Fall 2021

GIplot: An R Package for Visualizing the Summary Statistics of a Quantitative Variable

Siddhanta Phuyal
DePauw University, siddhantaphuyal_2023@depauw.edu

Mamunur Rashid
DePauw University, mrashid@depauw.edu

Jyotirmoy Sarkar
Indiana University Purdue University Indianapolis

GIplot: An R Package for Visualizing the Summary Statistics of a Quantitative Variable

Authors

Siddhanta Phuyal
DePauw University, Department of Mathematical Sciences
2 E. Hanna Street, Greencastle, IN 46135 USA
ORCiD: 0000-0002-0675-3033
siddhantaphuyal_2023@depauw.edu

Mamunur Rashid
DePauw University, Department of Mathematical Sciences
2 E. Hanna Street, Greencastle, IN 46135 USA
ORCiD: 0000-0001-8759-3803
mrashid@depauw.edu

Jyotirmoy Sarkar
Indiana University-Purdue University Indianapolis
Department of Mathematical Sciences
402 N. Blackford Street, Indianapolis, IN 46202 USA
ORCiD: 0000-0001-5002-5845
jsarkar@iupui.edu

Abstract

A GI plot is a graphical tool that pictorially represents the five-number summary, the mean, the standard deviation, the sample size, and flags potential outliers that are *c* standard deviations away from the mean together with their frequencies. The GI plot was developed by Sarkar and Rashid (2021) as a companion to the boxplot. The purpose of this paper is to provide an overview of the `GIplot` package and introduce the main functionality of the package using several examples.

Keywords: Five-number summary; Seven-number summary; Gaussian intervals; R; GIplot package.

1. Introduction

The GI plot was introduced by Sarkar and Rashid (2021) as a companion to the boxplot (Spear, 1952, 1962; Tukey, 1977). A standard boxplot depicts five-number summary and flags potential outlier values but does not report the frequencies of such values. A GI plot displays the seven-number-summary, the sample size, and potential outliers, including their counts.

The purpose of this paper is to introduce the user to the functionalities of the `GIplot` package. The paper is organized as follows: In section 2, we briefly explain the limitations of boxplot and how `GIplot` remedies those limitations; then, in section 3, we introduce the functions and features of the package along with some illustrative examples.

2. `GIplot` as a companion to Boxplot

A boxplot is a pictorial representation of the five summary statistics: minimum, lower quartile, median, upper quartile, and maximum. A box is drawn beginning at the lower quartile and ending at the upper quartile and split at the median. Also, two whiskers are drawn stretching outside the box: The left whisker stretches to one and half times the interquartile range below the lower quartile and the right whisker to one and half times the interquartile range above the upper quartile. A boxplot identifies outliers, which are observations outside the boundary of the whiskers. However, there are two limitations to a boxplot. First, a boxplot does not show the mean and the standard deviation, which are two important pieces of information for a distribution. Second, a boxplot does not show the frequencies of the outliers. Developed as a companion to a boxplot, a GIplot shows all the information present in a boxplot along with the mean, the standard deviation, the sample size, and the frequencies of outliers.

We modify the standard boxplot in the following ways to construct a GI plot:

a) As the height of the box in a boxplot is non-informative, we replace the box with a straight dashed line enclosed by parentheses. The left parenthesis represents the value in the dataset at or above the mean minus c*sd, and the right parenthesis is the value at or below the mean plus c*sd. Any values that lie outside of parentheses are flagged as outliers, together with a special symbol to denote their frequencies. See below for the symbols and their specific meanings.

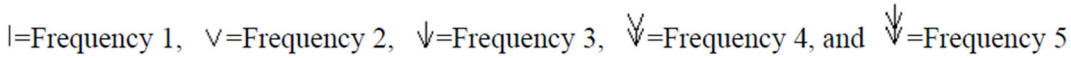

Figure 1: Outlier Frequency symbols and their meanings.

b) On the dashed line, we add a small line segment which is perpendicular to and intersects with the dashed line. The line segment represents the median. We also add open and closed square brackets which represent the lower and upper quartiles, respectively. Additionally, we add an arrow (in solid, bold) whose starting point represents the mean, and whose length represents the standard deviation.

c) We print the sample size adjacent to the GIplot; however, it is optional, and the user may choose to remove the sample size.

3. Overview of the package

The `GIplot` package was developed to help users easily construct a GIplot. The package has a generic function `GIplot()` which selects the appropriate method based on the class of the input: If the class of the input is a formula, then the formula method, `GIplot.formula()`, is used; for any other class of input, the default method, `GIplot.default()`, is used. A disclaimer is warranted: The default method is designed to work for double, integer, list, and data.frame class only. For any other class of input such as matrix, `GIplot()` may not produce the correct result.

