Overview
1. PCA of UK food data
2. PCA of RNA-seq data
Muddy Point Assessment
Session Information

BIMM-143, Lecture 8

Code

Hands on with Principal Component Analysis (PCA)

Barry Grant < http://thegrantlab.org/bggn213/ (http://thegrantlab.org/bggn213/) >
2019-02-04 (14:01:29 PST on Mon, Feb 04)

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 (https://bioboot.github.io/bimm143_W19/class-material/pca/). Here we will perform a principal component analysis of several different data sets and examine the results.

1. PCA of UK food data

Suppose that we are examining the following data (https://bioboot.github.io/bggn213_f17/class-material/UK_foods.csv), 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.

	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

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 example (and indeed the second main example further below).

Lets read this data (https://bioboot.github.io/bggn213_f17/class-material/UK_foods.csv) from the provided `UK_foods.csv` input file (note we placed this file in a `data` sub-

directory within or working R studio *project* directory.

Hide

Hide

```
x <- read.csv("data/UK_foods.csv")
```

Q1. How many rows and columns are in your new data frame named `x` ? What R functions could you use to answer this questions?

Hide

Hide

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

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

▶ **HINT**:

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 Rmarkdown document as this is intended for interactive use and will stop the *Knit* rendering process when you go to *Knit* and generate HTML, PDF, MD etc. reports.

Hide

Hide

```
## Preview the first 6 rows
___(x)
```

	X <fctr>	England <int>	Wales <int>	Scotland <int>	N.Ireland <int>
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

6 rows

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 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):

Hide

Hide

```
# Note how the minus indexing works
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

	England <int>	Wales <int>	Scotland <int>	N.Ireland <int>
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

6 rows

This looks much better, now lets check the dimensions again:

Hide

Hide

```
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:

Hide

Hide

```
x <- read.csv("data/UK_foods.csv", row.names=1)
head(x)
```

	England <int>	Wales <int>	Scotland <int>	N.Ireland <int>
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

6 rows

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?

▶ **HINT**:

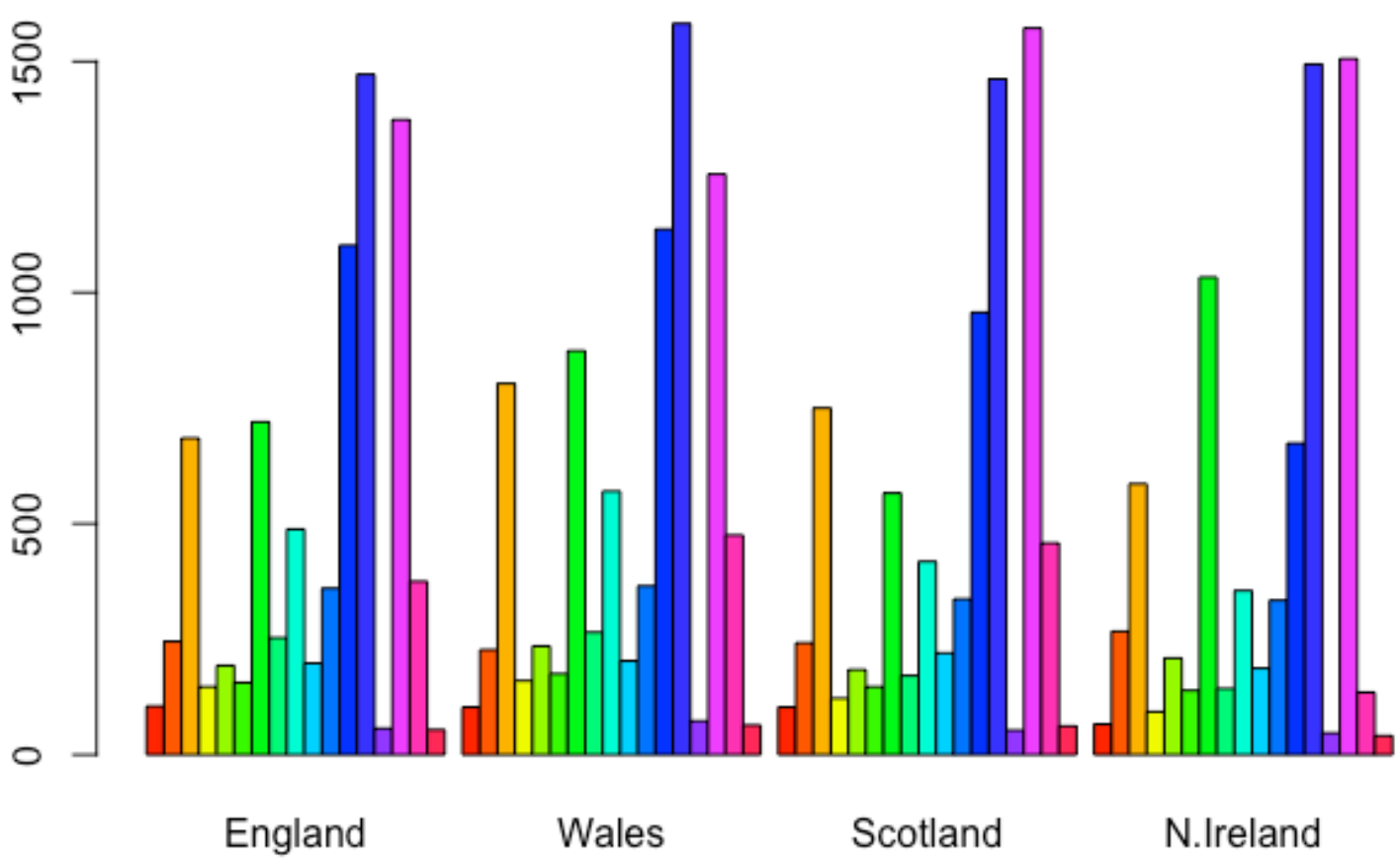
Spotting major differences and trends

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:

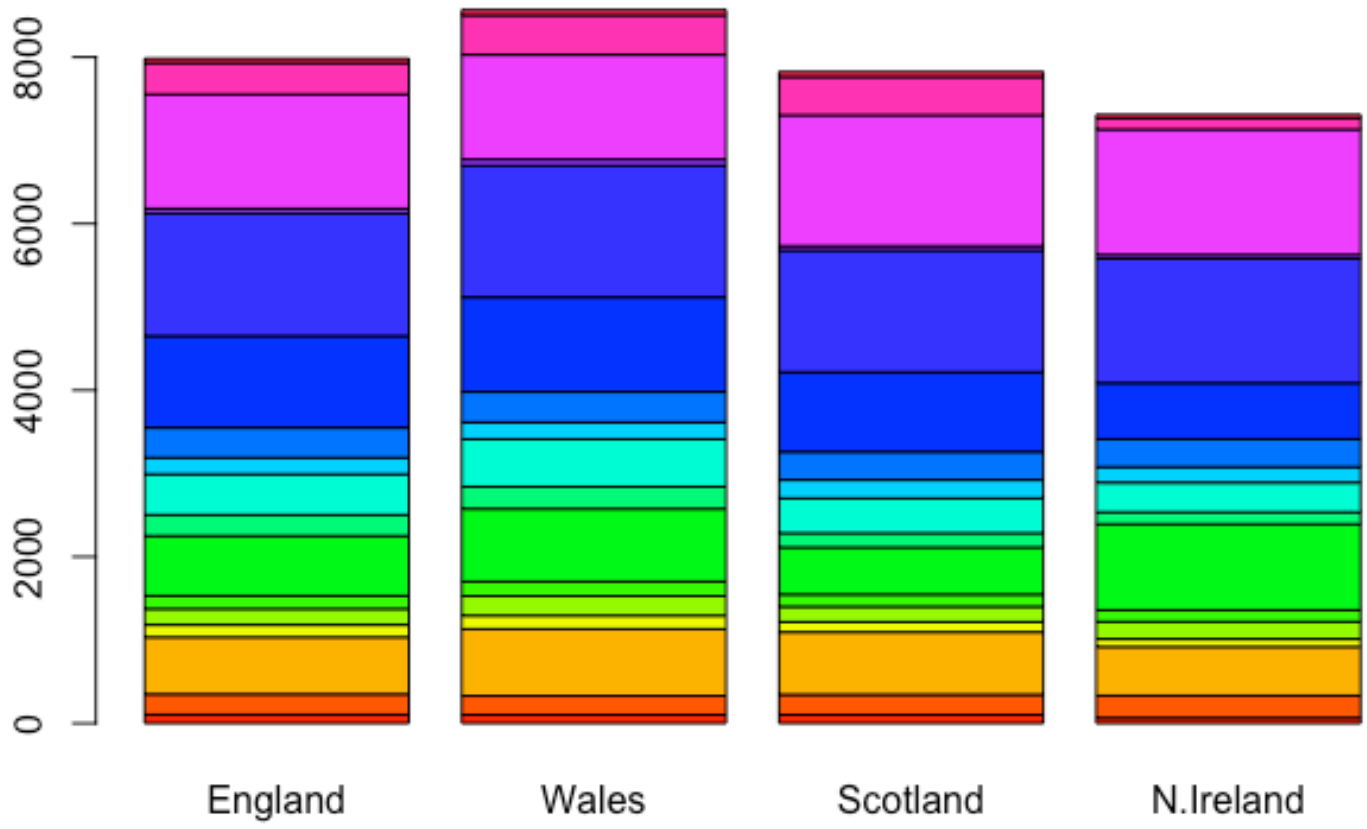
Hide

Hide

```
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?



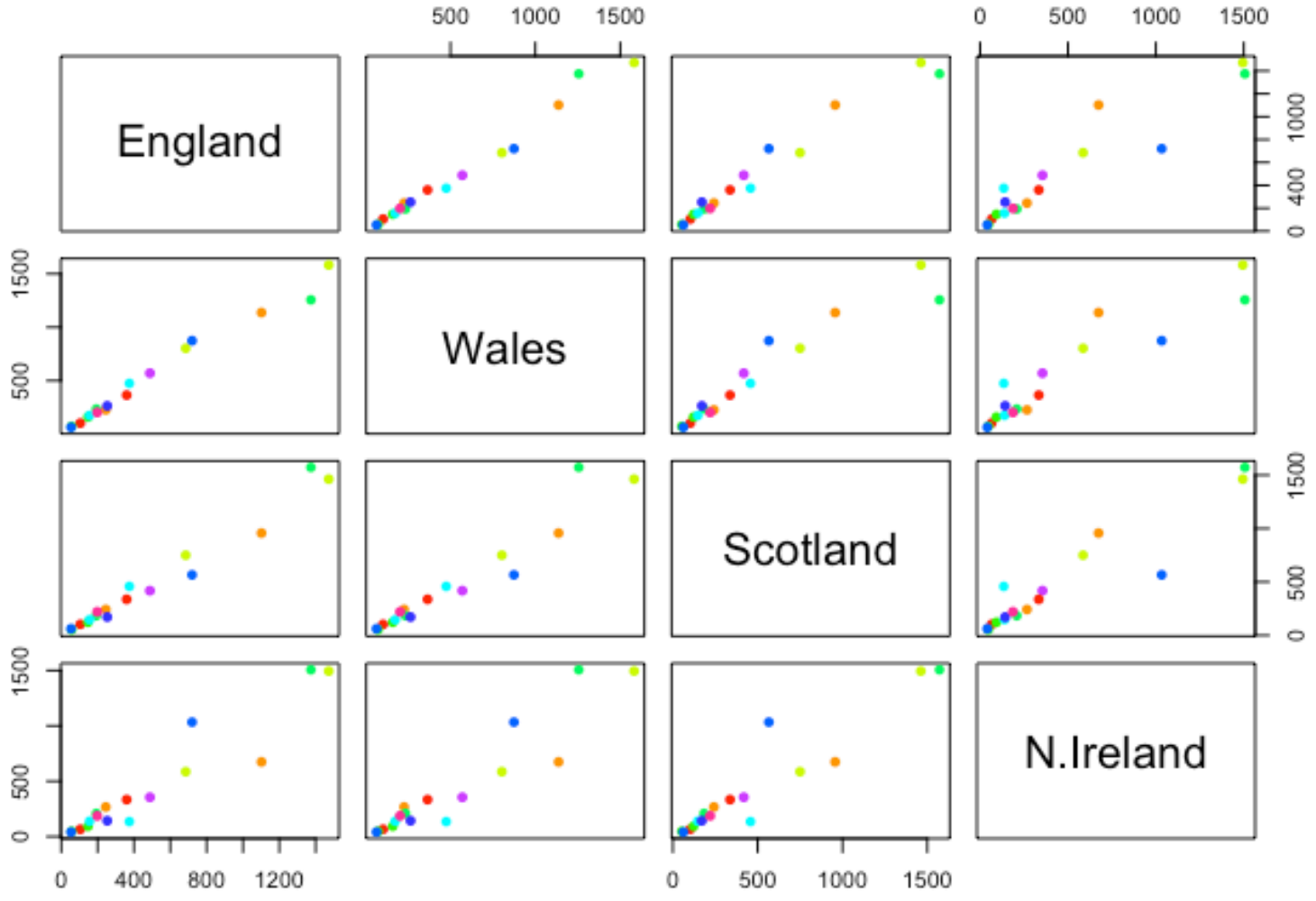
▶ **HINT**:

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?

Hide

Hide

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



Even relatively small datasets can prove chalanging to interpertate
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?

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.

Hide

Hide

