Class07: Machine Learning 1

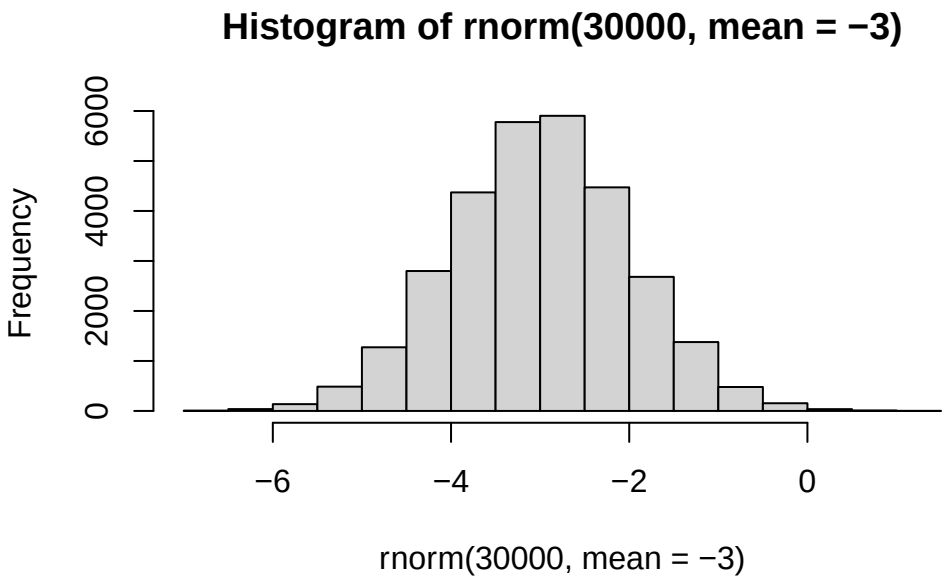
Daira

In this class we will explore and get practice with clustering and Principal component analysis (PCA)

Clustering with K-means

First create data with cluster where we known where the results should be

```
hist(rnorm(30000, mean=-3))
```



```r
#mean is where you want center
#sd is standard deviation how spread out (wider bigger, smaller thinner)
```

```r
rnorm(30, -3)
```

```
 [1] -2.5766759 -0.9111242 -1.6855494 -3.6599887 -3.2456450 -3.2551787
 [7] -2.4722340 -3.0234807 -0.6818453 -2.5884747 -3.5225187 -3.1483292
[13] -1.7171880 -3.6655688 -3.4386074 -3.4920678 -2.6872423 -3.3156387
[19] -2.5534356 -3.9891338 -3.3446265 -4.1336107 -2.6923840 -2.3898981
[25] -5.3563255 -2.2957951 -2.7342426 -2.6410415 -0.8099617 -1.9405197
```

```r
rnorm(30, 3)
```

```
 [1] 2.645709 1.010699 2.477444 2.800395 3.859844 3.143073 3.417874 2.873892
 [9] 2.713433 3.686195 2.458880 2.300318 4.045876 4.983359 3.870819 3.950549
[17] 2.546109 2.588911 2.645492 4.364626 3.879186 4.061124 2.488273 3.388780
[25] 1.412463 2.250458 3.437762 4.369825 1.459730 3.810238
```

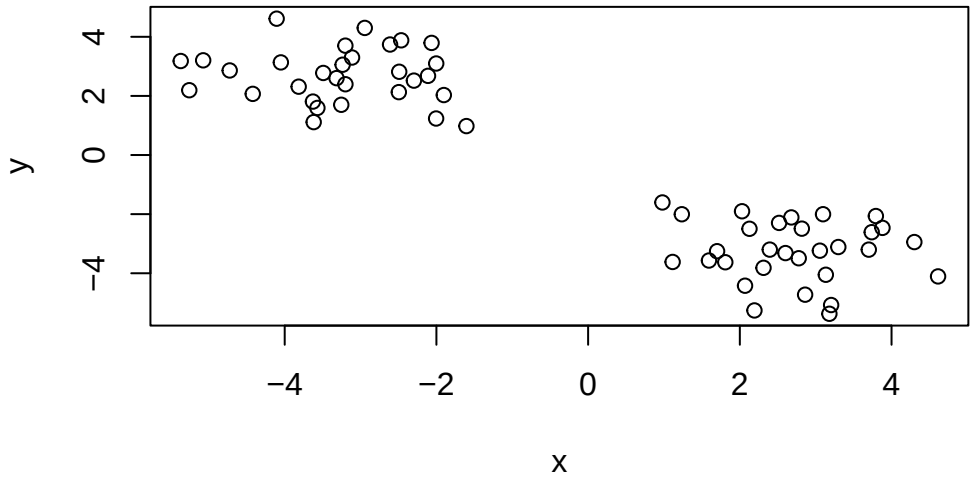
```r
##PUT THEM IN SAME VECTOR WITH CONCATENATE
tmp <- c(rnorm(30, -3), rnorm(30,3))
#now we want to bind them
x <- data.frame(x=tmp, y=rev(tmp))
x
```

```
           x         y
1  -2.0039777 1.2352632
2  -3.1996575 2.3949712
3  -4.7250506 2.8607571
4  -3.6179929 1.1139389
5  -1.9023776 2.0292902
6  -2.0026744 3.0968452
7  -1.6044065 0.9793455
8  -3.2363831 3.0553626
9  -4.0507547 3.1336826
10 -2.9446292 4.3005616
11 -2.0636238 3.7932961
12 -3.1124975 3.2984015
13 -2.4644260 3.8804728
14 -2.4903420 2.8173591
```

```
15 -3.4918903  2.7767597
16 -4.1064348  4.6135876
17 -5.3709016  3.1806707
18 -3.2544474  1.7000139
19 -3.5690937  1.5928979
20 -5.0749911  3.2039088
21 -3.2004072  3.7000659
22 -2.4955045  2.1274116
23 -3.3172081  2.6014360
24 -2.6111055  3.7390175
25 -4.4212182  2.0686351
26 -3.8158082  2.3123132
27 -3.6278523  1.8084154
28 -2.2967443  2.5175873
29 -2.1109263  2.6787698
30 -5.2582204  2.1926611
31  2.1926611 -5.2582204
32  2.6787698 -2.1109263
33  2.5175873 -2.2967443
34  1.8084154 -3.6278523
35  2.3123132 -3.8158082
36  2.0686351 -4.4212182
37  3.7390175 -2.6111055
38  2.6014360 -3.3172081
39  2.1274116 -2.4955045
40  3.7000659 -3.2004072
41  3.2039088 -5.0749911
42  1.5928979 -3.5690937
43  1.7000139 -3.2544474
44  3.1806707 -5.3709016
45  4.6135876 -4.1064348
46  2.7767597 -3.4918903
47  2.8173591 -2.4903420
48  3.8804728 -2.4644260
49  3.2984015 -3.1124975
50  3.7932961 -2.0636238
51  4.3005616 -2.9446292
52  3.1336826 -4.0507547
53  3.0553626 -3.2363831
54  0.9793455 -1.6044065
55  3.0968452 -2.0026744
56  2.0292902 -1.9023776
57  1.1139389 -3.6179929
```

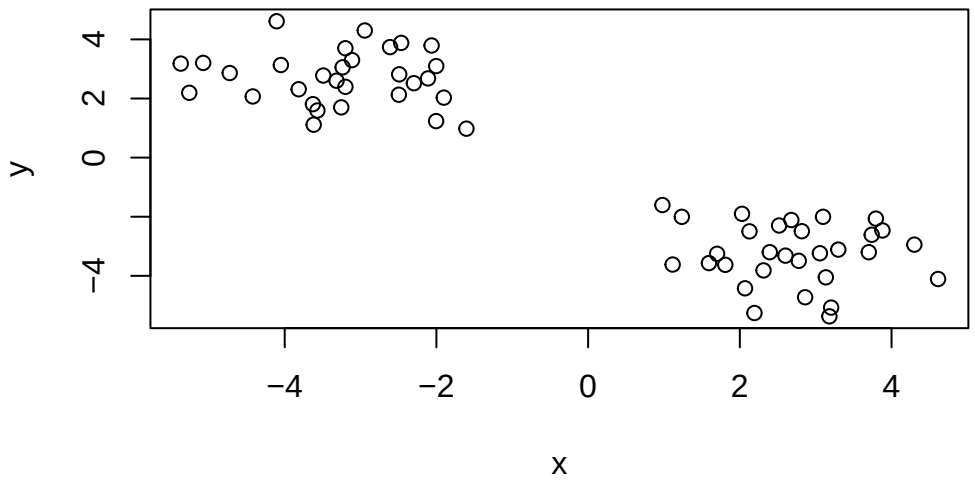
```
58  2.8607571 -4.7250506
59  2.3949712 -3.1996575
60  1.2352632 -2.0039777
```

