Plot Making

Wynne Moss

September 1, 2019

General: plotting in R markdown

Important: In R markdown, all plotting commands for a single plot must be enclosed in curly brackets `{}` If you need to do any calculations (e.g. take the mean, run a linear model, subset dataframe), these should be done outside (before) the curly brackets. Non-plotting code inside the curly brackets will produce an error when knitting.

Sometimes, it's easiest to open a blank R script and play with your plotting functions there. This has the benefit of opening the plot in the plotting window so you can play with the size, and export it. Once your plot is looking great, you can copy-paste the code back into your .Rmd document

Stripcharts

Stripcharts are a good way of representing data when:

- Your predictor (independent) variable is CATEGORICAL
- Your response (dependent) variable is CONTINUOUS

Stripcharts are similar to bar charts, but have the advantage of displaying the raw data.

An informative stripchart should:

- Show all the data points
- Have clearly labeled axes
- Show the central tendency (mean or median) of each group
- Show a measure of uncertainty (standard deviation, standard error, or confidence interval) for each group

Since the reader doesn't know what your error bars mean (are they SD, SE, CI?) a legend or informative caption should be used to explain them.

The basic steps for a stripchart are:

1) Find the means and standard deviations or standard error for each group and store in vectors
2) Plot the points, and overlay the means and error bars

Modifications to the default will improve your plot. Consider adding these arguments to your stripchart command:

- `method = "jitter"`: spreads the points out along the x axis to avoid overlapping datapoints
- `ylim = c(0, 100)` : changes the minimum (0) and maximum (100) of the y axis. Change to fit your data.
- `ylab = "Y axis label"` and `xlab = "X axis label"` : change axis labels
- `pch = 16`: change pch number to plot different types of symbols. 16 is a closed circle
- `col = "colorname"`: change the color of points
- `cex = 1.2`: changes the size of points. Anything greater than 1 is larger than the default; less than 1 is smaller than the default.

Remember, all plotting commands for a single plot must be enclosed in curly brackets `{}`

Stripchart example 1 (4 groups, with SD)

This dataset is a built-in R dataset giving the weights of chicks on different diets.

```
# look at the data (the data are already in R as an example dataset)
head(ChickWeight)
```

```
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1
```

```
str(ChickWeight)
```

```
## Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   578 obs. of  4 variable
##  $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
##  $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
##  $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 15 15 15 15 ...
##  $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "outer")=Class 'formula'  language ~Diet
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "labels")=List of 2
##   ..$ x: chr "Time"
##   ..$ y: chr "Body weight"
##  - attr(*, "units")=List of 2
##   ..$ x: chr "(days)"
##   ..$ y: chr "(gm)"
```

