Plots with Means and Error Bars

Crispin Jordan

11/02/2021

What is this document?

Our videos illustrate how to plot data using boxplots. Sometimes we might prefer to plot the mean value of a treatment with appropriate error bars, instead. This document will show you how to do this.

This document will also introduce you to several **new functions**:

- `axis()`
- `tapply()`
- `points()`
- `arrows()`

A note about importing data.

Recent versions of **R** import data differently than previous versions. Specifically, the `read.table()` function previously assigned data columns with letters or words as a 'Factor', which is sometimes helpful. More recent verions do not do so, and instead treat everything as numbers (even when data are not numeric). This new default behaviour can leads to errors. To remedy, we can add the `stringsAsFactors = TRUE` option to the `read.table()` command. For example, we will use a dataset called, `MS1_Data.csv`; the code, below, will import the data:

```
df <- read.table("MS1_Data.csv", header = TRUE, sep = ',', stringsAsFactors = TRUE)
```

These data were collected as part of an exercise for first year biomedical students at the University of Edinburgh. Let's start with a quick look at the data available:

```
head(df)
```

```
##   Sex Adj..Height Adj..Weight      BMI Waist.meas Hip.meas Waist.Hip.Ratio.WHpR
## 1   F       1.565          59 24.08925         73       97             0.7525773
## 2   F       1.615          54 20.70374         63       84             0.7500000
## 3   F       1.645          61 22.54229         80       88             0.9090909
## 4   F       1.745          56 18.39065         71       87             0.8160920
## 5   F       1.675          48 17.10849         64       84             0.7619048
## 6   M       1.855          77 22.37705         84       89             0.9438202
##   Waist.Height.Ratio.WHtR Dom.Hand Dom.Grip Non.dom.Grip dom.non.dom.ratio
## 1               0.4591195        R     27.0         25.0         1.0800000
## 2               0.3841463        R     21.0         21.0         1.0000000
## 3               0.4790419        R     27.0         26.0         1.0384615
## 4               0.4011299        R     22.0         20.0         1.1000000
## 5               0.3764706        R     20.0         17.0         1.1764706
## 6               0.4468085        R     53.2         53.6         0.9925373
##   Peak.Exp.Flow Sys.BP Dias.BP Heart.Rate
## 1           450    109      78         86
## 2           350    109      75         80
```

```
## 3           300    120       65          86
## 4           440     99       66          75
## 5           420     88       61          73
## 6           550    131       69          86
```

That's a lot of data; much more than we need for this document. Let's work with only the first two columns, which denote a subject's `Sex` and `Adj..Height` (adjusted height):

```
df.small <- df[,1:2]
head(df.small)
```

```
##   Sex Adj..Height
## 1   F       1.565
## 2   F       1.615
## 3   F       1.645
## 4   F       1.745
## 5   F       1.675
## 6   M       1.855
```

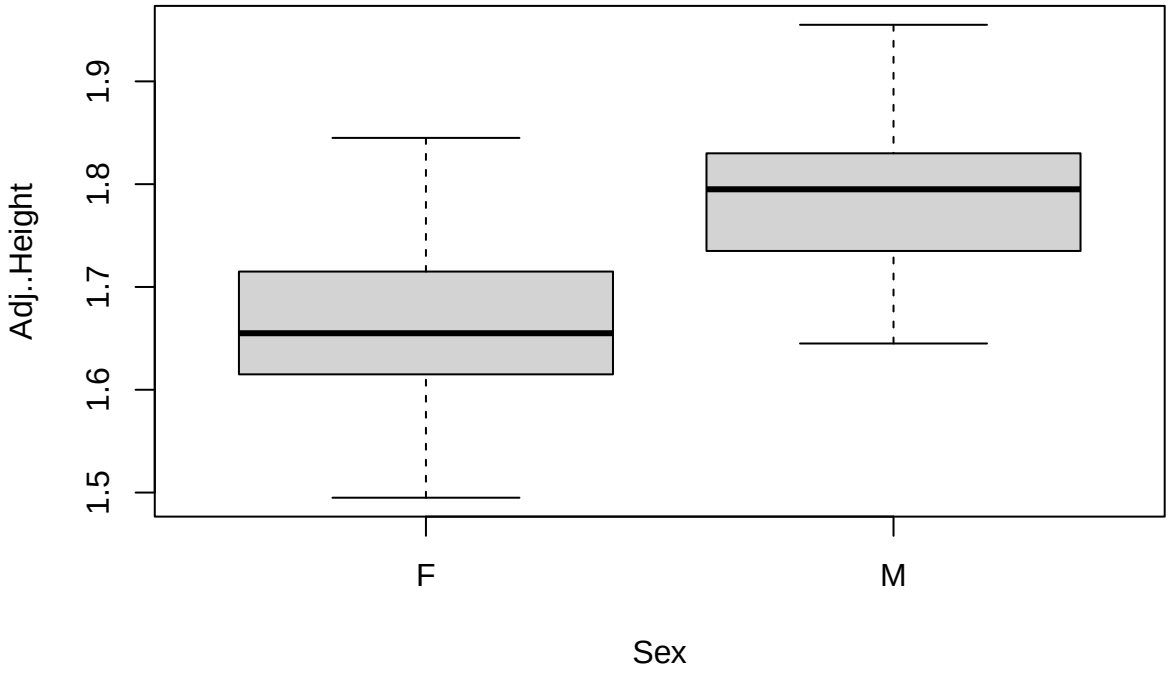
```
summary(df.small)
```

```
##  Sex      Adj..Height
##  F:265   Min.   :1.495
##  M:100   1st Qu.:1.625
##          Median :1.685
##          Mean   :1.694
##          3rd Qu.:1.755
##          Max.   :1.955
```

