class07

Job Humberto Rocha Hernandez PID A59023124

Overview.

In this hands-on session, we will examine some real life multivariate data in order to explain, in simple terms what principal component analysis (PCA) achieves. This builds on our PCA introduction page. If you have not visited and fully explored that page yet then please do it now! Here we will perform a principal component analysis of several different data sets of increasing complexity and examine the results.

Getting organized

Before analyzing any dataset we need to get ourselves organized by following the steps below:

First open a new RStudio Project for today's work called class07 (File > New Project > New Directory > New Project, making sure to save this along with the rest of your course work for this class).

Then open a new Quarto Document (File > New File > Quarto Document) for saving your code and accompanying narrative notes. We will use this to generate our lab report later (see section 3 below).

In your new Quarto document be sure to keep the YAML header but remove any boilerplate example text and code (i.e. delete from line 6 or so onward) so you have a clean document to work in and add your content to.

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. This would be equivalent to our samples and genes respectively from the lecture video example (and indeed the second main example further below).

Data import

First we will read the provided UK_foods.csv input file (note we can read this directly from the following tinyurl short link: "https://tinyurl.com/UK-foods" (which simply points to a CSV file on our class website).

```
#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? *17 rows and 5 columns. I can use function 'dim()'*

```
dim(x)
```

```
[1] 17  5
```

Checking your data

It is always a good idea to examine your imported data to make sure it meets your expectations. At this stage we want to make sure that no odd things have happened during the importing phase that will come back to haunt us later.

For this task we can 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).

Side-Note: Never leave a View() function call uncommented in your Quarto document as this is intended for interactive use and will slow down or even stop the Rendering process when you go to generate HTML, PDF, MD etc. format reports.

```
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
```

Hmm, it looks like the row-names here were not set properly as we were expecting 4 columns (one for each of the 4 countries of the UK - not 5 as reported from the dim() function).

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 and sometimes what we want - but not in this case. 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):

```
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 looks much better, now lets check the 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), see below:

```
x <- read.csv(url, row.names=1)
head(x)
```

