Notes on creating 2d plots with scaled points in R

Tyler Grimes

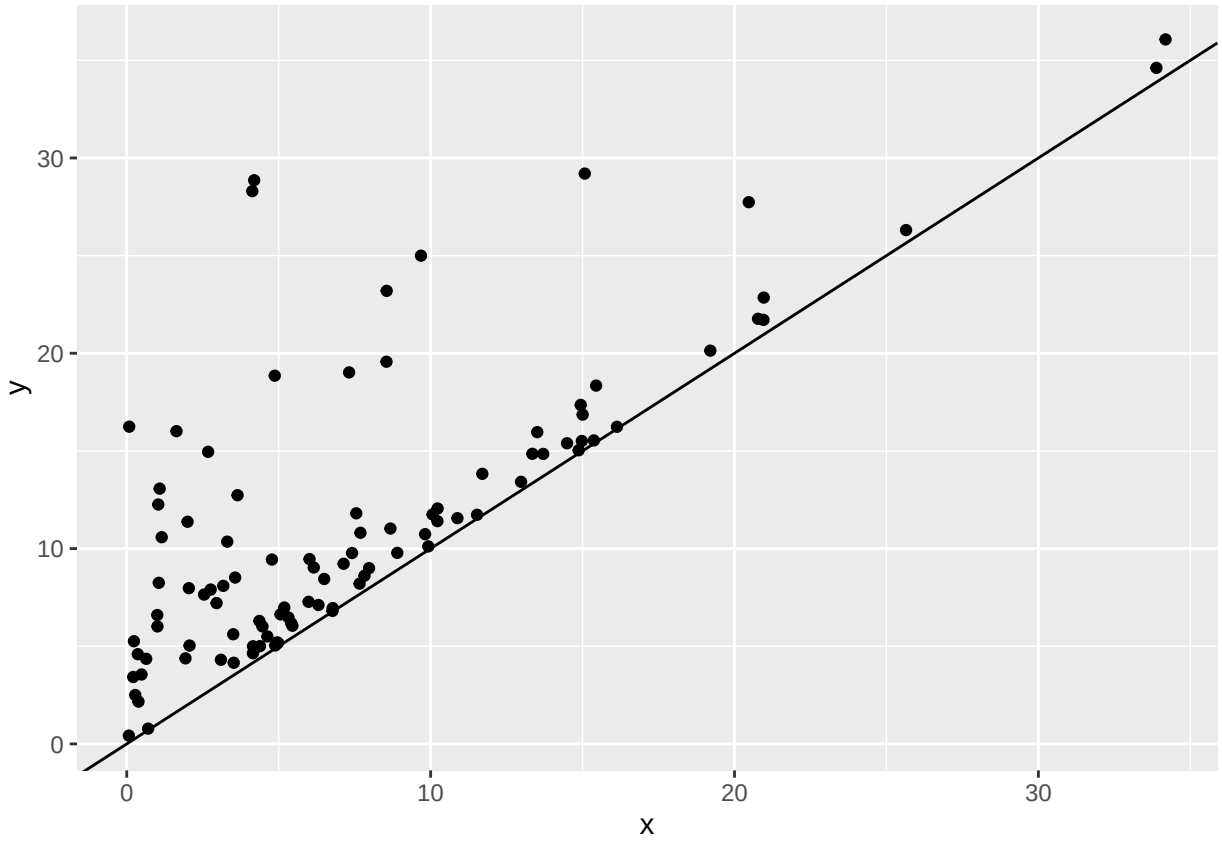
February 12, 2016

Suppose we have some data with two continuous variables that we want to graph. Let's generate such a data set and create a scatterplot. We will use the ggplot2 library throughout these notes. A basic scatterplot can be created using ggplot() and geom_points().

```
set.seed(7)
n <- 100 #Number of observations.

#Two continuous variables, x and y, with y > x.
x <- rexp(n, 1/8)
y <- apply(rbind(rexp(n, 1/8), x), 2, max) + abs(rnorm(n, 1, 1))

#Some categorical variable; can have any number of levels.
labels <- sample(c("Male", "Female", "Unkown"), size = n, replace = TRUE, prob = c(0.49, 0.46, 0.05))

dataset <- data.frame(x = x, y = y, labels = labels)
head(dataset)
```