We can see that we about 2.5 times as much data for females as we do for males. And lots of data for our purposes!

We learned previously to plot data like these with a boxplot:

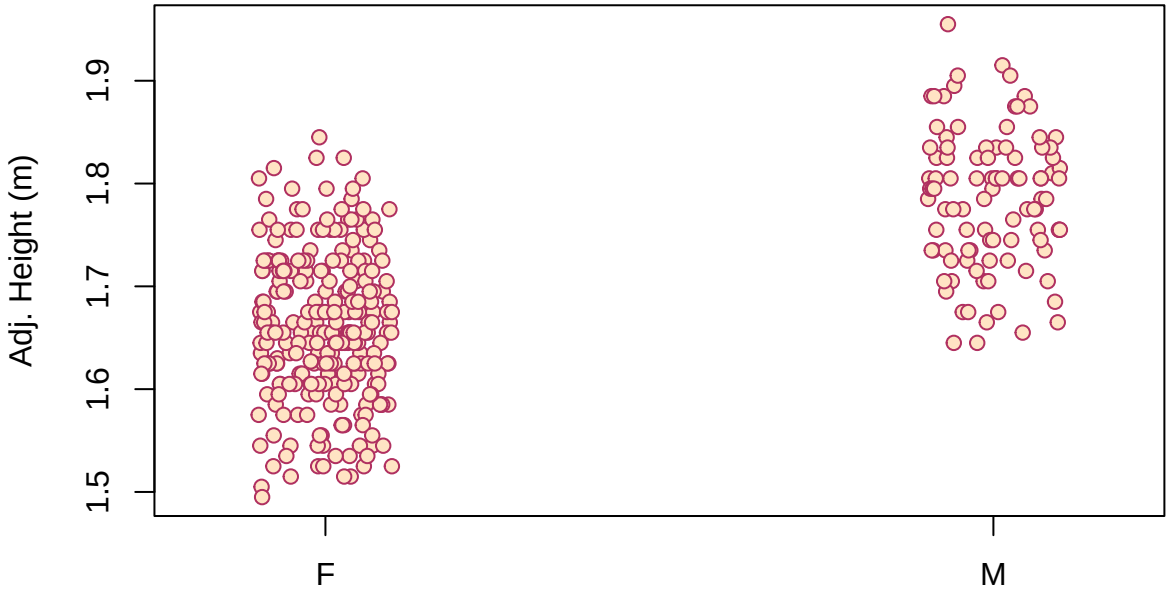
```
boxplot(Adj..Height ~ Sex, data = df.small)
```



But, what do we do if we do not want to produce a boxplot? Can we plot data with a mean and error bars, instead? (Yes, of course...) Please note that many approaches exist, including very nice options using `ggplot2`. Below, I demonstrate one possible approach. You can find loads of additional advice at https://rseek.org/.