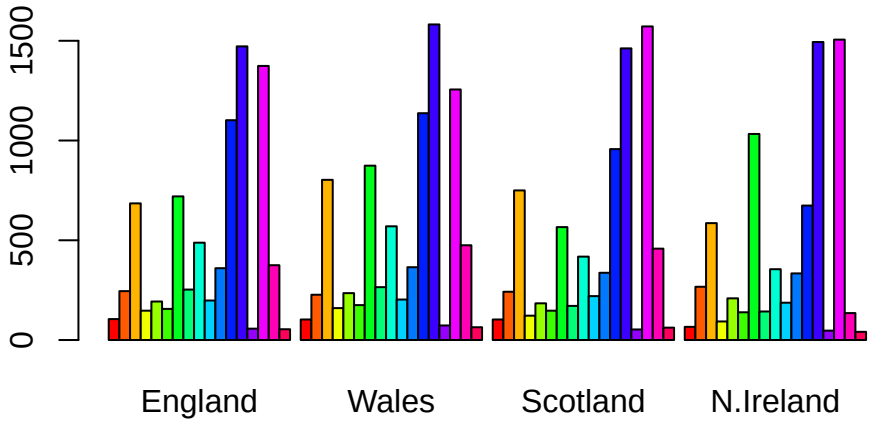
```
                England Wales Scotland N.Ireland
Cheese              105   103      103        66
Carcass_meat        245   227      242       267
Other_meat          685   803      750       586
Fish                147   160      122        93
Fats_and_oils       193   235      184       209
Sugars              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 since it's cleaner and less lines of code. Additionally if I run the code many times the second approach gives the same result, while the first code looses the first column everytime is re-run (data overwriting).***

Spotting major differences and trends

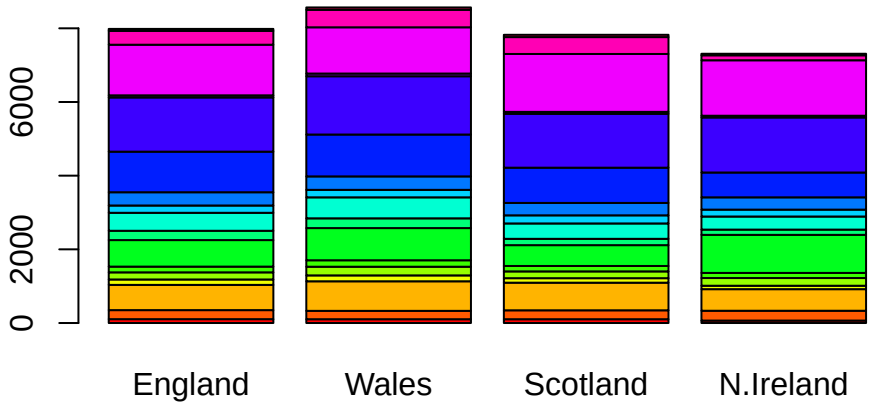
A cursory glance over the numbers in this table does not reveal much of anything. Indeed in general it is difficult to extract meaning in regard to major differences and trends from any given array of numbers. Generating regular bar-plots and various pairwise plots does not help too much either:

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



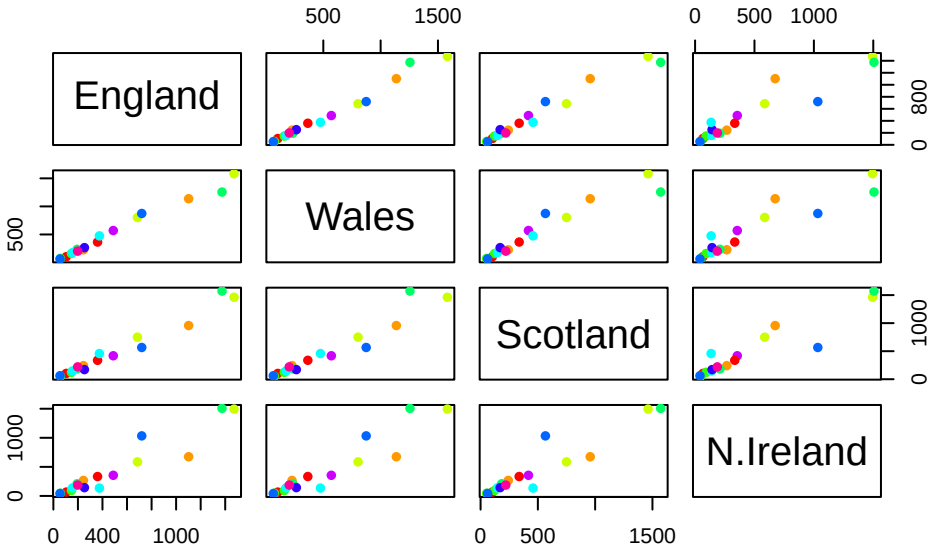
Q3: Changing what optional argument in the above barplot() function results in the following plot? ***Argument 'beside' set to F.***

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



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? ***Each plot is a pair-wise comparison for each contry dataset. Dots that are in the diagonal means that have similar values in both datasets. The more dots in the diagonal means that the two datasets have a better correlation.***

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



Even relatively small datasets can prove challenging to interpret 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? ***It looks like it's the most dissimilar (different) compared to the rest of them.***

PCA to the rescue

We need some way of making sense of the above data. Are there any trends present which are not obvious from glancing at the array of numbers?

Traditionally, we would use a series of pairwise plots (i.e. bivariate scatter plots) and analyse these to try and determine any relationships between variables, however the number of such plots required for such a task is clearly too large even for this small dataset. Therefore, for large data sets, this is not feasible or fun.

PCA generalizes this idea and allows us to perform such an analysis simultaneously, for many variables. In our example here, we have 17 dimensional data for 4 countries. We can thus 'imagine' plotting the 4 coordinates representing the 4 countries in 17 dimensional space. If

there is any correlation between the observations (the countries), this will be observed in the 17 dimensional space by the correlated points being clustered close together, though of course since we cannot visualize such a space, we are not able to see such clustering directly (see the lecture slides for a clear description and example of this).

To perform PCA in R there are actually lots of functions to chose from and many packages offer slick PCA implementations and useful graphing approaches. However here we will stick to the base R prcomp() function.

As we noted in the lecture portion of class, 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.

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

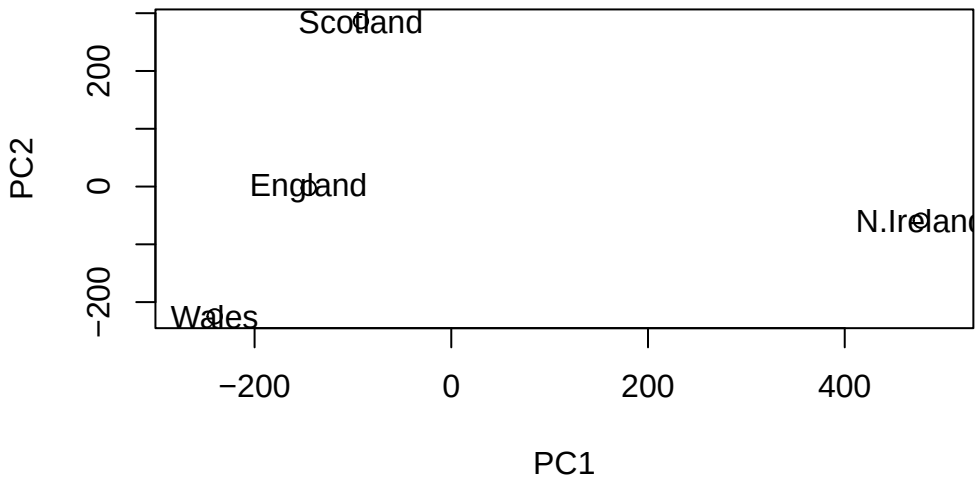
```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

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.

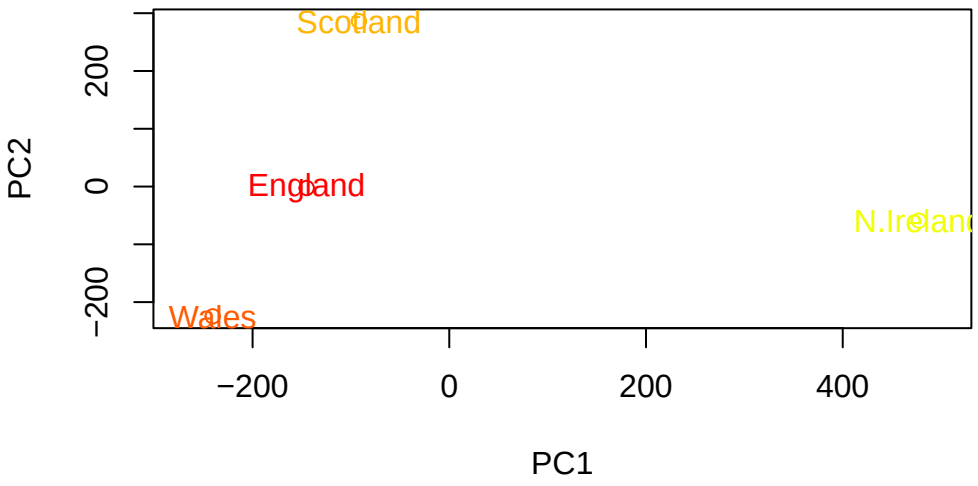
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

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



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

In our food example here, the four 17 dimensional coordinates are projected down onto the two principal components to obtain the graph above. As part of the PCA method, we automatically obtain information about the contributions of each principal component to the total variance of the coordinates. This is typically contained in the Eigenvectors returned from such calculations. In the prcomp() function we can use the summary() command above or examine the returned pca$sdev (see below).