The formula method, `GIplot.formula()`, builds on the default method to produce the GIplot. First, the formula method constructs a data.frame object using the `model.frame()` function and the user provided dataset. The first column of the newly constructed data.frame object contains the long data which has to be grouped based on other variables. The data for other grouping variables are present in the other columns of the data.frame object. The method, then, extracts the grouping variables and constructs a vector. The `split()` function uses the vector of grouping variables to split the long data into multiple groups. The list obtained from the `split()` is then fed to the `GIplot.default()` to construct a GIplot. The R code for constructing GIplot using the formula method is:

```
GIplot.formula <- function (formula, dataset = NULL,horizontal=
TRUE,names=c(),add=FALSE,at=0,valueOfc=2.33,axisLabel="",main=paste("GIPlot
of ",axisLabel),spsize=T,...){
  df <- model.frame(formula,data = dataset)
  numberOfColumns<- ncol(df)
  vec <- c()
  for (d in 2:numberOfColumns){
    vec <- c(vec,df[d])
  }
  e<- split(df[1],vec)
  if(length(names)==0) {
    names = names(e)
  }
  GIplot.default(e,names = names,horizontal=horizontal,add=add,at=at,
                 valueOfc=valueOfc,axisLabel=axisLabel,main=paste("GIPlot of
",axisLabel),spsize = spsize)

}
```

The first parameter for `GIplot.default()` is `x`, which is the data for which GIplot is to be plotted. The second parameter, …, ensures that the user can feed any number of datasets. However, the user should note that the class of all the datasets should be consistent. For example, If the class of the first dataset is double, then the other datasets should also be doubles. If the classes of the datasets are mixed, the result may not be correct. Although the purpose of … is to allow the user to feed any number of datasets, it can also accept a set of random numbers or other user-specified variables that may not be meaningful. Thus, the user may feel that the function is accepting their random parameters; however, the user should note that such variables do affect the result. Hence, it is strongly advised that users name the parameters whenever they wish to provide values for the

parameters. For example, when providing the value for the parameter `valueofc`, the user should use `GIplot (x, …, valueofc=2)` instead of `GIplot(x,…,2)`.

At first, the function creates a list of the datasets from the datasets provided by the user through the function's parameter. For example, If the classes of all datasets are double, then the function creates a list of double vectors. The code used is:

```
#creating a list of the datasets
  arguments <- list(...)
  if ((typeof(x)=="double")||(typeof(x)=="integer")){
    arguments <- c(list(x),arguments)
  }
  if (typeof(x)=="list"){
    arguments <- c(x,arguments)
  }
  if (typeof(x)=="data.frame"){
    li <- list()
    nOfCol = ncol(x)
    for(h in 1:nOfCol){
      li <- c(li,x[h])
    }
    arguments <- c(li,arguments)
  }
```

Then, the function finds the maximum and the minimum value from the list. Note that this is the maximum and the minimum of all the datasets. The maximum and the minimum are used to calculate the `xlim` and `ylim` which are used by the `plot()` function to determine the plot area. The following code has been used to find the maximum and the minimum value:

```
#finding the maximum and minimum value in the whole list of datasets
  n <- length(arguments)
  xlimitmin <- 0
  xlimitmax <- 0
  for(u in 1:n){
    g <- unlist(arguments[u])
    if (u==1){
      xlimitmin <- min(g)
      xlimitmax <- max(g)
    }
    if (u>1){
      if(min(g)<xlimitmin){
        xlimitmin <- min(g)
      }
      if(max(g)>xlimitmax){
        xlimitmax <- max(g)
      }
    }
  }
```

The function uses a for-loop to handle each dataset in the list. For the first dataset, `plot()` function is used to create a new plotting window with the optional parameter `add = FALSE`. On the other hand, if `add = TRUE`, then the next GIplot is superimposed on the existing plot. For example, if the user wants to superimpose GIplot on a scatter plot, the user has to produce a scatter plot first

and then use `GIplot()` function with `add = TRUE`. Note that the default value for `add` is `FALSE`. The function also allows the user to decide the location of the GIplot when they are using `add = TRUE`. The parameter, `at`, takes the value and prints the GIplot at that location. The default value for `at` is 0. However, depending on the plot window, GIplot may not be visible if it is drawn outside the visible window. So, the user should carefully choose the value of the parameter `at`. The `add = TRUE` and `at` parameters can be used along with the parameter `horizontal = FALSE/TRUE` to produce either vertical or horizontal GIplots at the desired location on the existing plots. This feature can also be used if the user finds the axis produced by the `GIplot()` function undesirable. For example, in the following case the user may find interpreting the GIplot difficult because of axis being shorter than needed.

