TRUMAN STATE UNIVERSITY

Department of STATISTICS

Multivariate Visualizations

DISTINCT BY DESIGN

Good Design

Excellent graphics should...

- show the data,
- emphasize the *substance* of the data,
- avoid distortion,
- strive for high information density,
- make large data sets coherent,
- encourage comparison,
- reveal several layers of detail,
- serve a clear purpose, and
- be integrated with statistical and verbal description.
- "Graphical excellence is nearly always multivariate."
  Tufte, *The Visual Display of Quantitative Information*

How can we display multivariate data?

- Use more dimensions (but we only have three).
- Show *projections* of the data into two dimensions.
- Map variables to other aesthetics (color, size, even time...)
- Unfold along common axes.
- Plot multiple dimension axes in two dimensions.
- "Iconify" multiple dimensions at a single point.
- Use mathematical dimension-reduction algorithms.

Example: 1959 Nutrition

In 1959, the *The Yearbook of Agriculture*, published by the United States Department of Agriculture, included nutritional information on 3 oz. servings of "meat, fish and fowl."

```r
library(cluster.datasets)
data("nutrients.meat.fish.fowl.1959")
nutr.orig <- nutrients.meat.fish.fowl.1959
nutr <- nutr.orig %>% column_to_rownames("name") %>% scale()
colnames(nutr) <- str_to_title(colnames(nutr))
rownames(nutr) <- str_to_title(rownames(nutr))
nutr.df <- as.data.frame(nutr)
head(nutr.df)
```

```
                   Energy    Protein        Fat    Calcium       Iron
Braised Beef     1.3101024  0.2352002  1.2897287 -0.4480464  0.1495365
Hamburger        0.3714397  0.4704005  0.3125618 -0.4480464  0.2179685
Roast Beef       2.1005553 -0.9408009  2.2668955 -0.4736761 -0.2610553
Beefsteak        1.6559256  0.0000000  1.6450621 -0.4480464  0.1495365
Canned Beef     -0.2708033  0.7056007 -0.3092717 -0.3455273  0.9022882
```

Adding a Factor...

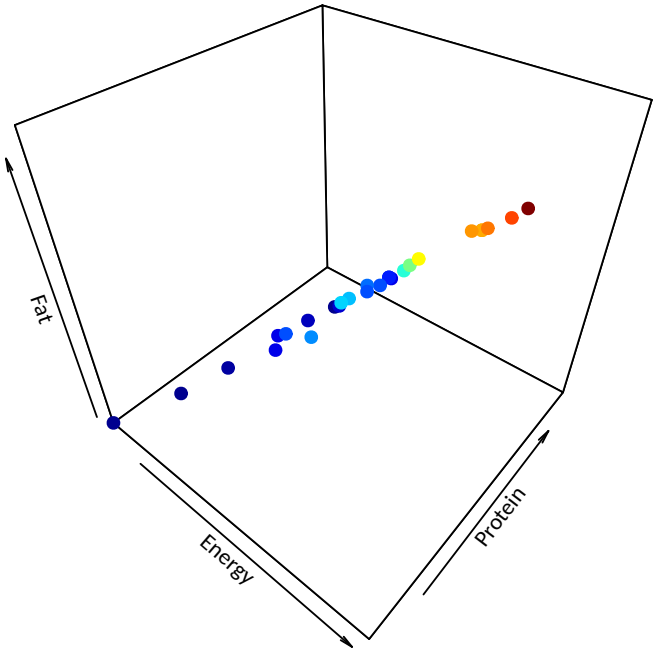
Note that I'm keeping this designation in a separate variable, since several routines below assume we have a completely numeric data frame.

```
nutr.type <- c(
  "Meat", "Meat", "Meat", "Meat", "Meat", "Fowl", "Fowl",
  "Meat", "Meat", "Meat", "Meat", "Meat", "Meat", "Meat",
  "Meat", "Fish", "Fish", "Fish", "Fish", "Fish", "Fish",
  "Fish", "Fish", "Fish", "Fish", "Fish", "Fish"
)
```

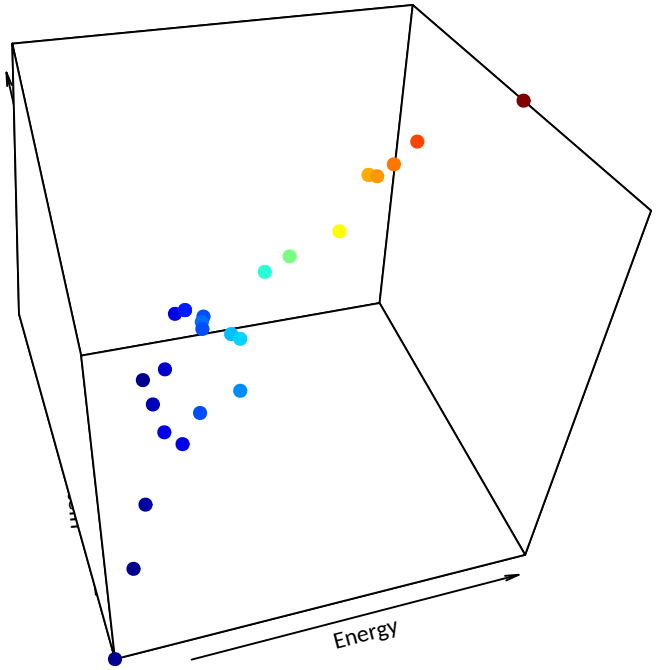
Three-Dimensional Graphs

Three-dimensional plots (projected to two) can have issues.

```r
library(plot3D)
with(nutr.df,
 scatter3D(x=Energy, y=Protein, z=Fat
  xlab="Energy", ylab="Protein",
  zlab="Fat", pch=19, colkey=FALSE))
```

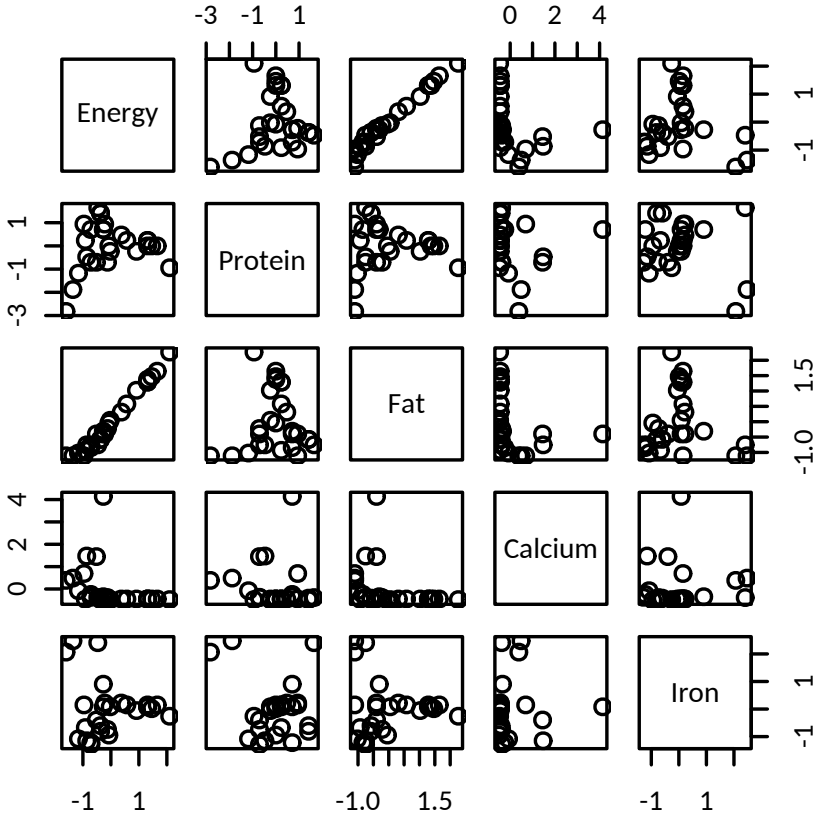



```r
# Rotation can help.
with(nutr.df,
 scatter3D(x=Energy, y=Protein, z=Fat,
  theta=-15, xlab="Energy", ylab="Prot
  zlab="Fat", pch=19, colkey=FALSE))
```

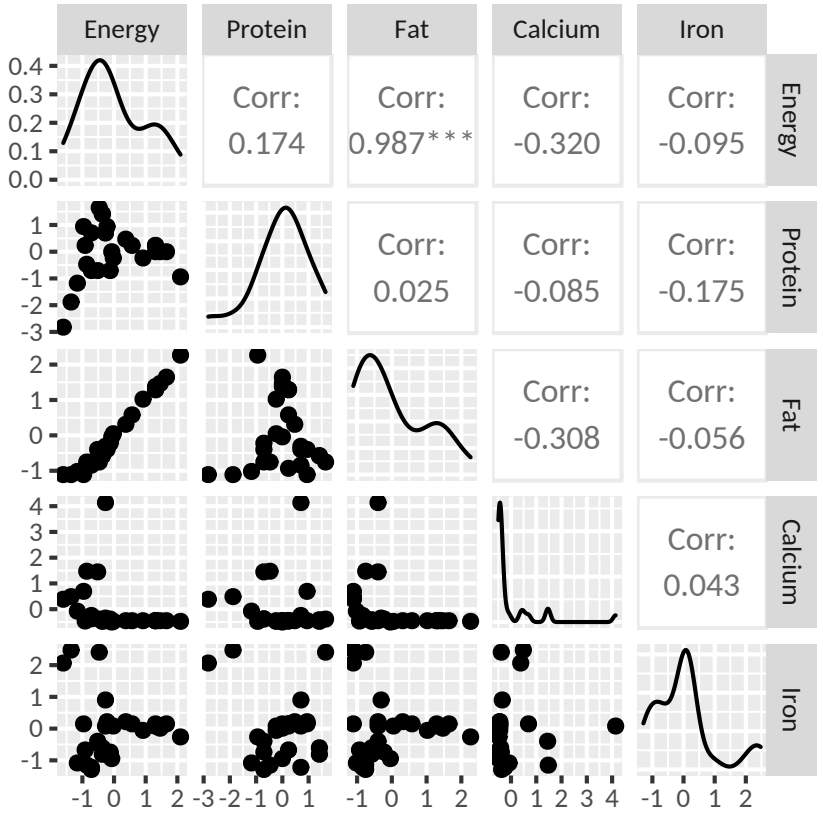


Paired scatterplots can be an alternative.

```r
# From base R:
pairs(nutr.df)
```

