Plotting data - Answers

Crispin Jordan

18/10/2020

Question 1 - Plants plot

Part (a)

We import the data like this:

```r
plants <- read.table("BMS2_plants.csv", header = TRUE, sep = ',')
```

Part (b)

We can identify the names of the columns in several ways. Here's the first, using the function `head()`, which displays the top few lines of a dataframe. (Can you guess what the function `tail()` does? Try it!)

```r
head(plants)
```

```
##   TreatPool Line Days2Flower Height LengthInmm WidthInmm
## 1   Control    3          61     70      59.05     32.28
## 2   Control    9          53     65      76.48     47.35
## 3   Control   13          61     52      62.31     37.84
## 4   Control   17          71     62      55.02     33.54
## 5   Control   27          50     65      62.28     34.14
## 6   Control   37          47     34      52.07     37.36
```

We can see that there are six columns, with the headings, `TreatPool`, `Line`, `Days2Flower`, `Height`, `LengthInmm`, `WidthInmm`.

Alternatively, we could use the function, `summary()` to obtain a summary of each column in the data.frame, including the name of each column:

```r
summary(plants)
```

```
##      TreatPool        Line        Days2Flower       Height     
##  Control  :180   Min.   : 1.00   Min.   :44.00   Min.   : 5.00  
##  Herbivory: 89   1st Qu.: 9.00   1st Qu.:50.00   1st Qu.:46.00  
##  Water    : 89   Median :19.00   Median :55.00   Median :58.00  
##                  Mean   :18.67   Mean   :55.49   Mean   :57.04  
##                  3rd Qu.:29.00   3rd Qu.:61.00   3rd Qu.:68.00  
##                  Max.   :37.00   Max.   :76.00   Max.   :90.00  
##    LengthInmm       WidthInmm    
##  Min.   :  3.48   Min.   : 3.06  
##  1st Qu.: 37.55   1st Qu.:22.32  
##  Median : 50.68   Median :30.72  
##  Mean   : 54.69   Mean   :32.09  
##  3rd Qu.: 74.51   3rd Qu.:41.47  
##  Max.   :103.32   Max.   :60.13  
```

Notice the same column headings. As a third options, we could use the function `str()`:

```r
str(plants)
```

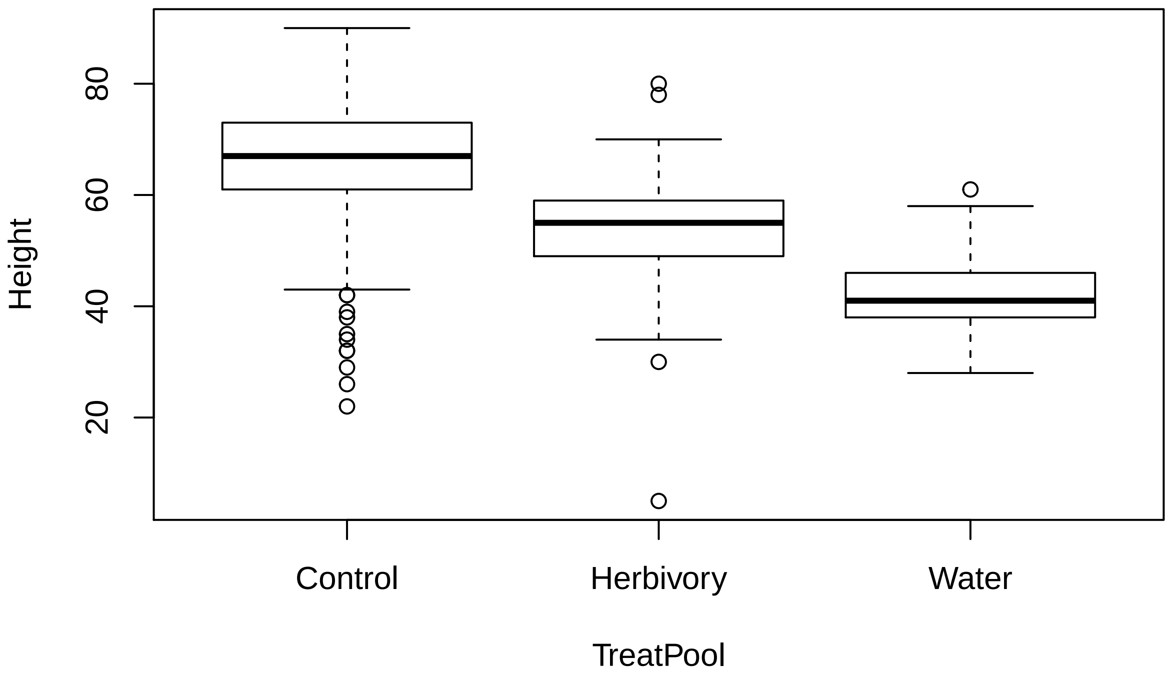
```
## 'data.frame':    358 obs. of  6 variables:
##  $ TreatPool  : Factor w/ 3 levels "Control","Herbivory",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Line       : int  3 9 13 17 27 37 1 3 5 7 ...
##  $ Days2Flower: int  61 53 61 71 50 47 61 50 55 61 ...
##  $ Height     : int  70 65 52 62 65 34 65 64 66 64 ...
##  $ LengthInmm : num  59 76.5 62.3 55 62.3 ...
##  $ WidthInmm  : num  32.3 47.4 37.8 33.5 34.1 ...
```