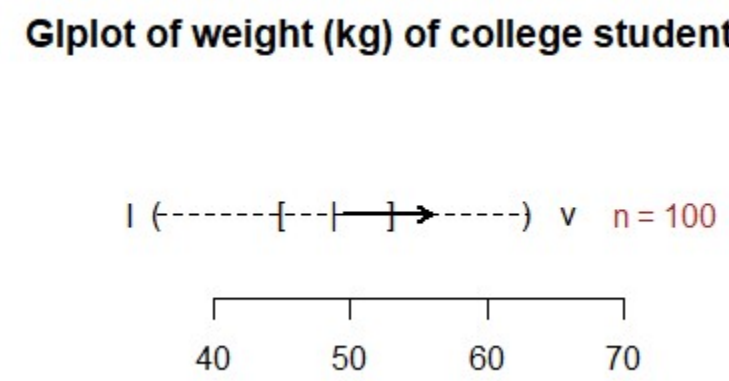

Figure 2: GIplot with undesirable axis produced by `GIplot()` function.

In such cases the user can use the add and at parameter along with the `plot()` function to produce desired output. An example is shown below:

```
data<- c(53,52,54,42,46,57,53,49,50,46,54,53,45,48,
         54,54,66,44,45,34,63,52,56,45,60,59,59,46,
         49,46,54,50,52,48,49,52,50,42,51,43,57,51,
         51,62,37,62,47,36,49,46,53,42,53,37,48,42,
         40,53,56,37,50,49,43,45,48,58,56,50,50,55,
         48,53,46,49,51,66,49,49,45,53,38,49,44,51,
         42,49,53,58,39,54,44,46,54,49,43,50,40,49,
         57,40)
plot(c(),c(),xlim = c(30,70),ylim = c(1,3),
     main ="GIplot of weight (kg) of college students",
     frame.plot = F,yaxt="n",ylab = "",xlab = "")
GIplot(data,horizontal = T,add=T,at=2)
```

The output obtained from the above block of codes is:

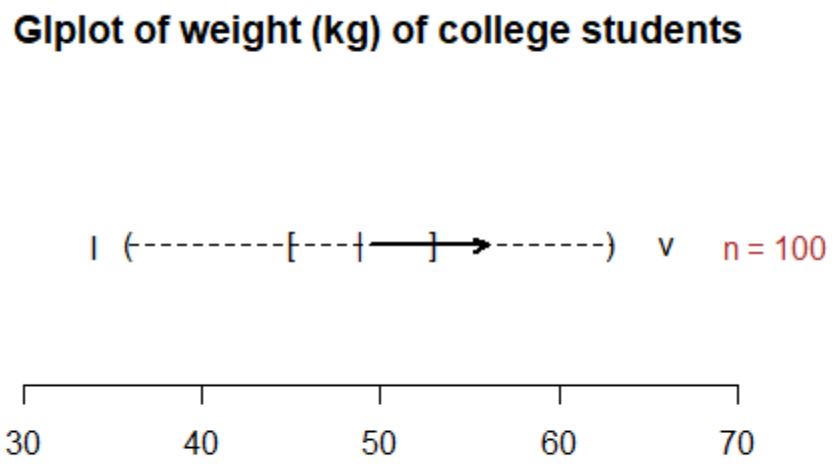

Figure 3: GIplot reproduced using `plot()` function and parameters: add and at.

However, if the user wants to create a GIplot on a new plotting window, the user does not need to worry about the parameters `add` and `at`. The first GIplot is printed just above the axis if the `horizontal = TRUE`, which is the default option. GIplots for subsequent datasets are printed at some value above the prior GIplots. The order in which GIplots are printed follows the order in which the datasets are provided. If the user changes the value for `horizontal` to `FALSE`, vertical GIplots are produced with the first being to the right of the vertical axis and subsequent GIplots to the right at regular gaps.

A center justified axis label is printed just below the horizontal axis if `horizontal = TRUE`. If `horizontal = FALSE`, the center justified axis label is printed to the right of the vertical axis. The axis label can be changed through the parameter `axisLabel`. Initially the main title of the GIplot is "GIplot of '' pasted with the axis label. However, the user can change the title by using the parameter `main`. If the user does not want to add the axis label and the title to their GIplot, they can pass an empty string to the respective parameters.

If more than one GIplots are printed, the package provides an option to name the respective GIplots. The names to the GIplots are printed on the vertical axis if the orientation of the GIplots is horizontal and vice versa. The user can provide names through the parameter `names`, which can be either a logical value specifying whether the annotations are to be made at the tickmarks or a character or expression vector of labels to be placed at the tickmarks. If the user does not modify this parameter, the default numerical annotations are printed at the tickmarks. However, if the user decides to modify this parameter, the user should note that the number of strings or characters in the vector for the `names` parameter should match the number of GIplots printed. If the numbers do not match, an error message is printed. We have decided not to print the name of the GIplot in the case of a single GIplot because the user can include that information either through the axis label, or through the title.

