Hands on with Principal Component Analysis

Joshua Cheung

02/07/2022

1. PCA of UK food data

Data and import

we read the UK_foods.csv input file provided on the worksheet.

```
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?

```
## Complete the following code to find out how many rows and columns are in x?
dim(x)
```

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

so there are 17 rows and 5 columns in the data frame.

Checking your data

We preview the first 6 rows of the data frame we named x.

```
## 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 see that the row names are incorrectly set as the first column of our x data frame. So we attempt to set the rownames() to the first column and also remove the first column with the -1 column index.

```
# We use minus indexing here.
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
```

The result is much more desirable. Now let's check the dimensions of data frame x again.

```
dim(x)
```

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

Note that an alternative way to fix this problem is to read the file again and set the row.names argument of read.csv() to be the first column (specifically, set 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
```

This yields the same desired result.

> 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?

```
# Un-comment the last three lines to test if running this a second time will change data
# preview (it does).
# rownames(x) <- x[,1]
# x <- x[,-1]
# head(x)
```

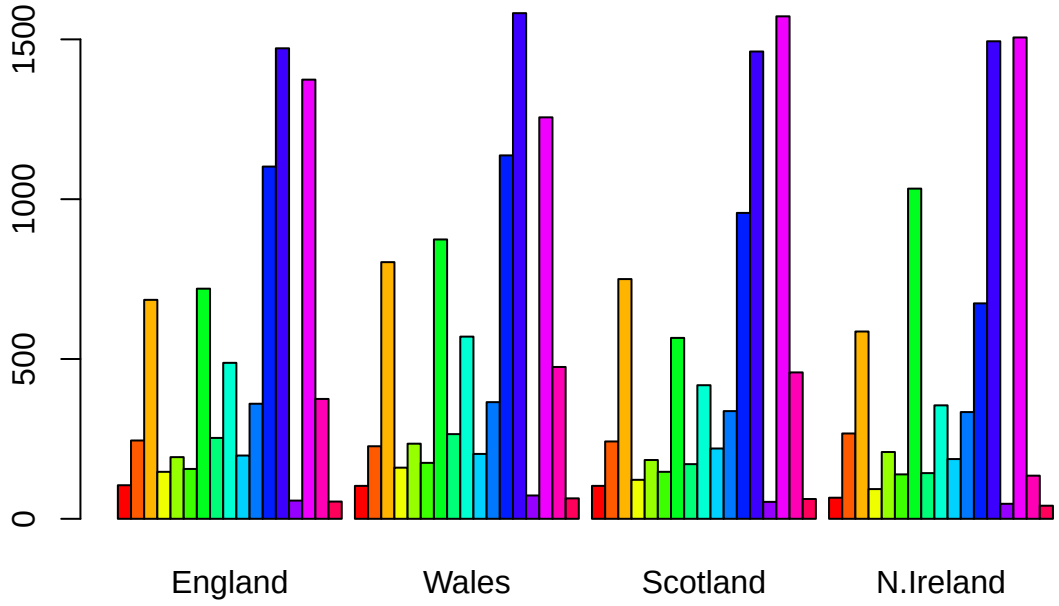
We see that running the first approach's code block (the one with x <- [,-1]) another time has set rownames() to the column corresponding to England and removed the previous iteration's row names. Thus to answer the above questions, the methods where we set the row.names argument of read.csv() to be the first column, is preferable. Running the code block corresponding to this method multiple times will not change the data preview generated. Thus set the row.names argument of read.csv() to be the first column appears to be more robust under certain circumstances.

```
# Un-comment the last three lines to test if running this multiple time will change data
# preview (it doesn't).
# x <- read.csv(url, row.names=1)
# x <- read.csv(url, row.names=1)
# head(x)
```

Spotting major differences and trends

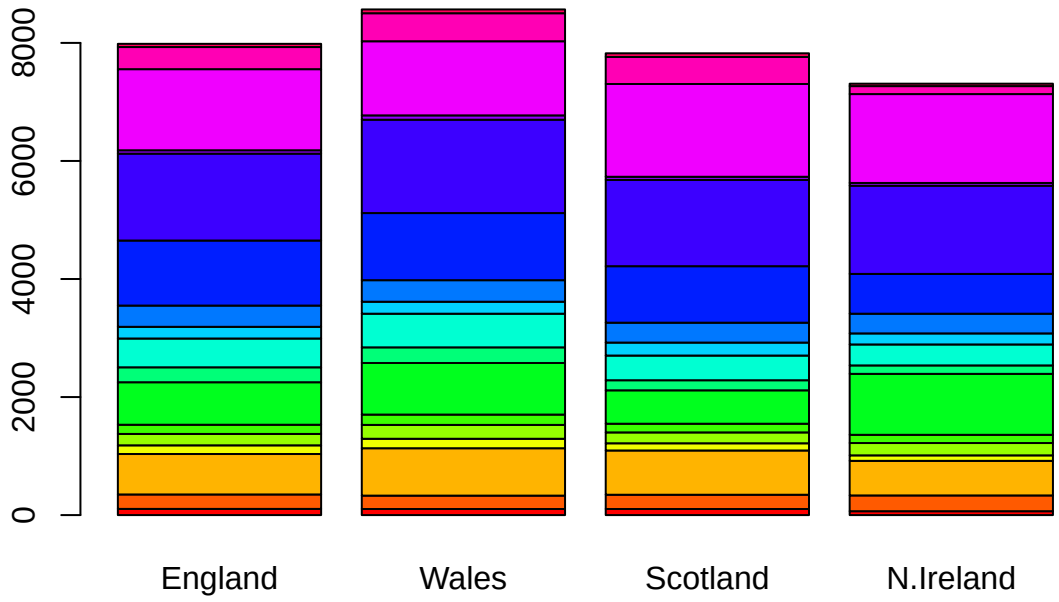
It is generally difficult to extract meaning in regard to major differences and trends in a the given array to numbers in the data frame. We generate a regular bar plot to see if this help with data visualization:

```
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?