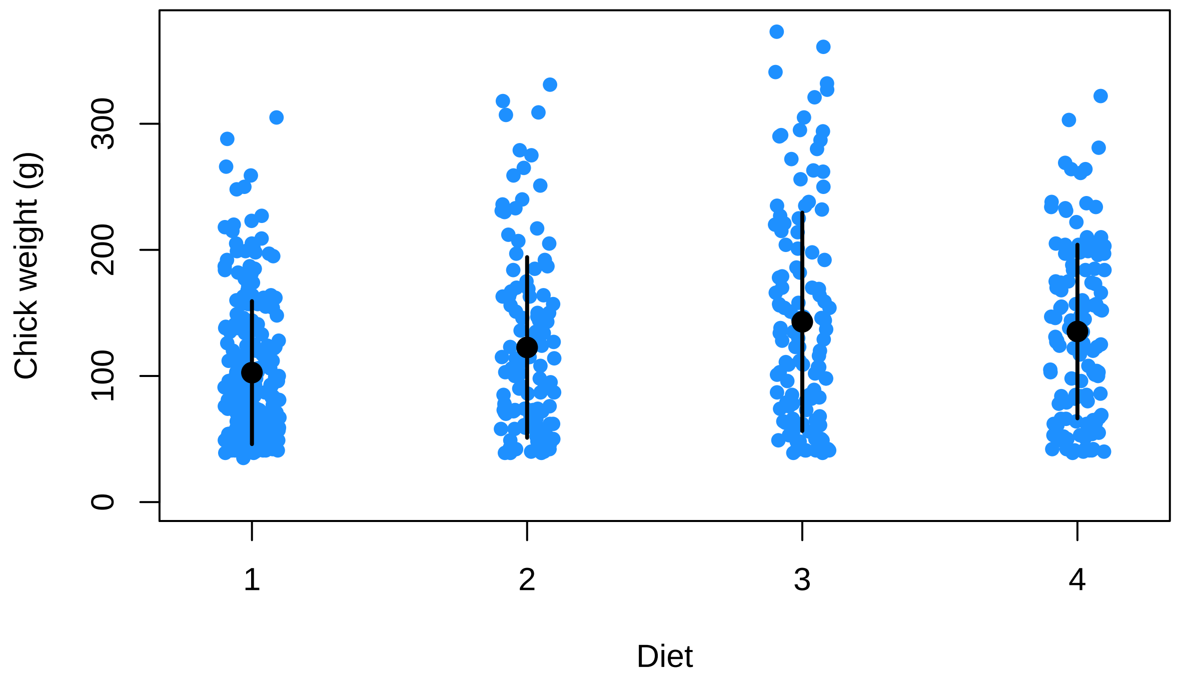
```
# get the mean weight per diet
mean.weights <- aggregate(weight~Diet, ChickWeight, FUN = mean)
# notice that the output from this is a table
# if we want to get the weights out for plotting we need to index it
# e.g. mean.weights$weight
mean.weights
```

```
##   Diet   weight
## 1    1 102.6455
## 2    2 122.6167
## 3    3 142.9500
## 4    4 135.2627
```

```
# get the standard deviation in weight for each diet
sd.weights <- aggregate(weight~Diet, ChickWeight, FUN = sd)
sd.weights
```

```
##   Diet   weight
## 1    1 56.65655
## 2    2 71.60749
## 3    3 86.54176
## 4    4 68.82871
```

```r
# plot the data (we have 4 diets, so we are indexing from 1:4)
{stripchart(weight~Diet, vertical = TRUE, method = "jitter",
           pch = 16, col = "dodgerblue", data = ChickWeight,
           ylab = "Chick weight (g)", xlab = "Diet", ylim = c(0,375))

# add in the means
# mean.weights$weight is the y value
# 1:4 are the x positions since there are 4 categories ("factor levels")
# to automate the x positions, ask it to go from 1 to however many factor levels I have
# length(levels(mean.weights$Diet)) will give you 4

points(mean.weights$weight ~ c(1:length(levels(mean.weights$Diet))),
       pch = 16, col = "black", cex = 1.5)
# the old way would be to tell it we have 4 categories. either is fine:
# points(mean.weights$weight ~ c(1:4), pch = 16, col = "black", cex = 1.5)

# now, add in the standard deviation bars.
# this is the toughest part!
# x0 = : gives the x positions of the bottom of the error bars
# X0 = 1:4 means there are 4 groups OR use R to tell you how many groups using length(levels)
# x1 = : gives the x positions of the top of the error bars
# x1 = 1:4 for 4 groups OR use length again
# the bars are VERTICAL so we want the starting and ending x positions to be the same

# y0 = : gives the y positions of the bottom of the error bars (mean - SD)
# y1 = : gives the y positions of the top of the error bars (mean + SD)

# can change line width (lwd) and color

segments( x0 = c(1:length(levels(mean.weights$Diet))), x1 = c(1:length(levels(mean.weights$Diet))),
          y0 = mean.weights$weight - sd.weights$weight,
          y1 = mean.weights$weight + sd.weights$weight,
          col = "black", lwd=2)
}
```



Let's do the same thing using `tapply` instead of `aggregate`. Many people prefer using the `tapply` function. Either function works, but they give the means and standard deviations in slightly different formats.

```
# get mean chick weight for each diet using tapply:
mean.weights.2 <- tapply(ChickWeight$weight, ChickWeight$Diet, mean)
mean.weights.2
```

```
##        1        2        3        4
## 102.6455 122.6167 142.9500 135.2627
```