```
# Use the prcomp() PCA function
pca <- ___( 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)

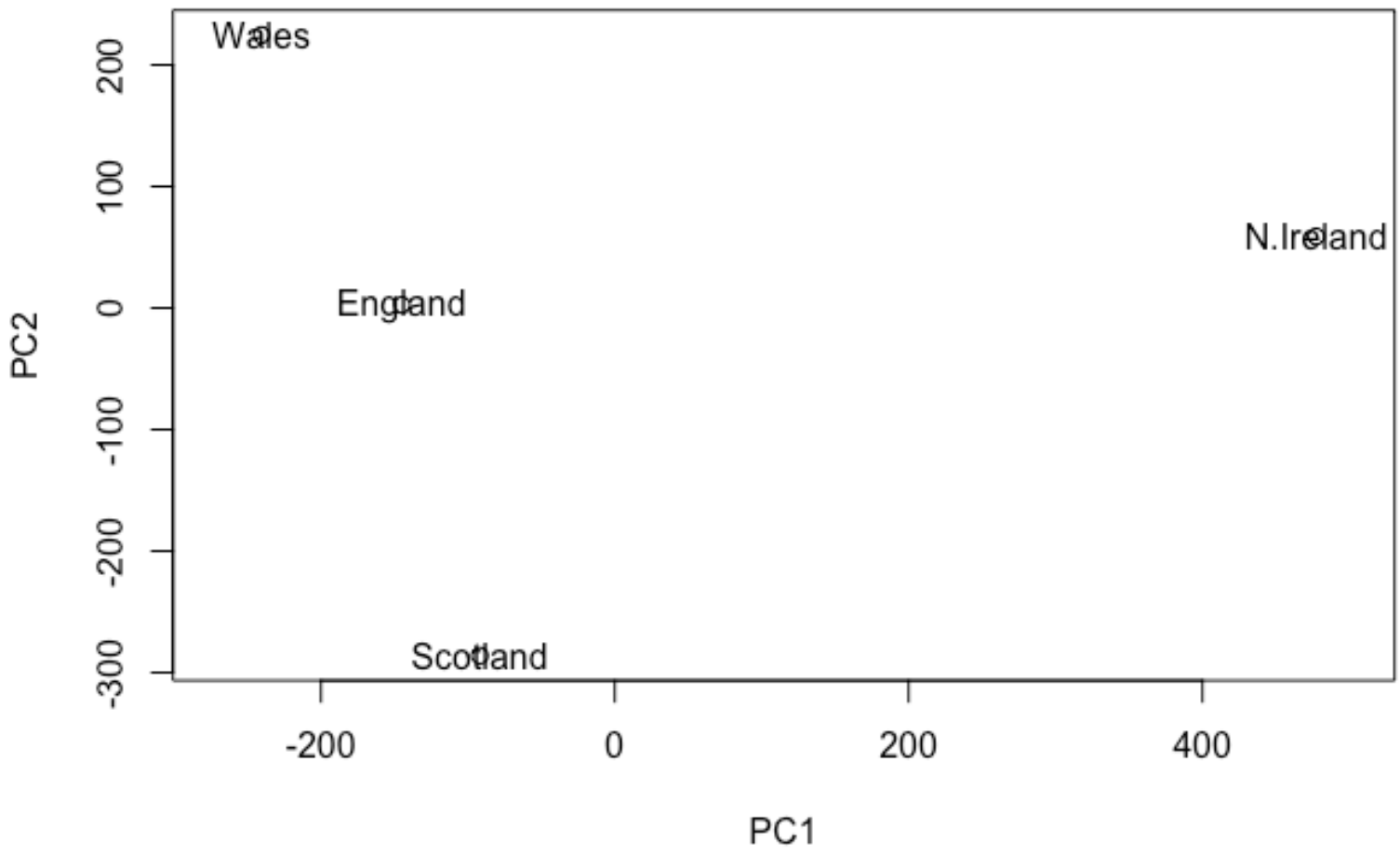
represent useful ways to view and further investigate our data set. Lets start with a simple plot of PC1 vs PC2.

