PCA Week 6

Yash Garodia

09/02/2022

##1. PCA of UK Data

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".

```
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?

```
nrow(x)
```

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

```
ncol(x)
```

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

```
dim(x)
```

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

```
#There are 17 rows and 5 columns. You can use the nrow() function to find the number of rows and ncol()
```

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).

```
#View(x)
# To Preview the first 6 rows, we use the head() function
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 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):

```
#minus indexing used to remove the specific row or column
#The rownames() functionw will remove the name of the first column in the header
rownames(x) <- x[,1]
#The function below will remove the first column as a whole.
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
```

These are the right dimensions we are looking for! 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)

```
#This function will set the first row as the first set of columns
x <- read.csv(url, row.names = 1)
head(x)
```

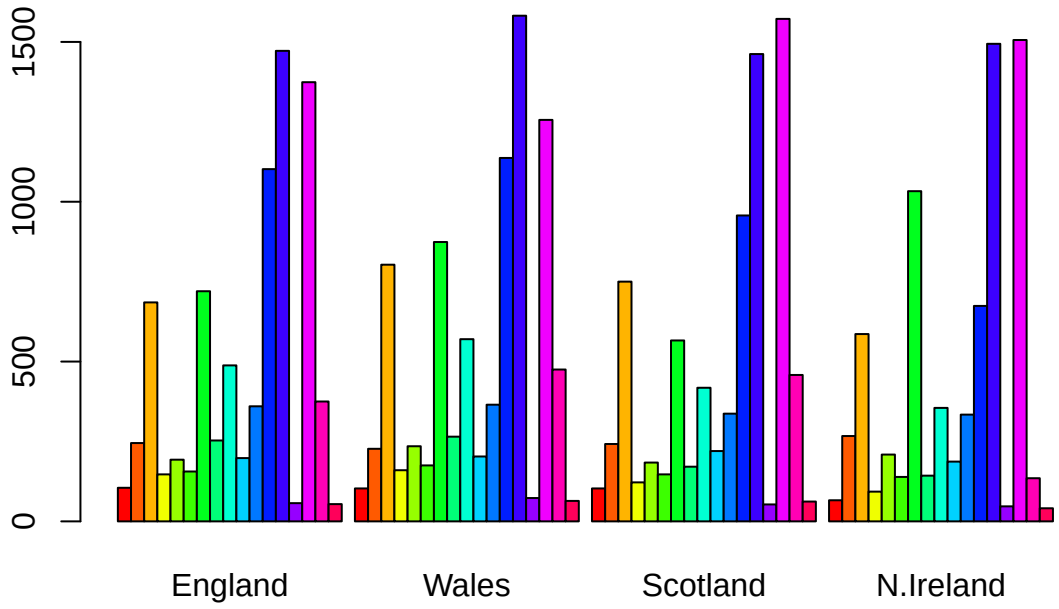
```
##                England Wales Scotland N.Ireland
## Cheese             105   103      103        66
## Carcass_meat       245   227      242       267
## Other_meat         685   803      750       586
## Fish               147   160      122        93
## Fats_and_oils      193   235      184       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 involving the use of row.names because it is faster and more robust than the other. Also because if you run the first code (x <- x[,-1]) more than once even by mistake then you lose another column. In the second approach, there is no scope for such an error.

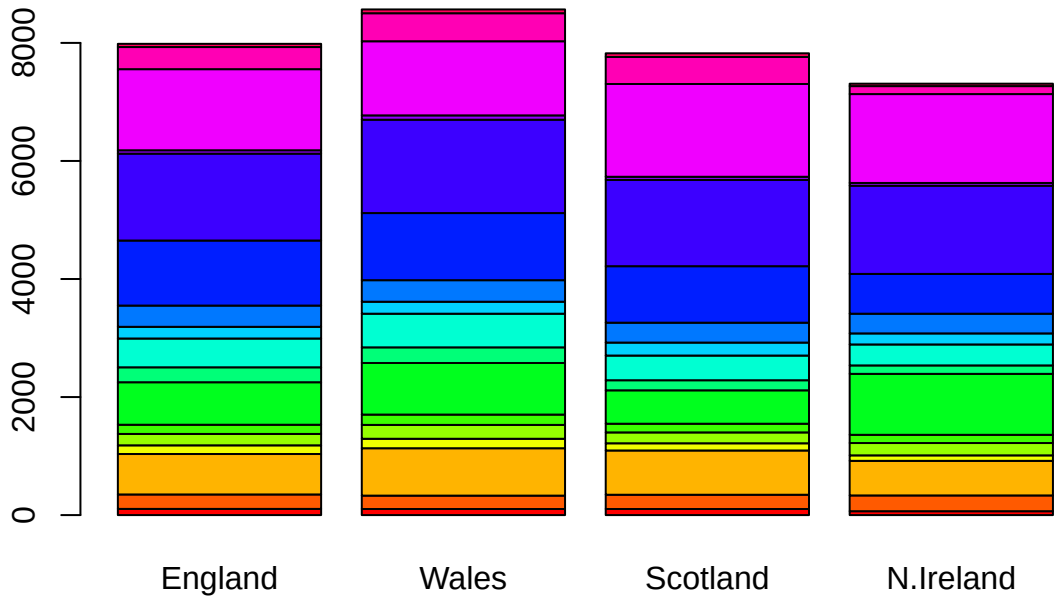
A cursory glance over the numbers in this table does not reveal much of anything. Indeed in general it is difficult to extract meaning in regard to major differences and trends from any given array of numbers. Generating regular bar-plots and various pairwise plots does not help too much either:

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



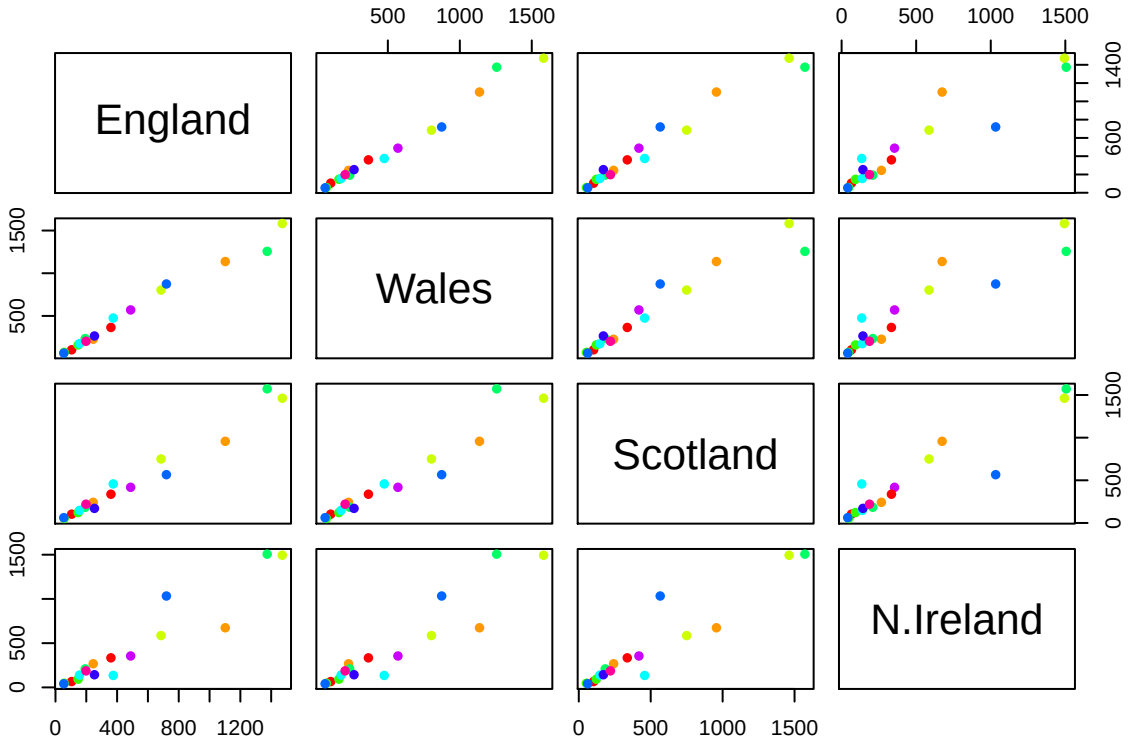
> Q3: Changing what optional argument in the above barplot() function results in the following plot?

