Copenhagen Part 2: Visualizations while performing clustering

Intro

Clustering was not intuitive to me when I first started hearing about it. Now it is a go to tool of mine for understanding my data. It only really made sense when I learned how to visualize my data. Here I present a workflow for approaching a clustering problem using the `kohonen` package.

Clustering: Intuitively, clustering is the problem of partitioning a finite set of points in a multidimensional space into classes (called clusters) so that 1. the points belonging to the same class are similar and 2. the points belonging to different classes are dissimilar

Discussion Break

- How many of you use clustering regularly?
- What sorts of clustering methods do you use?
- What is the difference between supervised and unsupervised clustering methods?
- What sorts of questions can you ask with clustering?
- How do you know which clustering method to use?

Setting up your enviroment

```
# Install missing libraries
install.packages("factoextra")
install.package("naniar")
install.packages("RCurl")
install.packages("cowplot")
install.packages("tidyverse")
install.packages("kohonen")
install.packages("cowplot")
```

Loading libraries

```
library(tidyverse)
library(factoextra)
library(dplyr)
library(naniar)
library(RCurl) # to access datafile
library(kohonen)
library(cowplot)
```

The Data: Titanic survival dataset

Get the data!

`RCurl` is a useful tool to access data files on Github.

tip: Make sure you use the "raw" data URL link if you ever plan on using this tool another time.

```
x <- getURL("https://raw.githubusercontent.com/Geoyi/Cleaning-Titanic-Data/master/titanic_original.csv")
titanic <- read.csv(text = x)
```

```
# # write out to save dataset, so you can access without internet.
# write.csv(titanic, "./titanic.csv")
# titanic <- read.csv("./titanic.csv")
```

Getting to know the titanic dataset

You can read about the data here.

```
## My goto for checking my first
head(titanic)
```

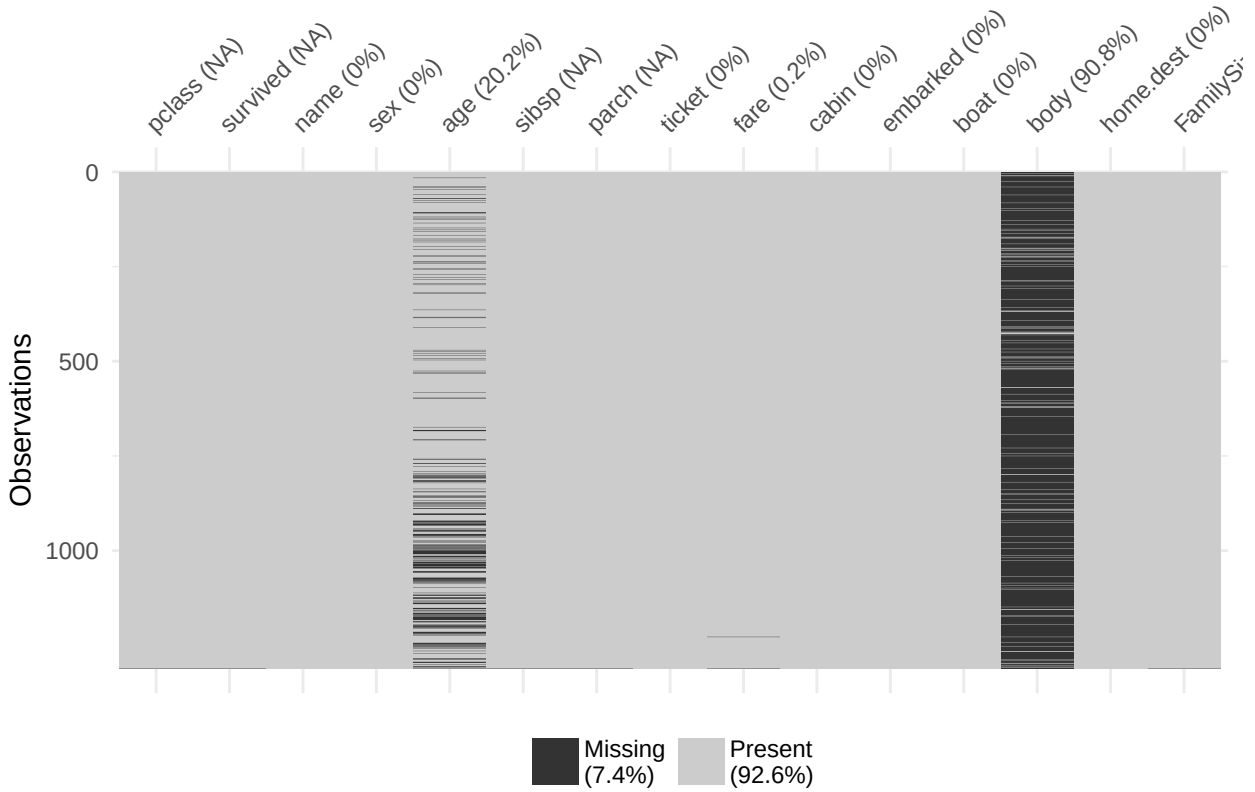
```
##   pclass survived                                             name    sex
## 1      1        1                    Allen, Miss. Elisabeth Walton female
## 2      1        1                   Allison, Master. Hudson Trevor   male
## 3      1        0                     Allison, Miss. Helen Loraine female
## 4      1        0             Allison, Mr. Hudson Joshua Creighton   male
## 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6      1        1                               Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked boat body
## 1 29.0000     0     0  24160 211.3375      B5        S    2   NA
## 2  0.9167     1     2 113781 151.5500 C22 C26        S   11   NA
## 3  2.0000     1     2 113781 151.5500 C22 C26        S        NA
## 4 30.0000     1     2 113781 151.5500 C22 C26        S       135
## 5 25.0000     1     2 113781 151.5500 C22 C26        S        NA
## 6 48.0000     0     0  19952  26.5500     E12        S    3   NA
##                         home.dest
## 1                     St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
## 4 Montreal, PQ / Chesterville, ON
## 5 Montreal, PQ / Chesterville, ON
## 6                     New York, NY
```

```
str(titanic)
```

```
## 'data.frame':    1310 obs. of  14 variables:
##  $ pclass   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : int  1 1 0 0 0 1 1 0 1 0 ...
##  $ name     : Factor w/ 1308 levels "","Abbing, Mr. Anthony",..: 23 25 26 27 28 32 47 48 52 56 ...
##  $ sex      : Factor w/ 3 levels "","female","male": 2 3 2 3 2 3 2 3 2 3 ...
##  $ age      : num  29 0.917 2 30 25 ...
##  $ sibsp    : int  0 1 1 1 1 0 1 0 2 0 ...
##  $ parch    : int  0 2 2 2 2 0 0 0 0 0 ...
##  $ ticket   : Factor w/ 930 levels "","110152","110413",..: 189 51 51 51 51 126 94 17 78 827 ...
##  $ fare     : num  211 152 152 152 152 ...
##  $ cabin    : Factor w/ 187 levels "","A10","A11",..: 45 81 81 81 81 151 147 17 63 1 ...
##  $ embarked : Factor w/ 4 levels "","C","Q","S": 4 4 4 4 4 4 4 4 4 2 ...
##  $ boat     : Factor w/ 28 levels "","1","10","11",..: 13 4 1 1 1 14 3 1 28 1 ...
##  $ body     : int  NA NA NA 135 NA NA NA NA NA 22 ...
##  $ home.dest: Factor w/ 370 levels "","?Havana, Cuba",..: 310 232 232 232 232 238 163 25 23 230 ...
```

```
## Engineer new feature
titanic$FamilySize <- 1 + titanic$sibsp + titanic$parch
```

```
## What kind of NAs we got going on?
naniar::vis_miss(titanic)
```




```
## Remove columns that could be trouble or you feel will not add to cluster
## Identity.
rem_cols <- c("body", "ticket", "name", "cabin", "boat", "home.dest", "age", "sibsp", "parch")
titanic <- titanic %>%
  select(-one_of(rem_cols))

## check
colnames(titanic)
```

```
## [1] "pclass"     "survived"   "sex"        "fare"       "embarked"
## [6] "FamilySize"
```

Setting up data depending on data type

First, what are my data types again?

```
sapply(titanic, typeof)
```

```
##     pclass   survived        sex       fare   embarked FamilySize
##  "integer"  "integer"  "integer"   "double"  "integer"   "double"
```

Right away I notice that two columns I want to include are being treated as a data type that is not appropriate (survived and pclass). When you input data into R, R often treats all number data as a number, even if the number is a factor. This can cause many problems, so I usually spot check right away.

Discussion break on data types

- Does everyone understand why?

- What sort of problems / errors will this cause later on?

