October 25, 2023 Class 07 Machine Learning 1

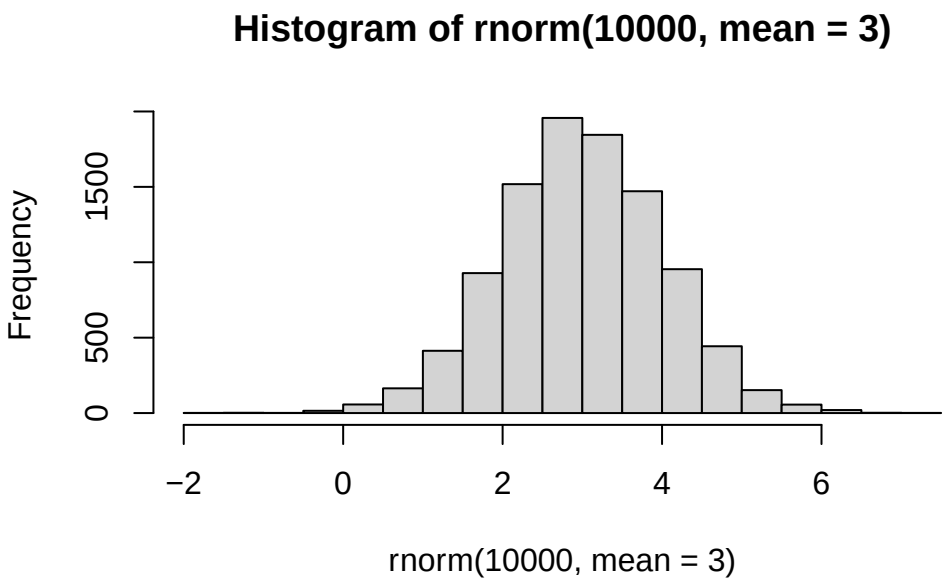
Savannah Bogus A69027475

In Class work

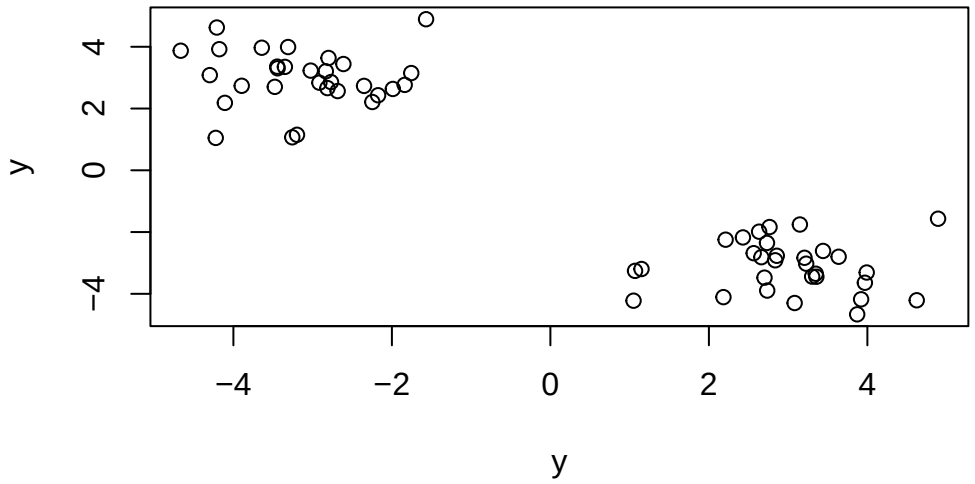
Clustering

We're going to start with k-means clustering, which is a quick way of doing things, although it is missing some things, which is why later we'll still learn the hclustering approach later, which is bottom-up. Let's make up some data using rnorm.

```
hist(rnorm(10000,mean=3))
```



```r
tmp<-c(rnorm(30,3),rnorm(30,-3))
y<-cbind(y=tmp,y=rev(tmp))
plot(y)
```



Now, we're going to use kmeans on this stuff.

```r
k<-kmeans(y,centers=2, nstart=20)
k
```

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

Cluster means:
          y         y
1 -3.102381  2.986699
2  2.986699 -3.102381

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 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] 44.07306 44.07306
 (between_SS / total_SS =  92.7 %)

Available components:

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

Q1 in class. How many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

Q2 in class. The clustering result i.e. membership vector?

```
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 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
```

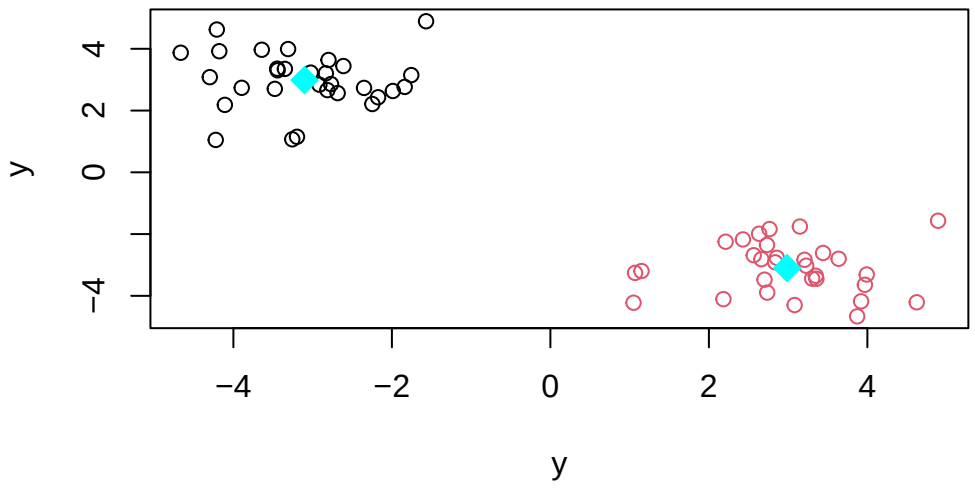
Q3 in class. The center of the clusters?

```
k$centers
```

```
          y         y
1 -3.102381  2.986699
2  2.986699 -3.102381
```

