lab7R

Rachel Kraft

2/12/2022

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

Assign the data set link to url and the read of this csv file to variable x

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

#Use the dim() function to find out how many rows and columns there are in x

dim(x)
```

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

Examine the data set to ensure it meets our expectations by using the head() function to print 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
```

There are only 4 columns of data, but the dim() function above told us to expect 5. We can fix this by using the rownames() function to set it to the first column.

```
rownames(x) <- x[,1]
# removes the first column with the -1 column index
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
```

Check the dimensions again to ensure our fix was correct

```
dim(x)
```

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

Another method of solving this problem is to read the data file again and set row.names of the read.csv() function to the first column (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
```

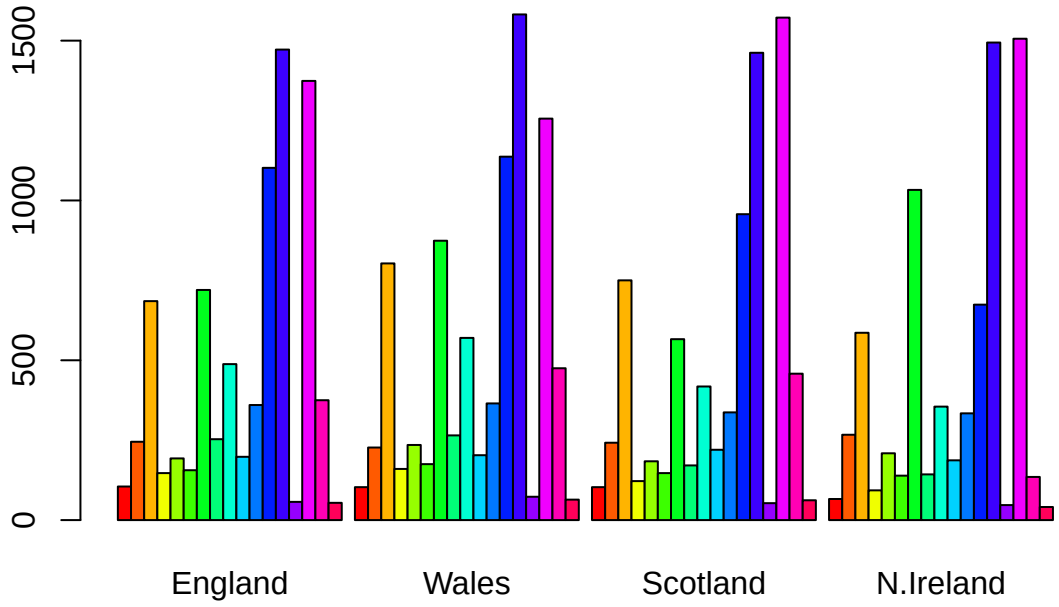
> Q2) Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

I would prefer the alternate method (the second one) to solve the row-names problem because it involves one simple command with the read.csv() function. If you run the code block for the first approach using the -1 column index, it will keep deleting the first column and our data may be accidentally deleted.