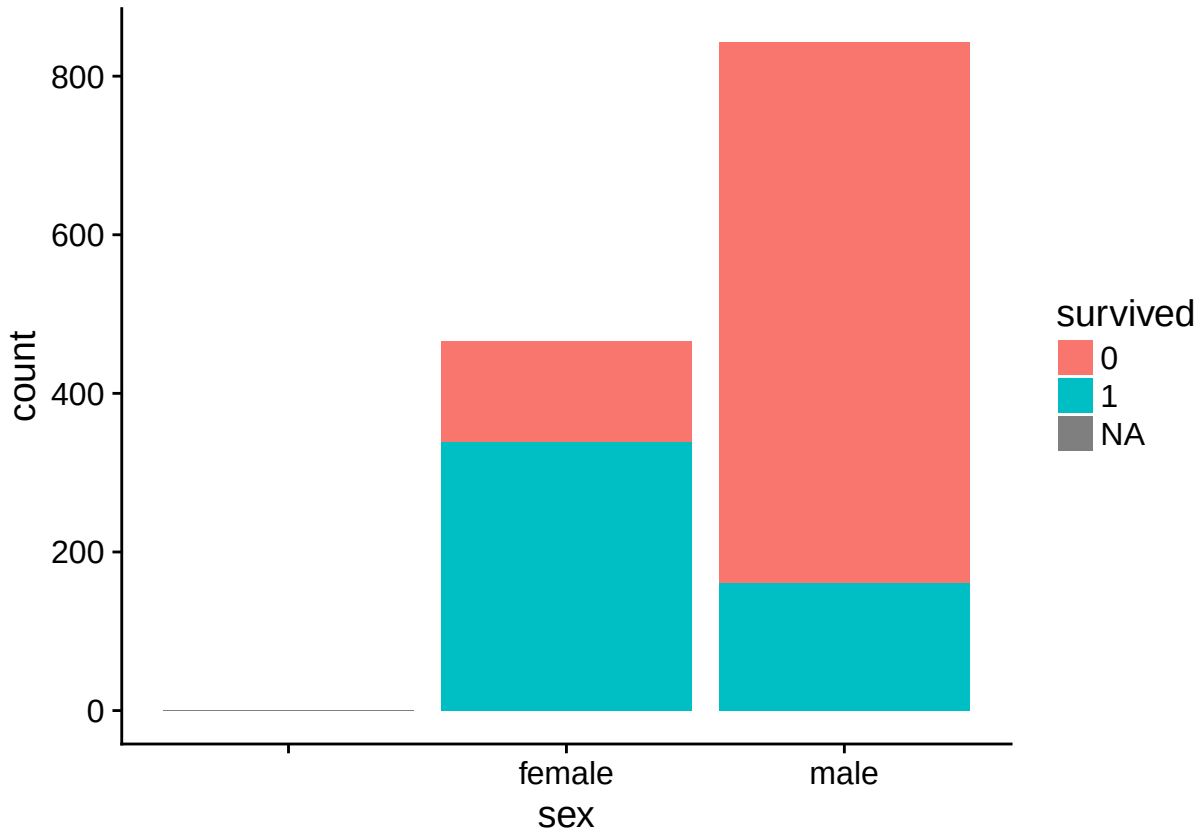
```
## Set numbered factor columns to factor data type
titanic$pclass <- as.factor(titanic$pclass)
titanic$survived <- as.factor(titanic$survived)

## Set double to numeric, which will end up rounding
## ticket price to nearest doller

titanic$fare <- as.numeric(titanic$fare)
titanic$FamilySize <- as.numeric(titanic$FamilySize)
```

Once the data types are all set, I like to just be curious about the data and visualize a few things to understand the dataset. Let's just play around a bit, we earned it!

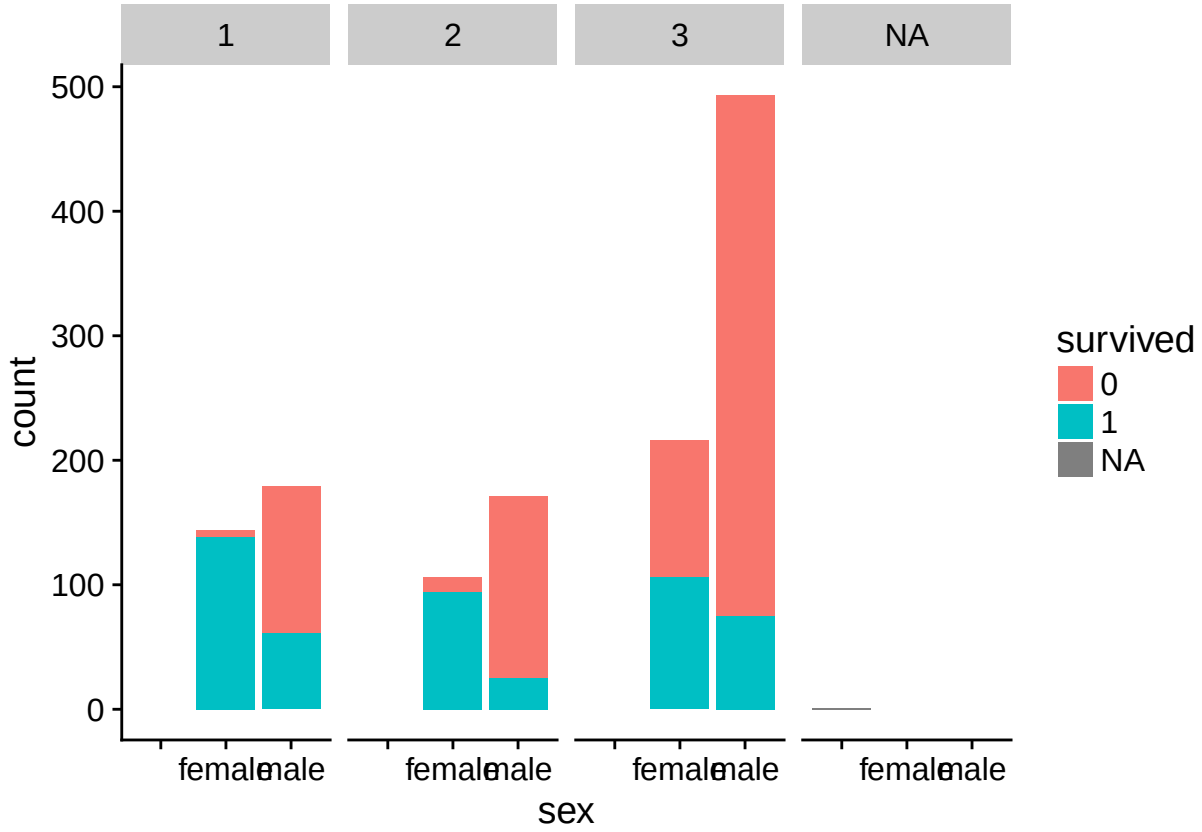
```
colnames(titanic)
```

```
## [1] "pclass"     "survived"   "sex"        "fare"       "embarked"
## [6] "FamilySize"
```

