STAT S400: Statistical Computing with R

Assignment #4 – An Introduction to Graphics

As for Assignment 3, a script file named `Assignment4.R` is posted on the course website along with all other relevant files for this assignment. Use this script to save time while going through the following illustrations. In fact, some of the code in the script file will serve as useful templates on which code for some of the assigned problems can be built.

Getting Data to Play With

We will continue to use R's *regular sequence* and *random sample generator* features to create samples on which to illustrate uses of the various new functions encountered here.

So, to start, create a regular sequence of numbers

```
> indepVar <- seq(from = 1, to = 10, length.out = 25)
```

then shuffle the numbers around to make things exciting. First set the random number generator seed so as to be able to duplicate output in the following illustrations

```
> set.seed(seed = 7)
```

and then create a new permutation of the data in `indepVar`

```
> indepVar <- sample(x = indepVar, size = 25, replace = FALSE)
```

Now, to get another set of numbers, first set the seed

```
> set.seed(seed = 17)
```

and then generate numbers that are from a *normal distribution*, and that correspond to the numbers in `indepVar`

```
> depVar <- 2 + 3*indepVar + rnorm(n = 25, mean = 0, sd = 2.75)
```

In statistics, the term *explanatory* (or *predictor*) *variable* is typically used for *independent variable* (or "x-values"), and the term *response variable* refers to the *dependent variable* (or "y-values").

The purpose of giving the two vectors such awkward (but suggestive) names will become evident in the illustrations that are to follow.

Relevant Resources for this Assignment

The R Book: Chapters 2, 4 and 5

An Introduction to R: Chapter 12

Your favorite elementary statistics text: Relevant topics/chapters.

Some Preliminary Ideas for Graphics Tasks

We will use one graphics function in R to look at some preliminary ideas with respect to using R for graphing data. These ideas extend to any type of chart or graph that is desired, sometimes easily and sometimes with a little creativity.

The Graphics Window

Begin by constructing an *xy*-scatterplot, with user-specified axes-labels, a main title, and a sub-title, using

```
> plot(x = indepVar, y = depVar,
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)",
+     sub = "(This is just for fun)")
```

Like most other graphics functions in R, when called, the function `plot` opens a new default-sized *graphics window* (figure not shown). It then constructs the plot most appropriate for the data provided (this is quite a versatile function). If a line-graph is desired, instead of a scatterplot, include the `type` argument to instruct R to do so. For example, (figure not shown, but what a mess!)

```
> plot(x = indepVar, y = depVar, type = "l",
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)",
+     sub = "(What a mess!)")
```

Notice that if a graphics window is already open, `plot` or any other high-level graphics function, when called, will plot the desired figure in the existing graphics window (again, figure not shown).

Now suppose that the circles plotted in the previous scatterplot are boring. One could make use of the `pch` argument as follows.

```
> plot(x = indepVar, y = depVar, pch = 11,
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)",
+     sub = "(This really is a lot of fun!)")
```

Try using integers from 0, 1, 2, . . . , 25 – replace 11 by the number you want to try and re-run the above code.

Identifying Case Numbers of Plotted Points

Go back to the basic scatter plot, leave out the corny sub-title this time around. That is, run (figure not shown)

```
> plot(x = indepVar, y = depVar,
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)")
```

Now suppose you are interested in the case numbers of some of the plotted points (that is, where do the corresponding *xy*-pairs lie in the data). Here is what you do.

With the understanding that `indepVar` is plotted horizontally (the x-values) and `depVar` is plotted vertically (the y-values), run

```
> identify(x = indepVar, y = depVar)
```

then move the cursor onto the graphics window. The cursor will take on the appearance of cross-hairs ("+"). Now, as shown in Figure 1, move the cross-hairs to a point of interest and left-click your mouse. The case number (15 in this case) should appear right next to the point, that means this point corresponds to the fifteenth data entry in the xy-pairs. The location, relative to the point of interest, at which the case number appears (15 in this case) depends on where the cross-hairs are when you left-click the mouse. Explore this with some other points.

When you are done, right-click your mouse, then "**Stop**" the session, see Figure 2. Close the graphics device window before continuing, right click on the little "☒" in the top-right corner.

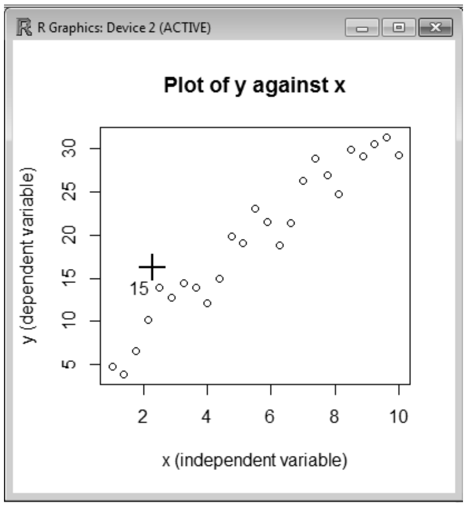

Figure 1: Using the `identify` function to label plotted points with their corresponding case numbers.

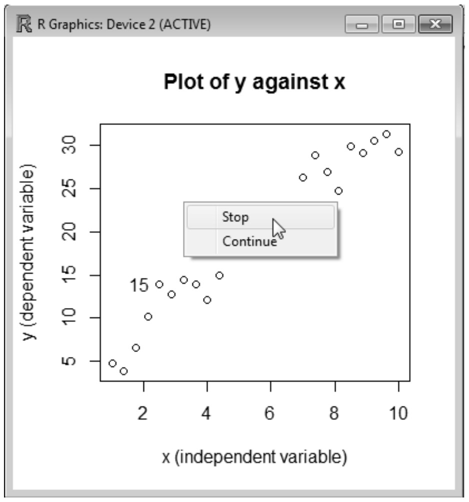

Figure 2: Stopping an `identify` session. The inserted label(s) are saved to the graphics image.

Customizing the Plotting Window's Size

When graphics functions, such as `plot`, are called the resulting window has a default size of 7x7 inches (even if it does not appear to be so). In the Windows environment the function `win.graph` provides a basic means for simply resizing a plotting window – see the documentation pages for `win.graph` for more functions/options in this area. MacIntosh users will have to use the function `quartz`.