In this case approximately 67% of the variance in the data is accounted for by the first principal component, and approximately 97% is accounted for in total by the first two principal components. In this case, we have therefore accounted for the vast majority of the variation in the data using only a two dimensional plot - a dramatic reduction in dimensionality from seventeen dimensions to two.

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. Ask Barry about this if you are unsure what we mean here.

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.

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

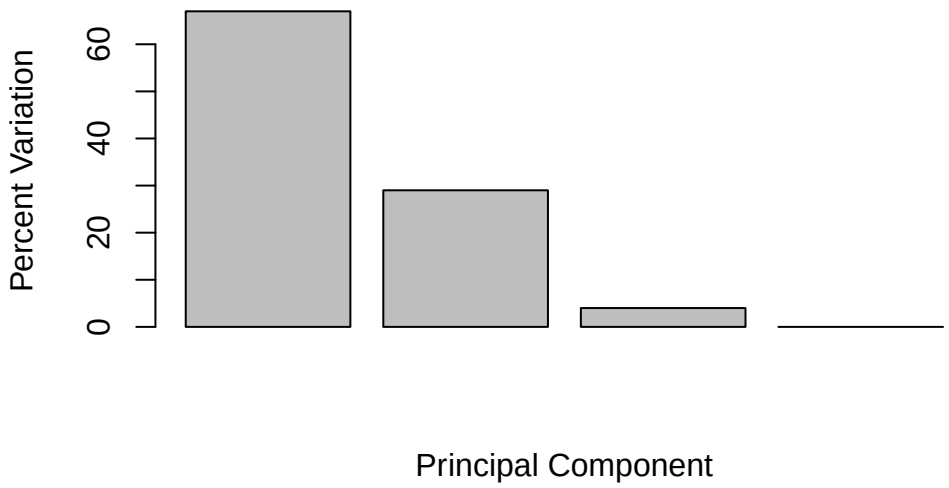
```
[1] 67 29  4  0
```

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

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 3.175833e-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.

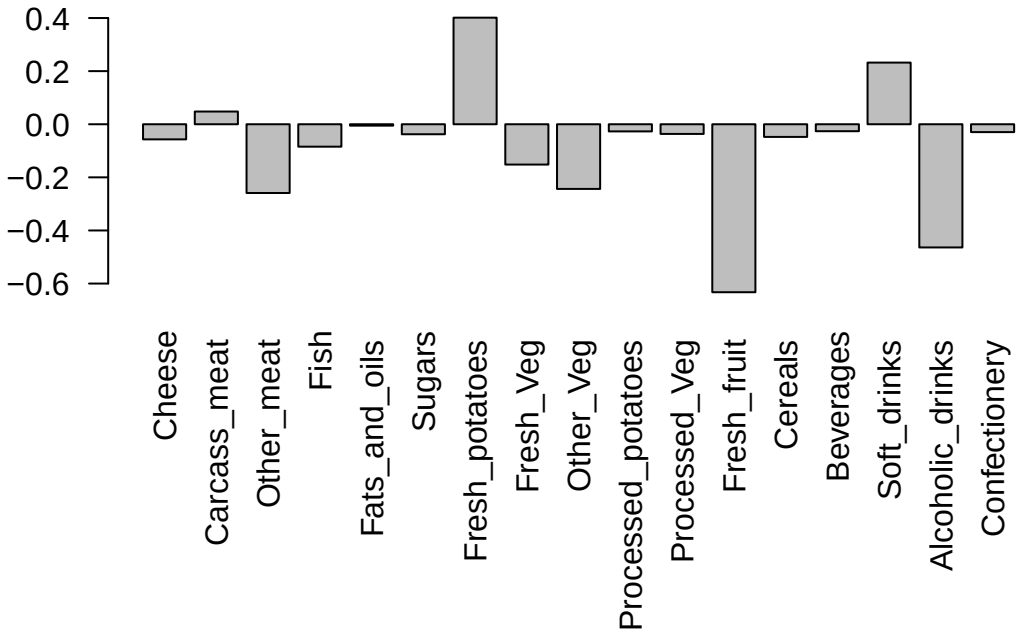
```r
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



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:

```
## 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 )
```

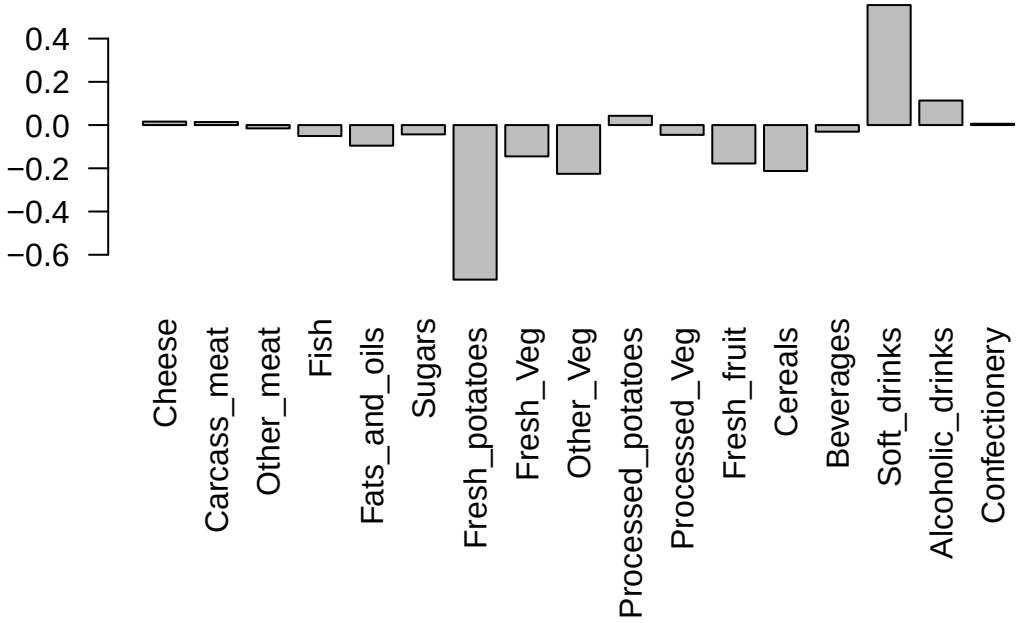


Here we see observations (foods) with the largest positive loading scores that effectively "push" N. Ireland to right positive side of the plot (including Fresh_potatoes and Soft_drinks).

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 maninly tell us about? ***Food groups are Fresh_potatoes and Soft_drinks. PC2 tells us about the main food consumption differences (Fresh_potates and Soft_drinks) between Wales and N. Ireland***

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

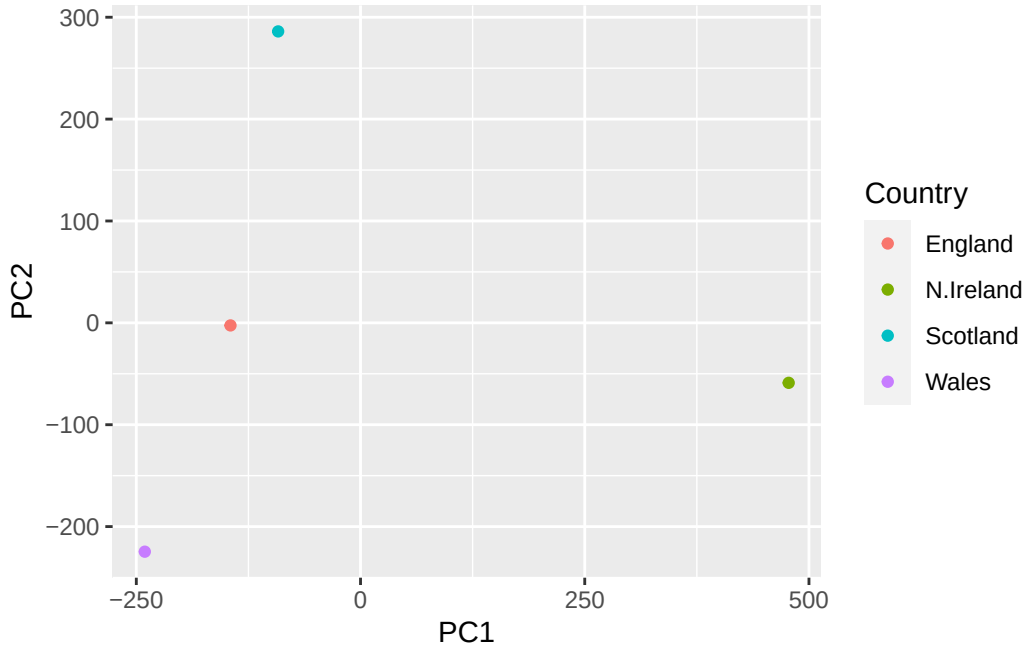


Using ggplot for these figures

We could use the ggplot2 package to make somewhat better figures than all of the above "base" R plots() and barplots(). Recall that ggplot works with data.frames and unfortunately most of the output of these older base R functions like prcomp() are lists of vectors and matrices.

So first we will need to take whatever it is we want to plot and convert it to a data.frame with the as.data.frame() function. Then to make our plotting life easier we will also add the food labels as a column (called "Food") to this data frame with the rownames_to_column() function from the tibble package (you might need to install this):

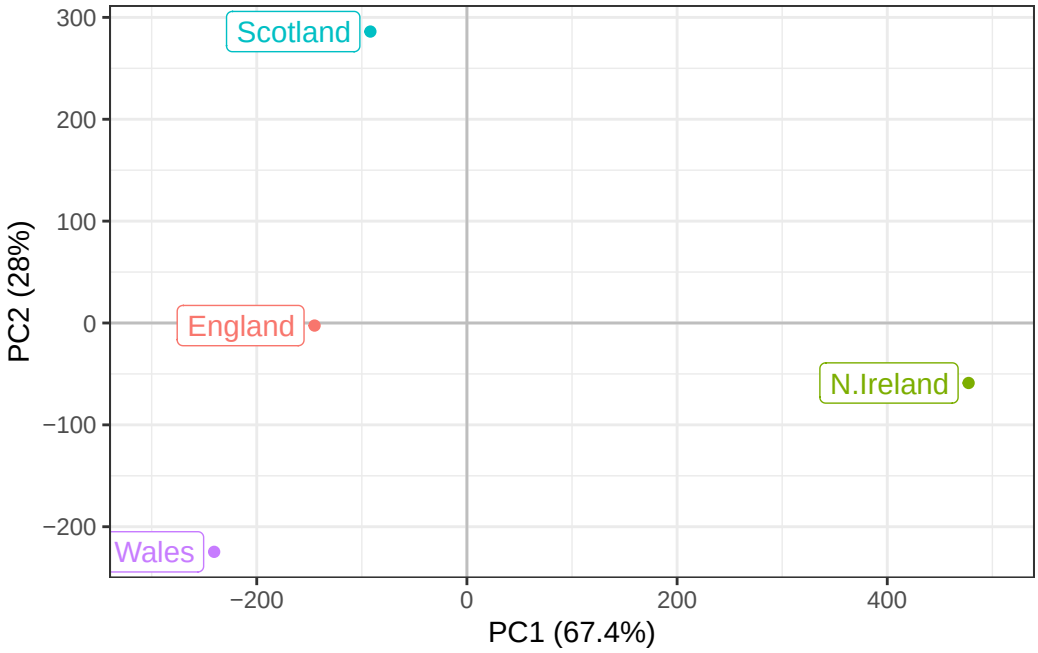
```
library(ggplot2)

df <- as.data.frame(pca$x)
df_lab <- tibble::rownames_to_column(df, "Country")