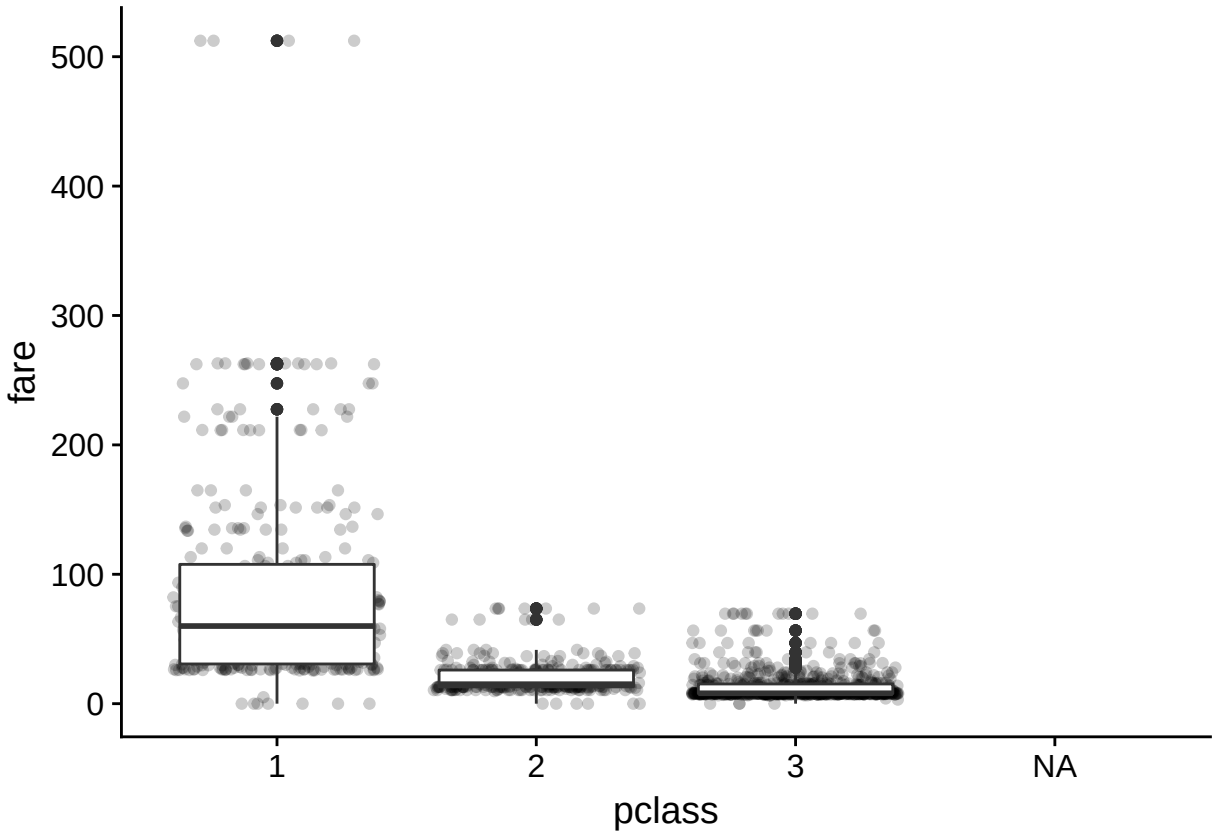
```
## What is the male female breakdown of survials?
titanic %>%
  ggplot(., aes(x = sex, fill = survived)) +
    geom_bar()
```




```
## What is the male female breakdown of survials in each class?
titanic %>%
  ggplot(., aes(x = sex, fill = survived)) +
    geom_bar() +
  facet_grid(.~pclass)
```




```
## What kind of fare was everyone paying anyway?
ggplot(titanic, aes(pclass, fare)) +
  geom_jitter(alpha = .2) +
  geom_boxplot()
```



Get data ready for clustering

The `kohonen::supersom()` function accepts multilayered data sets, a named list in which each element consists of a matrix with an equal number of observations(rows). The function lets us specify which layer to include and lets us chose difference distance measures for each layer.

But first, we need to deal with the factors. Super Organized Maps (and neural networks in general) can not deal in category (factor) data types. You have to be creative on how you input them. In this case we are going to use `kohonen::classvec2classmat()` function which turns a factor column vector into a class matrix and vice versa. Differently from functions that convert a factor column vector into dummy variables, this function adds a single binary column per factor level i.e. one hot encoding.

```
## let's start by giving our dataset a fresh name.
data_val <- titanic

## identify which columns are numeric and which are factors
numerics = summarise_all(data_val, is.numeric ) %>%
  as.logical()

factors = names(data_val) %>%
  .[!numerics]

numerics = names(data_val) %>%
  .[numerics]

## Set up for loop
data_list = list()
```

```
distances = vector()

## This takes each factor column and makes into a matrix. Then you will have a list of matrices.

for (fac in factors){
  data_list[[fac]] = kohonen::classvec2classmat( data_val[[fac]] )
  distances = c(distances, 'tanimoto')
}

## This scales all the numeric data and
data_list[['numerics']] = scale(data_val[,numerics])
distances = c(distances, 'euclidean')

## Check out the data
str(data_list)
```

```
## List of 5
##  $ pclass  : num [1:1310, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "1" "2" "3"
##  $ survived: num [1:1310, 1:2] 0 0 1 1 1 0 0 1 0 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "0" "1"
##  $ sex     : num [1:1310, 1:3] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "" "female" "male"
##  $ embarked: num [1:1310, 1:4] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "" "C" "Q" "S"
##  $ numerics: num [1:1310, 1:2] 3.44 2.28 2.28 2.28 2.28 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "fare" "FamilySize"
##   ..- attr(*, "scaled:center")= Named num [1:2] 33.3 1.88
##   .. ..- attr(*, "names")= chr [1:2] "fare" "FamilySize"
##   ..- attr(*, "scaled:scale")= Named num [1:2] 51.76 1.58
##   .. ..- attr(*, "names")= chr [1:2] "fare" "FamilySize"
```

```
head(data_list$pclass)
```

```
##      1 2 3
## [1,] 1 0 0
## [2,] 1 0 0
## [3,] 1 0 0
## [4,] 1 0 0
## [5,] 1 0 0
## [6,] 1 0 0
```

```
names(data_list)
```

```
## [1] "pclass"   "survived" "sex"      "embarked" "numerics"
```

```
str(data_list$numerics)
```

```
##  num [1:1310, 1:2] 3.44 2.28 2.28 2.28 2.28 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "fare" "FamilySize"
##  - attr(*, "scaled:center")= Named num [1:2] 33.3 1.88
##   ..- attr(*, "names")= chr [1:2] "fare" "FamilySize"
##  - attr(*, "scaled:scale")= Named num [1:2] 51.76 1.58
##   ..- attr(*, "names")= chr [1:2] "fare" "FamilySize"
```

Performing the clustering

Now we finally get to performing the SOM.

How do you know how many dimensions my grid will be?

- There are ways to do this appropriately, but it is a little complex of a topic. One rule of thumb is just make sure you don't get too many empty nodes. If you are doing a proper analysis, look into some of the tests you can perform here. It also fun to just change the number of dimensions and explore how the clusters change. When you do this you get a better idea how well the neural network performs.

How do you know how many iterations you need?

- This all depends on the training plot which we will look at after making the plot.

```
## Setting the map dimension and iterations.
map_dimension = 8
n_iterations = 1000

## Set up SOM grid
## This is the actual grid that the clustering will occur
som_grid = kohonen::somgrid(xdim = map_dimension,
                            ydim = map_dimension,
                            topo = "hexagonal")

## I always set my seed to 8, becasue it is my favorite number
set.seed(8)
m = kohonen::supersom(data_list,
                      grid = som_grid,
                      rlen = n_iterations,
                      alpha = 0.05,
                      whatmap = c(factors, 'numerics'),
                      dist.fcts = distances,
                      maxNA.fraction = .5)
```

Visualizing using the Kohonen plotting functions

The Kohonen plotting functions are *really* amazing. They are simple to use and are extremely informative.

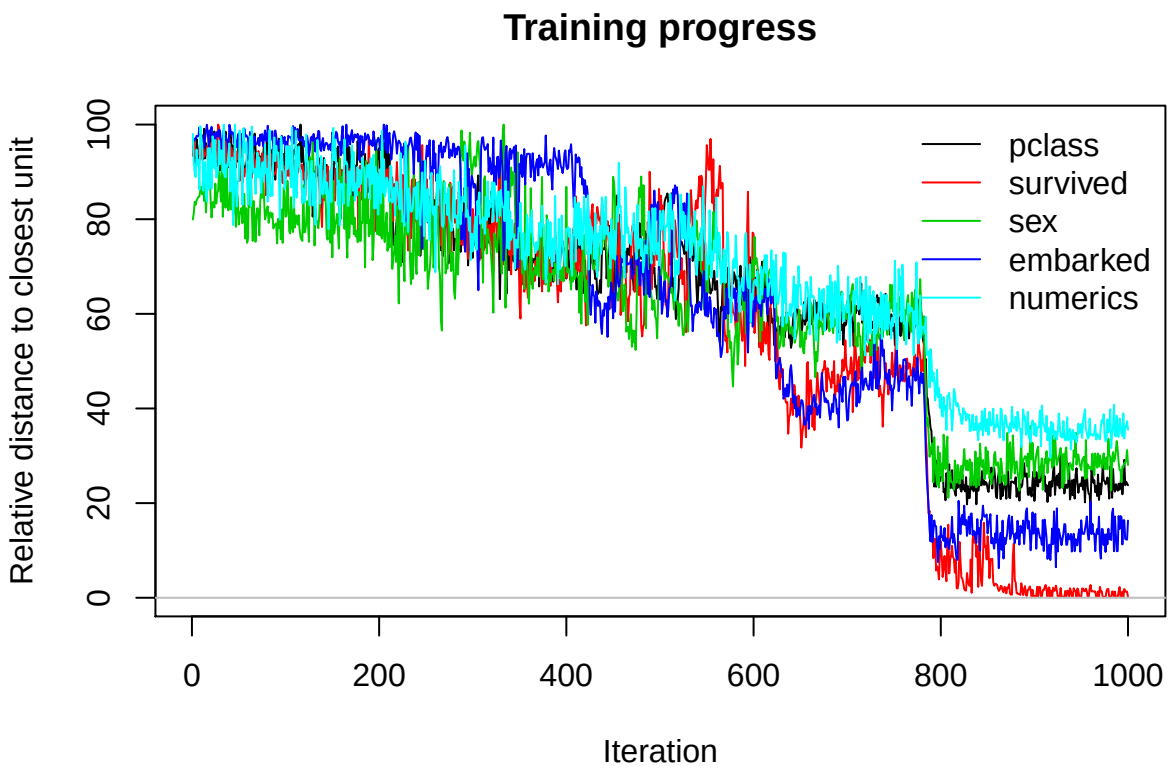
Training Plot `changes`

As the SOM training iterations progress, the distance from each node's weights to the samples represented by that node is reduced. Ideally, this distance should reach a minimum plateau. This plot option shows the

progress over time. If the curve is continually decreasing, more iterations are required.