As a fourth option, we could just enter the name of the data.frame, `plants`, which would cause R to display the entire data.frame (or a lot of it, anyway, if the data.frame was really large), including the column headings. We'll not do that here because `plants` has a lot of output and we want to conserve space (I realize that this is what the question asked you to do! Ironic?). But you should try it! You can then scorl to the top of the data.frame to find the column names.

Part (c)

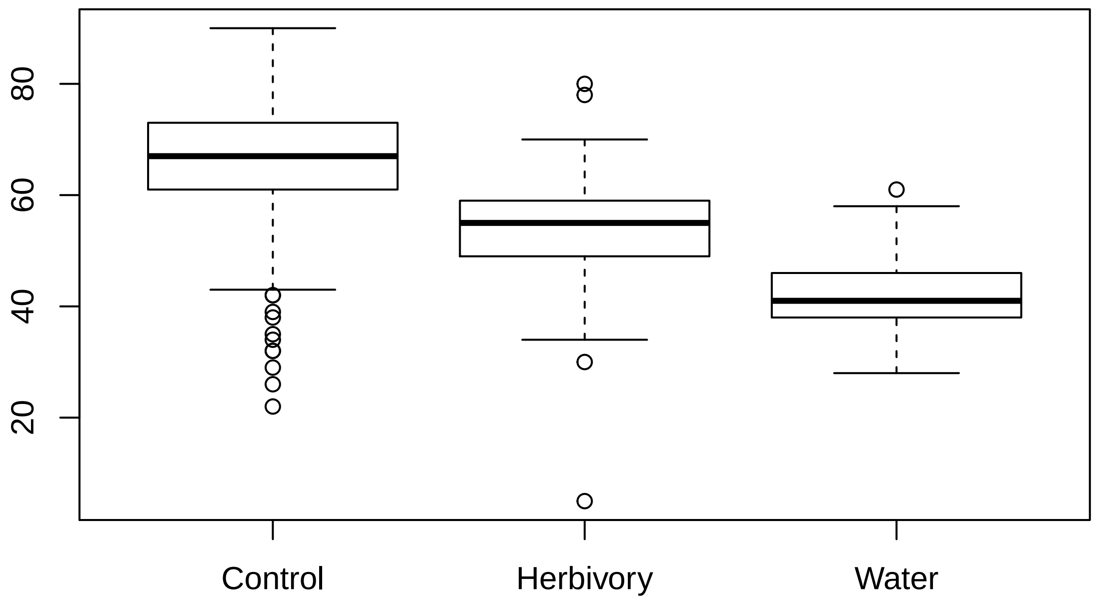
Now let's create a boxplot. We do this using the `boxplot()` function. Within the `boxplot()` function we need to designate the columns in our data.frame that contain the data for the y-axis (i.e., the *dependent* variable) and the x-axis (the *independent* variable). From the question (c), we can infer that the two columns to focus upon are `TreatPool` and `Height`; the variable `TreatPool` indicated the three treatments that were applied to plants. Our hypothesis (which was so carelessly omitted from the question) is that the treatment that a plant experiences (`TreatPool`) will affect its `Height`. Based on this hypothesis, we infer that we expect `Height` to depend on `TreatPool`; therefore `Height` is the *dependent* variable, and will therefore be placed on the left of the '`~`' within the `boxplot()` function. We also need to tell the function where to find these data, by including `data = plants`, where `plants` is the name of our data.frame. With all of this in mind, we make a boxplot like this:

```r
boxplot(Height ~ TreatPool, data = plants)
```



Notice that **R** automatically used the column names as axis lables. That's a bit ugly, so we'll get rid of them by providing 'empty' labels for the x- and y-axes, like this:

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
```



Much better! Don't worry: **Part (i)** tells us to add axis labels.

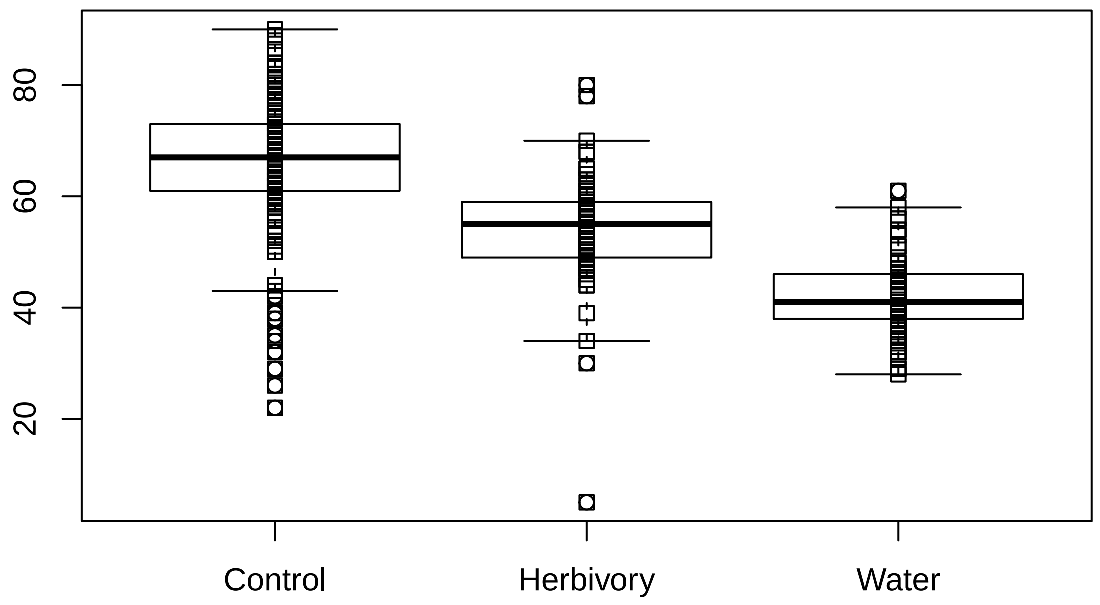
Part (d)

Recall that the thick line within each boxplot indicates the median. From inspecting the plot, it looks to me like the median of the `Control` treatment is about 67, the median of the `Herbivory` treatment is about 55, and the median of the `Water` treatment is about 40.

Part (e)

We can add individual points like this (following the advice from the video https://media.ed.ac.uk/media/1_kbtid8o2):

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE)
```

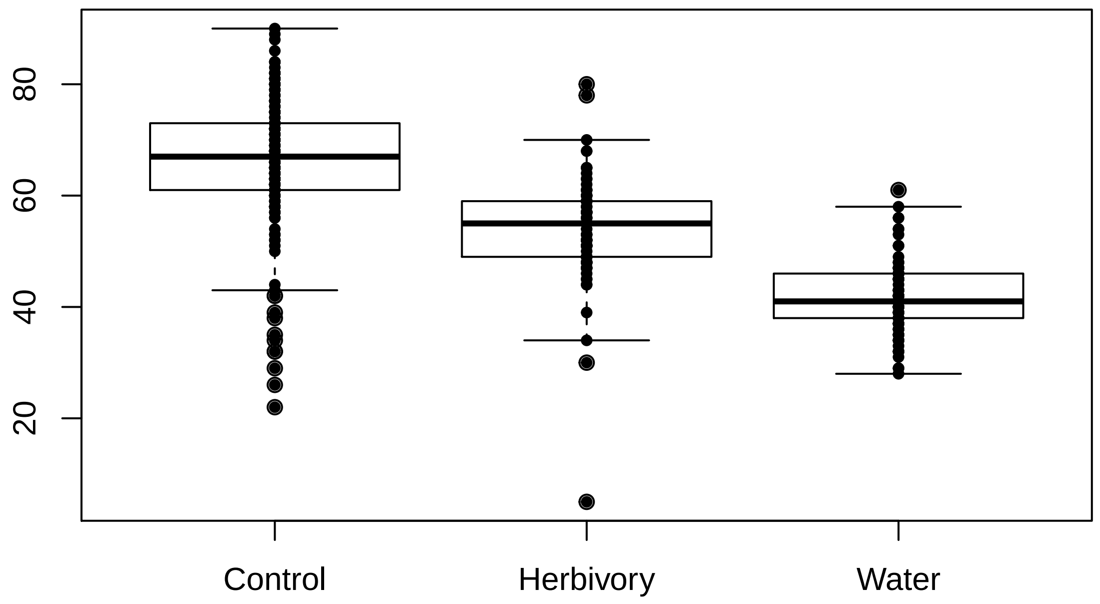


The option `add = TRUE` causes the output from `stripchart()` to be added to the previous plot that was made (i.e., the output from `boxplot()`), whereas the option `vertical = TRUE` causes the points to be oriented vertically (not horizontally, which is the default orientation). By orienting the points vertically, the output of `stripchart()` matches the orientation of `boxplot()` output.