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

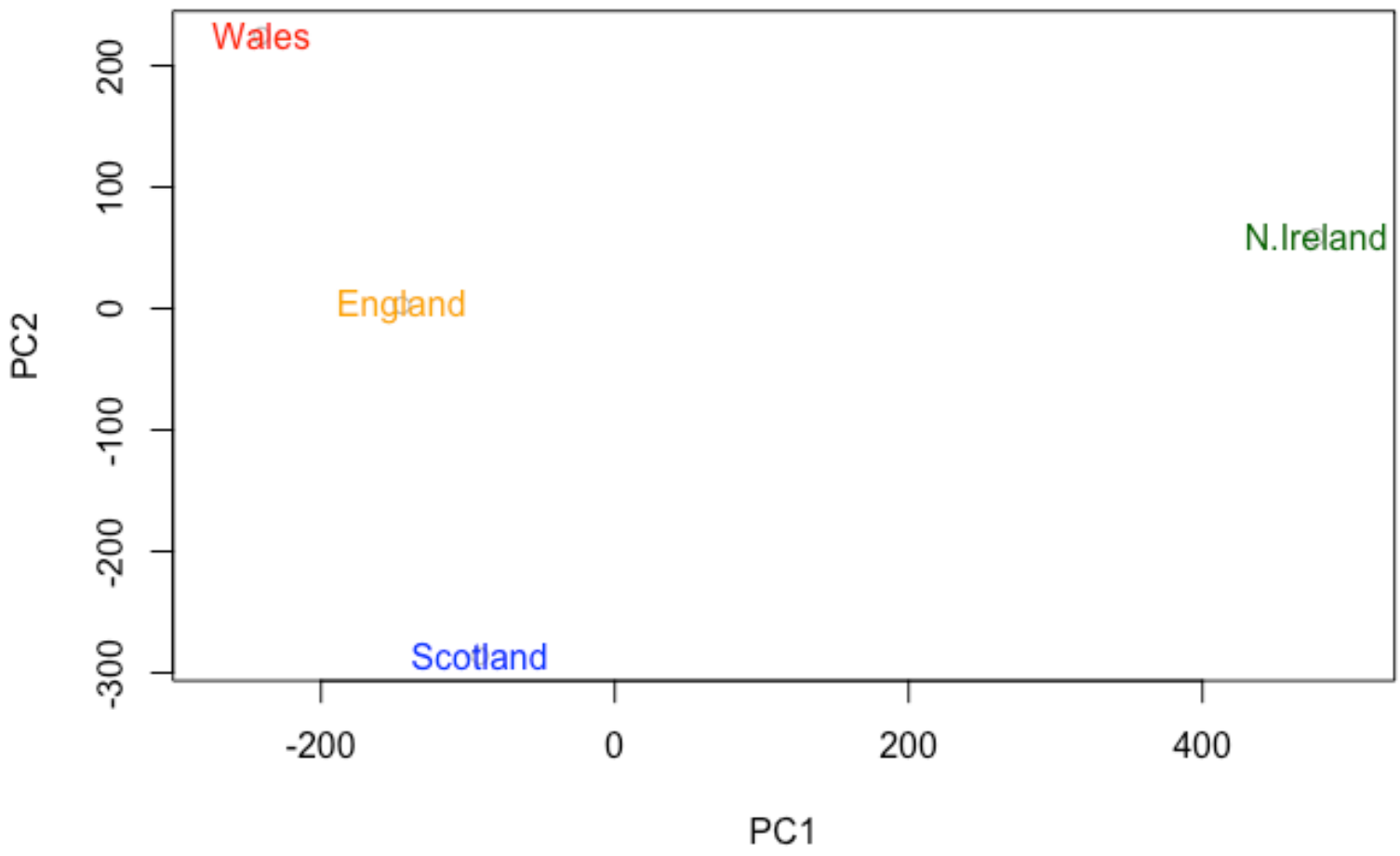
Hide

Hide

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



▶ **HINT**:

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.

Hide

Hide

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

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

Hide

Hide

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

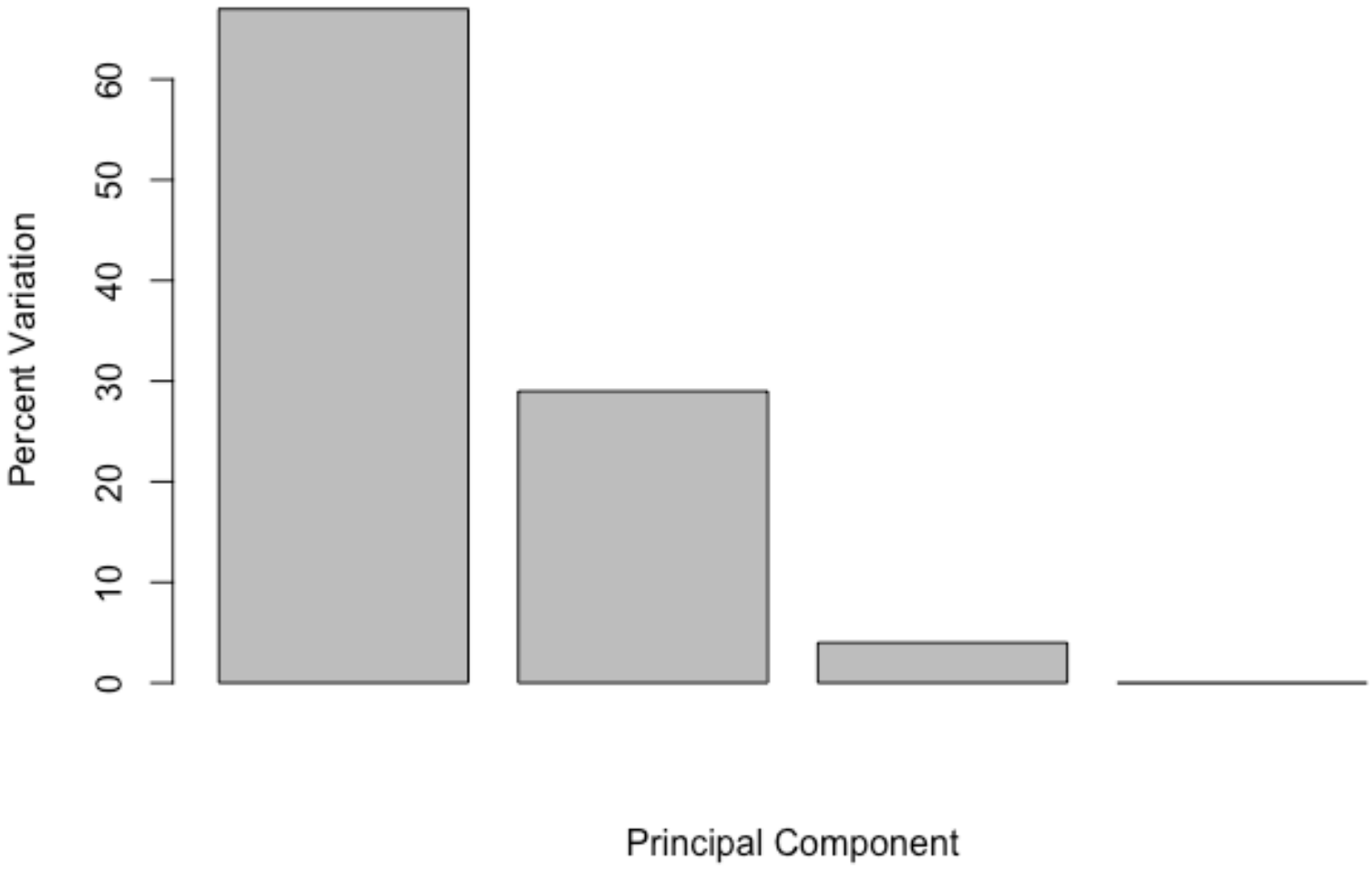
```
##                             PC1       PC2      PC3           PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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.

Hide

Hide

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



Digging deeper (variable loadings)

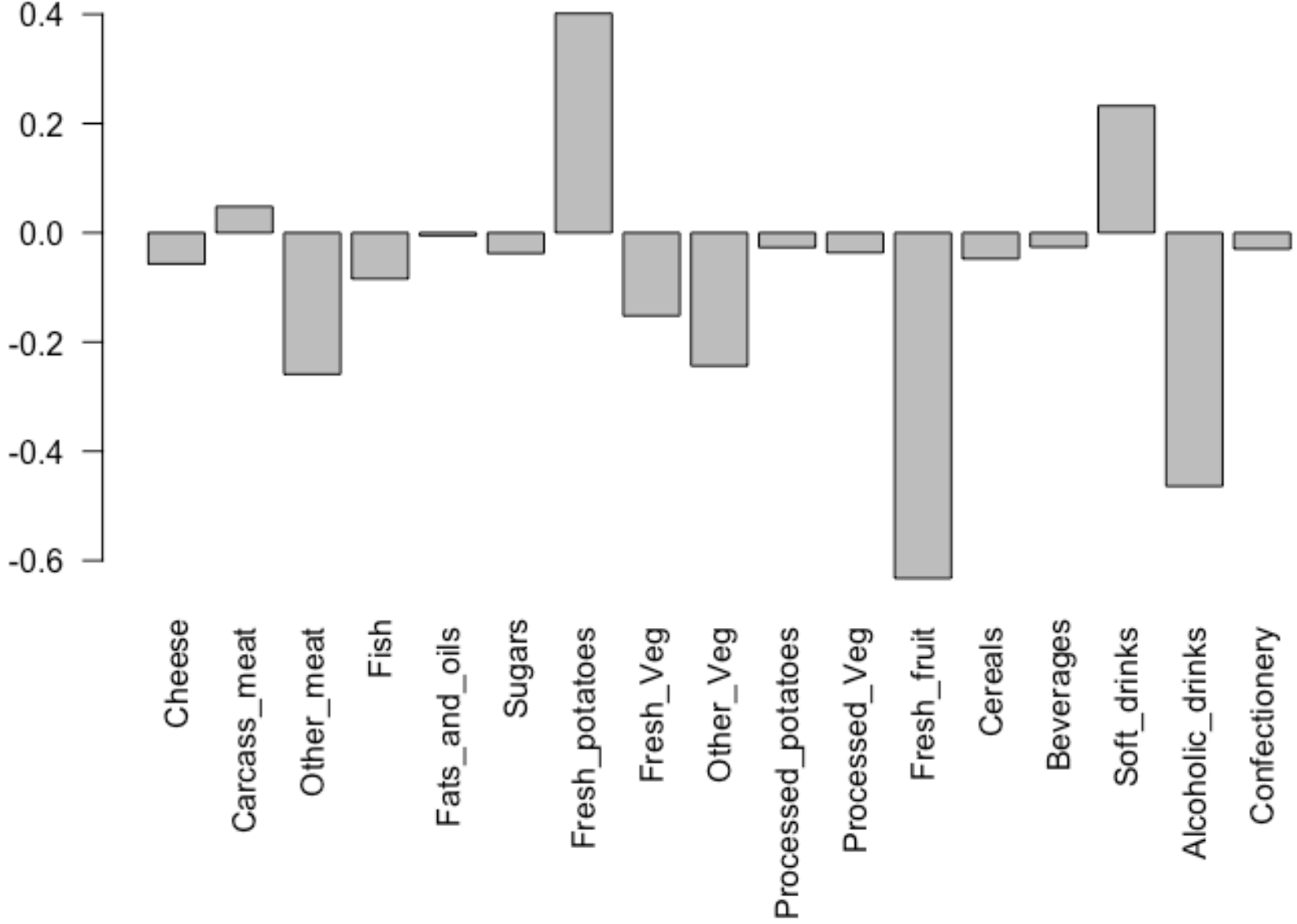
You can include R code in the document as follows:

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:

Hide

Hide