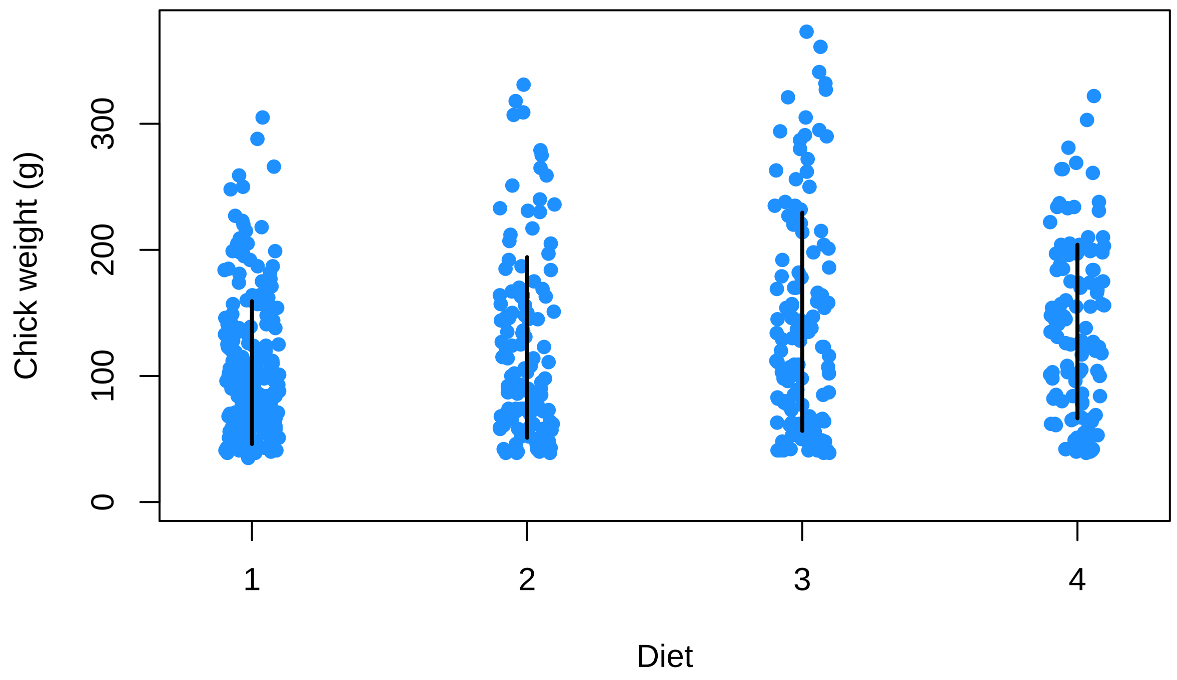
```
# note the difference in format from the aggregate version:
mean.weights
```

```
##   Diet   weight
## 1    1 102.6455
## 2    2 122.6167
## 3    3 142.9500
## 4    4 135.2627
```

```
# get standard deviation in chick weight for each diet:
sd.weights.2 <- tapply(ChickWeight$weight, ChickWeight$Diet, sd)
sd.weights.2
```

```
##        1        2        3        4
## 56.65655 71.60749 86.54176 68.82871
```

```
# plot the data:
{stripchart(weight~Diet, vertical = TRUE, method = "jitter",
           pch = 16, col = "dodgerblue", data = ChickWeight,
           ylab = "Chick weight (g)", xlab = "Diet", ylim = c(0,375))
  segments( x0 = c(1:4), x1 = c(1:4),
          y0 = mean.weights.2 - sd.weights.2,
          y1 = mean.weights.2 + sd.weights.2,
          col = "black", lwd=2)
}
```



Stripchart example 2 (two groups, with standard error)

This dataset gives the growth rate of teeth for guinea pigs on vitamin C supplements delivered via orange juice or ascorbic acid.

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

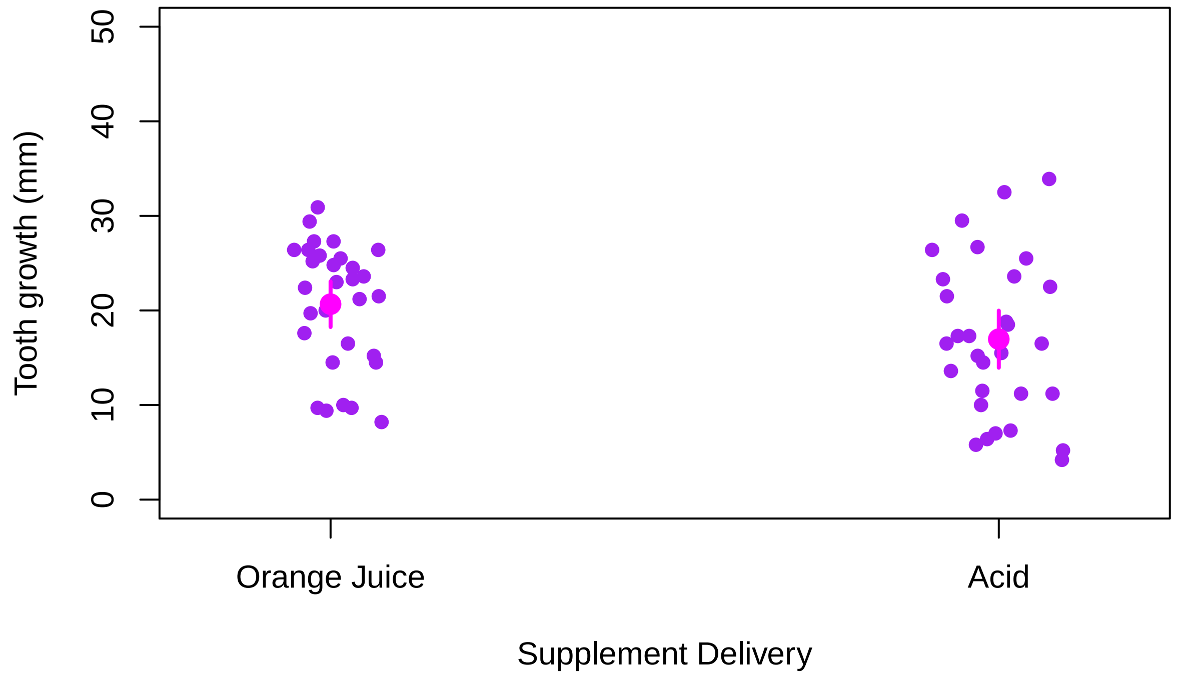
```
# get mean tooth length per group
mean.tooth <- aggregate(len~supp, data = ToothGrowth, FUN = mean)
mean.tooth
```

```
##   supp      len
## 1   OJ 20.66333
## 2   VC 16.96333
```

```
# get standard error
# first need to write our own function for SE since R doesn't have one
# don't change this code! just copy paste to your document
st.err <- function(x) {sd(x)/sqrt(length(x))}

# apply this new function to get the standard error in tooth length for each group
se.tooth <- aggregate(len~supp, data = ToothGrowth, FUN = st.err)
se.tooth
```

```
##   supp      len
## 1   OJ 1.206005
## 2   VC 1.509163
```

```
# now plot the raw data (this time we only have two groups)
# in this plot I changed the default names on the X axis to provide more information
{stripchart(len~supp, vertical = TRUE, method = "jitter",
           pch = 16, col = "purple", data = ToothGrowth,
           ylab = "Tooth growth (mm)", xlab = "Supplement Delivery",
           group.names = c("Orange Juice", "Acid"), ylim = c(0,50))
# add the means
points(mean.tooth$len ~ c(1:2), pch =16, cex =1.5, col = "magenta")

# add the standard errors (we want +/- 2 SE which corresponds well to significance test)
segments(x0 = c(1:2), x1 = c(1:2),
         y0 = mean.tooth$len - 2*se.tooth$len,
         y1 = mean.tooth$len + 2*se.tooth$len,
         lwd = 2, col = "magenta")
}
```



