Class 07: machine learning 1

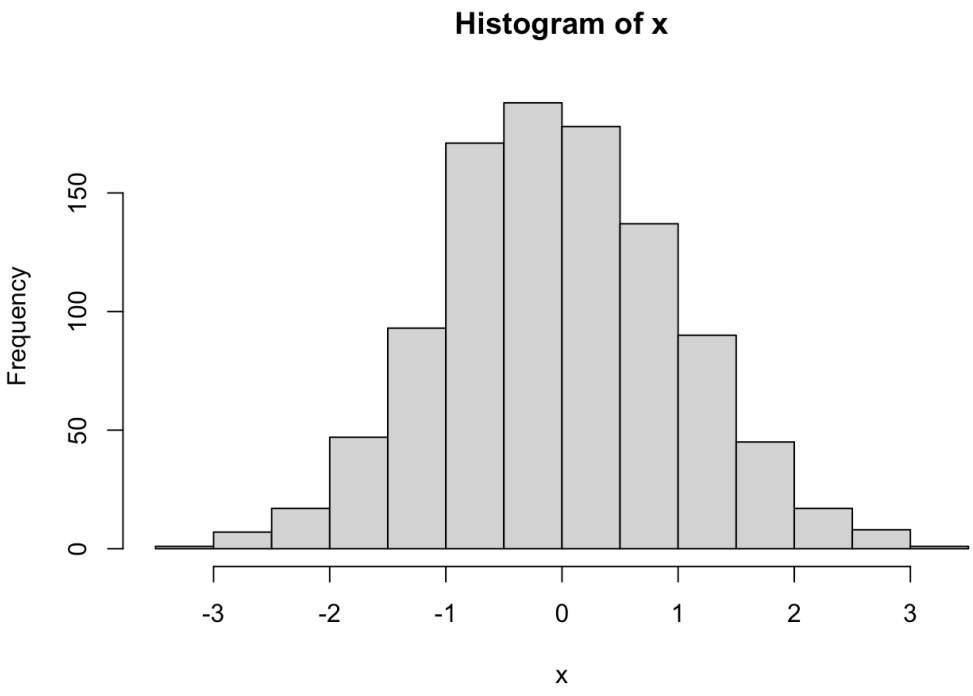
AUTHOR
Lisa Chen (PID: A17082974)

##clustering methods The broad goal here is to find groupings (clusters) in your input data.

Kmeans

1st let's make up some data to cluster

```
x <- rnorm(1000)
hist(x)
```



make a vector of length 60 with 30 points centered at -3 and 30 points centered at +3

```
tmp <-c(rnorm(30, mean=-3), rnorm(30, mean=3))
tmp
```

```
 [1] -2.9518996 -0.7827734 -1.7318752 -3.1115964 -1.8610008
-2.2718214
 [7] -3.9678016 -3.8656070 -3.9810432 -2.3832023 -2.6293429
-1.8678918
[13] -4.1068040 -3.4800152 -2.4794825 -3.6767563 -4.7963726
-3.4123336
[19] -3.2444365 -3.5443769 -3.0204870 -3.0085322 -2.9176768
-1.9424662
[25] -3.5753728 -4.1667347 -1.2990372 -1.5418209 -2.1225637
-4.2290965
[31]  3.7922418  4.9277223  2.9082018  2.5006931  3.6771648
3.7659838
[37]  1.4497013  3.1369309  2.9501651  4.0875572  3.6604628
2.6377046
[43]  3.6506235  2.8175113  2.9379289  3.5623454  3.0398269
3.7737301
[49]  3.4970648  3.7059298  3.3818532  1.2477756  3.6874507
2.1120629
[55]  2.1779770  3.6603735  0.9791801  2.8033344  1.7976011
2.7429565
```