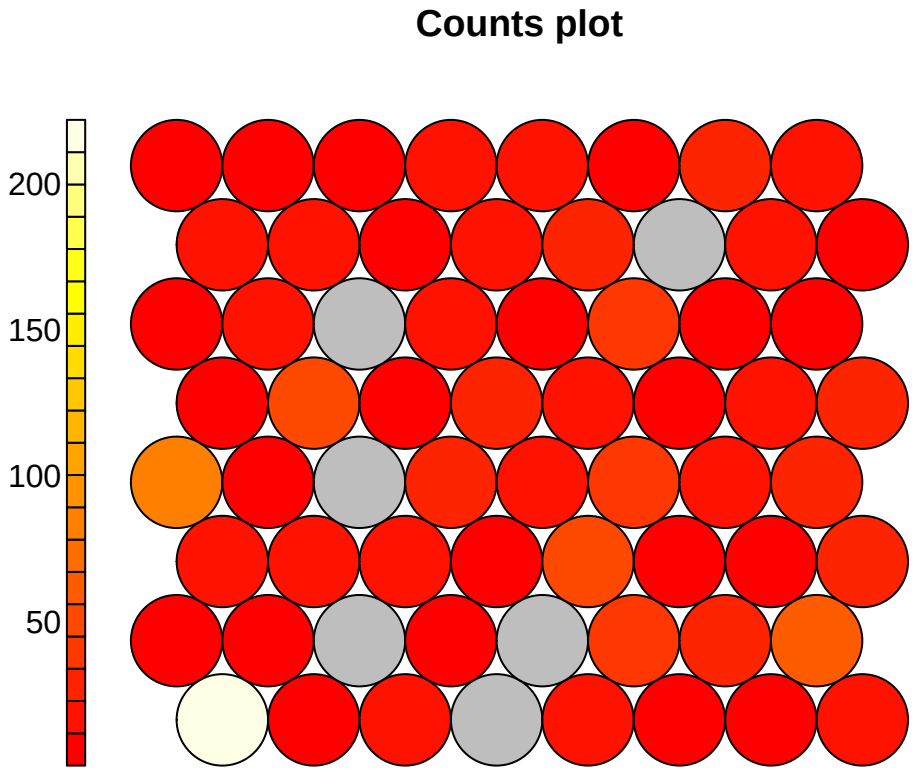
- Let's go back and change the iterations to something low.

```
plot(m, type = "changes")
```

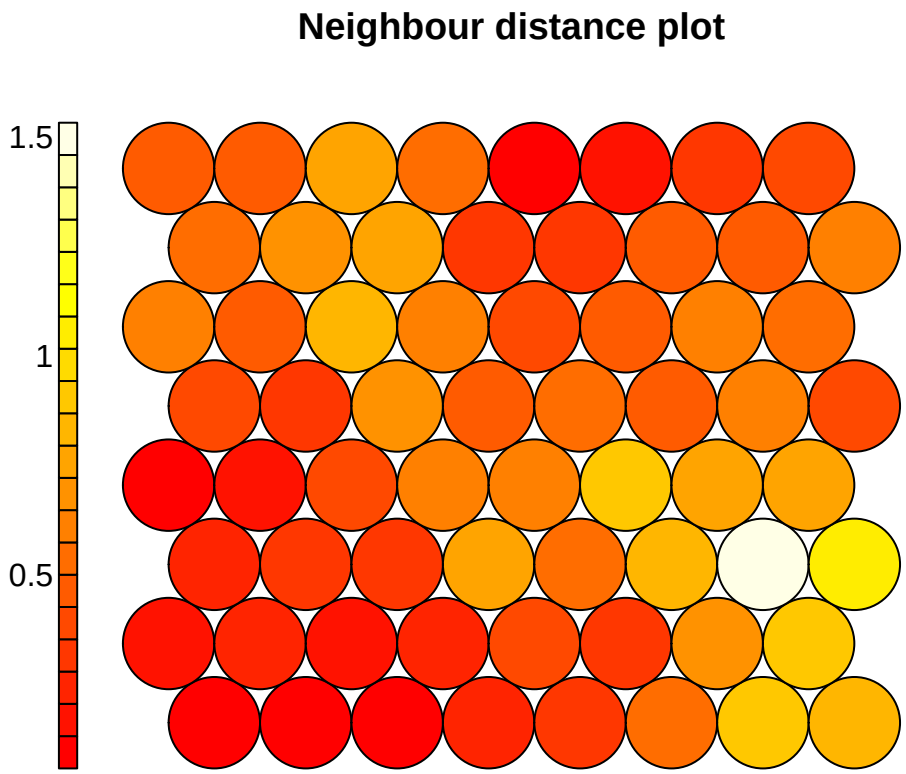


Let's look into some of the other default plotting functions.

```
plot(m, type = "counts")
```

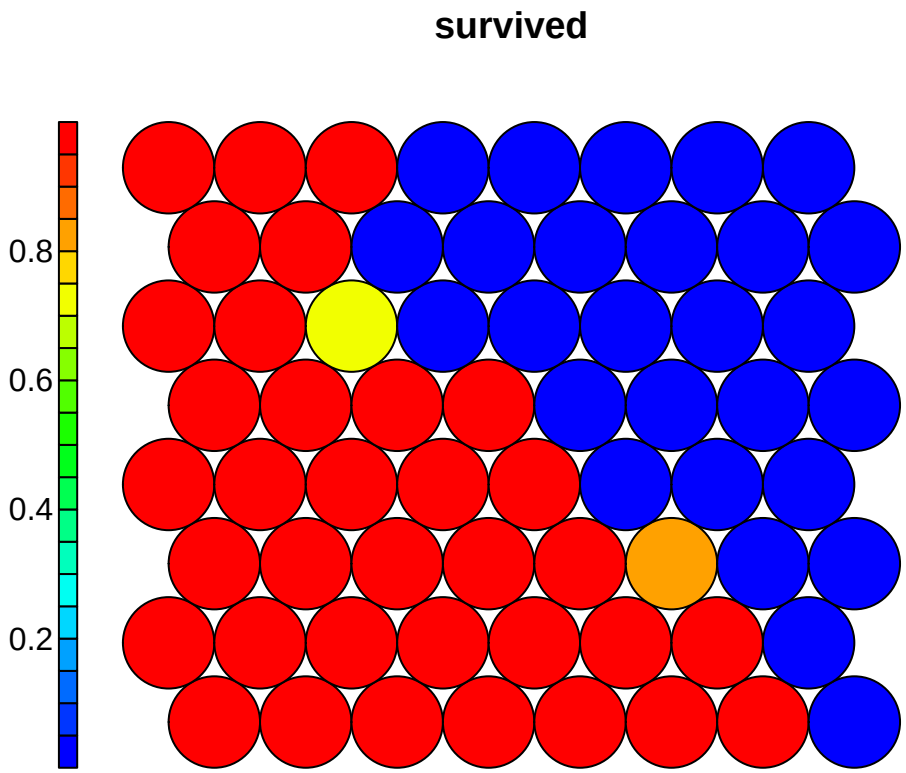



```
plot(m, type = "dist.neighbours")
```