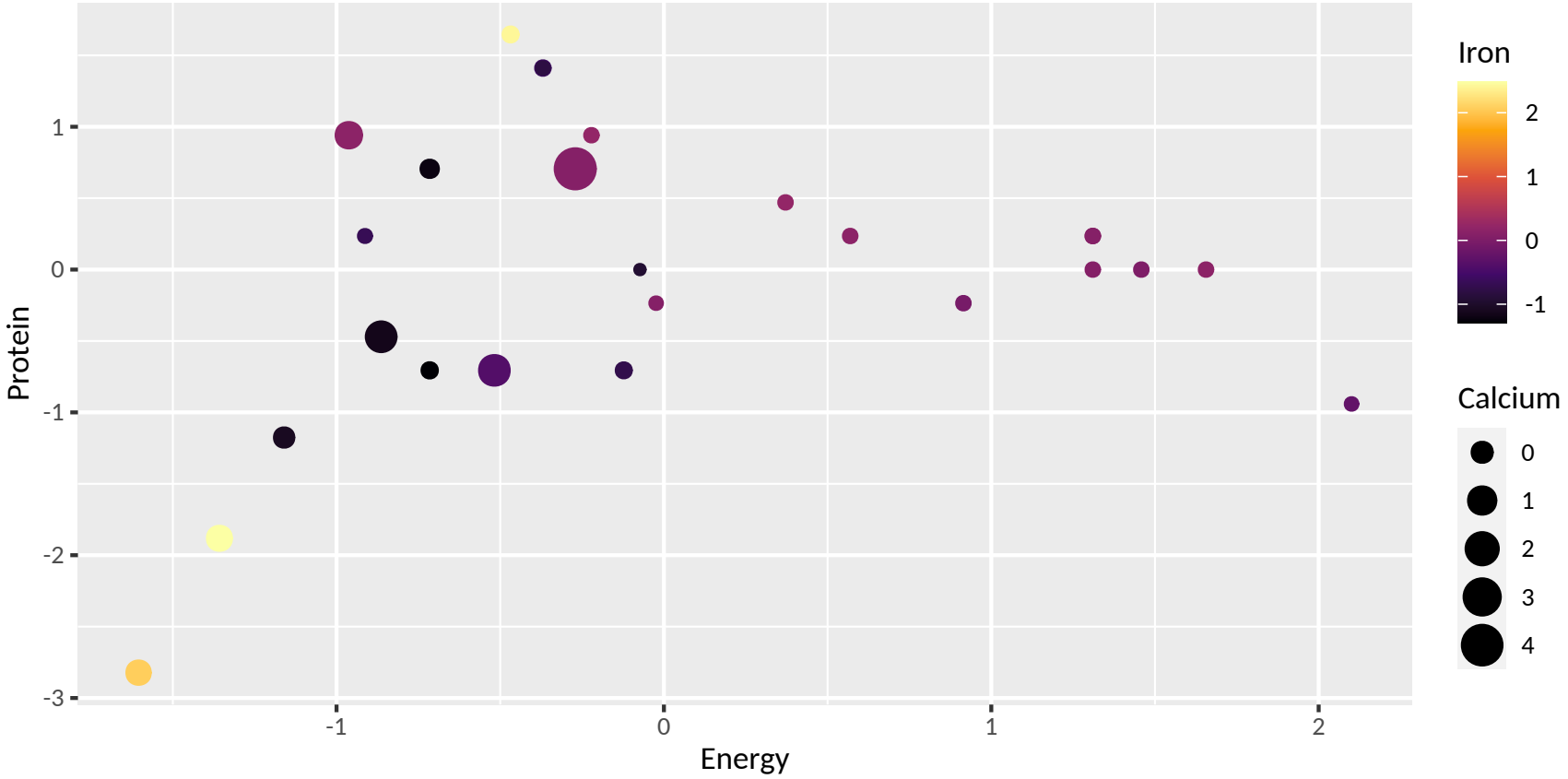



```r
library(GGally)
ggpairs(nutr.df)
```



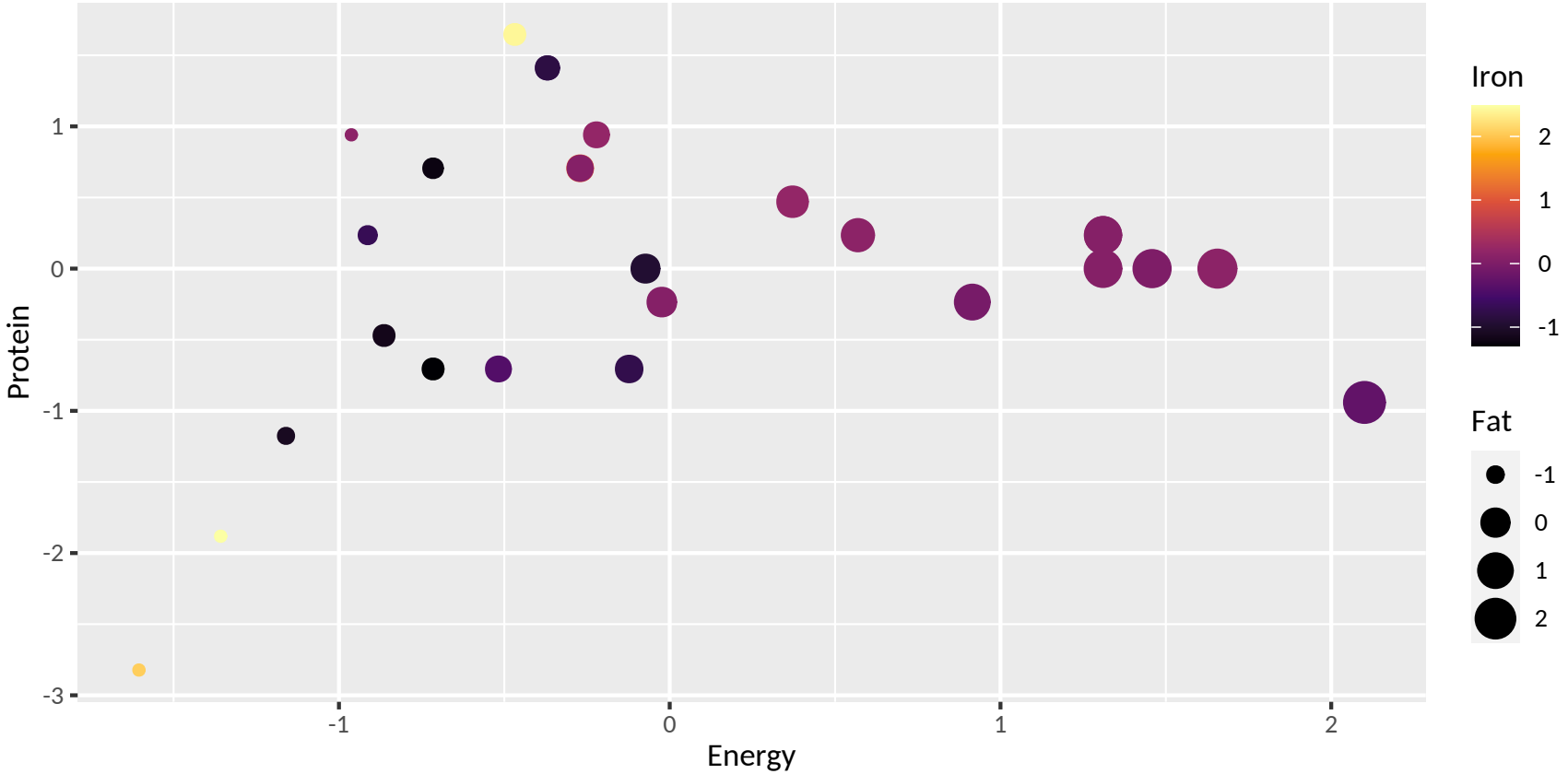
Variables can be mapped to other aesthetics.

```
ggplot(nutr.df) +
  geom_point(aes(x=Energy, y=Protein, color=Iron, size=Calcium)) +
  scale_color_viridis_c(option="B")
```

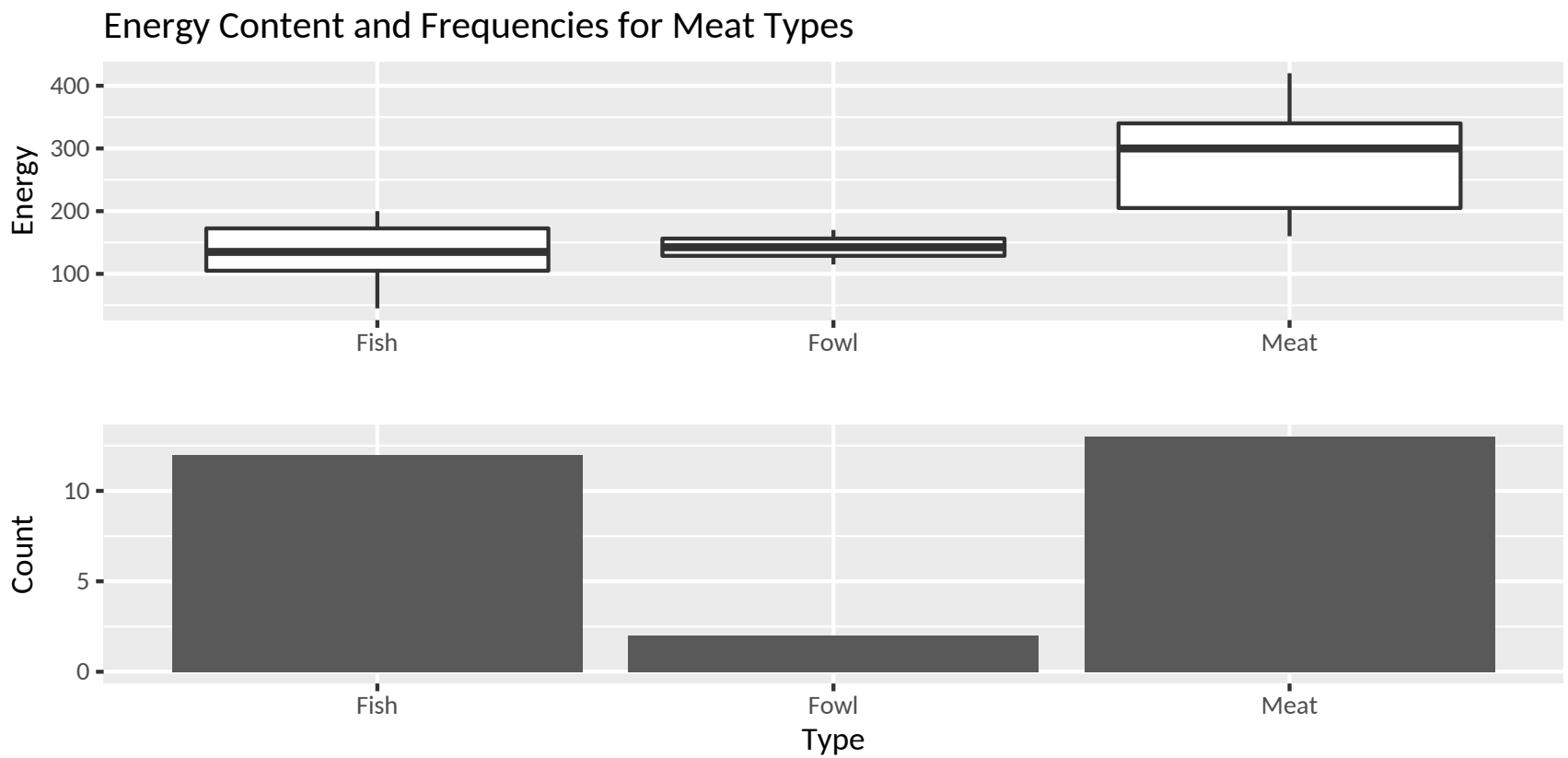


Some mappings may be more natural than others.

```
ggplot(nutr.df) +
  geom_point(aes(x=Energy, y=Protein, color=Iron, size=Fat)) +
  scale_color_viridis_c(option="B")
```