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



Lets have a look

```
plot(x)
```



K-means

What do we need for this:

```
km <- kmeans(x, centers = 2, nstart=20)
##x is data, center is k # we assign, nstart is the iterations
```

It is important to not just run the analysis but to be able to get your important results back

Q1 How do I find cluster sizes?

```
?keans
```

```
No documentation for 'keans' in specified packages and libraries:
you could try '??keans'
```

```
km$size
```

```
[1] 30 30
```

Q2 how do i find cluster centers?

```
km$centers
```

```
          x         y
1 -3.248052  2.693457
2  2.693457 -3.248052
```

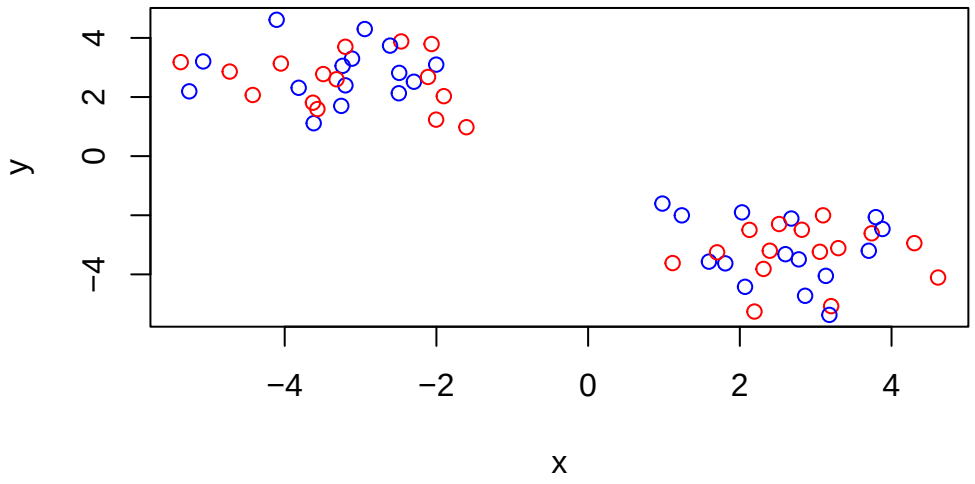
Q3 How about the main result - the cluster assignment for each value?

```
km$cluster
```

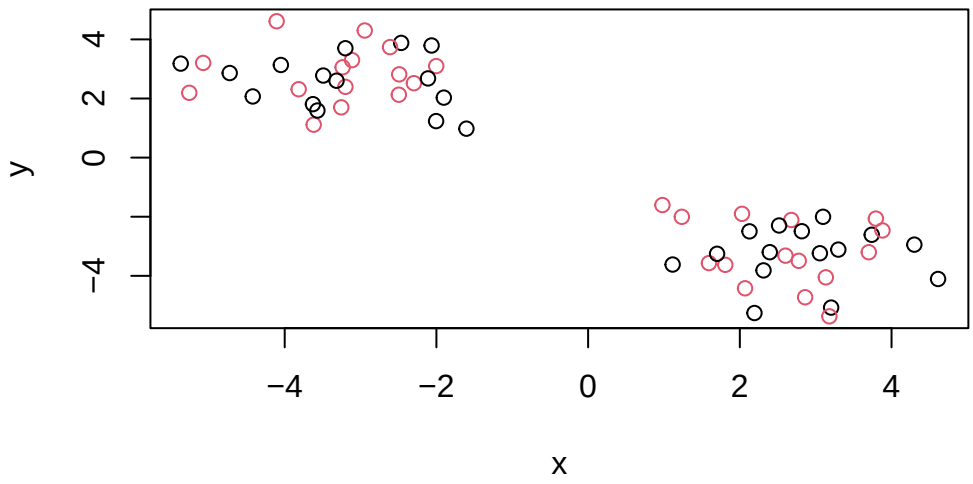
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Q4 Can we make a summary figure showing the results? that is the points colored by cluster assignment? and maybe add cluster centers?

```
plot(x, col=c("red", "blue"))
```

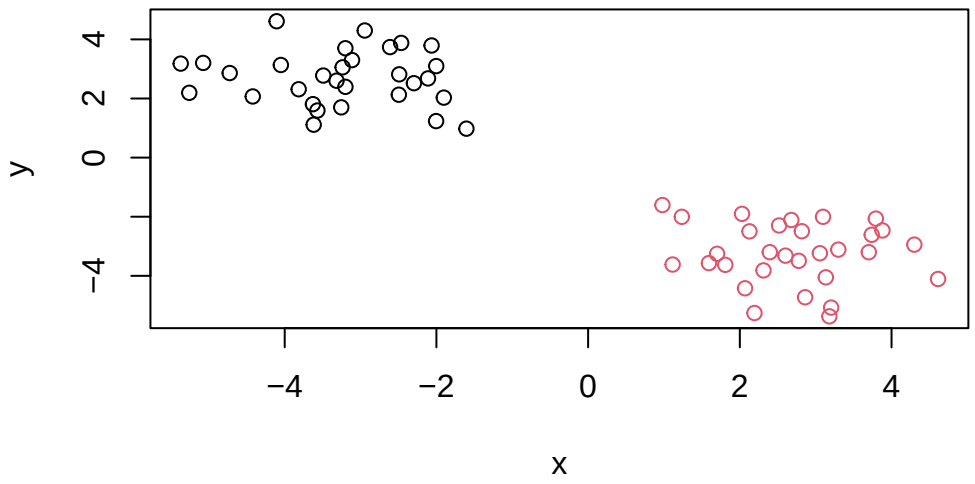



```
plot(x, col=c(1,2)) ##numbers of col are colors 1 is black, 6 is purple
```




```
## can we add information with cluster assignment based on color?
```

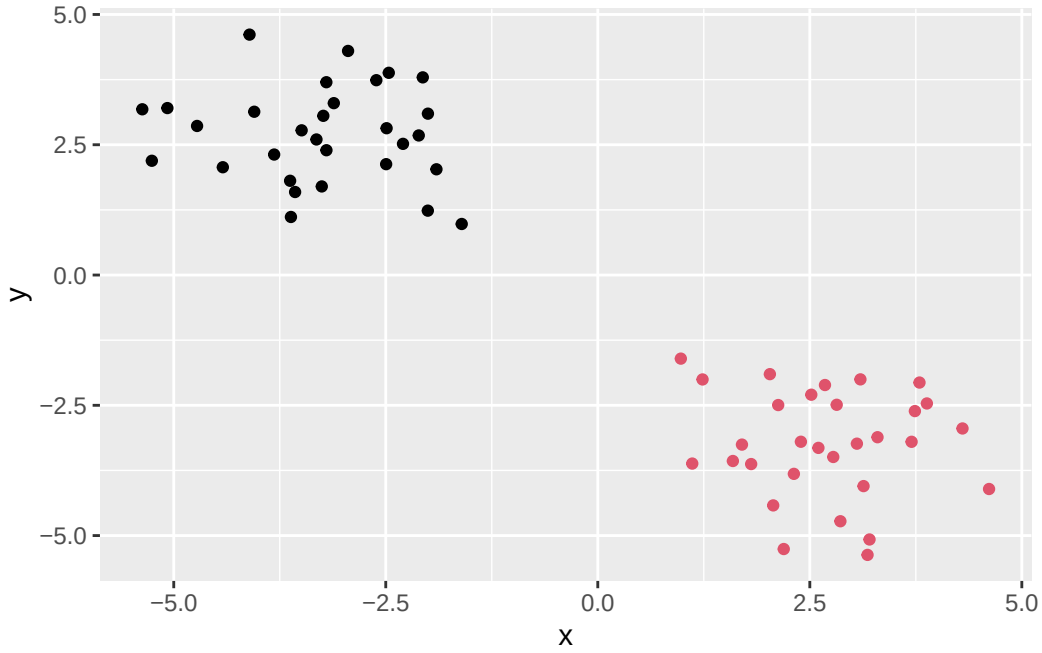
```
plot(x, col=km$cluster)
```



Lets do it in ggplot

need data, aes, and geoms

```
library(ggplot2)
ggplot(x) +
  aes(x, y) +
  geom_point(col=km$cluster)
```