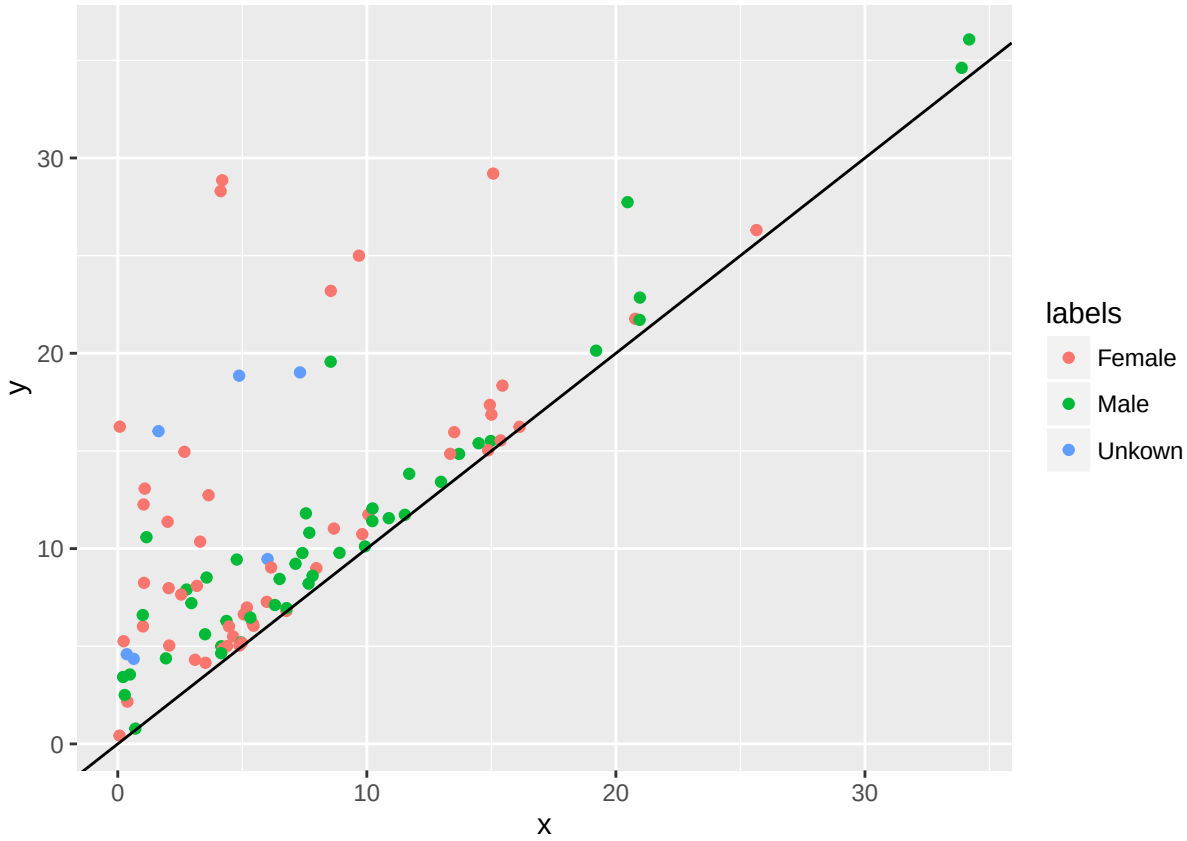
```
##            x         y labels
## 1  0.3867688  2.164599 Female
## 2 12.9760610 13.418927   Male
## 3 13.7051059 14.847355   Male
## 4  6.4975510  8.445236   Male
## 5  4.3649911  6.296662   Male
## 6 20.9621297 22.852688   Male
```

```
#Basic plot from ggplot2
ggplot(data = dataset) +
  geom_point(aes(x, y)) +
  geom_abline(intercept = 0, slope = 1)
```