> Spotting major differences and trends

Generating regular bar plots does not help too much with looking for trends and analyzing the data, for example:

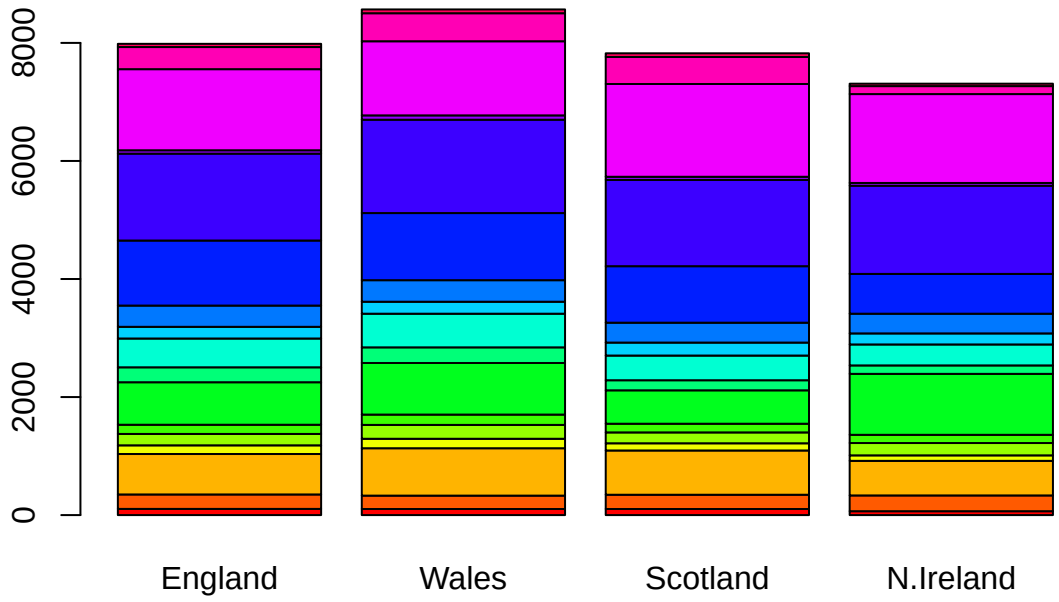
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



Q3) Changing what optional argument in the above barplot() changes it to a horizontally stacked one?

Set beside=FALSE or leave this argument out (the default)

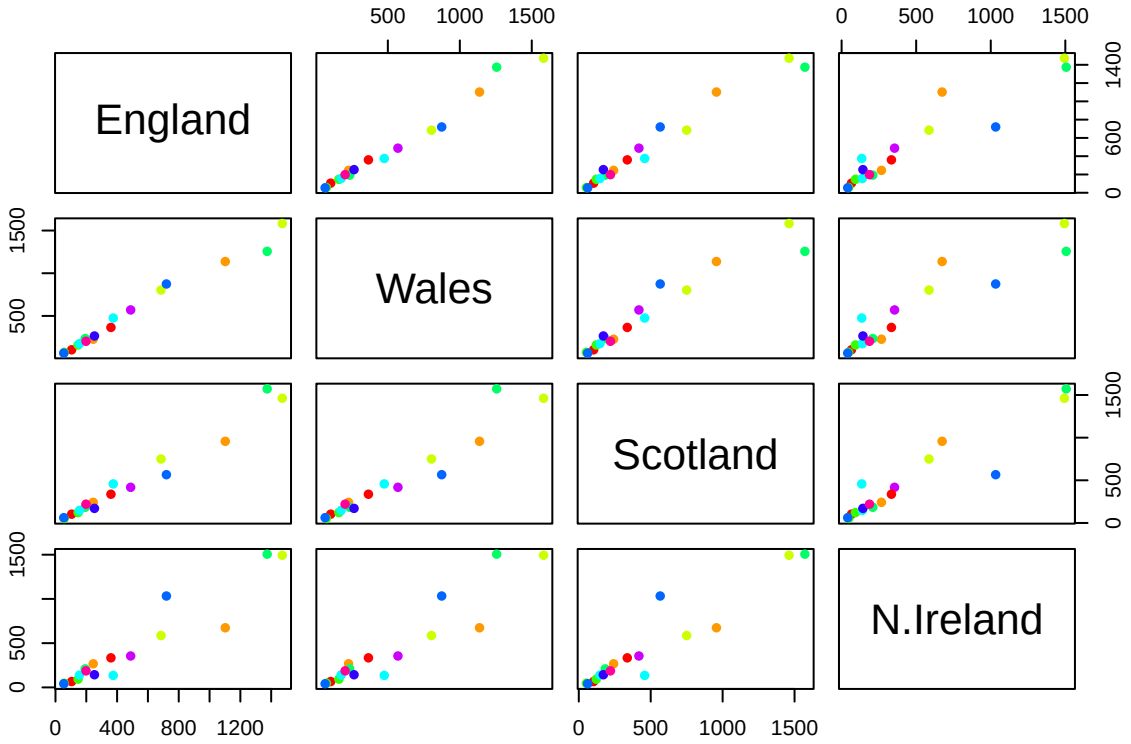
```
barplot(as.matrix(x), col=rainbow(nrow(x)))
```



Q5) Generating all pairwise plots may help. 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?

The pairs() function returns a matrix of scatterplots for dataset x. It gives a plot of all countries compared against each other. Each plot along each axis is represented by that country. The first column and row represents England, second column is Wales, etc. If the people in both countries eat the same amount, there will be a point along the diagonal line.

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



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

When comparing N.Ireland to the other countries, the points in the plots generally lie above the diagonal. This means that the values for food consumption from the data-set are generally higher in the other countries than N.Ireland.

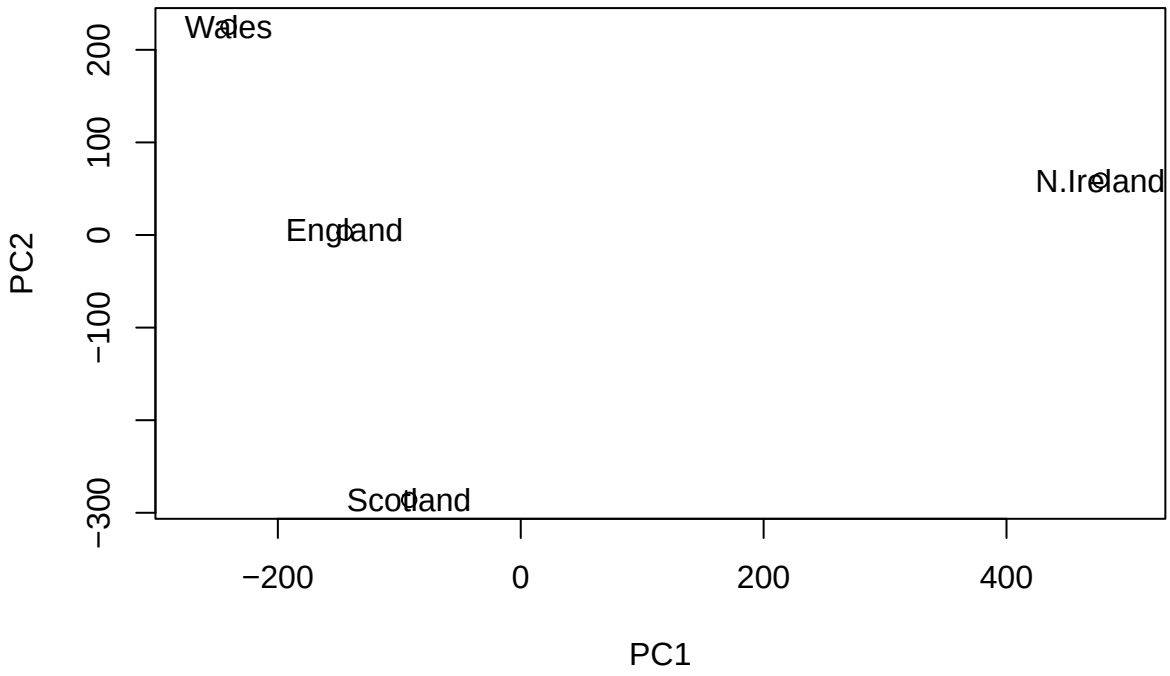
We can use PCA to make more sense of the data and look for trends. Use the prcomp() PCA function on our data and transpose our data.frame matrix with the t() function

