Machine learning 1

Hope Adams

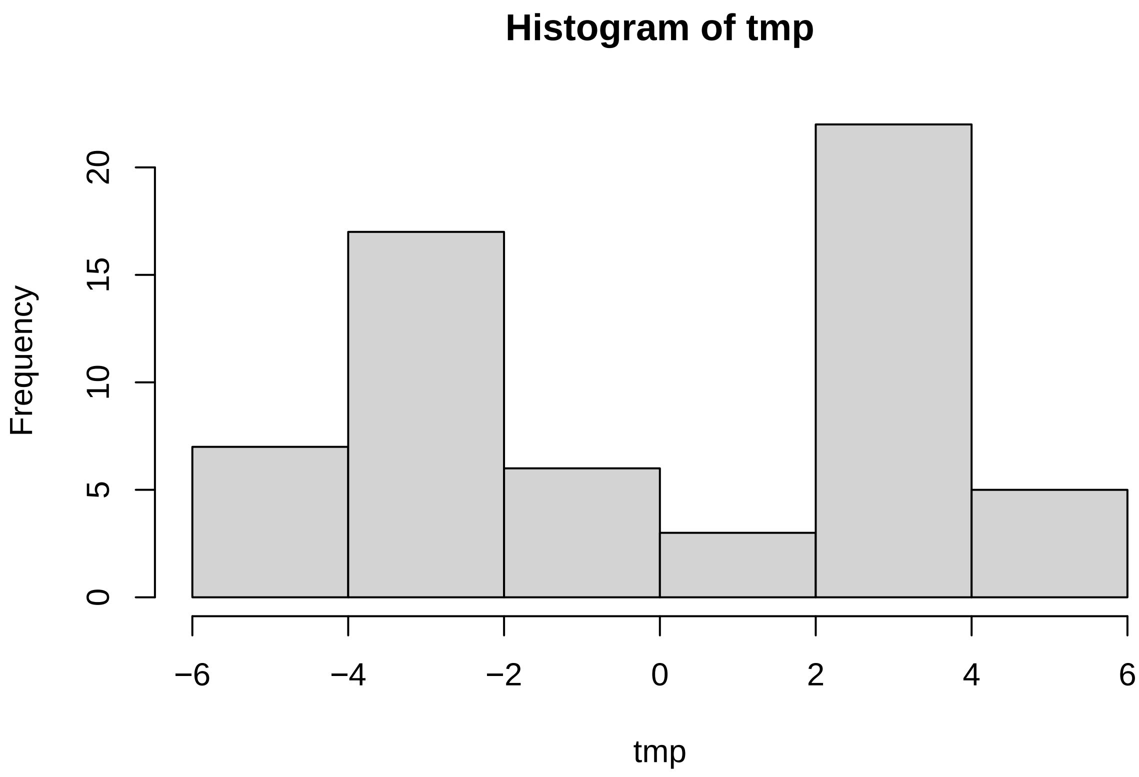
10/21/2021

Kmeans clustering

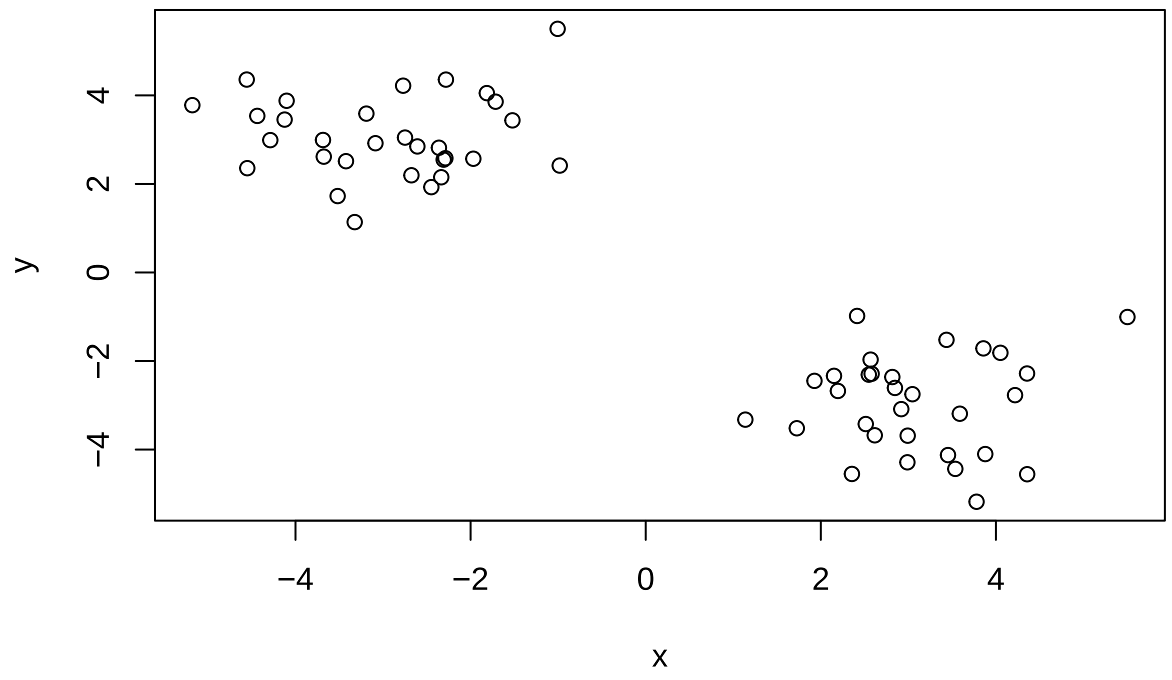
The function in base R to do Kmeans is called 'kmeans()'

First make up some new data where we know what the answer should be:

```
# rnorm gives random values centered at a certain point (30 centered at -3)
tmp <- c(rnorm(30, -3), rnorm(30, 3))
hist(tmp)
```



```
# cbind combines two vectors ?
x <- cbind(x=tmp, y=rev(tmp))
plot(x)
```



Q: Can we use kmeans() to cluster this data setting k 2 and nstart to 20? YES

```
km <- kmeans(x, centers = 2, nstart=20)
km
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.078722 -2.965943
## 2 -2.965943  3.078722
##
## Clustering vector:
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 59.52488 59.52488
##  (between_SS / total_SS =  90.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q: How many points are in each cluster?

```r
km$size
```

```
## [1] 30 30
```

There are 30 points in each cluster

Q: What 'component' of your result object details cluster assignment/membership?

```r
km$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

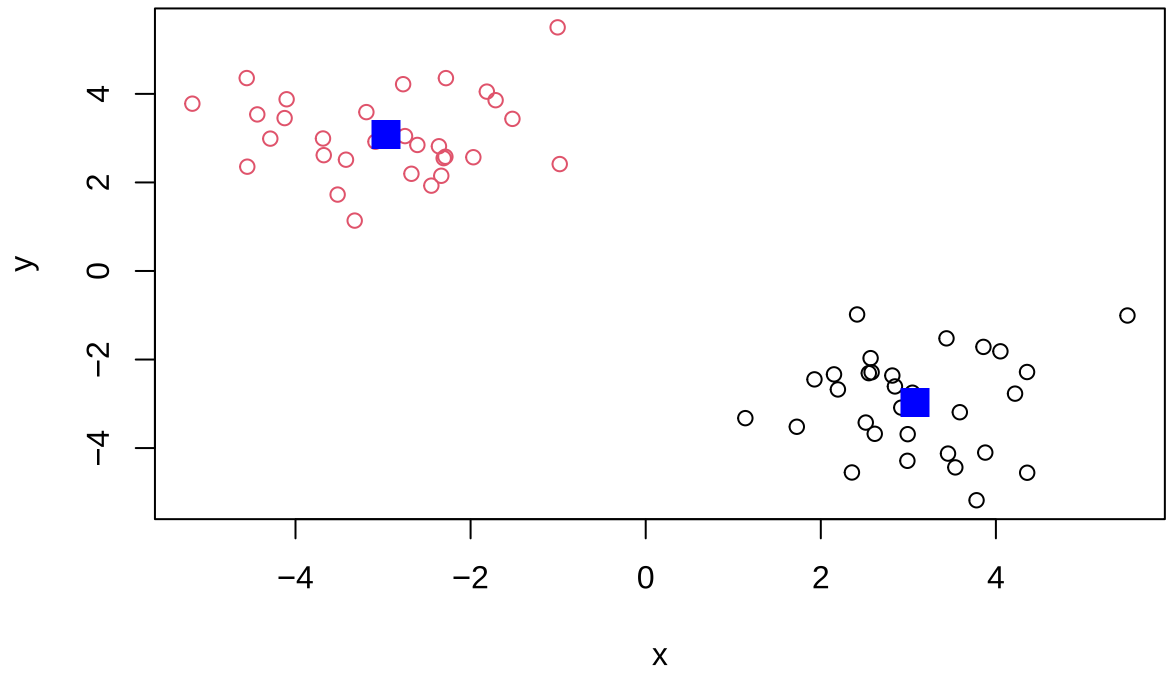
Q: What 'component' of your result object details cluster center?

```r
km$centers
```

```
##           x         y
## 1  3.078722 -2.965943
## 2 -2.965943  3.078722
```