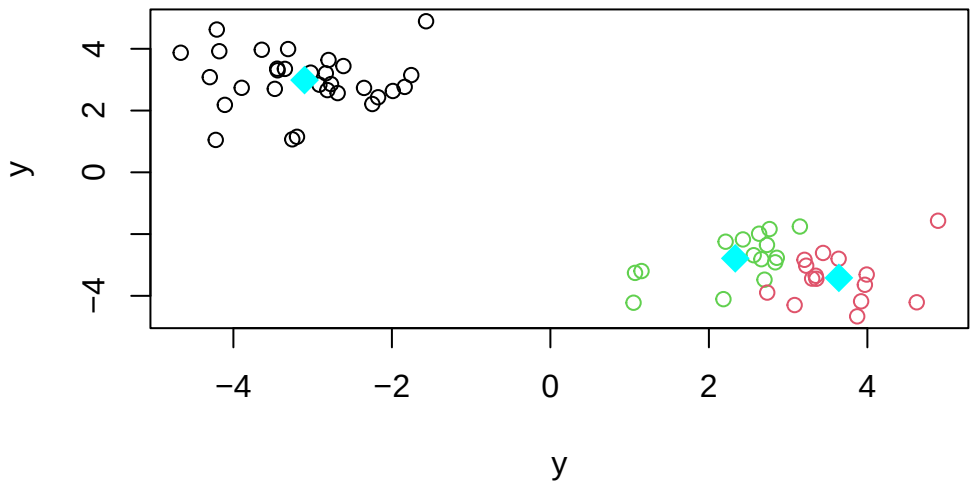
Q4. Plot of data colored by clustering results with optionally the cluster centers shown

```
plot(y,col=k$cluster)
points(k$centers,col="cyan",pch=18,cex=2)
```



Q5 in class. Run kmeans again but cluster into 3 groups and plot the results like we did above.

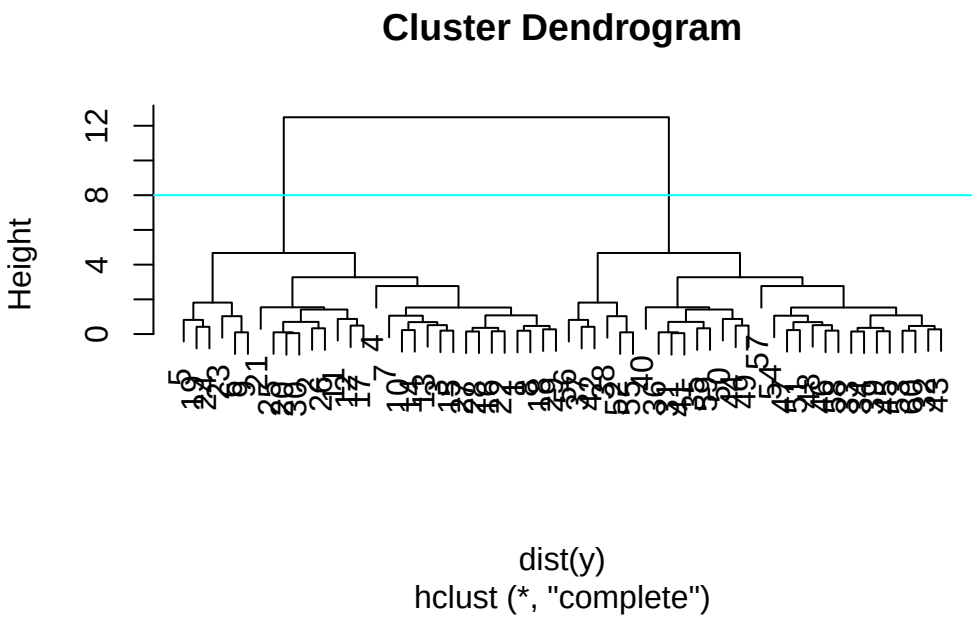
```
k3<-kmeans(y,centers=3, nstart=20)
plot(y,col=k3$cluster)
points(k3$centers,col="cyan",pch=18,cex=2)
```



Hierarchal Clustering

hclust has an advantage in that it can reveal structure in the data rather than imposing a structure in the data, as k-means can and will if you choose k sub-optimally. `hclust()` is the main base function which requires a distance matrix, NOT THE DATA ITSELF. How do we generate a distance matrix?

```
hc<-hclust(dist(y))
plot(hc)
abline(h=8,col="cyan")
```

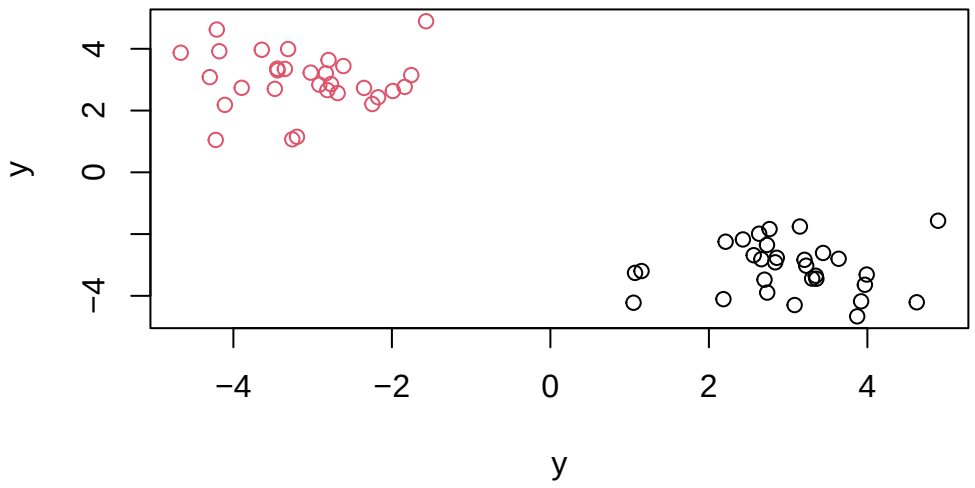


The function to get our clusters/groups from a hclust object is called `cutree()` with ONLY 1 T

```
groups<-cutree(hc,h=8)
```

Q. plot our hclust results in terms of our data colored by cluster

```
plot(y,col=groups)
```



Principal Component Analysis (PCA)

Lab Sheet: UK Foods Data

We're going to work with data from the UK about food which is 17 dimensional data as it has 17 foods over 4 countries.

```
x<-read.csv("https://tinyurl.com/UK-foods")
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
7     Fresh_potatoes     720   874      566      1033
8          Fresh_Veg     253   265      171       143
9          Other_Veg     488   570      418       355
10 Processed_potatoes     198   203      220       187
```

```
11      Processed_Veg      360   365      337       334
12        Fresh_fruit     1102  1137      957       674
13            Cereals     1472  1582     1462      1494
14          Beverages       57    73       53        47
15        Soft_drinks     1374  1256     1572      1506
16   Alcoholic_drinks      375   475      458       135
17      Confectionery       54    64       62        41
```