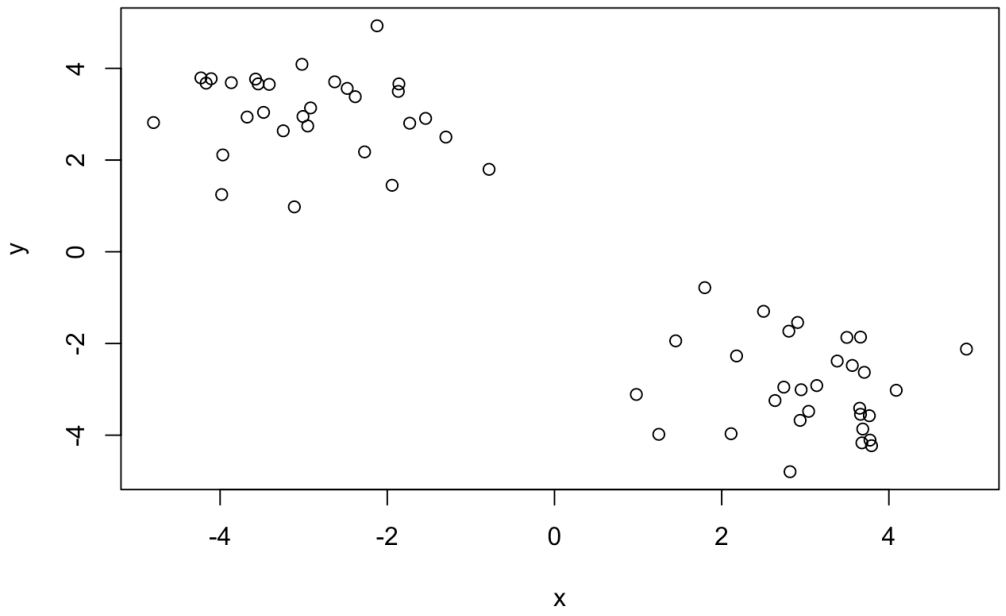
create x and y data set with 2 groups of points.

```
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
               x          y
 [1,] -2.9518996  2.7429565
 [2,] -0.7827734  1.7976011
 [3,] -1.7318752  2.8033344
 [4,] -3.1115964  0.9791801
 [5,] -1.8610008  3.6603735
 [6,] -2.2718214  2.1779770
 [7,] -3.9678016  2.1120629
 [8,] -3.8656070  3.6874507
 [9,] -3.9810432  1.2477756
[10,] -2.3832023  3.3818532
[11,] -2.6293429  3.7059298
[12,] -1.8678918  3.4970648
[13,] -4.1068040  3.7737301
[14,] -3.4800152  3.0398269
[15,] -2.4794825  3.5623454
[16,] -3.6767563  2.9379289
[17,] -4.7963726  2.8175113
```

```
[18,] -3.4123336  3.6506235
[19,] -3.2444365  2.6377046
[20,] -3.5443769  3.6604628
[21,] -3.0204870  4.0875572
[22,] -3.0085322  2.9501651
[23,] -2.9176768  3.1369309
[24,] -1.9424662  1.4497013
[25,] -3.5753728  3.7659838
[26,] -4.1667347  3.6771648
[27,] -1.2990372  2.5006931
[28,] -1.5418209  2.9082018
[29,] -2.1225637  4.9277223
[30,] -4.2290965  3.7922418
[31,]  3.7922418 -4.2290965
[32,]  4.9277223 -2.1225637
[33,]  2.9082018 -1.5418209
[34,]  2.5006931 -1.2990372
[35,]  3.6771648 -4.1667347
[36,]  3.7659838 -3.5753728
[37,]  1.4497013 -1.9424662
[38,]  3.1369309 -2.9176768
[39,]  2.9501651 -3.0085322
[40,]  4.0875572 -3.0204870
[41,]  3.6604628 -3.5443769
[42,]  2.6377046 -3.2444365
[43,]  3.6506235 -3.4123336
[44,]  2.8175113 -4.7963726
[45,]  2.9379289 -3.6767563
[46,]  3.5623454 -2.4794825
[47,]  3.0398269 -3.4800152
[48,]  3.7737301 -4.1068040
[49,]  3.4970648 -1.8678918
[50,]  3.7059298 -2.6293429
[51,]  3.3818532 -2.3832023
[52,]  1.2477756 -3.9810432
[53,]  3.6874507 -3.8656070
[54,]  2.1120629 -3.9678016
[55,]  2.1779770 -2.2718214
[56,]  3.6603735 -1.8610008
[57,]  0.9791801 -3.1115964
[58,]  2.8033344 -1.7318752
[59,]  1.7976011 -0.7827734
[60,]  2.7429565 -2.9518996
```

```
plot(x)
```




```
k <- kmeans(x, centers=2)
k
```

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1  3.035669 -2.932341
2 -2.932341  3.035669

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 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] 51.86166 51.86166
 (between_SS / total_SS =  91.2 %)