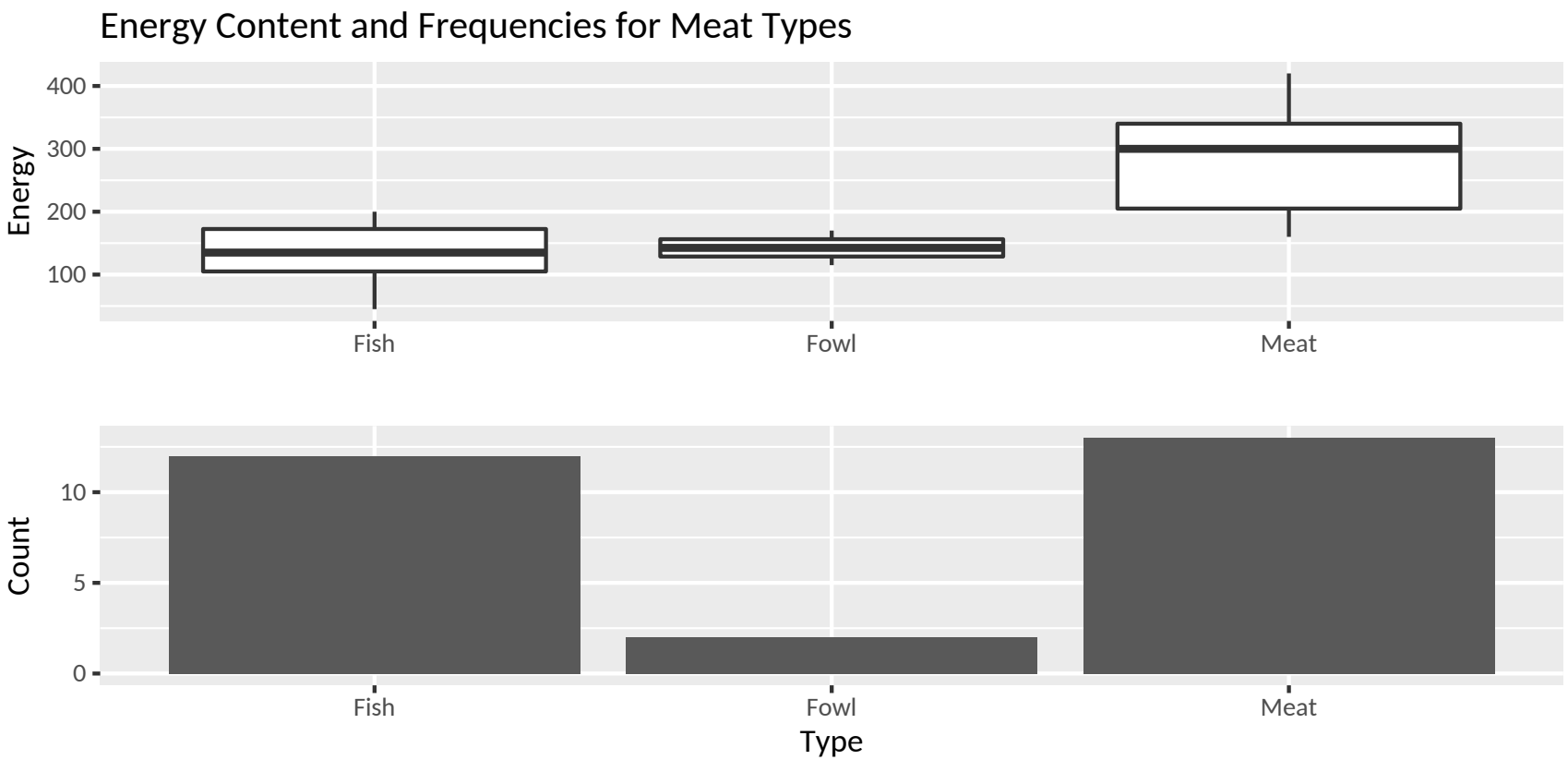


Multiple plots can be aligned along common axes.




```r
ggplot(nutr.df) +
  geom_point(aes(x=Energy, y=Protein, color=Iron, size=Calcium)) +
  scale_color_viridis_c(option="B")
```

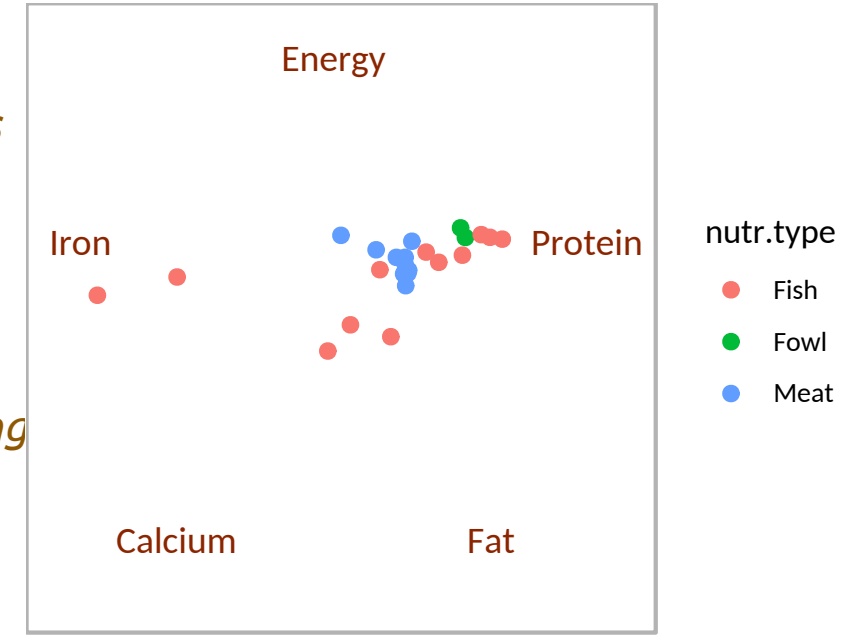
Multiple plots can be aligned along common axes.



Multiple Coordinate Axes in Two Dimensions

Radviz plots plot axes around a circle and map values to "spring constants" after rescaling all variables.

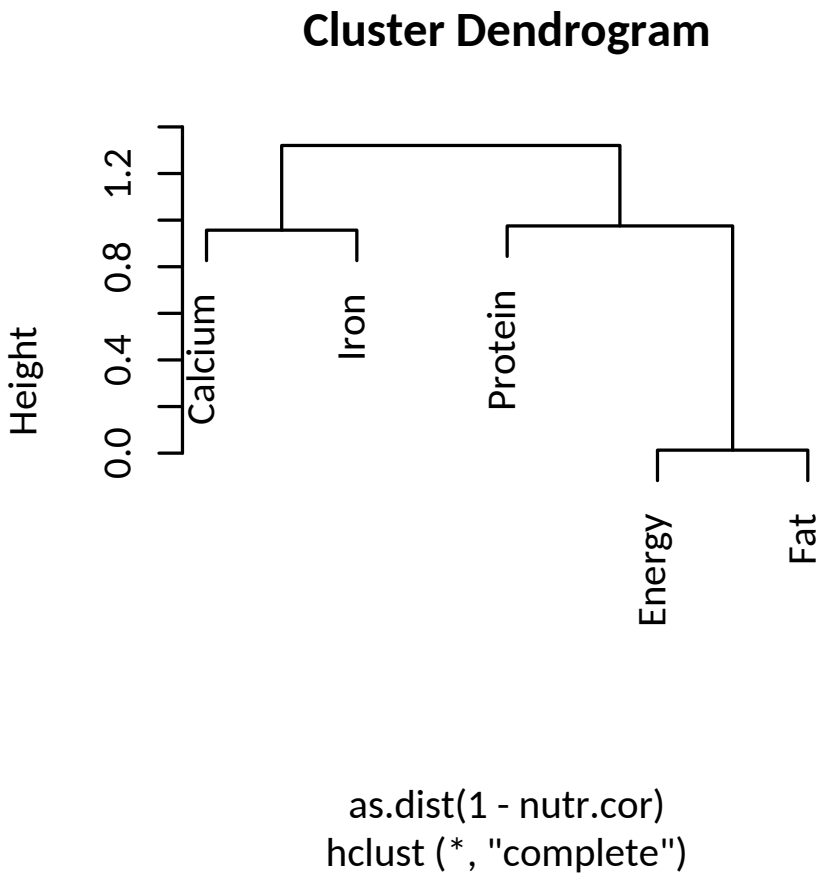
```r
library(Radviz)
# The `do.L` command rescales vectors
# to fall in the [0, 1] interval.
# map_dfc applies do.L to each col.
nutr.df01 <-
  map_dfc(nutr.df, do.L)
# Create the coordinates of the spring
nutr.springs <-
  make.S(names(nutr.df01))
# Create radviz coordinates.
nutr.radviz <-
  do.radviz(nutr.df01, nutr.springs)
# Plotting a radviz object
# creates a "ggplot" object.
plot(nutr.radviz) +
  geom_point(aes(color=nutr.type))
```



Radviz plotting is influenced by variable order. One suggestion: Cluster similar variables closer to each other.

This code defines a dissimilarity between variables as `1 - cor`.

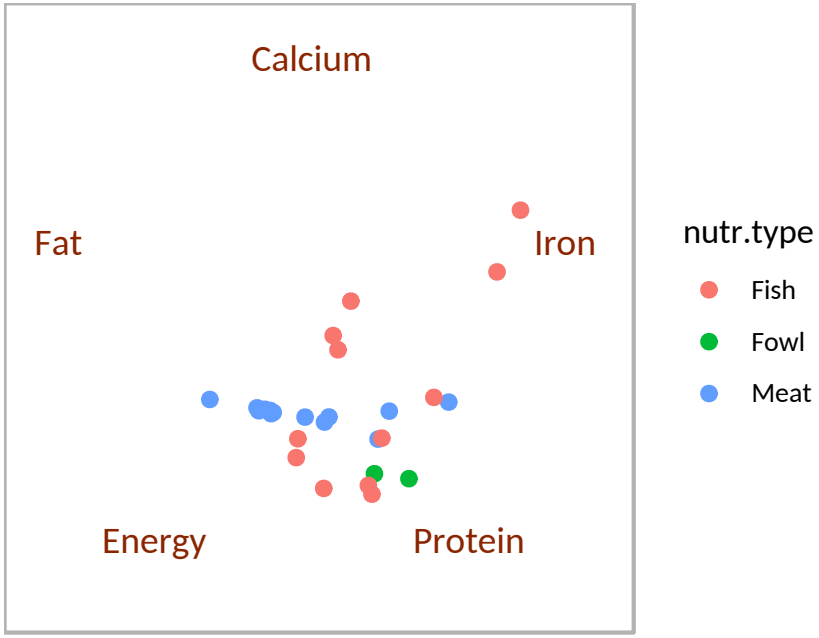
```
nutr.cor <- cor(nutr.df)
nutr.var.hclust <-
  hclust(as.dist(1-nutr.cor))
nutr.ord <-
  nutr.df[, nutr.var.hclust$order]
plot(nutr.var.hclust)
```



Radviz plotting is influenced by variable order. One suggestion: Cluster similar variables closer to each other.