Q1

```
dim(x)
```

```
[1] 17  5
```

There are 17 rows and 5 columns. Next, I'm going to check the importing of the data.

```
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
```

We should only have 4 columns for the 4 countries, not 5, so we need to use `rownames` to fix this.

```
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
```

```
dim(x)
```

```
[1] 17  4
```

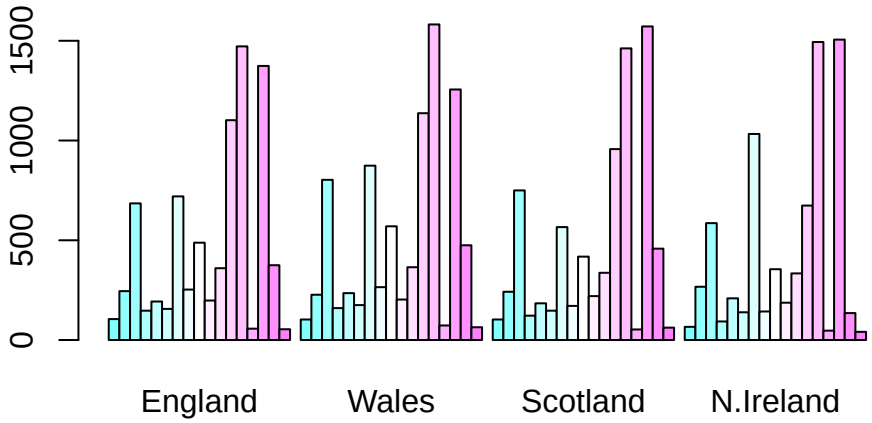
Now, we've got those rownames fixed, and the dimensions are correct.

Q2

I think the second method is more robust for me personally because it requires less typing. However, if you don't know what your data looks like before reading the CSV file, you wouldn't necessarily know whether or not the first column is row names or not, so it may not always be an option. But if you use the `read.csv` function with the `row.names` adjustment, you'd be less likely to mess up down the line, I think, since you wouldn't ever be working with data where the rownames were a whole column.

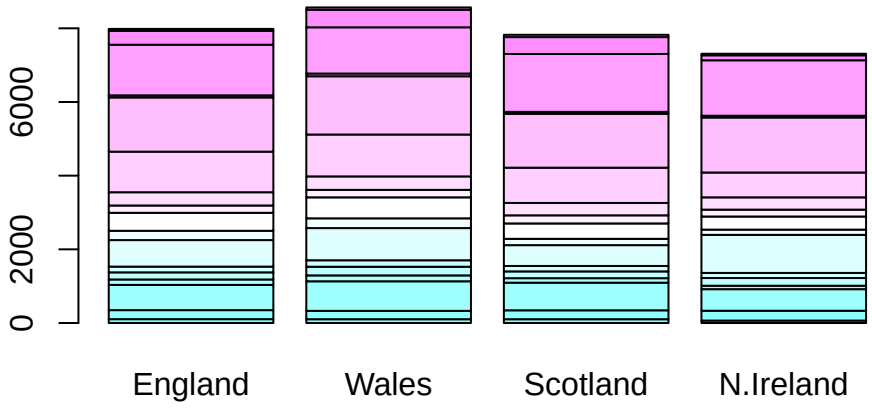
Q3

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



What is changed to get the other barplot in the html document?

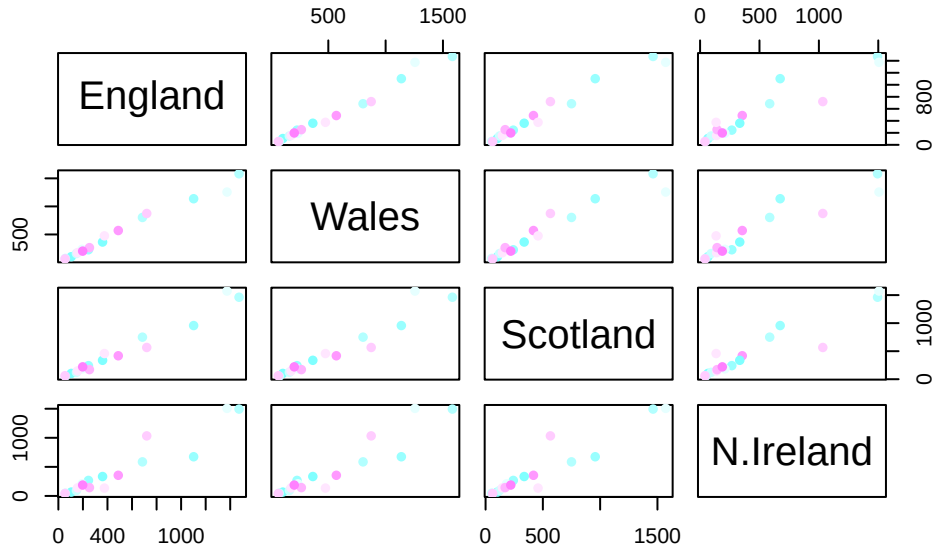
```
barplot(as.matrix(x),  col=cm.colors(nrow(x)))
```



I removed beside=T

Q5 (did we skip Q4??)

```
pairs(x, col=cm.colors(10), pch=16)
```



This is a pairwise plot with the points being rainbow and the `pch=` part making the points solid circles not hollow. I think this compares country vs country who is eating what, and then the diagonals say "england" because it's comparing england to england. If they eat the exact same amount of food for whicheer food it is, that dot will end up on the diagonal.

Q6

The upper right corner with data comapring N. Ireland looks different, but I'm honestly not sure what it is representing.