For the illustration in question, first open a window of width 3.5 inches and height 4 inches,

```
> win.graph(width = 3.5, height = 4)
```

Now construct the scatterplot as has been done in earlier illustrations (figure not shown) by running

```
> plot(x = indepVar, y = depVar,
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)")
```

You should see a resized window. However, the included labels and axes scales might appear a little large in relation to the figure size.

Customizing Graphics Parameter Settings

Close the graphics device window, and then open a new window with the same size,

```
> win.graph(width = 3.5, height = 4)
```

then, through the function `par`, set the pointsize to 10 points, and scale all plotted images by a factor of 0.75,

```
> par(ps = 10, cex = 0.75)
```

then reconstruct the xy-scatter plot,

```
> plot(x = indepVar, y = depVar,
+     main = "Plot of y against x",
+     xlab = "x (independent variable)",
+     ylab = "y (dependent variable)")
```

and, finally, identify some points of interest

```
> identify(x = indepVar, y = depVar)
[1] 1 3 12 24
```

The output line

```
[1] 1 3 12 24
```

(or whatever occurs when you run the code) lists the case numbers identified. One might, at this point decide to insert a sub-title for the resulting figure. Do this using, for example,

```
> title(sub = "(Wooooohoooooo!)")
```

Figure 3 shows the end result.

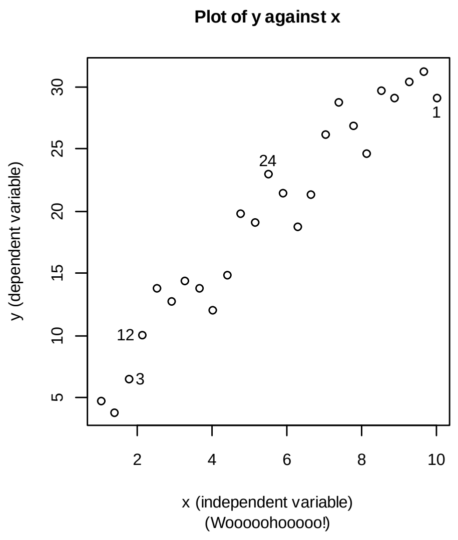

Figure 3: The end result of playing around with the earlier xy-data.

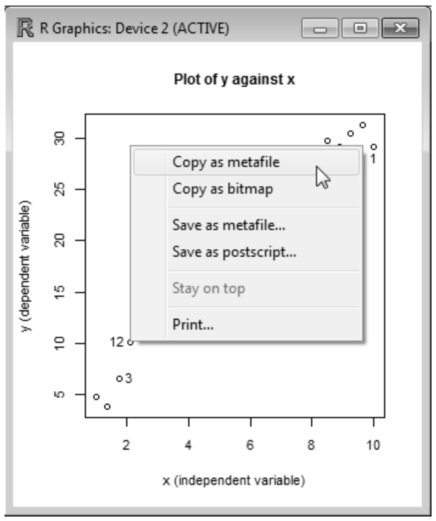

Figure 4: Copying a graphics image as a metafile to paste in (for example) an MS Word document. Notice other options available.

The function `par` provides for an enormous variety of options in setting graphics parameters. Browse the documentation pages for `par` to familiarize yourself with what is available. You can also find an alphabetical listing of graphics parameters and their purposes in *The R Book* (see pp. 935-957).

Exporting Graphics Images from R

Figure 3 was exported in a Windows environment as follows. First right-click your mouse on the graphics window, then left-click on "**Copy as metafile**," see Figure 4. Now go to your MS Word (or other) document and and paste the image using the appropriate method.

When using the copy-paste approach, I have found this to produce the best results – as compared to copying the image as a bitmap object. MacIntosh users, the steps might differ for you. See the help resources that accompany the "R for MacIntosh" package you downloaded. Also, remember that there is always "Google."

As suggested in *The R Book* (p. 242), and the options apparent from the above process, there are alternatives to the method just described. Graphics images can be saved (exported) to a variety of file types commonly used for graphics images. The following table lists some file types, and the corresponding R function to use. Details on each function can be found in the relevant documentation pages (remember `help` and `?`).

To create file of type	**Use the function**
Bitmap (`*.bmp`)	`bmp`
JPEG (`*.jpeg`)	`jpeg`
TIFF (`*.tiff`)	`tiff`
PNG (`*.png`)	`png`
PDF (`*.pdf`)	`pdf`
Postscript (`*.ps`)	`postscript`

Once an image has been saved to a file (of the desired type), it can be imported/inserted into a document (an MS Word document for example) using the appropriate insertion steps.

Before continuing, clear the workspace of all objects using the `rm` function, and then also clear the console using `Ctrl+L`.

Statistical Charts and Graphs

Recall the story of Bubba from the previous assignment. The data from his previously described study, plus an additional variable `age`, are stored in the file `Assignment4.RData`.

Load the data in this file in the usual manner, and then explore the variable classes (vector or factor?) and contents using methods described in earlier assignments before continuing. In addition to being used for the following illustrations, these data are to be used for the problems assigned for submission.

As a reminder, Bubba's data are organized under the following variables.

score: Score earned on the final – possible scores 0-100%

course: Which course the student was in – possible entries include `proj` (project-based) or `trad` (traditional)

gender: Gender of the student – possible entries include `male` or `female`.

field: Field of the student – possible entries include `natsci` (Natural Sciences), `socsci` (Social Sciences), `mathsci` (Mathematical Sciences), `other` (any other field).

age: Age of the student – the sample space includes ages (rounded up to the nearest whole number).

Here are illustrations of some of the more commonly used statistical charts and graphs.

Bar Charts

A default bar chart of the (overall) distribution of fields can be obtained as follows, see Figure 5. First Set a window size and graphics parameters

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
```

then get frequencies of fields (overall)

```
> nobs <- with(data = bubbaStudy, expr = table(field))
```

and finally onstruct a bar chart of fields (overall)

```
> barplot(height = nobs)
```

We'll use this chart to illustrate some more basic graphics customizing tools.

First, notice that the vertical axis stops a bit short. We may want it to go a bit higher. Second, there is no horizontal line identifying the horizontal axis. Let's put one in. Third, let's turn this into a *Pareto Chart*; that is, arrange the bars in descending heights. Finally, stick in some labels and titles. The result appears in Figure 6, and here is the code. First prepare the plotting window.

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
```

Next, get frequencies in descending order

```
> nobs <- sort(x = nobs, decreasing = TRUE)
```

Now construct a bar chart (see next page for the code).