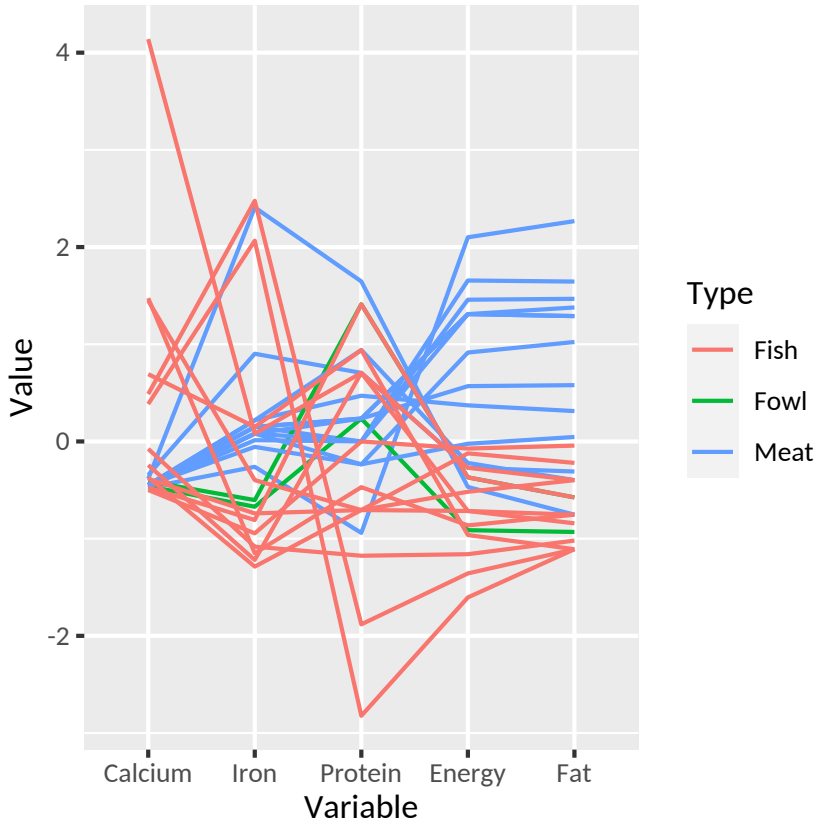
This code defines a dissimilarity between variables as `1 - cor`.

```
nutr.cor <- cor(nutr.df)
nutr.var.hclust <-
  hclust(as.dist(1-nutr.cor))
nutr.ord <-
  nutr.df[, nutr.var.hclust$order]
nutr.ord01 <-
  map_dfc(nutr.ord, do.L)
nutr.springs <-
  make.S(names(nutr.ord01))
nutr.radviz <-
  do.radviz(nutr.ord01, nutr.springs)
plot(nutr.radviz) +
  geom_point(aes(color=nutr.type))
```



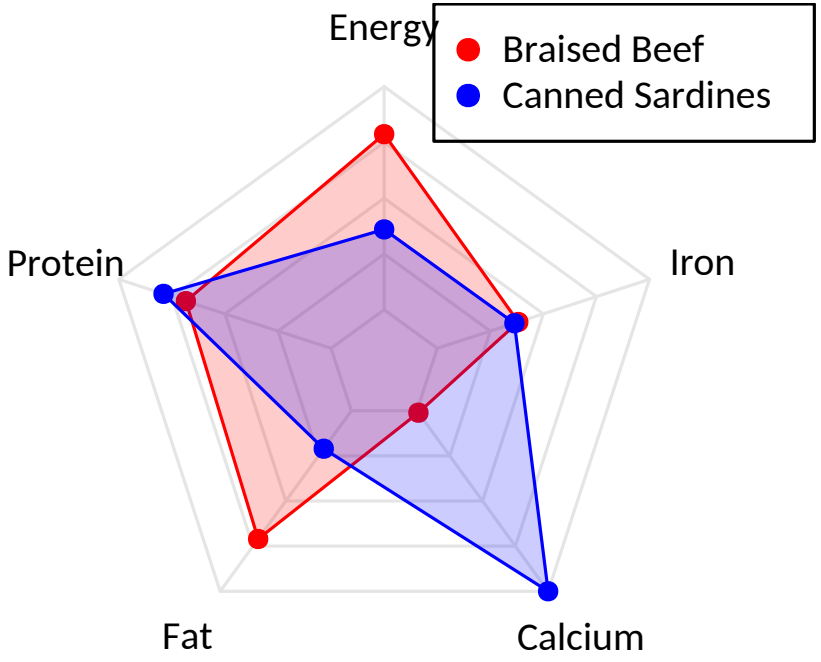
Parallel coordinate plots plot coordinate axes sequentially. Easiest comparisons are between neighboring variables.

```
library(GGally)
# Unlike some others, `ggparacoord`
# likes to have a label column.
# So we'll add one.
nutr.ord %>%
  bind_cols(Type=nutr.type) %>%
  ggparcoord(columns=1:5,
             groupColumn=6) +
  xlab("Variable") + ylab("Value")
```



Radar plots use radial axes and a polygon to represent a single data point.

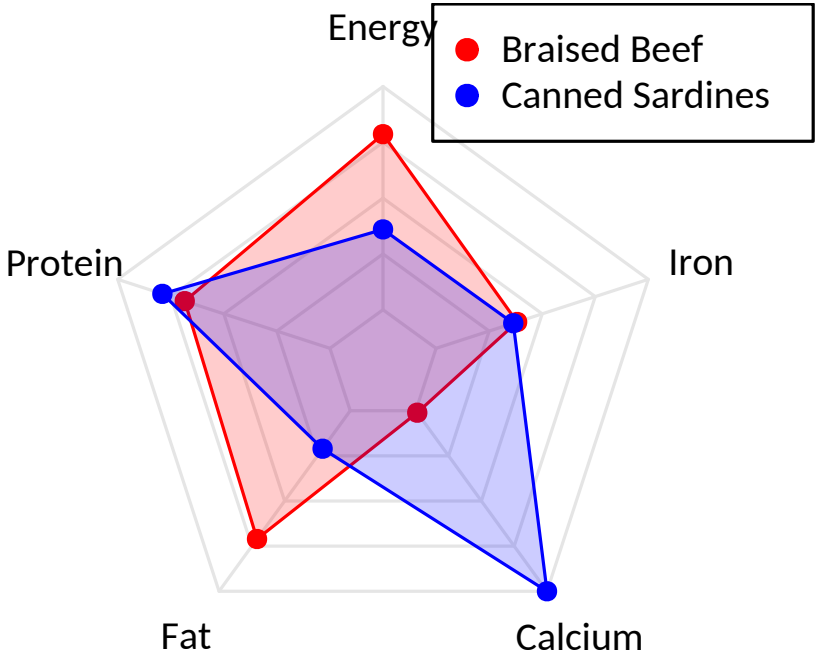
```r
library(fmsb)
# The `radarchart` needs max and min
# in rows 1 and 2 of the data frame.
nutr.rad <- rbind(rep(1, 5), rep(0, 5),
                  nutr.df01)
# Colors for line and fill.
col.vec <- c(rgb(1, 0, 0, 1),
             rgb(0, 0, 1, 1))
fill.vec <- c(rgb(1, 0, 0, 0.2),
              rgb(0, 0, 1, 0.2))
radarchart(nutr.rad[c(1:2, 3, 27),],
           plty=1, cglty=1,
           cglcol="grey90",
           pcol=col.vec, pfcol=fill.vec)
legend("topright",
  legend=rownames(nutr.df)[c(1,25)],
  pch=19, col=col.vec)
```

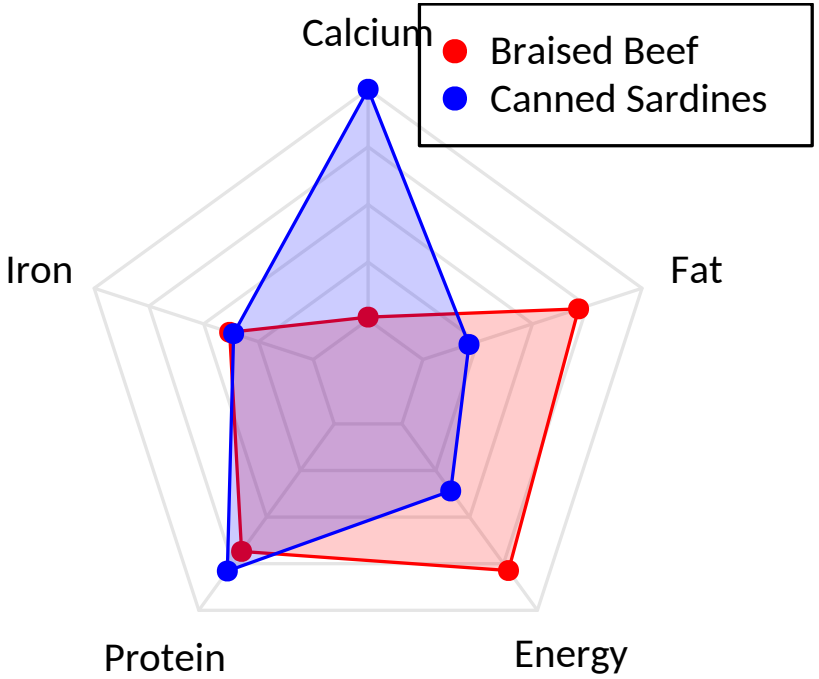


Install with

```
devtools::install_github("ricardo-bion/ggradar", dependencies = TRUE)
```

Radar plot shapes can depend on category organization.