Next, we're going to work with PCA analyses. The normal R PCA implementation function is `prcomp()` which expects *observations* to be rows and *variables* to be columns, so we need to transpose the data frame. (I haven't been in a math class in so long. I missed linear algebra transposes.)

```
dim(t(x))
```

```
[1]  4 17
```

```
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
```

Q7

Plotting PC1 (which accounts for ~67% of the variance) and PC2 (which accounts for ~29% of the variance) Something a lot of people look at is the "score plot" ie. "PC plot, PC1 vs PC2 plot, etc etc"

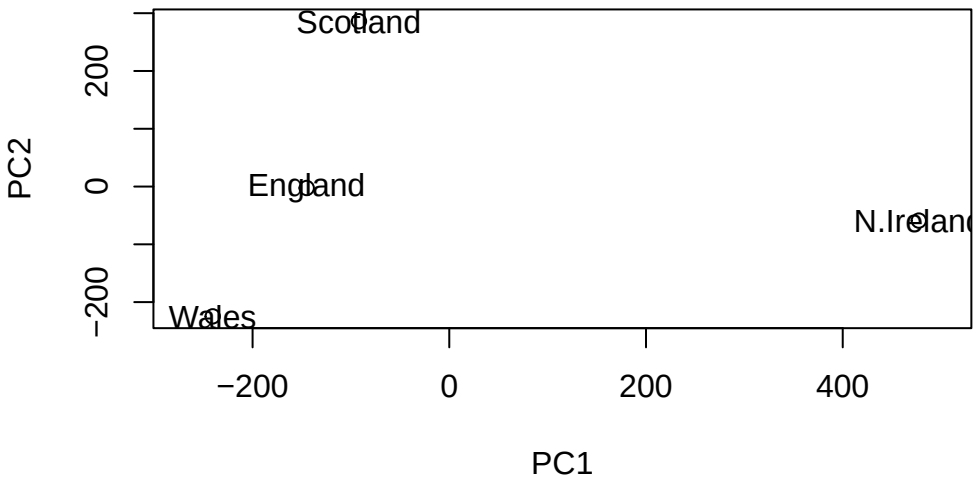
```
pca$x
```

```
                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
```

DUDE I;M LOSING IT WHY IS MY PC2 NEGATIVE???????????

Okay you just told us all that it's totally fine. Like, I know it's arbitrary, but it's making all my plots backwards, and I hate that, and I don't know why

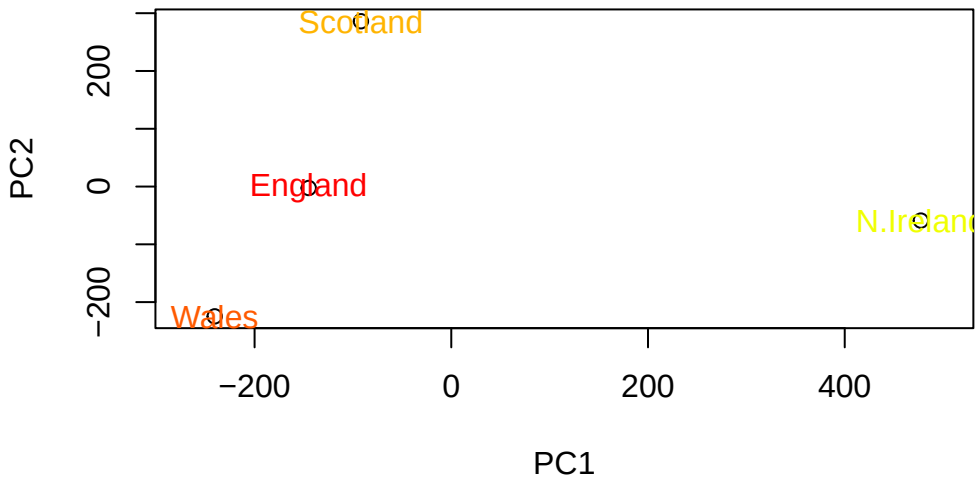
```
plot(pca$x[,1],pca$x[,2],xlab="PC1",ylab="PC2",xlim=c(-270,500))
text(pca$x[,1],pca$x[,2],colnames(x))
```



Q8

Change the colors of the countries.

```
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=rainbow(nrow(x)))
```




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

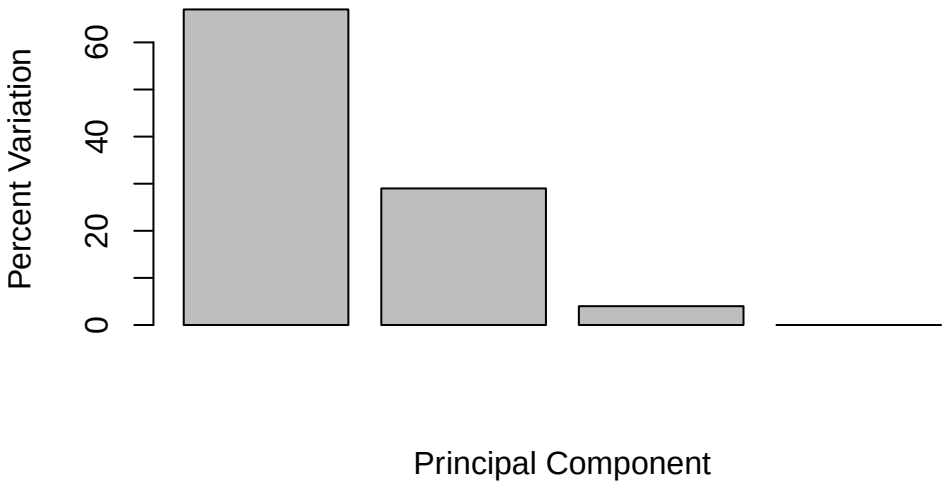
```
[1] 67 29  4  0
```

```
z<-summary(pca)
z
```

```
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
```

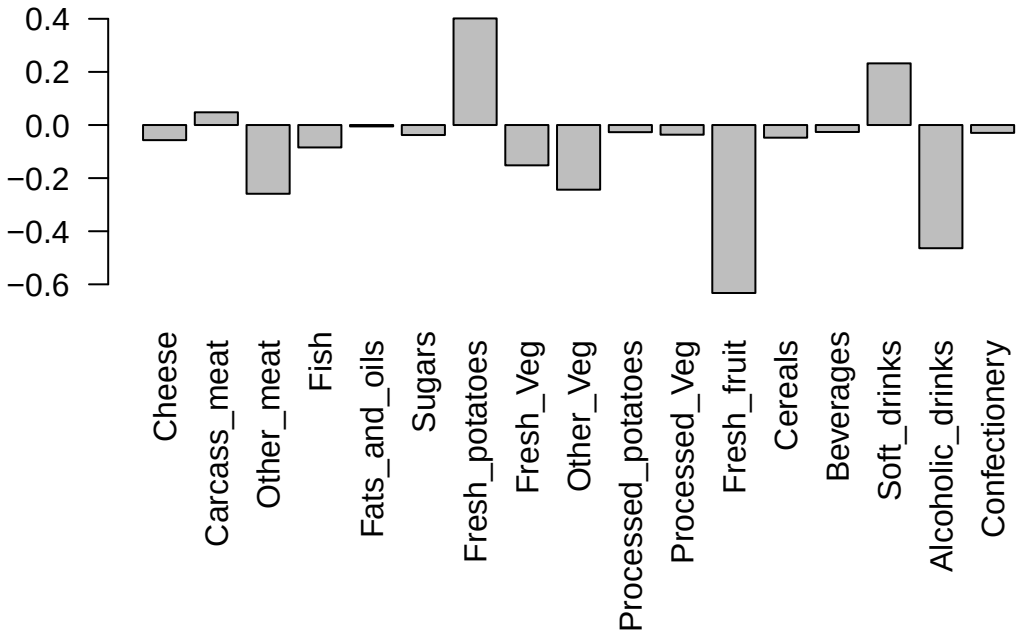
The information I'm getting from the code above (variance and summary) can itself be summarized in a plot of the variances/eigenvalues wrt the principal component number (eigenvector number) given below