Q: Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```r
plot(x,col=km$cluster)
points(km$centers, col="blue", pch=15, cex=2)
```



hclust

```
?hclust()
```

Hierarchical Clustering

A big limitation with k-means is that we have to tell it K (the number of clusters we want).

Analyze the same data with hclust()

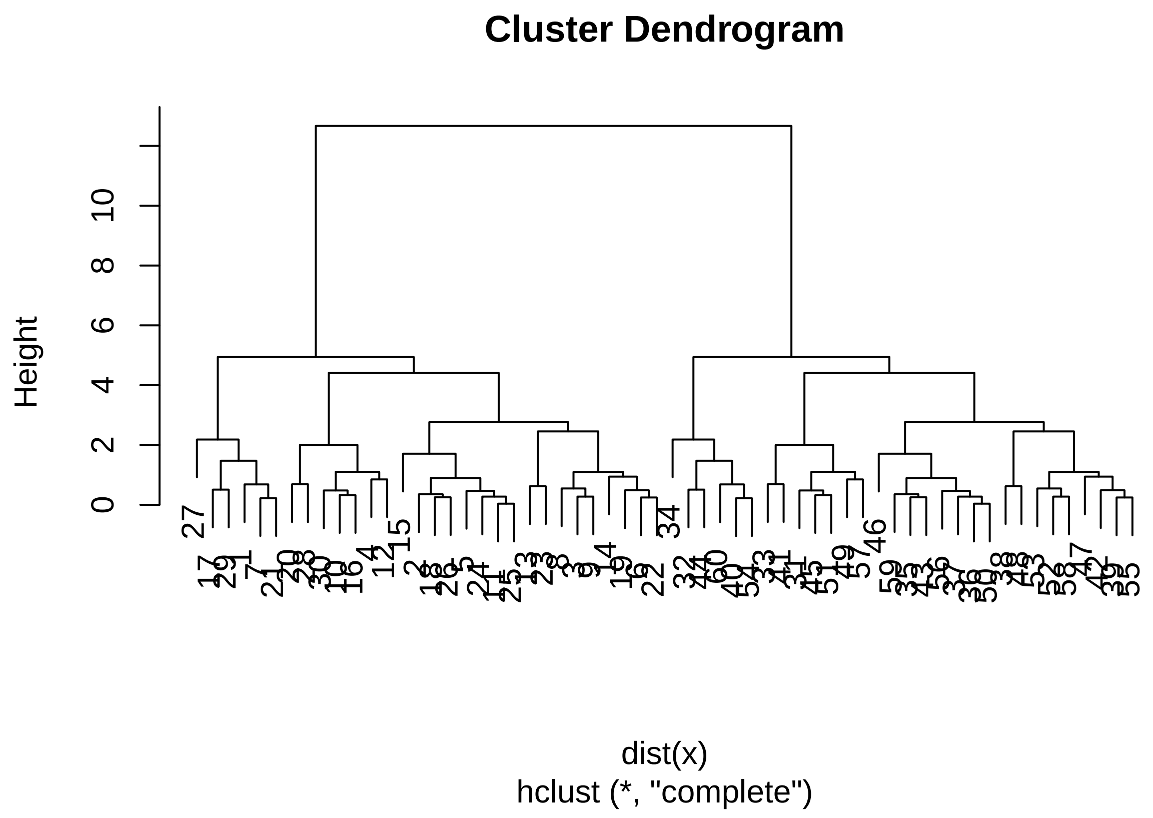
Demonstrate the use of dist(), hclust(), plot() and cutree() functions to do clustering. Generate dendrograms and return cluster assignment/membership vector…

```
hc <- hclust( dist(x))
hc
```

```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

There is a plot method for hclust result objects. Let's see it

```
plot(hc)
```



To get our cluster membership vector we have to do a little bit of work. We have to "cut" the tree where we want and think it makes sense. We can cut the tree at a specific height (ex: cut the tree at height 8) using the 'cutree()' function.

```
cutree(hc, h=6)
```

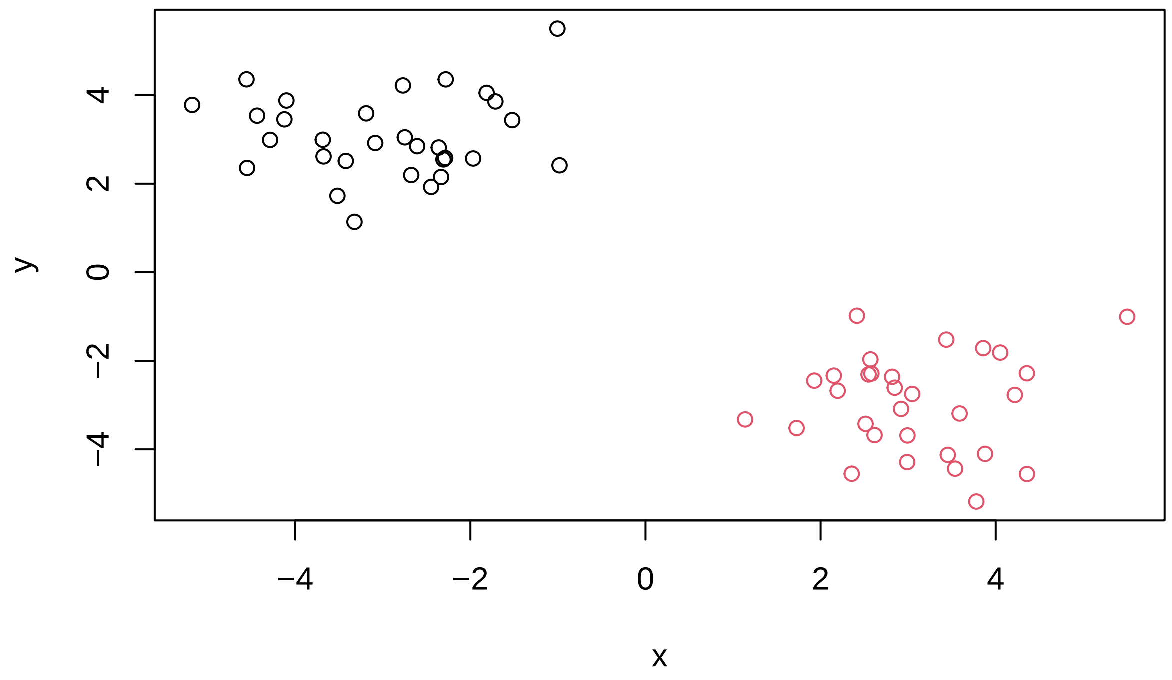
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

We can also cut the tree by telling it to cut by a certain number of groups (look at tree first then decide how many clusters we want) by writing k = # of clusters

```
grps <- cutree(hc, k=2)
```

Make our results plot

```
plot(x, col=grps)
```



Start Principal Component Analysis of UK Food Data

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

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

```
ncol(x)
```

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

Look at data imported:

```
View(x)
```

Have 5 columns when we should have 4. We need to take "x" (first column) out of column numbering

```
rownames(x) <- x[,1]
# tell x to be everything but the first column
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
```

Issue: if we keep running that code it'll take out the first column every time we rerun.