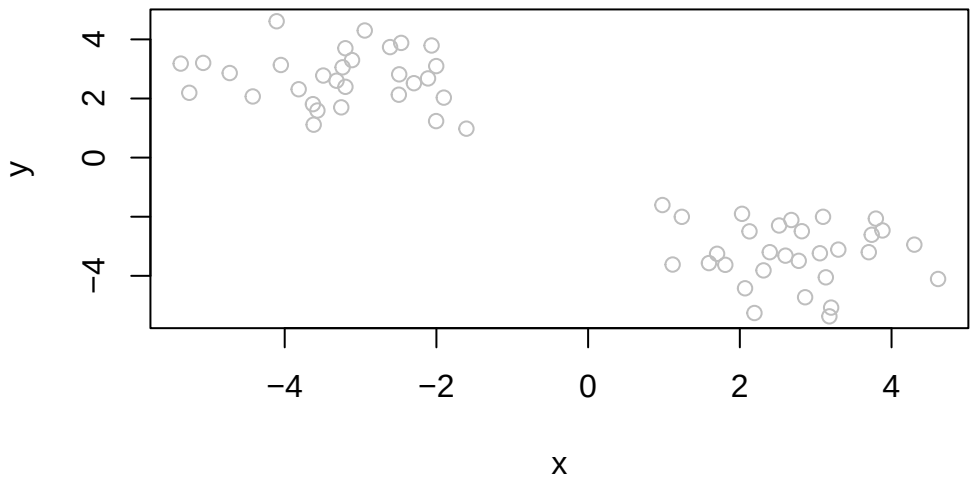


##nothing to do with cluster but make up a color vector

```
mycols <- rep("gray", 60)
mycols
```

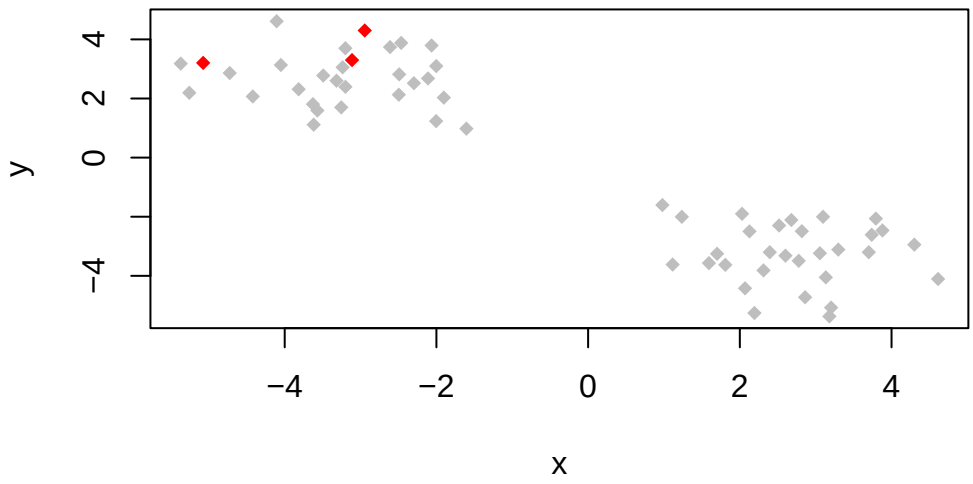
```
 [1] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
[11] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
[21] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
[31] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
[41] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
[51] "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray" "gray"
```

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



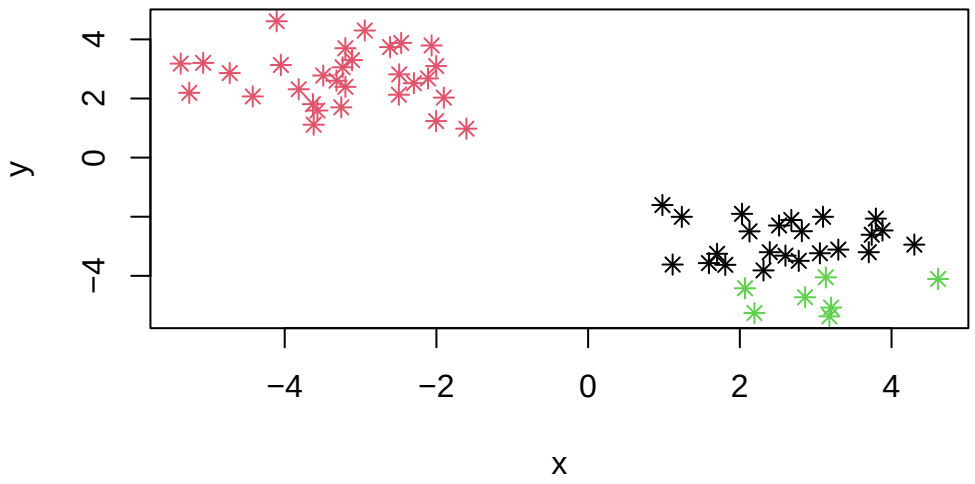
Lets highlight points 10,12,20 as red

```
mycols[c(10, 12, 20)] <- "red"
plot(x, col=mycols, pch=18)
```



#lets try with different number of centers (aka Ks)

```
kmnew <- kmeans(x, centers=3)
plot(x, col=kmnew$cluster, pch=8)
```



What we get out of this, is the sum of squares

```
kmnew$tot.withinss
```

```
[1] 90.66141
```

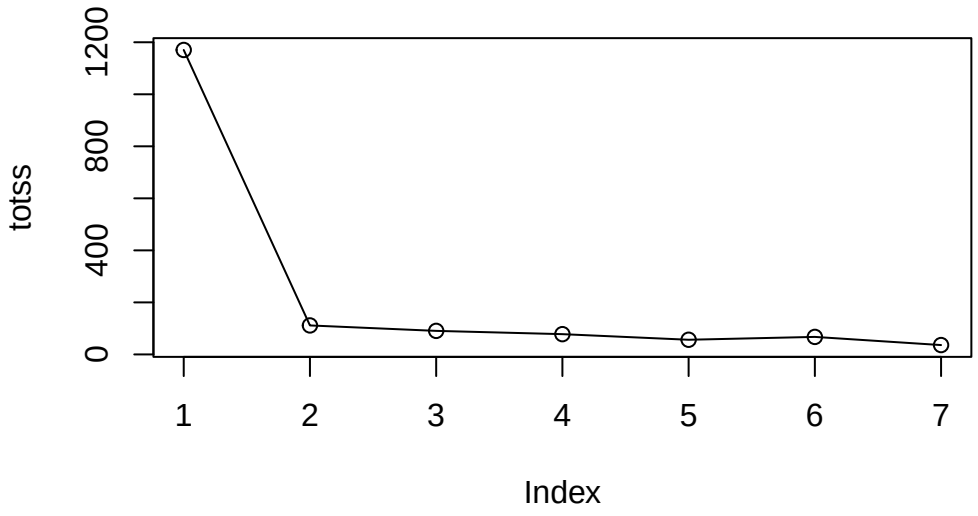
```
#we keep track of this for different K numbers, keep the one with smallest SS
```

Lets do a for loop

What we want to do is try out different numbers of K from 1 to 7, we can write a `for` loop to do this for us and calculate `$tot.withinss` each time

```
totss <- NULL
k <- 1:7

for(i in k) {
totss <- c(totss, kmeans(x, centers = i)$tot.withinss)
}
```

```
plot(totss, typ="o")
```