Available components:
```

```
[1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

> Q. from your result object `k` how many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

> Q. what "component" of your result object details the cluster membership?

```
k$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
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
```

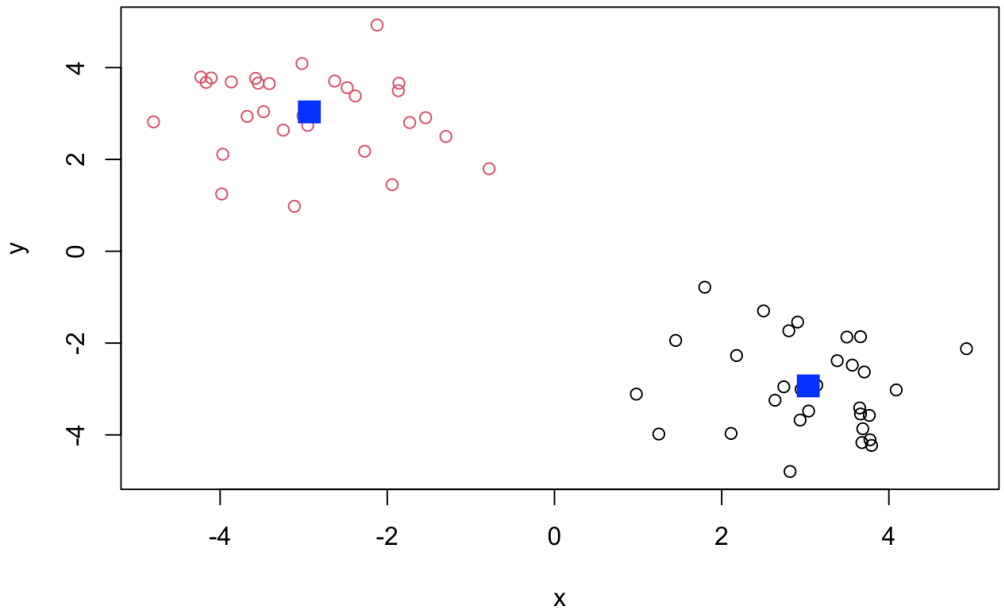
> Q. cluster centers?

```
k$centers
```

```
          x         y
1  3.035669 -2.932341
2 -2.932341  3.035669
```

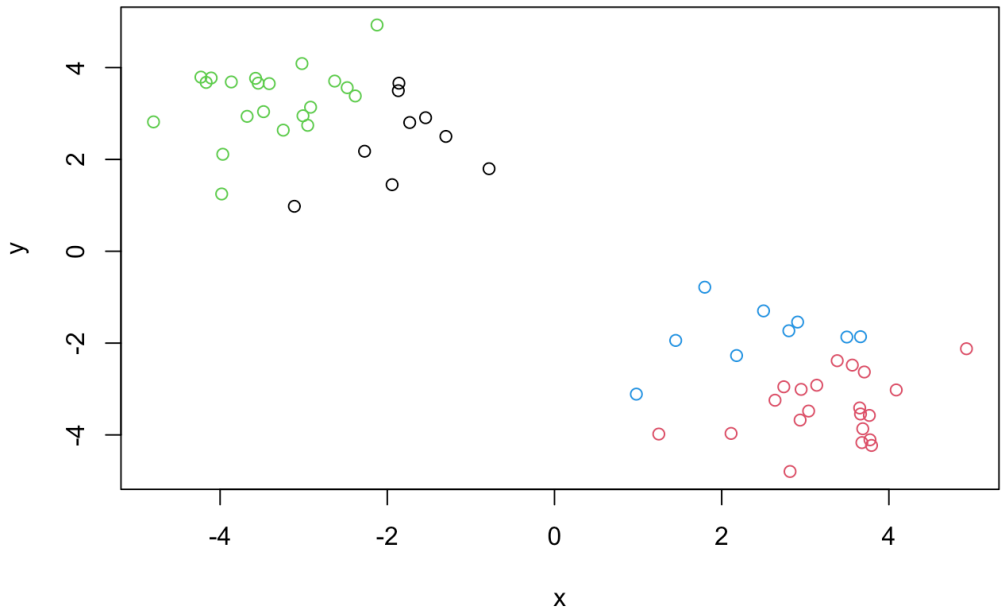
> Q. Plot of our clustering results

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



we can cluster into 4 groups

```
# kmeans
k4 <- kmeans(x, centers=4)
# plot results
plot(x, col=k4$cluster)
```



A big limitation of kmeans is that it does what you ask even if you ask for silly clusters.

