Basic commands; 1-Factor GLM; plotting means with error bars

Crispin Jordan

05/04/2022

Most common commands for 1-factor GLM

We'll use the data in the file, `aphid.csv` for our example.

Importing data:

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

Examine the data:

```
aphid   #Providing the name of the dataframe causes R to print the contents
head(aphid)  # Display the top of the dataframe
tail(aphid)  #Display the end of the dataframe
str(aphid)   #Examine the sturcture of the dataframe
```

How to convert a variable to a factor:

Here, we demonstrate using the variable, `infectionStatus`.

```
aphid$infectionStatus <- factor(aphid$infectionStatus)
```

Plot the data:

This is a great way to inspect your data. *Below, we provide an alternative way to present your data that some journals may prefer.*

```
boxplot(color ~ infectionStatus, data = aphid)
stripchart(color ~ infectionStatus, data = aphid, add = TRUE, vertical = TRUE, method = "jitter", pch =
```

Formulating the model:

```
#Generic code:
model.output <- lm(dependent.variable ~ independent.variable, data = my.data)
#Specific example:
aphid.lm <- lm(color ~ infectionStatus, data = aphid)
```

To view residuals:

```
#Generic code:
plot(model.output)
#Specific example:
plot(aphid.lm)
```

View model coefficients:

```
#Generic code:
summary(model.output)
#Specific example:
summary(aphid.lm)
```

Obtain overal results in ANOVA table form:

```
#Generic code:
anova(model.output)
#Specific example:
anova(aphid.lm)
```

post-hoc tests

Open emmeans library:

```
library(emmeans)
```

Calculating the mean values of each level of a factor:

```
#Generic code:
emmeans.output <- emmeans(model.output, "Name.of.factor.in.model")
emmeans.output  #This line of code allows us to see the calculated means
#Specific example:
aphid.emmeans <- emmeans(aphid.lm, "infectionStatus")
aphid.emmeans   #This line of code allows us to see the calculated means
```

Making pairwise contrasts, and obtain effect sizes with SE's:

```
#Generic code
pairs.output <- pairs(emmeans.output)
pairs.output  #This code allows us to see the output we stored from `pairs()`
#Specific example
aphid.pairs <- pairs(aphid.emmeans)
aphid.pairs  #This code allows us to see the output we stored from `pairs()`
```

Obtain 95% CI's for effect sizes

```
#Generic example:
confint(pairs.output)
#Specific example:
confint(aphid.pairs)
```

Plotting data with both individual values and means and SE's

Before plotting the data, let's first determine the means and SE's to be added to the plot.

We can calculate the means for each group like this:

```
#Generic:  my.means <- tapply(data.fr$dep.var, data.fr$indep.var, mean)
#Specific example:
aphid.means <- tapply(aphid$color, aphid$infectionStatus, mean)
aphid.means
```

```
##   infected   original uninfected
##   37.81818   20.54167   18.65556
```

To calculate standard errors, we also need to know the sample size for each group. We can obtain it following the same approach as we did to calculate the mean values (the function, `length()` provides a count of the number of entries of something, in this case, the number of measurements in the variable `color` for each level of `infectionStatus`):

```
aphid.n <- tapply(aphid$color, aphid$infectionStatus, length)
aphid.n
```

```
##   infected   original uninfected 
##         11         12          9
```

Finally, to calculate SE's, we need information about the standard deviation for the data. We have two options:

- calculate `sd` for each level of `infections` using the raw data in our dataframe (`aphid`);
- use the estimate of `sd` for the residuals of our model.

We will use the latter approach. One benefit of this approach is that the SE's we generate from it will directly reflect the information used to infer differences between groups in our general linear model.

