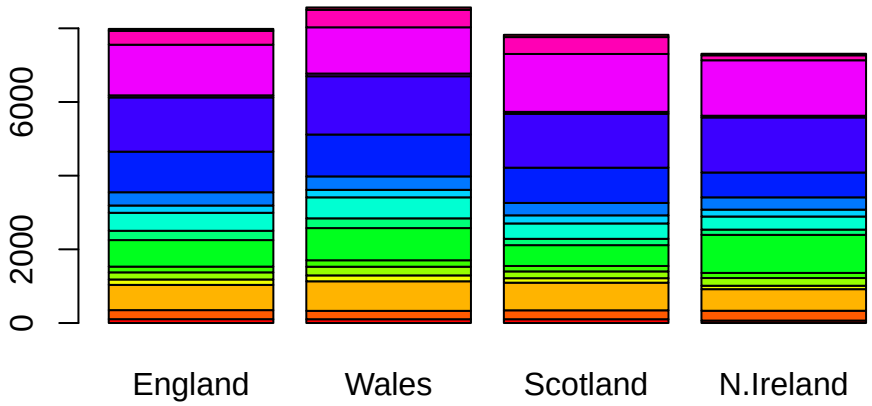
A cursory glance over the numbers in this table does not reveal much of anything. Indeed in general it is difficult to extract meaning in regard to major differences and trends from any given array of numbers. Generating regular bar-plots and various pairwise plots does not help too much either:

```
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?

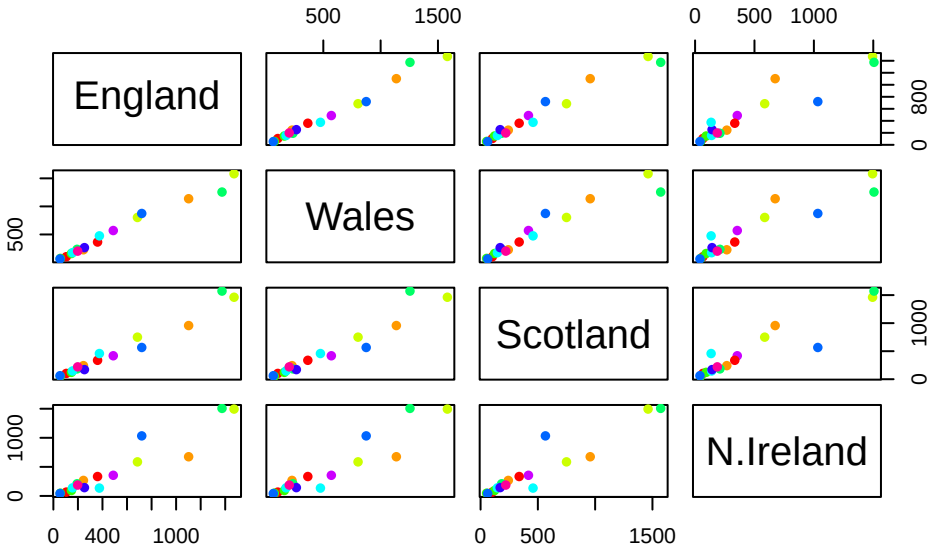
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Changing beside argument to False results in the stacked bar 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()
pairs(x, col=rainbow(10), pch=16)
```



Pairs() function is generating a matrix of scatterplots for the dataframe. These pairwise plots are comparing two countries at a time. For each food item, it plots the value in each country being compared on an x (one country) and y(other compared country) plot. If a given point lies on the diagonal this means that the countries have the same or very similar value for that item.

Even relatively small datasets can prove challenging to understand. Given that it is quite difficult to make sense of even this relatively small data set. Hopefully, we can clearly see that a powerful analytical method is absolutely necessary if we wish to observe trends and patterns in larger datasets.

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

```
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
```

It appears that the main differences of Ireland in comparison to England, Wales and Scotland is in fresh potatoes production, cheese, and alcoholic drinks with more dramatic differences.