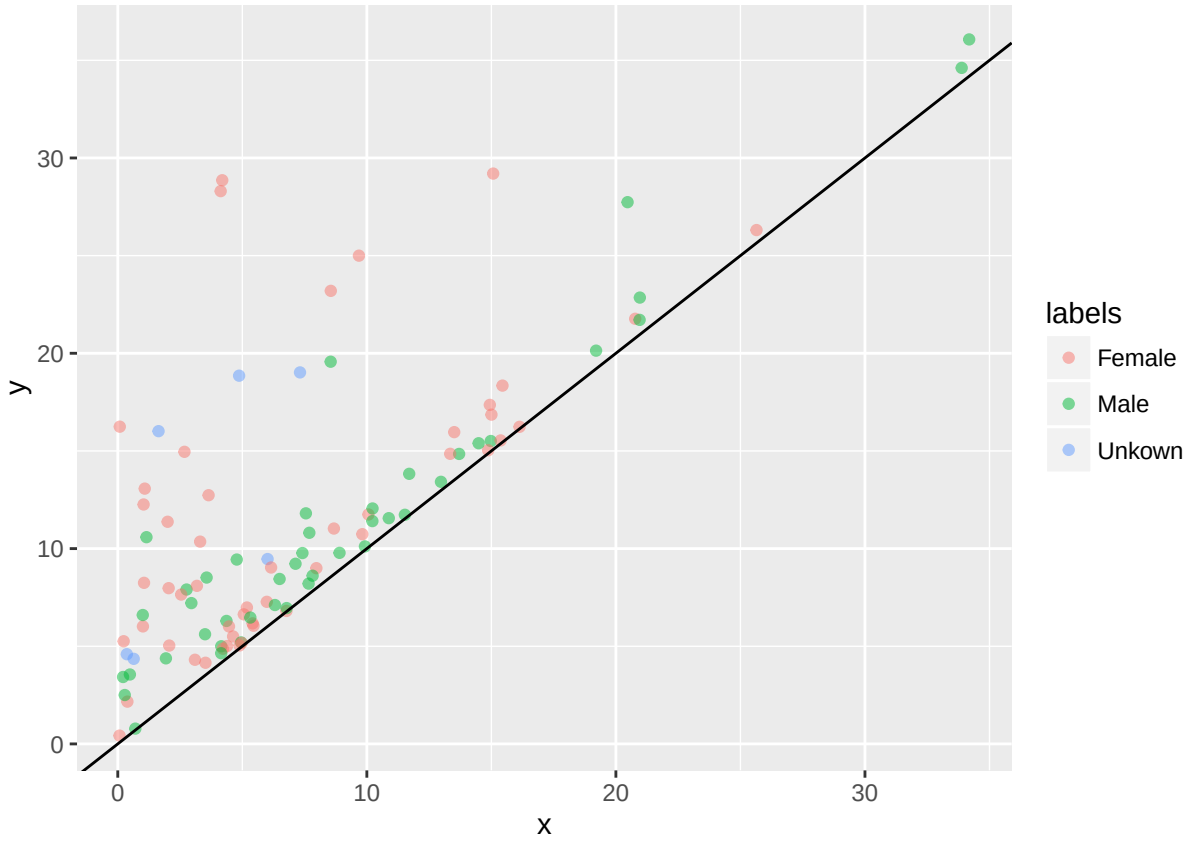
Now suppose the observations are categorized and we wish to color-code each point. This is done by setting another parameter (color) in the aesthetics of geom_point(). Note, by setting the parameter inside aes(), a legend is automatically created. If the legend is unwanted, we could set the parameter outside of the aesthetics method.

```
#Added color based on label.
ggplot(data = dataset) +
  geom_point(aes(x, y, color = labels)) +
  geom_abline(intercept = 0, slope = 1)
```