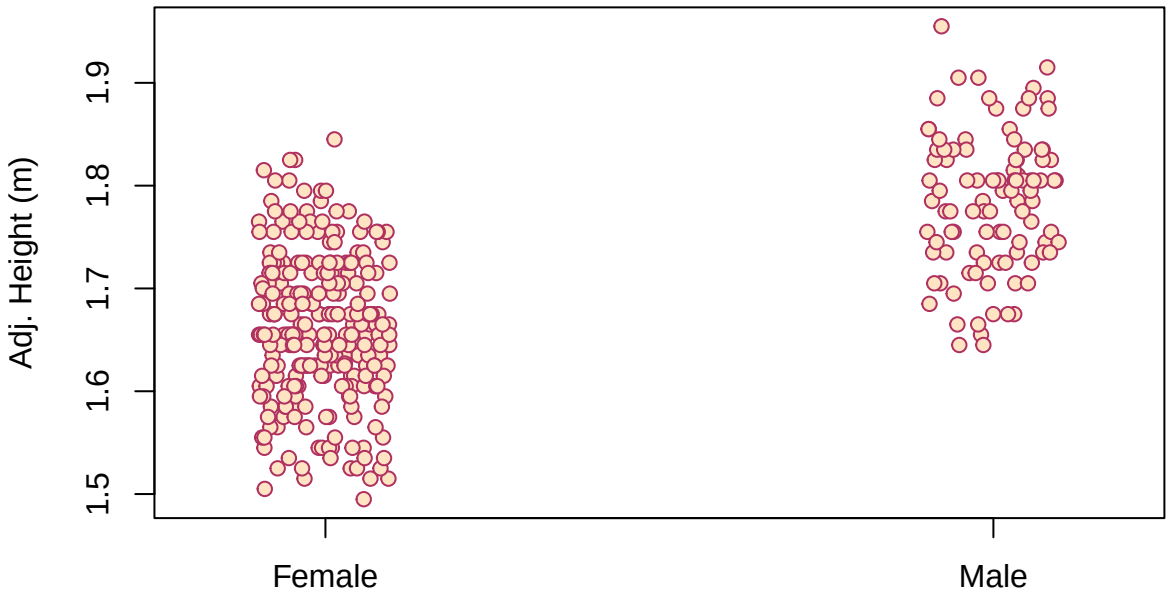
Now, lets start by plotting all of the raw data points. We'll use `stripchart()`. Importantly, we'll use our new column `SexAsNum`, for the x-axis; later in this document we will leverage this approach to specify coordinates at which we will plot mean values and error bars.

```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)")
```



This looks OK, but we have letters as labels on our x-axis. It would be better to have the names of the categories, instead. We can accomplish this with:

```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
```



We have introduced two new options / functions:

- The option `xaxt = 'n'` causes `stripchart()` to *not* produce letters for the x-axis (the option `yaxt = 'n'` also exists, and has the same effect for the y-axis).

- The `axis()` function provided our new axis labels. The '1' specifies that we're labeling the x-axis; `at=c(1,2)` specifies the values on the x-axis at which the labels will appear (**R** counts the position of the first set of data (`F`, in our case) as position 1, and counts upwards); finally, `labels = c("Female","Male")` indicates the labels we'd like to appear *at* the locations specified by the *at* option.

Notice that our figures do not show the value of zero along the y-axis; we could specify this using, for example, `ylim = c(0,(max(df.small$Adj..Height)+0.1))`, if we wished. I will not do so here.

Adding mean values to the plot.

Now we will add treatment mean values and error bars.

We can quickly obtain the mean values for each level of `Sex` using the `tapply()` function:

```
tapply(df.small$Adj..Height, df.small$Sex, mean)
```

```
##        F        M
## 1.659574 1.784750
```

What does `tapply()` do? (It is very handy.) `tapply()` considers the data that are listed first (`df.small$Adj..Height`), and *applies* the function that is listed third (`mean`); specifically, it applies this function to each of the levels listed second (`df.small$Sex`). Nice, eh?