Hierarchical Clustering

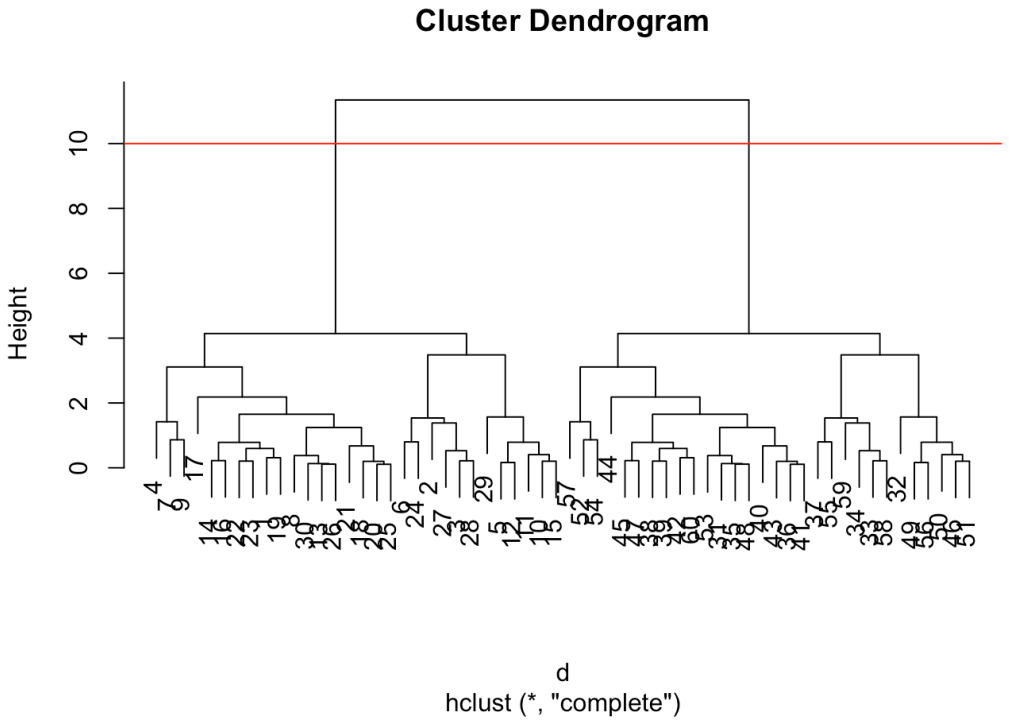
The main base R function for Hierarchical Clustering is `hclust()` unlike `kmeans()` you can not just pass it your data as input. You first need to calculate a distance matrix.

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

```
Call:
hclust(d = d)

Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

Use `plot()` to view results.

```
plot(hc)
abline(h=10, col="red")
```



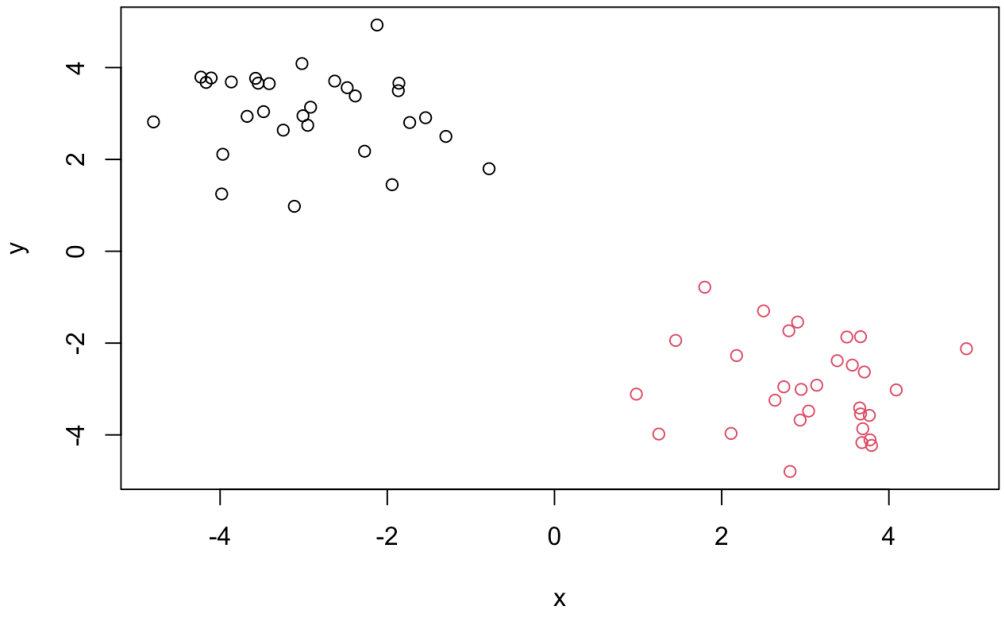
To make the "cut" and get our cluster membership vector, we can use the `cutree()` function.