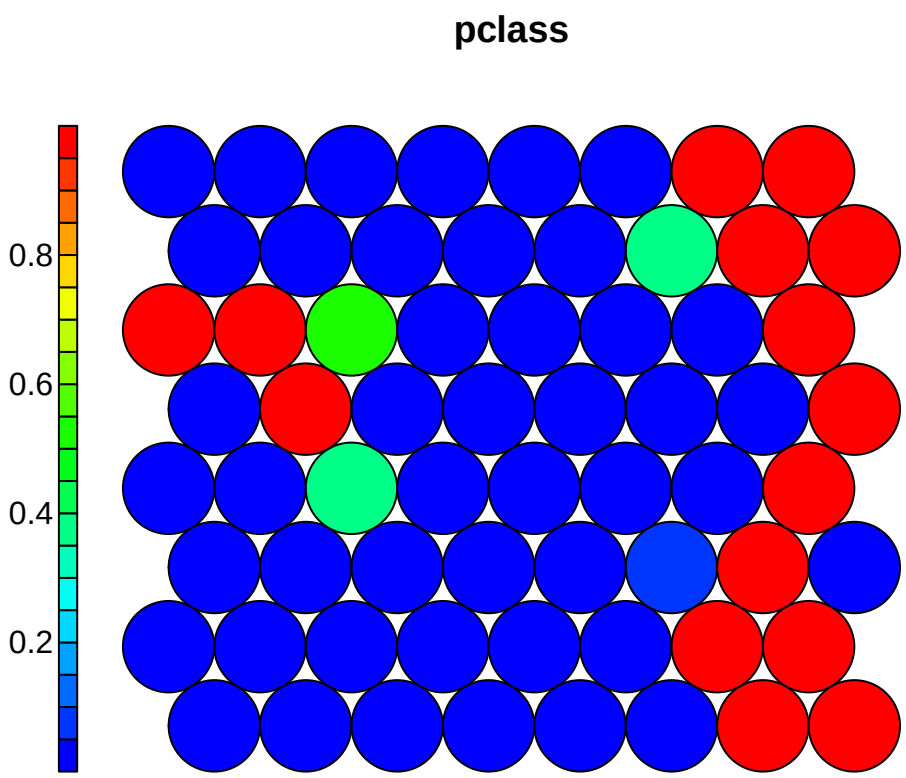



```
## A little clunky on how to get the property plots to work with nested lists
## But the property plots are useful.
```

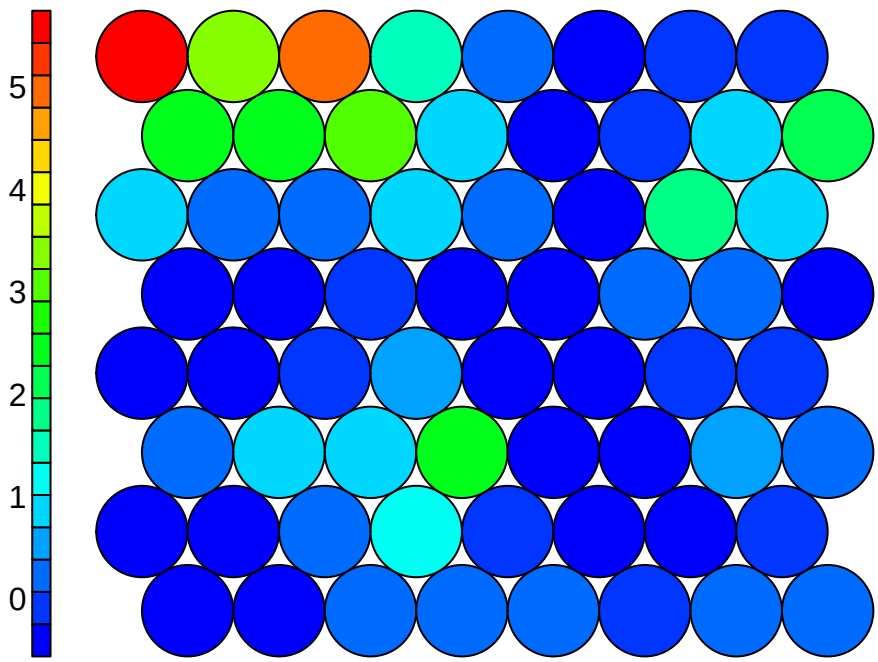
```
coolBlueHotRed <- function(n, alpha = 1) {rainbow(n, end=4/6, alpha=alpha)[n:1]}


plot(m, type = "property", property = as.matrix(m$codes$survived[,1]), main = names(m$codes)[2], palette
```



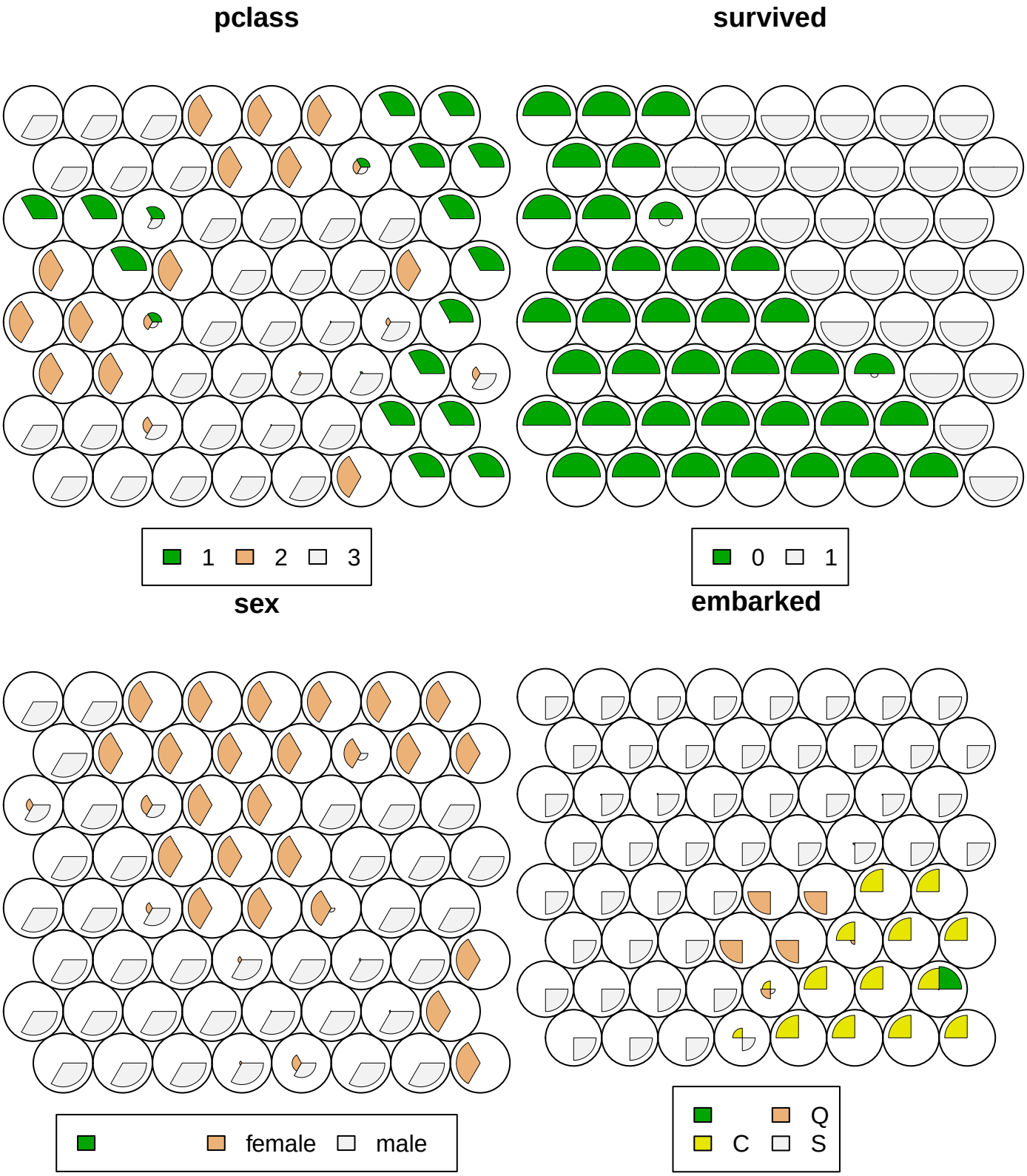

```
plot(m, type = "property", property = as.matrix(m$codes$pclass[,1]), main = names(m$codes)[1], palette.
```