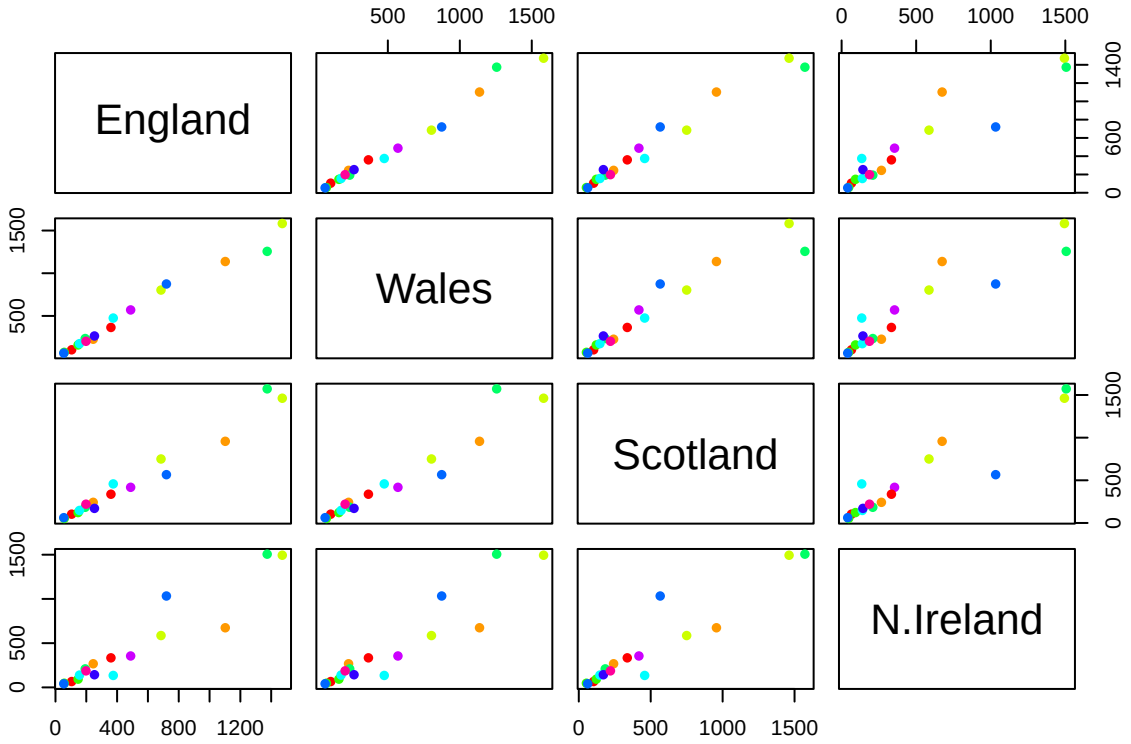
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Thus, we see that changing the optional argument beside to FALSE yields the plot desired. This is because changing the beside argument to false, sets the columns to be portrayed as stacked bars, while a TRUE argument would portray the columns as juxtaposed bars.

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)
```



For this plot, if a given point lies on the diagonal for a given plot, it means that the consumption in grams (per person per week) for a particular type of food stuff between the two countries on the plot is equal.

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

In general, Northern Ireland differs from the other countries in the UK as we see that pots where Northern Ireland is one of the axes have more points significantly off the diagonal relative to plots with the the other countries, where Northern Ireland is not one of the axes.

PCA to the rescue

We note that there are lots of function in R to choose form to perform PCA in R. For our purposes here we will use the main base R PCA function is called 'prcomp()'. Also, we will need to give the function the transpose of our input data.

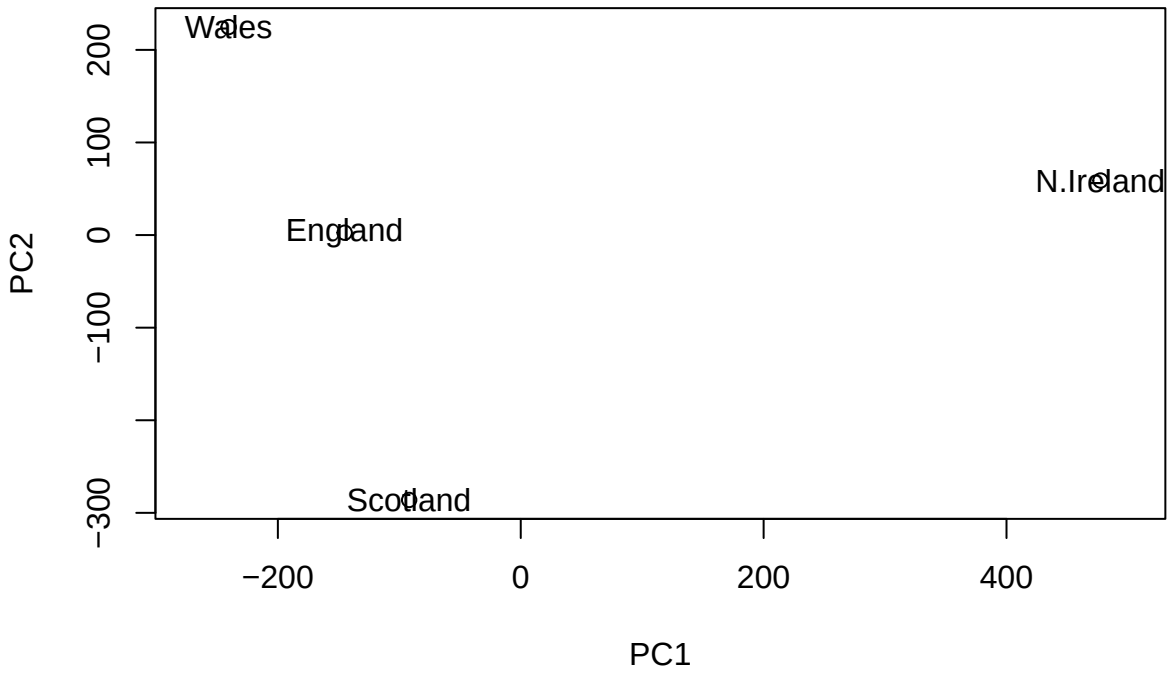
```
pca <- prcomp(t(x))
summary(pca)
```

