Wine Map

by Fred

What is the Mapper?

1. From data set $\mathbb{X}$
2. We create a matrix of distances
3. From a quick analysis of our distance matrix, we reasonably choose ϵ
4. By using our chosen ϵ, we create an adjacency matrix
5. We cover our space
   (a) We select a point at random
   (b) We draw a ball of radius ϵ around it (initialize Cover i).
   (c) Points within ϵ of our chosen point are allocated to Cover i.
   (d) We select an uncovered/unallocated point and repeat the last two steps until all points have been covered.
6. We create a new adjacency matrix of covers where two covers are adjacent if their intersection is nonempty.
7. We select additional information for coloring and sizing our nodes
   In this case:
   - Coloring: Red Wines were given a value of 1, White Wines 0. Node Color was determined by mean value of wine type in each cover.
   - Sizing: Nodes are sized by the number of points in each cover.
8. Visualize it

Step 1: Packages and Data

```r
##### Packages ####
library(tidyverse)
library(igraph)
library(RColorBrewer)
library(FredsVietorisRips)
```

```r
##### Import Data ####
red <- read.csv("D:winequality-red.csv")
white <- read.csv("D:winequality-white.csv")

##### Add column Type ####
```

```r
# Column of
#   1's for Red
#   0's for White
red$type <- 1
white$type <- 0

##### Sample and Merge ####
set.seed(2020)
sample_size <- 50
df <- red %>%
  sample_n(sample_size) %>%
  full_join(sample_n(white, sample_size))
```

Step 2: Distance Matrix

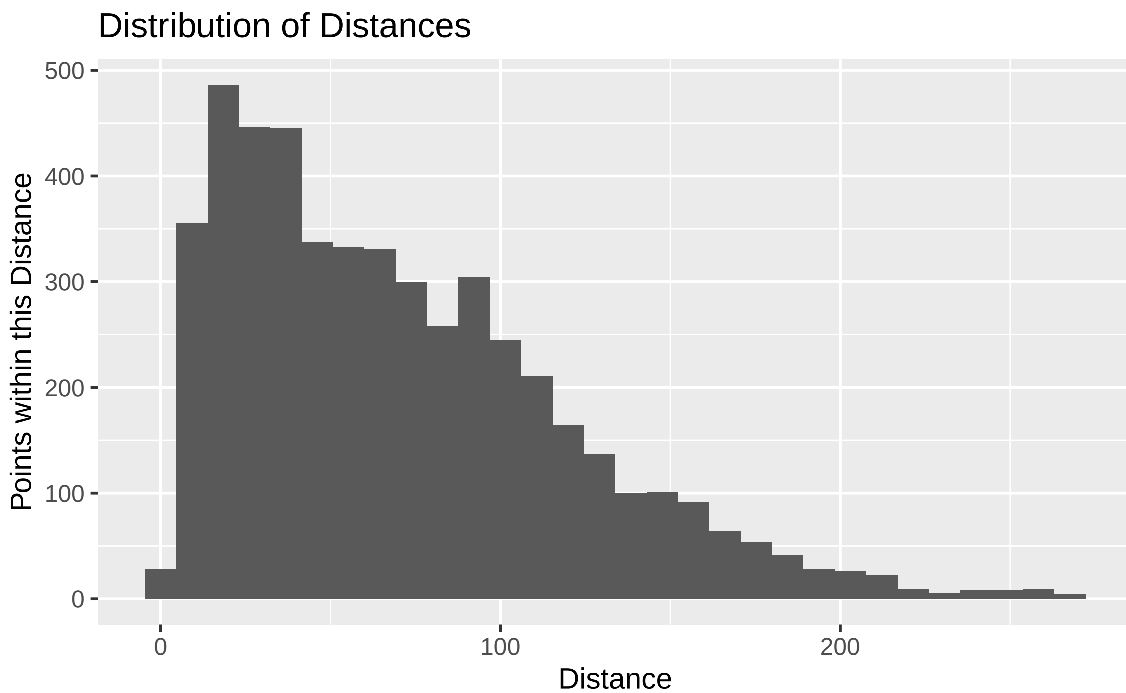
```r
##### Distance Matrix ####
# Euclidean distance, first 11 columns
# Not including the "Quality" indicator
d <- df[1:(ncol(df)-1)] %>%
  dist() %>%
  as.matrix()
```

Step 3: Choose ϵ

This is where things get a bit subjective

We can look at a histogram of the distances between points to get a feel for which ϵ might be most appropriate.

```r
##### Tidy Data ####
d_tidy <- TidyDistanceFrame(d)

##### Distance Distribution ####
d_tidy %>%
  ggplot(aes(Distance)) +
  geom_histogram(bins = 30) +
  ylab("Points within this Distance") +
  ggtitle("Distribution of Distances")
```



"Epsilon Frame"

We felt that just a histogram alone was inadequate to determine a sufficient ϵ, so instead we created a simple "Epsilon Frame," where we track the number of connections made out of the total possible for a given set, $\binom{n}{2}$, as well as the number of components present.