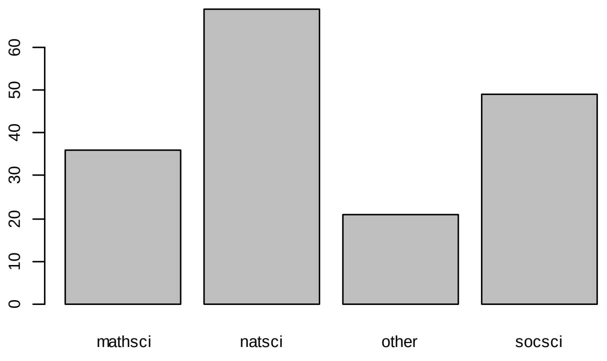

Figure 5: A default bar chart using the function `barplot`.

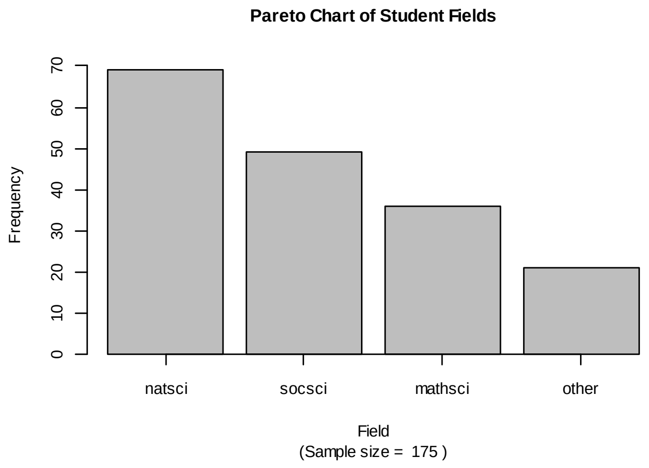

Figure 6: A jazzed up Pareto Chart of student fields

```
> barplot(height = nobs,
+     # Set range for vertical axis, up 5% of maximum
+     ylim = c(0, 1.05*max(nobs)),
+     # Insert horizontal axis and tick marks
+     axis.lty = 1,
+     # Insert label for horizontal axis
+     xlab = "Field",
+     # Insert label for vertical axis
+     ylab = "Frequency",
+     # Insert main title
+     main = "Pareto Chart of Student Fields",
+     # And, maybe, stick in a subtitle indicating the sample size
+     sub = paste("(Sample size = ", sum(nobs),")"))
```

The arguments `ylim` (and `xlim`), `xlab`, `ylab`, `main`, and `sub` show up in every graphics function for which they are appropriate. Arguments such as `axis.lty`, and others, may be function-specific, and may also be part of the `par` function's list of arguments. Many parameter settings can be made using the `par` function prior to plotting, and sometimes within a plotting function call.

Now run the following code and see Figure 7 for the results. First get the window ready

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
```

Then, get a two dimensional contingency table

```
> nobs <- with(data = bubbaStudy, expr = table(field, gender))
```

Now construct a bar chart

```
> barplot(height = nobs,
+     # Set range for vertical axis, up 20% of maximum
+     ylim = c(0, 1.20*max(nobs)),
+     # Insert horizontal axis and tick marks
+     axis.lty = 1,
+     # Place bars beside each other
+     beside = TRUE,
+     # Set spaces between bars
+     space = c(0, 0.5),
+     # Insert a legend
+     legend.text = rownames(nobs),
+     # Provide some legend argument settings
+     args.legend = c(bty = "n", x = "topright"),
+     # Insert label for horizontal axis
+     xlab = "Field",
+     # Insert label for vertical axis
+     ylab = "Frequency",
+     # Insert main title
+     main = "Bar Chart of Student Fields by Gender",
+     # And, maybe, stick in a subtitle indicating the sample size
+     sub = paste("(Overall sample size = ", sum(nobs),")"))
```

For finishing touches, insert sizes of male and female samples

```
> sizes <- colSums(nobs)
> mtext(text = paste("(",sizes,")", sep = ""),
+     at = c(2.5, 7), side = 1, line = 2, cex = 0.75)
```

Figure 8 is obtained as follows. First get a three dimensional contingency table of `field` by `gender` and `course`.

```
> nobs <- with(data = bubbaStudy, expr = table(field, gender, course))
```

Enter `nobs` in the console to see what it looks like.

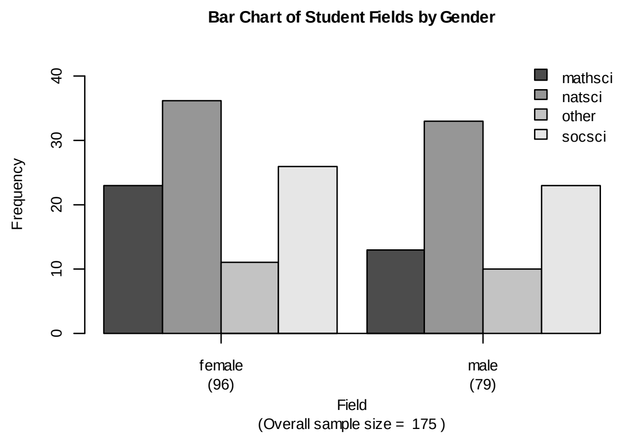

Figure 7: Bar charts for fields by gender.

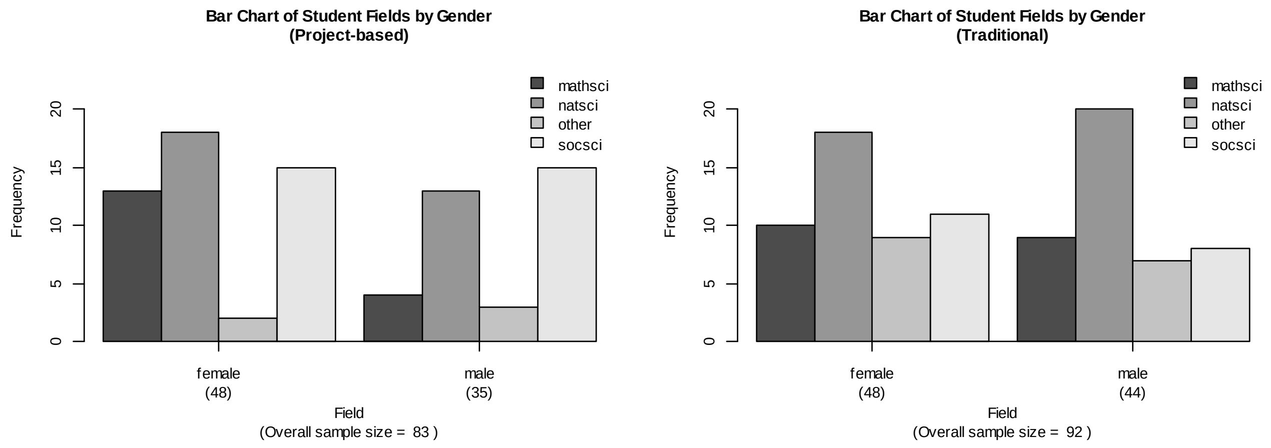

Figure 8: Bar chart of student fields by gender and course type.

Next, get the window ready.