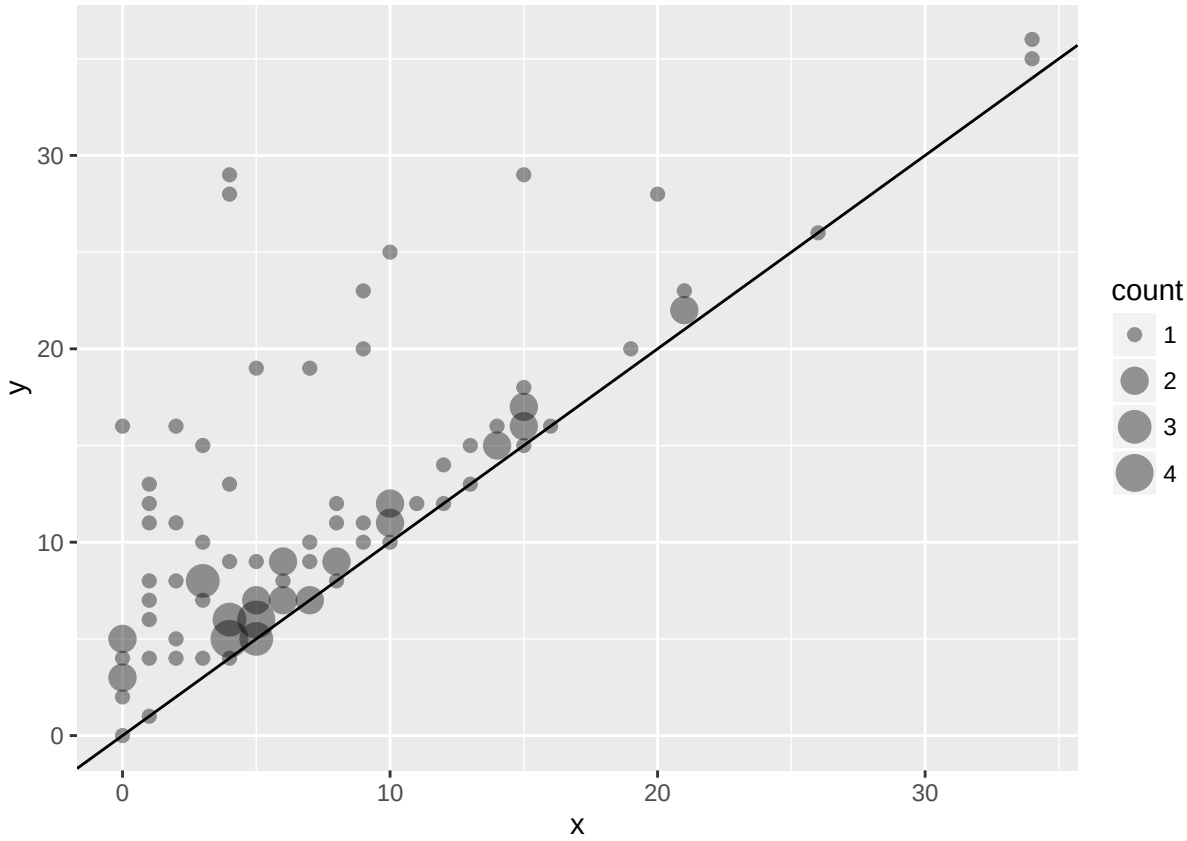
We now have colors, but some of the points overlap. If two points overlap, we are only able to see one of them and hence lose some information. To resolve this, the points can be made transparent by ajusting the alpha parameter (from 0-1: invisible to opaque). We do not want a legend created, so set alpha outside of aes().