```
grps <- cutree(hc, h=10)
grps
```

```
 [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 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 2 2
```

Make a plot of our data colored by hclust results

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



Principal Component Analysis (PCA)

Here we will do Principal Component Analysis (PCA) on some food

```
url <- "https://tinyurl.com/UK-foods"
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

```
#rownames(x) <- x[,1]
#x <- x[, -1]
#x
```

> Q1. How many rows and columns are in your new data frame named

> x? What R functions could you use to answer this questions?

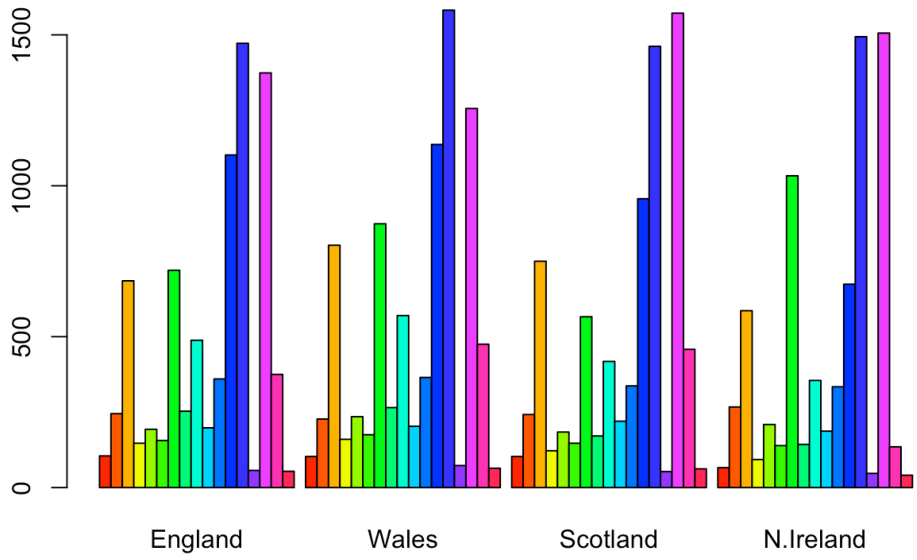
```
dim(x)
```

```
[1] 17  4
```

> 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?

Second approach because if you run the first code multiple times, it keeps eliminating the first column.

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



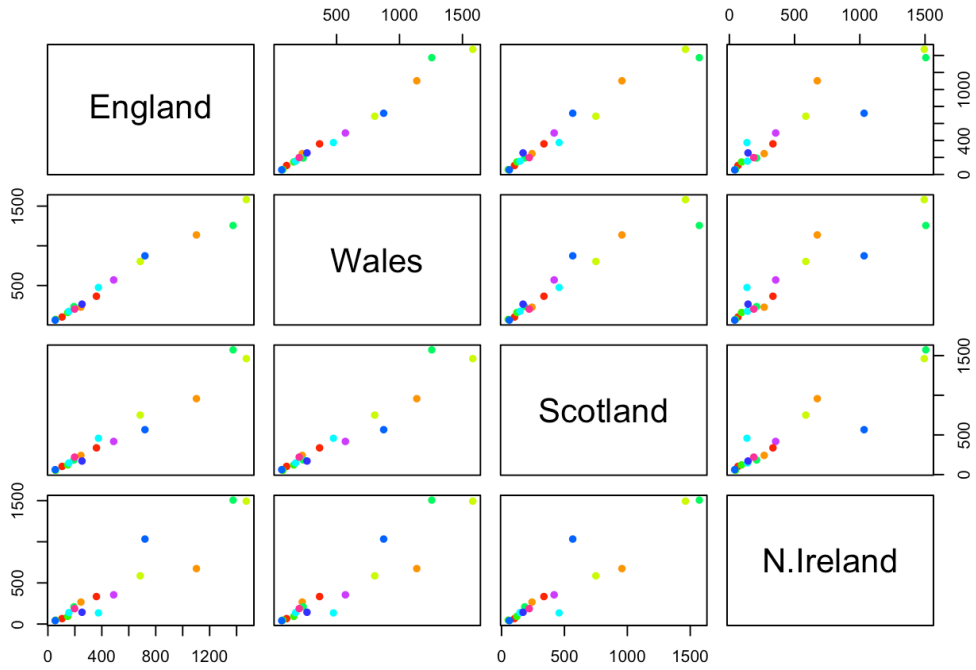
> Q3: Changing what optional argument in the above barplot() function results in the following plot?