# Our first basic plot
ggplot(df_lab) +
  aes(PC1, PC2, col=Country) +
  geom_point()
```



And then we can get carried away and make this look much nicer:

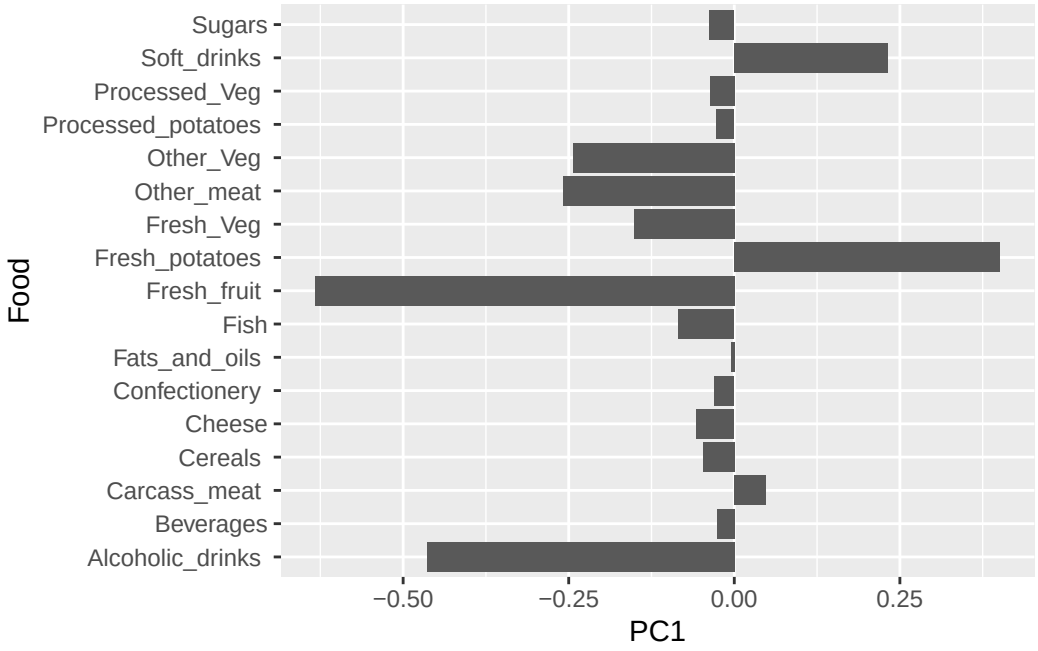
```
ggplot(df_lab) +
  aes(PC1, PC2, col=Country, label=Country) +
  geom_hline(yintercept = 0, col="gray") +
  geom_vline(xintercept = 0, col="gray") +
  geom_point(show.legend = FALSE) +
  geom_label(hjust=1, nudge_x = -10, show.legend = FALSE) +
  expand_limits(x = c(-300,500)) +
  xlab("PC1 (67.4%)") +
  ylab("PC2 (28%)") +
```

```
  theme_bw()
```



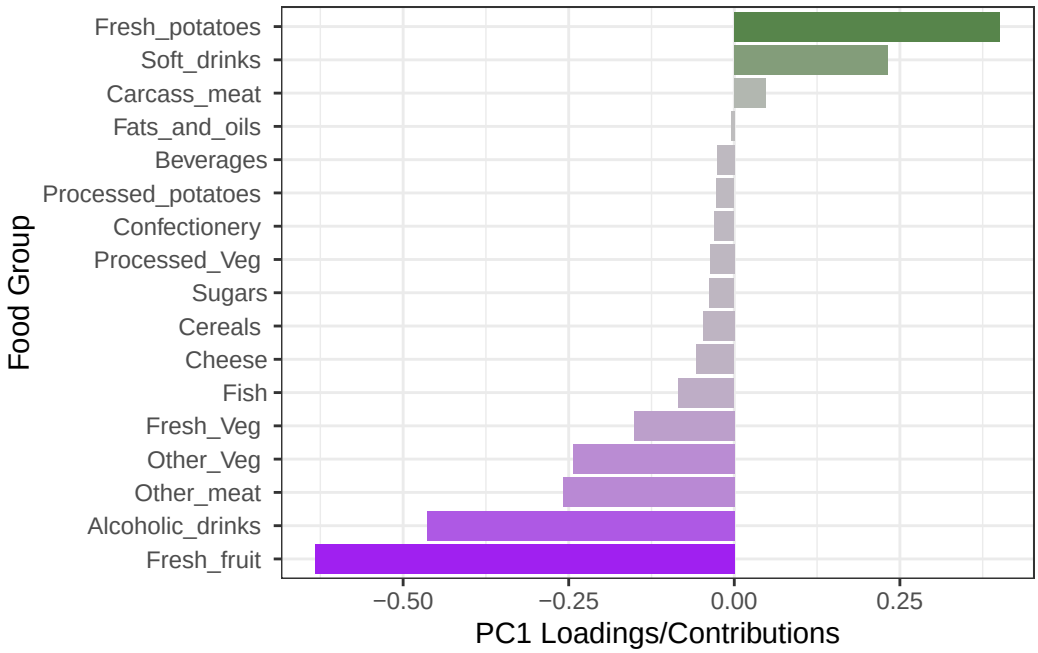
Let's do the same for our loadings/PC contributions figures. This data is stored in the pca$rotation object that we convert to a data frame, add the useful row names as a new column and then plot and customize with additional ggplot layers. Which do you prefer, base graphics or ggplot?