```
> win.graph(width = 10, height = 3.5)
> # Use mfrow to split window into 1 row with 2 columns
> par(mfrow = c(1, 2), ps = 10, cex = 0.75)
```

Now construct bar charts using a *for-loop*. First get dimension 3 names

```
> courses <- dimnames(nobs)[[3]]
```

then run the loop

```
> for (i in courses)
+     # Begin for-loop body
+     {
+     barplot(height = nobs[, , i],
+         # Set range for vertical axis, up 20% of maximum
+         ylim = c(0, 1.20*max(nobs)),
+         # Insert horizontal axis and tick marks
+         axis.lty = 1,
+         # Place bars beside each other
+         beside = TRUE,
+         # Set spaces between bars
+         space = c(0, 0.5),
+         # Insert a legend
+         legend.text = rownames(nobs[ , , i]),
+         # Provide some legend argument settings
+         args.legend = c(bty = "n", x = "topright"),
+         # Insert label for horizontal axis
+         xlab = "Field",
+         # Insert label for vertical axis
+         ylab = "Frequency",
+         # Insert main title
+         main = paste("Bar Chart of Student Fields by Gender\n",
+             "(",ifelse(i == "trad", "Traditional", "Project-based"),
+             ")", sep = ""),
+         # And, maybe, stick in a subtitle indicating the sample size
+         sub = paste("(Overall sample size = ", sum(nobs[ , , i]),")"))
+     # Insert sizes of male and female samples
+     sizes <- colSums(nobs[ , , i])
+     mtext(text = paste("(",sizes,")", sep = ""),
+         at = c(2.5, 7), side = 1, line = 2, cex = 0.75)
+     # End loop body
+     }
```

The techniques used to obtain Figures 7 and 8 can be extended to other charts and graphs. *Sometimes*, the `apply` function can be used in place of a for-loop to construct multiple charts or graphs such as for Figure 8 – see script file for an example. However, I have found the for-loop to be easier (more intuitive), particularly when creating customized figures.

Pie Charts

The *Pie chart* equivalent for Figure 8, see Figure 9, can be obtained using code of the following form. First, load a new (user-defined) function.

```
> get.percents <- function(x){round(x/sum(x)*100, 1)}
```

Then, use the `apply` function on `get.percents` to create a percentage equivalent of the 3-dimensional array `nobs`. Call the new array `pobs`. Look at `pobs` to make sure the percentages are correct (good practice when doing a first run of code).

```
> pobs <- apply(X = nobs, MARGIN = c(2, 3), FUN = get.percents)
```

Next, Create pie chart slice labels containing percentages and be sure to take a look at contents.

```
> pobLabels <- array(data = paste(
+     dimnames(pobs)[[1]],":  ", pobs,"%", sep = ""),
+     dim = c(4, 2, 2), dimnames = dimnames(pobs))
```

Then, get character vectors to build chart main titles

```
> courses <- c(proj = "Project-based", trad = "Traditional")
> genders <- c(female = "Females", male = "Males")
```

Now get the window ready.

```
> win.graph(width = 8, height = 8)
> # Use mfrow to split window into 2 rows and 2 columns
> par(mfrow = c(2, 2), ps = 10, cex = 0.75)
```

And, finally, use *nested for-loops* to produce Figure 9.

```
> for (i in dimnames(pobs)[[3]]){
+     for(j in dimnames(pobs)[[2]]){
+         pie(x = pobs[ , j , i], labels = pobLabels[ , j, i],
+             col = gray(seq(from = 0.5, to = 1, length.out = 5)),
+             main = paste("Pie Chart of Field for", genders[j], "\n in\n",
+                 courses[i], "Course"))}}
```

Before continuing, use the `rm` function to clear all objects from the workspace, except the data frame `bubbaStudy`. Also clear the console.

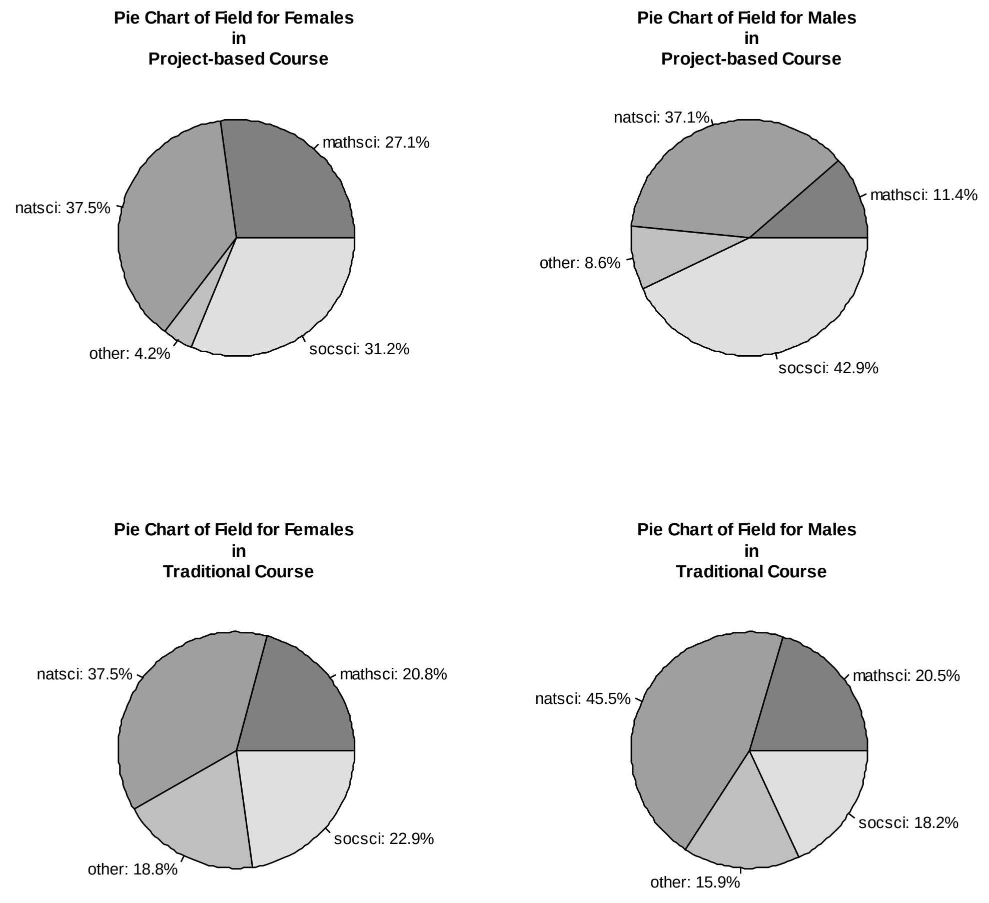

Figure 9: Pie chart equivalents of Figure 8.

Boxplots

A basic horizontal *boxplot* of participant `age`, in `bubbaStudy`, can be obtained using