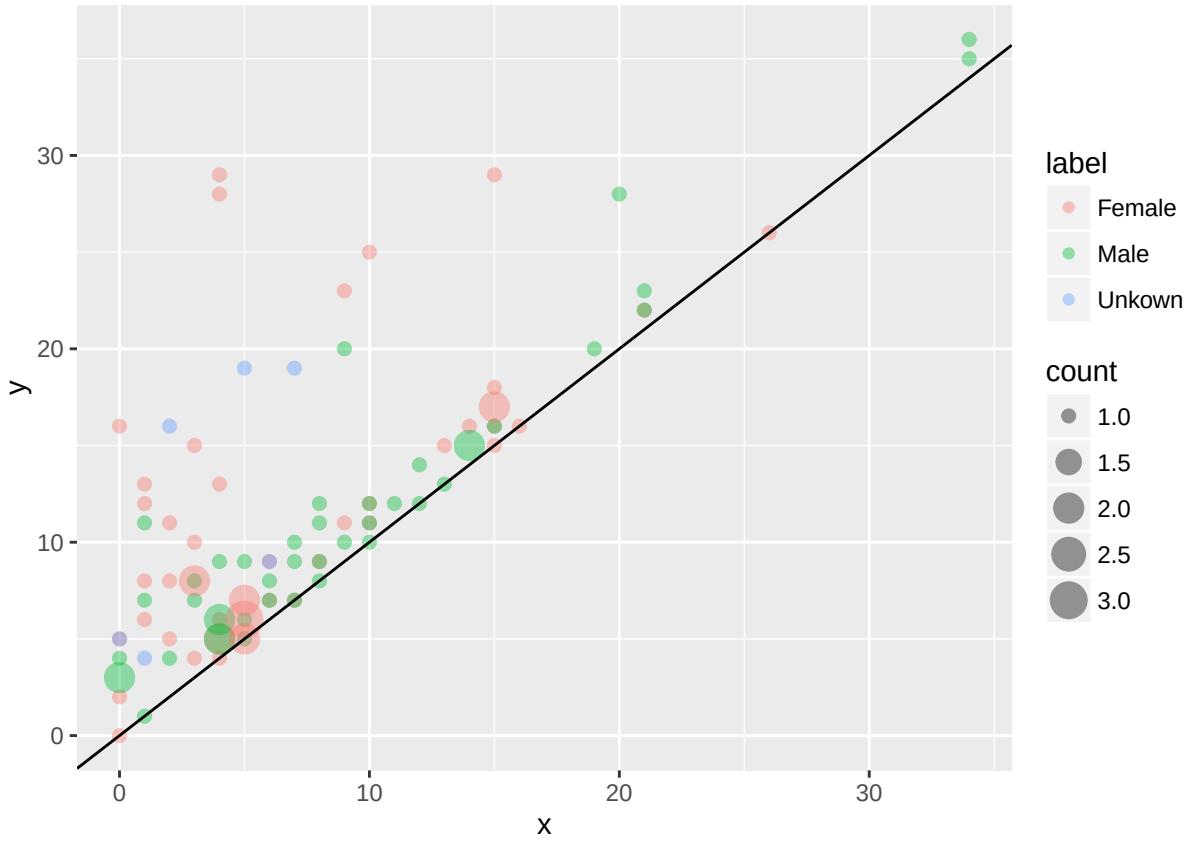
```
#Added color based on label.
ggplot(data = dataset) +
  geom_point(aes(x, y, color = labels), alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1)
```



Two overlapping points of different categories are now noticeable - the colors mix. If several points of the *same* color overlap, the point will become darker; so we can also see where the points are dense. We could stop at this stage, but let's consider another modification.

It would be nice if points very near each other would combine into one larger point. This would clean the graph up a little, and perhaps give another perspective on the data. If the variables were discrete, we could simply count the number of times an observation occurs at each coordinate and scale the points according to the count. With continuous data, we can follow this approach by first rounding to the nearest integer. A function plot.scaled.d() was created to implement this procedure, the code can be found in the appendix. Two plots from this method follows.