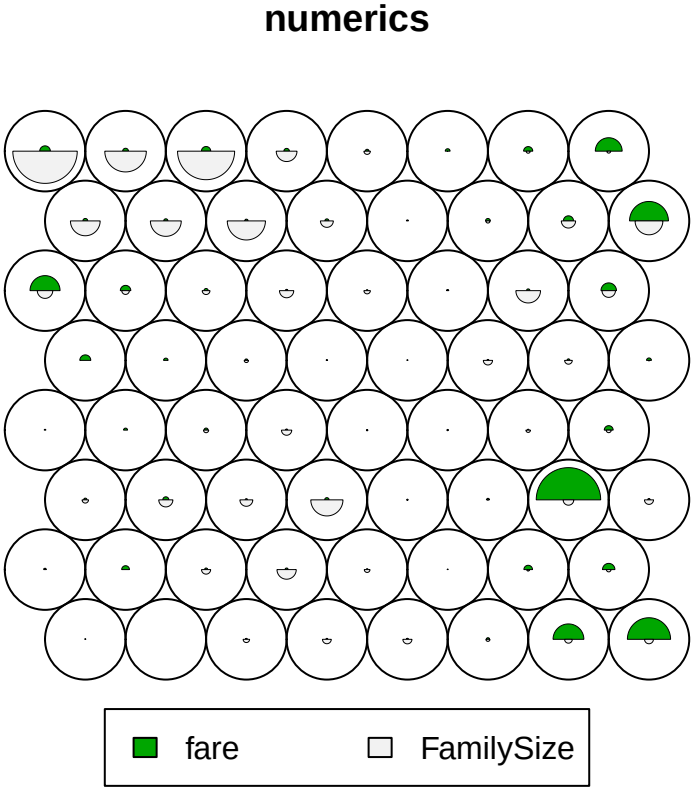



```
plot(m, type = "property", property = as.matrix(m$codes$numerics[,2]), main = "family size", palette.nar
```




```
plot(m, type = "codes")
```


pclass
survived
1
2
3
0
1
sex
embarked
female
male
Q
C
S



Understanding the output

Let check out the output of `supersom()`.

```
names(m)
```

```
##  [1] "data"             "unit.classif"     "distances"
##  [4] "grid"             "codes"            "changes"
##  [7] "alpha"            "radius"           "user.weights"
## [10] "distance.weights" "whatmap"          "maxNA.fraction"
## [13] "dist.fcts"
```

```
class(m) #class kohonen
```

```
## [1] "kohonen"
```

What we have here is a list of 13. The parts of the data that are really important are

- `unit.classif` = this is the cluster assignments.
- `codes` = the abstracted weights that describe each category in each cluster and ultimately decided what which person was put into which clusters.

Clusters only really made sense to me once I brought the data back in. The coolest part is that I can all the data back to the original data by attaching `unit.classif` back to our original data.

```
## the length is the same length as our titanic dataset
length(m$unit.classif)
```

```
## [1] 1310
```

```
dim(titanic)
```

```
## [1] 1310    6
```

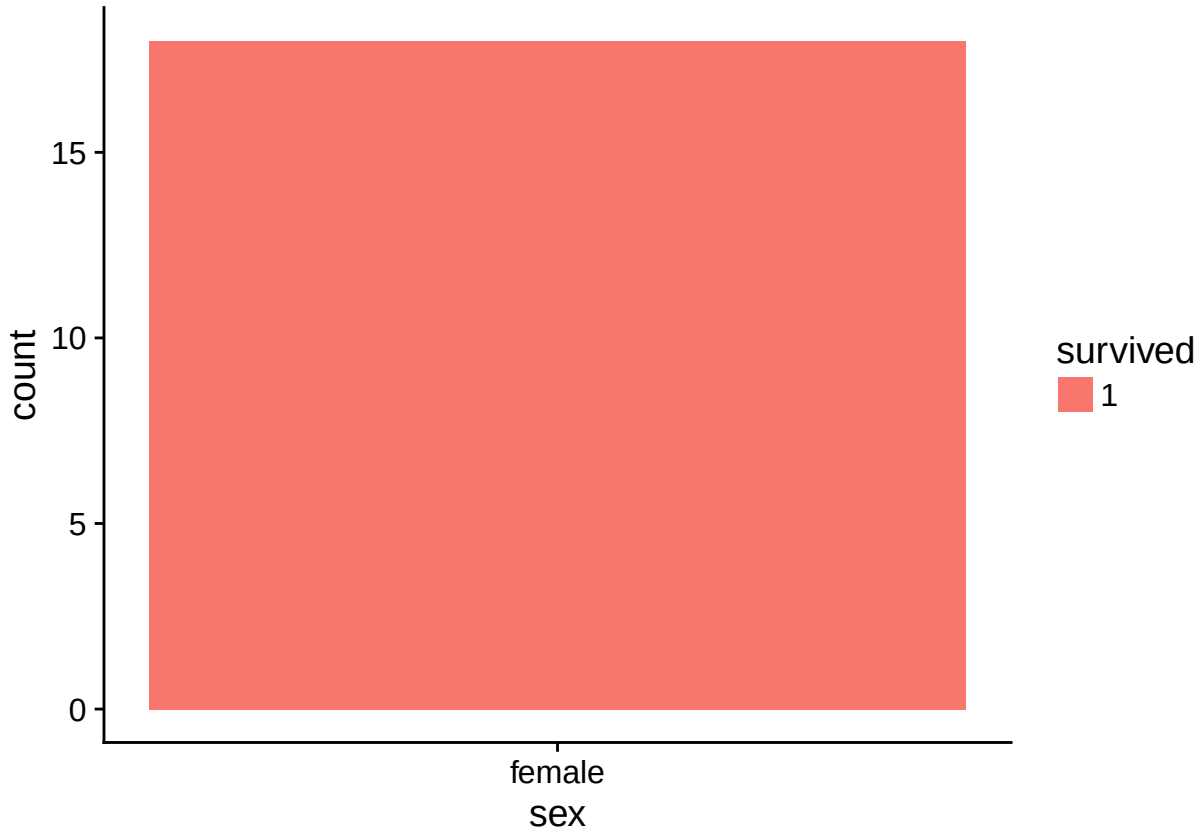
```
titanic_SOM <- cbind(titanic, as.data.frame(m$unit.classif))

## Change column header so it makes more sense
colnames(titanic_SOM)[7] <- "cluster"
```

Now we can start to understand each cluster. For example, let's look at a few clusters.

```
## Let's look at one cluster
titanic_SOM %>%
  filter(m$unit.classif == 52)
```

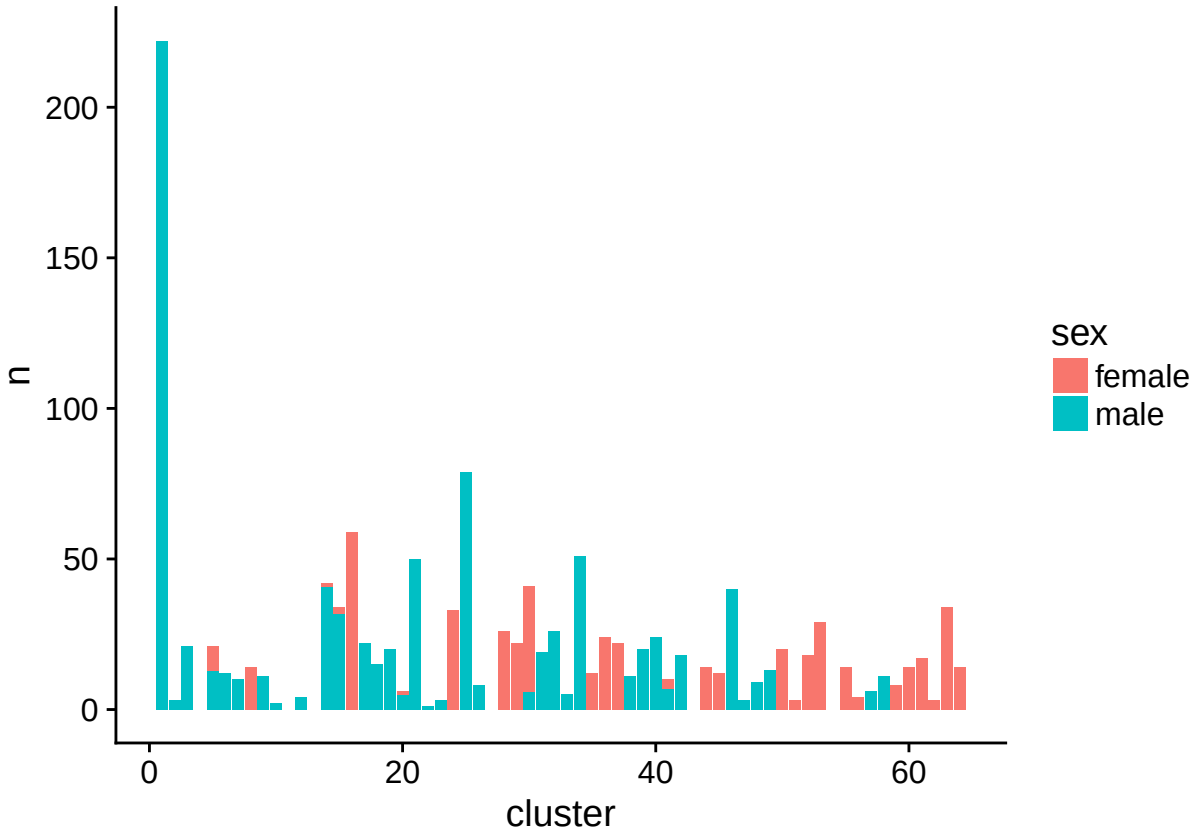
```
##    pclass survived    sex  fare embarked FamilySize cluster
## 1       2        1 female 39.00        S          3      52
## 2       2        1 female 39.00        S          3      52
## 3       2        1 female 29.00        S          3      52
## 4       2        1 female 30.00        S          3      52
## 5       2        1 female 30.00        S          3      52
## 6       2        1 female 26.25        S          3      52
## 7       2        1 female 26.25        S          3      52
## 8       2        1 female 36.75        S          3      52
## 9       2        1 female 32.50        S          3      52
## 10      2        1 female 14.50        S          3      52
## 11      2        1 female 26.25        S          3      52
## 12      2        1 female 26.25        S          3      52
## 13      2        1 female 26.00        S          3      52
## 14      2        1 female 26.00        S          3      52
## 15      2        1 female 26.00        S          3      52
## 16      2        1 female 13.00        S          3      52
## 17      2        1 female 23.00        S          3      52
## 18      2        1 female 23.00        S          3      52
```

```
titanic_SOM %>%
  filter(m$unit.classif == 52) %>%
  ggplot(., aes(sex, fill = survived)) +
    geom_bar()
```