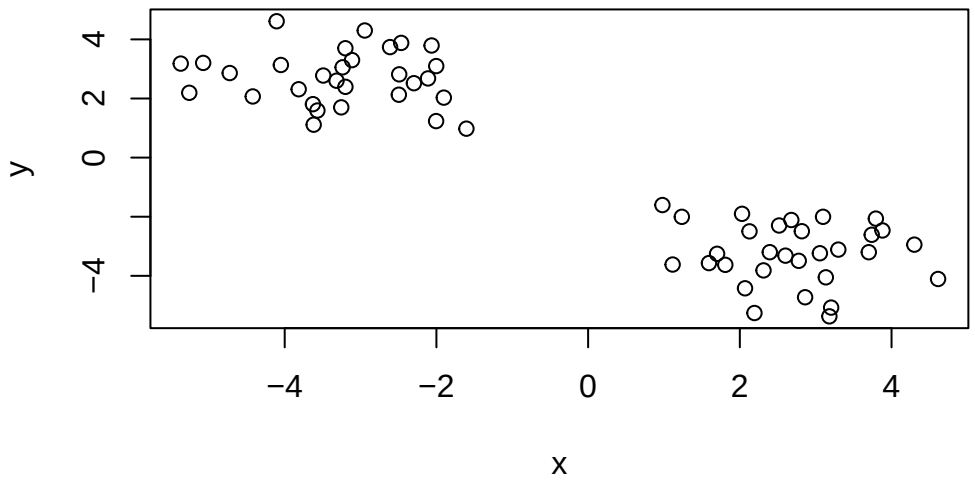



```
## typ is o gives points and line
```

Hierarchical clustering

bottom up, we will use our x again

```
plot(x)
```



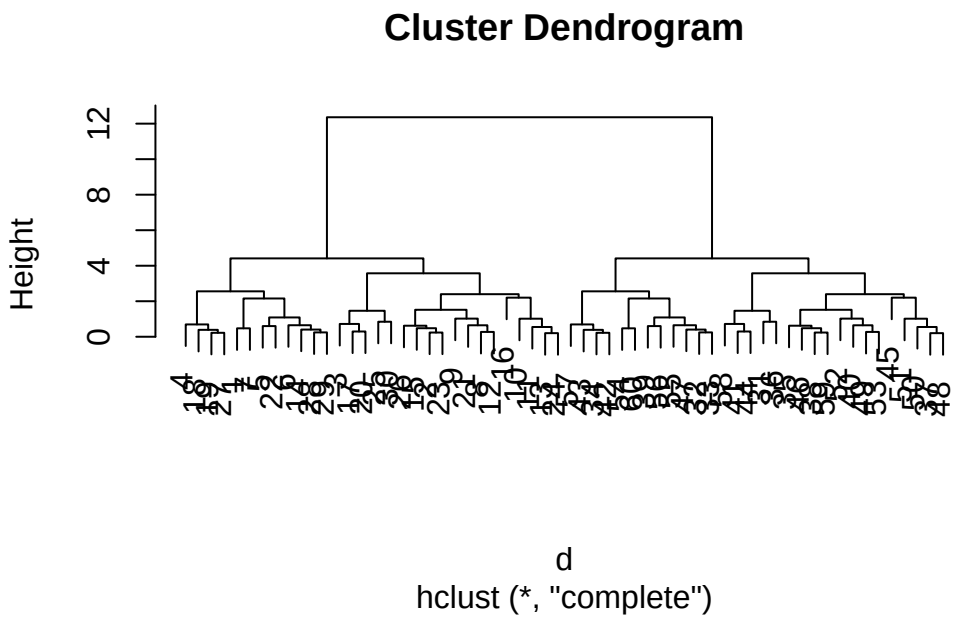
One of the key differences, is that we can't give `hclust` our input x like we did for `kmeans()` `hclust` is a lot more flexible, we can give it distance between things.

* **but first we need to calculate a distance matrix**. aka how far apart are each point from each other. we use the `dist` function by default will calculate euclidean distance which usese pythogram theorem.

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

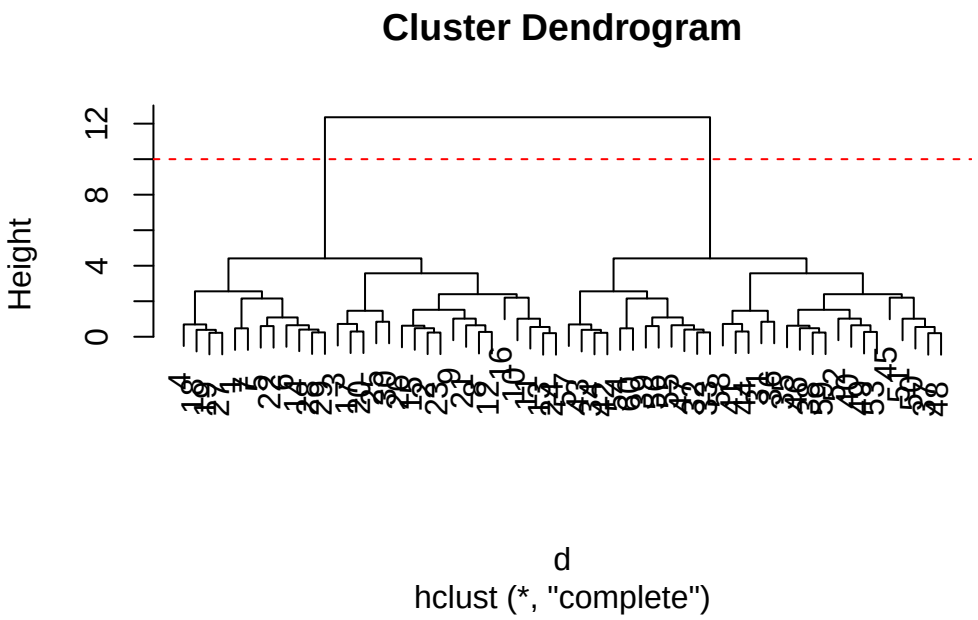
```
Call:
hclust(d = d)

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

```
plot(hc)
```



The print out is not too helpful, but the plot method is! lets look at it

```
plot(hc)
abline(h=10, col="red", lty=2)
```



##nums on one side are 1-30, the other side is 31-60. weird.

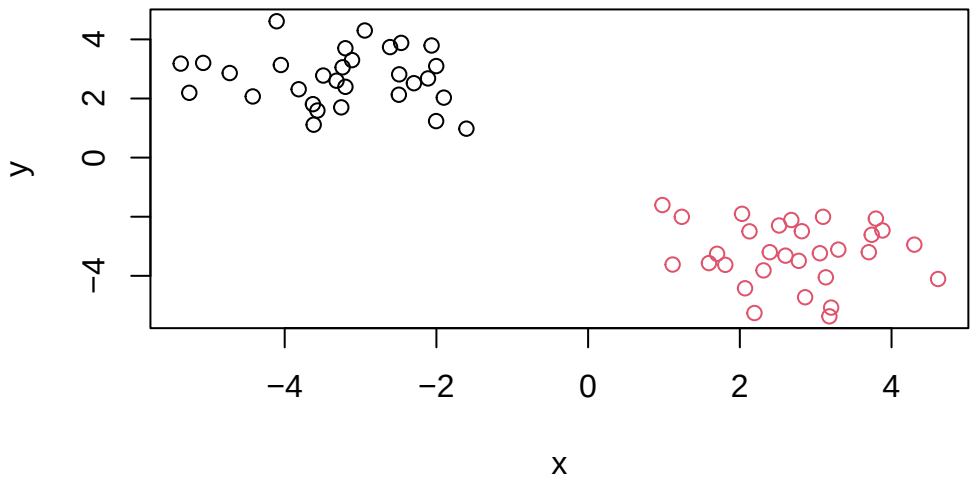
We can cut this tree, use `cutree` to get the all important cluster membership vector out of hclust

```
cutree(hc, h=10)
```

```
 [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
```

You can also set `k=` argument to `cutree()` argument to get k cluster groups. example

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

plot(x, col=groups)
```



Class PCA plots

PRINCIPAL Component Analysis! the most important things about your data The main base R functions to do PCA is called `prcomp()` Purpose for this is to reduce dimensions aka is to view the data in a useful way. PC's aka eigenvectors (what runs our credit card reader machines!). 1. reduce dimensionality 2. visualize multidimensional data 3. to choose the most useful variables (features) 4. to identify groupings of objects (e.g genes/samples) 5. to identify outliers

UK food

First part, PCA of UK food data

```
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?