```
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
```

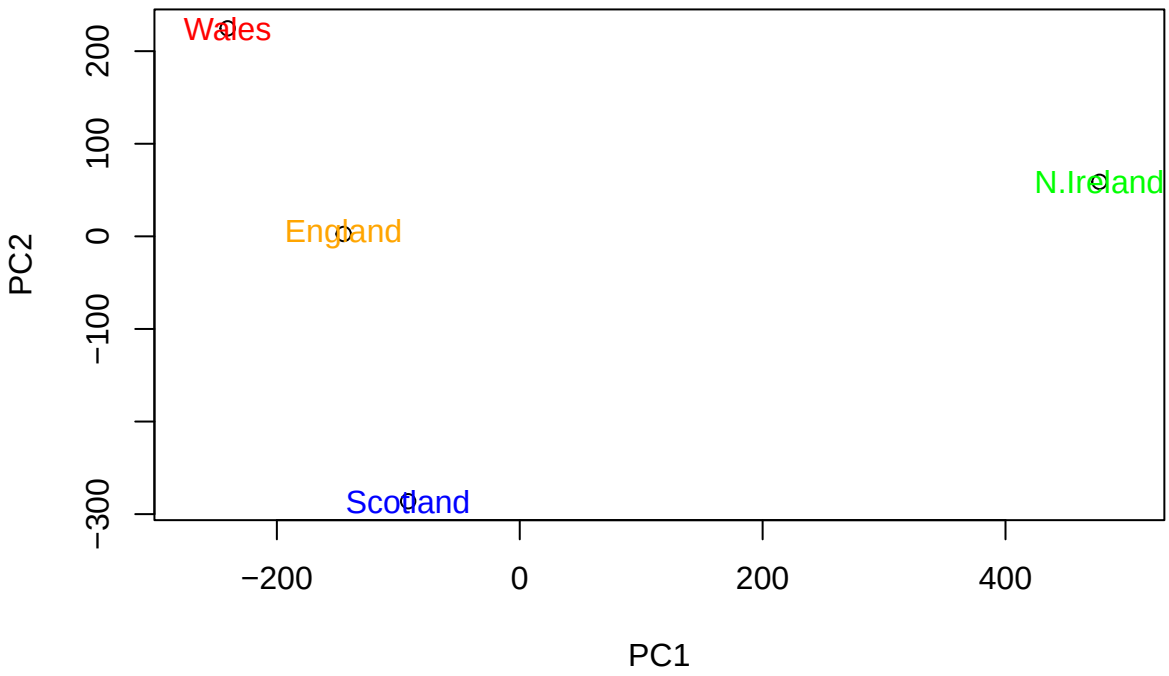
Q7) Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points

```
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 the start of this document

```
#we can provide a color vector as input to the text() function
country_cols <- 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=country_cols)
```



As part of the PCA method, usually we want to include enough principal components so 70% of the variation in data is accounted for

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

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

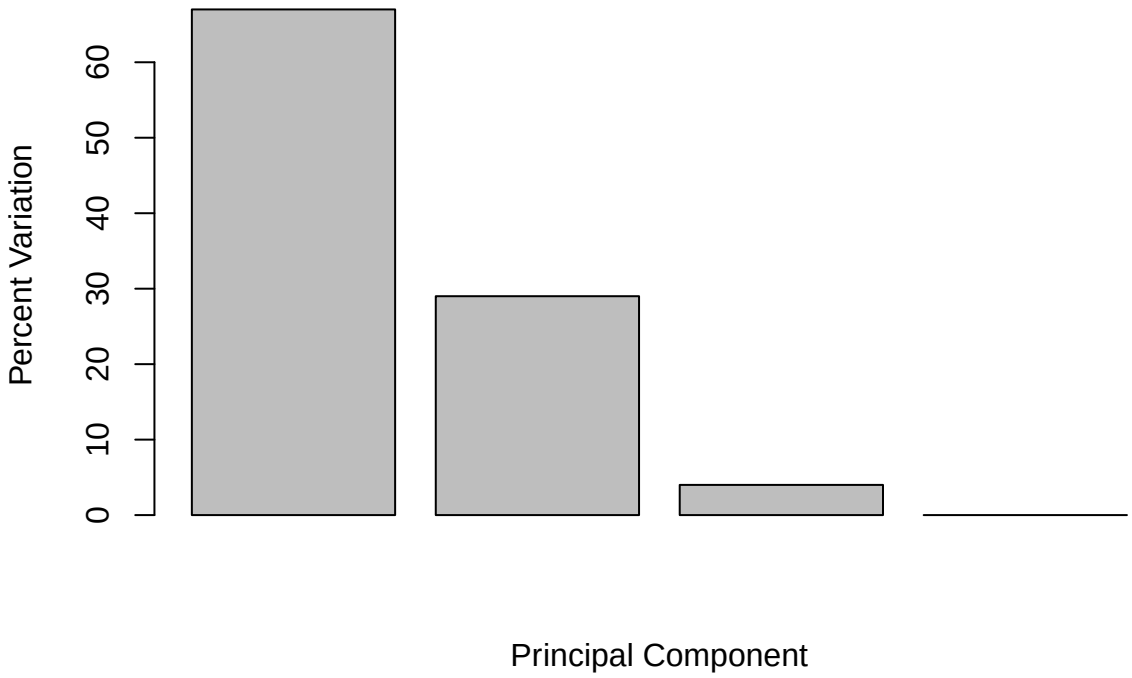
```
## [1] 67 29  4  0
```

```
## 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
```