```
#In order to solve this problem, we must look at how we want our barplot to be in comparison to what it
#?barplot
#The documentation for barplot suggests that changing the beside argument from true to false should sta
barplot(as.matrix(x), beside = F, col = rainbow(nrow(x)))
```



This is the result we were looking for! So the answer to the question is that we change the beside argument from True to False. > 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)
```



The displayed code creates matrices of scatterplots that look at all possible combinations of all pairs of countries against each other. In the first row, England is the y axis for all, while Wales is on the x axis for the second plot, Scotland for the third and N. Ireland for fourth. If the given point lies on the diagonal, it means that equal amounts of the food (the variable in this case) is being consumed by people from both countries.

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

There are much fewer data points on the diagonal for N.Ireland with other countries.

> 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.

#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? Lets try using PCA plot using the base prcomp() function:

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

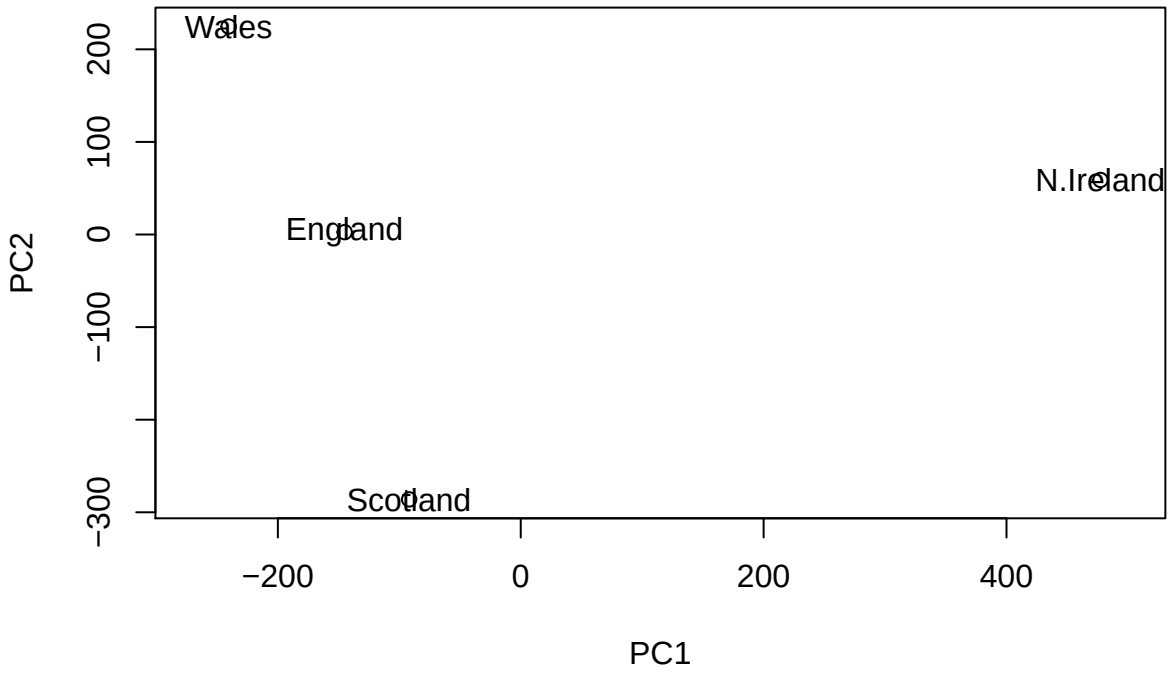
```
## Importance of components:
##                             PC1       PC2       PC3       PC4
```

```
## 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 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.

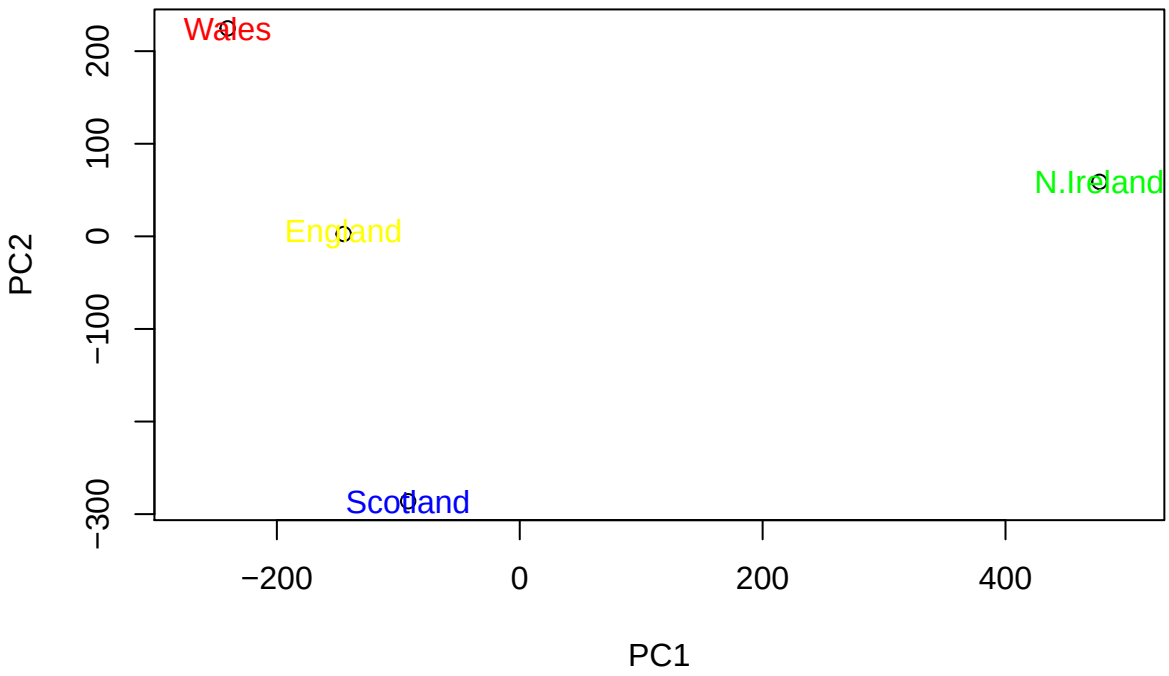
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
plot(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.