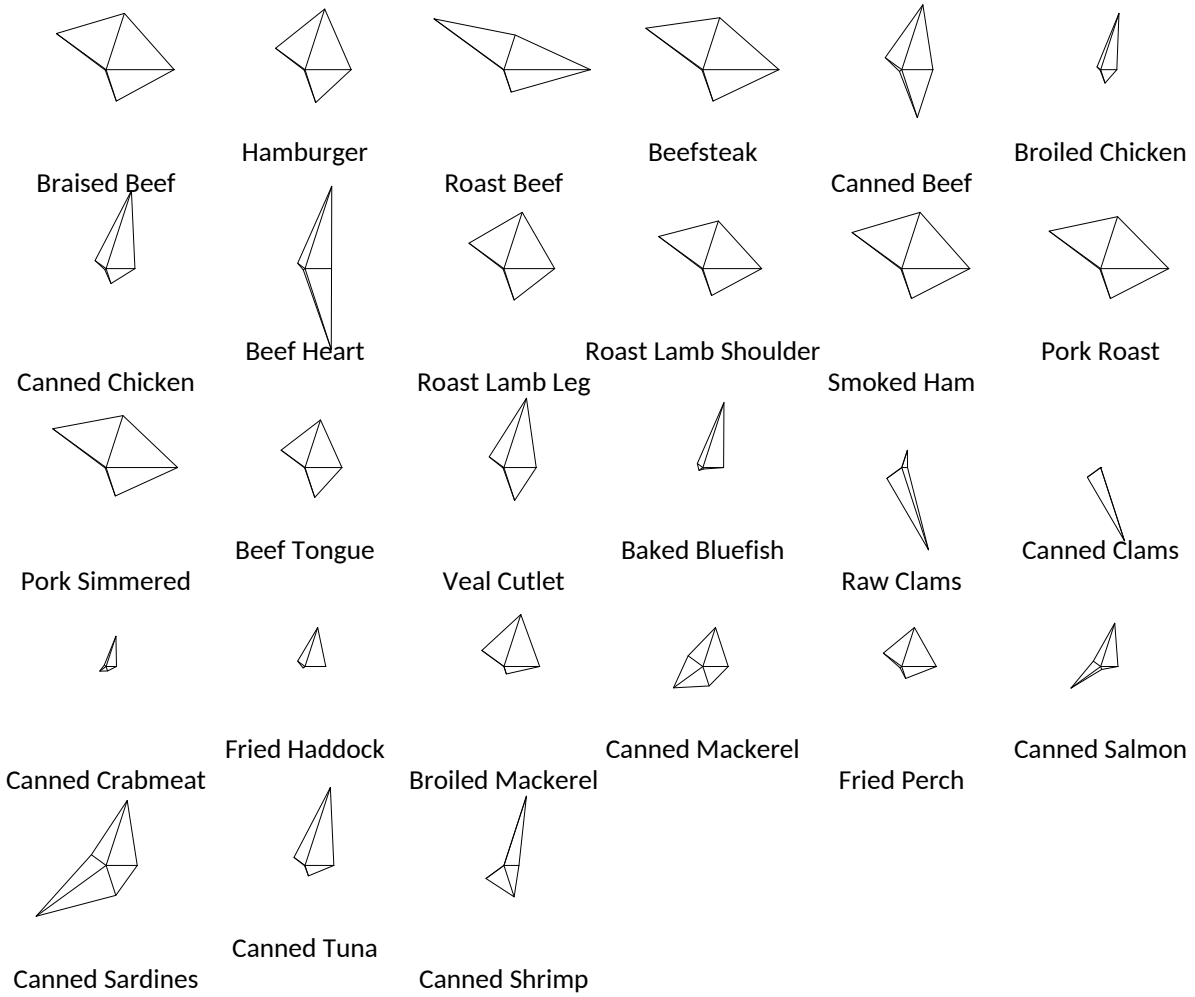




“Iconifying” Dimensions at Each Point

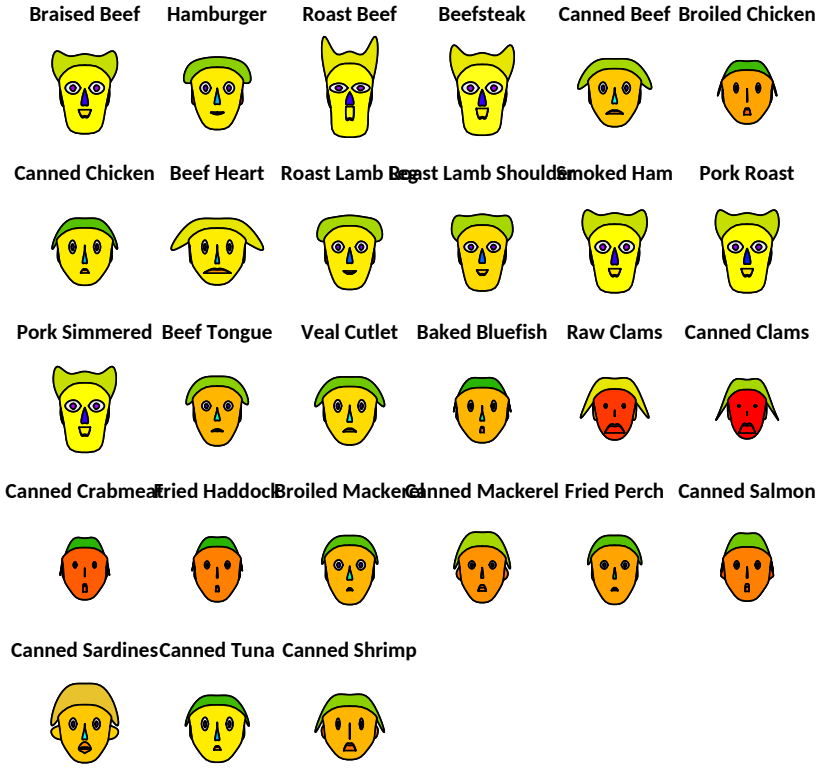
Panels of radar, or star, plots can help the search for similarity.

```
stars(nutr.df, labels=rownames(nutr.df), ncol=6, nrow=5, flip.labels=TRUE)
```



Chernoff faces allow the eye to find multi-dimensional similarities because humans are good at recognizing faces.

```r
library(aplpack)
faces(nutr.df, labels=rownames(nutr.df),
      nrow.plot=5, ncol.plot=6)
```

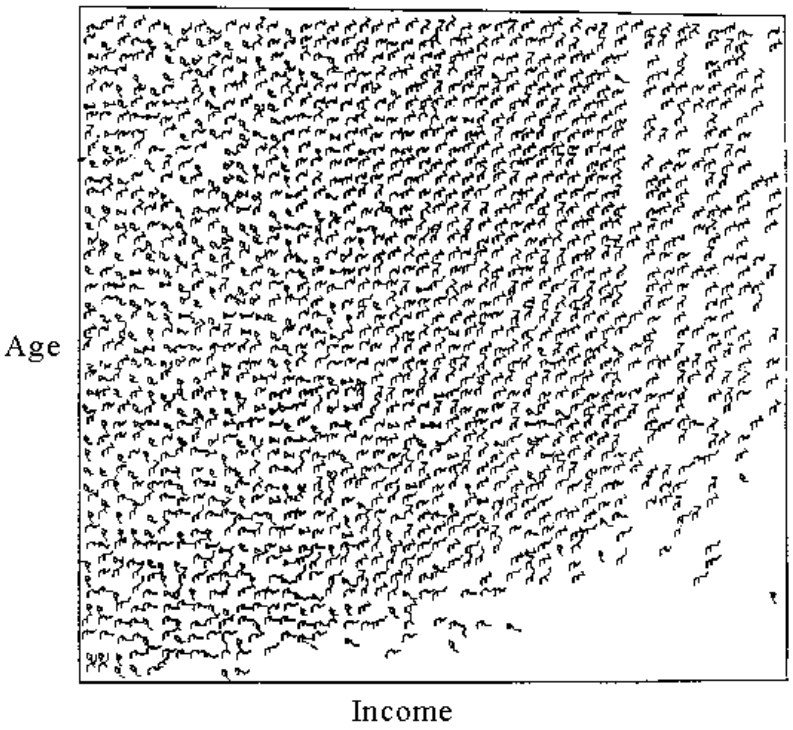



```
effect of variables:
 modified item        Var
 "height of face    " "Energy"
 "width of face     " "Protein"
 "structure of face" "Fat"
 "height of mouth   " "Calcium"
 "width of mouth    " "Iron"
 "smiling           " "Energy"
 "height of eyes    " "Protein"
 "width of eyes     " "Fat"
 "height of hair    " "Calcium"
 "width of hair    "  "Iron"
 "style of hair    "  "Energy"
 "height of nose   "  "Protein"
 "width of nose    "  "Fat"
 "width of ear     "  "Calcium"
 "height of ear    "  "Iron"
```

Zoom out, and these sort of plots become a texture.

Each stick shape represents…

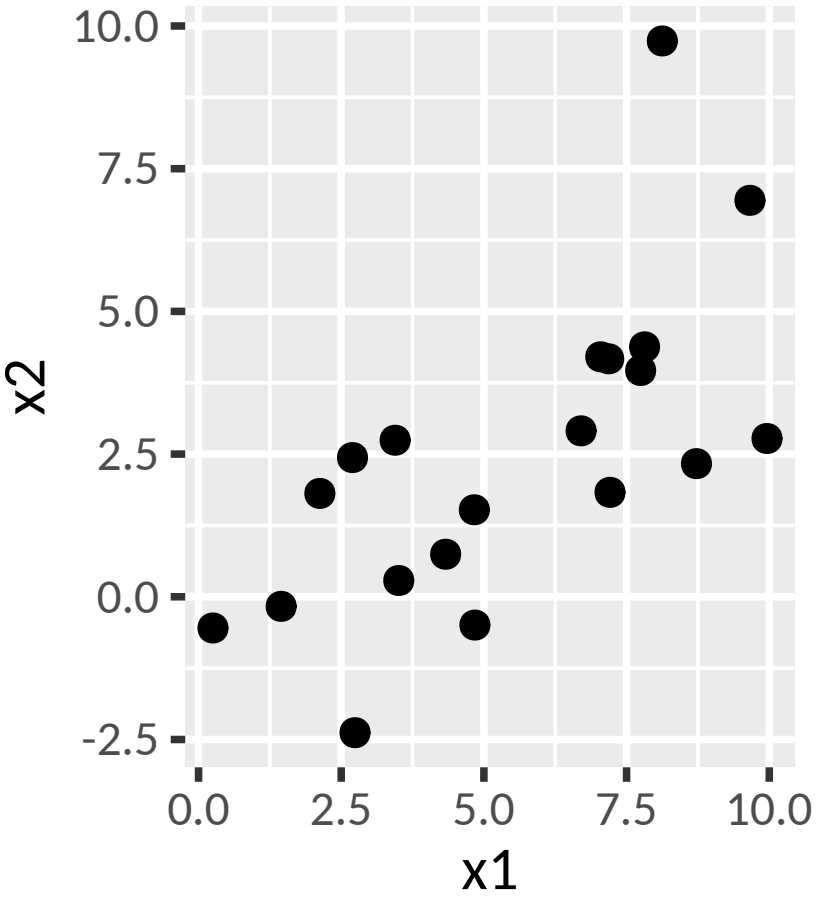
- Occupation
- Education
- Marital Status
- Male/Female