```
## 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
```

To make our new PCA plot (or PCA score plot), we access 'pca$x'.

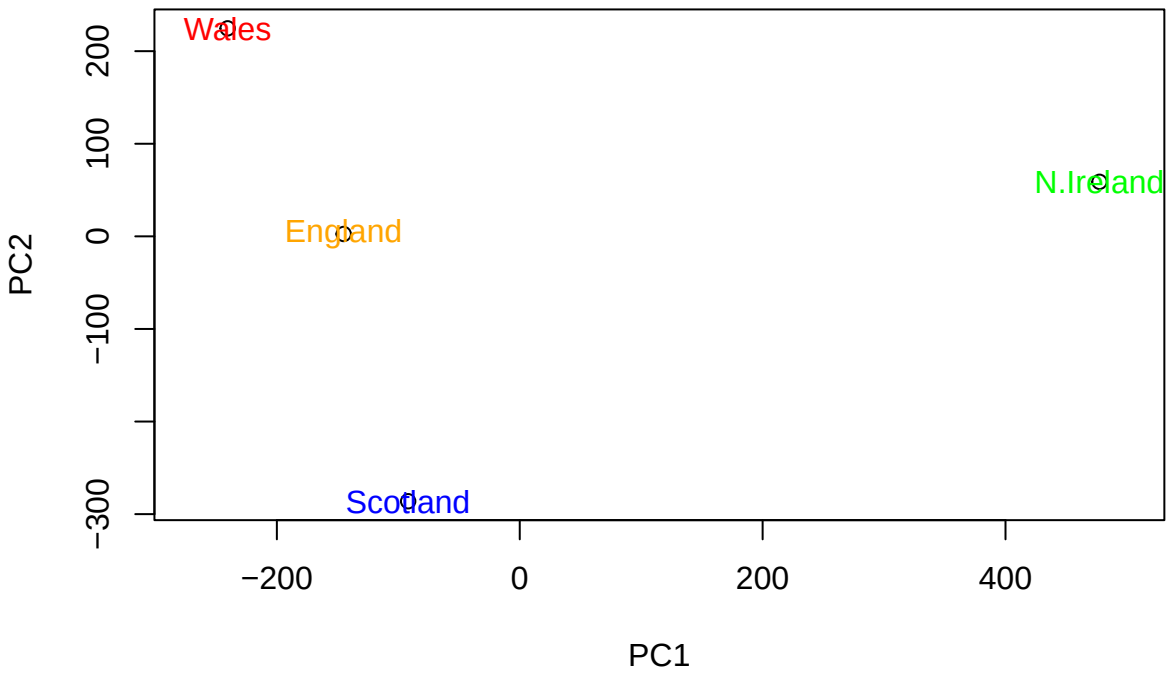
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot of 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.

```
# Plot of PC1 vs PC2 (with color now).
country_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=country_colors)
```



We now use the squard of pca$sdev (which stands for standard deviation) to compute 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
```