```
#First we create a variable to which we assign the colors. Colors are assigned in order of the countrie
country_cols <- c("yellow", "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)
```



> 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
```

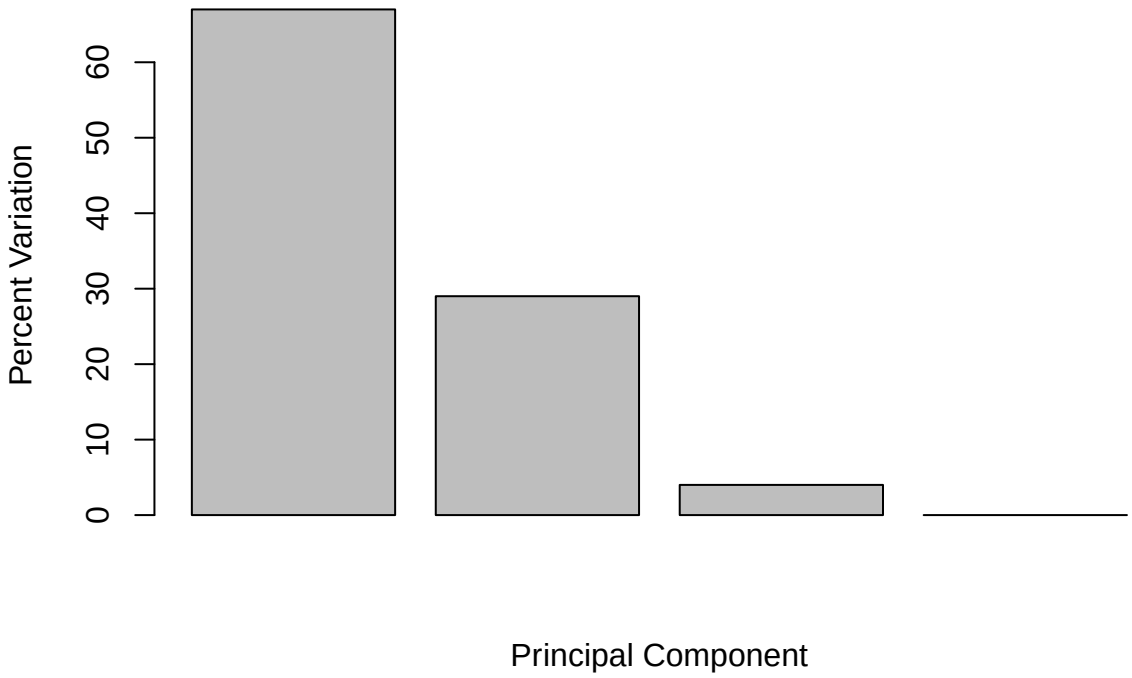
Or for the second row:

```
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.

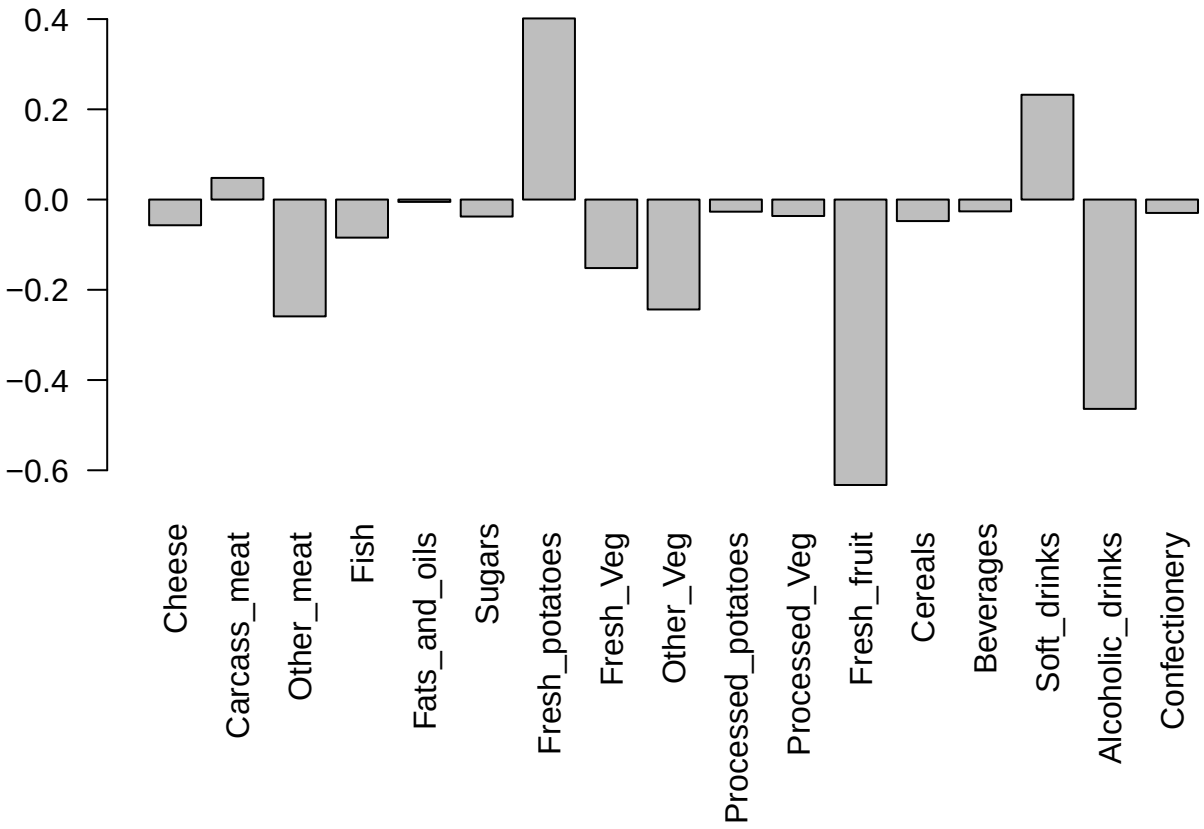
```
barplot(v, xlab = "Principal Component", ylab = "Percent Variation")
```



#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:

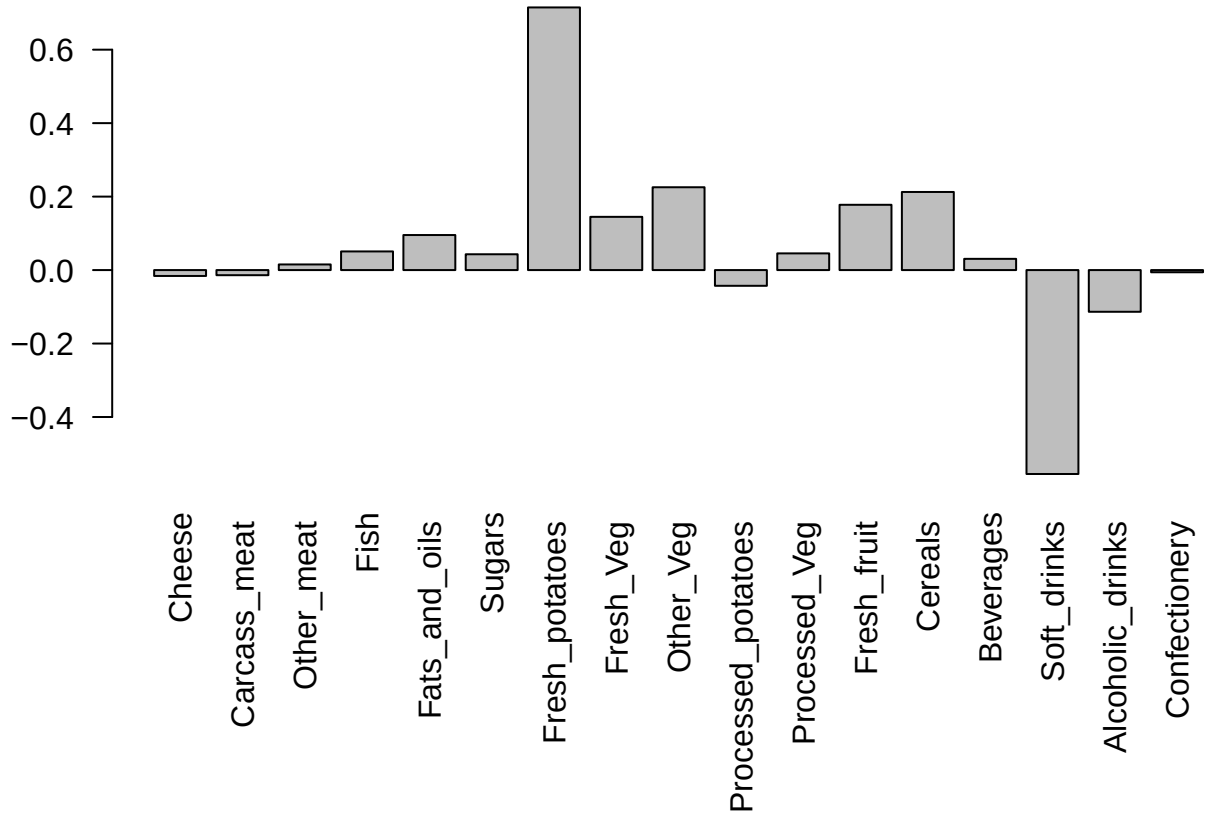
```
#Lets focus on PC1 as it accounts for more than 90% of variance
par(mar = c(10, 3, 0.35, 0))
barplot(pca$rotation[,1], las = 2)
```



> 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?