The R codes used for plotting, axis labels, and naming the GIplots are:

```
#plot function and title of the plot used only once if add ==FALSE
    if((j==1)&(add==FALSE)){
      plot(x,y,frame.plot = F,axes =FALSE,xlab="",ylab="",type="n"
           ,xlim = xlimit ,ylim = ylimit,las=1)
      if(horizontal==T){
        title(main = main, line = 2)
      } else{
        title(main = main, line = 3)
      }
    }

    #horizontal or vertical axis, axislabel and names of variables if there
is more than one GIplot.
    if (add==FALSE){
      if (j==n){
        axis(sideOfAxis,pos=ht,xpd=TRUE,las=1)
        mtext(text=axisLabel,side = sideOfAxis,line=2)
        if (sideOfAxis==1){
          if(n>1){
            axis(2,at=c(seq(1,h,1)),labels = names,las=1,xpd=TRUE,padj =
0.5,hadj=1)
          }
        } else {
          if(n>1){
            axis(side = 1,at=c(seq(1,h,1)),labels = names,xpd=TRUE,padj =
1,las =1,hadj=0.5)
          }
        }
      }
    }
```

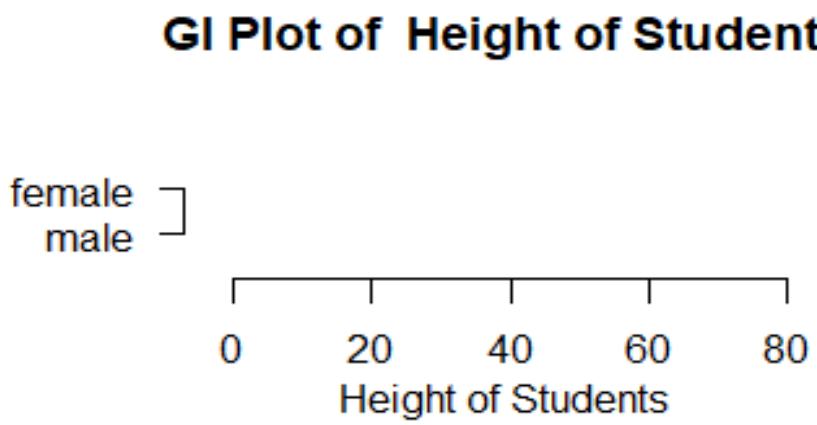

Figure 4: The title, axis, axis-label, and labels for the GIplot.

Now we explain the components of the GIplot and how the package constructs the GIplot.

GIplot uses the mean and the standard deviation to determine the regions for outliers. The parameter `valueofc`, is the multiplier for detecting the outliers in the dataset. The name `valueofc` is used because in our code we have used the mean plus/minus '`c`' times the standard deviation to determine the region for outliers where `c` is the multiplier. From here on, we may use '`c`' and '`valueofc`' interchangeably. The default value for `c` is 2.33, which is the 99th percentile of the standard normal distribution, assuming that the given data are (approximately) normally distributed. Recall that a boxplot flags as outliers those observations that fall below Q1-1.5*(Q3-Q1) or above Q3+1.5*(Q3-Q1). As a companion to a boxplot, we have used an alternative method to detect outliers. The method used in GIplot is suitable for (approximately) normal distributions.

In the package, the `text` function is used to print '(' at the location of a value in the dataset at or above the mean minus c*sd. Similarly, ')' is printed at the location of a value in the dataset at or below the mean plus c*sd. Note that because of the above-described refinement, the lengths of the two whiskers may be differently shortened and therefore the parentheses may not be symmetric around the mean. The two parentheses are joined by a dashed line segment which is produced using segments. Any data values that are outside the interval enclosed between '(' and ')' are flagged as outliers. This interval is the shortest interval which contains a suitable percentage of the variable under a normal probability model. For example, if `c = 2.33`, the interval is the shortest interval that includes 98 percent of the data, and misses 1% of data at each of the lower end and the upper end. The user may modify '`c`' if they wish to alter the probability of inclusion.

Similarly, the function `text` is used to print '[' at the lower quartile, '|' at the median, and ']' at the upper quartile. The package uses the `arrow` function to superimpose a bolded arrow on the dashed segment that connects '(' and ')'. The tail end of the arrow represents the mean, and the length of the arrow represents the standard deviation. Note that the symbols used above are true if the GIplot is `horizontal`. On the other hand, if the GIplot is vertical, the symbols are rotated at an angle of 90 degrees counterclockwise.