```
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

ggplot(ld_lab) +
  aes(PC1, Food) +
  geom_col()
```



We can now add some additional features to the plot, such as reordering the y axis by the PC1 loadings and selecting a rather ugly color scale (to match our country colors) and our prefered theme layer.

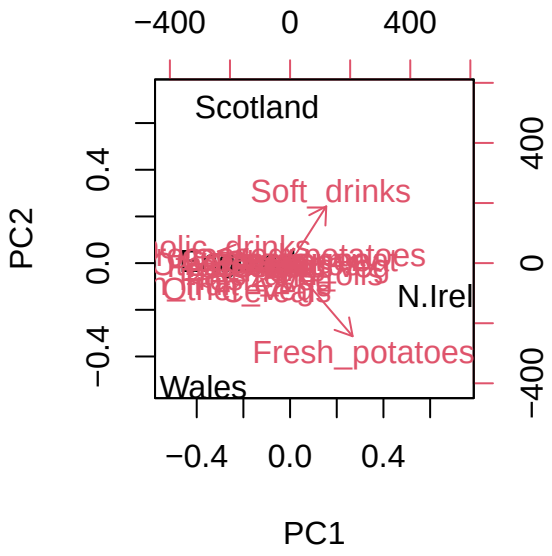
```
ggplot(ld_lab) +
  aes(PC1, reorder(Food, PC1), bg=PC1) +
  geom_col() +
  xlab("PC1 Loadings/Contributions") +
  ylab("Food Group") +
  scale_fill_gradient2(low="purple", mid="gray", high="darkgreen", guide=NULL) +
  theme_bw()
```



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)
```



Observe here that there is a central group of foods (red arrows) around the middle of each principal component, with four on the periphery that do not seem to be part of the group. Recall the 2D score plot (Figure above), on which England, Wales and Scotland were clustered together, whilst Northern Ireland was the country that was away from the cluster. Perhaps there is some association to be made between the four variables that are away from the cluster in the main PCA plot and the country that is located away from the rest of the countries i.e. Northern Ireland. A look at the original data in Table 1 reveals that for the three variables, Fresh potatoes, Alcoholic drinks and Fresh fruit, there is a noticeable difference between the values for England, Wales and Scotland, which are roughly similar, and Northern Ireland, which is usually significantly higher or lower.

Note: 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

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!

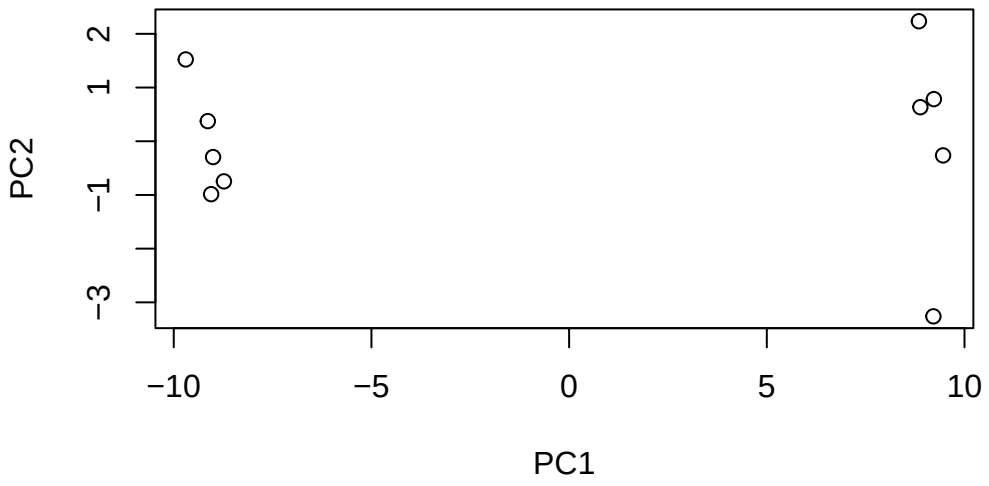
Q10: How many genes and samples are in this data set? *100 genes and 10 samples.*

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

```
[1] 100  10
```

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:

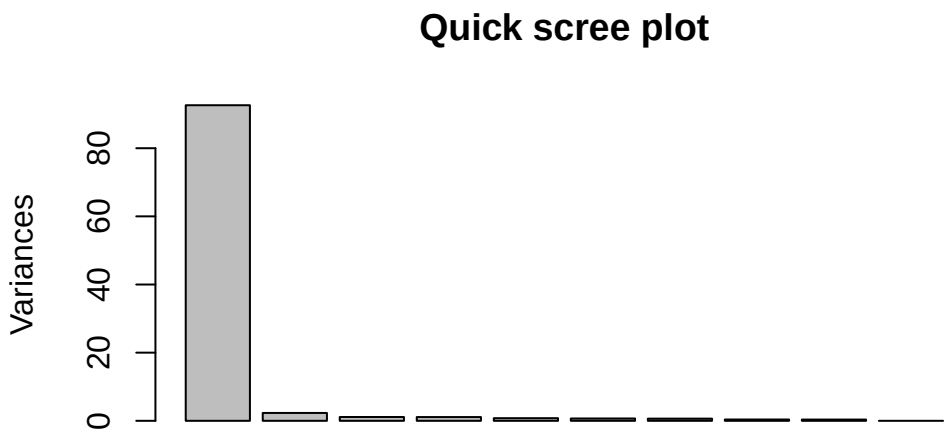
```
summary(pca)
```

```
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 3.457e-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")
```