```
sum(ncol(x))
```

```
[1] 5
```

```
sum(nrow(x))
```

```
[1] 17
```

```
##dim gives us the information for both
dim(x)
```

```
[1] 17  5
```

A: There are 17 rows and 5 columns, we used the R function sum for nrow and ncol (to get the sum of the rows and columns). we can also use dim to see it.

Now to change the matrix a bit…

```
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
```

Now check `dim()` again to see if size changed (column should be less now?) lets check

```
dim(x)
```

```
[1] 17  4
```

But I want to be better, so I want to do with `row.names` argument

```
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
```

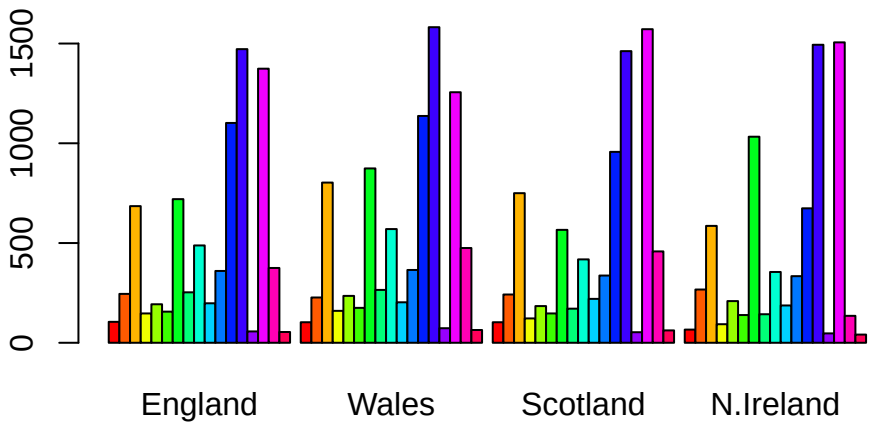
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

```
x <- read.csv(url, row.names=1)
##I think this approach is better and prefer it because you are telling it that first colu
```

A:I think the read() is better and prefer it because you are telling it that first column is row names and it doesn't have the reiterative process that could get rid of column data if you keep running it.

Okay, now using base R plot to look at data

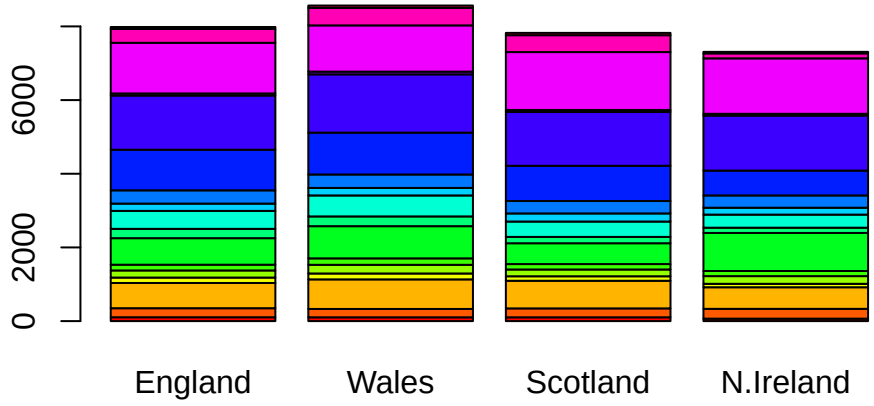
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



Very colorful but not very informative.

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

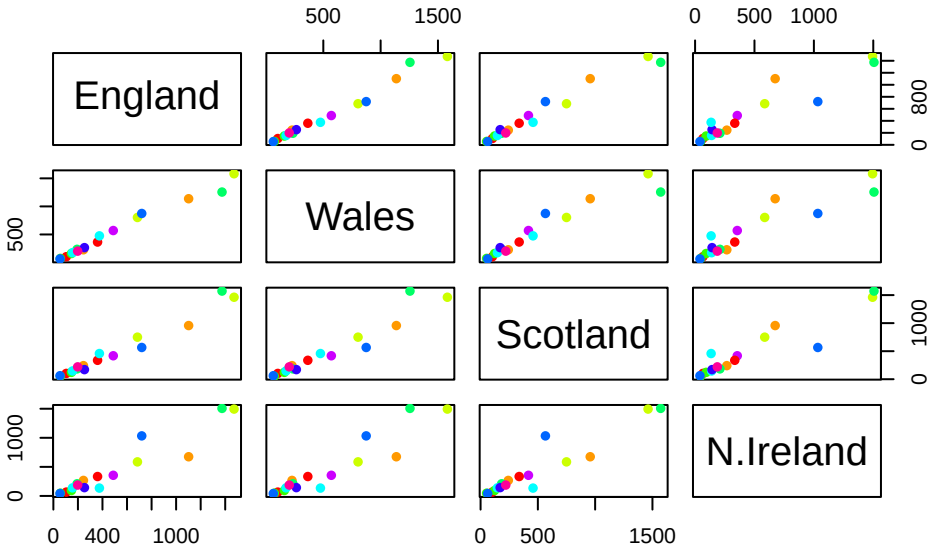
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```




```
##the optional argument you have to make beside=FALSE so it lays on top (stacked) not by t
```

Q5(should be 4?)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)
```



A: the pairs plot, shows us multi-panels, the way to interpret to read across or down, example for England across the top row, on x axis is other countries and y axis is England. More information, if the dots lie on the diagonal then they are similar between the two countries. If they are not diagonal then there are differences between the two countries.

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

A: In terms of the dataset, it looks like the main difference is that N. Ireland and other counties is due to variables like fresh_potatoes and fresh-fruit, they consume more fresh potatoes and less fresh fruit than other countries.

Now to USE PCA to look at it!

for PCA we need the transpose of the food data so we use `t()` function. it moves the food and countries switched.

```
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
```

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

```
pca$x
```

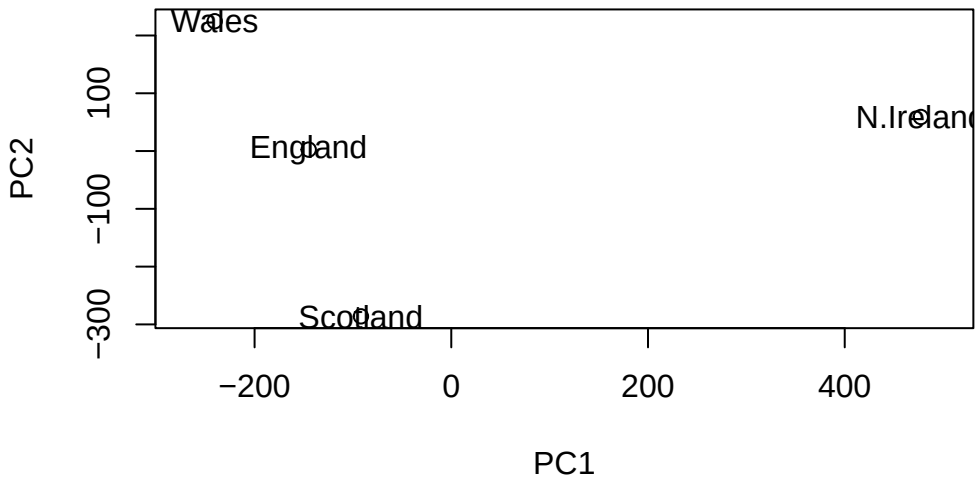
```
                PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```
#those are the new axis that PCA gave us to plot the data
```