The package also provides an option to print the sample size for the respective GIplots. For the horizontal GIplots the sample size is printed to the right of the GIplot and for the vertical GIplots the sample size is printed above the GIplot. The color of the text for printing the sample size is brown. Often the user forgets to include the sample size in their analysis of the data. A boxplot also does not include the sample size. We believe it to be an important piece of information. Thus, the sample size is displayed. However, the user can suppress it by using the parameter `spsize`. It takes the logical value, and the default value is `TRUE`. The user can suppress the sample size by passing `FALSE` to `spsize`.

The R code for printing GIplot and the spsize is:

```
    #braces and notches for the quartiles, median, and the outliers' boundary
    text(xOne,yOne,labels = "|",srt=angle,col = "black",xpd=add)
    text(xTwo,yTwo,labels = "[",srt=angle,col = "black",xpd=add)
    text(xThree,yThree,labels = "]",srt=angle,col = "black",xpd=add)
    text(xFour,yFour,labels = "(",srt=angle,col = "black",xpd=add)
    text(xFive,yFive,labels = ")",srt=angle,col = "black",xpd=add)

    #dashed straight line, arrow from the mean, and sample size.
    segments(xFour,yFour,xFive,yFive,lty = 2,xpd=add)
    if (horizontal==TRUE){
      arrows(mean(x),h,mean(x)+sd(x),h,length = 0.08,lty = 1,lwd = 2,xpd=add)
      if(spsize==T){
        text(max(x)+0.1*(max(x)-min(x)),yFive,labels = paste("n
=",length(x)),col = "brown",xpd=T,adj=0)
      }
    } else {
      arrows(h,mean(y),h,mean(y)+sd(y),length = 0.08,lty = 1,lwd = 2,xpd=add)
      if(spsize==T){
        text(xFive,max(y)+0.2*(max(y)-min(y)),labels = paste("n
=",length(x)),col = "brown",xpd=T,adj=c(0.5,0))
      }
    }
```

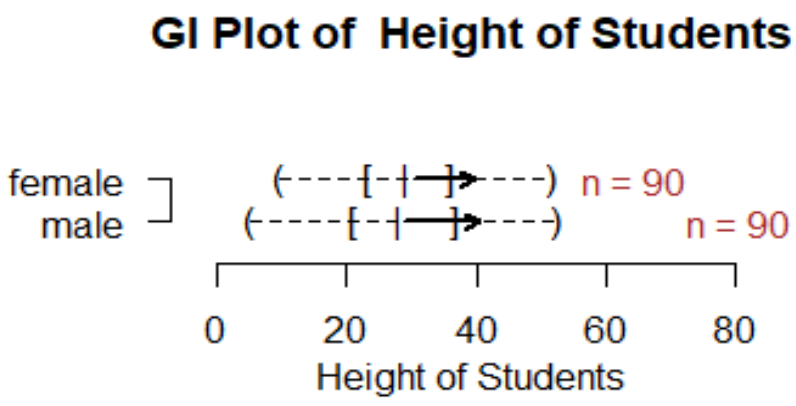

Figure 5: GIplot without the outlier frequencies.

There are two parts to displaying outliers in a GIplot: Outlier detection and display symbols for the number of outliers. The package sorts the data in the dataset in ascending order using the `sort` function. Then, it uses a `while` loop to go through the sorted data. The package starts with the smallest datum which is compared to mean plus/minus c*sd to determine whether it is an outlier or not. If the datum is less than mean *minus* c*sd or greater than mean *plus* c*sd, it is declared as an outlier. Once the data is determined an outlier, the function counts its frequency. Since the data are arranged in an ascending order, it checks whether the next datum is equal to the current data in the loop. If two adjacent data are equal the frequency count is increased by one. The count breaks when the next datum is different from the current datum, and the frequency of the current datum is noted. Next, the package calls another implicit function called `display` whose purpose is to display the symbol representing the frequency of the outlier. Note that the display symbols are not available to users to modify. It is used by the `GIplot` function to display the outliers. Once the symbol is printed, the count begins afresh from the next data value. The process is repeated until the last datum is scanned.

The R code used to detect the outliers and display the symbols is:

```
#finds the outliers and count their numbers.
    i <- 1
    while(i<=length(sortedx)){
      count <- 0
      if(sortedx[i] < mean-c*sd || sortedx[i] > mean+ c*sd){
        count <- 1
        if(i < length(sortedx)){
          while(sortedx[i+1]==sortedx[i]){
            count <- count +1
            i<- i + 1
            if(i == length(sortedx)){
              break
            }
          }
        }
      }
      display(count,sortedx[i],h)
      i<- i+1
    }
```

The `display` function works like any other function. The parameters for display function are the frequencies of the data values which determine the symbol to display, and two other quantities which determine the location at which the symbol is to be printed. The function uses '`if`' and '`else if`' statements to determine the symbol to display. See Figure 1 to understand the symbols and their meanings.