30
20
10
0
y
0
10
20
30
x
count
1
2
3
4



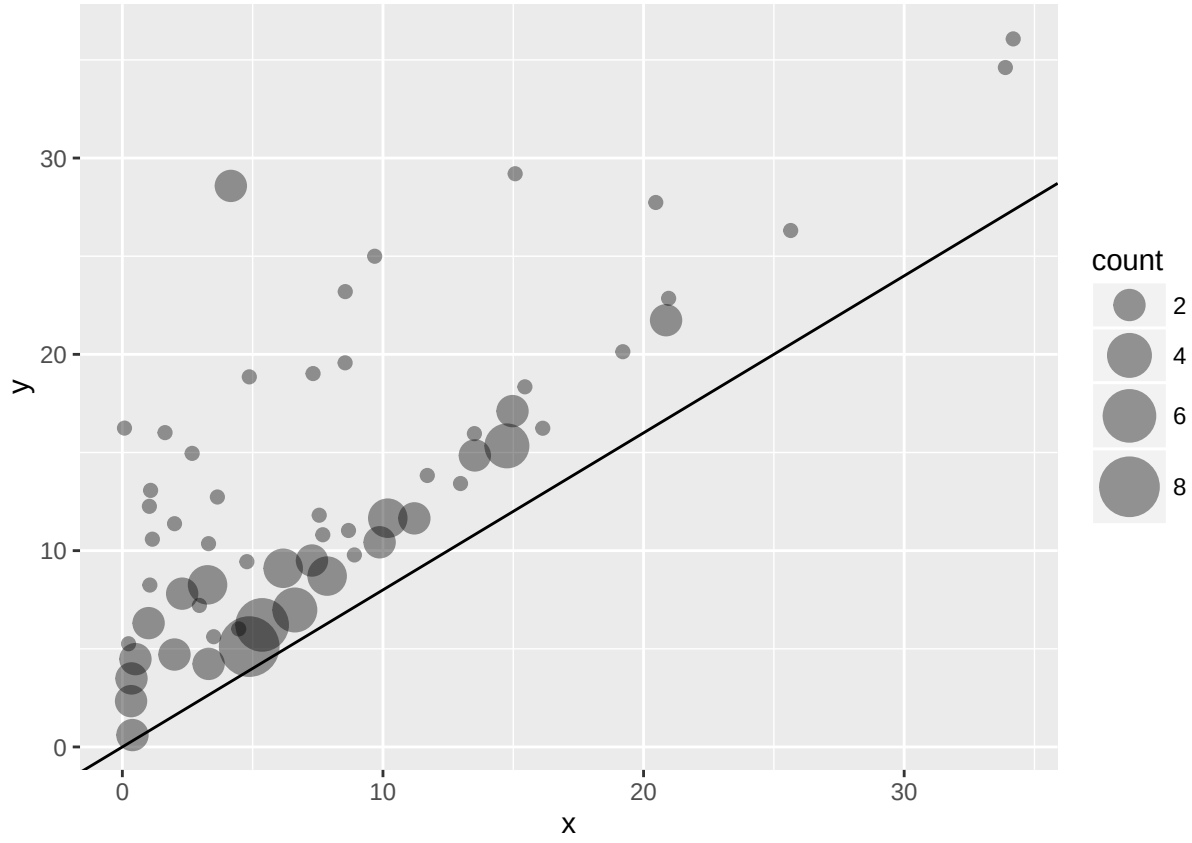
It seems that scaling the points has helped; the various clusters of points stand out more than before. The scaling can be adjusted according to preference in the scale_size_continuous() method.

For one last approach, rather than rounding the variables to get a discrete version, let's keep the original values. To apply the previous idea, we search the data for points that are near each other, using some distance measure. When two are found, they are combined by creating a new observation at their midpoint. The counts of the original two points are added together. This process is iterated until all points are suffenciently spread apart. plot.scaled() implements this procedure.