```
## 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?

▶ **HINT**:

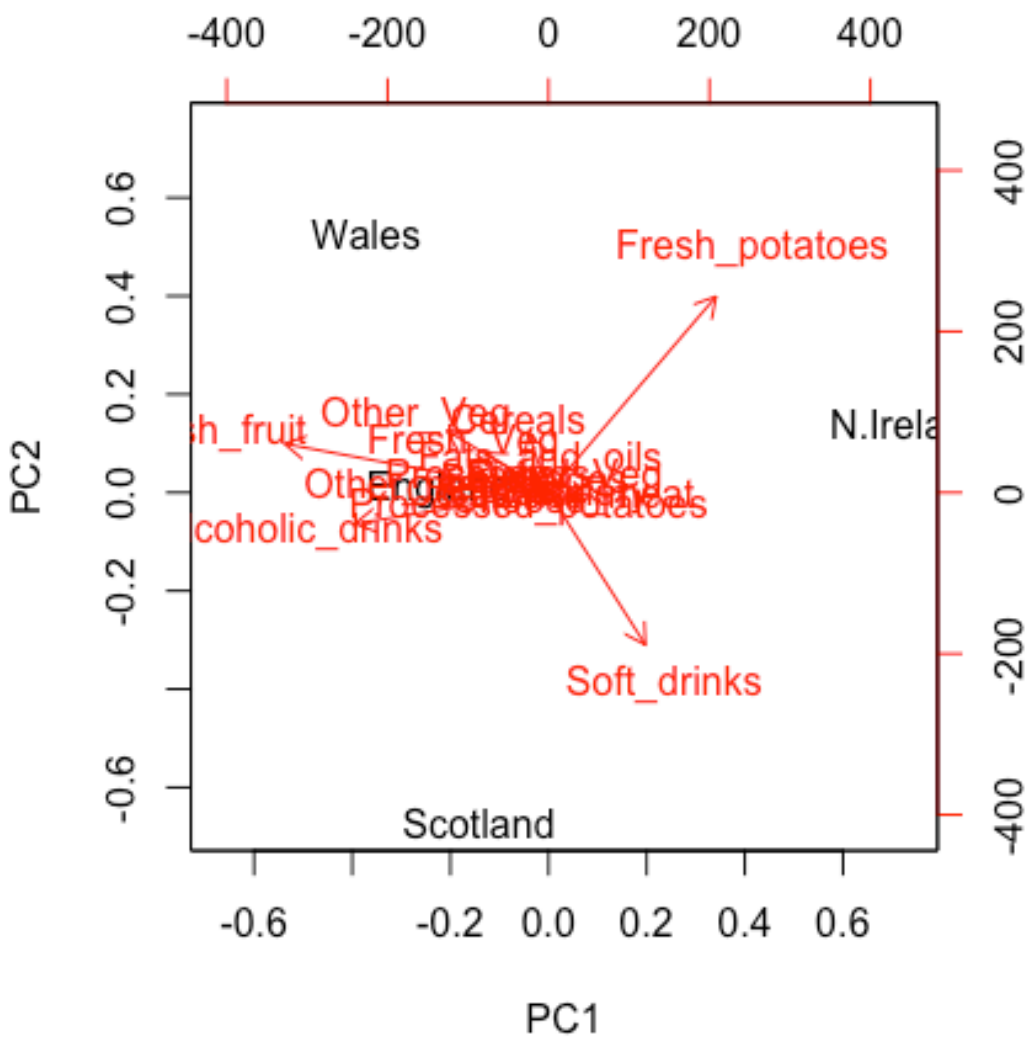
Biplots

Another way to see this information together with the main PCA plot is in a so-called biplot:

Hide

Hide

```
## 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 (https://bioboot.github.io/bggn213_W19/class-material/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*).

Hide

Hide

```
rna.data <- read.csv("data/expression.csv", row.names=1)
head(rna.data)
```

	wt1 <int>	wt2 <int>	wt3 <int>	wt4 <int>	wt5 <int>	ko1 <int>	ko2 <int>	ko3 <int>	ko4 <int>
gene1	439	458	408	429	420	90	88	86	90
gene2	219	200	204	210	187	427	423	434	433
gene3	1006	989	1030	1017	973	252	237	238	226
gene4	783	792	829	856	760	849	856	835	885
gene5	181	249	204	244	225	277	305	272	270
gene6	460	502	491	491	493	612	594	577	618

6 rows | 1-10 of 11 columns

NOTE: The samples are columns, and the genes are rows!

> **Q10**: How many genes and samples are in this data set?

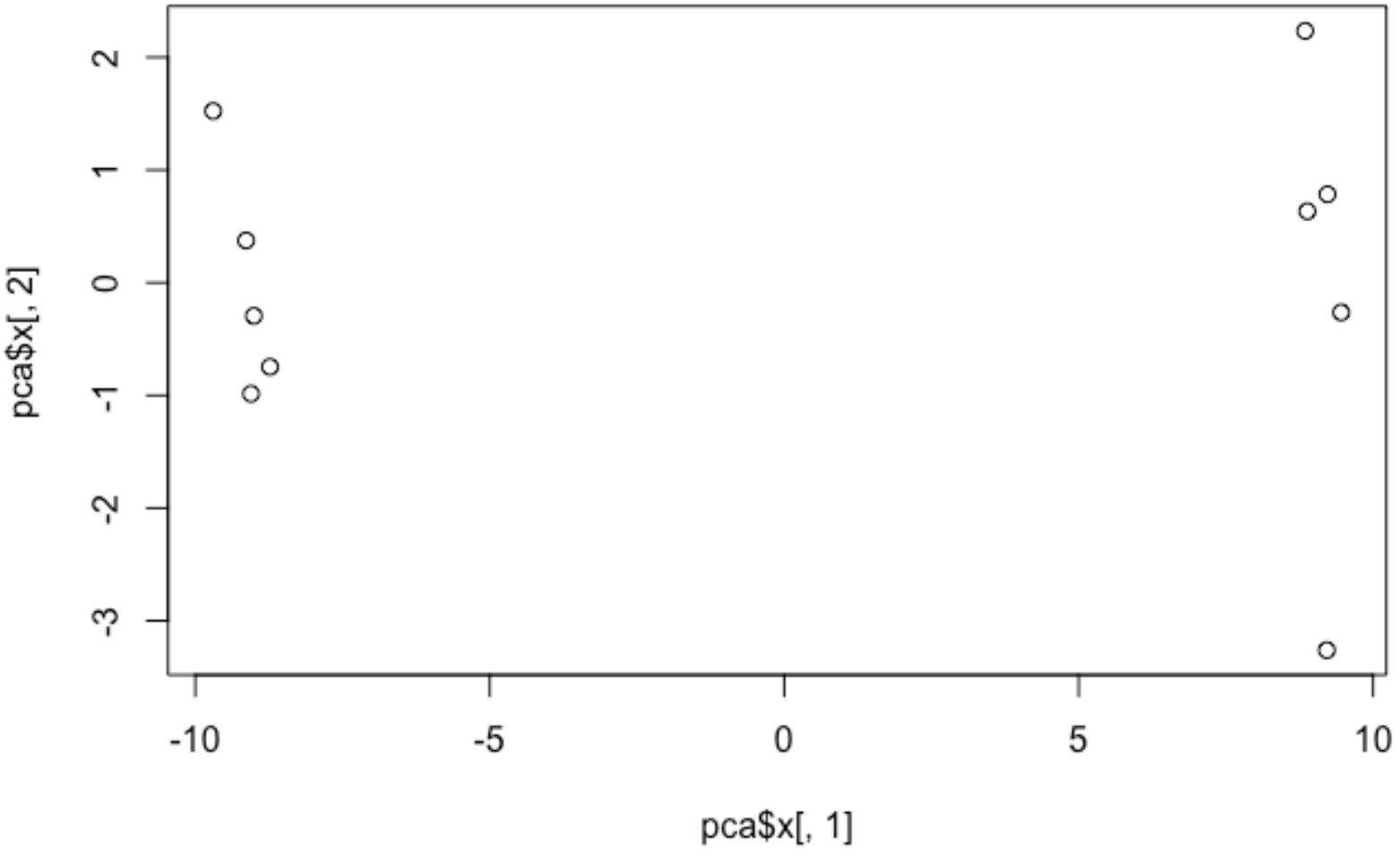
Generating barplots etc. to make sense of this data is really not an exciting or worthwhile option to consider. So lets do PCA and plot the results:

Hide

Hide

```
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

## Simple un ploished plot of pc1 and pc2
plot(pca$x[,1], pca$x[,2])
```



This quick plot looks interesting with a nice separation of samples into two groups of 5 samples each. Now 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:

Hide

Hide