```
> with(data = bubbaStudy, expr = boxplot(x = age, horizontal = TRUE,
+     xlab = "Age"))
```

However, as with all previous charts, refinements can be made. For example, Figure 10 was produced as follows.

First find the range of values for `age`, turns out to be $15 \leq$ `age` ≤ 59. Next, open a 5×2 window and construct a horizontal boxplot with the axes suppressed, and with the plot range set for $15 \leq$ `age` ≤ 60.

```
> with(data = bubbaStudy,
+     expr = boxplot(x = age, horizontal = TRUE, axes = FALSE,
+         ylim = c(15, 60), main = "Boxplot of Age of Participants"))
```

Now insert an axis with custom scaling, starting at 15, with tick-marks every 5 units, up through 60. Then insert an axis label.

```
> axis(side = 1, at = seq(from = 15, to = 60, by = 5))
> title(xlab = "Age")
```

The `boxplot` function has one nice feature. For example, Figure 11 was produced using

```
> win.graph(width = 5, height = 2.5)
> par(ps = 10, cex = 0.75)
> boxplot(formula = age ~ gender, data = bubbaStudy,
+     horizontal = TRUE, axes = FALSE,
+     ylim = c(15, 60), main = "Ages by Gender")
> axis(side = 1, at = seq(from = 15, to = 60, by = 5))
> axis(side = 2, at = c(1, 2), labels = levels(bubbaStudy$gender),
+     lty = 0)
> title(xlab = "Age")
```

Notice the *formula* manner in which the data variables to be plotted, `age` *by* `gender`, are identified in the function call.

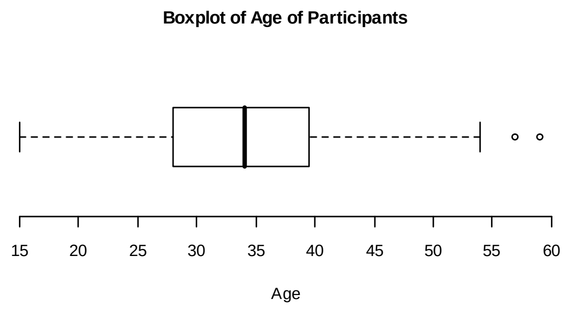

Figure 10: Boxplot of all participants' ages with custom scaling on the axis.

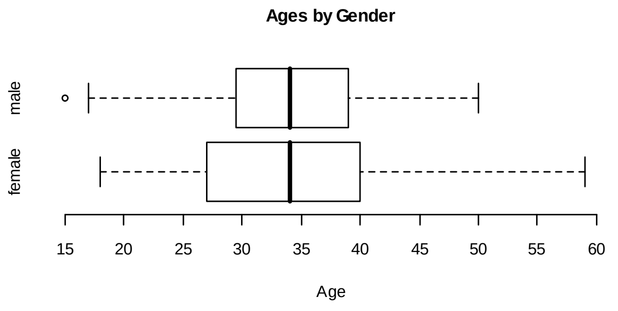

Figure 11: Boxplots of `age` by `gender`.

Stripcharts

A *stripchart* is another way to look at the *spread* of data. Unlike a boxplot, a stripchart can be made to display all points in the samples whose relative spreads are of interest. For example, the stripchart equivalent of Figure 11, shown in Figure 12, is obtained using

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
> with(data = bubbaStudy,
+     expr = stripchart(x = age ~ gender, method = "jitter",
+         vertical = TRUE, ylab = "Age", xlab = "Gender", pch = 1,
+         axes = TRUE, frame.plot = FALSE))
```

Improvements in appearance may be made in much the same manner as was done with previous figures. For example, Figure 13 was produced as follows. First prepare the plotting window

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
```

Then,

```
> with(data = bubbaStudy,
+     # Construct a vertical, jittered stripchart of age by gender
+     expr = stripchart(x = age ~ gender, method = "jitter", vertical = TRUE,
+         # Label the y- and x- axes, and use circles for the plotting symbol
+         ylab = "Age", xlab = "Gender", pch = 1,
+         # Set the x- and y-limits to spread the axes out a bit
+         ylim = c(15, 60), xlim = c(0.5, 2.5),
+         # Suppress the axes, and the plot frame
+         axes = FALSE, frame.plot = FALSE,
+         # Put in a main title
+         main = "Stripchart of Age by Gender"))
```

Next, insert a horizontal axis with tickmark labels for gender

```
> axis(side = 1, at = c(0.5, 1:2, 2.5),
+     labels = c("", levels(bubbaStudy$gender),""))
```

and, finally, insert a customized vertical axis with tick-mark labels oriented horizontally.