Stripchart Example 3: 2 way ANOVA data (interactions)

The tooth length data again, this time we want to know if there is an interaction between the dose they received and the method by which they received it. That is, does the effect of supplement dose on tooth growth DEPEND on how the supplement was given?

Important since dosage is a number, R will by default treat it like a continuous variable. With stripcharts, we want our x axis to have categories, so when plotting and in the anova, I will force R to treat this variable as a categorical (factor) by using the command `factor` around the column for dose.

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

```
# take a look at the ANOVA output
summary(aov(len~supp*factor(dose), ToothGrowth))
```

```
##                   Df Sum Sq Mean Sq F value   Pr(>F)    
## supp               1  205.4   205.4  15.572 0.000231 ***
## factor(dose)       2 2426.4  1213.2  92.000  < 2e-16 ***
## supp:factor(dose)  2  108.3    54.2   4.107 0.021860 *  
## Residuals         54  712.1    13.2                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

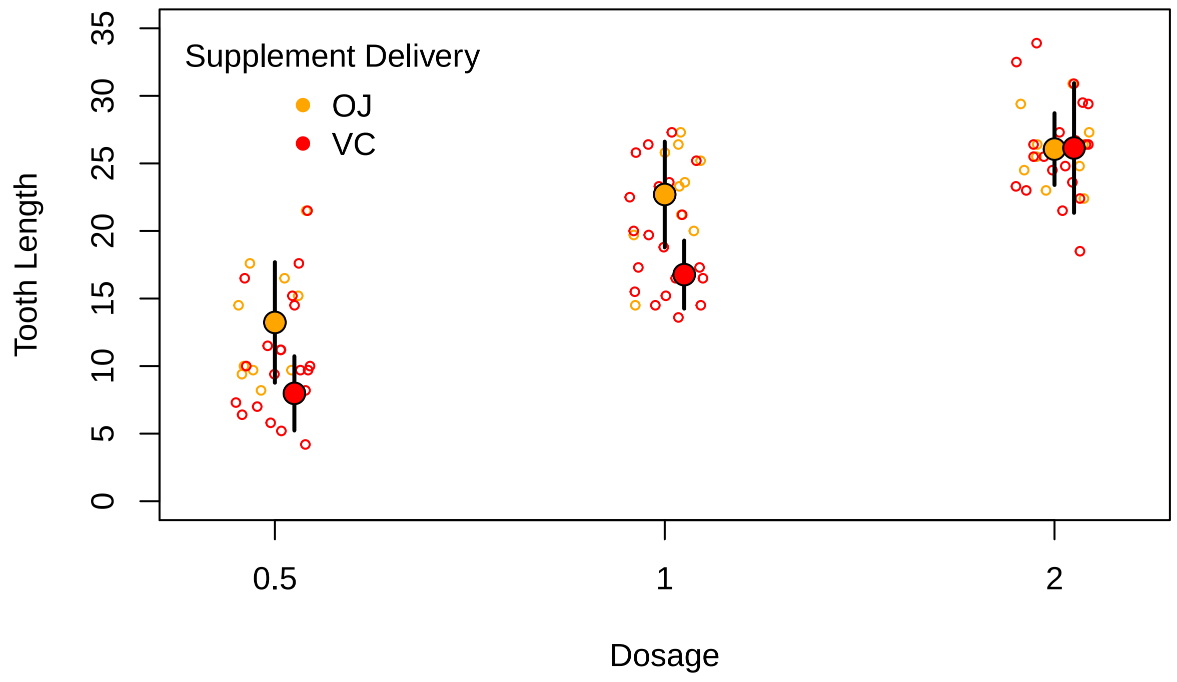
```
# here are the means/sds for each combination of supplement and dosage
means <- aggregate(len~supp*dose, data = ToothGrowth, FUN = mean)
sds <- aggregate(len~supp*dose, data = ToothGrowth, FUN = sd)
# for plotting, let's break this up into a set of means/sd per dosage for OJ
means.oj <- subset(means, supp == "OJ")
sd.oj <- subset(sds, supp == "OJ")
means.oj
```