Try a different way to take out the first column name:

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.name=1)
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
## 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
```

```
nrow(x)
```

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

```
ncol(x)
```

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

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

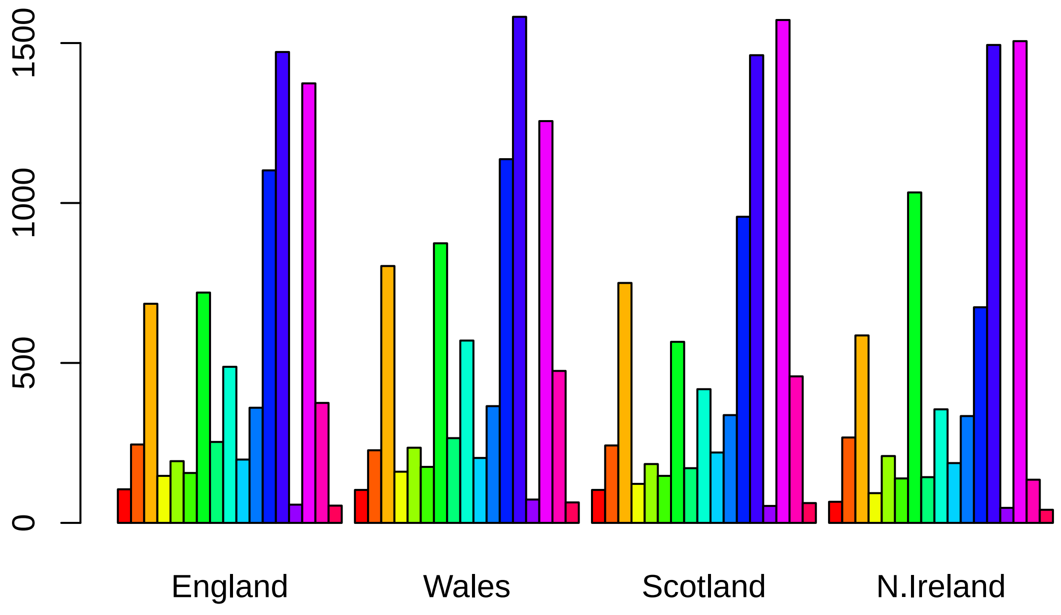
There are 17 rows and 4 columns

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?

The second option is better because it tells R to ignore the first column without taking out the first column’s name every time its run again.

Look at bar graph and see why it’s not helpful:

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

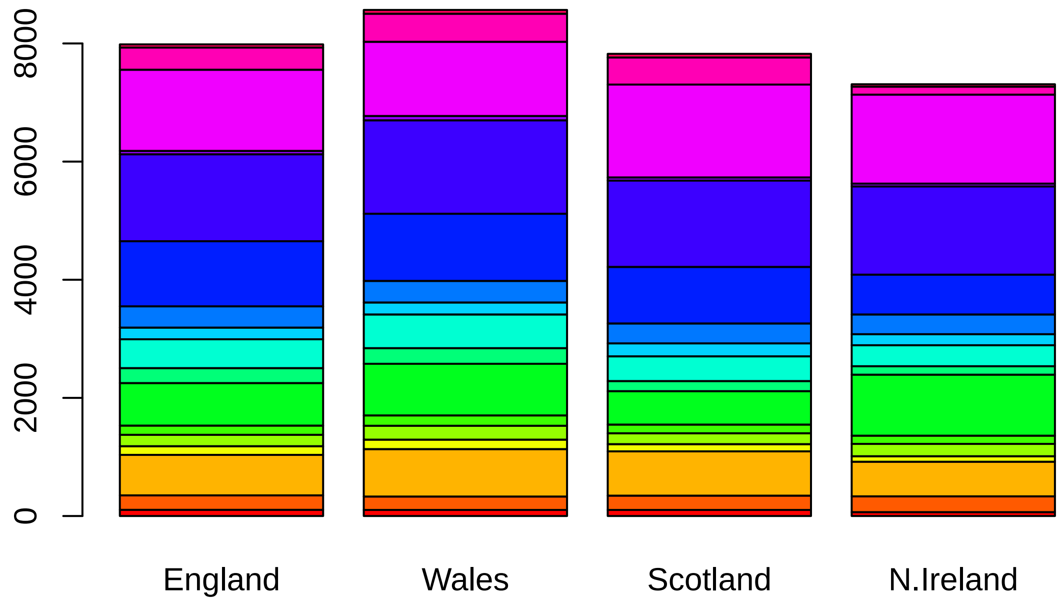


This doesn’t tell us the variation between the different countries, the rainbow isn’t telling any information besides the different food categories.

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

We can stack the graph by editing the plot above by making the “beside” argument false:

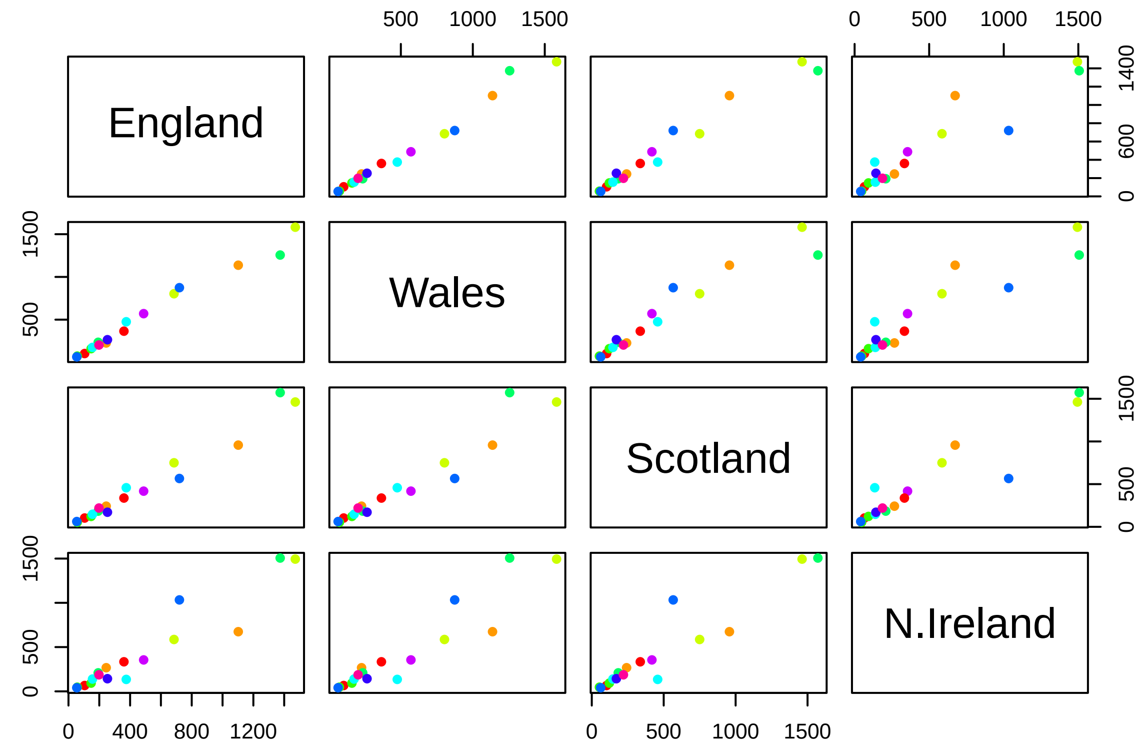
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



This is still unhelpful because it's only showing us that some of the countries eat more of one food than the others.

We can make a new graph:

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



# 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? This new graph is comparing two countries at a time depending on which graph you are looking at. The graph to the right of "England" and above "Wales" written in the box is comparing the two countries and their eating habits. The graph to the right of that graph is comparing "England" and "Scotland".

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

We can see that N. Ireland is has a food group in blue that they tend to eat much less than the rest of the countries.

Now we can try to use PCA to display our data more effectively

The main function in base R for PCA is 'prcomp()' This wants the transpose of our 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
```

"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."