You can change the Beside Argument.

> 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)
```



PCA to the rescue

the main "base" R function for PCA is called `prcomp()`. Here we need to take the transpose of our input

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

```
Importance of components:
                            PC1      PC2      PC3          PC4
Standard deviation     324.1502 212.7478 73.87622 2.921e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

> Q. how much variance is captured in 2 PCs

96.5%

to make our main “PC score plot” (aka “PC1 vs PC2 plot”, or “PC plot”, or “ordination plot”)

```
attributes(pca)
```

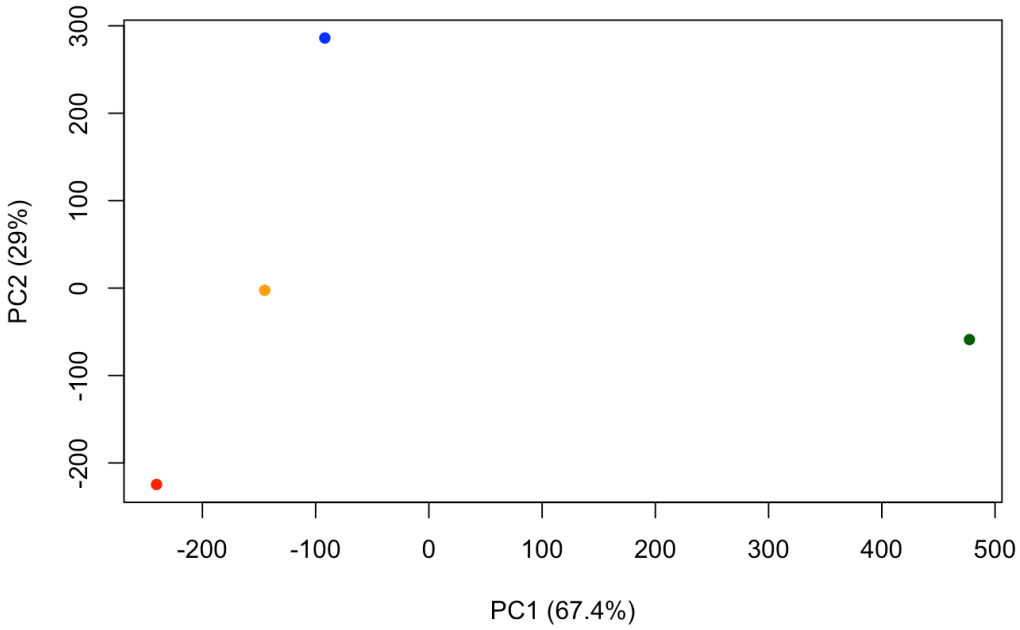
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

we are after the `pca$x’ result component to make our main PCA plot

```
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -9.152022e-15
Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