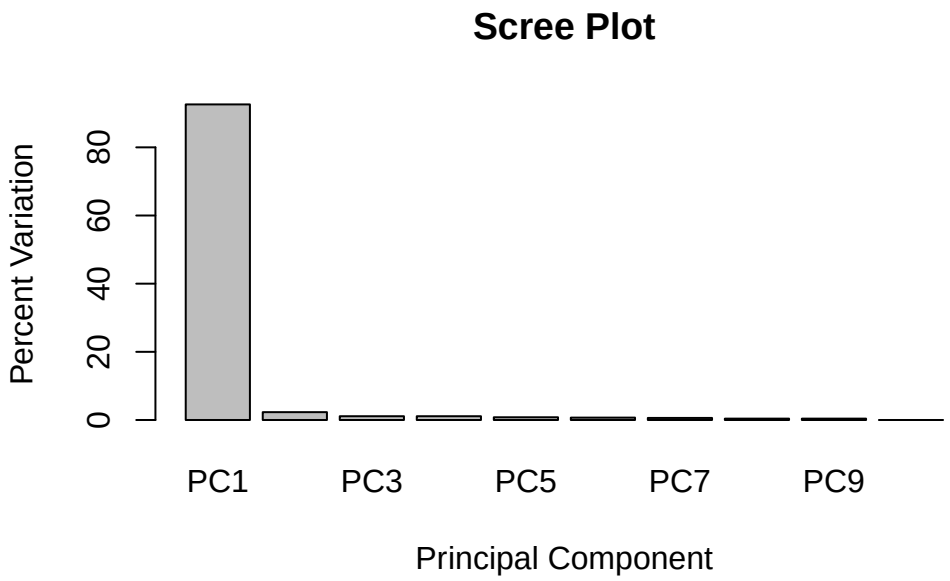
Let's make the above scree plot ourselves and in so doing explore the object returned from prcomp() a little further. We can use the square of pca$sdev, which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for:

```
## 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

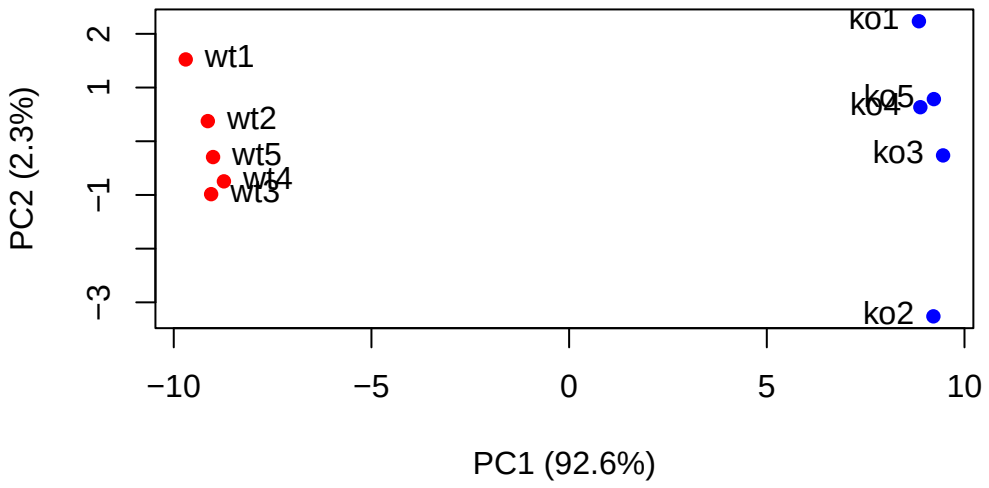
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



Again we can see from this plot that PC1 is were all the action is.

Now lets make our main PCA plot a bit more attractive and useful...

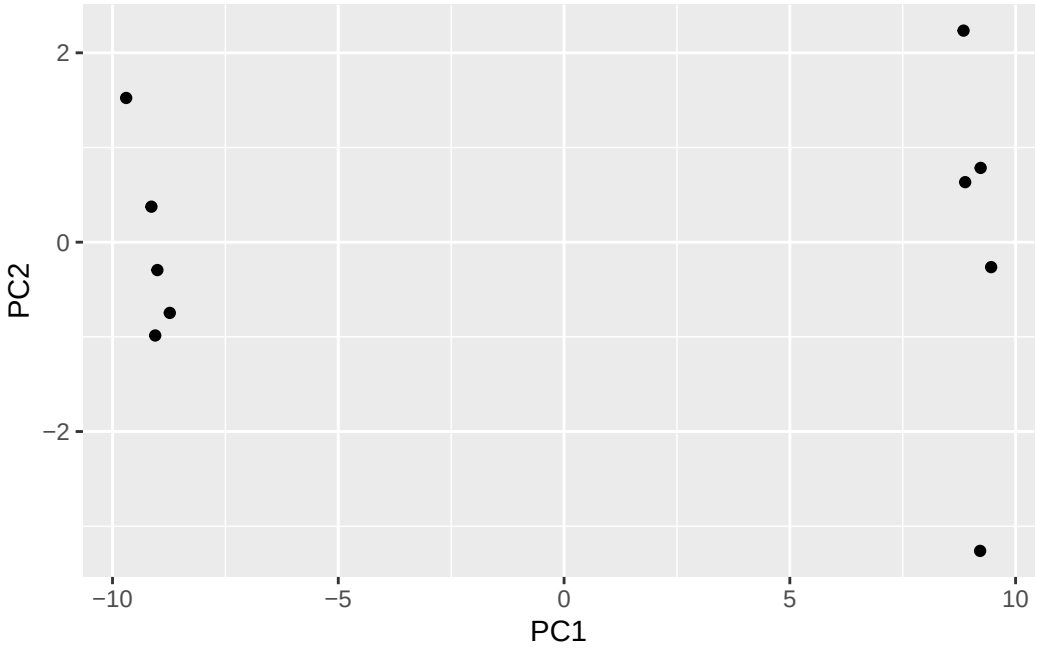
```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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