```
##   supp dose   len
## 1   OJ  0.5 13.23
## 3   OJ  1.0 22.70
## 5   OJ  2.0 26.06
```

```
# and a set of means/sd per dosage for VC
means.vc <- subset(means, supp == "VC")
sd.vc <- subset(sds, supp == "VC")
means.vc
```

```
##   supp dose   len
## 2   VC  0.5  7.98
## 4   VC  1.0 16.77
## 6   VC  2.0 26.14
```

```
# set a color palette for plots
palette(c("orange", "red"))
# create a stripchart. First, we'll just do the OJ data
# use a subset data within the data = command to get one set of data
{stripchart(len~factor(dose), col ="orange",
           data = subset(ToothGrowth, supp =="OJ"), vertical = TRUE, method = "jitter",
           pch = 21, ylim = c(0,35), xlab = "Dosage", ylab = "Tooth Length", cex = 0.6)

# now we can overlay the vc data on the same plot
# the argument add = TRUE lets us add this stripchart on top of the old one
stripchart(len~(factor(dose)), col = "red", vertical = TRUE, method = "jitter", pch =21,
       data = subset(ToothGrowth, supp = "VC"), cex = 0.6, add = TRUE)

# add segments for SD bars
# this part is tricky!
# if we didn't break up the means by OJ and VC it would be hard to overlay these bars
# this way we will do the error bars for OJ first:
segments(x0=1:3, x1 = 1:3, y0 = means.oj$len-sd.oj$len,
         y1 = means.oj$len+sd.oj$len, col ="black", lwd= 2)
# then the error bars for VC
# we are adding a small value (0.05) to the x values
# this way our error bars won't overlap
segments(x0=c(1:3+.05), x1 = c(1:3+.05), y0 = means.vc$len-sd.vc$len,
         y1 = means.vc$len+sd.vc$len, col ="black", lwd = 2)

# add points for means
points(means.oj$len~c(1:3), pch = 21, bg = "orange", cex = 1.5)
# add a small number to the x axis value for VC
# this way the means won't overlap
points(means.vc$len~c(1:3+.05), pch = 21, bg = "red", cex = 1.5)