We can obtain the estimate of the standard deviation of the residuals by running the model, and then using the `sigma()` function to extract the estimate of `sd` from the model output:

```
aphid.lm <- lm(color ~ infectionStatus, data = aphid)
model.sd <- sigma(aphid.lm)
model.sd
```

```
## [1] 4.258653
```

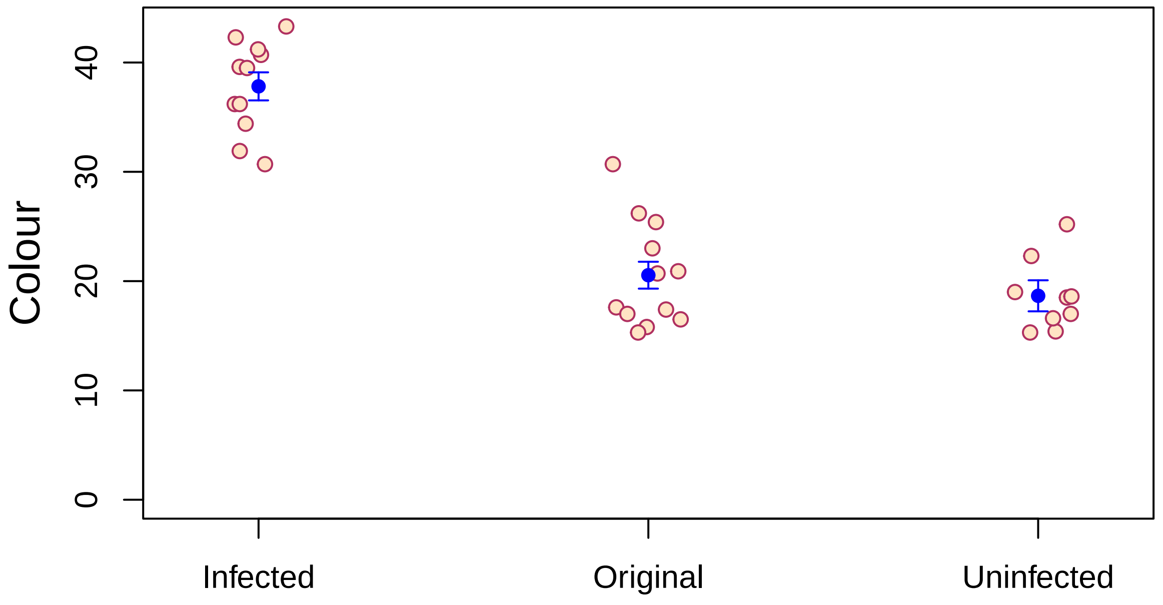
We can now calculate SE for each treatment group:

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

```
##   infected   original uninfected 
##   1.284032   1.229367   1.419551
```

And we can plot the data with means and SE's like this:

```
stripchart(color ~ infectionStatus, data = aphid, vertical = TRUE, method = 'jitter',
pch = 21, col = "maroon", bg = "bisque",ylab = "", xaxt = 'n',
ylim = c(0,max(aphid$color)))
mtext(side = 2, line = 2.5, "Colour", cex = 1.3)
axis(1,at=c(1,2,3), labels = c("Infected", "Original", "Uninfected"))
points(1:3,aphid.means, pch = 16, col = "blue")
arrows(1:3,aphid.means-my.SE,1:3,aphid.means+my.SE, length=0.05, angle=90, code=3, col = "blue")
```



See advice in the file that demonstrates similar plotting tricks when we have only 2 treatment groups for explanation of the above code.