```
#To change from PC1 to PC2 we just change the desired column from 1 to 2.
par(mar = c(10, 3, 0.35, 0))
barplot(pca$rotation[,2], las = 2)
```

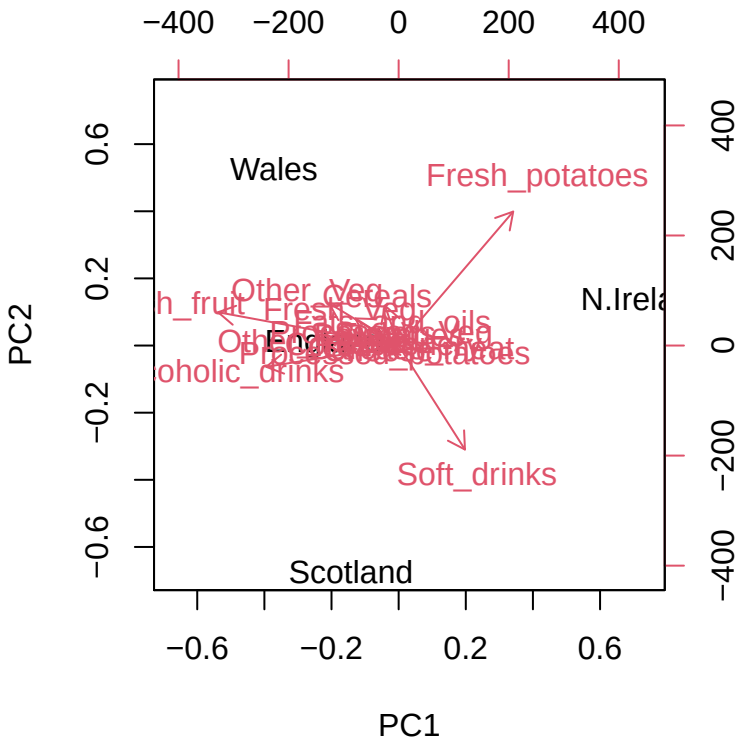


From this loadings plot, we can see that fresh potatoes and soft drinks feature prominently. This suggests that fresh potatoes and soft drinks are the primary variables driving PC2.Scotland has the highest level of soft drink consumption, explaining why it is at the bottom of the plot as soft drinks have a high negative score. Wales have the lowest soft drink consumption so they are at the top. Despite having high levels of soft drink consumption, the fact that N. Ireland also has high fresh potato consumption causes it to be pushed roughly towards 0 as the high negative and positive score almost completely cancel out; the fact that they have the highest potato consumption causes their score to be slightly above 0. England has moderate levels of both fresh potato consumption and soft drink consumption, explaining why they are at a height of 0.

#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.

##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).