I do not like to copy and paste if I can avoid it. So, to obtain these mean values, I will assign the output of `tapply()` to an object. We can obtain each mean using the square brackets, like this:

```
my.means <- tapply(df.small$Adj..Height, df.small$Sex, mean)
my.means[1]
```

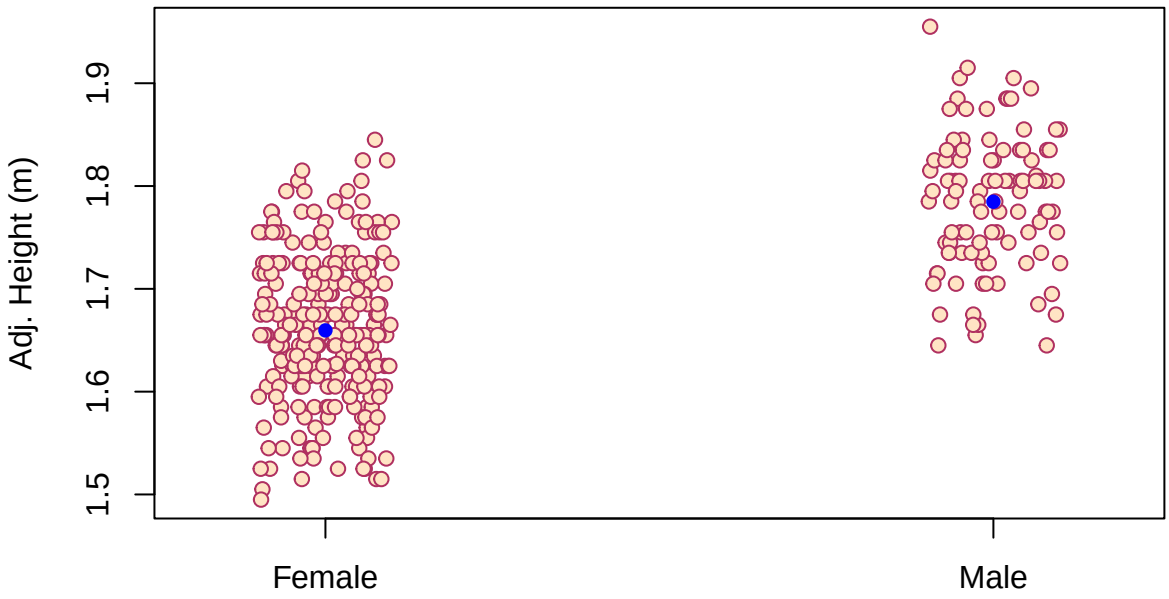
```
##        F
## 1.659574
```

```
my.means[2]
```

```
##       M
## 1.78475
```

Now we will add these add these means to the plot. We can do that using the function, `points()`.

```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
points(1:2,my.means, pch = 16, col = "blue")
```



Can you guess what `points()` is doing? First, it takes the values for the x-axis (`1:2`). Next, it takes values for the y-axis in the object, `my.means`. *(Note that we do not need to provide the y-values in an object, as I have done here. For example, we could use `c(1.659574, 1.78475)`, instead.)* These are the essential bits for `points()`. I have added additional options (`pch` and `col`) to specify the type and colour of point to add, respectively.

The blue dots on the output indicate the mean values for each treatment. Nice!

Adding Standard Error as error bars

Finally, we'd like to add error bars.

Perhaps surprisingly, a common way of adding error bars is to use arrows (with flattened arrowheads).

Here's an example where the error bars equal the standard error (SE) of the mean. We calculate SE as the standard deviation (`sd()`) of a sample, divided by the square root of the sample size; we can use `length()` to determine the samples size (`length` counts the number of observations in a sample). We might use `tapply()` to perform this calculation for both levels of `Sex` (`F` and `M`), like this:

```
my.sd <- tapply(df.small$Adj..Height, df.small$Sex, sd)
my.sd
```

```
##          F          M
## 0.07077201 0.06631108
```

```
my.n <- tapply(df.small$Adj..Height, df.small$Sex, length)
my.n
```