# add a legend
legend(x = .75, y = 35, pch = 16, col = c("orange", "red"),
       legend = c("OJ", "VC"), title = "Supplement Delivery", bty = "n")}
```



Scatter plots

Scatter plots are appropriate when:

- Predictor (independent) variable is CONTINUOUS
- Response (dependent) variable is CONTINUOUS

A good scatterplot should have:

- Well-labeled axes
- Raw data plotted as points (often with different colors if categories exist)
- A best fit line or multiple best fit lines overlaid (if slopes are significant)
- A legend if colors are used

Arguments to modify your scatterplot include:

- `ylim and xlim` to modify the limits of the x and y axes (usually the default is good but you can play with modifications)
- `pch` to set the type of point used (`pch = 16` makes closed circles)
- `col` to set the color of point
- `cex` to set the sizes of points (`cex = 1.2` makes points 20 percent larger than default)
- `ylab` and `xlab` to set the axis titles

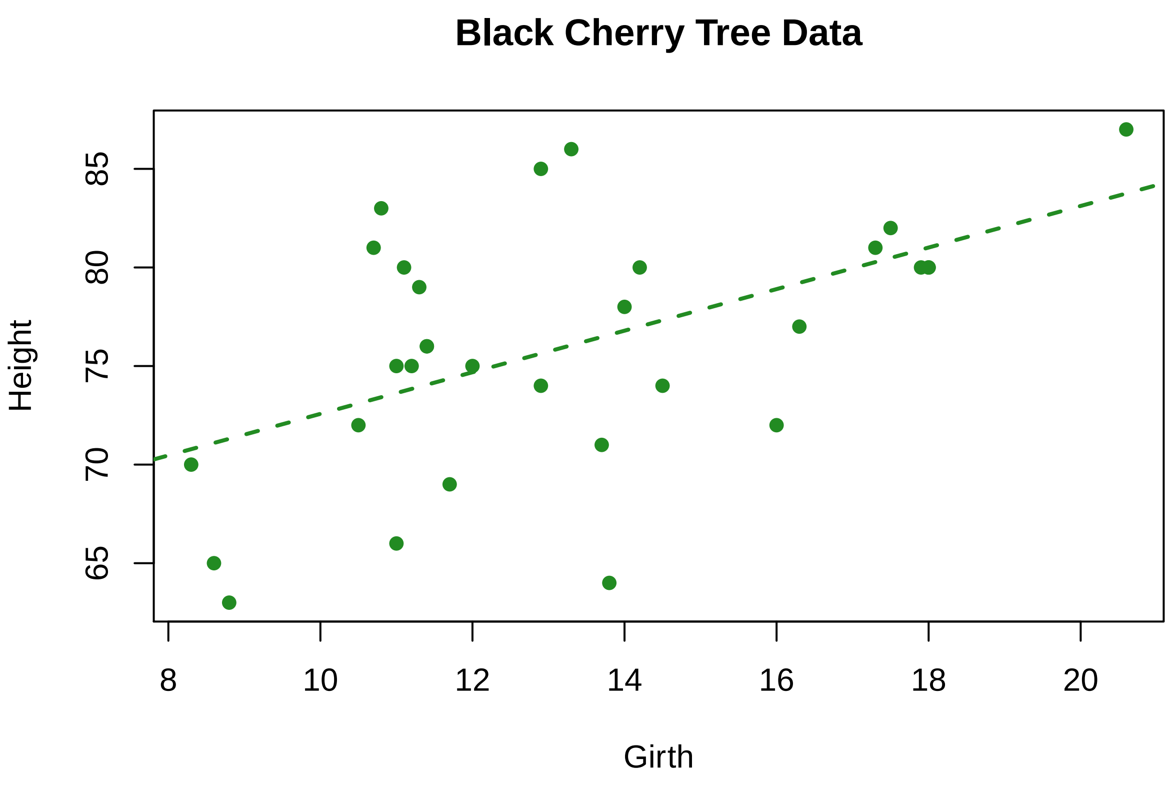
Scatterplot example 1: a simple regression

These data are tree girth vs. height, with a best fit line from linear regression overlaid. The response (y axis) variable always comes first.

```
# first get the best fit line (BEFORE plotting)
tree.lm <- lm(Height~Girth, data = trees)
summary(tree.lm)
```

```
##
## Call:
## lm(formula = Height ~ Girth, data = trees)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.5816  -2.7686   0.3163   2.4728   9.9456
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  62.0313     4.3833  14.152 1.49e-14 ***
## Girth         1.0544     0.3222   3.272  0.00276 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.538 on 29 degrees of freedom
## Multiple R-squared:  0.2697, Adjusted R-squared:  0.2445
## F-statistic: 10.71 on 1 and 29 DF,  p-value: 0.002758
```

```
# significant relationship between height and girth, so we want to plot the best fit line!

# plot the raw data
{plot(Height~Girth, data = trees,
     pch = 16, col = "forestgreen", main = "Black Cherry Tree Data")
# add in the best fit line (lty = 2 gives dashed rather than solid line)
# lwd makes a slightly thicker line
abline(tree.lm, lwd = 2, col = "forestgreen", lty = 2)
}
```



Scatterplot example 2: multiple categories and best fit lines with interactions

The dataset "Iris" has flower characteristics for a number of iris species. In this case we want to know if the length of the petal predicts length of the sepal, accounting for differences between species.

First, fit a linear model so we can get the best fit lines later on:

```r
iris.lm <- lm(Sepal.Length~Petal.Length * Species, iris)
```

Next we will plot the data, giving each species a different color.

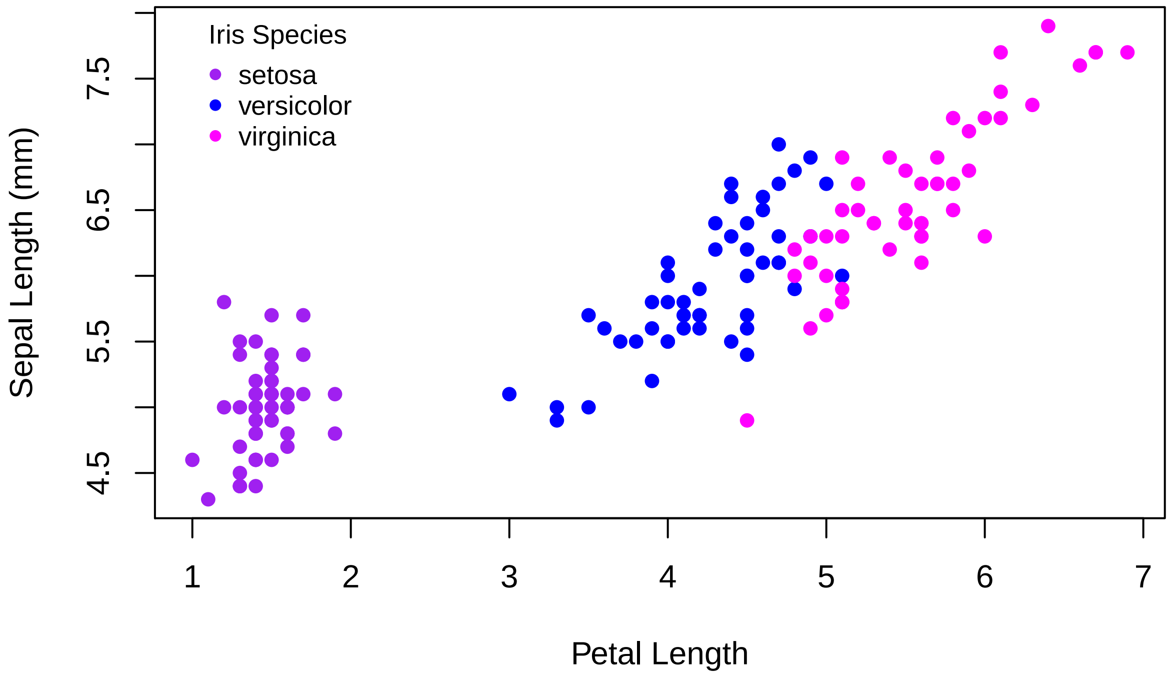
We will also add a legend. The legend arguments are:

`x =` and `y =` to set the location of the legend

`legend =` to set the labels for each entry in the legend. You can do this automatically, e.g. `legend = c("species 1", "species 2", "species 3")` or have R use your default factor names, e.g. `legend = levels(iris$Species)`

`pch =` to set the symbols for each entry in the legend (make sure they match your plot) `col =` sets the color of each symbol (again, make sure they match the order in your plot) `bty = n` removes the box around the legend. If you omit this argument, the legend will have a box around it. `title =` optional; makes an overall title for the legend `cex = 0.8` sets the size of the font and symbols to 80% of default.

```r
# first let's make a vector of the colors we want to use.
# We have 3 species, so we'll have 3 colors:
iris.colors <- c("purple", "blue", "magenta")

# make the default colors our new colors
palette(iris.colors)

# plot the data, with each Species getting a different color
# `col =` argument must have a categorical variable after it
{plot(Sepal.Length~Petal.Length, col = Species, iris, pch = 16,
     ylab = "Sepal Length (mm)", xlab = "Petal Length")

# add a legend
legend(x = 1, y = 8, legend = levels(iris$Species),
       pch = 16,  col = c("purple", "blue", "magenta"),
       bty = "n", title = "Iris Species", cex=.8)
}
```



```r
# re set the palette back to default
palette("default")
```

If we have multiple predictors in the models, we will have multiple best fit lines. There is not an automated way to plot each best fit line; instead, you will have to understand the output of your model and plot the best fit lines by hand.

Let's get the coefficients for the best fit lines out of the model