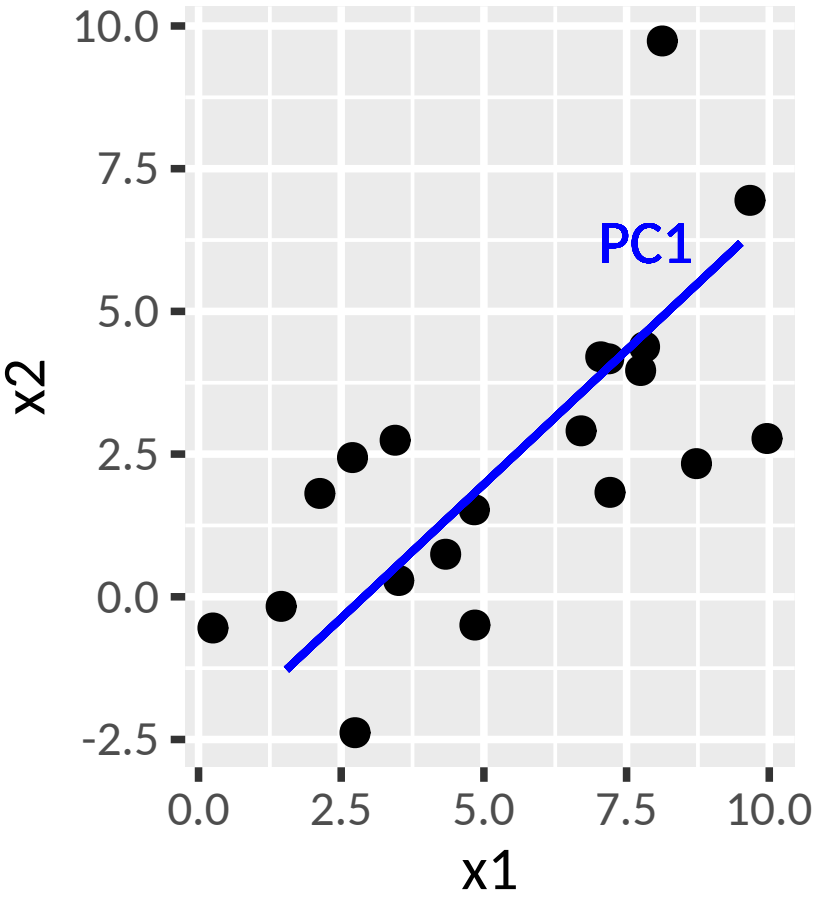
Dr. Yan Liu, Wright State University

Dimension Reduction

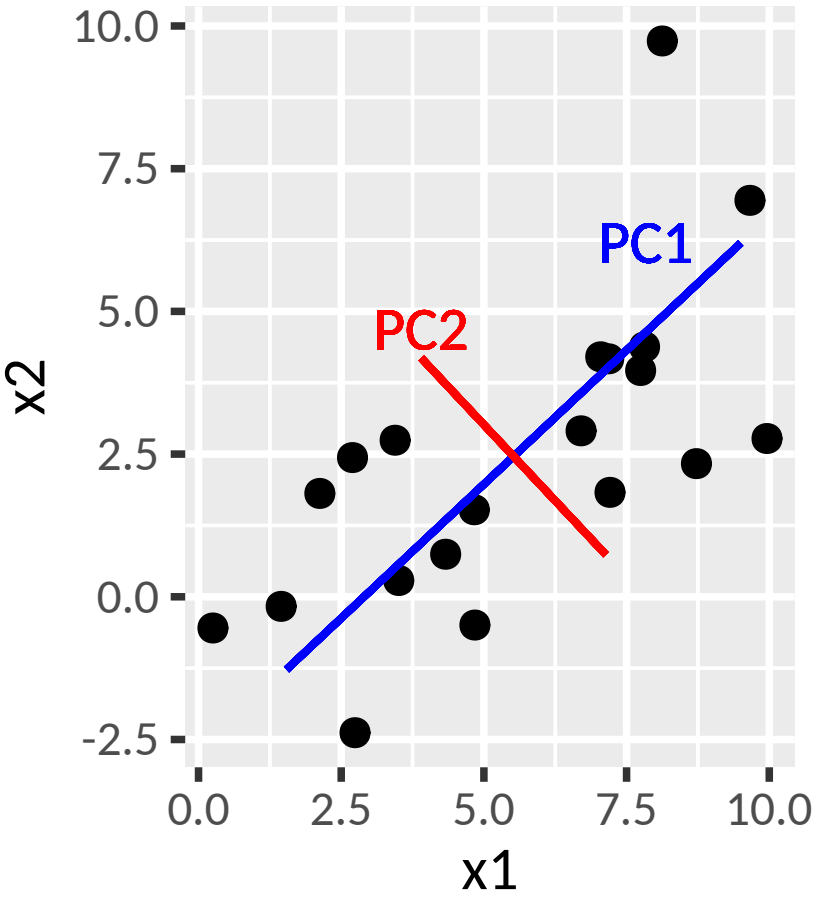
Principal components analysis identifies perpendicular directions in which the data has maximum variance.



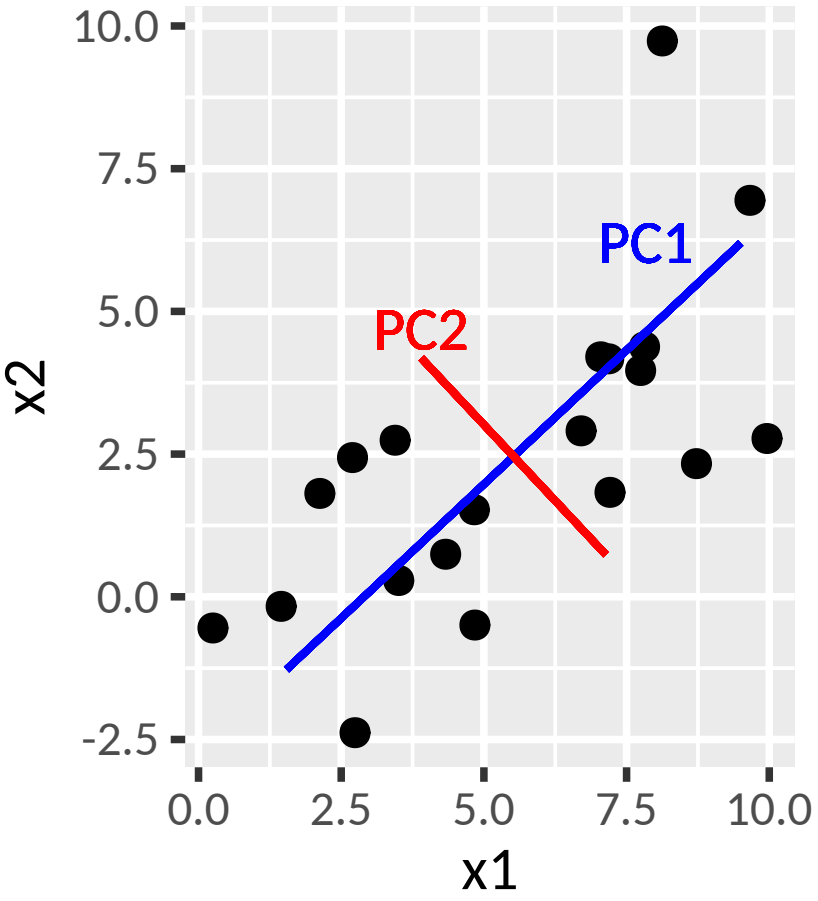
Principal components analysis identifies perpendicular directions in which the data has maximum variance.



Principal components analysis identifies perpendicular directions in which the data has maximum variance.



Principal components analysis identifies perpendicular directions in which the data has maximum variance.



This can be done by finding eigenvectors of the covariance matrix. (But let `prcomp` do it for you.)

```
cov(df)
```

```
          x1       x2
x1 8.197500 5.367189
x2 5.367189 7.518845
```

```
eigen(cov(df))
```

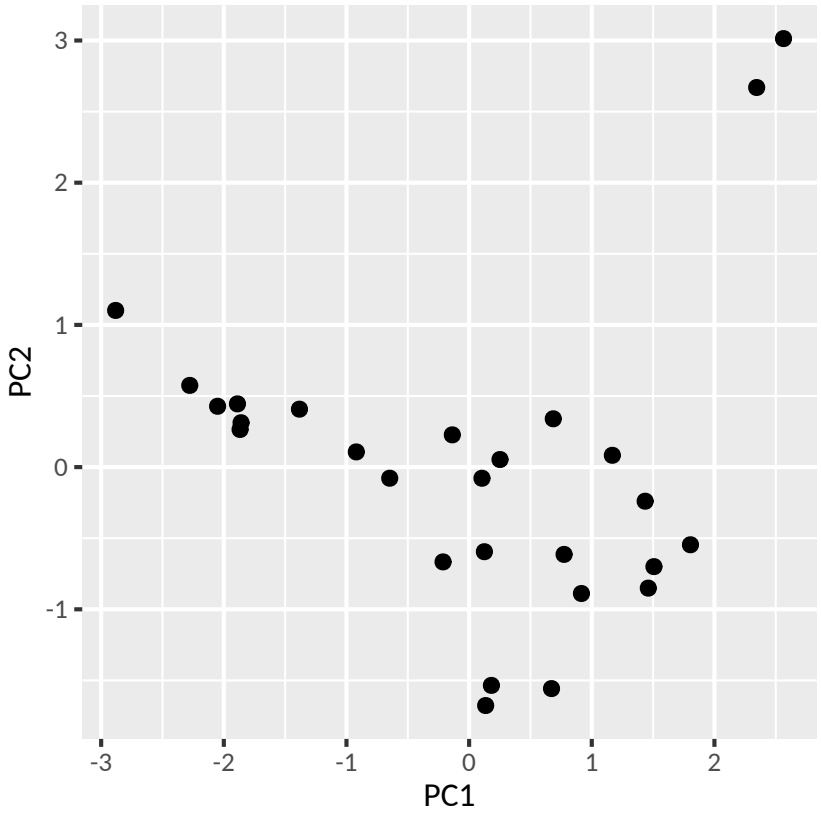
```
eigen() decomposition
$values
[1] 13.236078  2.480268

$vectors
           [,1]       [,2]
[1,] -0.7290736  0.6844353
[2,] -0.6844353 -0.7290736
```

PCA creates new "variables" that explain as much variance as possible.

Uses Include:

- Graphing
- Clustering/Prediction
- Data Compression

```
nutr.pca <- prcomp(nutr)
ggplot(as.data.frame(nutr.pca$x)) +
  geom_point(aes(x=PC1, y=PC2))
```



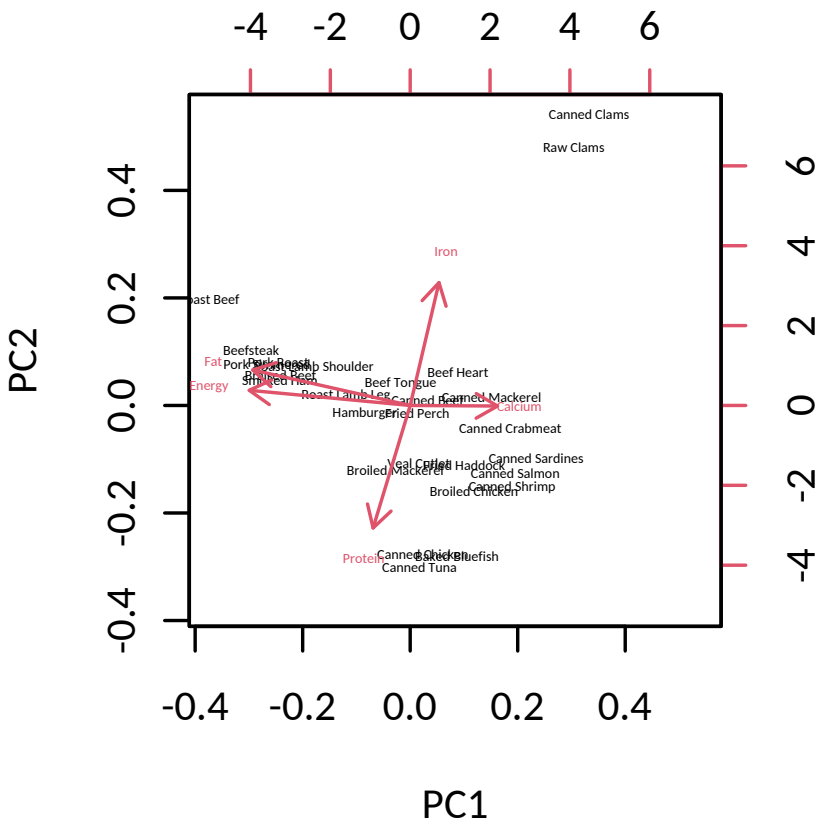
PCA creates new "variables" that explain as much variance as possible.