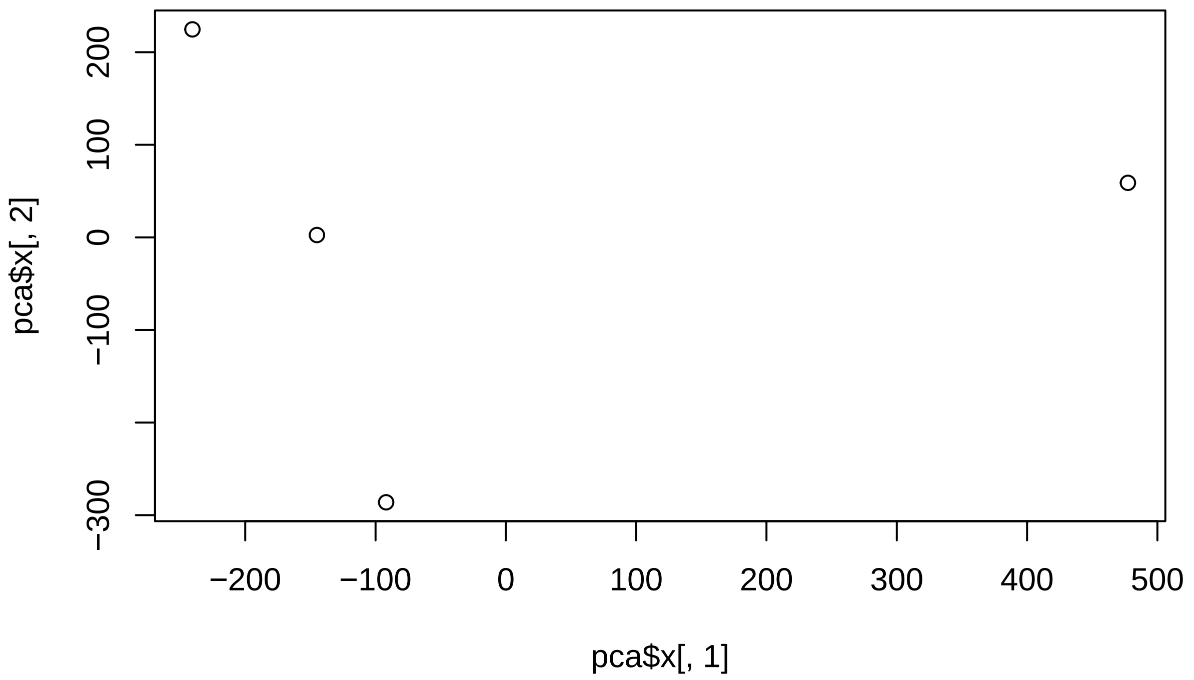
Since 96.5% of the data spread can be explained by PC1 and PC2 we can focus on those two. What can we do with pca:

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

```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

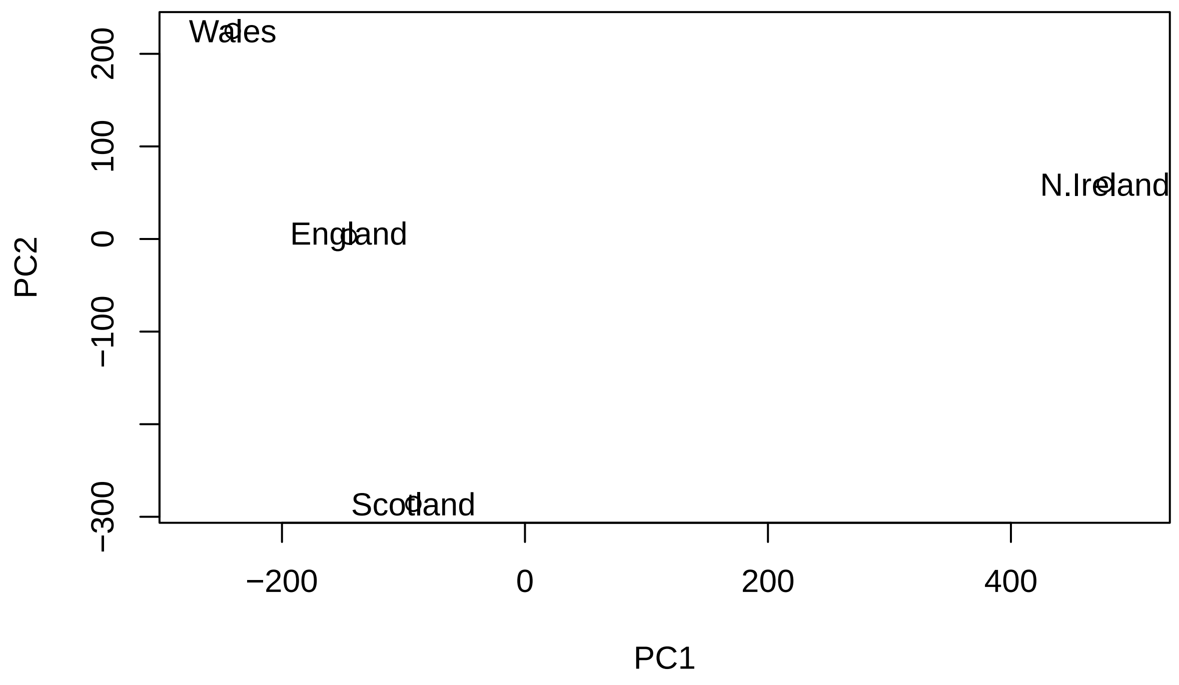
Lets use 'x'

```
plot(pca$x[,1], pca$x[,2])
```



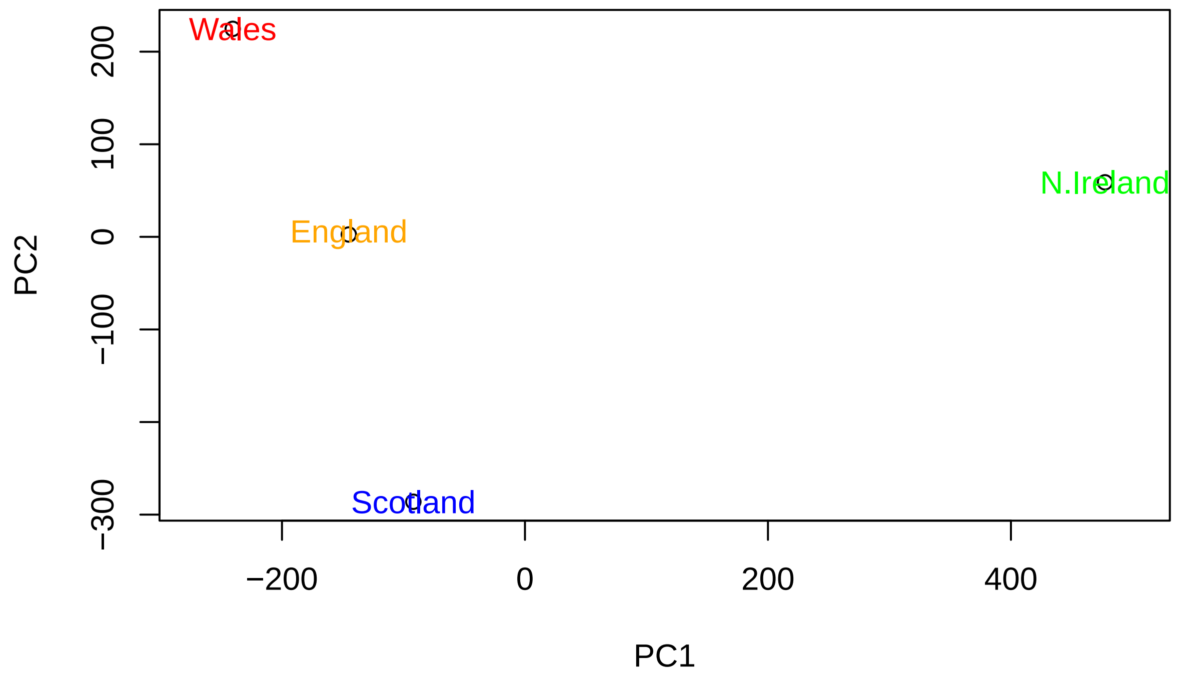
Add labels:

```
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))
text(pca$x[,1], pca$x[,2], colnames(x), col= c("orange", "red", "blue", "green"))
```



Next we want to find variation:

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

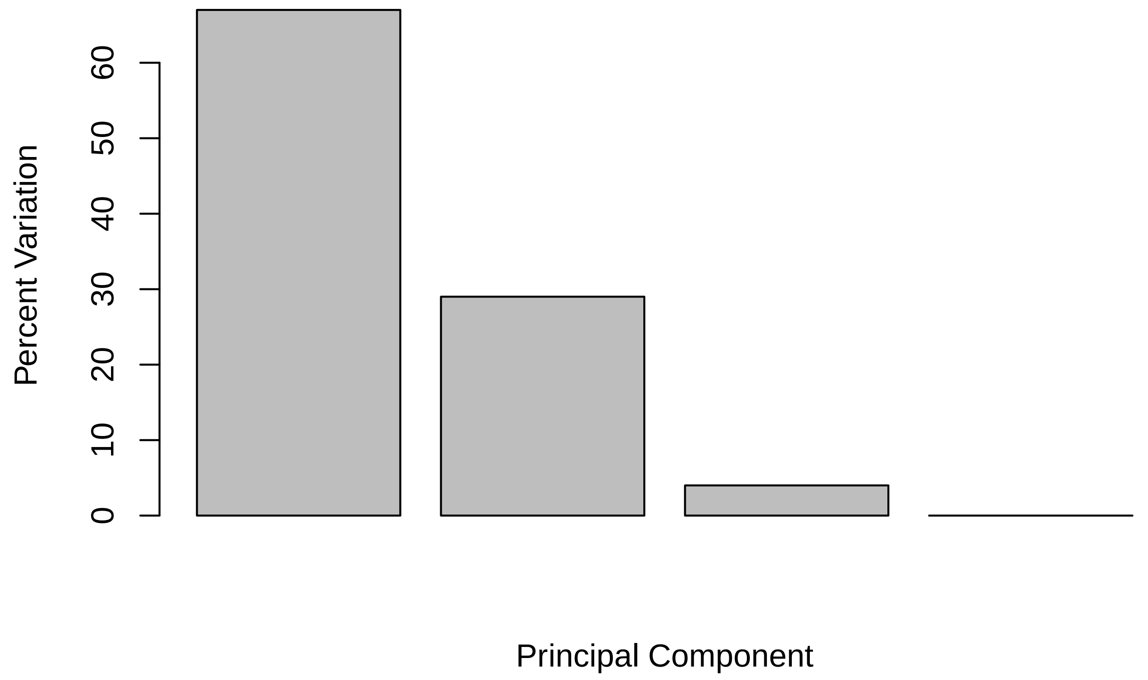
```
## [1] 67 29  4  0
```

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

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

Now we can summarize into a bar graph:

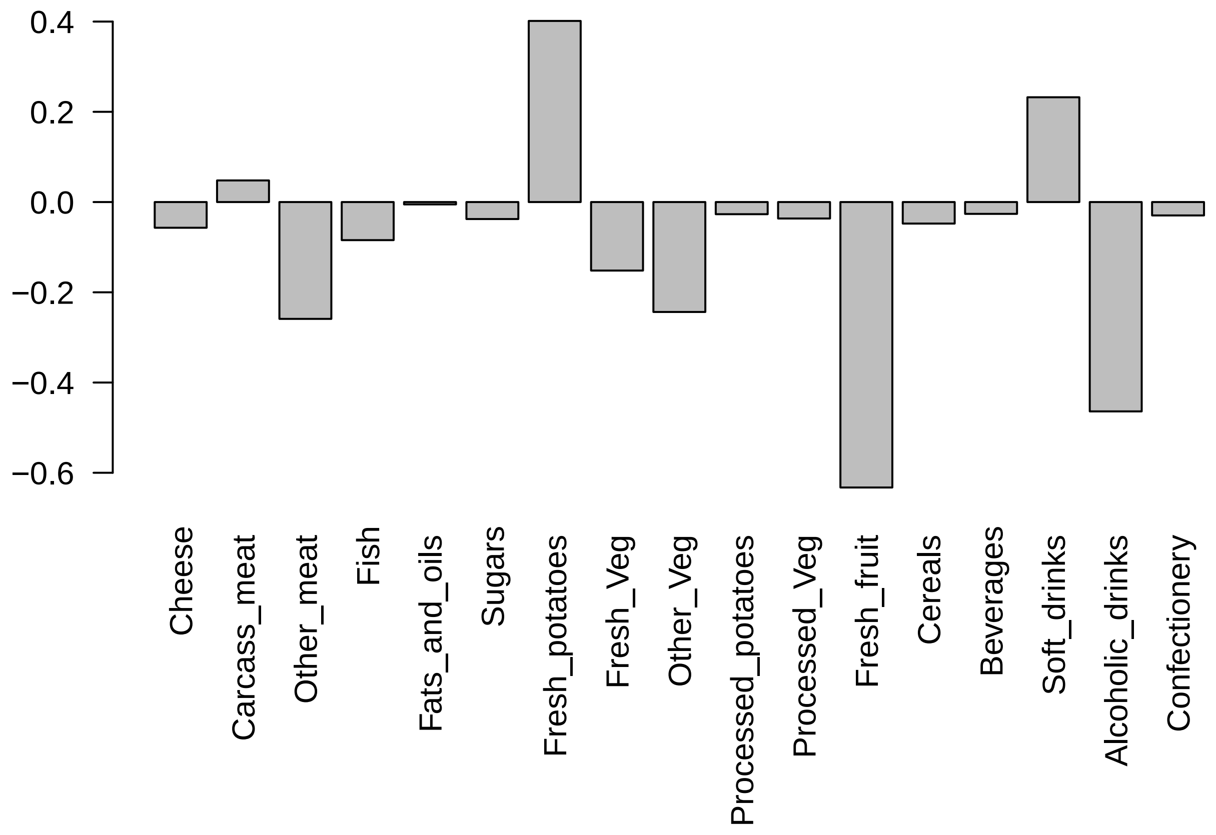
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



We see that PC1 accounts for most of the variation followed by PC2.

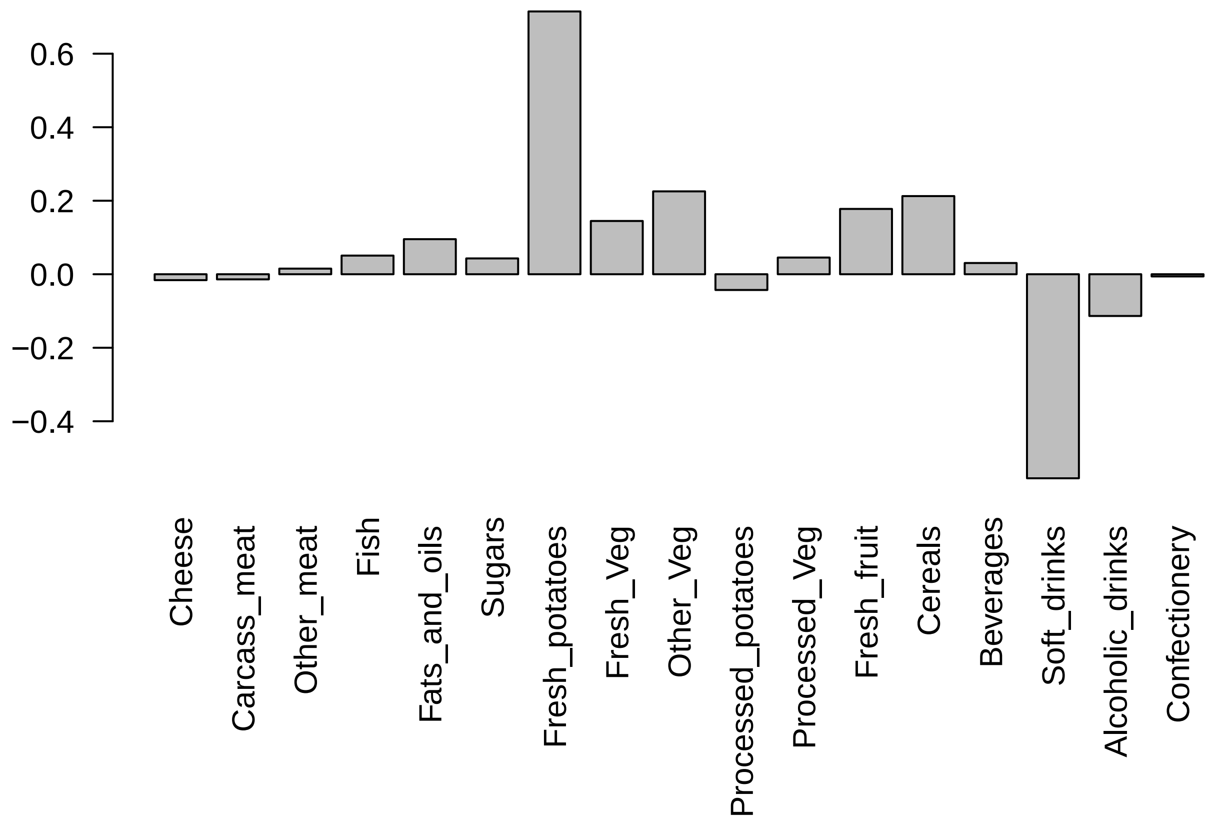
If we want to consider the variation within each of the rows specifically by looking at only PC1:

```
## 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 )
```



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

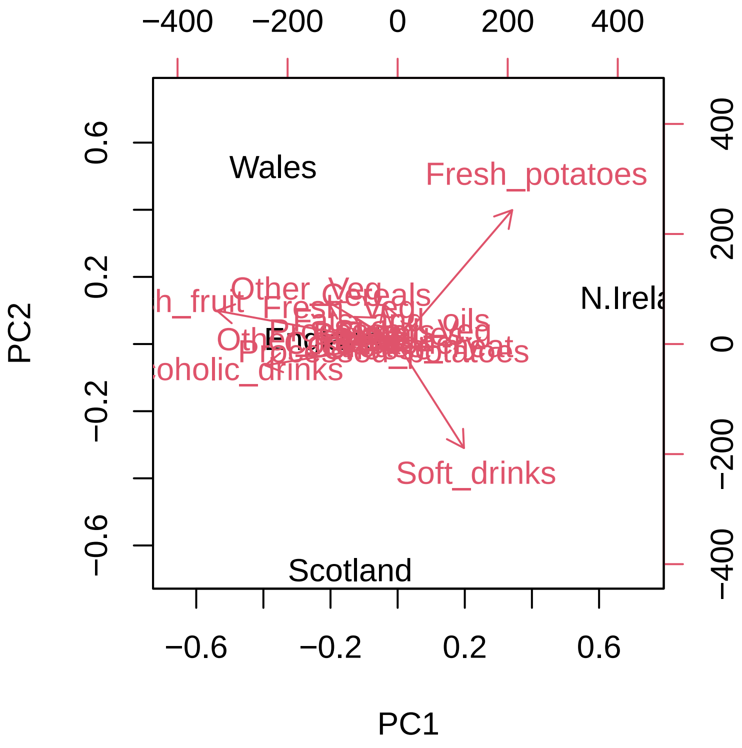
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



PCA2 tells us the factor that explains the next most variation.

Another way to run this data is with a bioplot:

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



Another way to graph PCA data is with RNA-seq

```
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
```

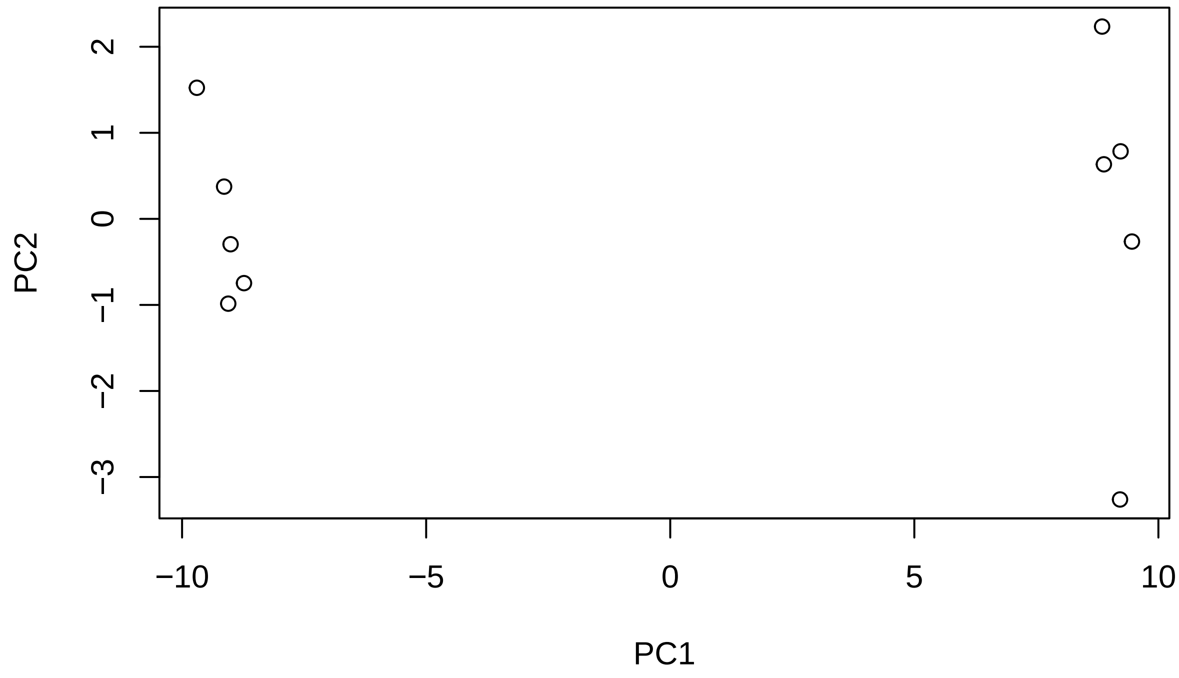
```
ncol(rna.data)
```

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

There are 100 genes and 10 samples per gene for wiltype and knockout.

Use PCA to plot results:

```r
## 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")
```



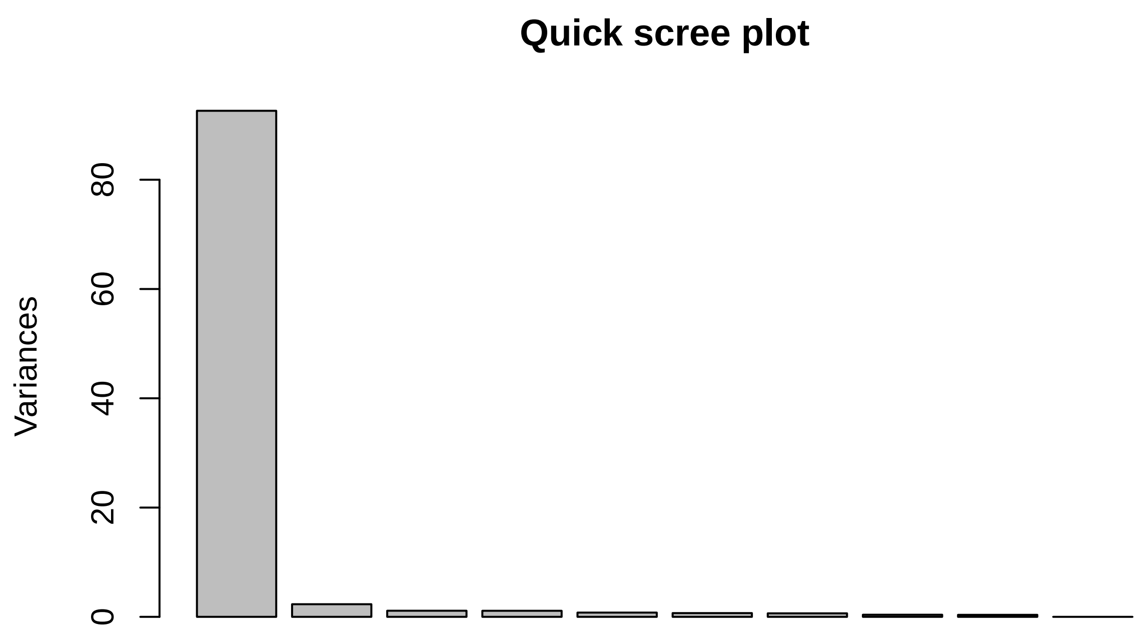
Summarize data:

```r
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
##                            PC8     PC9      PC10
## Standard deviation     0.62065 0.60342 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

Plot PCA:

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



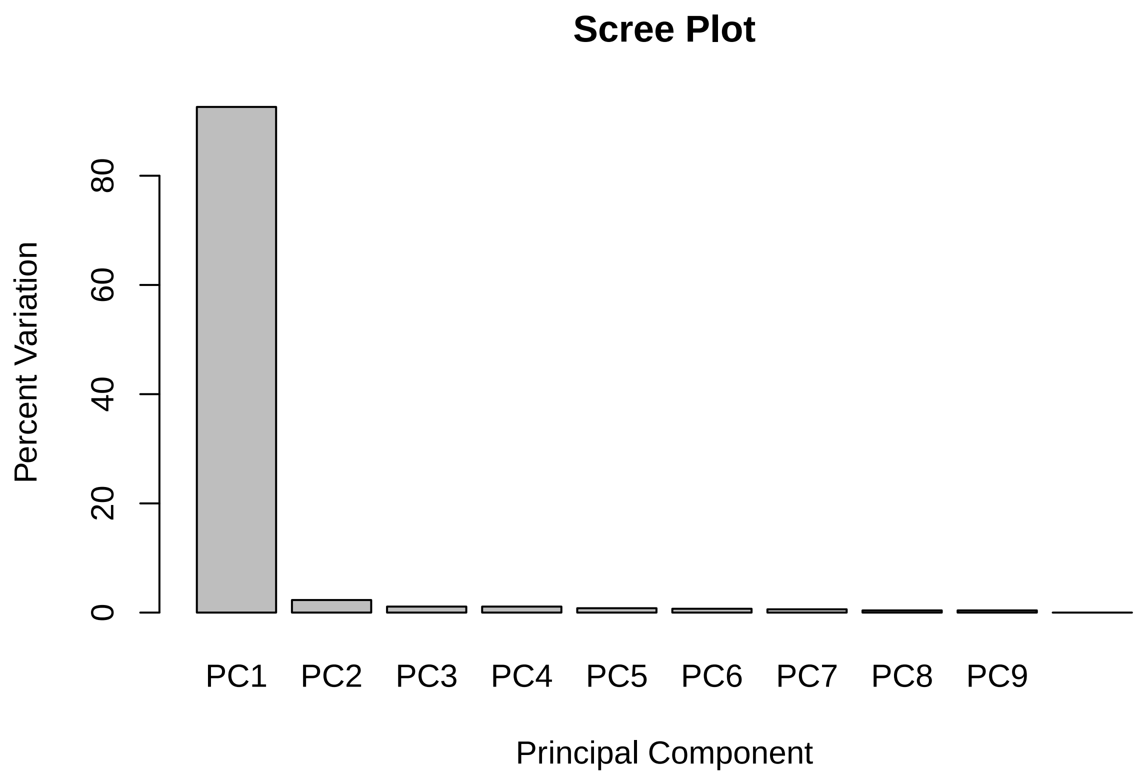
Find variance for the plot:

```r
## 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
```

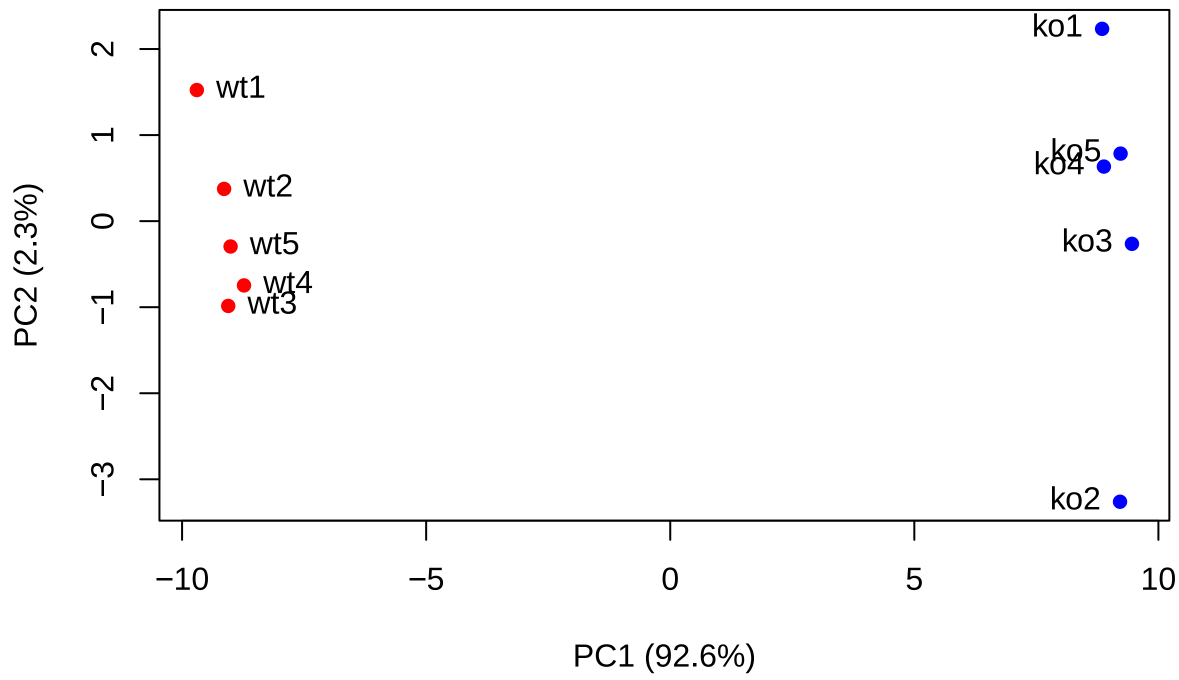
Generate scree plot again with variation:

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



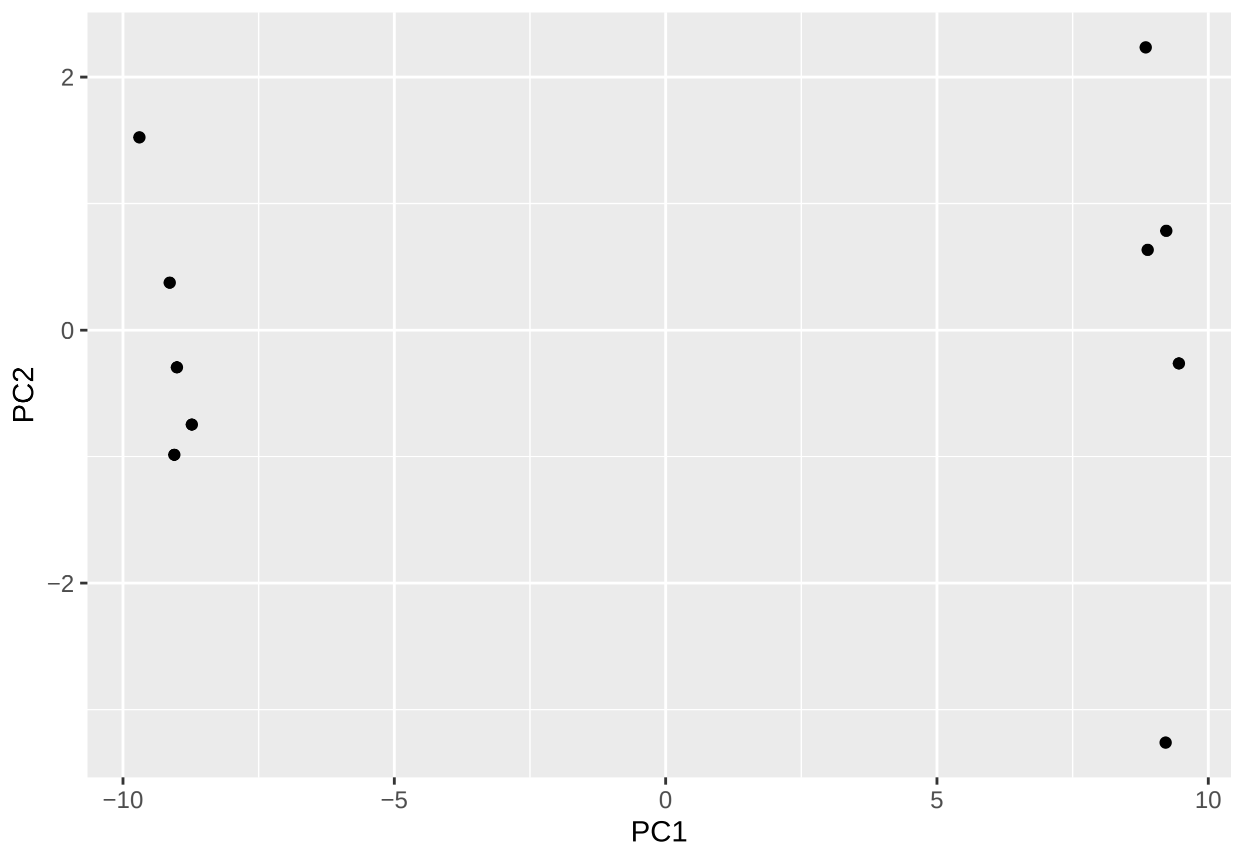
Make scree plot easier to understand:

```
## 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)))
```