Or lets look at the composite of all clusters

```
cluster_tally  <- titanic_SOM %>%
  group_by(cluster, sex, survived) %>%
  tally() %>%
  na.omit()

ggplot(cluster_tally, aes(x = cluster, y = n, fill = sex))  +
  geom_bar(stat = "identity")
```




```
## -  Why doesn't this work?
# titanic_SOM %>%
#   group_by(cluster, sex) %>%
#   tally() %>%
#   ggplot(cluster_tally, aes(x = cluster, y = n, fill = sex))  +
#   geom_bar(stat = "identity")
```

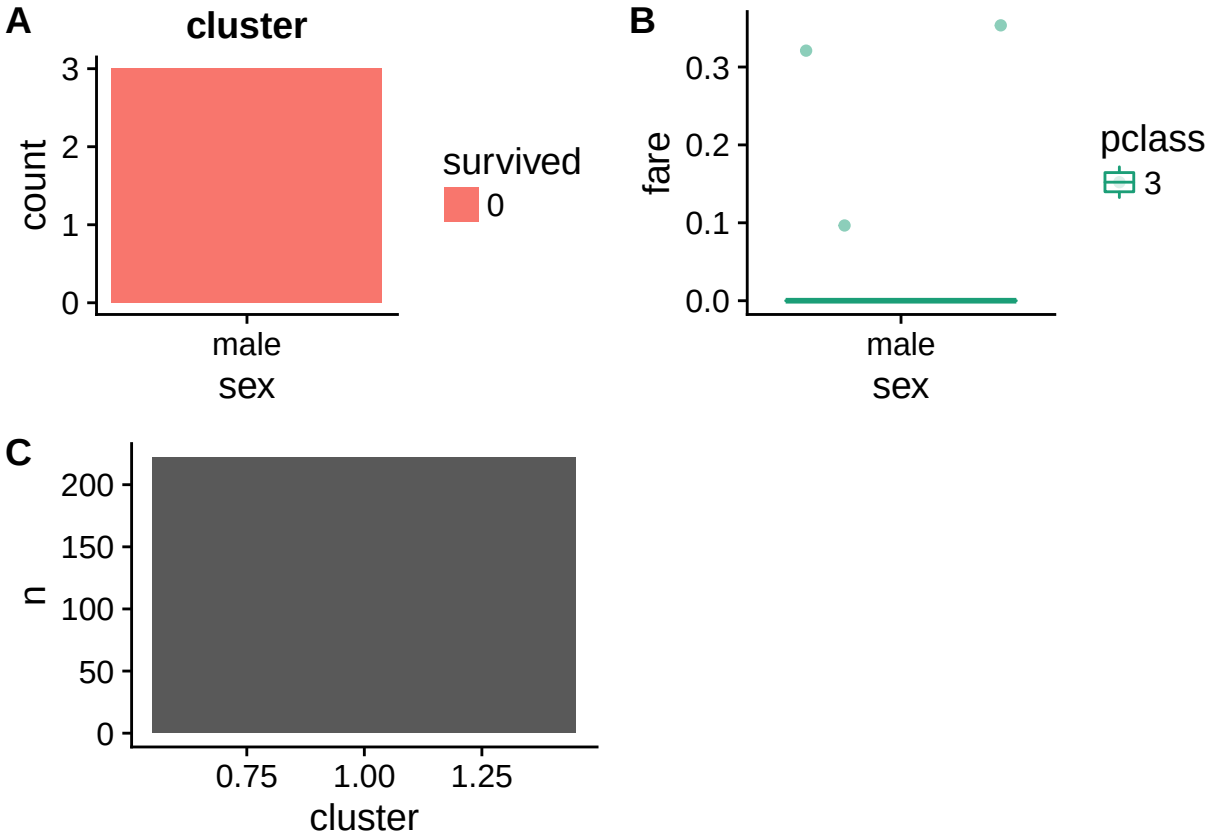
The graph above gives us an over view of how sex is distributed across clusters.

Or you can start to look into composite of all data within a single cluster.

```
cluster_descriptions <- function(clust_num) {
  #Graph 3
  survival_sex <- titanic_SOM %>%
    filter(cluster == clust_num) %>%
    ggplot(., aes(x = sex, fill = survived)) +
    geom_bar() +
    ggtitle(paste0(c("cluster", clust_num)))
  ## Graph 2
  fare_class <- titanic_SOM %>%
    filter(cluster == clust_num) %>%
    ggplot(., aes(x = sex, y = fare, color = pclass)) +
    geom_jitter(alpha = .5) +
    geom_boxplot(alpha = .7, outlier.size = 0) +
    scale_color_brewer(palette = "Dark2")
  ## Graph 3
  cluster_tally_1  <- titanic_SOM %>%
  filter(cluster == 1) %>%
  group_by(cluster, sex, survived) %>%
```

```
  tally() %>%
  na.omit()
  ## Graph 4
   tally <- ggplot(cluster_tally_1, aes(x = cluster,
                                        y = n))  +
     geom_bar(stat = "identity")
   ## put all together
  plot_grid(survival_sex, fare_class, tally, labels = c("A", "B", "C"), nrow = 2, align = "v")
}

cluster_descriptions(2)
```

```
## Warning in align_plots(plotlist = plots, align = align, axis = axis):
## Complex graphs cannot be vertically aligned unless axis parameter is set
## properly. Placing graphs unaligned.
```



A little messy, but you get the point. It is best to wrap these in functions and just explore the clusters. Go ahead and try a few different clusters.

You can try to add one more graph visualization to this function at home.

Clustering Clusters: Hierarchical Clustering

Do get a even smaller set of clusters, you can further perform hierarchical clustering on the clusters. I have messed around with this feature in the past and the individuals in these "larger" clusters are the same individuals if you set the original supersom map size. So if you REALLY want to do this, you can just set

the first map to prefer less clusters. Depends on what you are tying to ask. The main question I ask when I want less clusters is "What are the main data patterns that explain my data?"

```
# fuse all layers into one dataframe
codes <- tibble(layers = names(m$codes),
                codes = m$codes ) %>%
  mutate( codes = purrr::map(codes, as_tibble)) %>%
  spread(key = layers, value = codes) %>%
  apply(1, bind_cols) %>%
  .[[1]] %>%
  as_tibble()

# generate distance matrix for codes
dist_m <- dist(codes) %>%
  as.matrix()

# generate seperate distance matrix for map location
dist_on_map <- kohonen::unit.distances(som_grid)

#exponentiate euclidean distance by distance on map
dist_adj = dist_m ^ dist_on_map
```