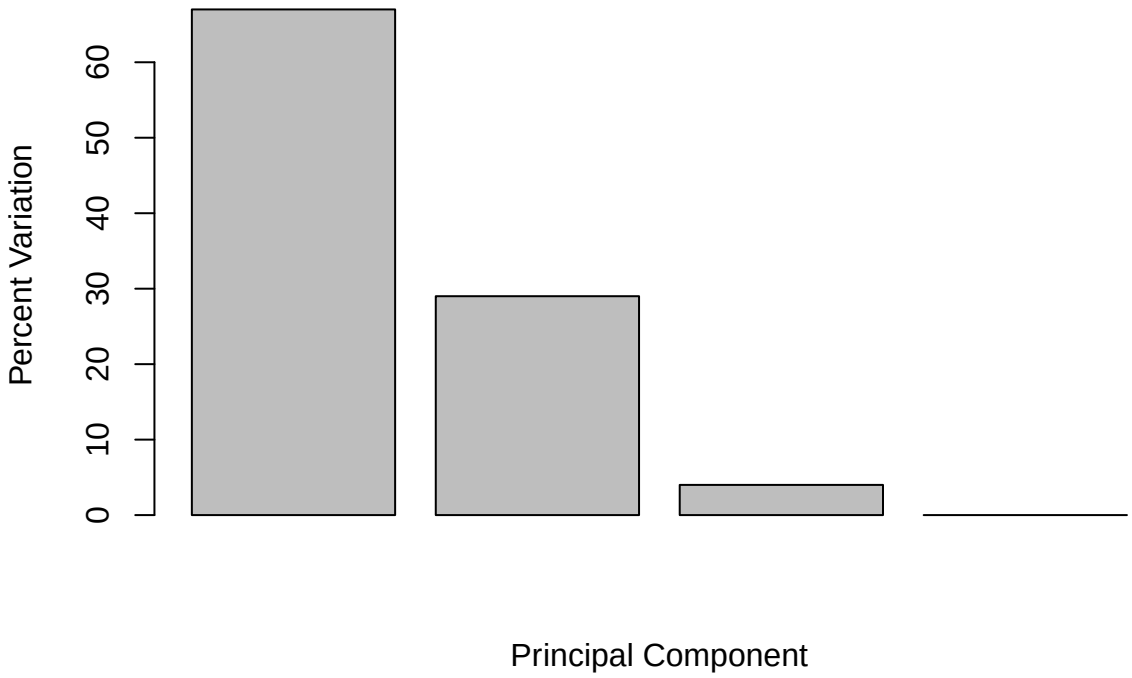
Also, we see we can get the same (albiet unrounded) results in the second row of the results from the following code.

```
# Observe the second row of the results.
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 this information in a plot of the variances (or eigenvalues) with respect to the principal component number (eigenvector number), which is given below.

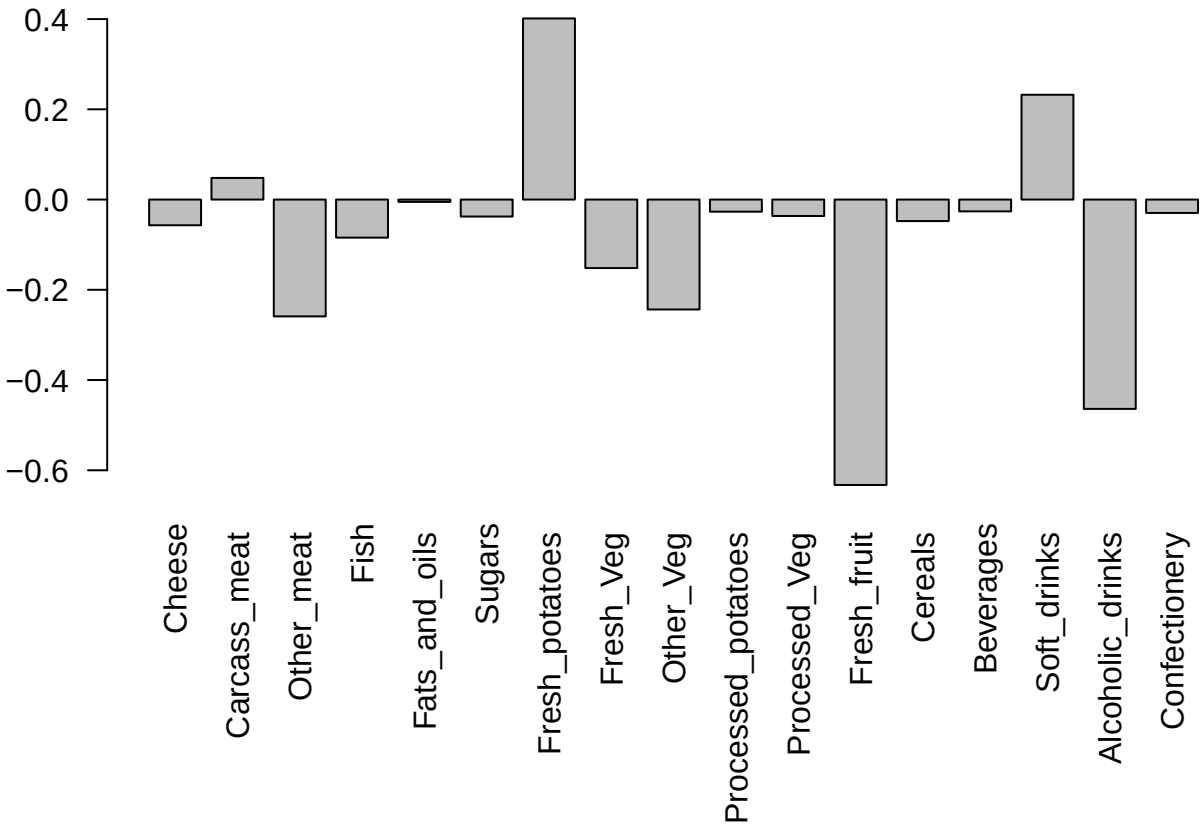
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



Digging deeper (variable loadings)

We now consider the influence of each of the original variables upon the principal components (usually know as loading scores). This information can be obtained from the prcomp() returned $rotation component.

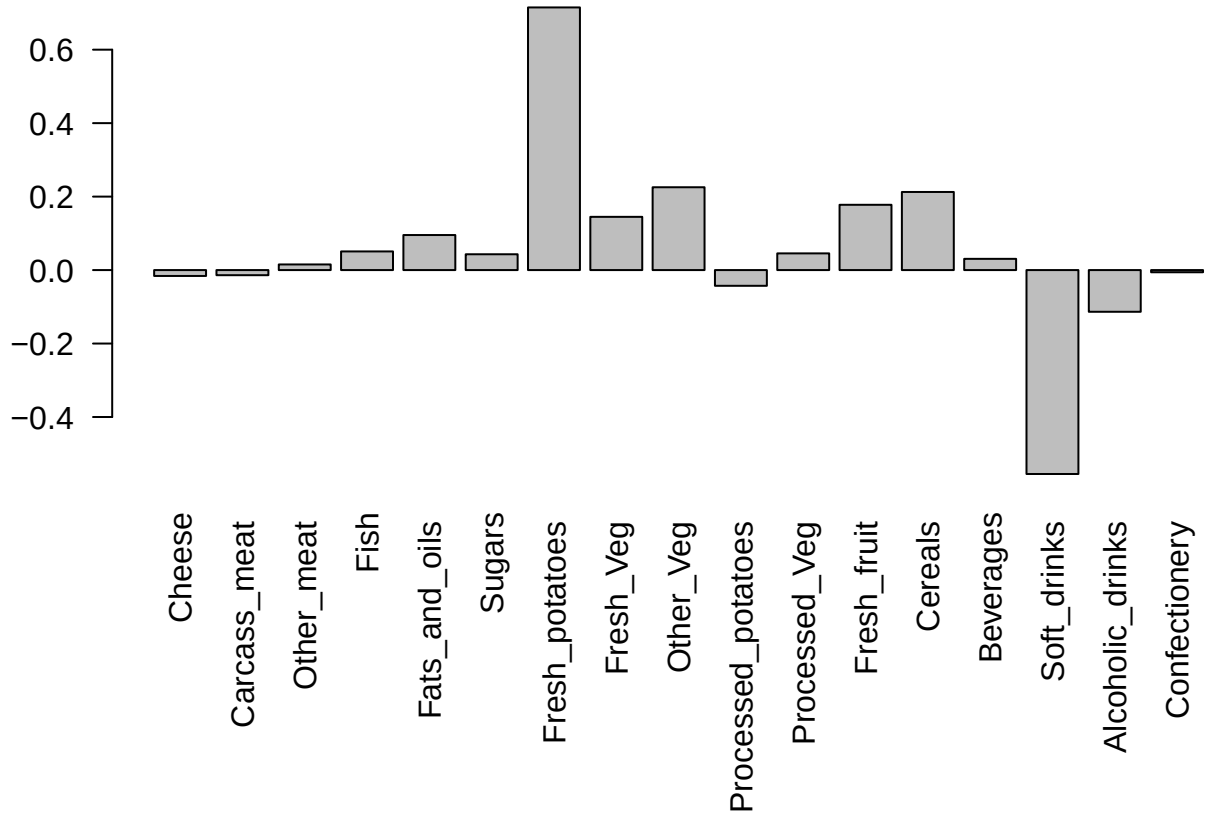
```
# Lets focus on PC1 as it accounts for > 90% of variance.
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



We see that foods (observations) with the largest positive loading scores that place Northern Ireland to the positive side of the plot include fresh potatoes and soft drinks. We also see that foods with the largest negative scores that place other countries on the negative side of the plot include fresh fruit and alcoholic drinks.

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

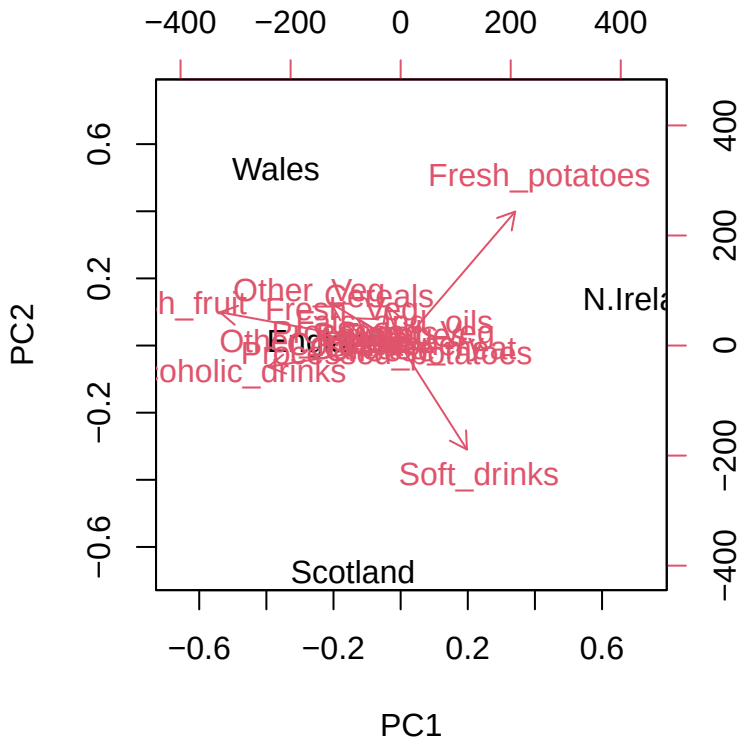
```
## We now focus on PC2.
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



Biplots

We note that another way to see this information together with the main PCA plot is in a biplot:

```
## Note R's inbuilt biplot() can be useful for small datasets.
biplot(pca)
```



We see that there is a central group of foods (shown with red arrows) around the middle of each principle component with four groups of food on the periphery that do not seem to be part of the group. Visual inspection of the original data provided to us, shows that for the variables fresh potatoes, alcoholic drinks, and fresh fruit, there the values for Ireland and noticeable different from the other three countries, England, Wales, and Scotland. Specifically for these variables, the values of England, Wales, Scotland, are roughly similar, while Northern Ireland' values are either significantly higher or lower.

2. PCA of RNA-seq data

We use a small RNA-seq count data set, and read it into a data frame called rna.data. Note the columns are individual samples (cells), and the rows are measurements taken for all the samples (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?

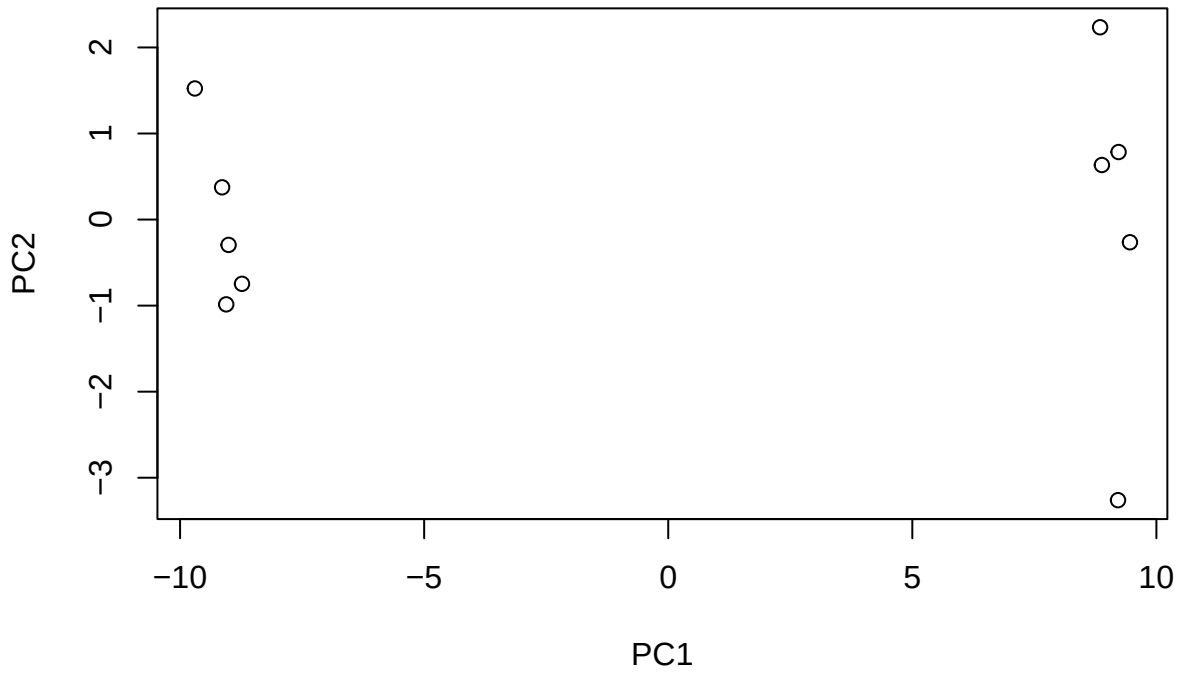
```
nrow(rna.data)
```

```
## [1] 100
```

So there are 100 genes in this data set.

We now move to performing a PCA and plotting the results. Don't forget to take the transpose of the data!

```
pca1 <- prcomp(t(rna.data), scale=TRUE)
# This gives a relatively un-polished plot of PC1 and PC2.
plot(pca1$x[,1], pca1$x[,2], xlab="PC1", ylab="PC2")
```



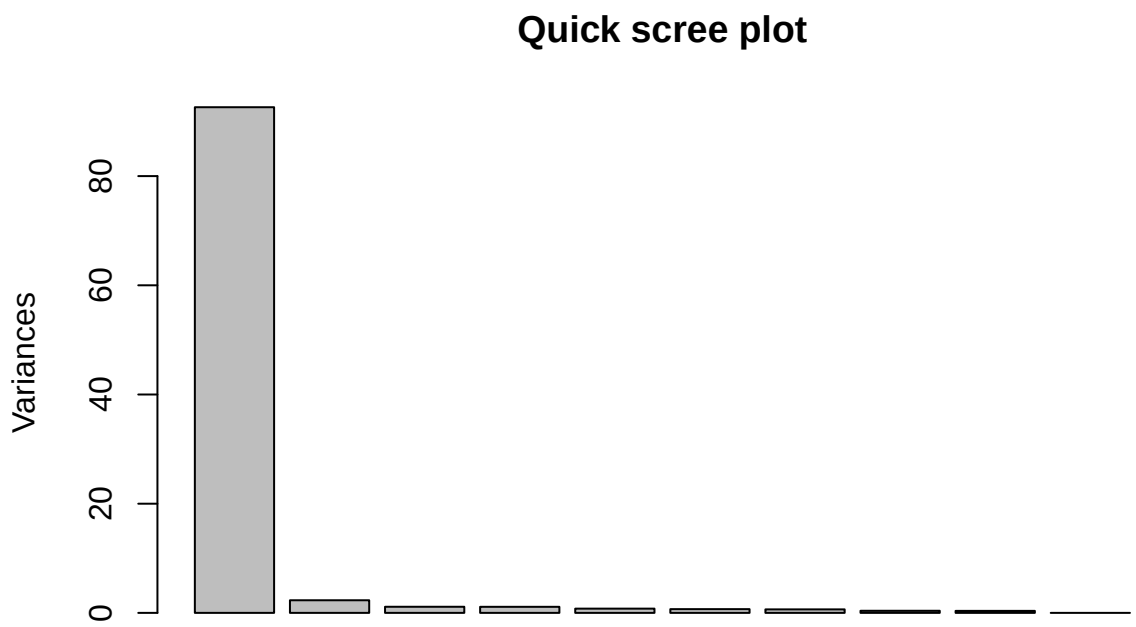
Now we examine a summary of how much variation in the original data each PC accounts for.