```
## Variance captured per PC
pca.var <- pca$sdev^2

## Precent variance is often more informative to look at
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

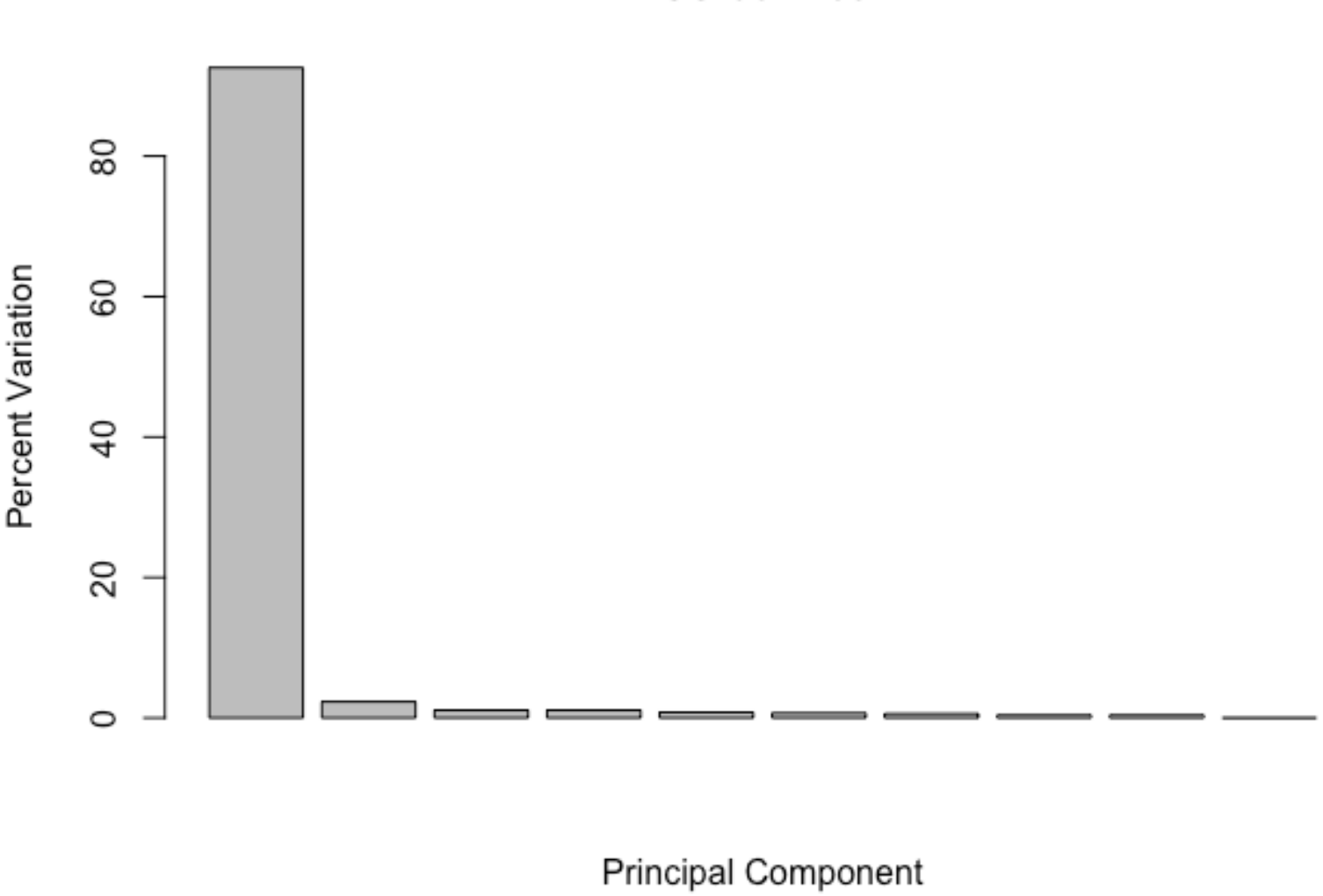
```
##  [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

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

Hide

Hide

```
barplot(pca.var.per, main="Scree Plot",
        xlab="Principal Component", ylab="Percent Variation")
```



> **We can see from these results that PC1 is were all the action is**. 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. This is quite amazing!

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

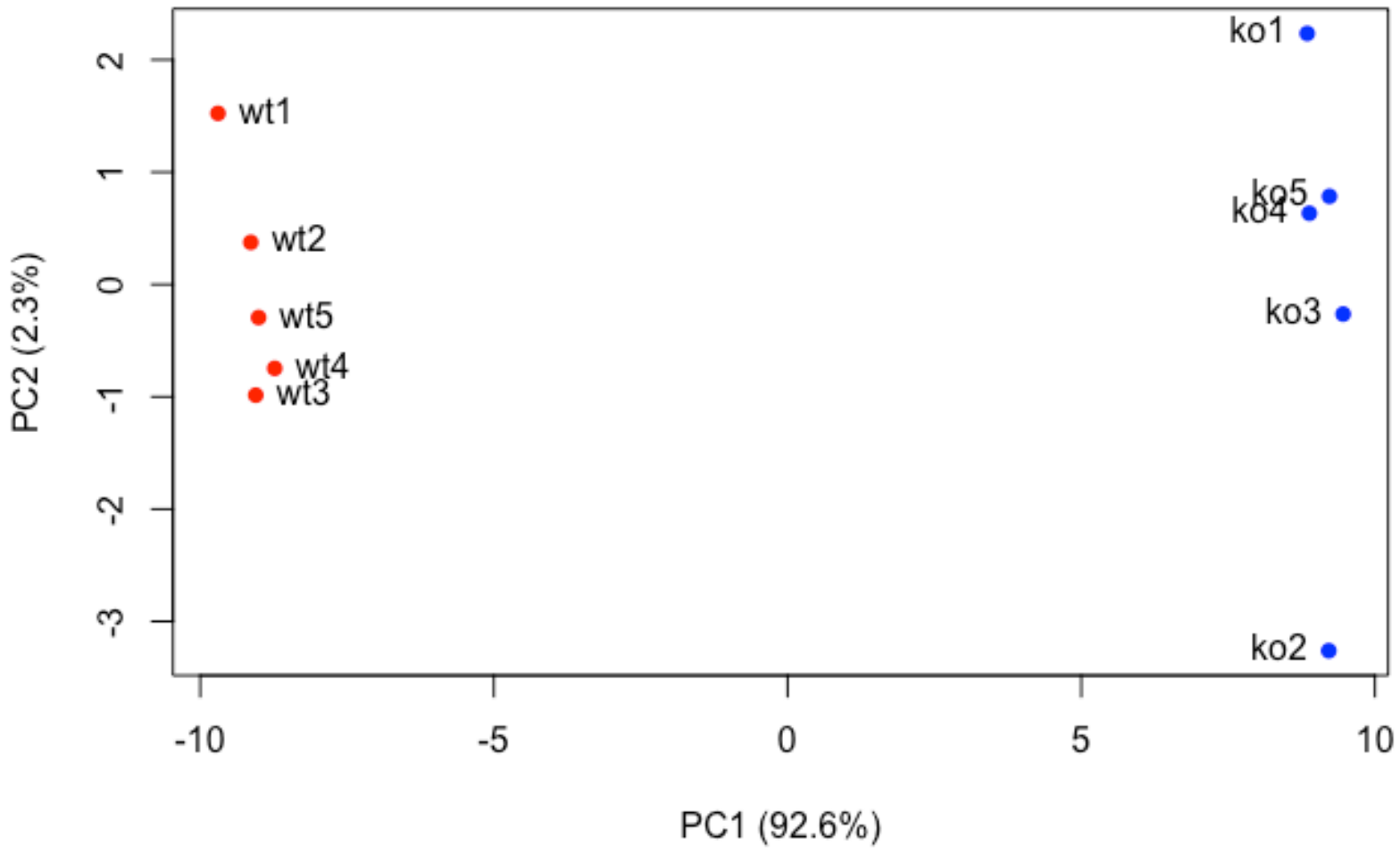
Hide

Hide