If the outlier frequency is greater than five, then the symbol for frequency 5 is printed first and the additional frequency is printed either above or to the left of the already printed symbol according as the orientation of the plot is horizontal or vertical. For example,

The figure below represents the frequency 7 if the GIplot is horizontal.

Figure 6: The symbol used if the outlier frequency is 7.

The R code for the display function is:

```
#displays the number of outlier diagram in the plot.
    display<-function(nOut,valOfx,valOfy){
      h <- valOfy
      if(nOut == 1){
        if (horizontal==TRUE){
          text(valOfx,h,labels=expression("I"),col="black",xpd=add)
        } else {
          text(h,valOfx,labels=expression("I"),srt=angle,col="black",cex =
1.2,xpd=add)
        }
      } else if(nOut == 2){
        if(horizontal==TRUE){
          text(valOfx,h,labels = expression("v"),col="black",xpd=add)
        } else {
          text(h,valOfx,labels = expression(">"),col="black",cex =
1.2,xpd=add)
        }
      } else if(nOut == 3){
        if (horizontal==TRUE){
          text(valOfx,h,labels =
expression("v"),col="black",xpd=add,adj=c(0.5,1))
          text(valOfx,h,labels = expression("I"), col="black",xpd=add)
        } else{
          text(h,valOfx,labels = expression(">"),col="black",cex =
1.2,xpd=add,adj=c(0,0.5))
          text(h,valOfx,labels =
expression("I"),srt=90,col="black",cex=1.2,xpd=add )
        }

      } else if(nOut == 4){
        if (horizontal==TRUE){
          text(valOfx,h,labels =
expression("v"),col="black",xpd=add,adj=c(0.5,0))
          text(valOfx,h,labels = expression("I"), col="black",xpd=add)
          text(valOfx,h,labels =
expression("v"),col="black",xpd=add,adj=c(0.5,1))
        } else {
          text(h,valOfx,labels=expression(">"),col="black",cex =
1.2,xpd=add,adj=c(0,0.5))
          text(h,valOfx,labels=expression(">"),col="black",cex =
1.2,xpd=add,adj=c(1,0.5))
          text(h,valOfx,labels =
expression("I"),srt=90,col="black",cex=1.2,xpd=add )
        }
```

```
      } else if(nOut == 5){
        if (horizontal==TRUE){
          text(valOfx,h,labels =
expression("v"),col="black",xpd=add,adj=c(0.5,1))
          text(valOfx,h,labels = expression("I"), col="black",xpd=add)
          text(valOfx,h,labels =
expression("v"),col="black",xpd=add,adj=c(0.5,0))
          text(valOfx,h,labels = expression("I"),col="black",xpd=add,adj =
c(0.5,0))
        } else {
          text(h,valOfx,labels=expression(">"),col="black",cex = 1.2,xpd=add)
          text(h,valOfx,labels =
expression("|"),srt=90,col="black",cex=1.2,xpd=add)
          text(h+(0.01*(n+1)),valOfx,labels = expression(">"),col="black",cex
= 1.2,xpd=add)
        }
      } else if(nOut > 5){
        display(5,valOfx,h)
        if(horizontal==TRUE){
          display(nOut-5,valOfx, h + (0.4*(1)))
        }else if (horizontal==FALSE){
          display(nOut-5,valOfx, h - (0.25*(1)))
        }

      }
    }
```

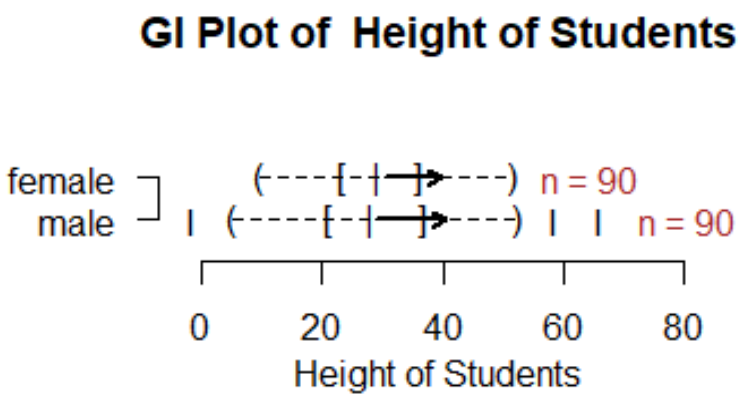

Figure 7: Final GIplots produced using the GIplot function.