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

Make the PC1 v PC2 plot, "score" plot, "PCA" plot

```
# Plot PC1 vs PC2
mycols <- c("orange","red","blue","darkgreen")
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```

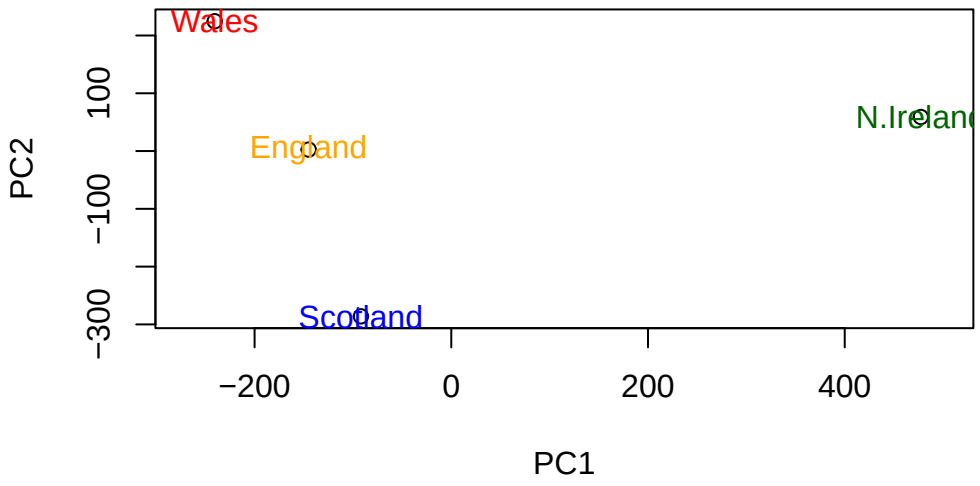



```
##using ggplot
library(ggplot2)
```

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.

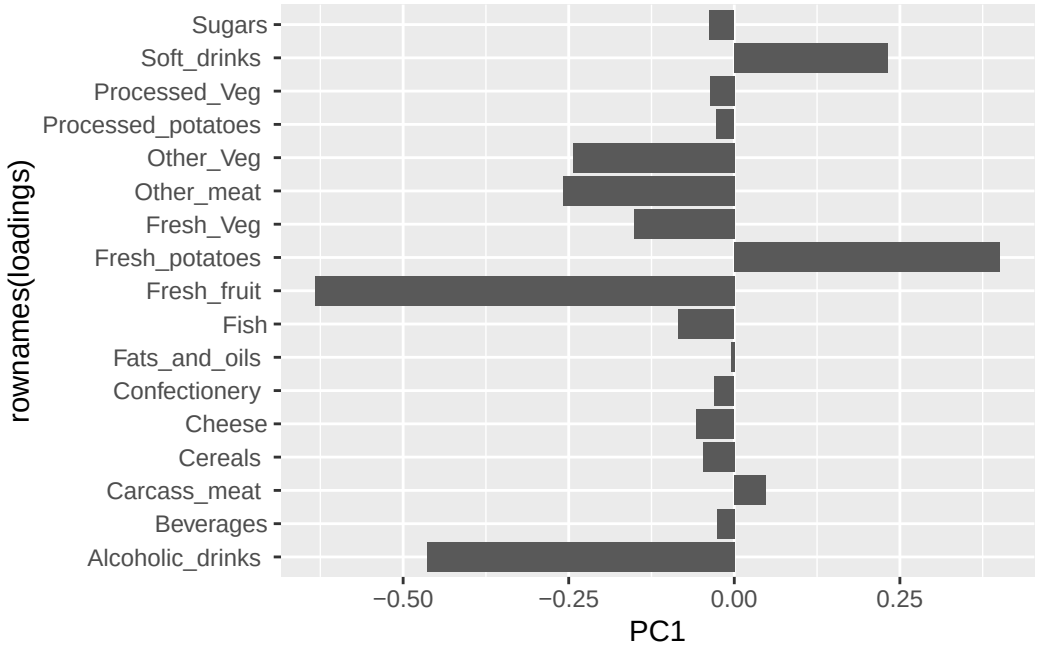
A: Yes, see below, added `col=mycols` to `text()`

```
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=mycols)
```



Lets look at how the original variables contribute to our new axis of max variance, aka PCs!

```
loadings <- as.data.frame(pca$rotation)

ggplot(loadings) +
  aes(PC1, rownames(loadings)) +
  geom_col()
```



Based on this, we can see that soft drinks and fresh potatoes are what N. Ireland had more of, and the negative stuff is what England has more of.

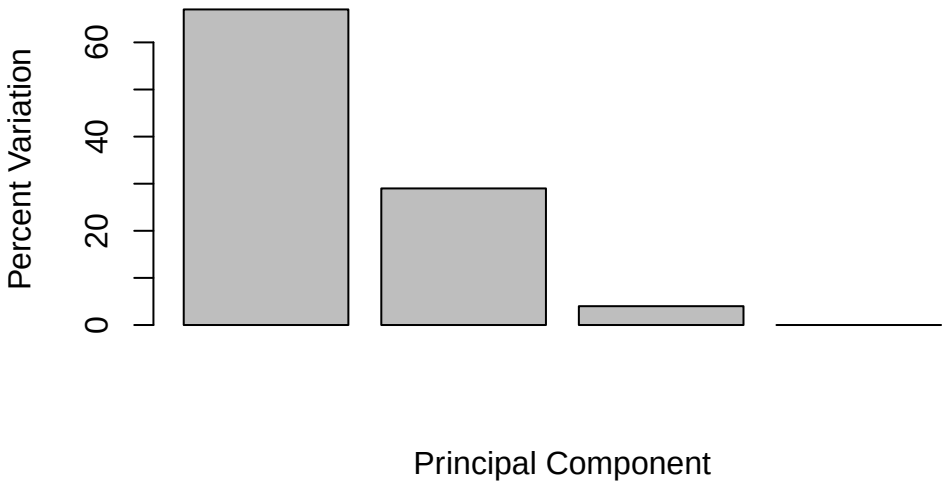
```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

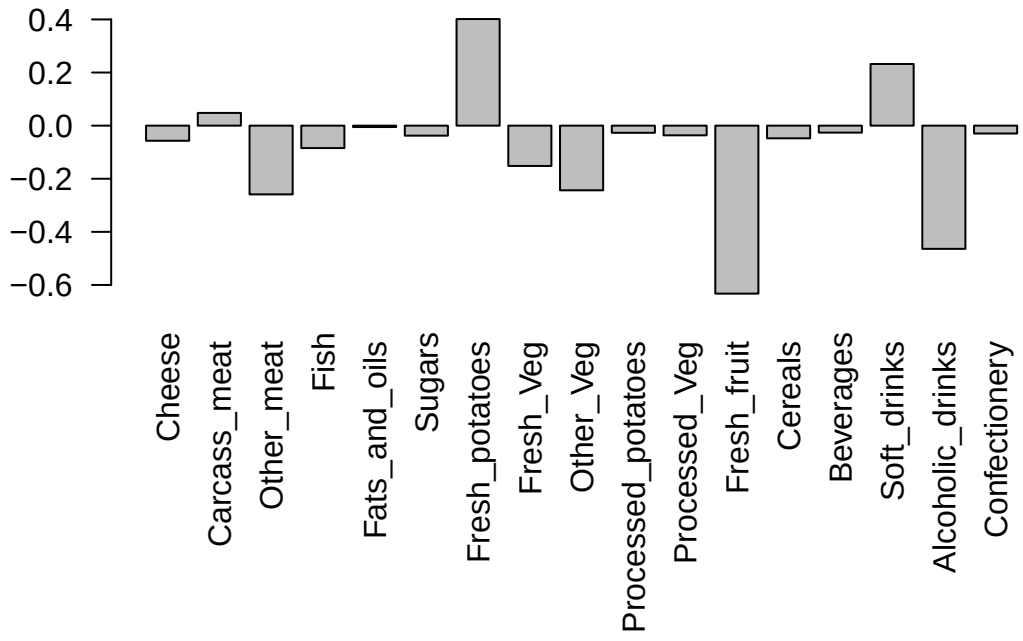
```
z <- summary(pca)
z$importance
```

```
                             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
```