Uses Include:

- Graphing
- Clustering/Prediction
- Data Compression

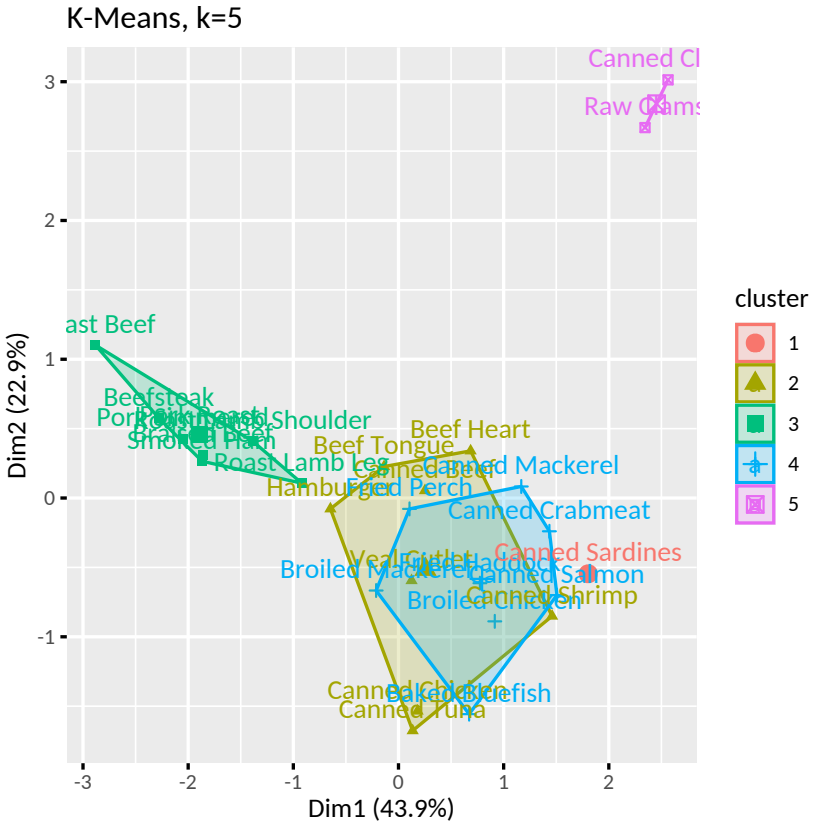
```
nutr.pca <- prcomp(nutr)
biplot(nutr.pca, cex=0.3)
# Sometimes also called "loadings"
nutr.pca$rotation
```

	PC1	PC2	PC3	PC4	PC5
Energy	-0.654	0.085	-0.151	-0.197	0.709
Protein	-0.151	-0.689	0.463	-0.526	-0.104
Fat	-0.640	0.200	-0.217	-0.132	-0.697
Calcium	0.355	-0.003	-0.651	-0.671	0.003
Iron	0.117	0.691	0.540	-0.466	0.010



Clustering can be visualized with principal components.

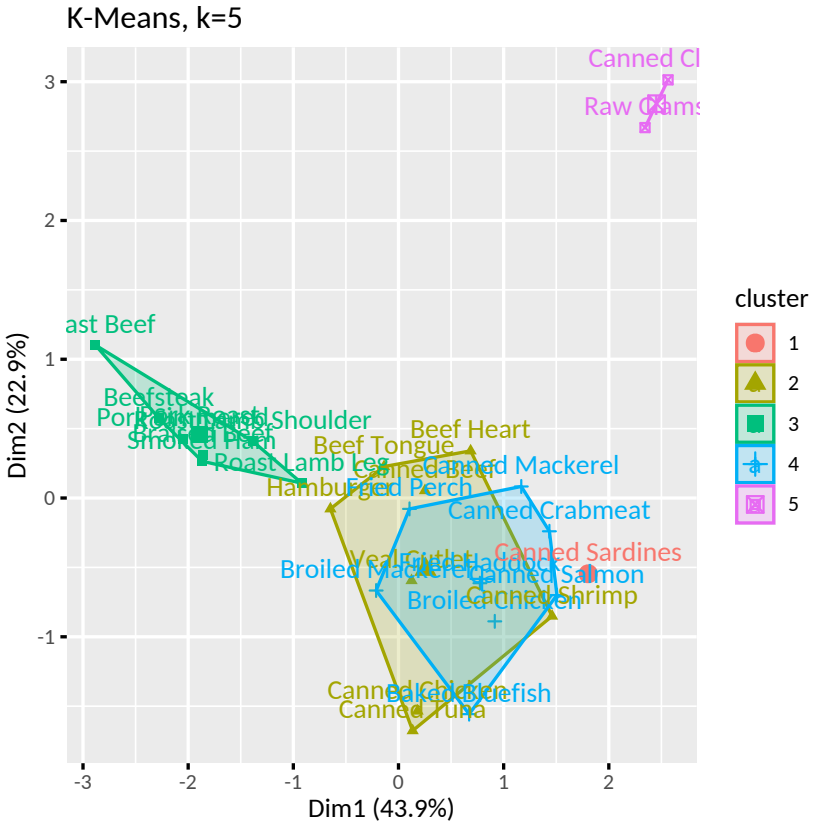
```
set.seed(329)
library(factoextra)
nutr.km <- kmeans(nutr, centers=5,
  nstart=20)
fviz_cluster(nutr.km, data=nutr,
  main="K-Means, k=5")
```



Clustering can be visualized with principal components.

```
set.seed(329)
library(factoextra)
nutr.km <- kmeans(nutr, centers=5,
  nstart=20)
fviz_cluster(nutr.km, data=nutr,
  main="K-Means, k=5")
```

Why do the factors overlap?



Multidimensional Scaling

Multi-dimensional scaling attempts to reduce dimensions while preserving distance between data points.

- Given a multi-dimensional data set $\vec{x}_1, \ldots, \vec{x}_n$, can we find a set of points $\vec{z}_1, \ldots, \vec{z}_n$ in a lower-dimensional Euclidean space (for example $\mathbb{R}^2$), such that $d_x(\vec{x}_i, \vec{x}_j) \approx ||\vec{z}_i, \vec{z}_j||$ for all i and j?
- If D is a *dissimilarity matrix* with entries $d_{ij} = d_x(\vec{x}_i, \vec{x}_j)$, can we find $\vec{z}_1, \ldots, \vec{z}_n$ that minimize the *stress function*

$$\text{stress} = \sum_{i<j} \left(d_{ij} - ||\vec{z}_i - \vec{z}_j||\right)^2?$$

- This idea can be applied even when the dissimilarities between the original $\vec{x}$'s is not a Euclidean distance, but some other measure.

Example: The Distance Matrix for Meat, Fish and Fowl.

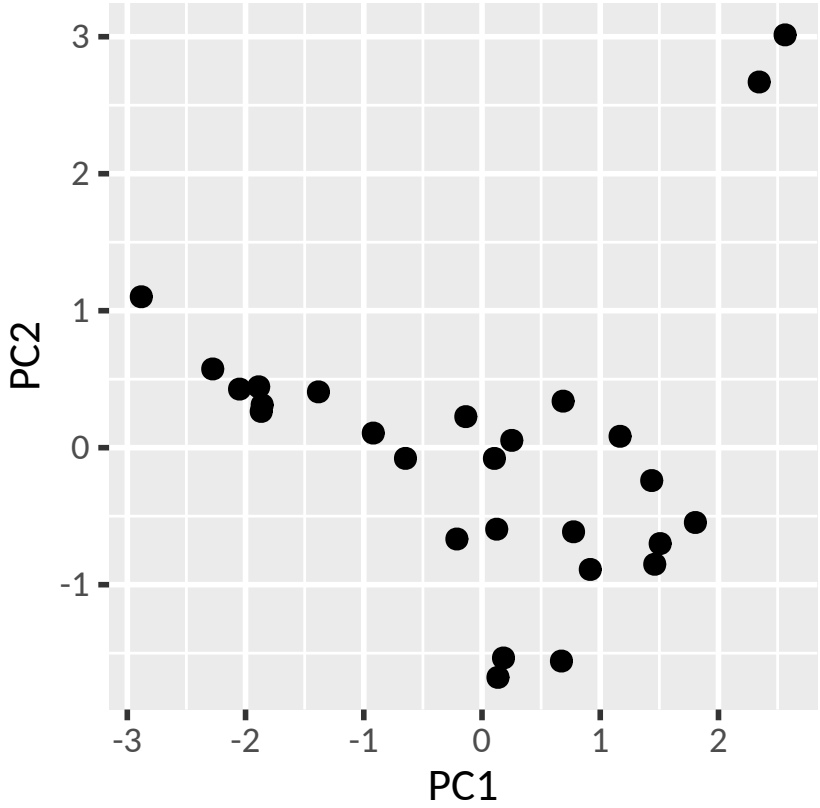
The algorithms take a distance matrix, not the original data.

```
nutr.dist <- dist(nutr)
nutr.dist
```

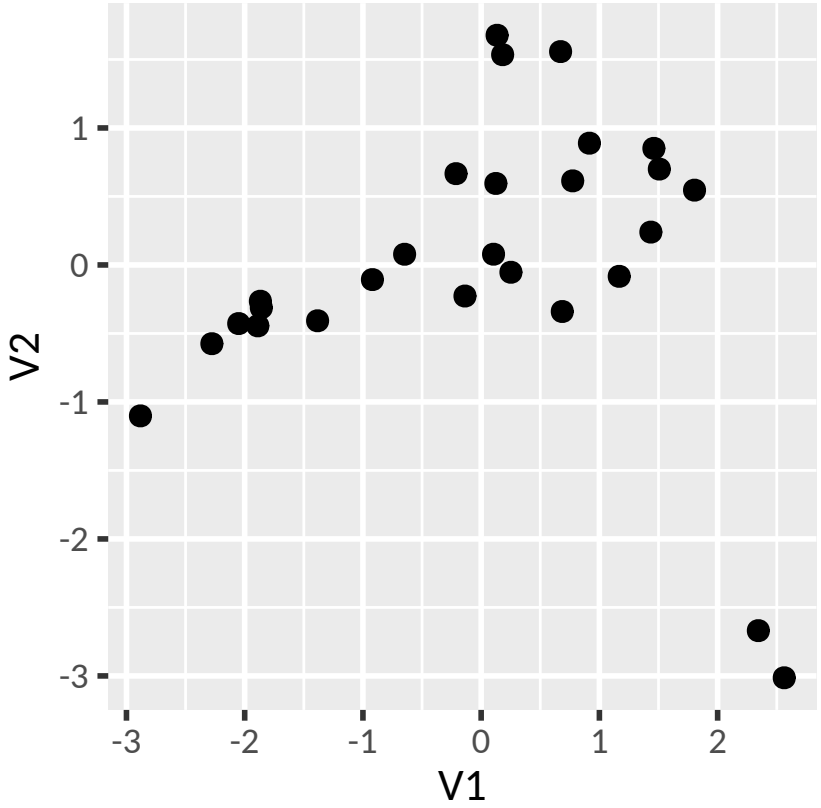
```
                    Braised Beef  Hamburger Roast Beef  Beefsteak Canned Beef
Hamburger             1.37693318
Roast Beef            1.76971526 3.00514082
Beefsteak             0.54879378 1.91086906 1.28012553
Canned Beef           2.41960469 1.15467556 4.04232701 2.93372154
Broiled Chicken       3.24792236 2.01084828 4.56735495 3.73712184  1.87376728
Canned Chicken        2.87273686 1.70270898 4.45371386 3.40506963  1.68798793
Beef Heart            3.79907517 2.83268851 5.43566380 4.25206933  1.84104857
Roast Lamb Leg        1.02674329 0.41240192 2.59740322 1.54044405  1.51423799
Roast Lamb Shoulder   0.70046358 1.17182323 1.86901537 1.01651211  2.23491927
Smoked Ham            0.06843197 1.38202525 1.75510108 0.55304390  2.44176050
Pork Roast            0.26056371 1.50248216 1.55460246 0.44192630  2.55553724
Pork Simmered         0.35718696 1.66684008 1.41857913 0.29891166  2.72876990
Beef Tongue           1.88483004 0.86283670 3.17173103 2.33214378  1.32783662
Veal Cutlet           2.38680488 1.03816317 4.03289606 2.93064368  0.73787182
Baked Bluefish        3.28400837 2.16274520 4.61285301 3.77137351  2.23441157
Raw Clams             4.86202685 4.06408614 5.71454639 5.14727391  3.41836469
```

“Classical” multidimensional scaling replicates PCA if you start with Euclidean distances.

```
# Note signs might vary.
ggplot(as.data.frame(nutr.pca$x)) +
  geom_point(aes(x=PC1, y=PC2))
```




```
nutr.cmds <- cmdscale(nutr.dist)
ggplot(as.data.frame(nutr.cmds)) +
  geom_point(aes(x=V1, y=V2))
```



Multidimensional scaling can be used when you start with a dissimilarity matrix, not the raw data.