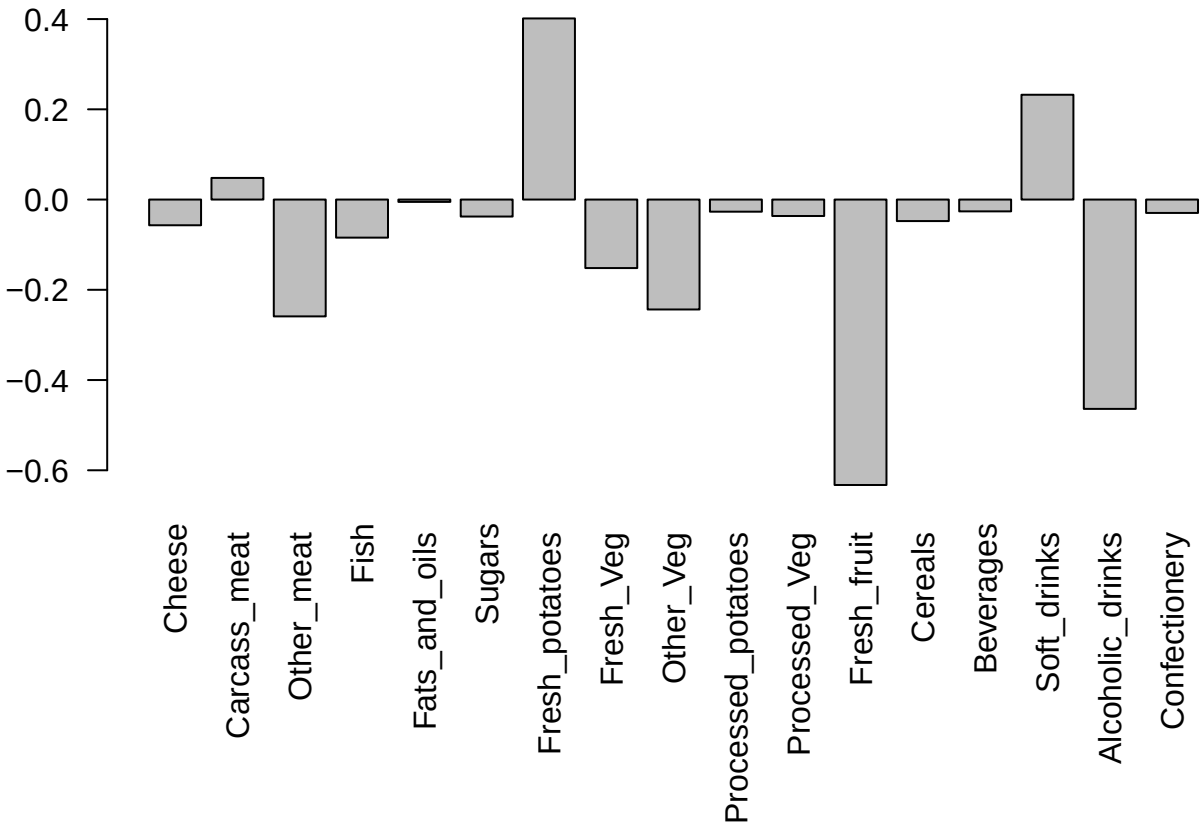
We can summarize the above information in a plot of variances with respect to the principal component number

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



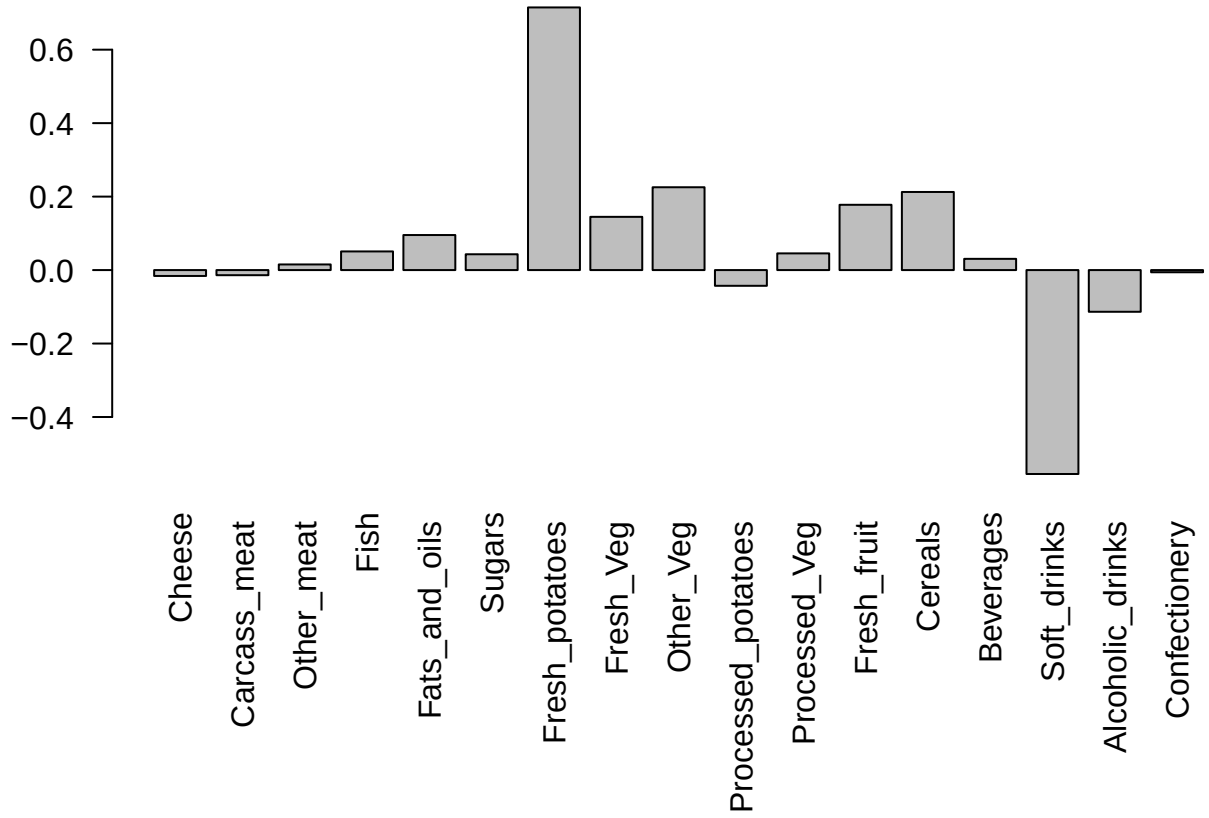
We can use loading scores from the prcomp() returned $rotation component

```
par(mar=c(10,3,0.35,0))
barplot(pca$rotation[,1], las=2)
```