Part (f)

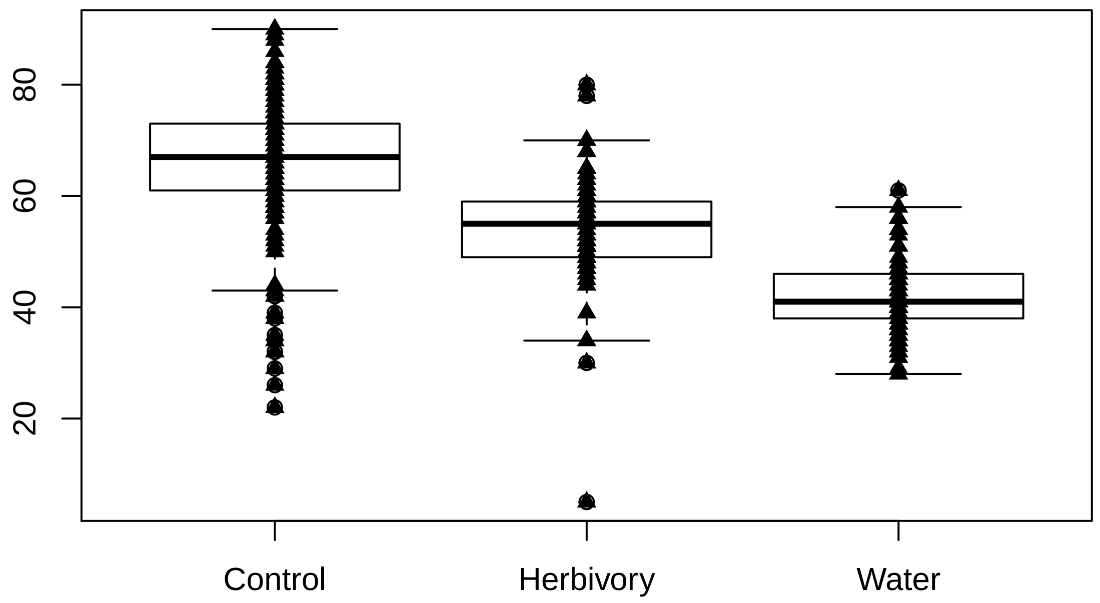
The squares in the plot are not very nice, in my opinion; squares were the default for `stripchart()`. The `pch` option allows us to specify different shapes by suggesting different numbers. For example, let's try setting `pch = 20`:

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 20)
```



...causing `stripchart()` to produce filled circles. What does `pch = 17` do? Let's find out:

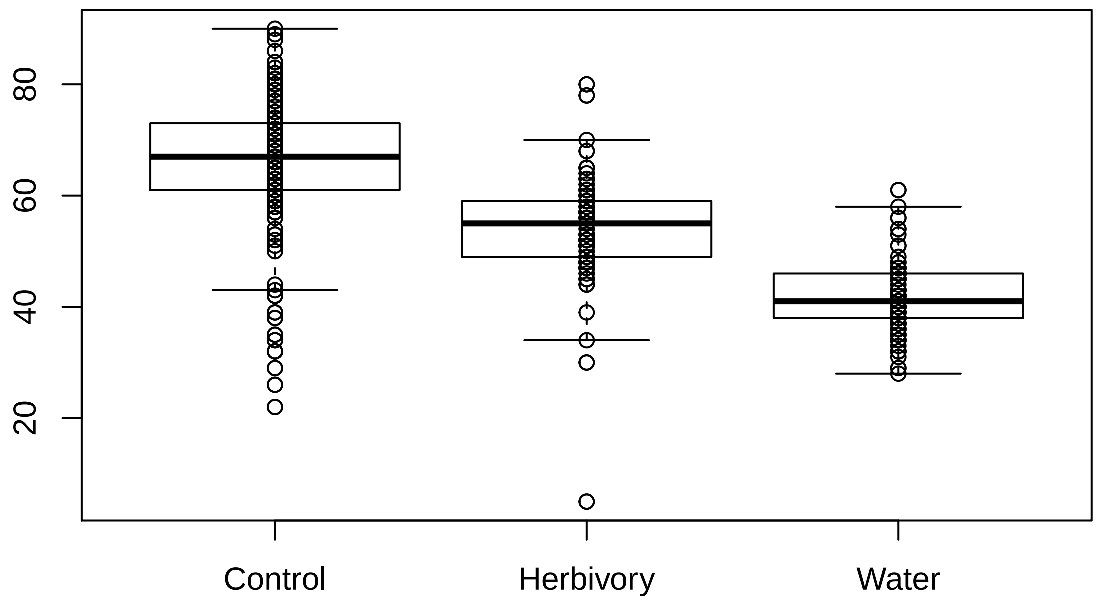
```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 17)
```



Ah! We get upward-facing triangles!

Finally, let's try `pch = 21`:

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21)
```