PCA base R prcomp() function. 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)
```

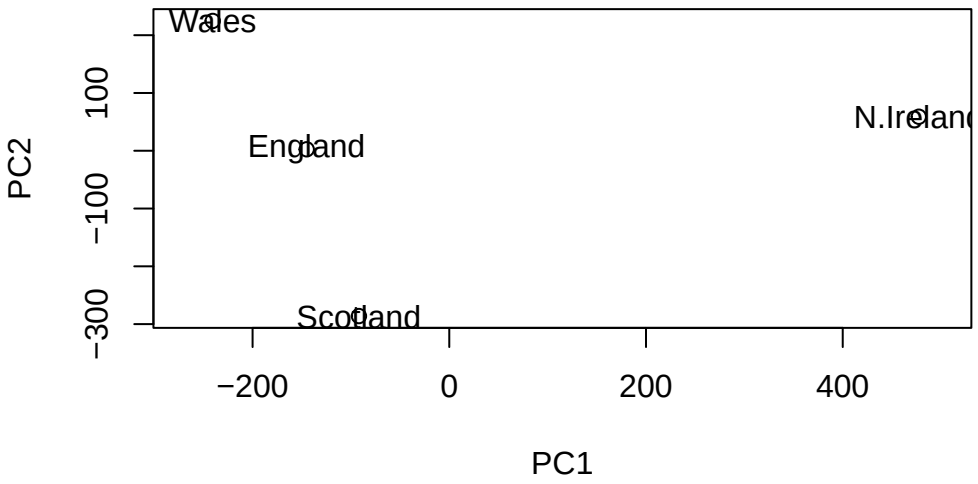
```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

The first task of PCA is to identify a new set of principal axes through the data. This is achieved by finding the directions of maximal variance through the coordinates in the 17 dimensional space. This is equivalent to obtaining the (least-squares) line of best fit through the plotted data where it has the largest spread. We call this new axis the first principal component (or PC1) of the data. The second best axis PC2, the third best PC3 etc.

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.

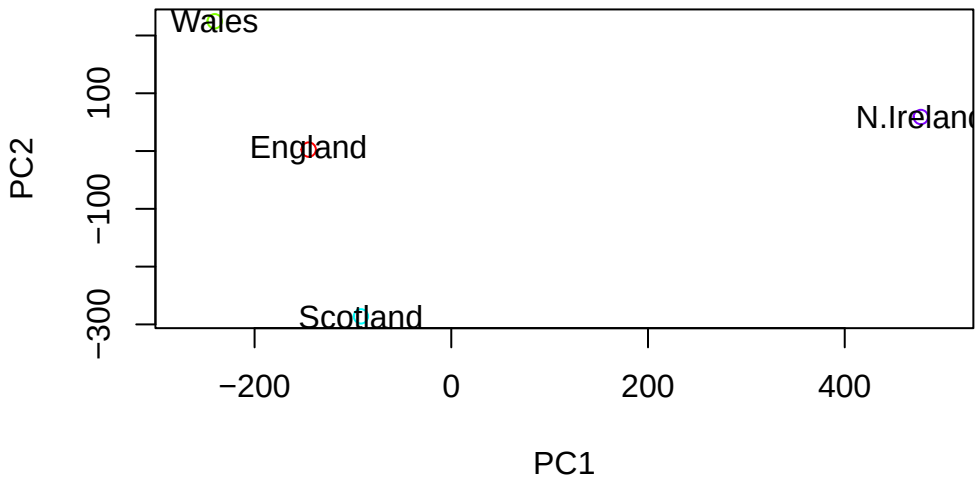
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

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

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



In practice, it is usually sufficient to include enough principal components so that somewhere in the region of 70% of the variation in the data is accounted for. Looking at the so-called scree plot can help in this regard