4. Summary

The Gaussian Interval Plot (GIplot) is a pictorial representation of the five-number summary, the mean, the standard deviation, and the sample size of a quantitative variable. It also flags potential outliers (with their frequencies) that are *c* standard deviations away from the mean.

We show some examples of GIplot using the `GIplot()` function and its parameters. The following datasets representing the weight (in pounds) of 100 high school girls and 90 high school boys have been used to obtain the Figure 8. The `valueofc` used to obtain Figure 8 is 2.33. We leave Figure 8 as an exercise for the user to reproduce it.

```
weightGirls <-c(162.42,201.77,153.12,157.81,149.88,166.50,121.96,
175.55,165.65,177.18,167.77,117.02,134.29,139.46,96.68,145.02,167.25,
165.57,133.35,149.71,164.51,129.00,179.08,125.59,184.19,150.24,163.41,
153.46,154.36,183.40,138.33,145.94,150.42,143.44,126.46,158.30,185.65,
147.07,143.22,156.57,174.29,152.96,164.83,153.67,159.06,153.11,160.14,
130.34,119.88,155.29,175.47,138.67,146.01,118.61,168.87,152.89,153.98,
144.31,143.65,140.09,153.24,151.32,164.04,129.56,162.06,158.89,123.63,
133.68,146.87,158.82,163.47,148.36,154.74,128.69,159.62,141.26,175.27,
159.89,139.48,122.33,163.20,105.18,127.09,131.46,139.69,131.50,136.76,
164.98,149.16,161.88,190.61,176.62,163.39,141.26,155.99,162.39,175.49,
168.88,157.38,141.40)

weightBoys <- c(185.96,174.61,147.21,177.73,179.94,143.14,169.14,191.31,
160.17,203.10,189.01,169.90,160.79,129.44,161.73,199.11,180.57,134.98,
156.53,177.79,176.19,164.89,150.07,196.53,206.76,157.98,191.54,189.40,
164.07,171.92,168.45,198.57,158.37,156.06,190.81,143.58,151.95,197.82,
179.67,153.78,200.86,167.64,142.45,228.92,142.20,163.72,162.10,137.37,
201.13,187.45,204.83,188.09,156.09,147.28,151.88,167.90,189.76,188.79,
148.65,153.70,145.88,163.30,205.35,211.48,147.20,186.39,188.70,160.17,
173.15,180.19,129.90,138.67,144.79,172.80,188.70,181.15,173.08,180.71,
185.78,165.43,195.76,217.72,198.10,180.14,189.63,193.61,184.64,152.42,
153.86,144.86)
```

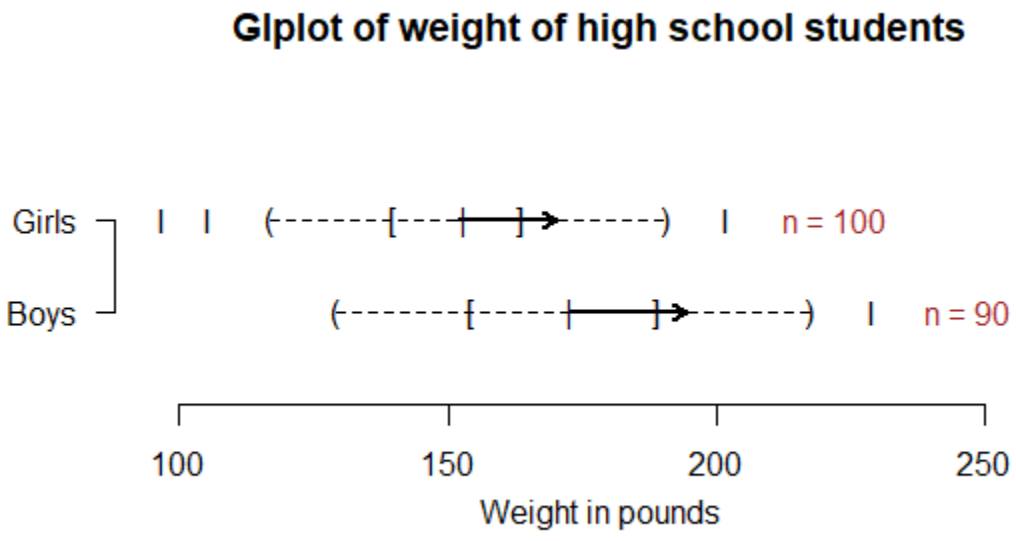

Figure 8: Horizontal GIplots of the weight of high school students.