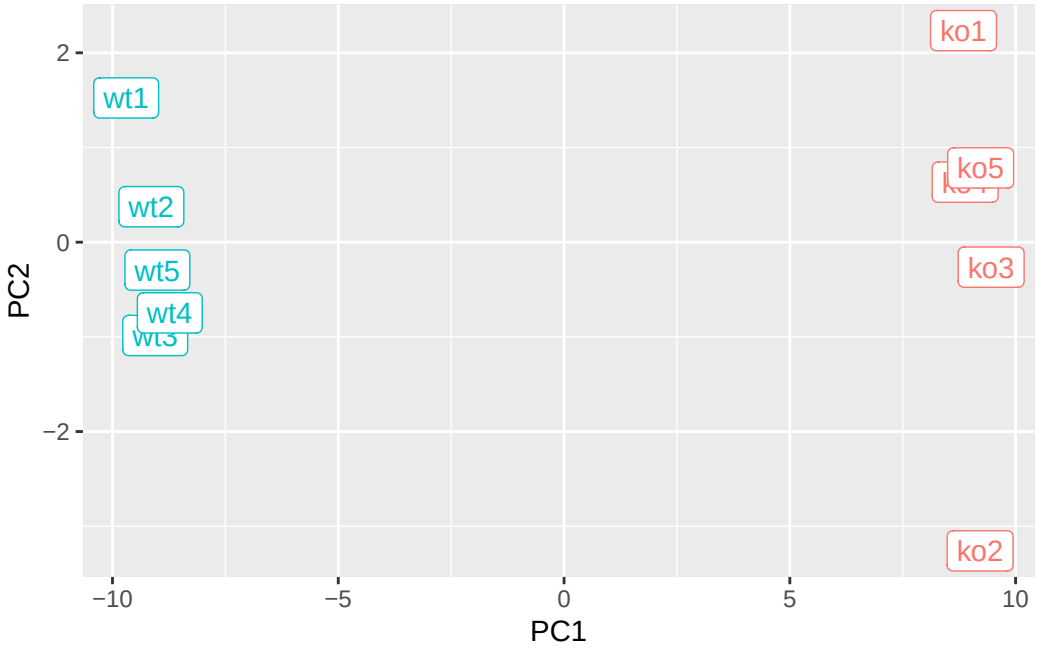
We could use the ggplot2 package here but we will first need a data.frame as input for the main ggplot() function. This data.frame will need to contain our PCA results (specifically pca$x) and additional columns for any other aesthetic mappings we will want to display. We will build this step by step below:

```
library(ggplot2)

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

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



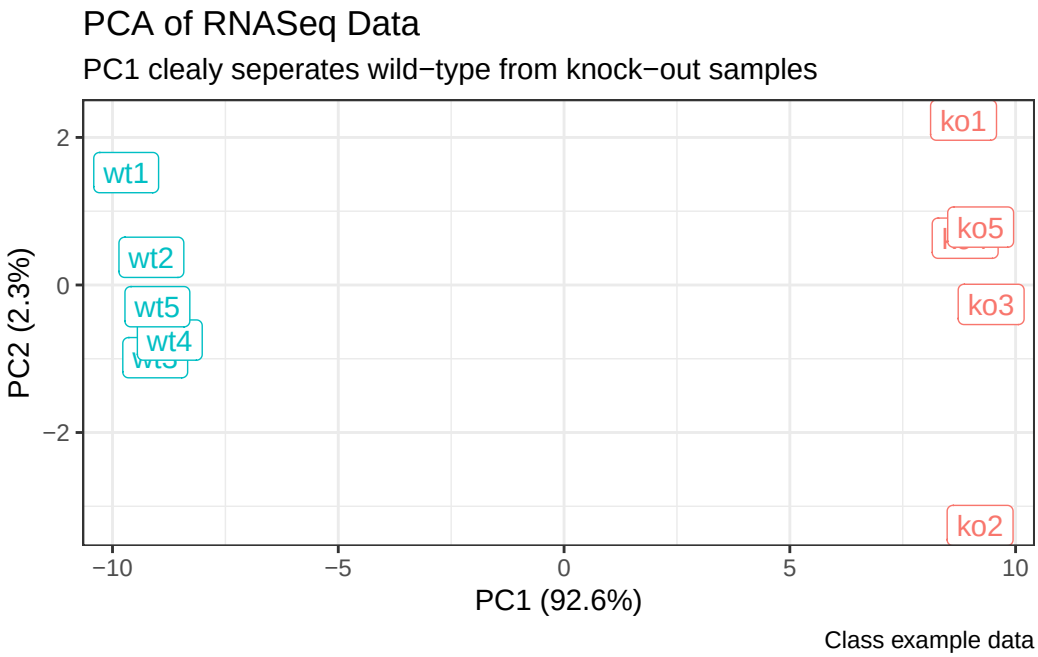
If we want to add a condition specific color and perhaps sample label aesthetics for wild-type and knock-out samples we will need to have this information added to our data.frame:

```
# 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
```



And finally add some spit and polish

```
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"
```

These may be the genes that we would like to focus on for further analysis (if their expression changes are significant - we will deal with this and further steps of RNA-Seq analysis in subsequent classes).

3. Producing a PDF report

Finally for this lab session, please compile a summary report of your work with answers to the above 10 questions and submit to gradescope. To do this you will need your working Quarto or RMarkdown document to be error free (i.e. you can source it without errors) and select the Render option with format: pdf in your YMAL header section.

4. SKIP: Sync to GitHub

If you have your GitHub account setup correctly (and your git tracked repo from a previous class already synced to GitHub) you can now sync today's work to GitHub.

Talk to Barry at this point for some extra discussion and guidance. Essentially, the way you do this will depend on how your current project is setup. Is it already a folder within your GitHub tracked folder? Or is it a separate directory/folder. If it is the later then you will want to quit R Studio and copy your folder into your GitHub tracked folder. Then open this new copy and sync to GitHub via the add/commit/push cycle we used previously. If it is the former then you should be fine to go through the git add/commit/push cycle. Again, discuss with Barry if this is unclear.