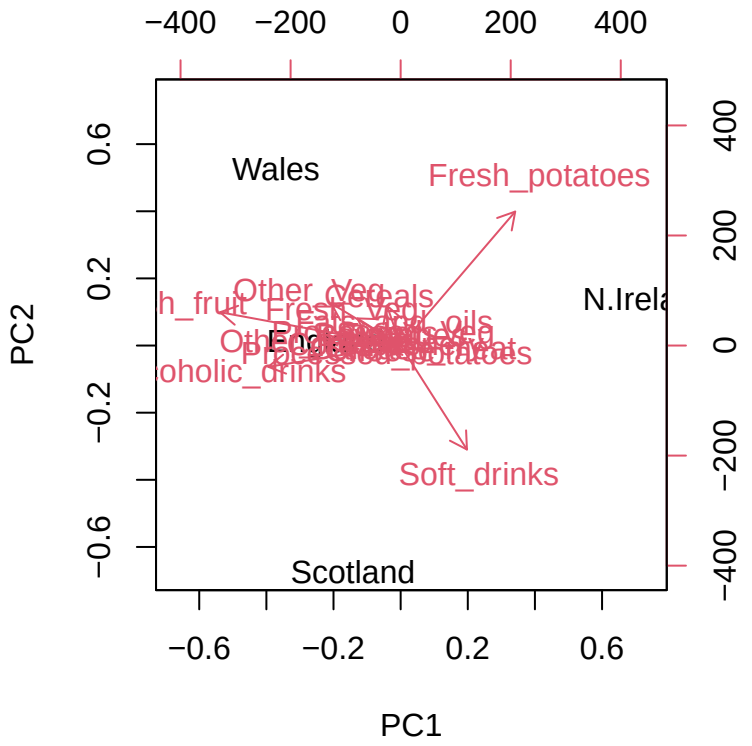
Q9) Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 mainly tell us about?

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



We can see this info together with the main PCA plot in a biplot()

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



PCA of RNA-seq data

We will read a small RNA-seq count data set into a data frame called rna.data, where columns are individual samples (cells), and rows are measurements (genes)

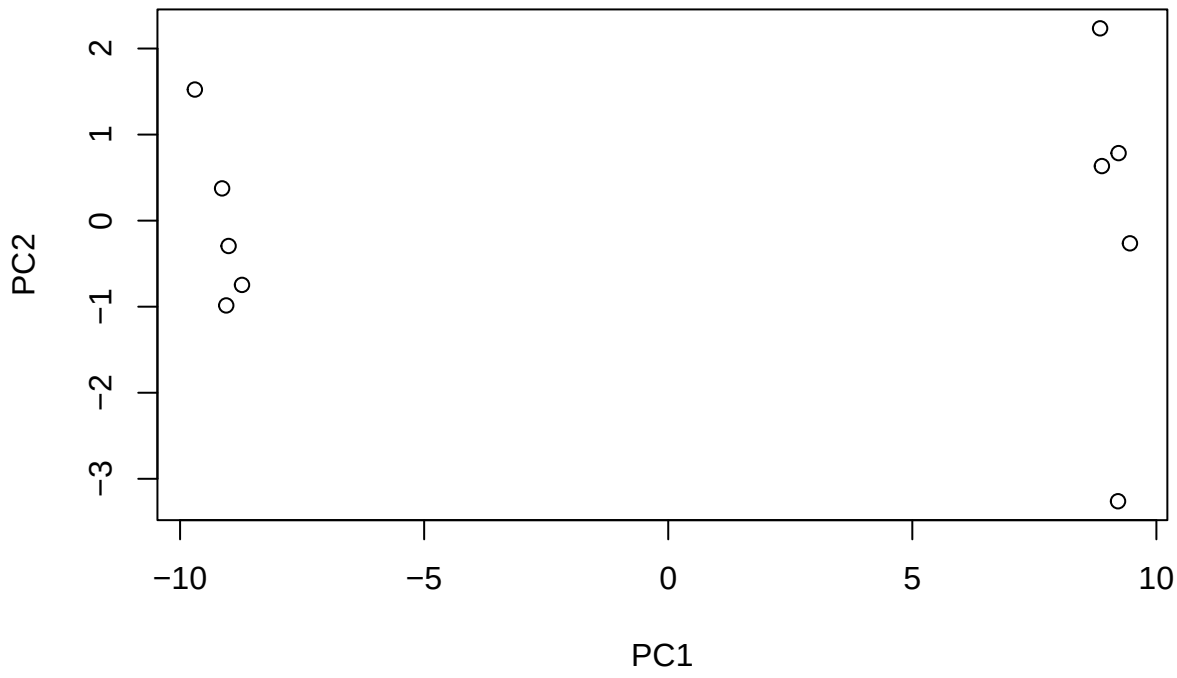
```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
##        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
```