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

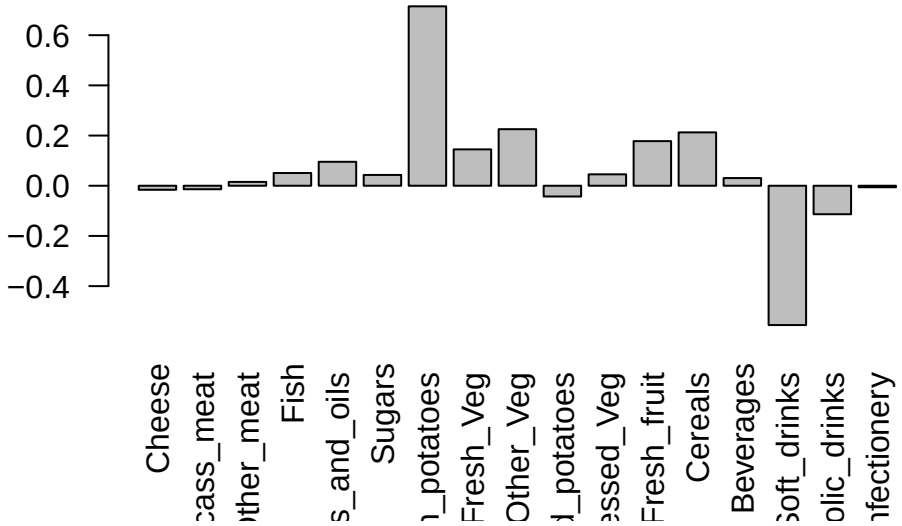



```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

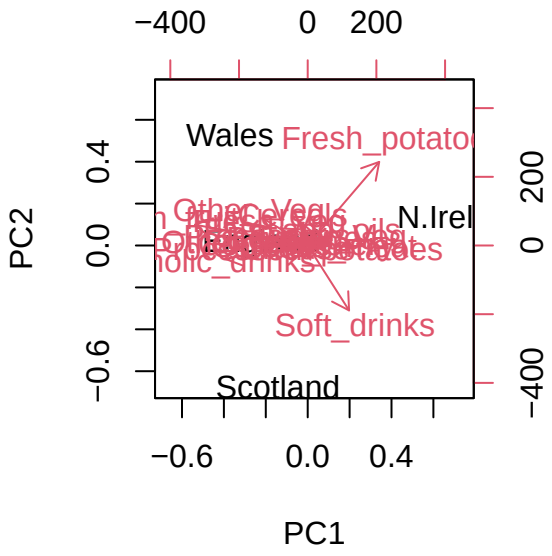
```
barplot( pca$rotation[,2], las=2 )
```



A: The two food groups that are featured prominently for PC2 are are soft drinks and fresh potatoes.

look at biplot

```
biplot(pca)
```



PCA of RNA Seq Data

```
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

```
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)
```

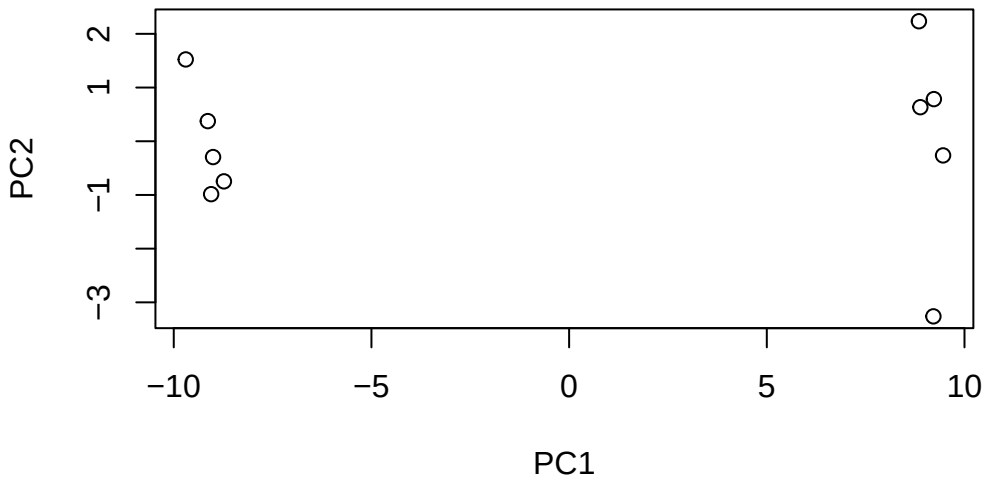
```
[1] 100
```

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

```
[1] 10
```

A: There are 100 genes and 10 samples for each.

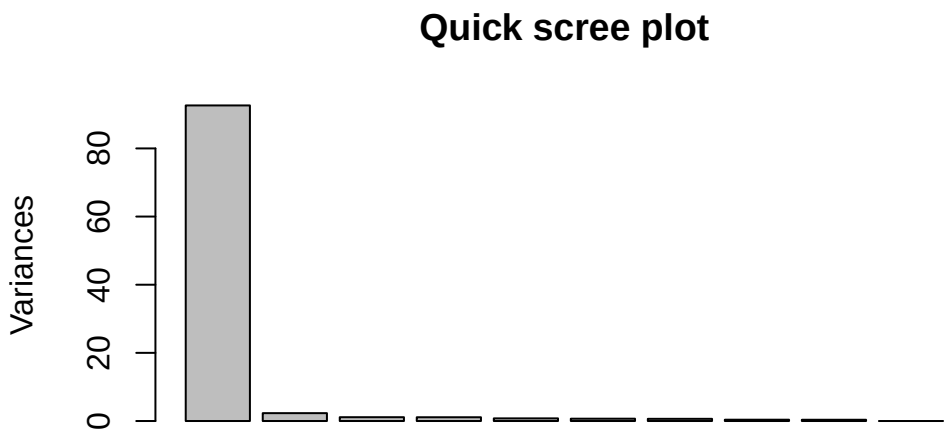
```
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

## 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
```

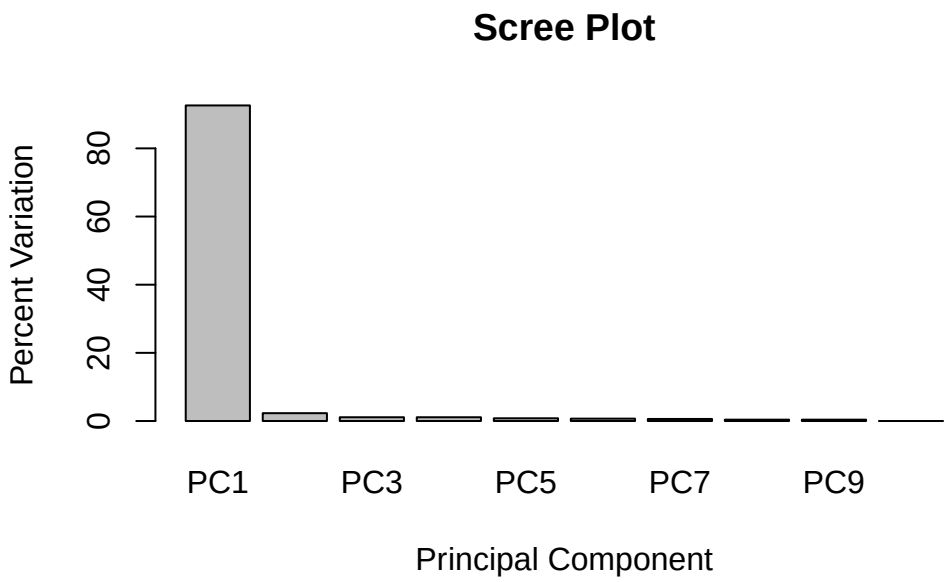
```
plot(pca, main="Quick scree plot")
```