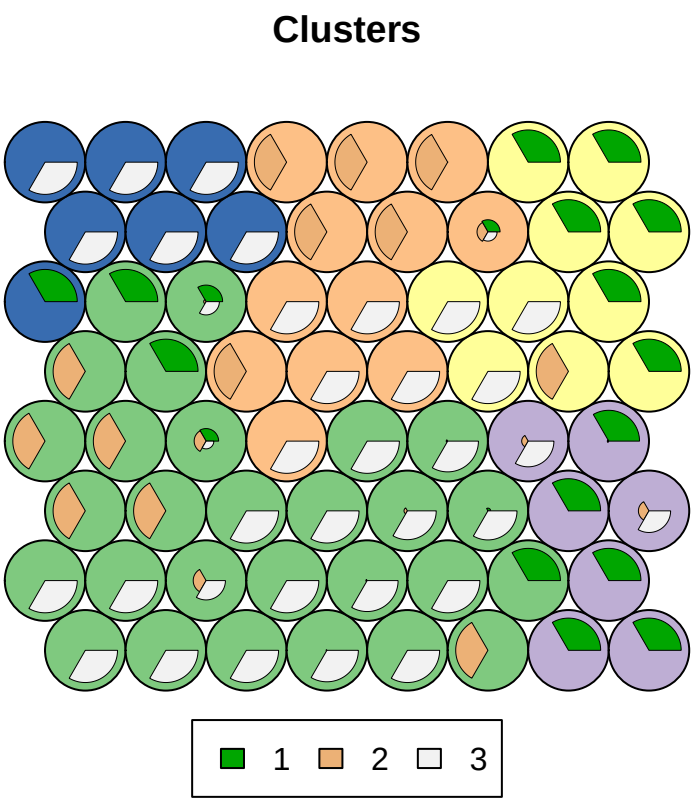
```
dist_adj =  dist_m ^ dist_on_map
clust_adj = hclust(as.dist(dist_adj), 'ward.D2')
```

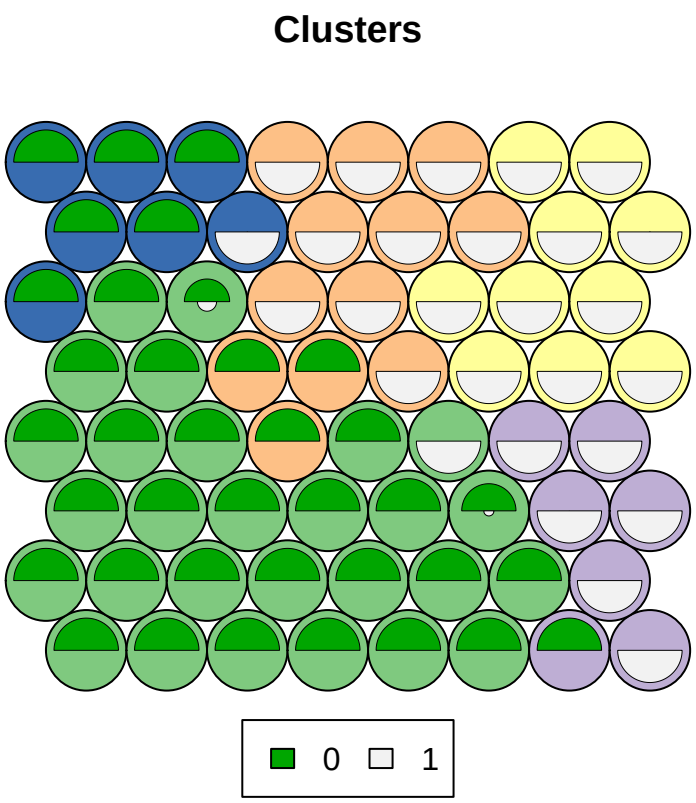
Set up the colors to define the clusters.

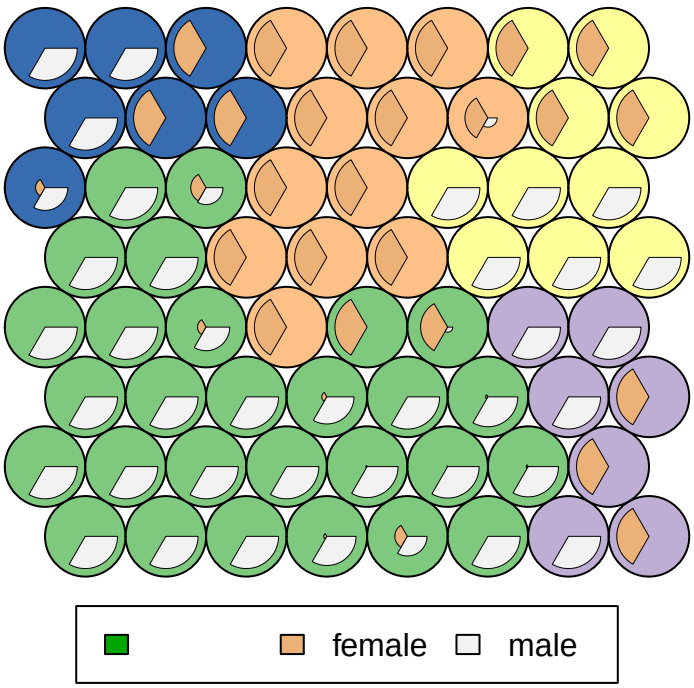
```
library(RColorBrewer)
n <- 64
qual_col_pals = brewer.pal.info[brewer.pal.info$category == 'qual',]
col_vector = unlist(mapply(brewer.pal, qual_col_pals$maxcolors, rownames(qual_col_pals)))
```

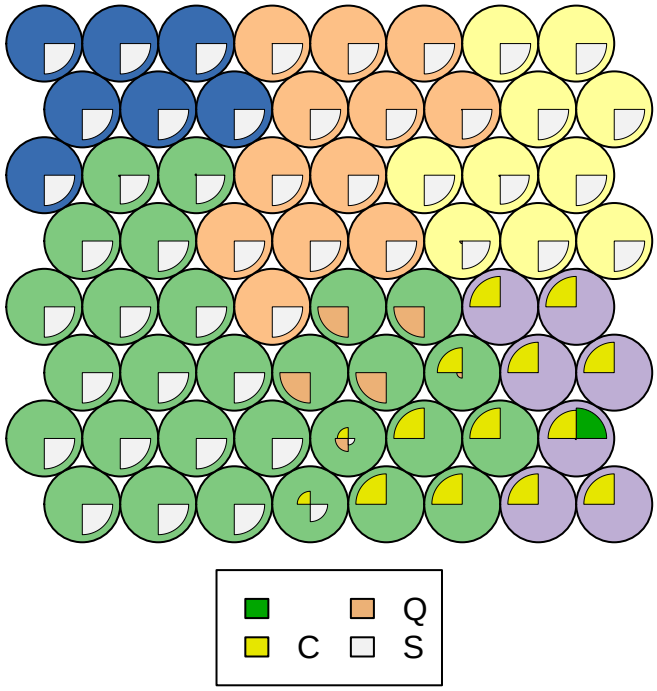
Now that we have the colors we can use the `cutree` function to further separate the clusters into larger units.

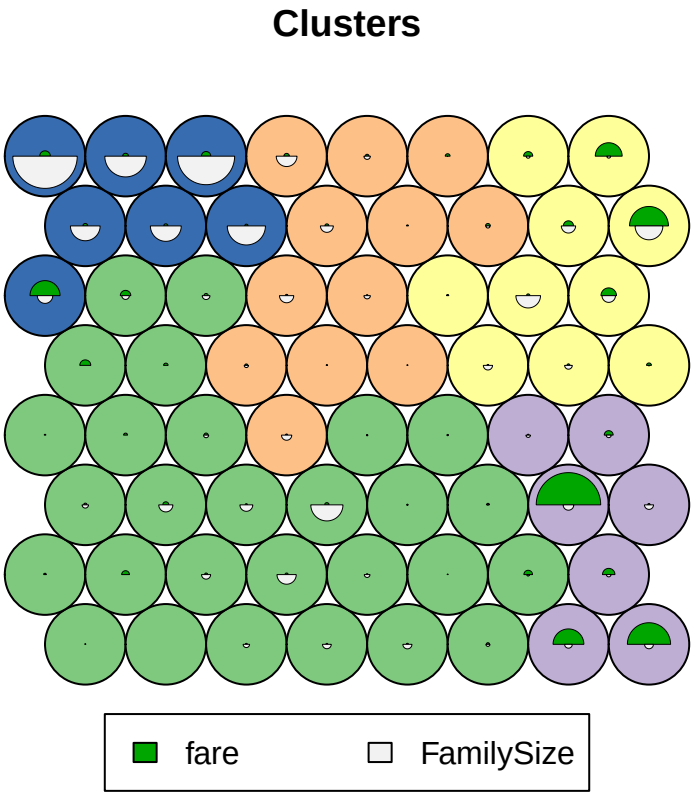
```
som_cluster_adj = cutree(clust_adj, 5)

## -  Make sure you look through all the plots generated
plot(m, type="codes", main = "Clusters", bgcol = col_vector[som_cluster_adj], pchs = NA)
```


Clusters
1
2
3


Clusters
0
1


Clusters
female
male


Clusters
Q
C
S



Resources

Developing SOM + Hierarchical Clustering + Connectivity Constrains - very nice overview of SOM analysis using the Kohonen package. Determing optimal number of clusters