Below 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
```

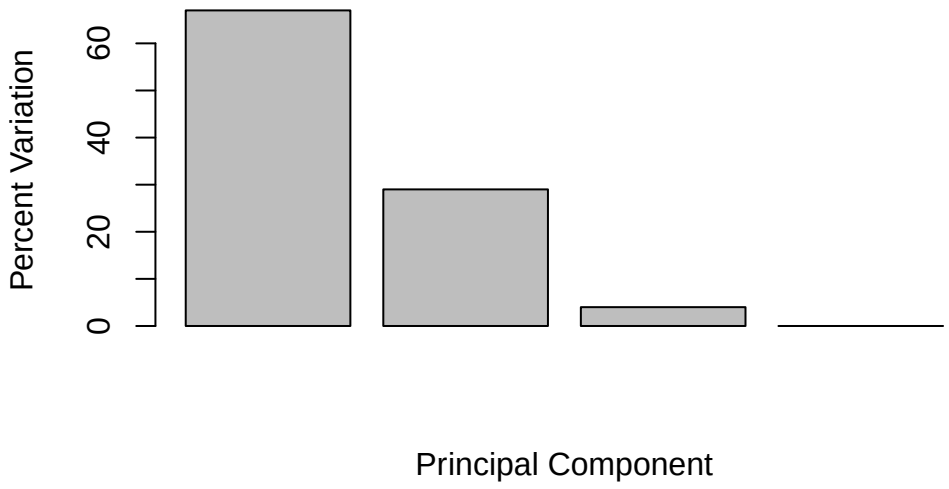
```
[1] 67 29  4  0
```

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

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

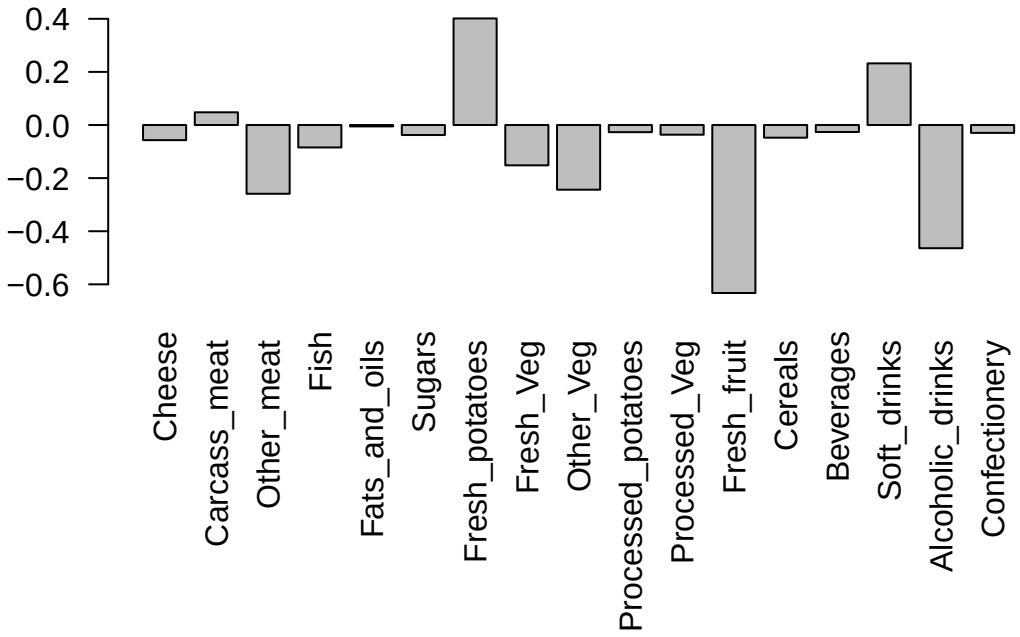
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



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:

```
## 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 )
```

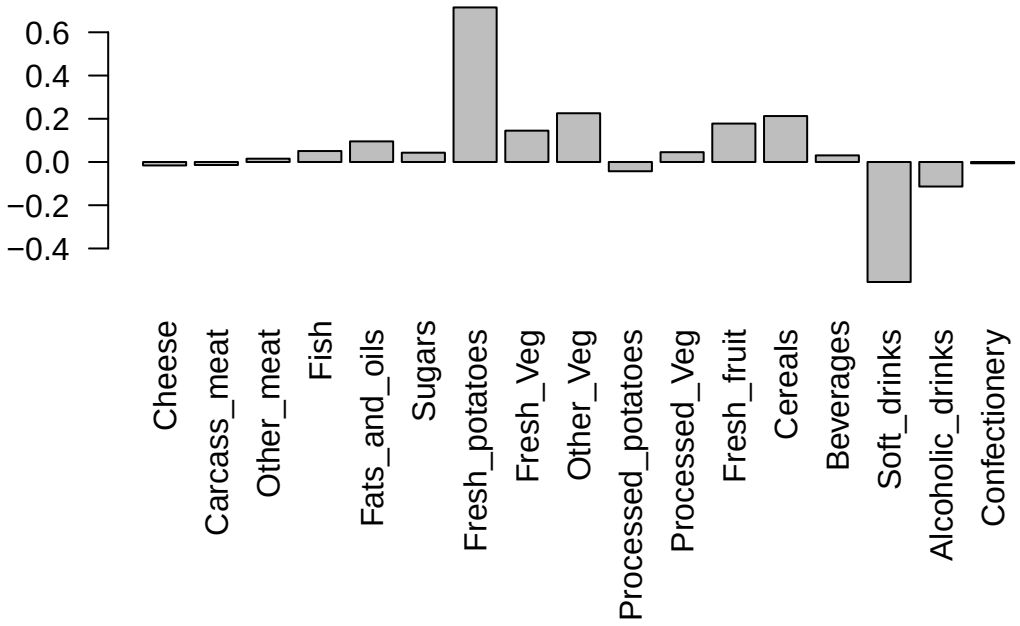


Here we see observations (foods) with the largest positive loading scores that effectively "push" N. Ireland to right positive side of the plot (including Fresh_potatoes and Soft_drinks).

We can also see the observations/foods with high negative scores that push the other countries to the left side of the plot (including Fresh_fruit and Alcoholic_drinks).

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

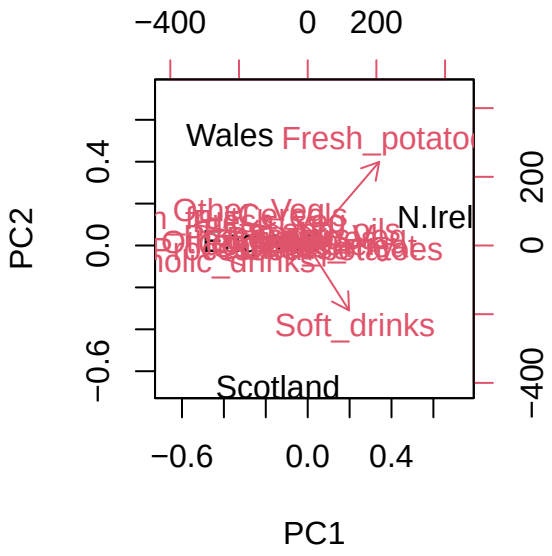
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



The two food groups that are featured prominently in the second PC are potatoes and soft drinks. PC2 mainly tells us about the 29% of variation captured among the dataset. We can see the second largest depicted features contributing to variance in the dataset.

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)
```



2. PCA of RNA-seq data RNA-seq results often contain a PCA (or related MDS plot). Usually we use these graphs to verify that the control samples cluster together. However, there's a lot more going on, and if you are willing to dive in, you can extract a lot more information from these plots. The good news is that PCA only sounds complicated. Conceptually, as we have hopefully demonstrated here and in the lecture, it is readily accessible and understandable.

In this example, a small RNA-seq count data set (available from the class website (expression.csv and the tinyurl short link: "https://tinyurl.com/expression-CSV") is read into a data frame called rna.data where the columns are individual samples (i.e. cells) and rows are measurements taken for all the samples (i.e. genes).

```
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?

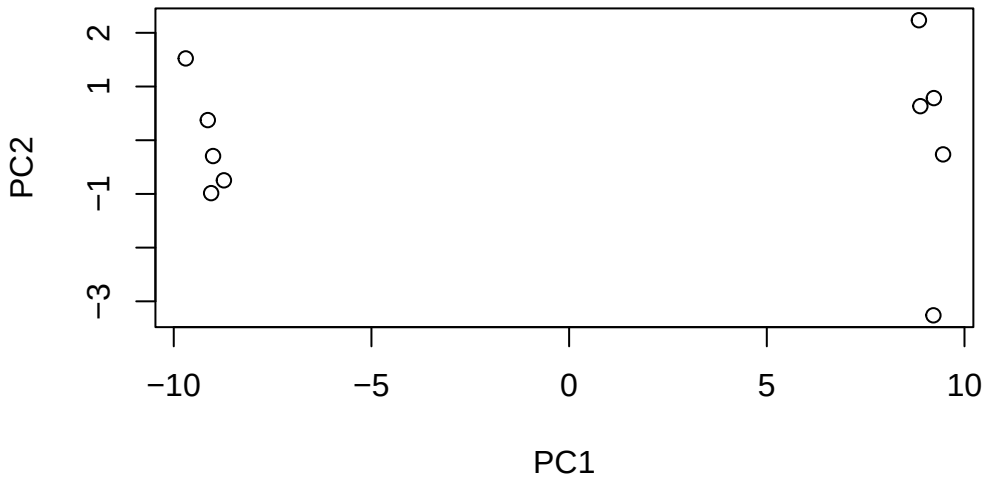
```
dim(rna.data)
```