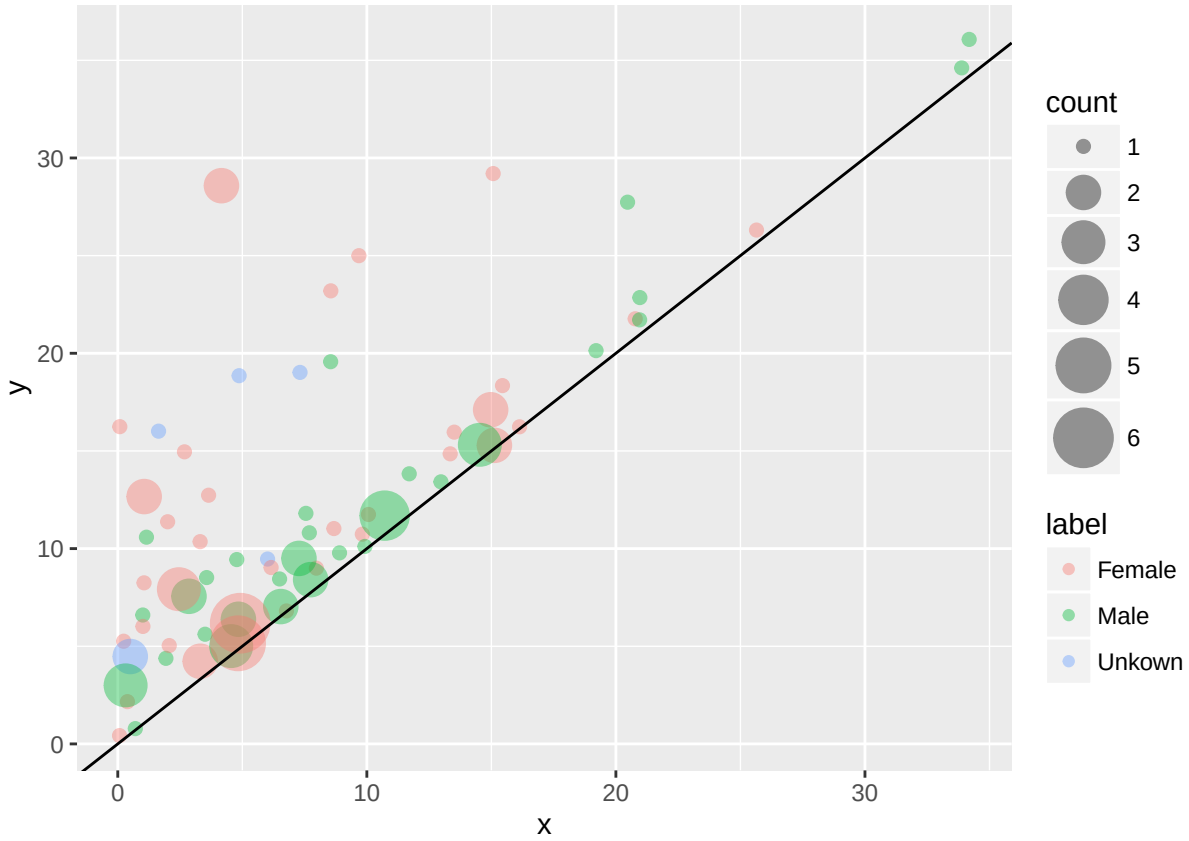
```
#Method for creating 2d plot with scaled points:
plot.scaled <- function(x, y, labels = NULL, scale_range = c(3, 7),
                        epsilon = 0.5, MAX_ITERATIONS = 50, max_size = 20,
                        xlab = "x", ylab = "y", main = "Scatterplot") {

  #Euclidean distance:
  d <- function(x1, y1, x2, y2) {
    return(sqrt((x1-x2)^2 + (y1-y2)^2))
  }

  #Method for obtaining counts:
  #Returns data frame with new x and y values, and their corresponding count.
  xy.with.counts <- function(x, y, epsilon = 0.5, MAX_ITERATIONS = 50) {
    #Number of observations.
    n <- length(x)

    #Create a vector of counts of length n initialized to 1.
    count <- rep(1, n)
```

```
  POINTS_MERGED <- TRUE
  iterations <- 0

  #Continue iterating until no points are merged.
  while(POINTS_MERGED && (iterations < MAX_ITERATIONS)) {
    POINTS_MERGED = FALSE
    iterations <- iterations + 1
    #Loop through each (x, y) data point.
    for(i in 1:n) {
      for(j in 1:n) {
        #If two points are within some distance,
        # and neither point hasn't already been absorbed.
        if(i != j && d(x[i], y[i], x[j], y[j]) < epsilon
           && count[i] > 0 && count[j] > 0) {
          x[i] <- (x[i] + x[j])/2
          y[i] <- (y[i] + y[j])/2
          count[i] <- count[i] + count[j]
          count[j] <- 0

          POINTS_MERGED = TRUE
        }
      }
    }
  }

  #Add a column to the table for the counts.
  xytable <- cbind(x, y, count = count)

  #Remove any coordinates that had zero counts.
  xytable <- xytable[count != 0, ]

  #Return the table.
  return(data.frame(xytable))
}

#Create a plot for the edited data (integer coordinates with counts).
plot <- NULL
xytable <- NULL

#If there are no labels, then create new table from entire data set.
if(is.null(labels)) {
  xytable <- xy.with.counts(x, y, epsilon, MAX_ITERATIONS)

  #Create plot.
  plot <- ggplot(data = xytable) +
    geom_count(aes(x = x, y = y, size = count), alpha = 0.4) +
    scale_size_continuous(range = c(2, max(c(10, xytable$count))))

#Else if there are labels, partition the data according to these and create
# new table from each partition.
} else {
  for(label in unique(labels)) {
    #Get counts table for each partition.
```

```
      #Append final table with each partition's data.
      xytable <- rbind(xytable,
                       cbind(xy.with.counts(x[labels == label], y[labels == label],
                                            epsilon, MAX_ITERATIONS),
                             label = label))
    }

    #Create plot.
    plot <- ggplot(data = xytable) +
      geom_count(aes(x = x, y = y, size = count, color = label), alpha = 0.4) +
      scale_size_continuous(range = c(2, max(c(10, xytable$count))))
  }

  return(plot)
}

#Without labels.
plot1 <- plot.scaled(x, y, epsilon = 0.8) +
  geom_abline(intercept = 0, slope = 0.8)
plot1
```




```
plot2 <- plot.scaled(x, y, labels, epsilon = 1) +
  geom_abline(intercept = 0, slope = 1)
plot2
```