```
summary(pca1)
```

```
## 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
```

So we see that 92.6% of the proportion of variance is captured in the first principle component. We now construct a simple barplot summary of the proportion of variance for each PC using the plot() function.

```
plot(pca1, main="Quick scree plot")
```



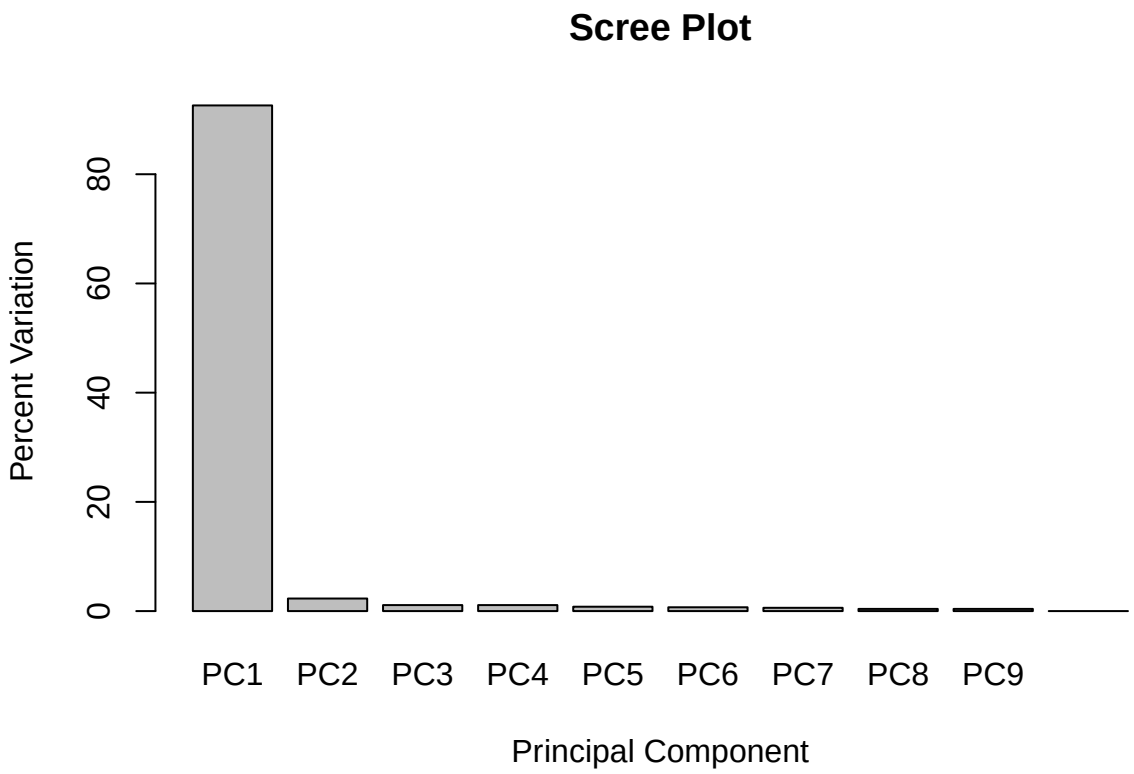
We now consider making the above scree plot ourselves and will further explore the object returned from prcomp(). Once again, we use pca1$sdev to compute how much variation in the original data each PC accounts for.

```
# The variance captured per PC is given by the following.
pca1.var <- pca1$sdev^2
# We use percent variance as this usually more informative to look at
pca1.var.per <- round(pca1.var/sum(pca1.var)*100, 1)
pca1.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 now use this information to generate our own scree-plot as follows.

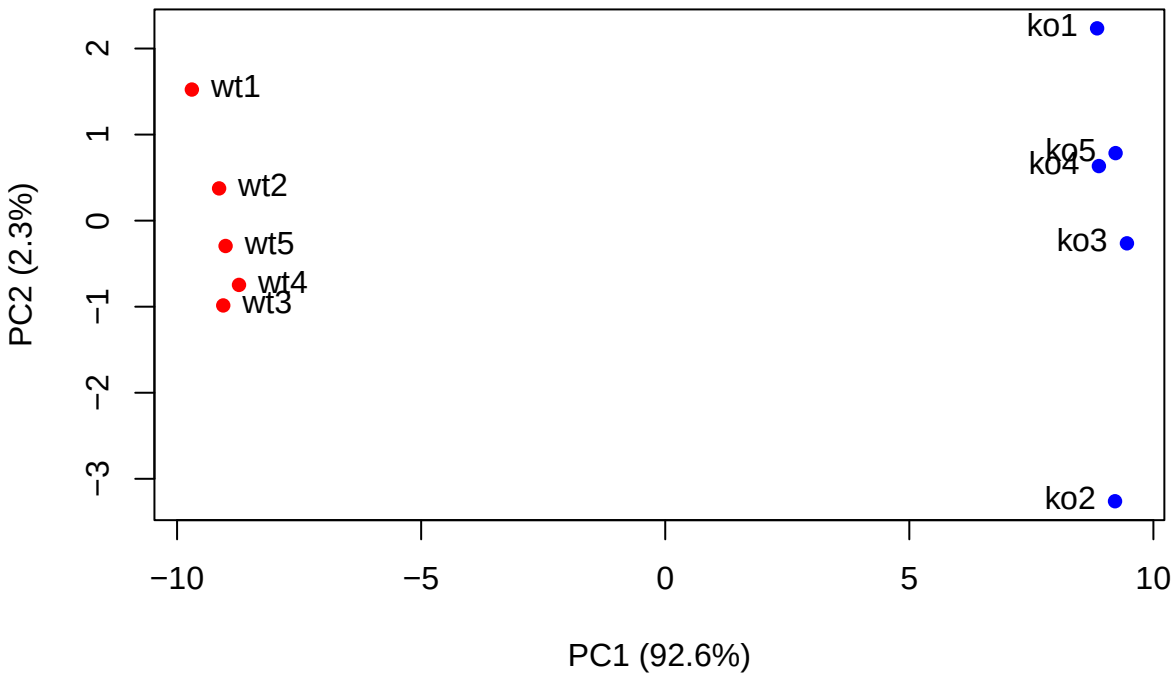
```
barplot(pca1.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



We now propose to make our PCA plot a bit more attractive and useful.