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



Apparently, we can also consider the influence of the original variables upon the pricinipal components (known as **loading scores**?). This information can be obtained from `prcomp()` returned `$rotation` component and can also be summarized with a call to `biplot()`

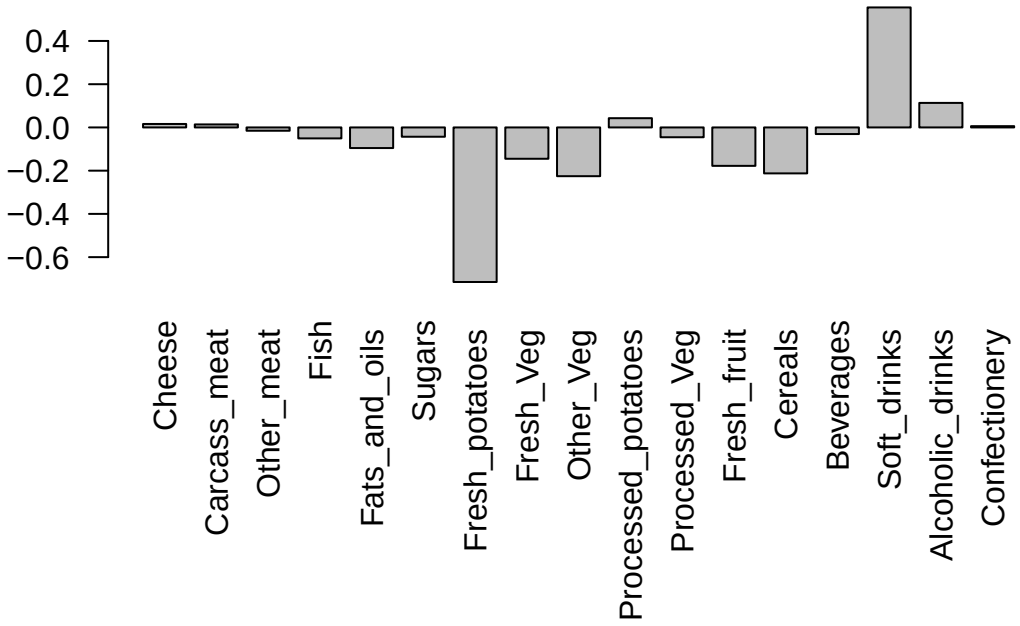
```
#this part has something to di with making axes easier to see and read
par(mar=c(10,3,0.35,0))

barplot(pca$rotation[,1],las=2)
```



In the above plot, we see the foods (observations) with the largest loading scores which effectively ““push”” N. Ireland to right positive side of the plot. ie. Look at Fresh_potatoes and Soft_drinks.We can also see that Fresh_fruit and Alcoholic_drinks push other countries to the left side of the plot.

Q9

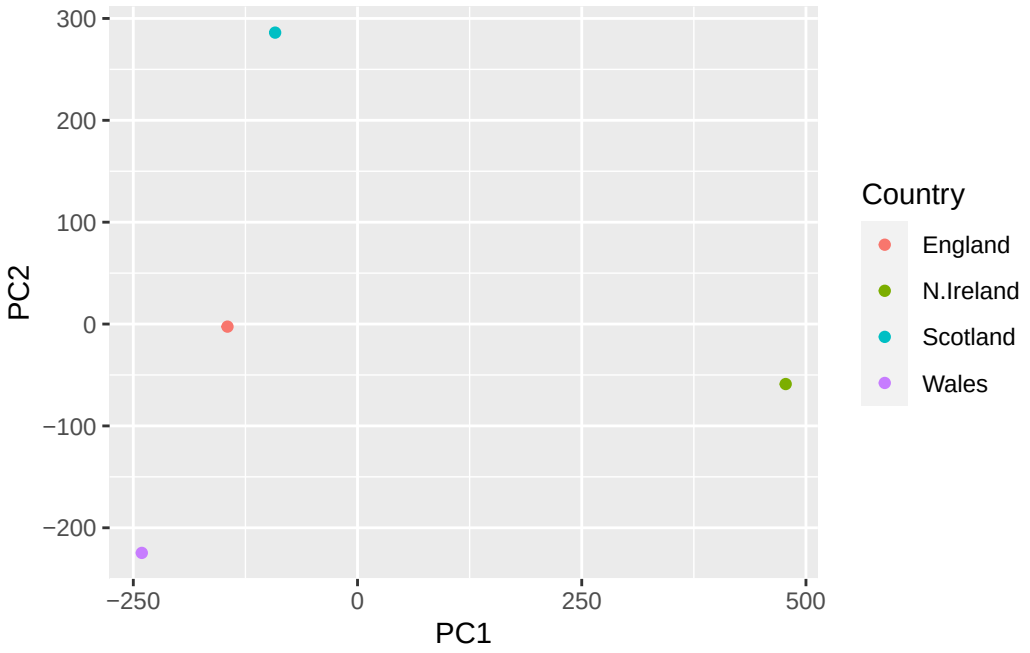
```
#this part has something to di with making axes easier to see and read
par(mar=c(10,3,0.35,0))

barplot(pca$rotation[,2],las=2)
```



For PC2, Soft_drinks and Fresh_potatoes feature prominently as well, soft drinks being positive and fresh potatoes being negative. This means we have the next biggest variance in Soft_drinks (positive) and Fresh_potatoes (negative)

Now, we're moving on to ggplot2.