These graphs more accurately represent the data, since we are not forcing the values into integers. There's one issue with this implementation though; the points are searched in the order that the data are in, so the center of the points may wander unpredictably. This is pronounced in the non-labeled example - the points move away unexpectedly from the $y = x$ axis. A better method might search for the *nearest* points (rather than *a nearby* point) to merge together at each iteration (similar to a k-means clustering algorithm). Also, as written this code may not scale well with the nested for-loops, but a vectorized version could be created.

Other Resources

ggplot2 help topics
jitter
hexbin/heatmap

Appendix

Code for plot.scaled.d().

```
#Method for creating 2d plot with scaled points:
plot.scaled.d <- function(x, y, labels = NULL, scale_range = c(3, 7),
                          xlab = "x", ylab = "y", main = "Scatterplot") {

  #Method for obtaining counts:
  xy.with.counts <- function(x, y) {
```

```
  #Round x and y vals to nearest integer.
  x <- round(x)
  y <- round(y)

  #Find maximum x value (used for indexing the following table).
  x.max <- max(x)

  #Create a table with all permutations of x and y values.
  xytable <- expand.grid(x = seq(0, x.max, 1), y = seq(0, max(y), 1))

  #Create a vector of counts with the same length as table.
  count <- vector("integer", length = nrow(xytable))

  #Loop through each (x, y) data point.
  for(i in 1:length(x)) {
    #Find the corresponding index in the table for the (x, y) coordinate.
    index <- (x.max + 1)*(y[i]) + (x[i]) + 1

    #Increment count for that coordinate.
    count[index] <- count[index] + 1
  }

  #Add a column to the table for the counts.
  xytable <- cbind(xytable, count = count)

  #Remove any coordinates that had zero counts.
  xytable <- xytable[xytable$count != 0, ]

  #Return the table.
  return(xytable)
}

#Create a plot for the edited data (integer coordinates with counts).
plot <- NULL
xytable <- NULL

#If there are no labels, then create new table from entire data set.
if(is.null(labels)) {
  xytable <- xy.with.counts(x, y)

  #Create plot.
  plot <- ggplot(data = xytable) +
    geom_count(aes(x = x, y = y, size = count), alpha = 0.4) +
    scale_size_continuous(range = c(2, max(c(6, xytable$count))))

#Else if there are labels, partition the data according to these and create
# new table from each partition.
} else {
  for(label in unique(labels)) {
    #Get counts table for each partition.
    #Append final table with each partition's data.
    xytable <- rbind(xytable,
                     cbind(xy.with.counts(x[labels == label],
```

```
                                          y[labels == label]),
                            label = label))
    }

    #Create plot.
    plot <- ggplot(data = xytable) +
      geom_count(aes(x = x, y = y, size = count, color = label), alpha = 0.4) +
      scale_size_continuous(range = c(2, max(c(6, xytable$count))))
    }

  return(plot)
}

#Without labels.
plot <- plot.scaled.d(x, y)
plot + geom_abline(intercept = 0, slope = 1)

#With labels.
plot <- plot.scaled.d(x, y, labels)
plot + geom_abline(intercept = 0, slope = 1)
```