```
> axis(side = 2, at = seq(from = 15, to = 65, by = 5), las = 1)
```

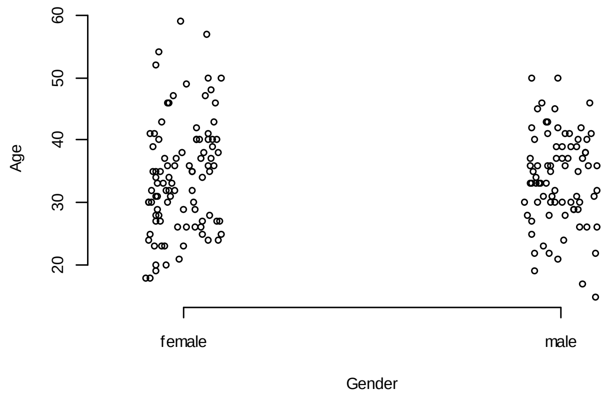

Figure 12: More or less basic vertical strip chart.

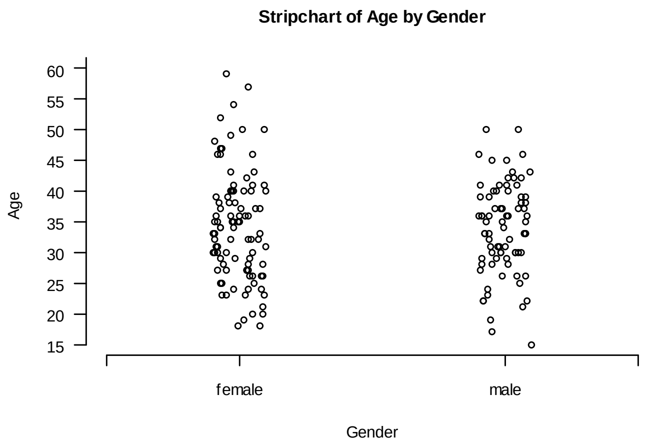

Figure 13: A fine-tuned jittered stripchart of `age` by `gender`.

Histograms

A *histogram* is yet another way to explore numerical data to get a feel for not only the spread of the data, but also the *shape* of the underlying distribution. The R function to construct a histogram is called `hist`, and, as with all previous plotting functions, there is the most basic `hist` function call.

For example, the default *density histogram* for `age` in `bubbaStudy` is (figure not shown)

```
> with(data = bubbaStudy,
+      expr = hist(x = age, freq = FALSE))
```

Sticking to the theme of producing nice looking figures, Figure 14 was produced using the following code. First prepare a window

```
> win.graph(width = 5, height = 3.5)
> par(ps = 10, cex = 0.75)
```

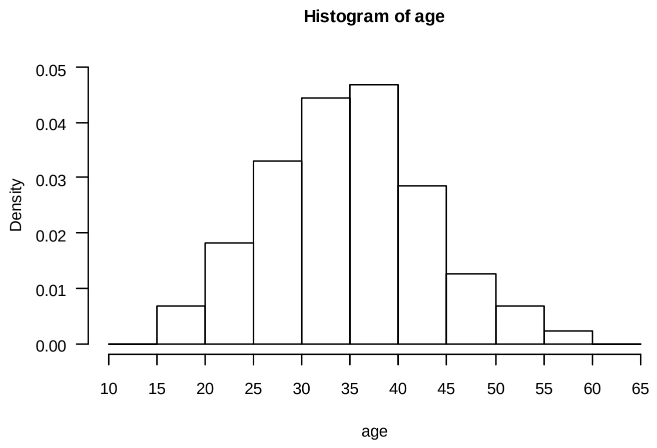

Figure 14: Density histogram of `age` using manually set class boundaries.

Then construct a density histogram without axes

```
> with(data = bubbaStudy,
+     # Construct a right-open, density histogram with axes suppressed
+     expr = hist(x = age, freq = FALSE, axes = FALSE, right = FALSE,
+         # Set axes limits (after checking the data)
+         ylim = c(0, 0.05), xlim = c(10,65),
+         # Identify user-defined class limits
+         breaks = seq(from = 10, to = 65, by = 5)))
```

Now put in customized horizontal and vertical axes

```
> axis(side = 1, at = seq(from = 10, to = 65, by = 5))
> axis(side = 2, at = seq(from = 0, to = 0.05, by = 0.01), las = 1)
```

If so desired, a normal distrubution density curve can be superimposed on a histogram as well, see Figure 15. To do this, first run the code to reproduce Figure 14, then create a large set of x-values over the range of `age` by running

```
> x <- seq(from = 10, to = 65, length.out = 101)
```

Then superimpose the density curve with the help of the `curve` function.

```
> with(data = bubbaStudy,
+     # Plot a density curve having mean and sd same as for dataset age
+     expr = curve(expr = dnorm(x, mean = mean(age), sd = sd(age)),
+         # Add this curve to the existing histogram, using a dashed line
+         add = TRUE, xlab = NULL, lty = 3))
```

Notice, in creating the histogram, as well as `x`, the range of x-values used is larger than the actual data range. This enables plotting the curve beyond the outer bars.

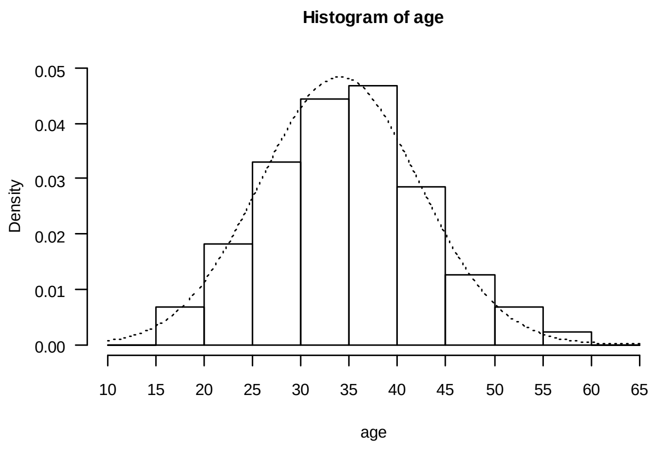

Figure 15: Density histogram of `age` with probability distribution curve superimposed.

There are a couple of things to keep in mind when using the `hist` function. Open the documentation page for `hist` while reading the following discussion.

First, the argument `breaks` is used to set the *class boundaries* for the *grouped frequency distribution* used to construct the histogram, and the right-hand rule is the default *inclusion rule*, determined by the argument `right = TRUE`. For example, suppose the partition set (or `breaks`) is the vector $(x_0, x_1, x_2, x_3, \ldots, x_{n-1}, x_n)$. Then the intervals within which the data are classified are defined by

$$[x_0, x_1]\,,\ (x_1, x_2]\,,\ (x_2, x_3]\,,\ \ldots,\ (x_{n-1}, x_n]\,.$$

If `right = FALSE` is used, then the intervals within which the data are classified are

$$[x_0, x_1)\,,\ [x_1, x_2)\,,\ [x_2, x_3)\,,\ \ldots,\ [x_{n-1}, x_n]\,.$$

Second, the default method used to determine the appropriate class boundaries is `breaks = "Sturges"`. This typically works quite well for most purposes, particularly when quick plots are desired for an initial examination of the data. The user-specified `breaks`, described in the previous illustrations, is useful when you wish to establish some consistency for more than one histogram of a particular variable, such as `age`, across levels of a factor, such as `gender`.

xy-Scatterplots

As described earlier, the default `plot` function performs this task – by the way, the (non-default) `plot` function does quite a bit more. For example, try running the following (figures not shown).