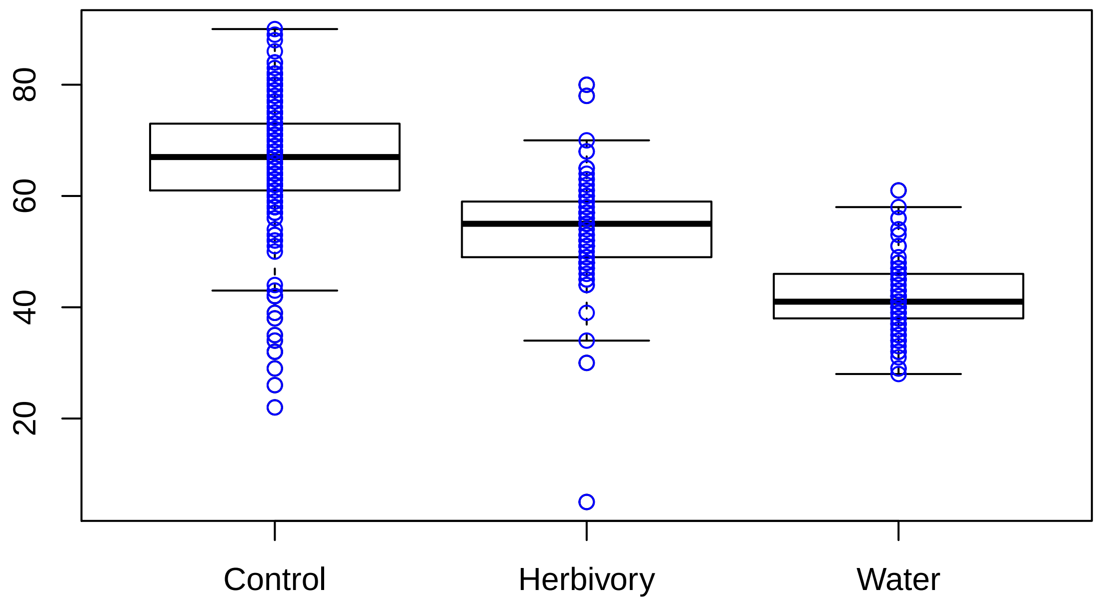


. . . and we get open circles. Try some more options to see what you get!

Part (g)

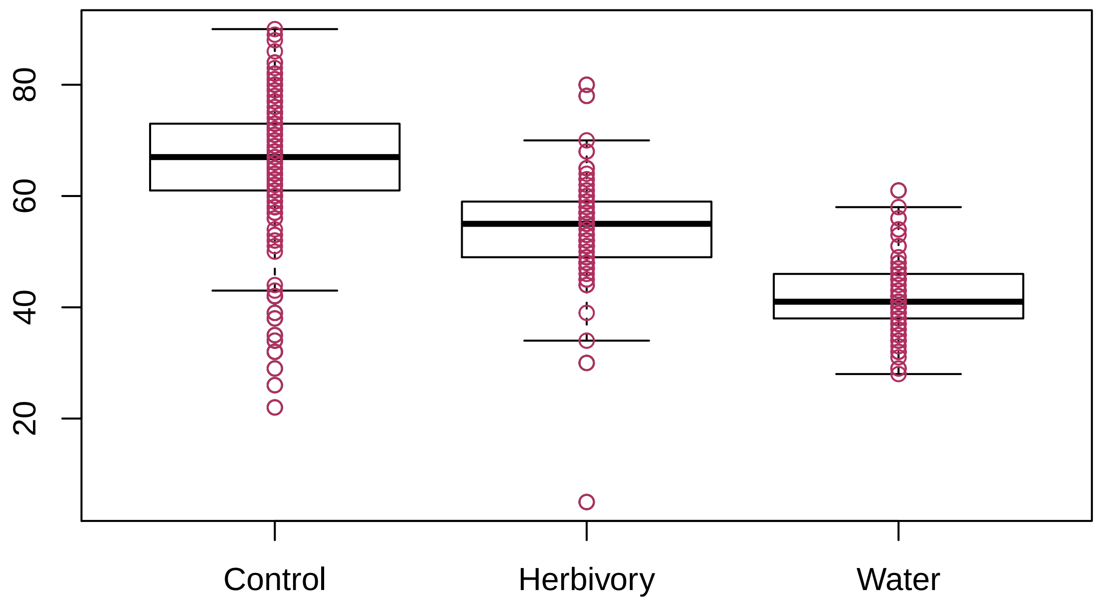
We can change the colour of the individual points with the `col` option. Let's start by trying blue:

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "blue")
```



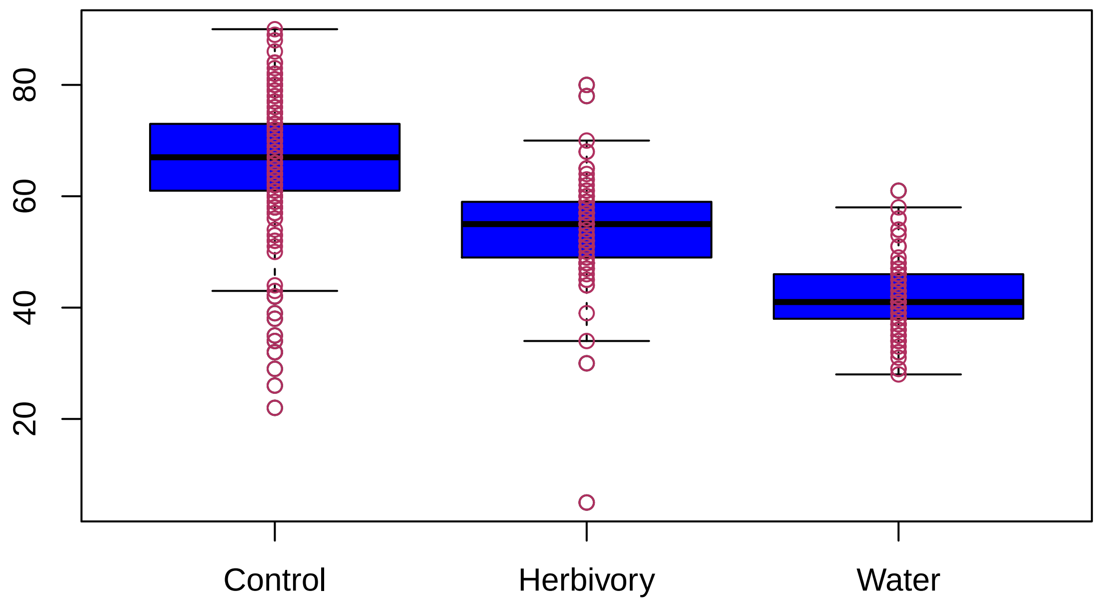
Nice? How about maroon?...

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon")
```



Although the question did not ask us to try this, we can also change the colour of the box-plot. Let's make the boxplots blue:

```r
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "", col = "blue")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon")
```

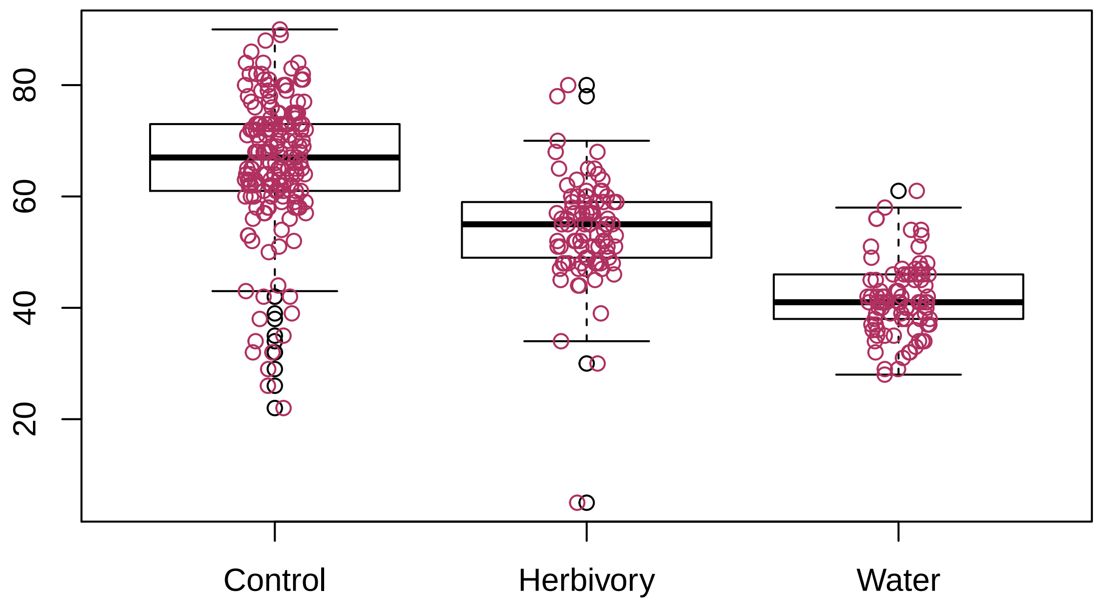


A bit weird? I agree. But hopefully you're starting to get a sense of what you can do, and how. We'll change the boxplots back to black in the next iteration.