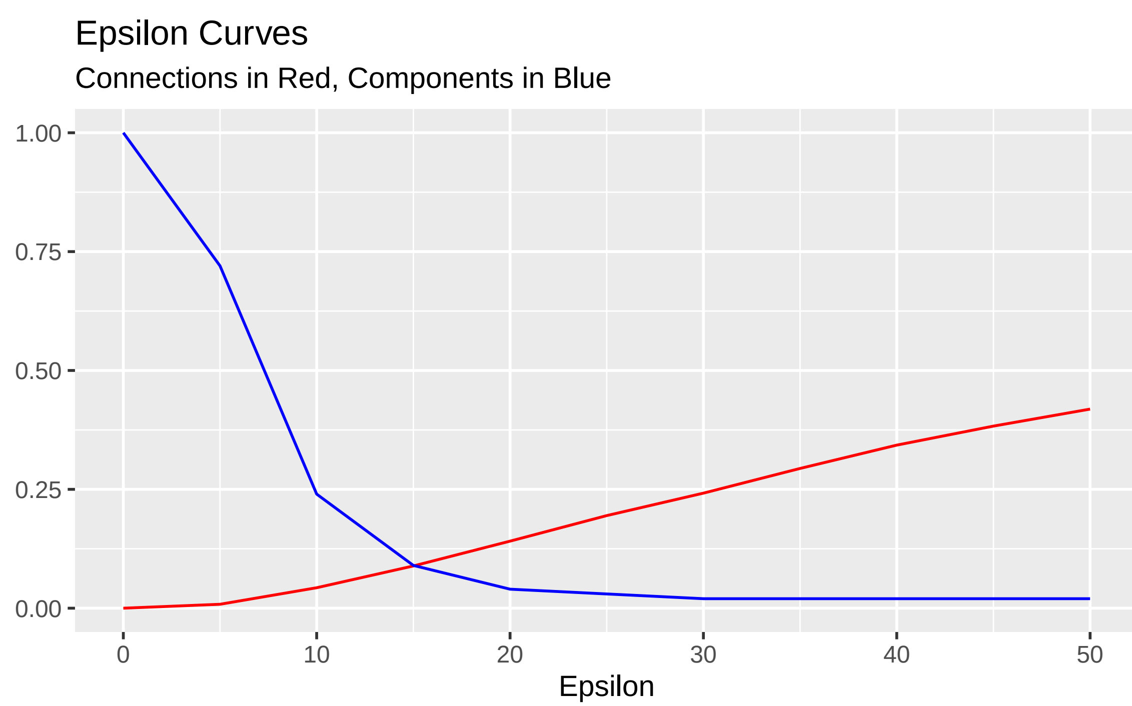
```
##### Create our "Epsilon Frame" ####
# From a vector of epsilon values
# we determine
# 1. The proportion of connections made
# 2. The number of components present
eframe <- CreateEpsilonFrame(d, seq(0, 50, by = 5))

##### Print Fancy Table ####
eframe %>%
  kable(caption = "Epsilon Frame")
```

Table 1: Epsilon Frame

Epsilon	Connections	Components
0	0.0000000	100
5	0.0082828	72
10	0.0430303	24
15	0.0888889	9
20	0.1410101	4
25	0.1947475	3
30	0.2420202	2
35	0.2939394	2
40	0.3430303	2

Epsilon	Connections	Components
45	0.3830303	2
50	0.4187879	2

```
##### Visualize Curves ####
eframe %>%
  mutate(Components = Components / 100) %>%
  ggplot(aes(x = Epsilon)) +
  geom_line(aes(y = Connections), color = "red") +
  geom_line(aes(y = Components), color = "blue") +
  ylab(NULL) +
  ggtitle("Epsilon Curves",
          "Connections in Red, Components in Blue")
```



Step 2: Open Covers

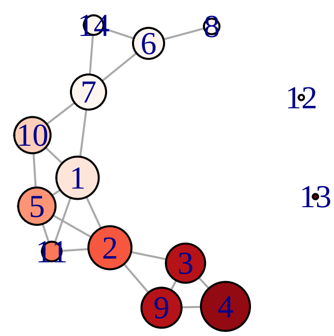
From our Epsilon Frame above, we opted to go with 25

```
epsilon <- 25
  ##### Open Covers ####
  # Distance matrix -> Adjacency Matrix -> Open Covers
  covers <- d %>%
    AdjacencyMatrix(epsilon) %>%
    OpenCoverEballs()

  ##### Adjacent Covers ####
  # Create a graph from an
```

```r
# Adjacency matrix of our covers
covers.plot <- covers %>%
  CoverAdjacencies() %>%
  graph_from_adjacency_matrix(mode="undirected")
```