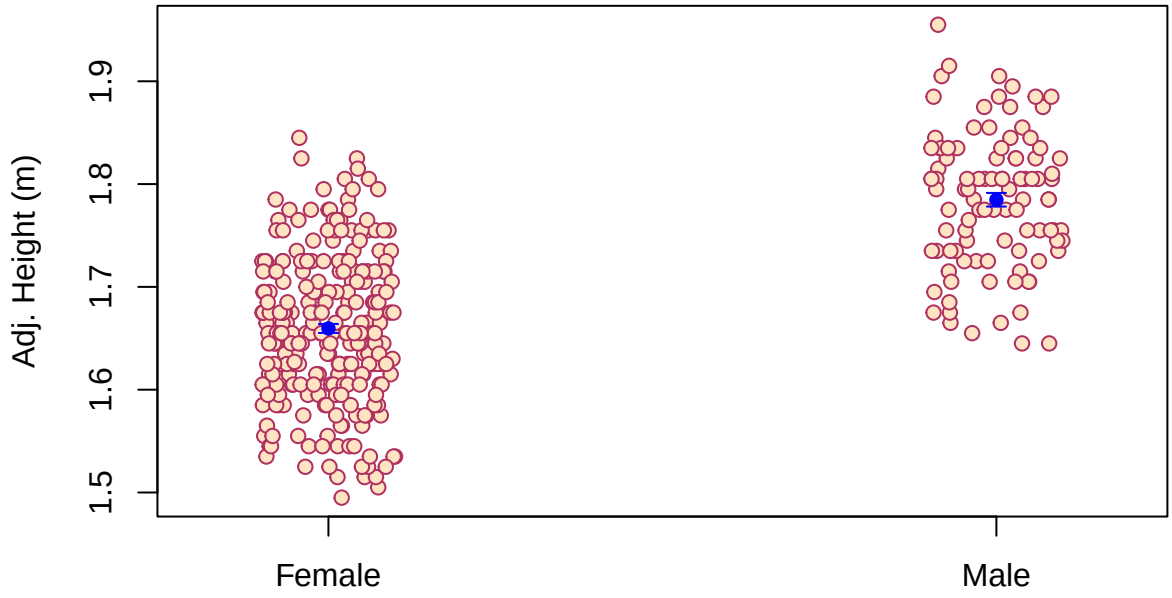
```
##   F   M
## 265 100
```

```
my.SE <-my.sd / sqrt(my.n)
my.SE
```

```
##           F           M
## 0.004347490 0.006631108
```

Now, we have SE's stored in the object, `my.SE`. Let's add arrows!

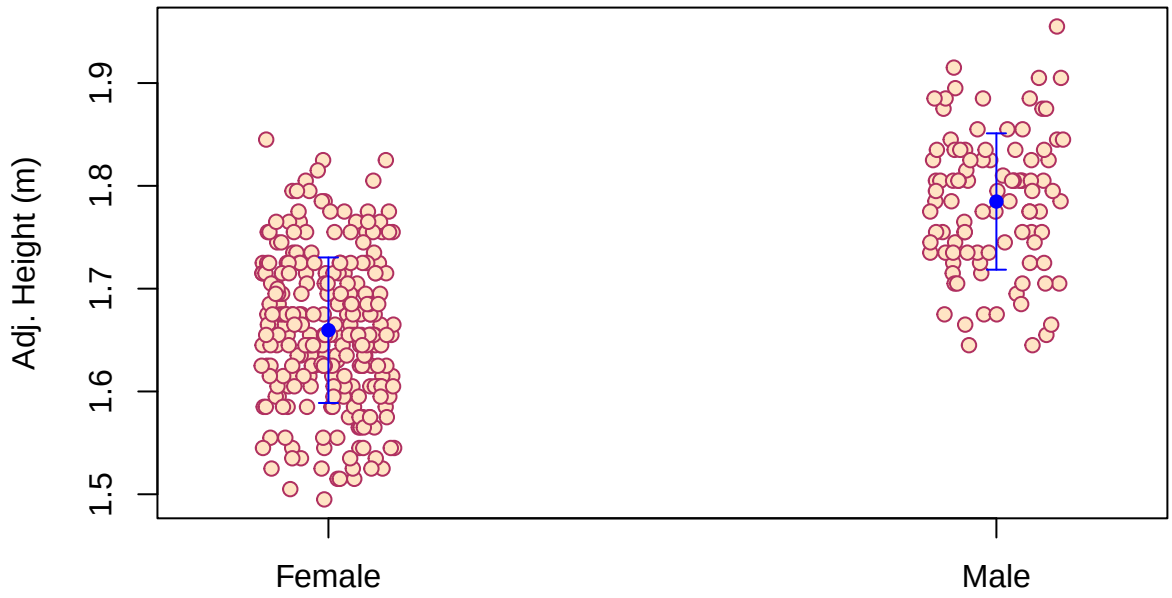
```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
points(1:2,my.means, pch = 16, col = "blue")
arrows(1:2,my.means-my.SE,1:2,my.means+my.SE, length=0.05, angle=90, code=3, col = "blue")
```



You will now see blue error bars, but they are very small (too small to let us appreciate our hard work).