Part (h)

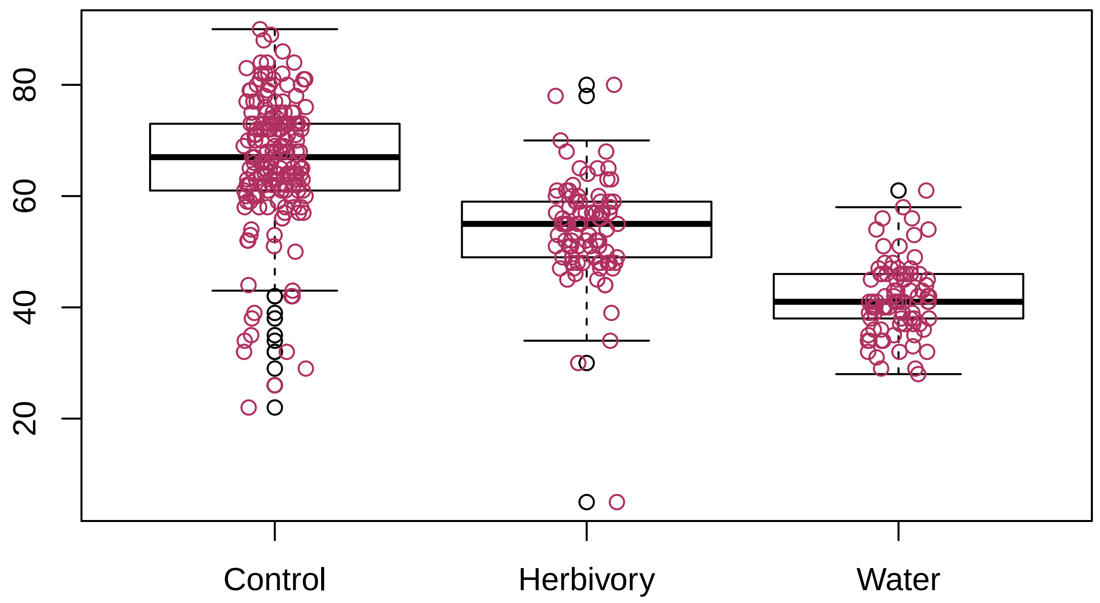
The individual points from `stripchart()` overlap with one another a lot, making it hard to see them. We can space them out using the option, `method = "jitter"`, like this:

```
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon", me
```



Let's provide the exact same commands again:

```
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon", me
```

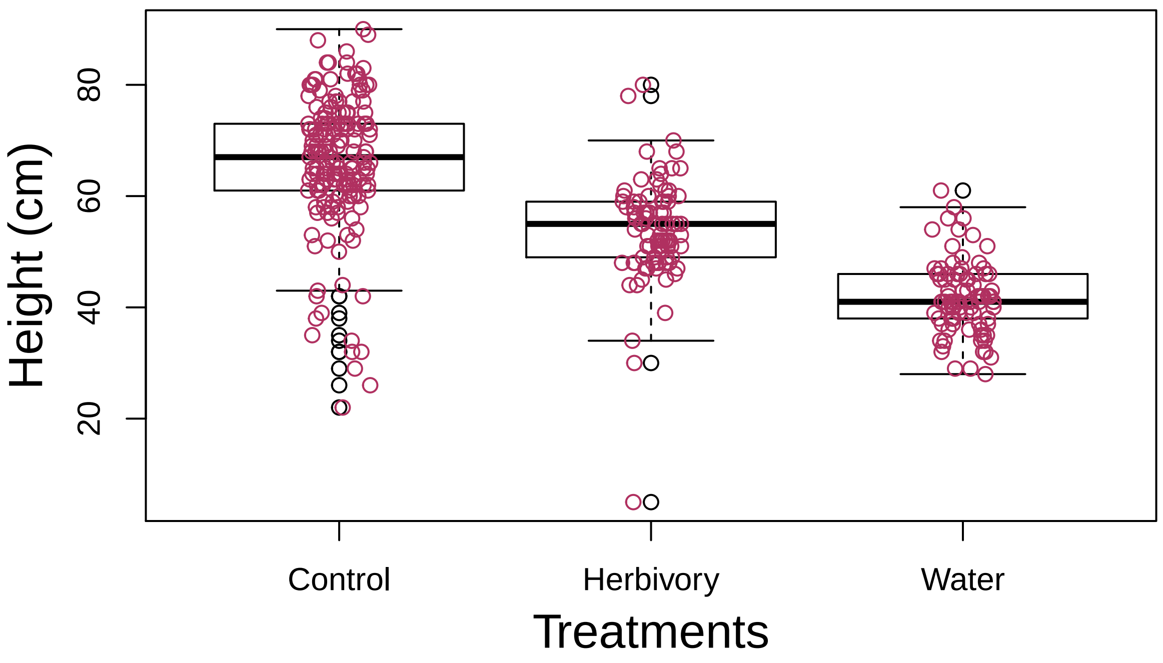


Now, compare the positions of the individual points in the two plots. You should see that they are not identical. This is not terribly important, but it is something that surprises many people when they first notice. The important thing to note is that the `method = "jitter"` option can make it easier to see individual points. That said, there are a lot of individual points in this plot, making it difficult to see them all; it is possible that this plot would be more effective if we omitted the individual points, and only presented the boxplots.