```
#Importing dataset from the URL
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names = 1)
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
## gene7     27  30   37   29   34 304  304  285  311  285
## gene8    175 182  184  166  180 255  291  305  271  269
## gene9    658 669  653  633  657 628  627  603  635  620
## gene10   121 116  134  117  133 931  941  990  982  934
## gene11   337 337  330  322  313 100   95   94  101   79
## gene12   214 194  213  192  207  97   91   89  124   97
## gene13   789 738  807  768  820 293  308  312  303  325
## gene14   458 490  493  446  496 694  682  679  702  719
## gene15   551 555  527  552  503 712  742  718  808  739
## gene16   390 400  403  402  401 755  765  730  713  740
## gene17   900 970  905  850  834 353  380  380  385  386
## gene18   951 991  991  983  984 217  195  195  196  197
## gene19   436 414  388  418  410 162  169  143  151  130
## gene20   244 266  228  223  240 540  536  577  538  513
## gene21   119  87   87   88   93 914  906  914  913  921
## gene22   156 170  150  167  155 346  372  393  416  384
## gene23    89  97   96   97   82 788  786  750  822  785
## gene24   570 567  563  587  563 424  481  489  456  465
## gene25   788 796  766  778  825 456  403  446  447  442
## gene26  1007 972  977 1003 1027 945  859  933  844  925
## gene27   937 876  901  958  957 414  405  383  437  394
## gene28   224 232  231  238  226 850  902  907  842  817
## gene29   809 869  815  788  781 482  484  518  498  491
## gene30   624 598  587  552  592 956  985  940  963  982
## gene31   218 259  213  204  213  69   86   59   65   46
## gene32   906 798  828  874  890 541  626  576  607  586
## gene33   262 291  258  271  279 534  566  570  565  563
## gene34   155 172  173  173  192 643  639  713  706  676
## gene35   100 104   94  114   90 212  228  233  229  258
## gene36   117 147  120  147  145 353  347  371  335  357
## gene37   286 262  260  270  293 360  375  361  348  374
## gene38   321 353  334  340  316 642  575  588  595  665
## gene39   388 372  345  373  359  50   45   39   44   35
## gene40   606 576  558  581  574 415  406  423  455  412
## gene41   379 377  362  346  354 991 1010 1020  976 1036
## gene42   471 492  473  470  471 401  401  426  425  418
## gene43   592 615  602  602  655 514  554  501  511  553
## gene44   755 733  775  687  776 255  245  251  249  252
## gene45    35  40   28   25   32 947  988  994  989  971
## gene46   758 734  704  761  672 567  575  596  607  611
## gene47    24  25   12   13   22 324  293  292  303  295
## gene48   100 113  136  117  103 912  940  901  950  868
## gene49   809 825  833  800  776 538  524  487  527  507
## gene50   955 994  994  975  973 175  158  191  218  183
## gene51   453 419  443  459  469 174  134  166  148  154
## gene52   327 320  324  321  318 489  470  495  451  457
## gene53   657 669  631  701  647 246  276  255  266  287
## gene54   678 638  676  683  671 259  247  238  214  235
```

```
## gene55   304 325  312  327  320 819  802  773  790  820
## gene56   659 687  659  667  639 109  102  105  119   96
## gene57   673 668  694  699  726  18   14   19   18   14
## gene58   785 772  817  766  784 467  474  460  461  481
## gene59   501 513  462  484  504  37   64   71   58   50
## gene60   232 228  193  247  231 997  983  997  990 1011
## gene61   928 936 1015  971  964 428  457  447  434  431
## gene62   159 169  163  151  166 869  975  955  929  948
## gene63   336 344  372  389  357 664  575  577  625  630
## gene64   968 888  907  914  883 886  855  844  848  862
## gene65   339 335  373  338  328 275  290  270  303  280
## gene66    35  32   45   37   38 765  746  756  758  761
## gene67    27  28   25   35   27 200  194  189  181  173
## gene68    80  69   87   87   81 693  693  677  683  688
## gene69   744 685  733  693  746 745  680  780  791  792
## gene70   766 739  751  720  738 645  603  610  598  612
## gene71   672 736  672  715  693 839  872  909  811  803
## gene72   526 553  534  511  529 922  819  878  832  853
## gene73   627 650  664  622  606 805  836  836  828  800
## gene74   468 466  477  469  494 703  661  669  632  640
## gene75   986 945 1006 1020 1024 359  358  346  356  345
## gene76   348 333  344  321  296 770  773  750  769  774
## gene77   719 714  734  693  682 620  567  582  614  546
## gene78   883 899  868  873  882 803  765  767  783  749
## gene79   837 883  864  807  854 210  239  234  258  220
## gene80   666 657  719  656  638 549  588  586  571  583
## gene81   804 735  771  763  813 613  587  591  563  613
## gene82   476 494  521  494  482 183  184  156  173  161
## gene83   438 430  477  457  481 466  525  518  474  478
## gene84   938 934  976  965  960 904 1011  949  947  934
## gene85    29  29   30   19   21 618  589  618  563  574
## gene86   810 830  760  796  807 486  542  507  471  543
## gene87   575 579  567  565  576 352  321  296  332  311
## gene88   451 471  494  447  470 540  583  572  551  591
## gene89   174 170  205  175  179 298  290  319  313  264
## gene90   158 122  138  159  128 863  896  869  841  873
## gene91   371 367  369  339  360 103   85   83   94   70
## gene92   853 798  866  843  823 934 1007  936  918 1005
## gene93   208 214  200  196  206 409  408  403  368  380
## gene94   555 584  574  599  581 292  341  335  324  299
## gene95   527 573  548  548  552 686  718  705  704  677
## gene96   589 607  579  536  583 497  479  479  467  504
## gene97   396 384  382  399  401 460  442  466  452  457
## gene98    33  27   39   42   33 977 1031 1033 1003  974
## gene99   321 343  349  367  343 949  947  982 1021 1010
## gene100   25  34   34   36   32 661  685  678  655  693
```

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

Genes are rows and samples are columns.

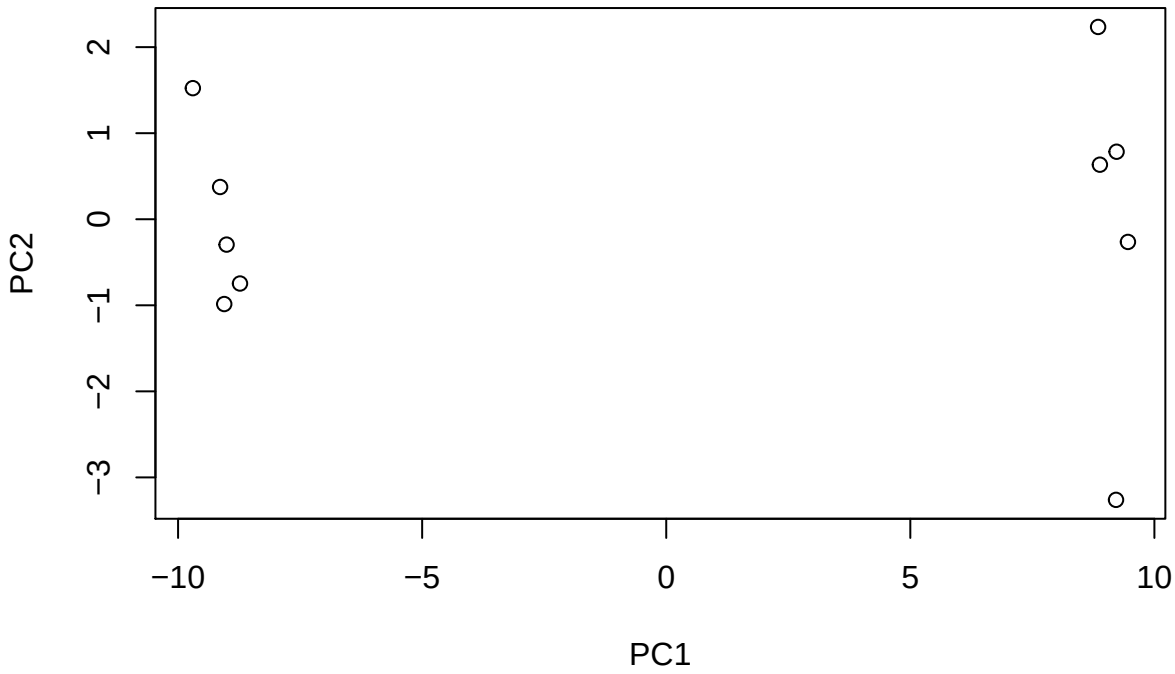
```
dim(rna.data)
```