```
> with(data = bubbaStudy, expr = plot(age ~field))
```

Next, try

```
> x <- rnorm(n = 25, mean = 5, sd = 7)
> y <- 2*x + runif(n = 25, min = -10, max = 10)
> z <- 3*y - 1.5*x + rnorm(n = 25, mean = 0, sd = 5.3)
> plot(data.frame(x, y, z))
```

The last figure (a *matrix scatterplot*) can be duplicated using

```
> pairs(cbind(x, y, z))
```

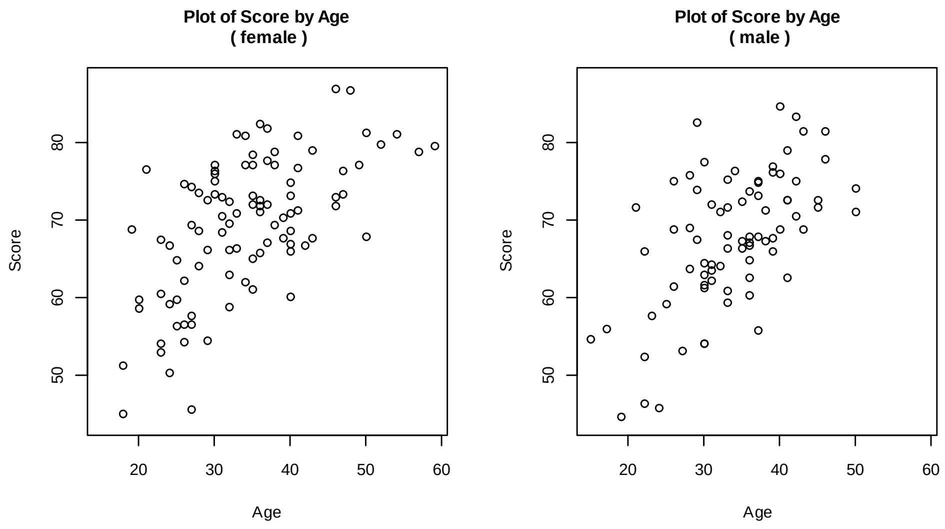

Figure 16: Scatterplots of `score` by `age`, one for each level of `gender`.

Go to the documentation page of `plot.default` to find out quite a bit on preparing, and customizing, *xy*-scatter plots when using the function `plot`.

You can prepare code to produce scatterplots of a particular variable, by a factor such as in Figure 16 (the code for this figure is not given, this will be one of your assigned tasks).

QQ-Plots

QQ normal probability plots provide a means for an informal way to determine if the underlying random variable for a particular sample might have a normal distribution. For example, a default QQ plot along with a QQ-line for reference purposes for the scores of female students in the mathematical sciences is shown in Figure 17. The QQ-line passes through the points that identify the first and third quartiles of the sample.

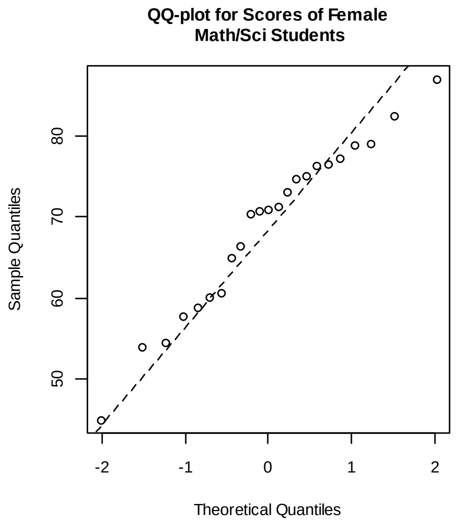

Figure 17: QQ-plot with a QQ-line for reference.

Figure 17 was obtained as follows. First set the plotting window properties

```
> win.graph(width = 3.5, height = 4)
> par(ps = 10, cex = 0.75)
```

Then extract the relevant data

```
> mathGirls <- with(data = bubbaStudy,
+     expr = score[(gender == "female") & (field == "mathsci")])
```

and, finally, construct the QQ-Plot with a specified main title.

```
> qqnorm(y = mathGirls,
+     main = "QQ-plot for Scores of Female\n Math/Sci Students")
```

For references purposes, a dashed qqline can be inserted as follows.

```
> qqline(y = mathGirls, lty = 2)
```

QQ-plots plots of standardized data can be used for the same purpose. Here is how you might do this. First get the standardized sample (z-scores), then use the `qqnorm` function as described earlier.

```
> mgs <- (mathGirls - mean(mathGirls))/sd(mathGirls)
> qqnorm(y = mgs,
+     main = "QQ-plot for Scores of Female\n Math/Sci Students",
+     sub = "(Standardized Scores)")
```

Here the line $y = x$ serves as an appropriate reference line

```
> abline(a = 0, b = 1, lty = 2)
```

Note that while the QQ-line can also be used as a reference line, many use the line $y = x$ for QQ-plots of standardized samples.

Exploratory Exercises

Find out more about the functions `par` (with special attention to the arguments `mfrow` and `mfcol`), `layout`, and `split.screen`. Try running