Step 3: Plot it

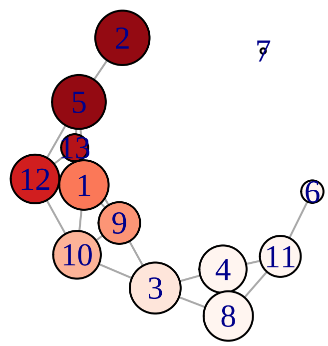
```r
##### Graph ####
  ##### Node Features ####
  # For each of our Covers (which becomes each of our Nodes)
  #   1) Turn the average number of Red wines in each Cover
  #       into an integer [0-11] + 1 to color the Node
  #      a) A Node of White wine only = 1 (white)
  #      b) A Node of Red wine only = 12 (darkpurple)
  #      c) All other Nodes are colored on a scale of [2-11]
  #   2) Size the each Node by the number of points (wines)
  #       in each Cover
  covers.size <- vector()
  covers.color <- vector()

  for (i in 1:length(covers)) {
    ##### Node Color ####
      # No Red Wines
    if ( mean( df$type[ covers[[i]] ] ) == 0 ) {
      covers.color[i] <- 1
      # No White Wines
    } else if ( mean (df$type[ covers[[i]] ] ) == 1) {
      covers.color[i] <- 12
      # All other Nodes
    } else {
      covers.color[i] <- round( mean( df$type[ covers[[i]] ]) * 10 ) + 1
    }
    ##### Node Size ####
    covers.size[i] <- length(covers[[i]])
  }

  ##### Coloring ####
  redscale <- brewer.pal(9, "Reds")
  redscale <- colorRampPalette(redscale)(12)
  V(covers.plot)$color <- redscale[covers.color]

  ##### Sizing ####
  V(covers.plot)$size <- log(covers.size) * 10

  ##### Plot ####
  plot(covers.plot)
```



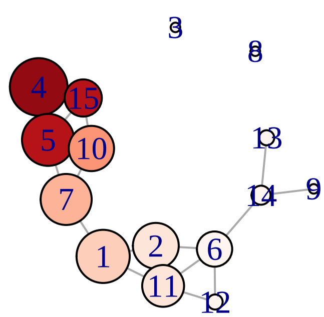
Rerun

```
##### Rerun the Mapper ####
# Each iteration takes a new sample of both data sets
map_iterations <- 10
for (i in 1:map_iterations) {
  RerunMapper(
        sample_size = 100,
              title = paste("Map", i))
}
```

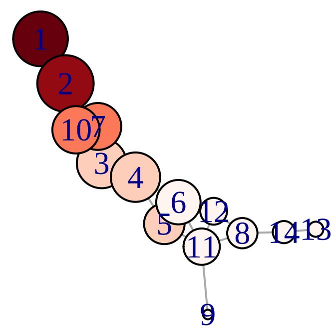
Map 1



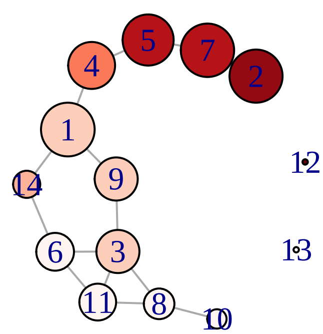
Map 2



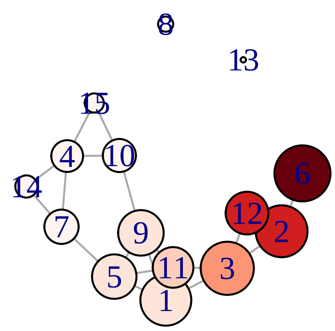
Map 3



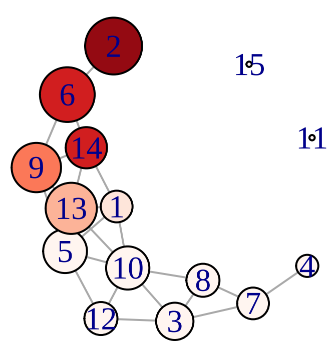
Map 4



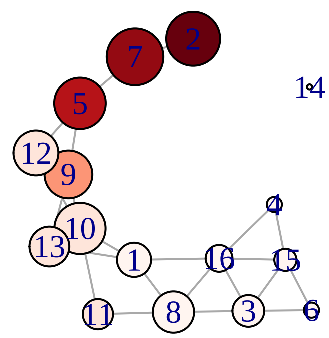
Map 5



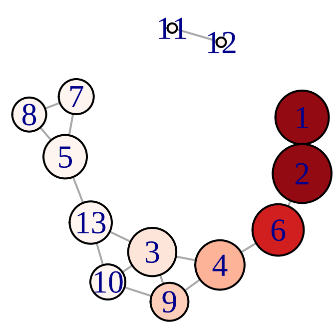
Map 6



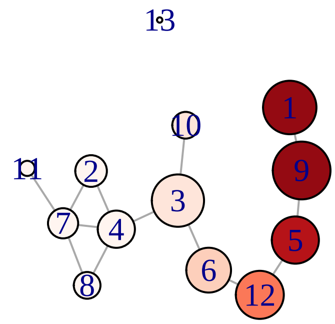
Map 7



Map 8



Map 9



Map 10