```
## [1] 100  10
```

There are 100 rows and 11 columns.

> 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")
```

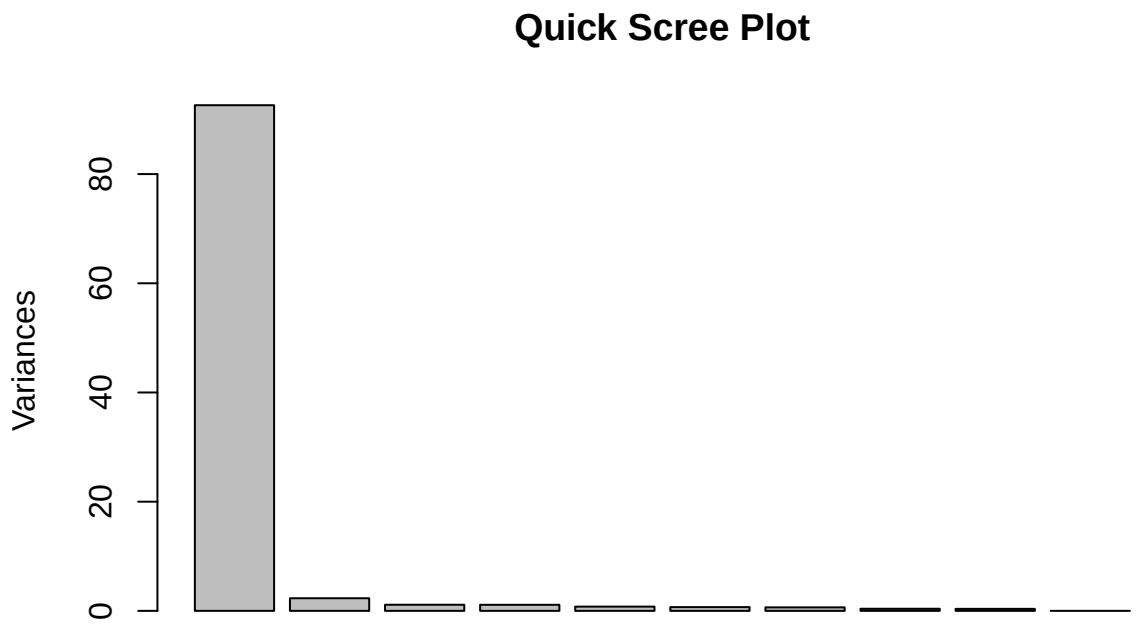



```
summary(pca)
```

```
## 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
```

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")
```



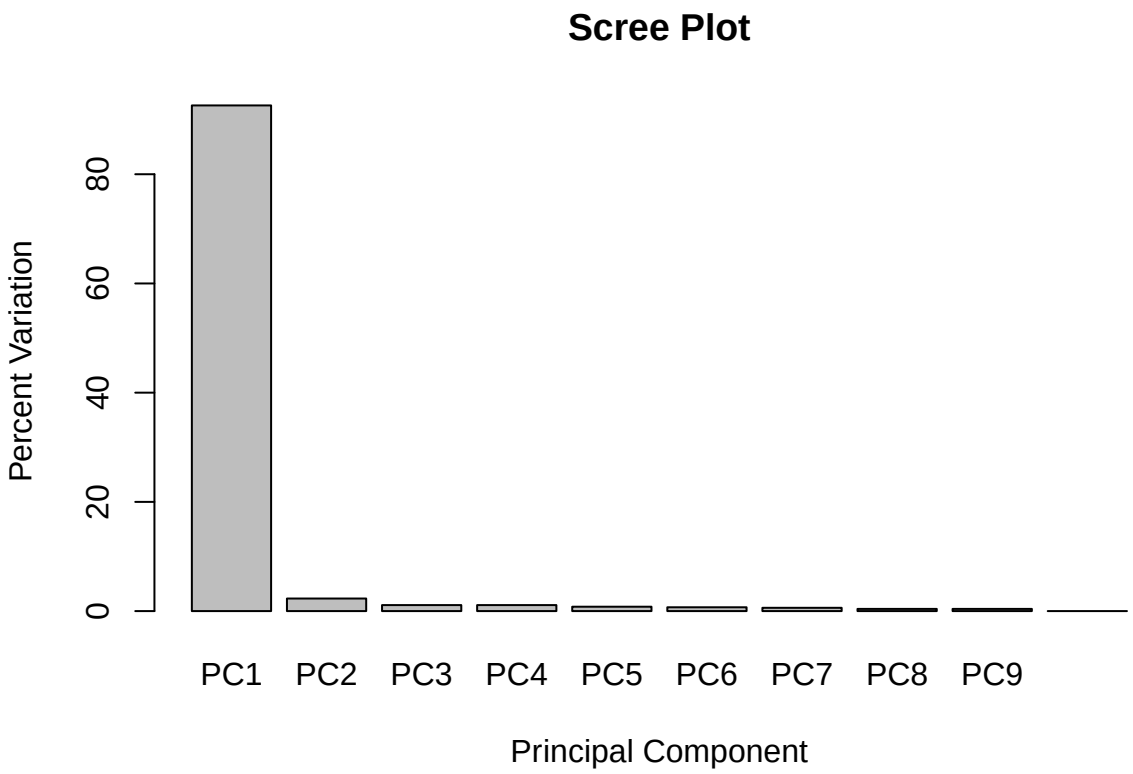
Let's make the above scree plot ourselves and in so doing explore the object returned from prcomp() a little further. We can use the square of pca$sdev, which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for:

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

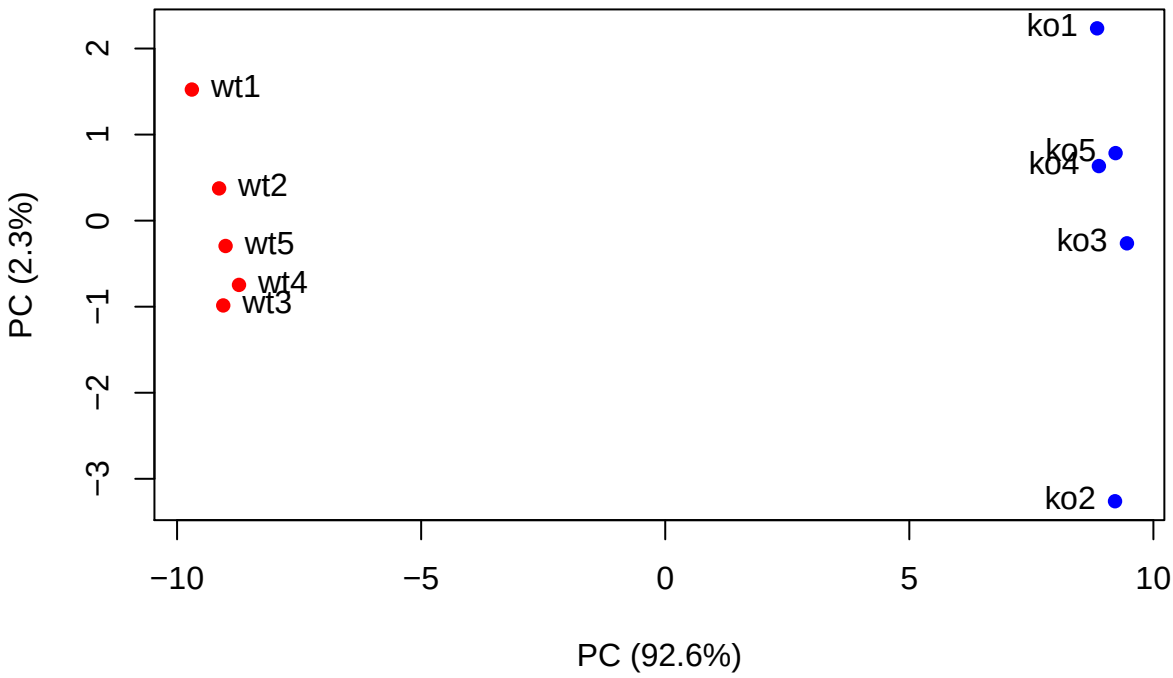
We can use this to generate our own scree-plot like this

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



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