```
> example(layout)
```

and

```
> example(split.screen)
```

to get a feel for what these can be used for. See, also, pp. 242-946 of *The R Book*. You will find all of these useful for customizing your graphics images some time or the other.

To be Submitted

The data frame `bubbaStudy` should be used for all of the tasks listed below. Make sure you prepare *well-documented*, and *clearly organized* code to perform the indicated tasks *completely* and *correctly*. Look to the code in the script file for ideas and/or templates.

1. Write code to duplicate (as near as possible) Figure 16 – you may size your plotting window as you see fit, but it should look nice (figure and font sizes should be neither too large nor too small).

   Hints: There are *at least* three approaches: brute-force, and two others (one of which involves using a `for` loop, and the other with the help of the `by` function. The `apply` and/or `lapply` functions might also find use with some creativity). Use the approach you find most intuitive.

2. Write code to duplicate (as near as possible) Figure 18. Notice that this contains Pareto Chart equivalents for Figure 8. Again, there are brute-force methods, and more elegant approaches. Use the approach that is most intuitive to you.

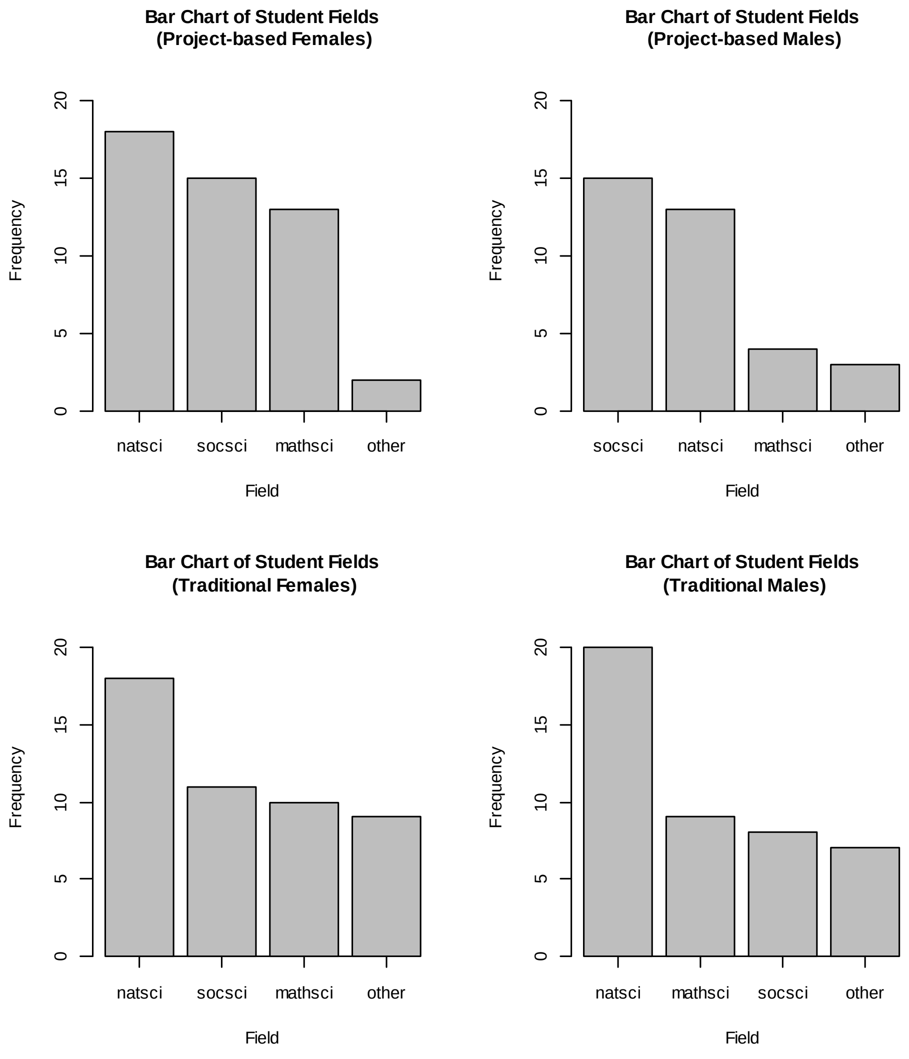

Figure 18: Pareto charts to get for Problem 2.

3. Make use of the `split.screen` *or* the `layout` function to produce a figure *equivalent to* Figure 19. You may use the default `breaks` setting for the histogram.

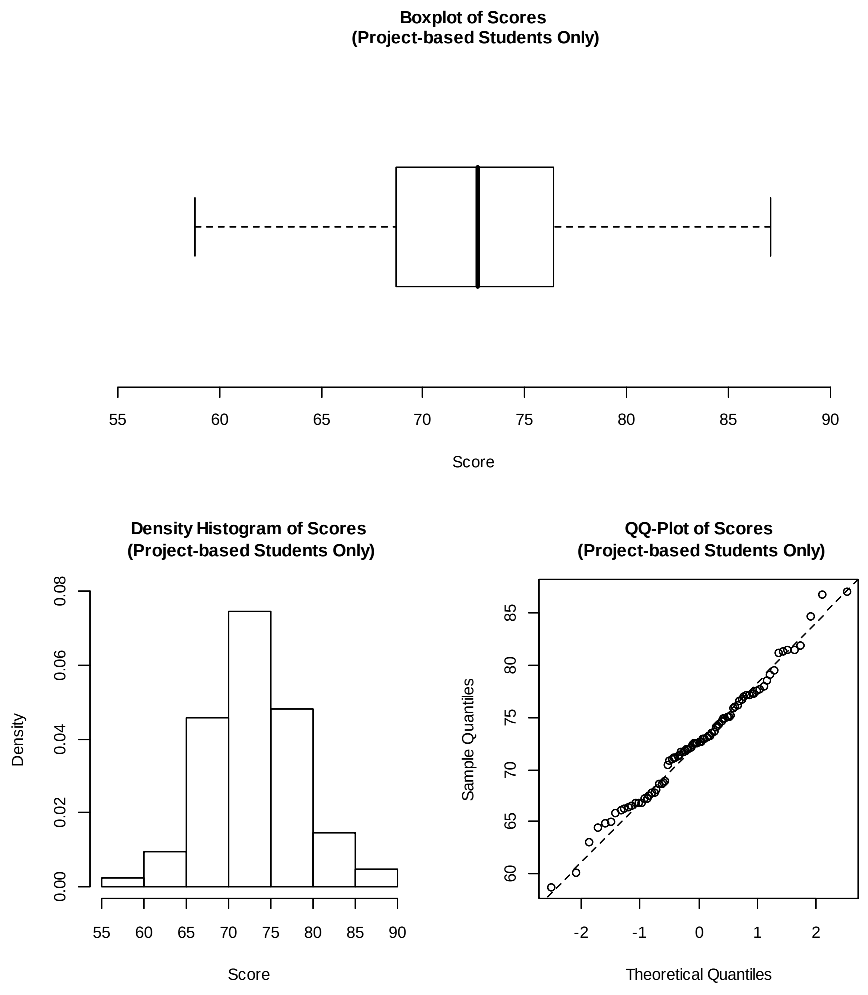

Figure 19: Plots to be obtained for Problem 3. Use scores of project-based students only.