Let's use standard deviation instead, simply so we can see the fruit of our labour more easily. (**Note that it is generally more meaningful to plot SE than standard deviation; we are only presenting standard deviation to make larger error bars, and help us see what we have done**). Here we go:

```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
points(1:2,my.means, pch = 16, col = "blue")
arrows(1:2,my.means-my.sd,1:2,my.means+my.sd, length=0.05, angle=90, code=3, col = "blue")
```



Nice!

Now, how have we used the `arrows()` function? The `arrows()` function takes the following arguments: `arrows(x0, y0, x1 = x0, y1 = y0, length = 0.25, angle = 30, code = 2, col = par("fg"), lty = par("lty"),lwd = par("lwd"), ...)`. The first two arguments (`x0,y0`) represent coordinates for one end of the arrow, and the third and fourth arguments represent x-y coordinates for the other end of the arrow. If you look at our code, we listed two `x0` coordinates (`1:2`) and two `y0` coordinates (remember that the objects, `my.means` and `my.SE` (for the first figure; we used `my.sd` in the second figure)). By providing two values at each of `x0` and `y0`, we were able to set the first set of coordinates for two arrows simultaneously. Note that our y-values involve the mean minus or plus the length of the error bar (SE or sd). We specified four options after we provided the x-y coordinates: `length` specifies the length of the arrow heads; `angle` specifies the angle at which the arrow heads meet the arrow's 'shaft'; `code = 3` specifies that we want to placve an arrow head on both ends of the arrow. (Finally, `col = "blue"` is obvious).

FYI, I obtained some of this advice by searching on https://rseek.org/, and found this very helpful website: https://stackoverflow.com/questions/13032777/scatter-plot-with-error-bars.

Calculating and plotting 95% Confidence Intervals

Some people prefer to plot 95% confidence intervals as their 'error bars' (e.g., rather than SE's). Here I demonstrate how to calculate a 95% CI for a mean value (of normally distributed data) and use it in a plot.

We can convert a **standard error** (calculated and plotted above) to a 95% CI by multiplying our estimated SE by an appropriate value of the t-distribution, $t_{0.05(2),df}$. The (2) refers to a 'two-tailed test', and *df* refers to the *degrees of freedom* (which we discuss in more detail when we introduce 1-Factor General Linear Models). Briefly, when calculating the 95% CI of a mean, the *df* for the mean equals the number of measurements for the mean (assuming they are independent) minus 1. If you recall from above, the means for females and males were calculated from sample sizes of 265 and 100, respectively (see output of `my.n`). Therefore, their *df*'s equal 264 and 99, respectively.

We can calculate the appropriate t-value ($t_{0.05(2),df}$) using:

```
qt(0.975,df=df.of.mean)
```

Therefore, the appropriate t-values for females equals:

```
qt(0.975,df=264)
```

```
## [1] 1.96899
```

and that for males equals:

```
qt(0.975,df=99)
```

```
## [1] 1.984217
```

Remembering that **R** calculated values for treatments in alphabetical order when we used `tapply()`, above (female, then male; *notice this order in the plots, too*), let's store these t-values in a vector in the same order:

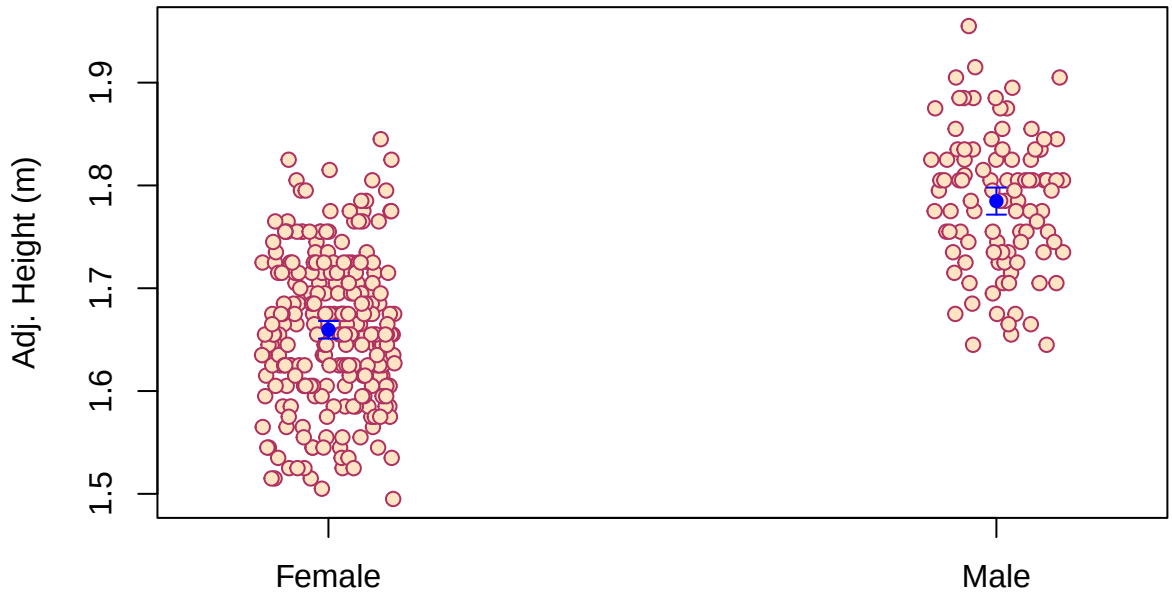
```
my.t <- c(qt(0.975,df=264),qt(0.975,df=99))
my.t
```

```
## [1] 1.968990 1.984217
```