```
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 ("PC (", pca.var.per[1], "%)"), ylab =

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

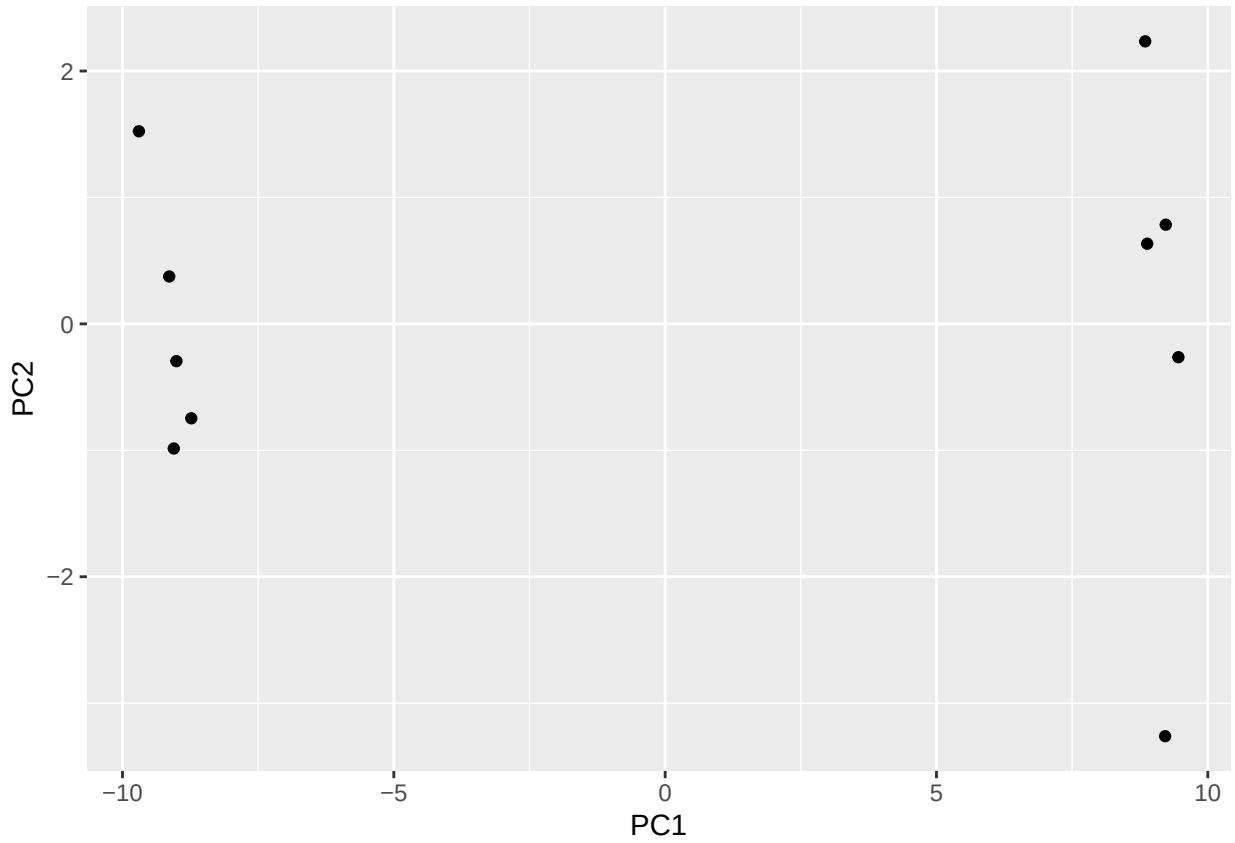


#Using ggplot > We could use the ggplot2 package here but we will first need a data.frame as input for the main ggplot() function. This data.frame will need to contain our PCA results (specifically pca$x) and additional columns for any other aesthetic mappings we will want to display. We will build this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

```
library(ggplot2)
```