```
## 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)))
```



Maybe this is an easier approach to understand for our coloring task:

Hide

Hide

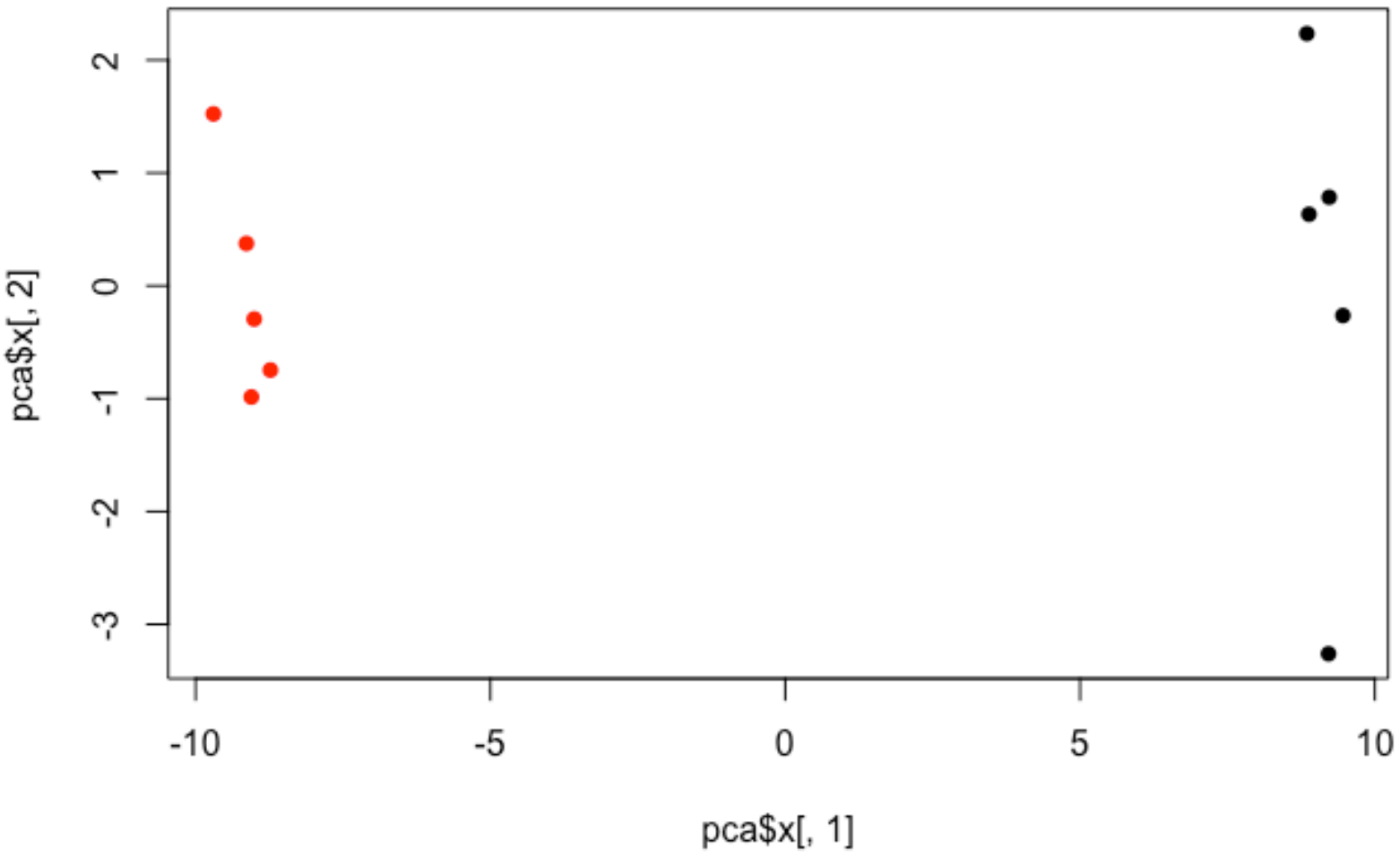
```
## Another way to color by sample type
## Extract the first 2 characters of the sample name
sample.type <- substr(colnames(rna.data),1,2)
sample.type
```

```
##  [1] "wt" "wt" "wt" "wt" "wt" "ko" "ko" "ko" "ko" "ko"
```

Hide

Hide

```
## now use this as a factor input to color our plot
plot(pca$x[,1], pca$x[,2], col=as.factor(sample.type), pch=16)
```



Find the top 10 measurements (genes) that contribute most to pc1 in either direction (+ or -).

Hide

Hide

```
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"  "ge
ne21"
##  [8] "gene56"  "gene10"  "gene90"
```

These may be the genes that we would like to focus on for further analysis (if there expression changes are significant - we will deal with this and further RNA-Seq analysis in Lectures 13-15).

Muddy Point Assessment

Link to today's muddy point assesment (https://docs.google.com/forms/d/e/1FAIpQLSdiQn7n6XvvRGq5AfQWaRa7G22-twFCN4bYNANCWSK09DBYZg/viewform).

Session Information

Hide

Hide

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/lib
Rblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/l
ibRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] formattable_0.2.0.2
##
## loaded via a namespace (and not attached):
##  [1] htmlwidgets_1.3 compiler_3.5.1  backports_1.1.2 magrittr_1.5
##  [5] rprojroot_1.3-2 tools_3.5.1     htmltools_0.3.6 yaml_2.2.0
##  [9] Rcpp_1.0.0      stringi_1.2.4   rmarkdown_1.10  knitr_1.20
## [13] jsonlite_1.6    stringr_1.3.1   digest_0.6.18   evaluate_0.12
```