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

Do PCA: first take the transpose of our data then create a simple plot of pc1 and pc2

```
pca <- prcomp(t(rna.data), scale=TRUE)
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```

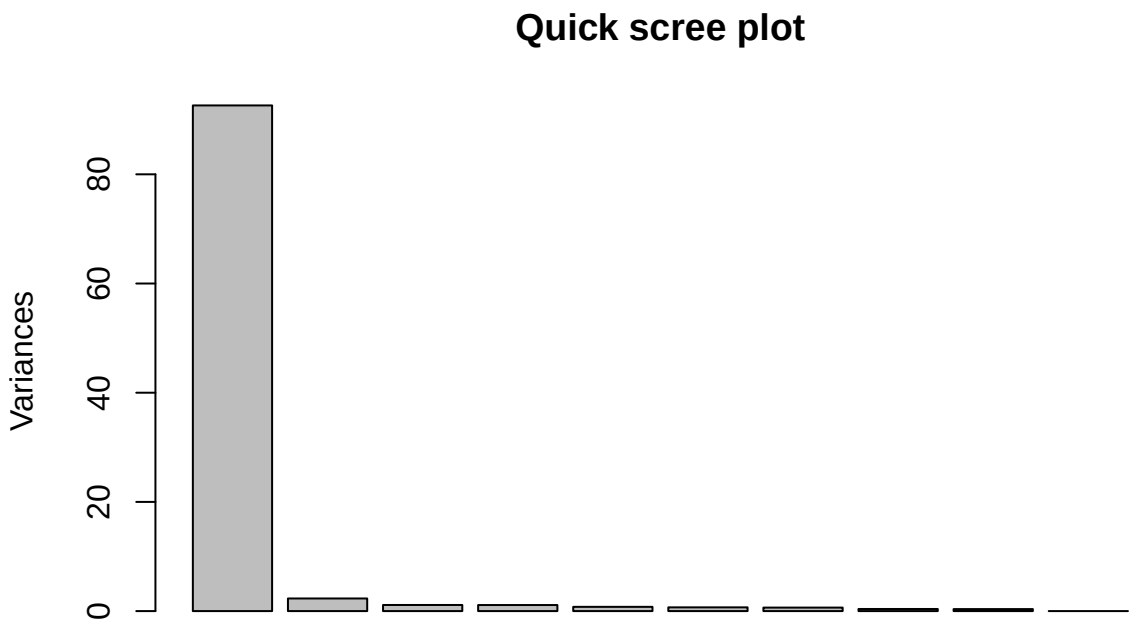


Summary of how much variation in the original data each PC accounts for

```
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
```

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



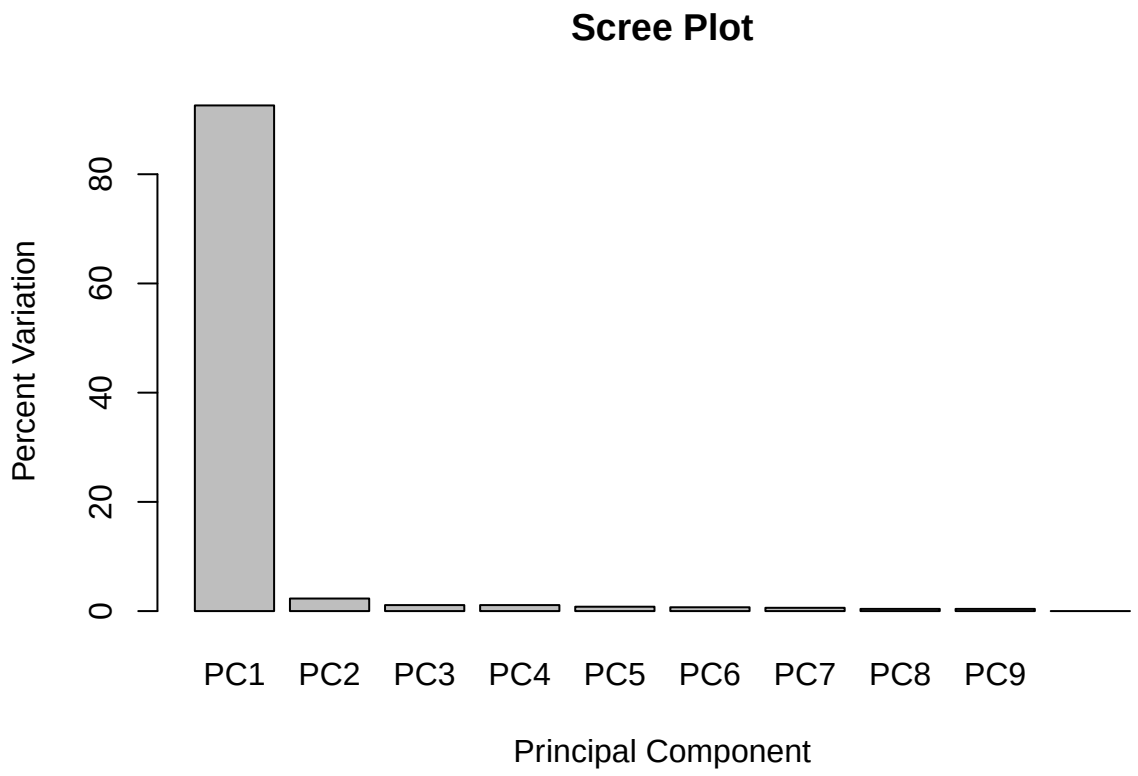
Use the square of pca$sdev to calculate how much variation each PC accounts for

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

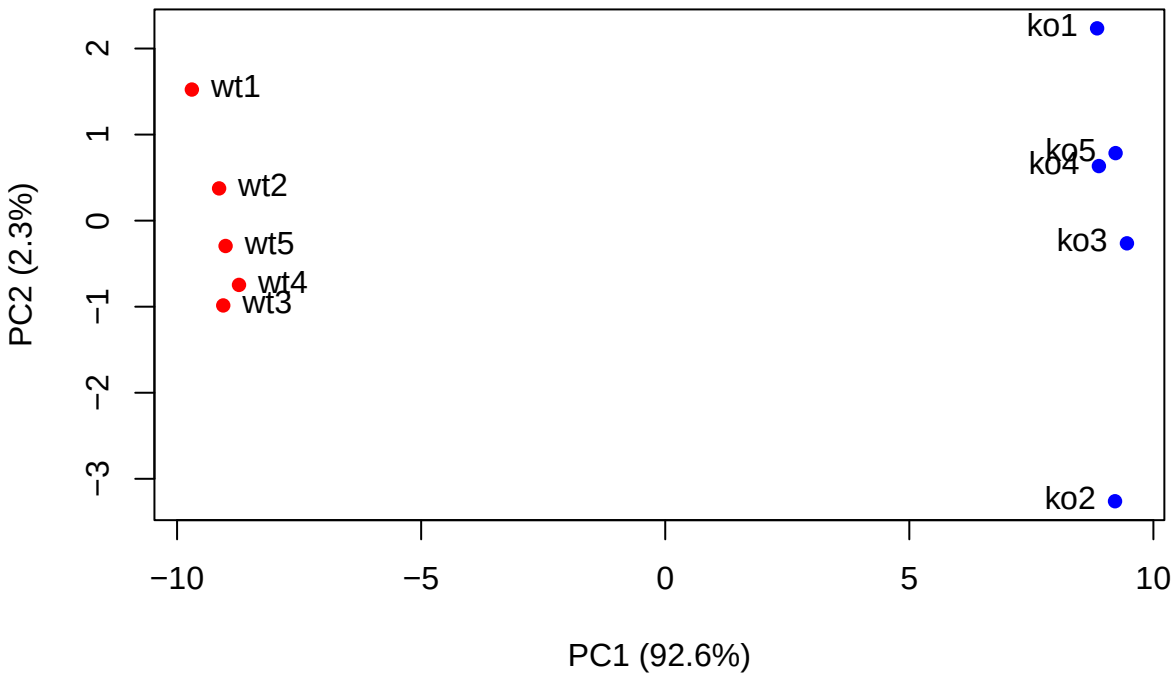
```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