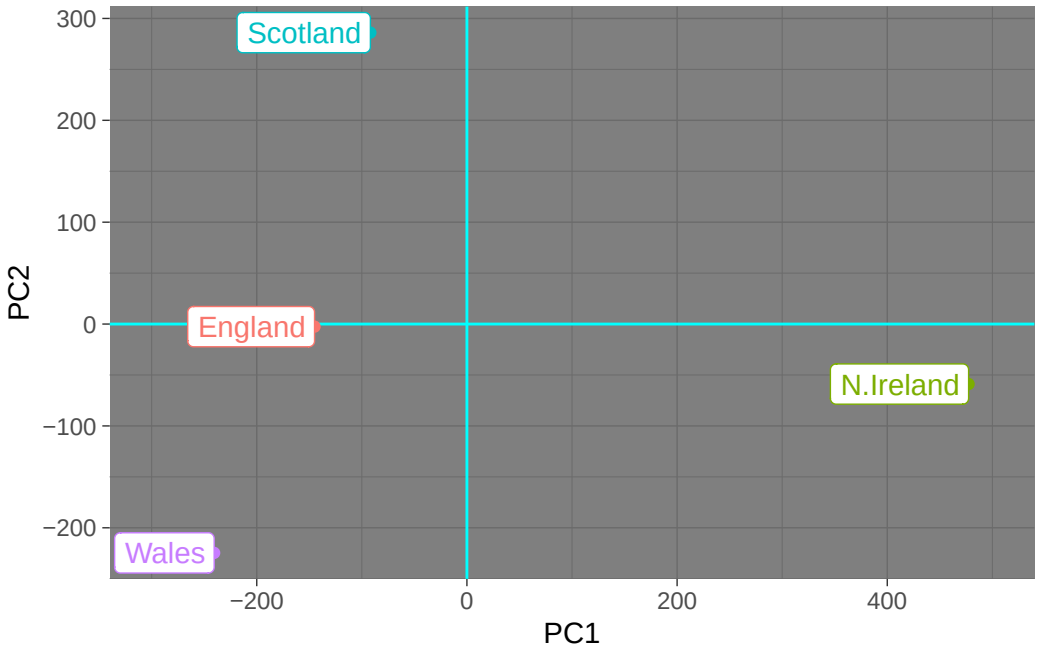
```
library(ggplot2)

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

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



We can make these plots way fancier looking, but I'll be real it's kind of a lot. I'm gonna add stuff one at a time because honsetly I don't understand what all of this is doing.

```
ggplot(df_lab)+
  aes(PC1, PC2, col=Country, label=Country)+
  geom_hline(yintercept=0,col="cyan")+
  geom_vline(xintercept=0,col="cyan")+
  geom_point(show.legend=FALSE)+
  geom_label(hjust=1,nudge=-10,show.legend=FALSE)+
  expand_limits(x=c(-300,500))+
  theme_dark()
```

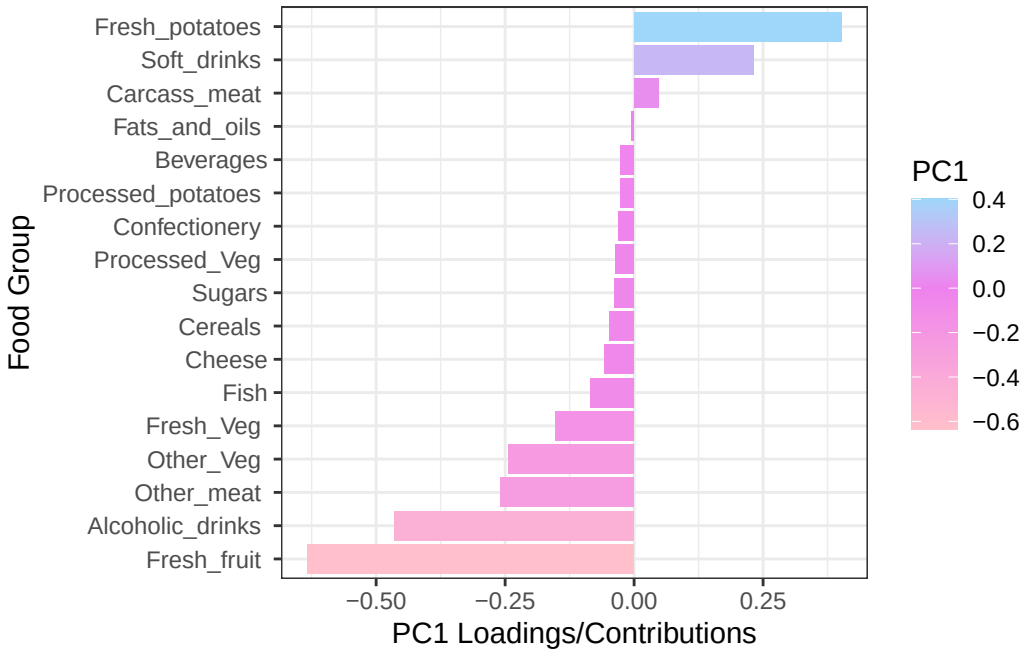
```
Warning in geom_label(hjust = 1, nudge = -10, show.legend = FALSE): Ignoring
unknown parameters: `nudge`
```



The ggplot plots can get wayyy fancier. I didn't realize this, but the order that you put each element into the plot matters. I was trying to add elements in the order of my familiarity/understanding of them, so I put the geom_hline and geom_vline nearer to the bottom, and this applied them AFTER the labels, which made those lines cut through the label. Lesson learned.

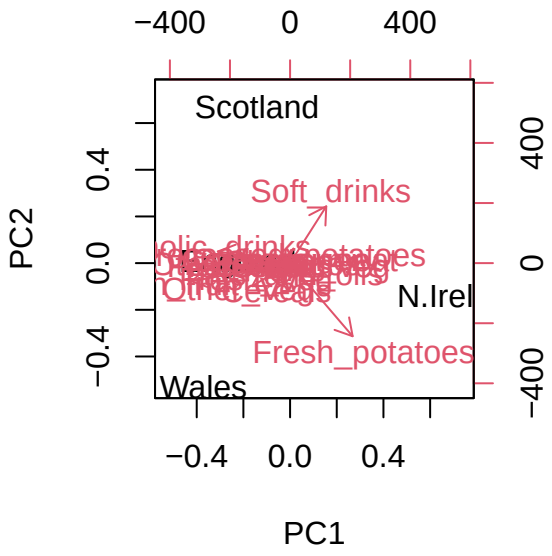
Next, it looks like we're doing the PC contributuions or loading scores, which is stored in `pca$rotation`

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

ggplot(ld_lab)+
  aes(PC1, reorder(Food, PC1),bg=PC1)+
  geom_col()+
  xlab("PC1 Loadings/Contributions")+
  ylab("Food Group")+
  scale_fill_gradient2(low="pink",mid="violet",high="cyan")+
  theme_bw()
```



Another way to do this is a biplot(), which can be useful for small datasets.

```
biplot(pca)
```



Looks bad.