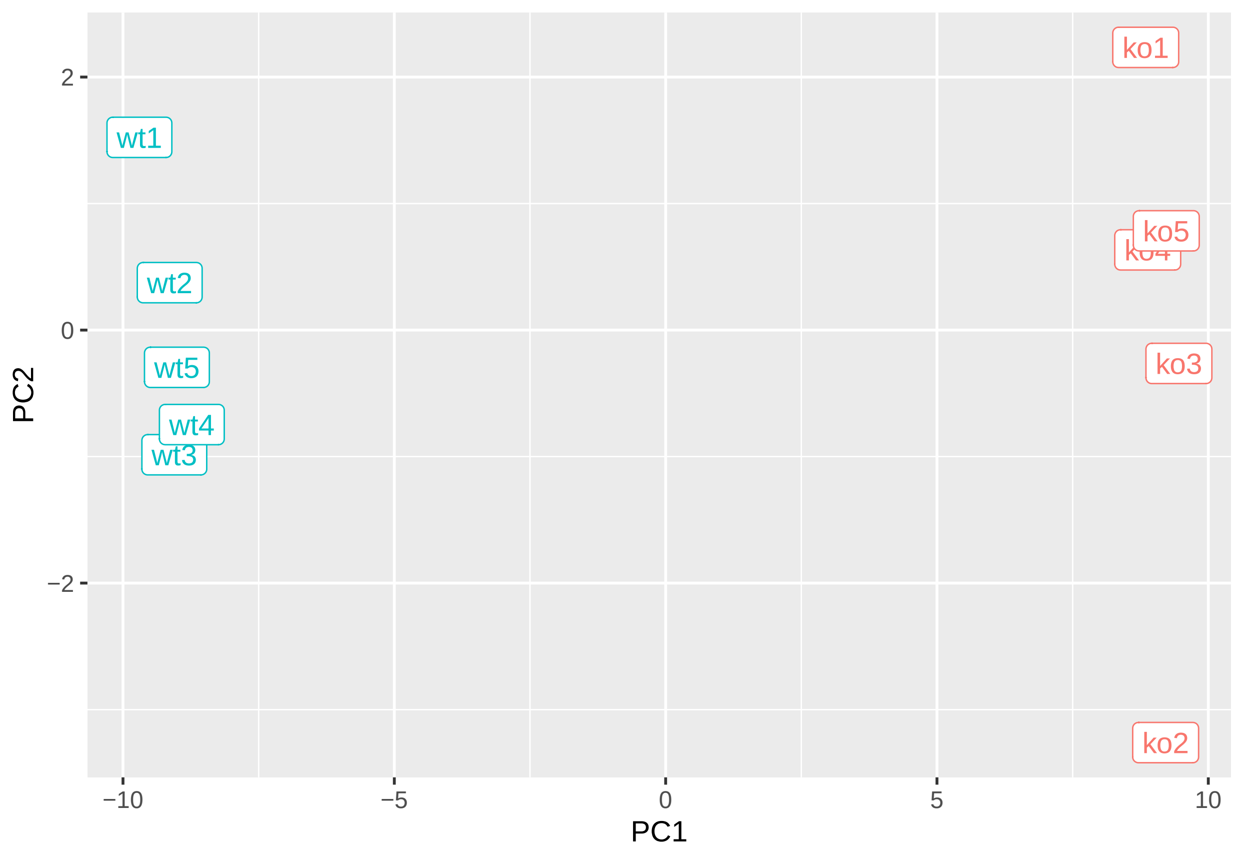
Try using ggplot to do this:

```
library(ggplot2)

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

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



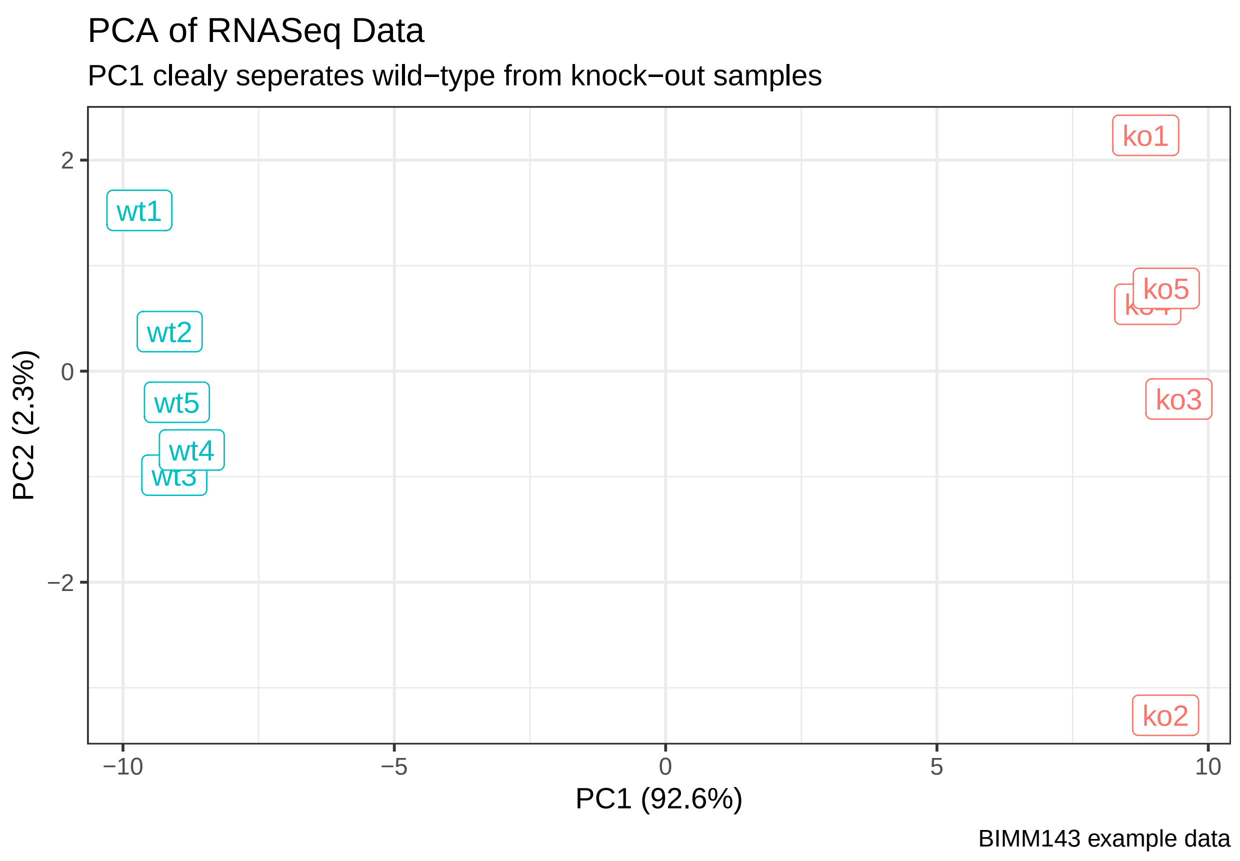
Specify colors for each sample type:

```r
# 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
```



Add more to titles:

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


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
PC2 (2.3%)
PC1 (92.6%)
BIMM143 example data
2
0
−2
−10
−5
0
5
10
wt1
wt2
wt5
wt4
wt3
ko1
ko5
ko4
ko3
ko2