```
mycols <-c("orange", "red", "blue", "darkgreen")
plot(pca$x[,1], pca$x[,2], col=mycols, pch=16, xlab="PC1 (67.4%
```



another important result is from PCA is how the original variables (in this case, the foods) contribute to the PCs.

this is contained in the `pca$rotation` object folks often call this the "loadings" or "contributions" to the PCs

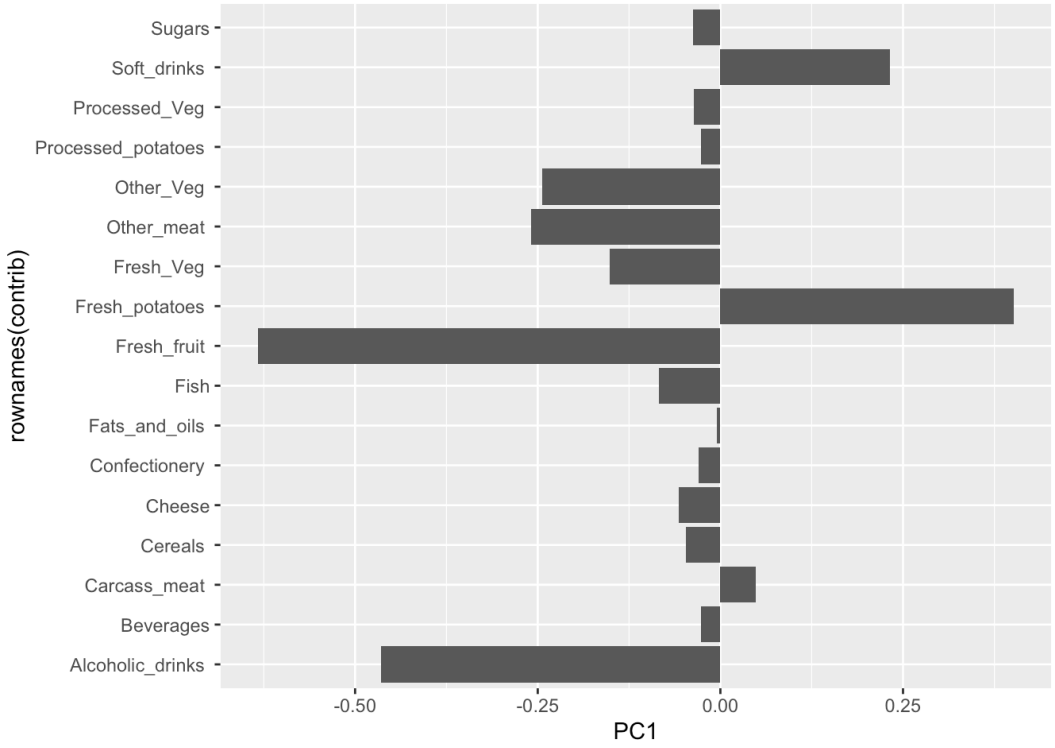
```
pca$rotation
```

```
                     PC1          PC2         PC3
PC4
Cheese         -0.056955380  0.016012850  0.02394295
-0.409382587
Carcass_meat    0.047927628  0.013915823  0.06367111
0.729481922
Other_meat     -0.258916658 -0.015331138 -0.55384854
0.331001134
Fish           -0.084414983 -0.050754947  0.03906481
0.022375878
Fats_and_oils  -0.005193623 -0.095388656 -0.12522257
0.034512161
Sugars         -0.037620983 -0.043021699 -0.03605745
0.024943337
Fresh_potatoes  0.401402060 -0.715017078 -0.20668248
0.021396007
```

```
Fresh_Veg           -0.151849942 -0.144900268  0.21382237
0.001606882
Other_Veg           -0.243593729 -0.225450923 -0.05332841
0.031153231
Processed_potatoes  -0.026886233  0.042850761 -0.07364902
-0.017379680
Processed_Veg       -0.036488269 -0.045451802  0.05289191
0.021250980
Fresh_fruit         -0.632640898 -0.177740743  0.40012865
0.227657348
Cereals             -0.047702858 -0.212599678 -0.35884921
0.100043319
Beverages           -0.026187756 -0.030560542 -0.04135860
-0.018382072
Soft_drinks          0.232244140  0.555124311 -0.16942648
0.222319484
Alcoholic_drinks    -0.463968168  0.113536523 -0.49858320
-0.273126013
Confectionery       -0.029650201  0.005949921 -0.05232164
0.001890737
```

we can make a plot along pc1

```
library(ggplot2)
contrib <-as.data.frame(pca$rotation)
ggplot(contrib) +
  aes(PC1,rownames(contrib)) +
  geom_col()
```


Sugars
Soft_drinks
Processed_Veg
Processed_potatoes
Other_Veg
Other_meat
Fresh_Veg
Fresh_potatoes
Fresh_fruit
Fish
Fats_and_oils
Confectionery
Cheese
Cereals
Carcass_meat
Beverages
Alcoholic_drinks
rownames(contrib)
-0.50
-0.25
0.00
0.25
PC1