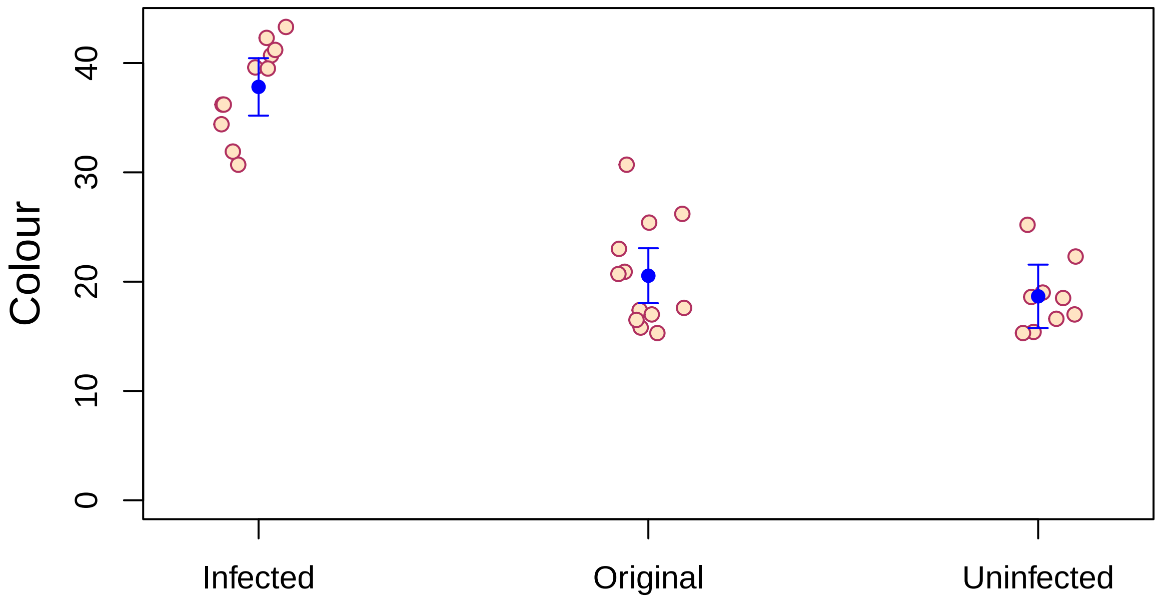
Plotting data with both individual values and means with 95% CI's

We can convert the SE's, calculated and plotted, above, by multiplying the SE's we calculated by the appropriate t-value. The function, `qt()`, can provide this value if we provide it with the appropriate proportion of the sampling distribution and degrees of freedom. The former value equals `0.975` (if we want to calculate 95% CI's specifically). The latter equals the within-treatment (i.e., 'residual', or 'error') degrees of freedom from our model. We can obtain this value by proving the name of our model output to the function, `df.residual()`.

Overall, given that our model output is stored in the object, `aphid.llm`, we can obtain the appropriate t-value with: `qt(0.975,df=df.residual(aphid.lm))`.

Here's the figure we can generate:

```
stripchart(color ~ infectionStatus, data = aphid, vertical = TRUE, method = 'jitter',
pch = 21, col = "maroon", bg = "bisque",ylab = "", xaxt = 'n', ylim = c(0,max(aphid$color)))
mtext(side = 2, line = 2.5, "Colour", cex = 1.3)
axis(1,at=c(1,2,3), labels = c("Infected", "Original", "Uninfected"))
points(1:3,aphid.means, pch = 16, col = "blue")
arrows(1:3,aphid.means-qt(0.975,df=df.residual(aphid.lm))*my.SE,1:3,
       aphid.means+qt(0.975,df=df.residual(aphid.lm))*my.SE,
       length=0.05, angle=90, code=3, col = "blue")
```



See the code, below, to see that we have calculated SE's and 95% CI's in the same way as done by the function, `emmeans`:

```
library(emmeans)
aphid.emmeans <- emmeans(aphid.lm, "infectionStatus")
aphid.emmeans  #means, SE's and 95% CI's calculated by emmeans
```

```
##  infectionStatus emmean   SE df lower.CL upper.CL
##  infected          37.8 1.28 29     35.2     40.4
##  original          20.5 1.23 29     18.0     23.1
##  uninfected        18.7 1.42 29     15.8     21.6
```

```
##
## Confidence level used: 0.95
```

```
my.SE  #Our estimated SE's calculated, above
```

```
##   infected   original uninfected
##   1.284032   1.229367   1.419551
```

```
aphid.means-qt(0.975,df=df.residual(aphid.lm))*my.SE  #Lower 95% CI
```

```
##   infected   original uninfected
##   35.19204   18.02733   15.75225
```

```
aphid.means+qt(0.975,df=df.residual(aphid.lm))*my.SE  #Upper 95% CI
```

```
##   infected   original uninfected
##   40.44432   23.05601   21.55886
```