We can incorporate this code for the t-values into the code we used, above, to plot of individual values with **SE's** (not standard deviation); we multiply the SE's (stored in `my.SE`) by their appropriate t-values (stored in `my.t`) to plot 95% CI's:

```
stripchart(Adj..Height ~ Sex, data = df.small, vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
```

```
points(1:2,my.means, pch = 16, col = "blue")
arrows(1:2,my.means-my.t*my.SE,1:2,my.means+my.t*my.SE, length=0.05, angle=90, code=3, col = "blue")
```



Notice that these error bars are a bit bigger (by about 1.9 times!) than we saw when plotting standard errors.

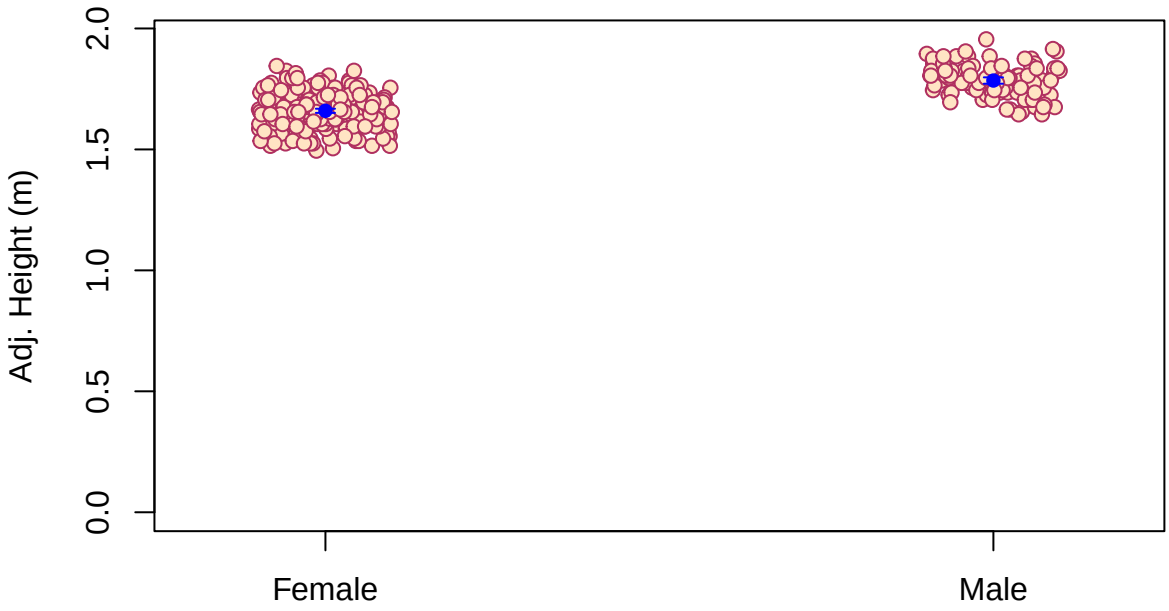
A final note: Consider an appropriate range for the y-axis

We've focused so far on how we can create our plot. Now that we have our code, we need to think about our plot and whether our plot displays the data in a way that meaningfully represents differences between the groups.