The following data represents the body weight (pounds) of 33 girls on high school marching band and the volume (ml) of air exhaled by them.

```
weight<-c(
86, 112, 147, 132, 104, 106, 124, 113, 110, 114, 122,
168, 118, 130, 118, 104, 112, 129, 109, 127, 139, 127,
150, 113, 113, 106, 120, 110, 92, 114, 129, 110, 128)

volume<-c(
178, 185, 167, 172, 175, 174, 163, 172, 175, 172, 173,
157, 173, 170, 173, 191, 171, 166, 177, 172, 172, 170,
164, 174, 175, 177, 174, 174, 181, 176, 174, 178, 182)
```

Figure 9 shows how GIplots can be superimposed on the scatter plot. The `valueofc` used is 2.33. We leave it as an exercise for the user to reproduce it.

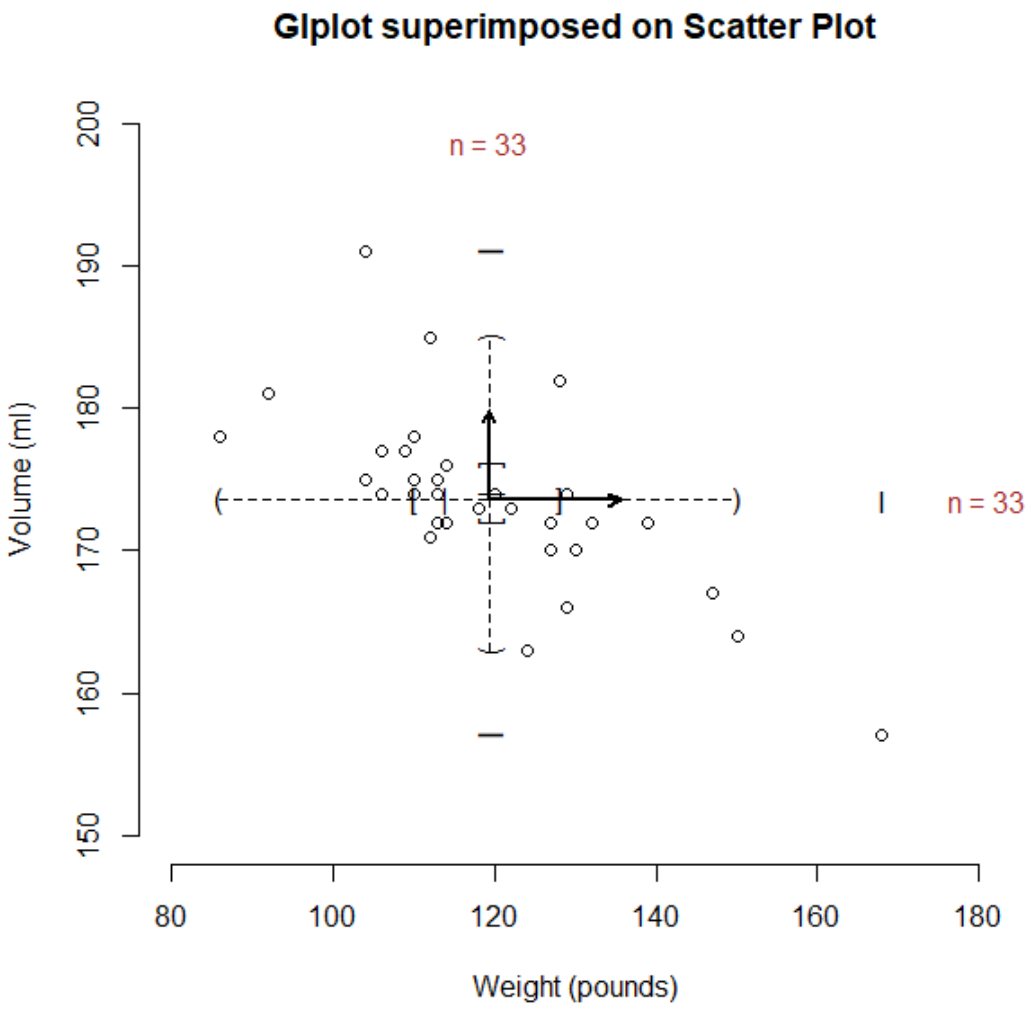

Figure 9: GIplots superimposed on the scatter plot for body weight (pounds) and volume (ml) of air exhaled by 33 girls on high school marching band.

References

Sarkar J. and Rashid M. (2021). IVY plots and Gaussian interval plots. *Teaching Statistics*. 43 (2), 85-90. https://doi.org/10.1111/test.12257

Spear, M. E. (1969). *Practical charting techniques.* New York: McGraw-Hill.

Spear, M. E. (1952). *Charting Statistics.* New York: McGraw-Hill.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison–Wesley, 1977.