```
# We use a vector of colors for the wild-type (wt) and Knock-out (ko) samples.
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"
# The following are the details for the plot.
plot(pca1$x[,1], pca1$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca1.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca1.var.per[2], "%)"))

text(pca1$x[,1], pca1$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```

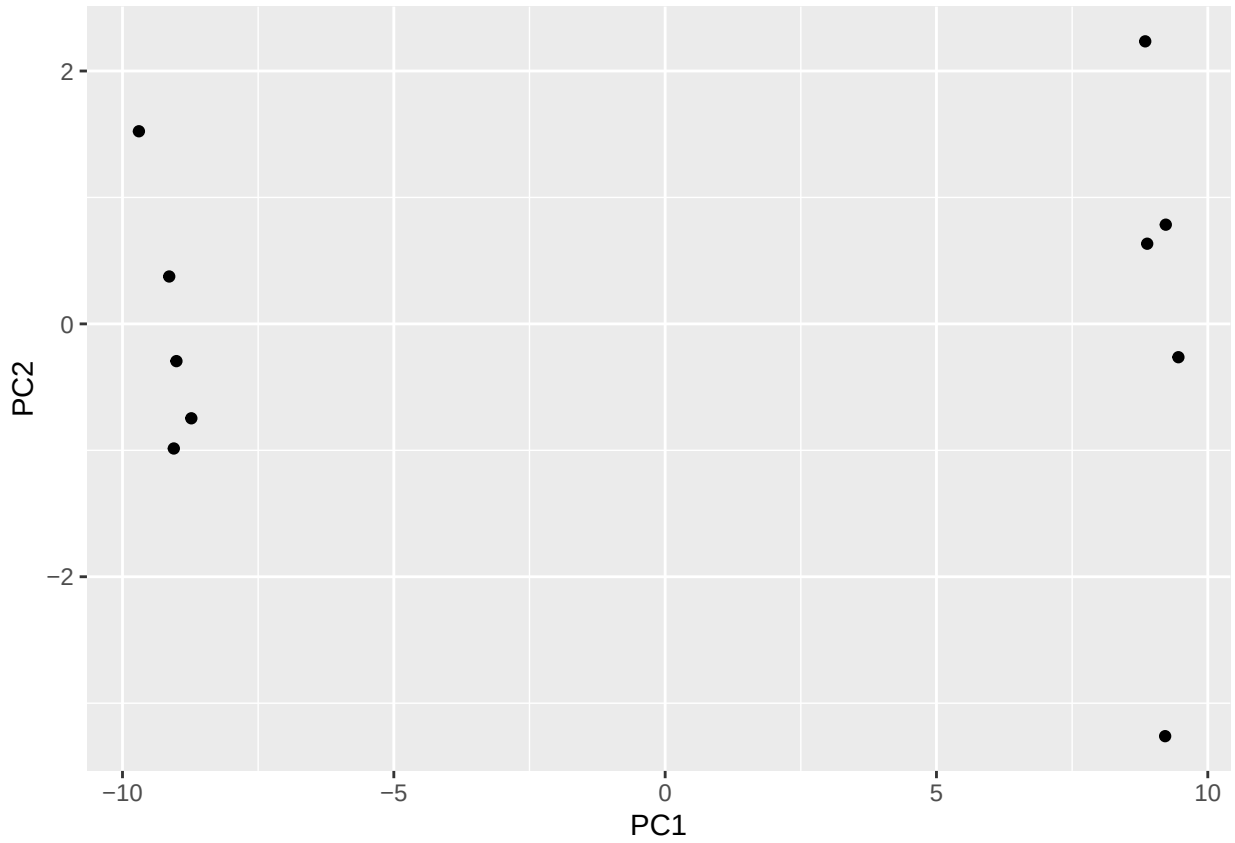


So all the knock-out gene experiment are located on the right, while all the wild-type gene experiments are on the right. This indicated the two groups are clearly distinct from each other.