Q10 incoming

```
rna.data<-read.csv("https://tinyurl.com/expression-CSV",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
```

```
nrow(rna.data)
```

```
[1] 100
```

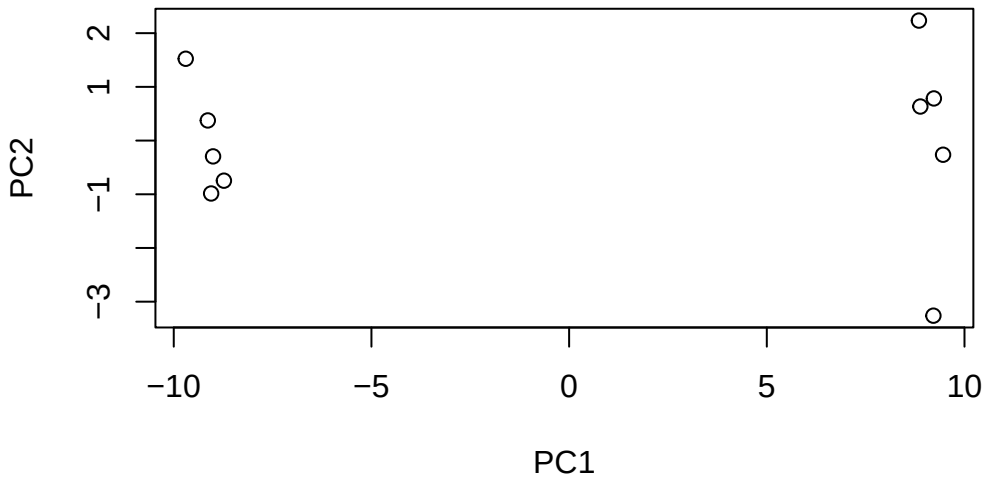
```
dim(rna.data)
```

```
[1] 100  10
```

There are 100 genes and 10 samples in this dataset.

This data has way too many dimensions to make bar graphs or what have you, so let's make a PCA and see where we're at. Don't forget to transpose!

```
pca2<-prcomp(t(rna.data),scale=TRUE)
plot(pca2$x[,1],pca2$x[,2],xlab="PC1",ylab="PC2")
```




```
summary(pca2)
```

```
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 3.345e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

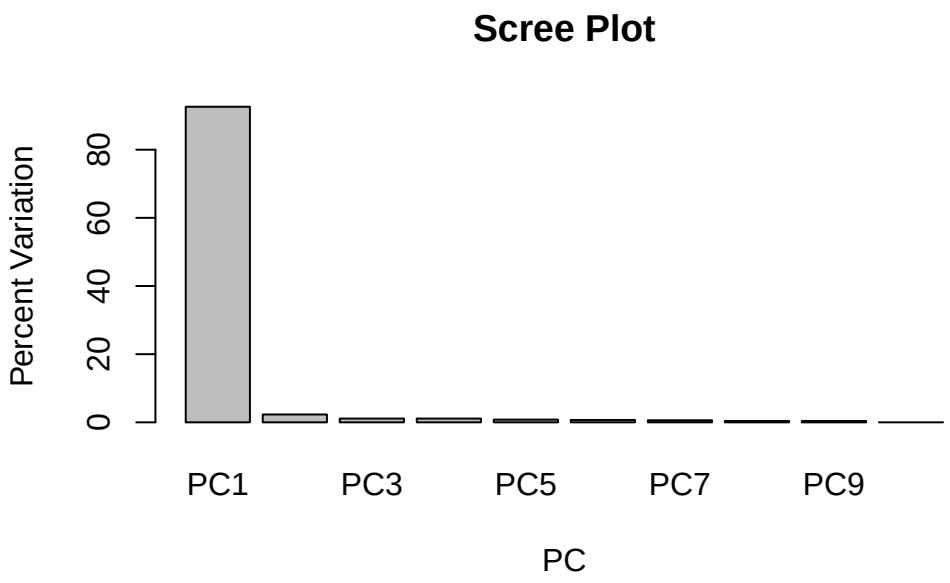
We're going to make our own Scree plot.

```
pca2.var<-pca2$sdev^2
#gonna look at percent variance
pca2.var.per<-round(pca2.var/sum(pca2.var)*100,1)
pca2.var.per
```

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

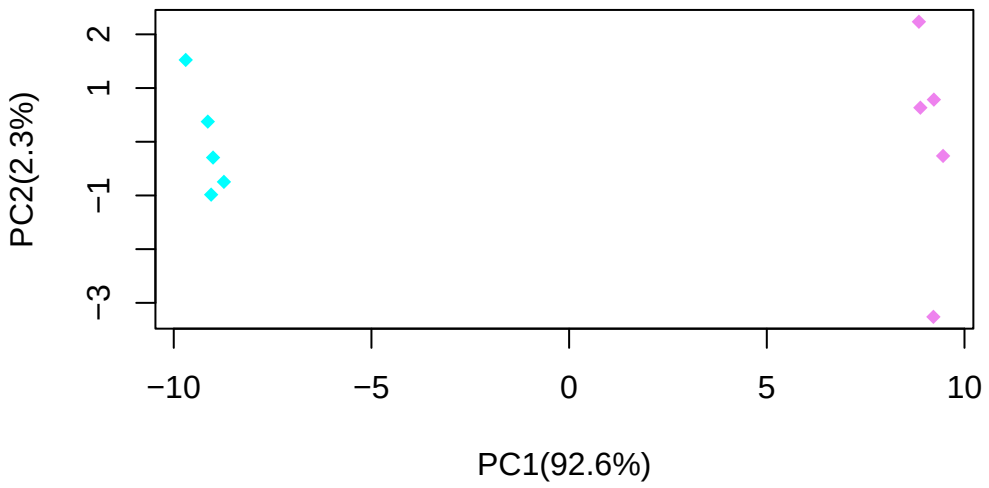
We can use this to generate our own barplot