Now use this to generate own scree-plot

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

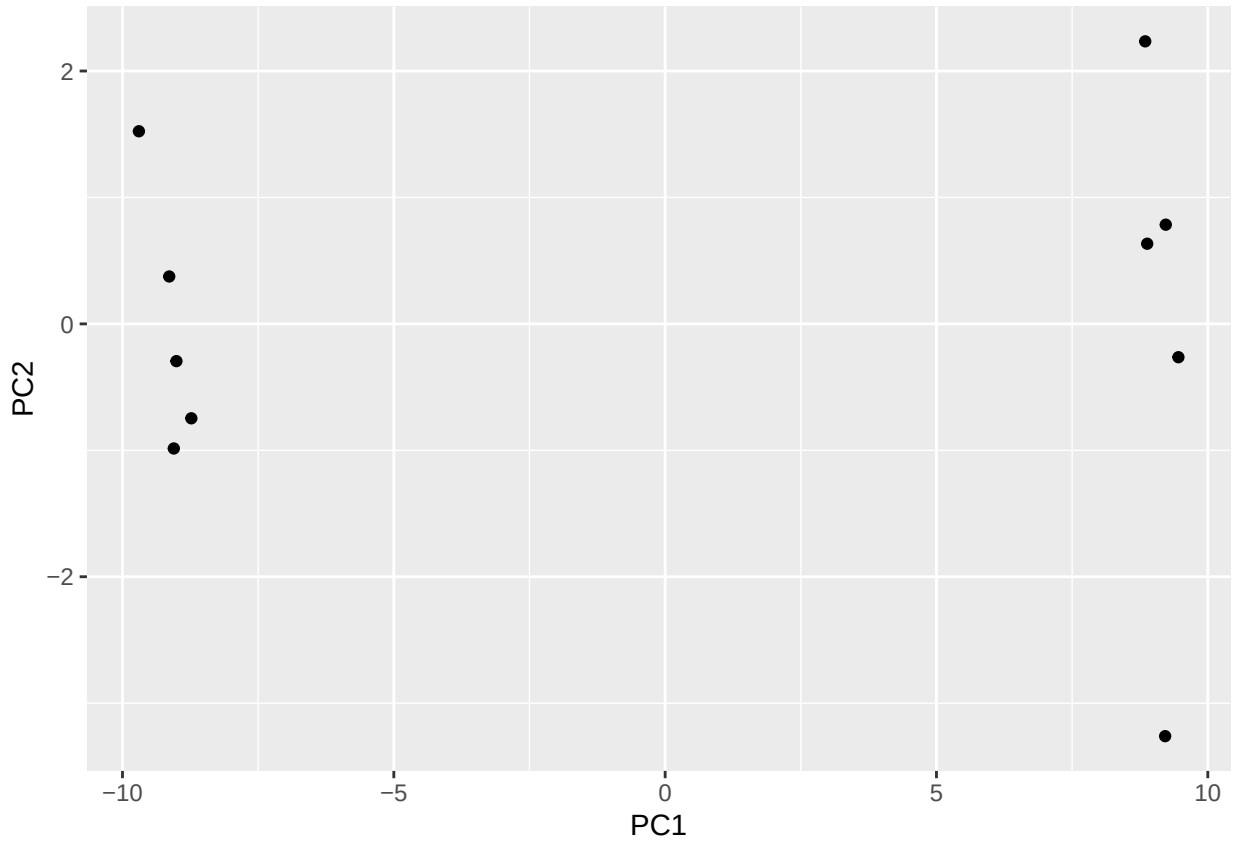



```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```



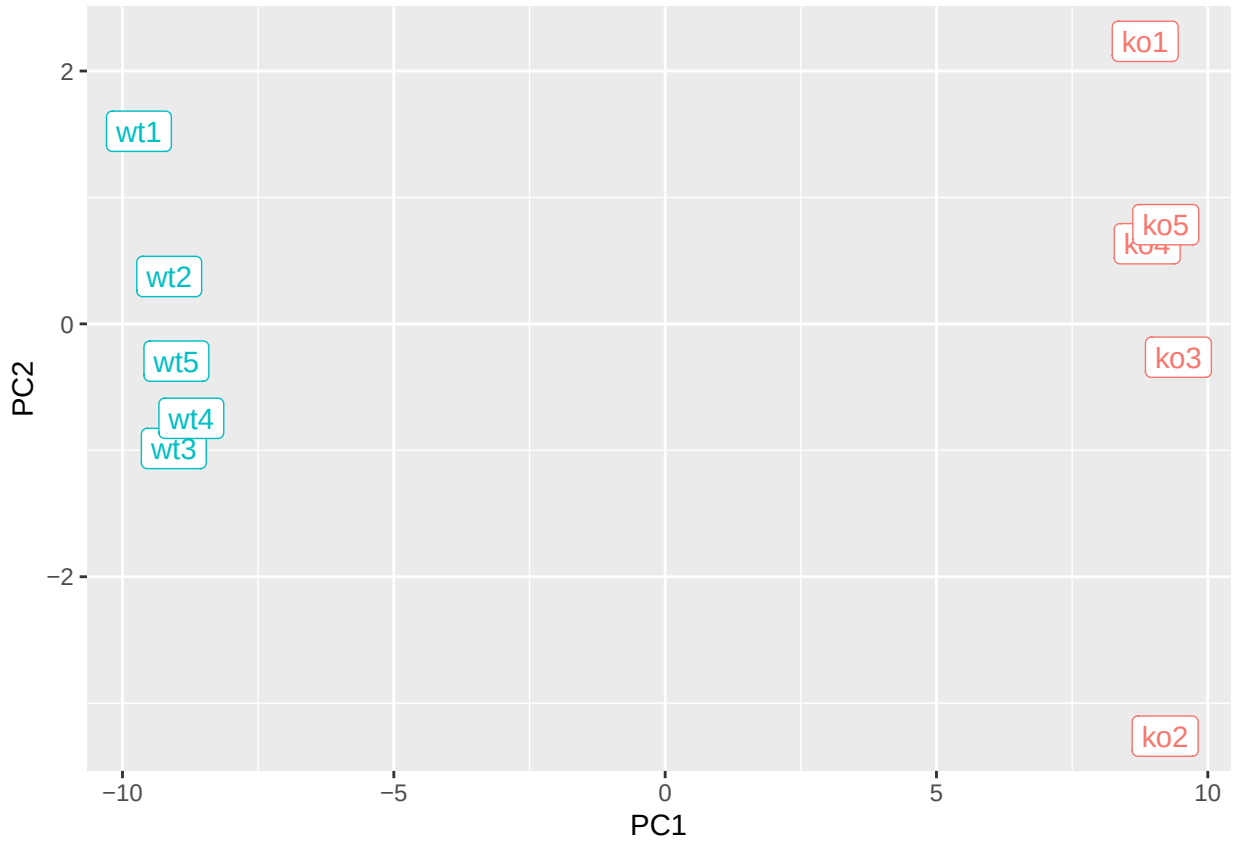
Make a data.frame for input for ggplot()