The *Hamming distance* is a simple distance measure that counts at how many places two sequences differ.

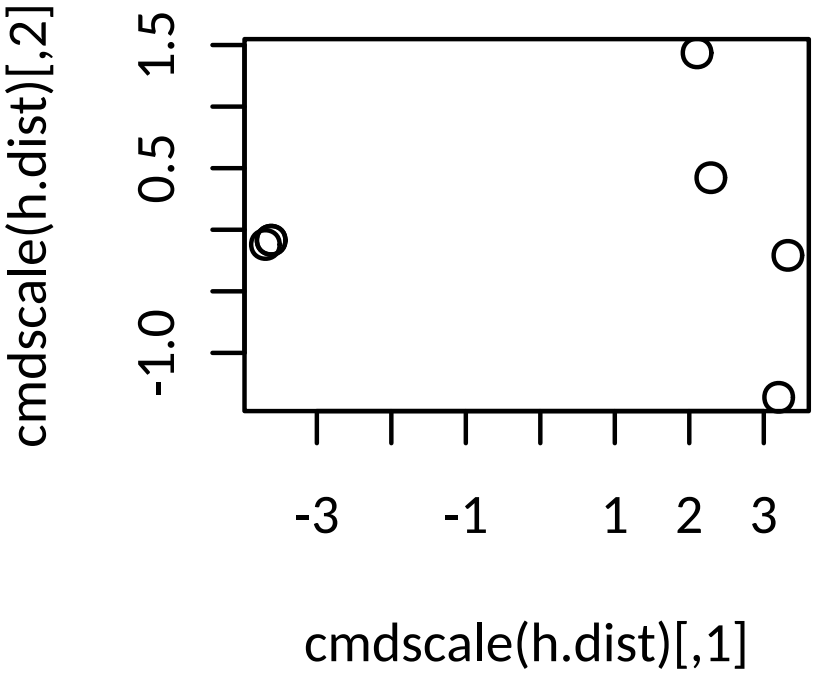
```
dna <- c("GACATTCCG", "GACAATCCG", "GACATTCGG",
  "AACACGTAC", "ATCACGTAC", "ATCACATAC", "AACAGGTAC")
h.dist <- matrix(nrow=length(dna), ncol=length(dna))
for(i in 1:length(dna)){
  for(j in 1:length(dna)){
    h.dist[i, j] <- sum(strsplit(dna[i], "")[[1]] != strsplit(dna[j], "")[[
  }
}
h.dist <- as.dist(h.dist)
```

Multidimensional scaling can be used when you start with a dissimilarity matrix, not the raw data.

```
h.dist
```

```
  1 2 3 4 5 6
2 1
3 1 2
4 6 6 6
5 7 7 7 1
6 7 7 7 2 1
7 6 6 6 1 2 3
```

```
plot(cmdscale(h.dist))
```



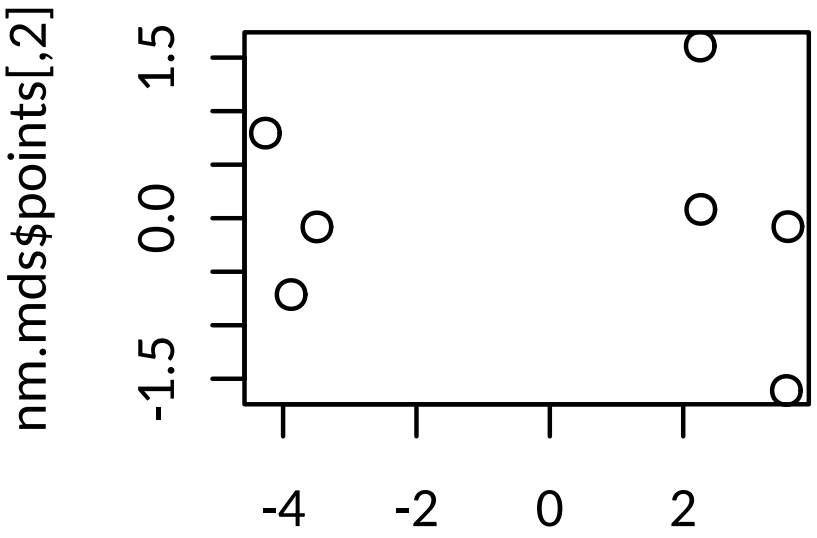
Non-metric multidimensional maps points into lower dimensions, preserving the *ordering* of distances.

- The metric value of distances may not be preserved, but points that are far apart remain far apart, and points that are close remain close.
- This is an iterative algorithm, and depends on a starting configuration.

```r
library(MASS)
nm.mds <- isoMDS(h.dist)
```

```
initial  value 4.363507
iter   5 value 0.837989
iter  10 value 0.058767
final  value 0.000000
converged
```

```r
plot(nm.mds$points)
```

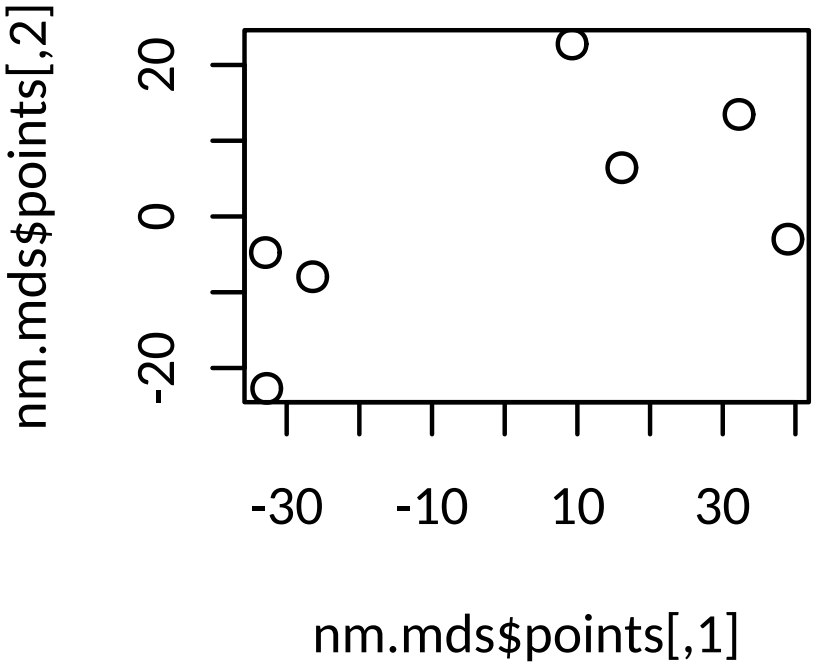


Two runs with different (random) starting configurations.

```r
set.seed(32)
nm.mds <- isoMDS(d=h.dist,
  y=matrix(runif(2*length(dna)),
           ncol=2))
```

```
initial  value 43.382979
iter   5 value 11.553085
iter  10 value 5.164965
iter  15 value 1.593107
iter  20 value 0.527715
iter  25 value 0.180297
iter  30 value 0.073584
iter  35 value 0.043111
final  value 0.001659
converged
```

```r
plot(nm.mds$points)
```

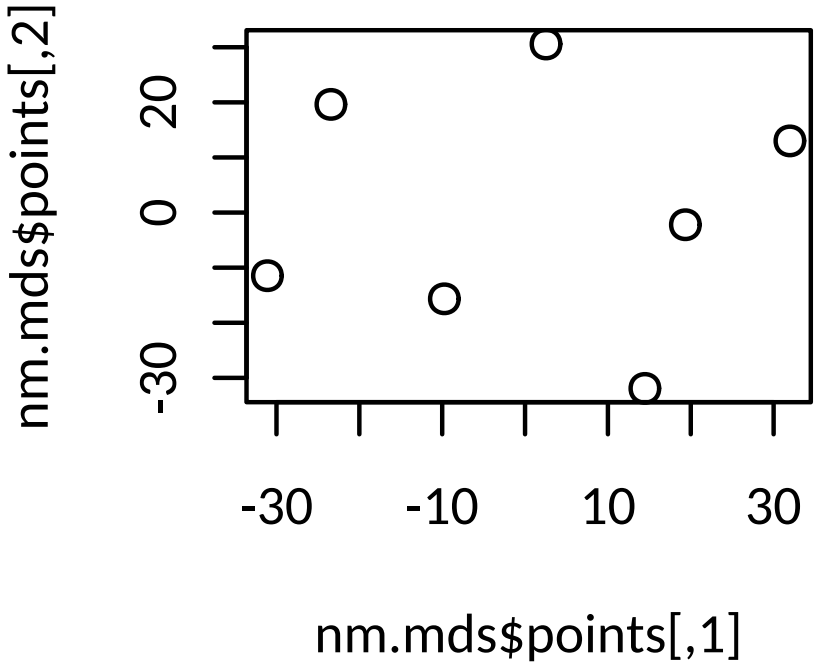


Two runs with different starting configurations.

```r
set.seed(33)
nm.mds <- isoMDS(d=h.dist,
  y=matrix(runif(2*length(dna)),
           ncol=2))
```

```
initial  value 49.555009
iter   5 value 31.764684
iter  10 value 30.631272
final  value 30.146274
converged
```

```r
plot(nm.mds$points)
```



Non-metrix MDS gives a non-overlapping view of meat clusters.

```r
set.seed(2873)
nutr.mds <- isoMDS(nutr.dist, trace=FALSE)
for(i in 1:10){
  tmp.mds <- isoMDS(nutr.dist,
    y=matrix(runif(2*nrow(nutr)),
             ncol=2), trace=FALSE)
  if(tmp.mds$stress < nutr.mds$stress){
    nutr.mds <- tmp.mds
  }
}
```

```r
plot(nutr.mds$points,
  col=nutr.km$cluster, pch=19,
  xlab="", ylab="")
text(nutr.mds$points,
     label=rownames(nutr),
     cex=0.5)
```