```
[1] 100  10
```

There are 100 genes and 10 samples in this rna seq dataset.

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")
```



This quick plot looks interesting with a nice separation of samples into two groups of 5 samples each. Before delving into the details of this grouping let's first examine a summary of how much variation in the original data each PC accounts for:

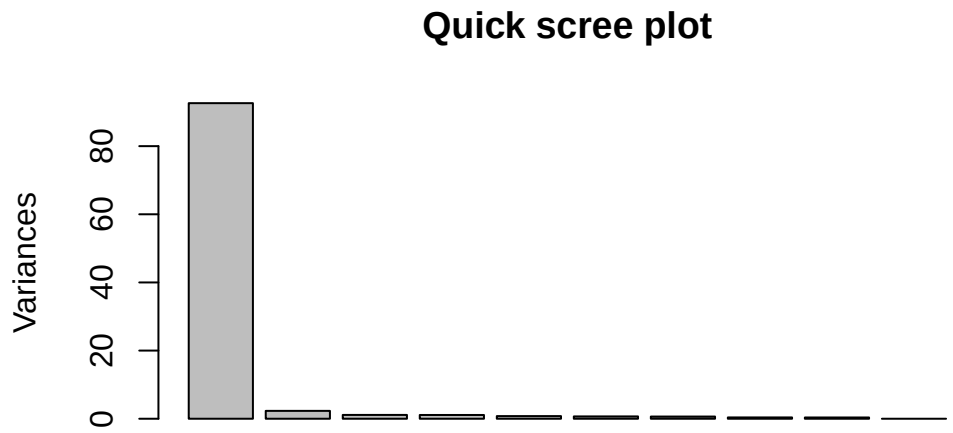
```
summary(pca)
```

```
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 this results that PC1 is were all the action is (92.6% of it in fact!). This indicates that we have sucesfully reduced a 100 diminesional data set down to only one dimension that retains the main essential (or principal) features of the origional data. PC1 captures 92.6% of the origional varance with the first two PCs capturing 94.9%. This is quite amazing!

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.

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



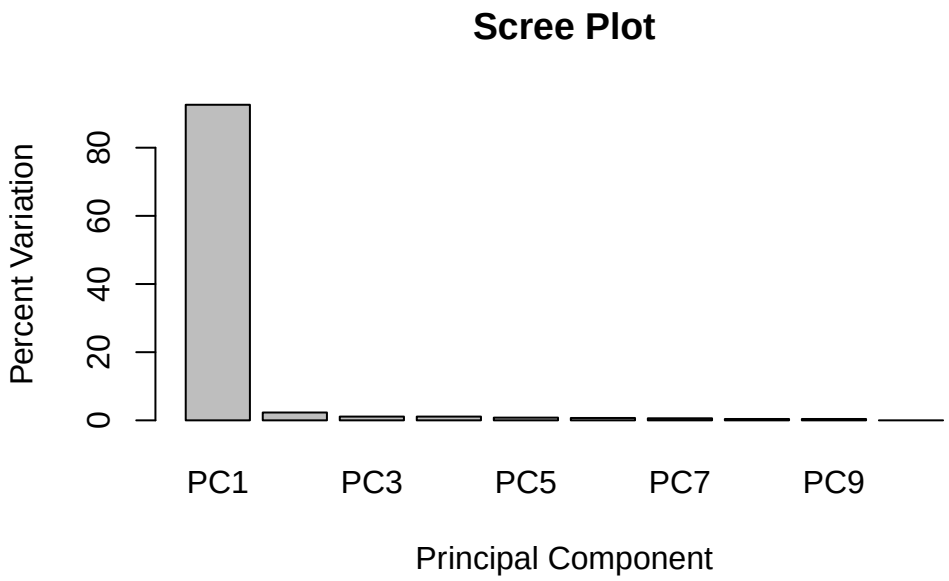
use the square of pca$sdev, which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for:

```
## 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
```

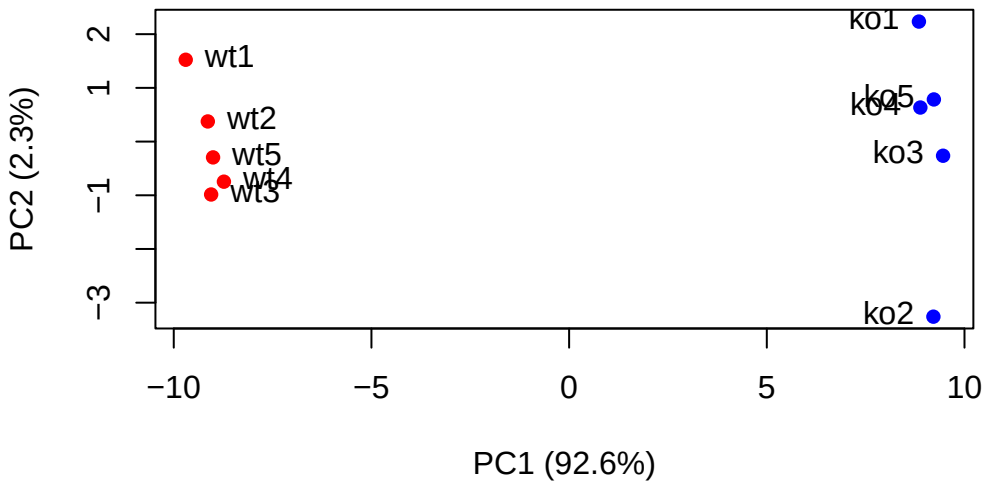
We can use this to generate our own scree-plot like this

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



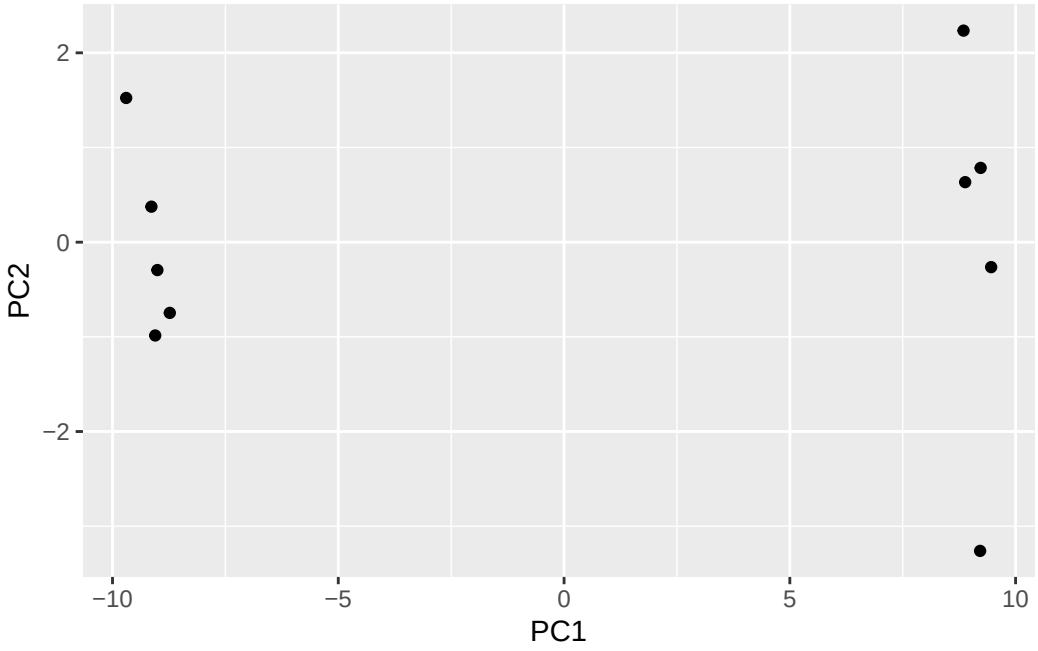