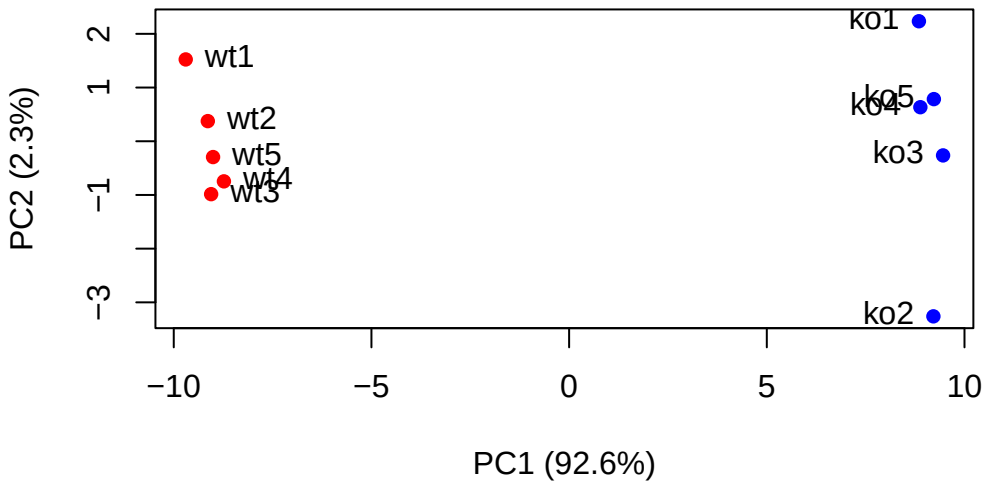

```
## Variance captured per PC
pca.var <- pca$sdev^2

## 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
```

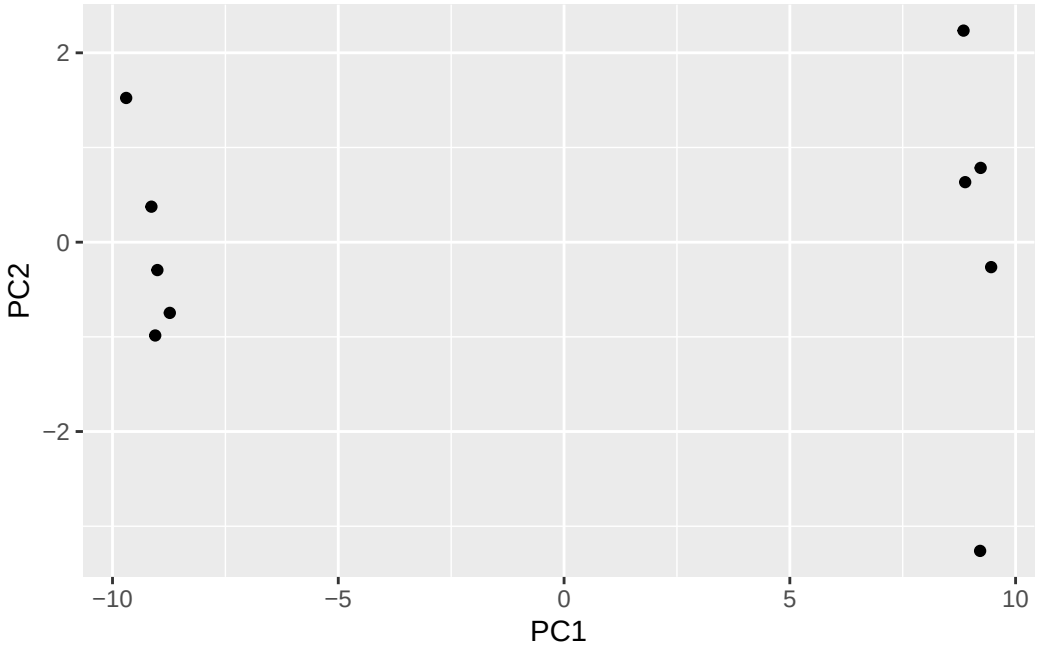
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



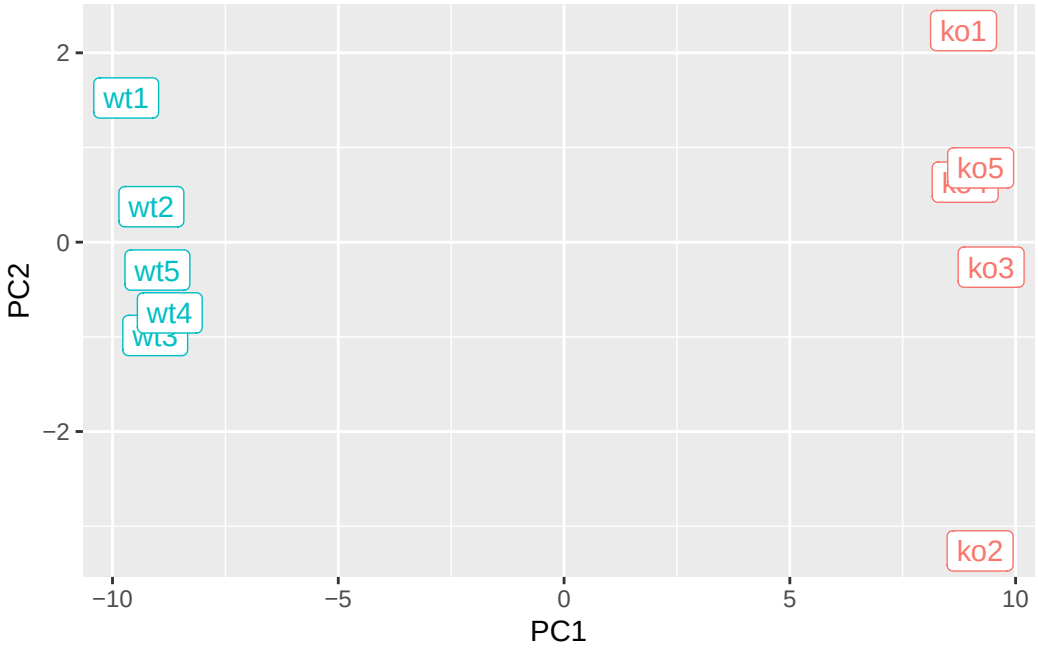

```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```



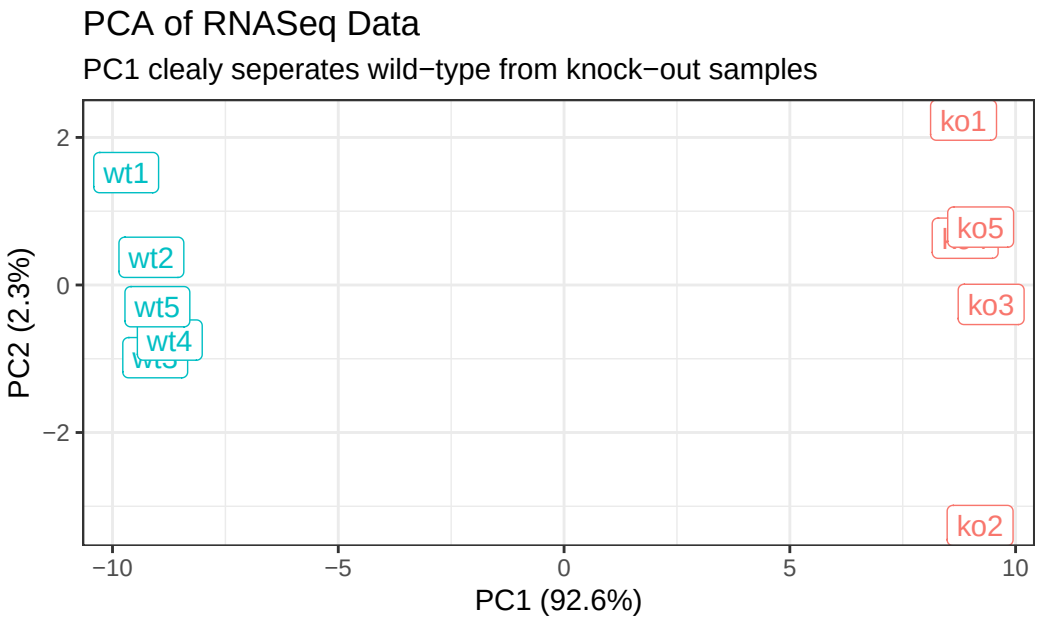

```
library(ggplot2)

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

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




```
# 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
```




```
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="Class example data") +
     theme_bw()
```



Optional

```
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"  "gene21"
[8] "gene56"  "gene10"  "gene90"
```