Part (i)

We will use the `mtext()` function to add axis labels. We can add axis labels like this:

```
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "")
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon", me
mtext("Treatments", side = 1, line = 2.5, cex = 1.5)
mtext("Height (cm)", side = 2, line = 2.5, cex = 1.5)
```

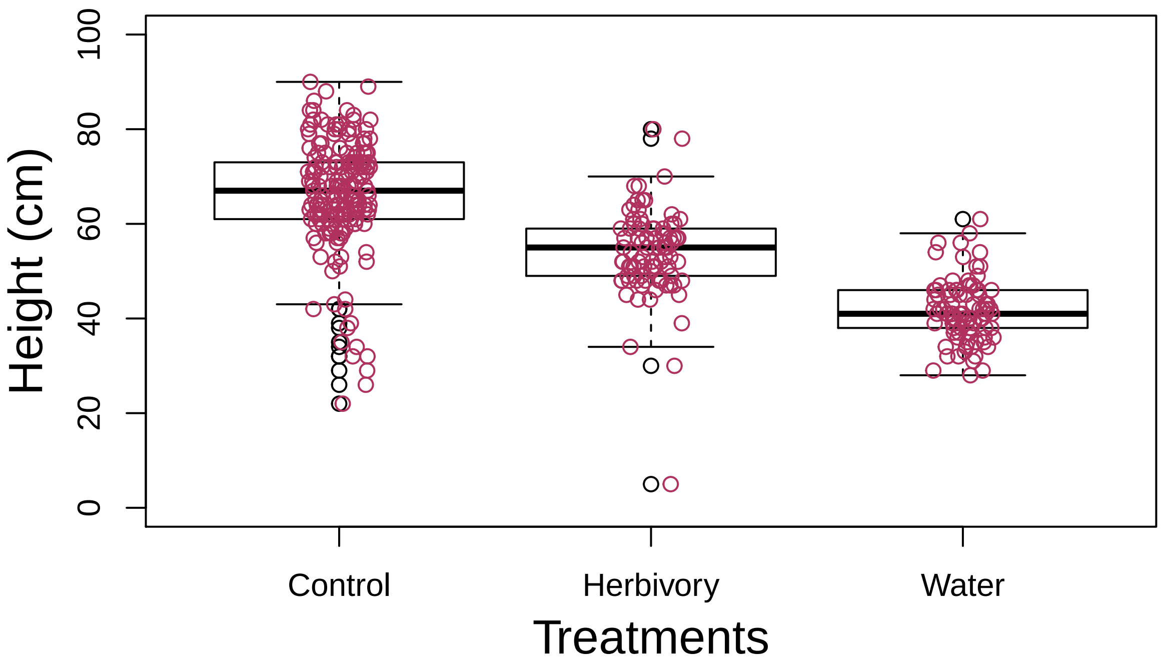


The options `side = 1` and `side = 2` cause the labels to be placed along the bottom x-axis and left-hand y-axis, respectively. What happens if we said `side = 3` or `side = 4`? Try it an find out. The options "`line =`" and "`cex =`" determine the distance that the label will be placed from the figure and the font size, respectively. Try changing their values to see what effect it has.

Part (j)

We can specify the range to be displayed on the y-axis using the option, `ylim`. This is important because we need to ensure that we present the entire plausible range of the data on the y-axis in order to no exaggerate the appearance of effects in our data. Often, this will mean that we ensure that the lowest value on the y-axis equals zero. For example:

```
boxplot(Height ~ TreatPool, data = plants, xlab = "", ylab = "", ylim = c(0,max(plants$Height) + 10))
stripchart(Height ~ TreatPool, data = plants, add = TRUE, vertical = TRUE, pch = 21, col = "maroon", me
mtext("Treatments", side = 1, line = 2.5, cex = 1.5)
mtext("Height (cm)", side = 2, line = 2.5, cex = 1.5)
```



The code '`max(plants$Height) + 10`' causes R to determine the maximum value of `Height` and then add 10 to it, and this sum will be the set as the upper limit to the figure. (We added 10 so that the plot does not stop abruptly at the greatest value of `Height`.)

Now, to finish, try changing the y-axis to some other maximum or minimum values. for example, try setting the range of the y-axis from -50 to 500.