```
library(ggplot2)
df <- as.data.frame(pca$x)
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```

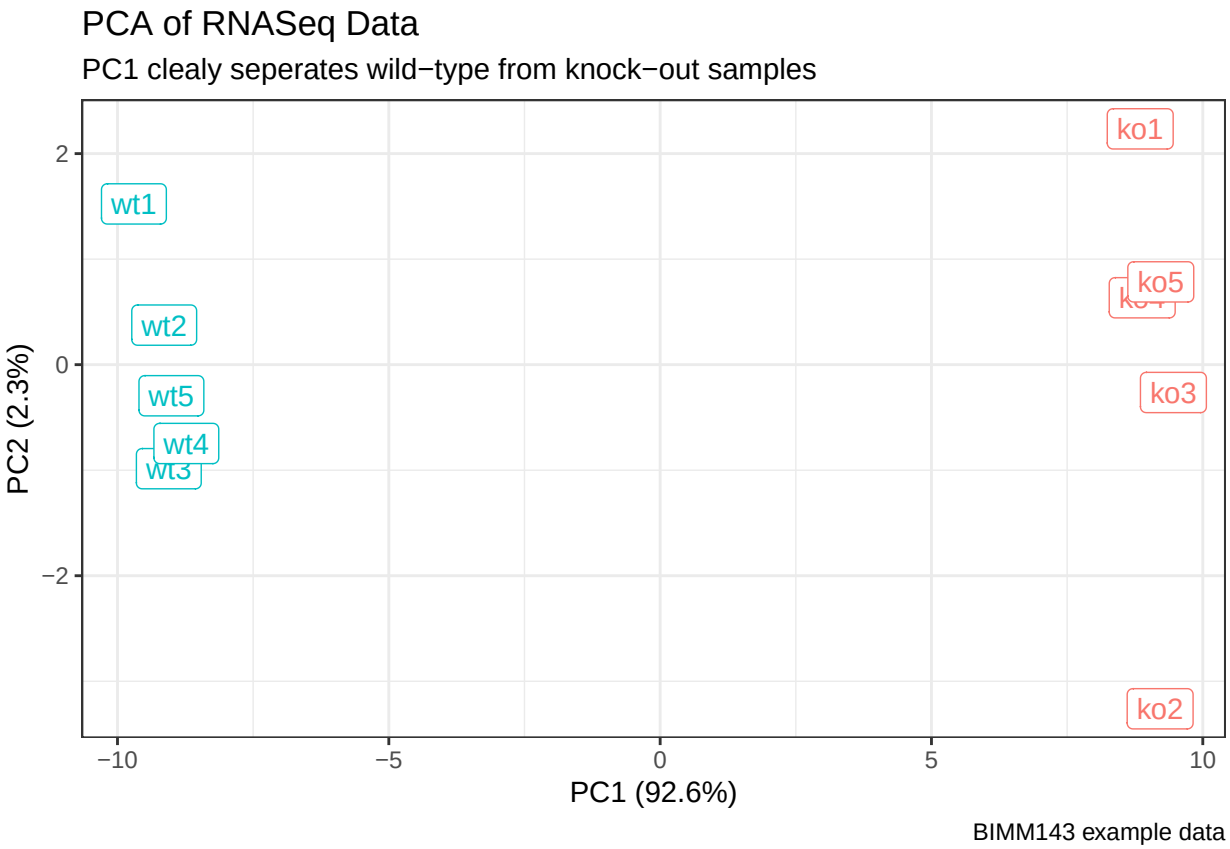


Add condition specific color or labels for wild-type or knock-out samples

```
# 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
```




```
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()
```


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
ko1
wt1
ko5
wt2
ko3
wt5
wt4
wt3
ko2
2
0
−2
PC2 (2.3%)
−10
−5
0
5
10
PC1 (92.6%)
BIMM143 example data