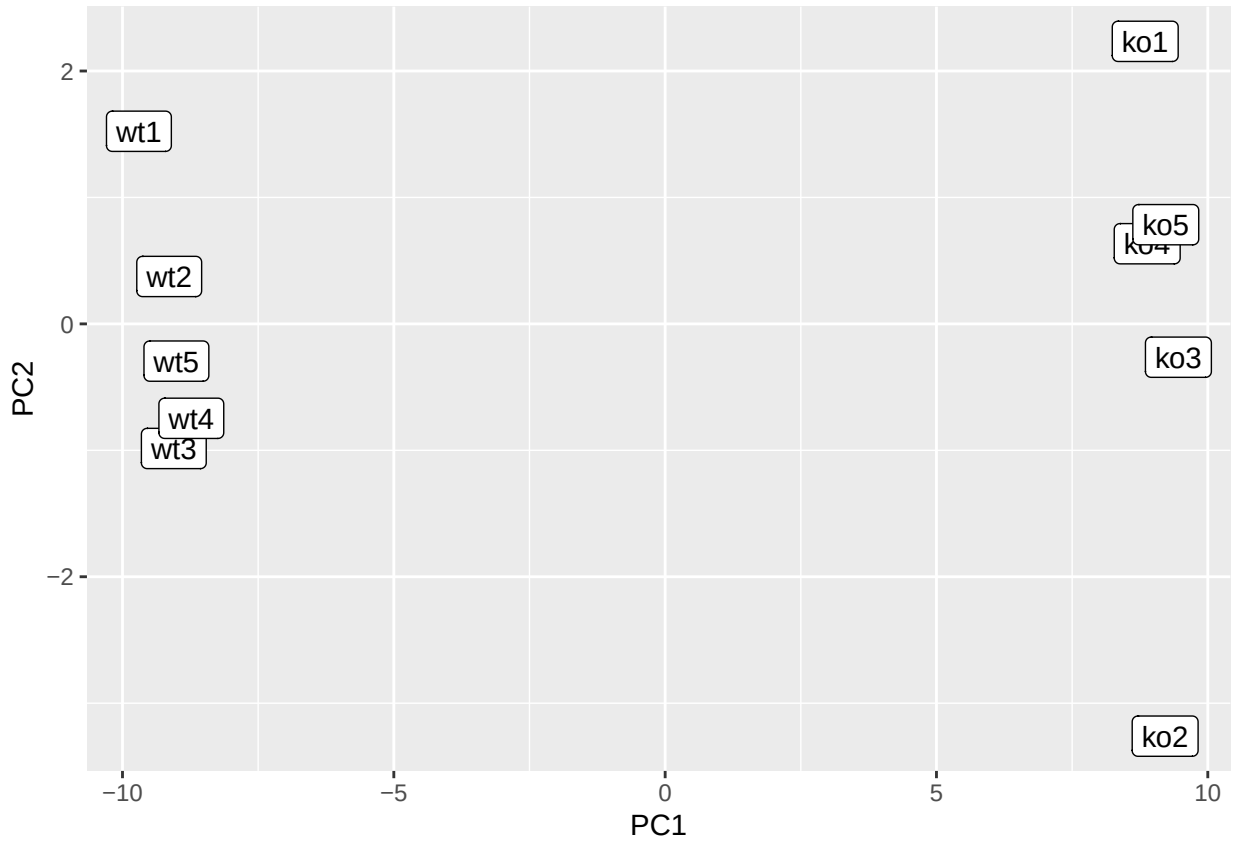
```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

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

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



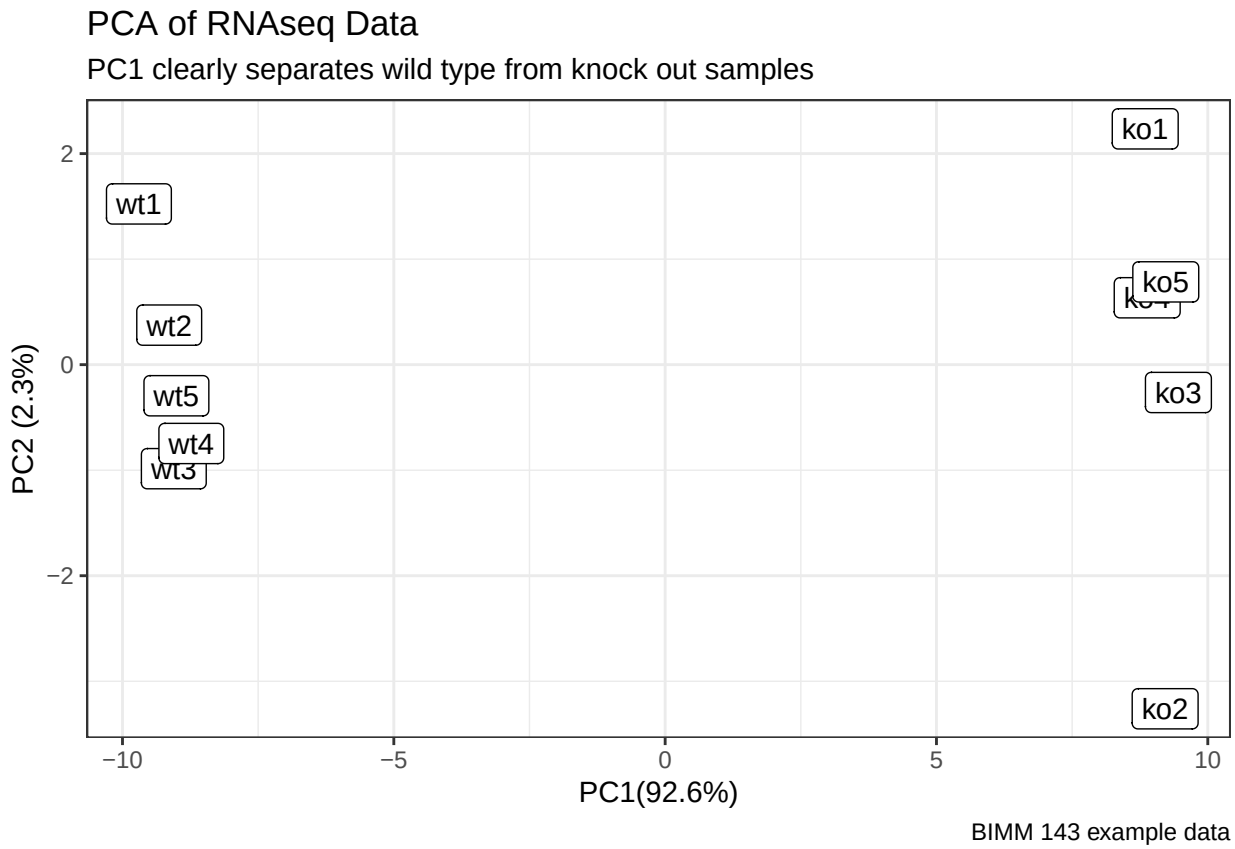
> If we want to add a condition specific color and perhaps sample label aesthetics for wild-type and knock-out samples we will need to have this information added to our data.frame:

```
# 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, condition) + geom_label(show.legend = FALSE)

p
```



And finally add some spit and polish

```
p + labs(title = "PCA of RNAseq Data", subtitle = "PC1 clearly separates wild type from knock out sampl
```



#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_scores_ranked <- sort(gene_scores, decreasing = TRUE)

## show the names of the top 10 genes
top_10_genes <- names(gene_scores_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).

```
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
```

```
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.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] ggplot2_3.3.5
##
## loaded via a namespace (and not attached):
##  [1] pillar_1.6.3     compiler_4.0.0   highr_0.9        tools_4.0.0
##  [5] digest_0.6.28    evaluate_0.14    lifecycle_1.0.1  tibble_3.1.5
##  [9] gtable_0.3.0     pkgconfig_2.0.3  rlang_0.4.11     DBI_1.1.1
## [13] yaml_2.2.1       xfun_0.29        fastmap_1.1.0    withr_2.4.2
## [17] stringr_1.4.0    dplyr_1.0.7      knitr_1.36       generics_0.1.0
## [21] vctrs_0.3.8      grid_4.0.0       tidyselect_1.1.1 glue_1.4.2
## [25] R6_2.5.1         fansi_0.5.0      rmarkdown_2.11   farver_2.1.0
## [29] purrr_0.3.4      magrittr_2.0.1   scales_1.1.1     ellipsis_0.3.2
## [33] htmltools_0.5.2  assertthat_0.2.1 colorspace_2.0-2 labeling_0.4.2
## [37] utf8_1.2.2       stringi_1.7.5    munsell_0.5.0    crayon_1.4.1
```