You may recall from our videos (discussing plotting data) that, a y-axis with a range that only spans the range of the data will tend to exaggerate the apparent differences between groups. This is what we did, above. As a result, our plot, above, likely exaggerates the apparent difference between femaes and males. That's bad for our conclusions and for science.

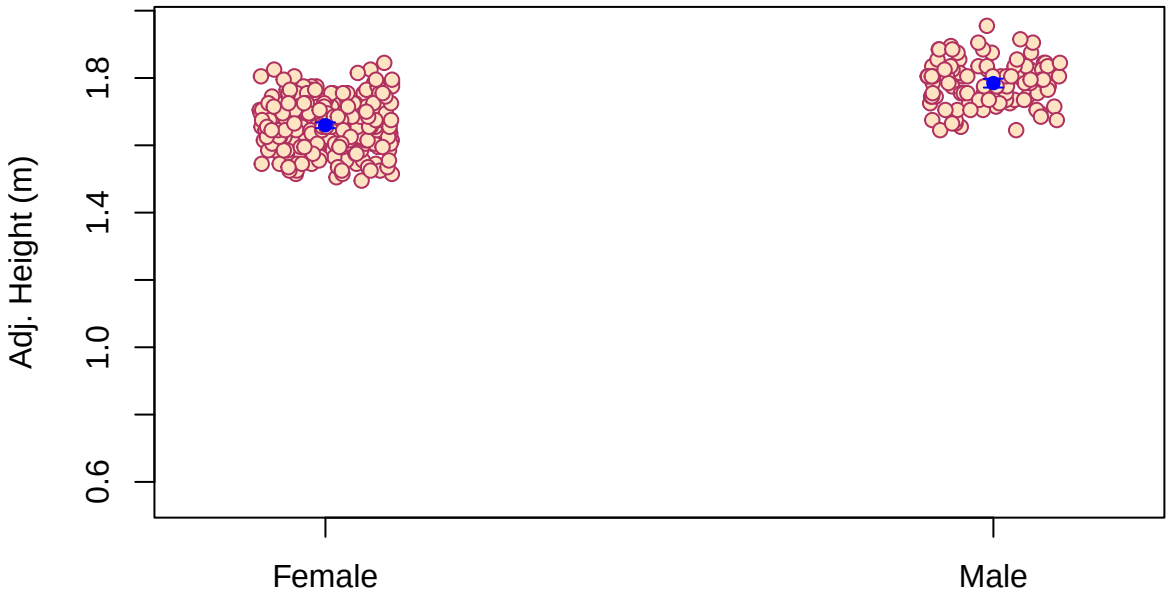
If we set the y-axis to have a range of zero to the maximum height value (`ylim=c(0,max(df.small$Adj..Height))`), our plot looks like this (again, with 95% CI's):

```
stripchart(Adj..Height ~ Sex, data = df.small, ylim=c(0,max(df.small$Adj..Height)),
           vertical = TRUE, method = 'jitter',
           pch = 21, col = "maroon", bg = "bisque",ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
points(1:2,my.means, pch = 16, col = "blue")
arrows(1:2,my.means-my.t*my.SE,1:2,my.means+my.t*my.SE, length=0.05, angle=90, code=3, col = "blue")
```



. . . Now, the difference between females and males appears much smaller! But, does it make sense to have the y-axis start at zero? Maybe not, because it is biologically impossible to have a height of zero (this is equivalent to having a mass of zero). So, our choice of starting at zero may be too extreme. We might, instead, start the y-axis at the smallest known height of an adult human (0.55m, according to Google!). Doing so might display the data on a biologically reasonable scale without exaggerating differences between females and males. We do so here (note the change to `ylim = ...`):

```
stripchart(Adj..Height ~ Sex, data = df.small, ylim=c(0.55,max(df.small$Adj..Height)),
           vertical = TRUE, method = 'jitter', pch = 21, col = "maroon", bg = "bisque",
           ylab = "Adj. Height (m)", xaxt = 'n')
axis(1,at=c(1,2), labels = c("Female","Male"))
points(1:2,my.means, pch = 16, col = "blue")
arrows(1:2,my.means-my.t*my.SE,1:2,my.means+my.t*my.SE, length=0.05, angle=90, code=3, col = "blue")
```



The point is to select a biologically meaningful scale that accurately (e.g., without exaggeration) reflects the differences between groups. This takes some thinking on your part, but it will help readers make appropriate

conclusions from your work. That's good for science.

Enjoy plotting!