Using ggplot

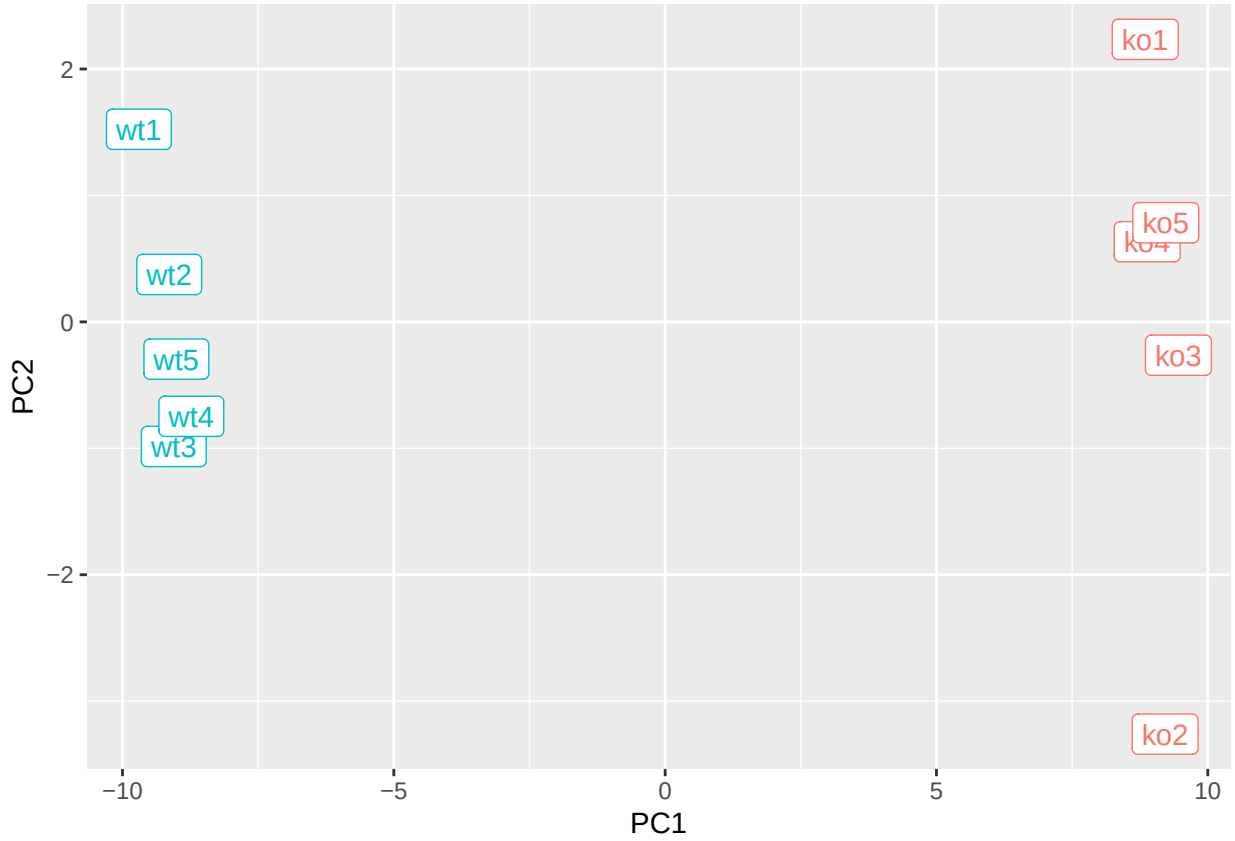
At this point we can uses the ggplot2 package, but will still need a data.frame as input for the main ggplot() function. This data frame will need our PCA results (pca1$x) and additional columns for the other aesthetic mappings.

```
# Recall ggplot2.
library(ggplot2)
# Define the data frame we need.
df <- as.data.frame(pca1$x)
# Our first basic plot.
ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



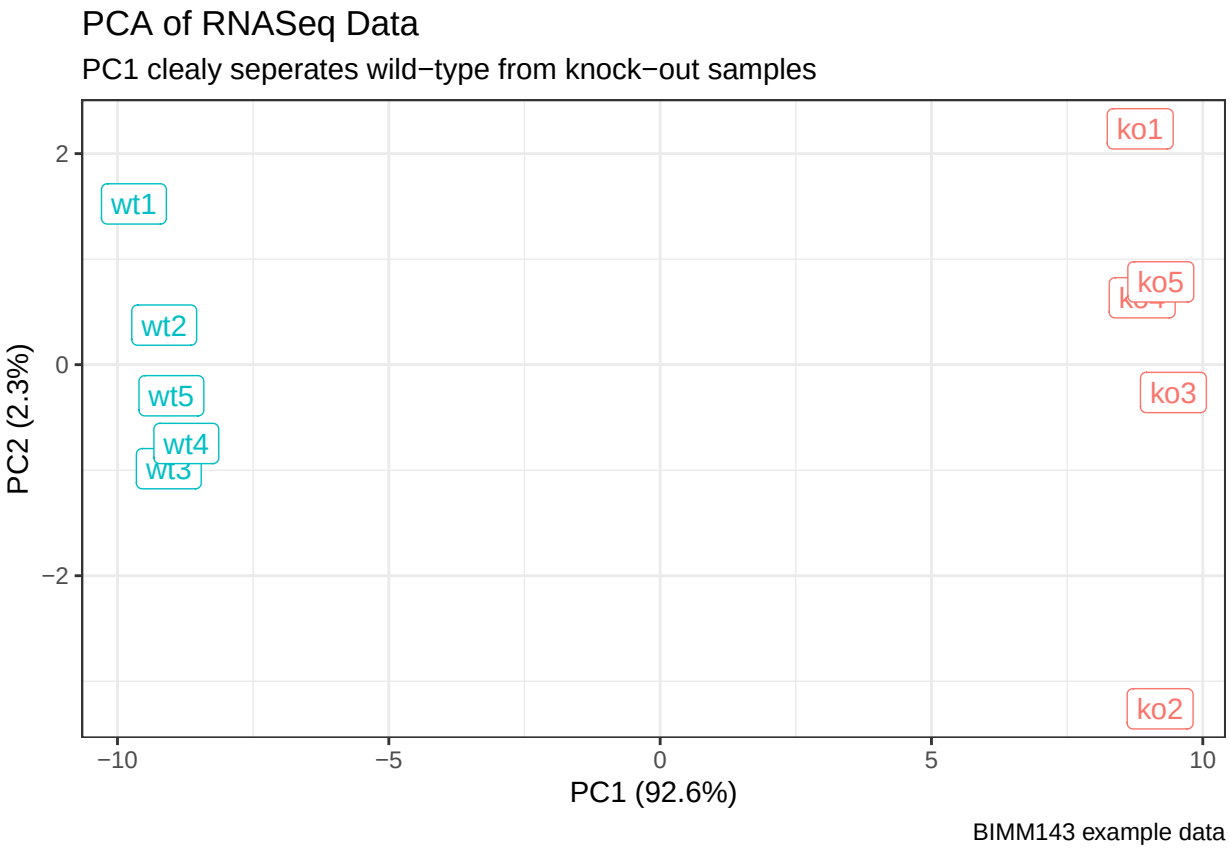
We now add a condition specific color and sample label aesthetics for wild-type and knock-out samples, and add this information to our data frame.

```
# We add a 'wt' and 'ko' condition column.
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)
# We now add this to the plot and define the plot as 'p'.
p <- ggplot(df) +
        aes(PC1, PC2, label=samples, col=condition) +
        geom_label(show.legend = FALSE)
p
```



We now add additional details such as a title, a subtitle, a caption, and additional details on each axes. We also update the theme to a more conservative "black and white" theme.

```
p + labs(title="PCA of RNASeq Data",
       subtitle = "PC1 clealy seperates wild-type from knock-out samples",
       x=paste0("PC1 (", pca1.var.per[1], "%)"),
       y=paste0("PC2 (", pca1.var.per[2], "%)"),
       caption="BIMM143 example data") +
     theme_bw()
```



Optional: Gene loadings

For this section, we find the top 10 measurements (or genes) that contribute the most to PC1 in either direction (+ or -).

```
loading_scores <- pca1$rotation[,1]
# We 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)
# We now 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"
```

Thus these genes, are possible candidates to focus on for further analysis.

3. Producing a PDF report

We have successfully compiled a summary PDF report of our work with answers to to it. Hopefully if the grader is reading this, that indicates we have successfully uploaded the PDF to Gradescope.

4. Sync to GitHub

The instructor for this course left explicit instruction to skip this section.