```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
```

```
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)))
```



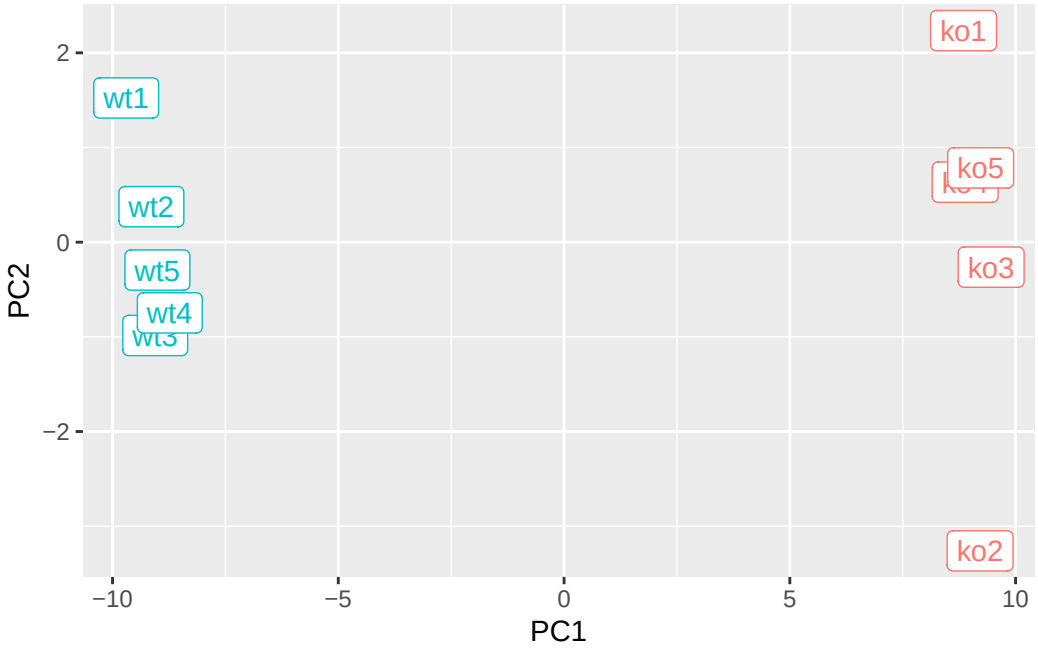
Using ggplot

```
library(ggplot2)

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

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



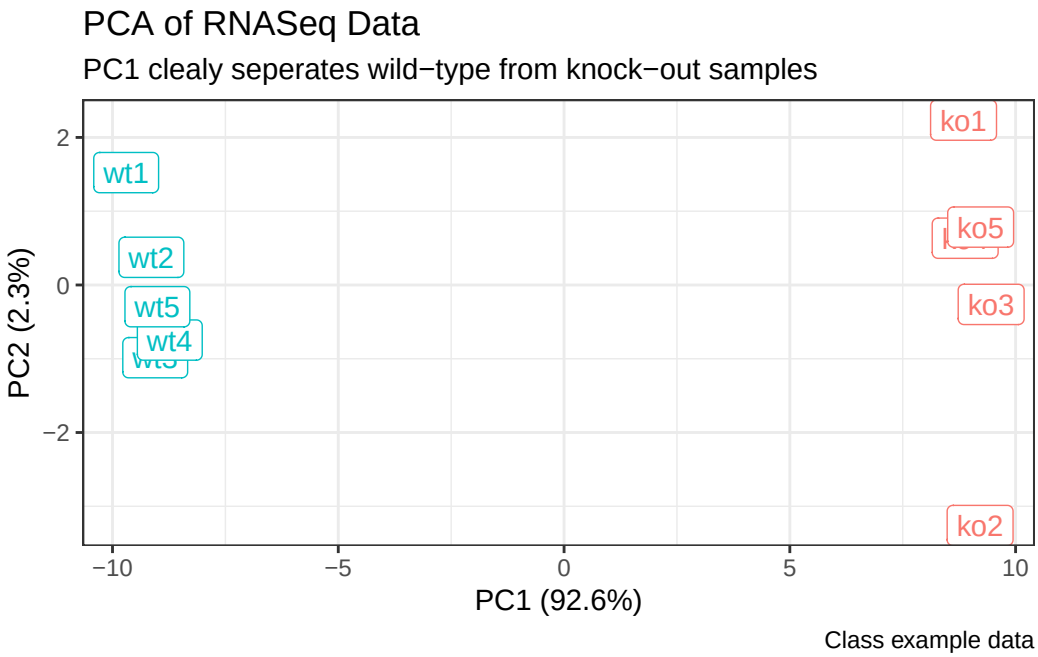
If we want to add a condition specific color and perhaps sample label aesthetics for wild-type and knock-out samples we will need to have this information added to our data.frame:

```
# 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
```



add axis details and title

```
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()
```



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"
```