```
barplot(pca2.var.per,main="Scree Plot",
        names.arg=paste0("PC",1:10),
        xlab="PC",ylab="Percent Variation")
```



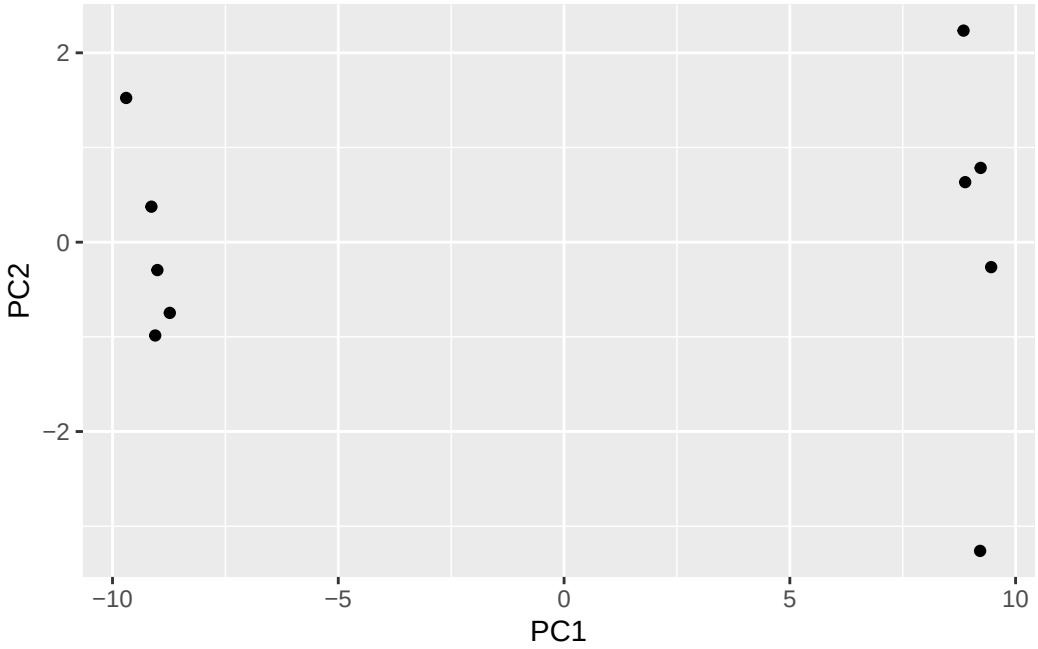
Next, we're going to make our main PCA plot more attractive and more useful.

```
colvec<-colnames(rna.data)
colvec[grep("wt",colvec)]<-"cyan"
colvec[grep("ko",colvec)]<-"violet"

plot(pca2$x[,1],pca2$x[,2],col=colvec,pch=18,
     xlab=paste0("PC1(", pca2.var.per[1],"%)"),
    ylab=paste0("PC2(", pca2.var.per[2],"%)"))
```



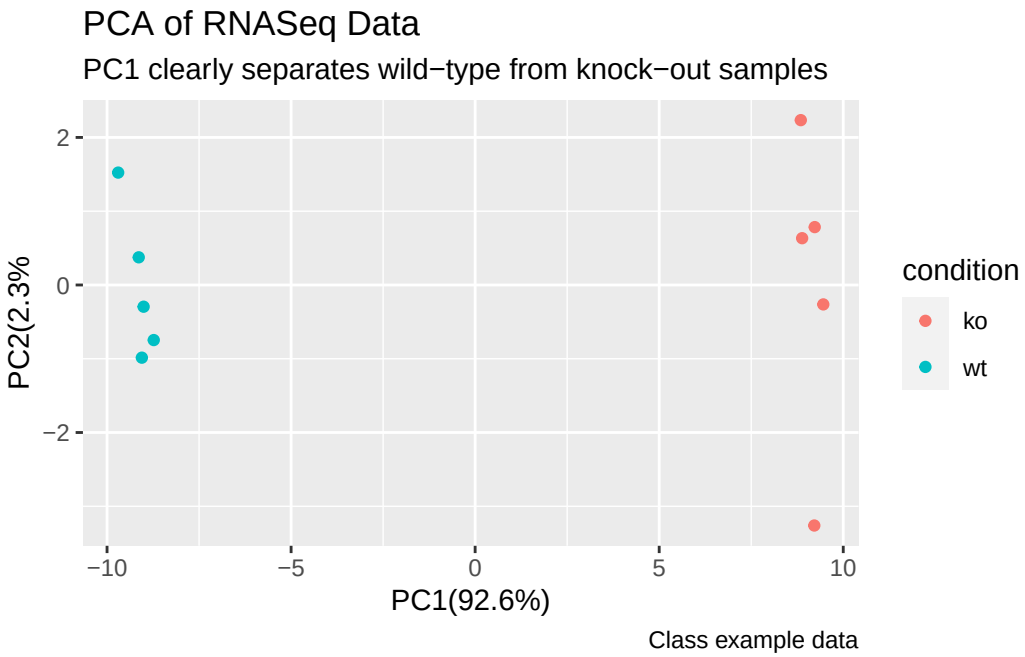
Let's try all this junk again with ggplot.

```
library(ggplot2)
df2<-as.data.frame(pca2$x)

ggplot(df2)+
  aes(PC1,PC2)+
  geom_point()
```



I think we should color by "condition" or ko vs wt. We're going to add a wt and ko condition column to the original data.

```
df2$samples<-colnames(rna.data)
df2$condition<-substr(colnames(rna.data),1,2)

ggplot(df2)+
  aes(PC1,PC2,col=condition)+
  geom_point()+
  labs(title="PCA of RNASeq Data",
       subtitle="PC1 clearly separates wild-type from knock-out samples",
       x=paste0("PC1(",pca2.var.per[1],"%)"),
       y=paste0("PC2(",pca2.var.per[2],"%"),
       caption="Class example data")
```



Gene Loadings

```
loading_scores<-pca2$rotation[,1]

gene_scores<-abs(loading_scores)
gene_score_ranked<-sort(gene_scores,decreasing=TRUE)

top_10_genes<-names(gene_score_ranked[1:10])
top_10_genes
```

```
[1] "gene100" "gene66"  "gene45"  "gene68"  "gene98"  "gene60"  "gene21"
[8] "gene56"  "gene10"  "gene90"
```