```r
# save the coefficients from your lm output!
# this will make your life easier when plotting
coefs.iris <- iris.lm$coefficients
```

Here, we have a significant interaction between species and sepal length. Therefore, we expect the slopes of the lines to differ between each species.

The species "setosa" is the reference level for the model (we can tell because it is not given a fixed effect term and it is the first alphabetically). Thus, the line equation for setosa is:

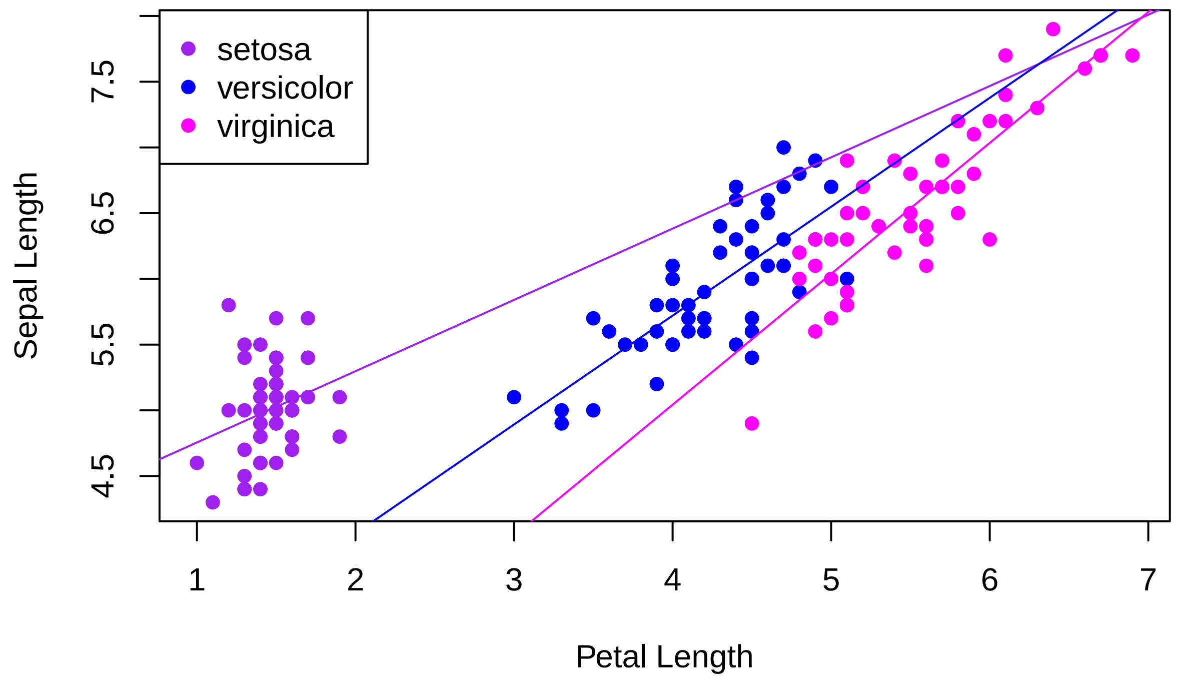
- Sepal.Length = 4.21 + 0.54 (Petal.Length)
- or, Sepal.Length = coefs[1] + coefs[2] x Petal.Length

The line equation for the versicolor species is:

- Sepal.Length = coefs[1] + coefs[2] x Petal.Length + coefs[3] + coefs[5] x Petal.Length
- or, rearranging: Sepal.Length = coefs[1] + coefs[3] + (coefs[2] + coefs[5]) x Petal.Length

We'll use these equations to add best fit lines to our plot:

```r
# set the colors again and make them the defaults
iris.colors <- c("purple", "blue", "magenta")
palette(iris.colors)

# plot the raw data
{plot (Sepal.Length~ Petal.Length, data = iris, col = Species, xlab = "Petal Length", ylab =
      "Sepal Length", pch = 16)

# setosa best fit line. a = intercept, b = slope
  abline(a = coefs.iris[1], b = coefs.iris[2], col = 1)

# versicolor best fit line
  abline(a = coefs.iris[1] + coefs.iris[3], b = coefs.iris[2]+coefs.iris[5], col = 2)

# virginica best fit line
  abline(a = coefs.iris[1] + coefs.iris[4], b = coefs.iris[2]+coefs.iris[6], col = 3)

# legend
  legend("topleft", col=c(1:3), legend = levels(iris$Species), pch = 16)
}
```



```r
# set the palette back to its default colors
palette("default")
```

If the lines don't appear to match your data well, you likely have swapped coefficients; so make sure to carefully look at your plot.

Scatterplot 3: adding confidence interval around best fit line

You should always include measures of uncertainty in your best fit line. Remember, the slope and intercepts are *estimates* and you may have varying degrees of confidence in those estimates. Thus, plotting confidence intervals around your best fit lines can convey that uncertainty to your reader. Be very cautious of plots that do not show uncertainty!

For this tutorial, I will show the confidence interval around the setosa line only, but in practice you'd need to do this for every species, since each one has a separate best fit line. In other words, every best fit line you show in a plot should have a display of uncertainty.

To get the best fit line, we need predicted (fitted) values from the model. In the `newdata` argument of the predict function, I will give it a range of petal length values (or x-axis values) for which I want to plot the best fit line. Using the actual range of x values in your data is a good way to do this and avoid extrapolating.

```
# STEP 1: get a vector of X values (values of petal length) over the range of the data
x.vals <- seq(from = min(iris$Petal.Length), to = max(iris$Petal.Length), by = 0.1)
# this goes from the minimum to maximum petal length in increments of 0.1
x.vals[1:10]
```

```
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9
```

```
# STEP 2: add species name for the line you want to draw
species.name <- rep("setosa", length(x.vals))

# STEP 3: create a dataframe that contains both predictors in the model:
# predictors are sepal length and species
df <- data.frame(Petal.Length = x.vals, Species = species.name)
# this is essentially "dummy" data to get a best fit line
head(df)
```

```
##   Petal.Length Species
## 1          1.0  setosa
## 2          1.1  setosa
## 3          1.2  setosa
## 4          1.3  setosa
## 5          1.4  setosa
## 6          1.5  setosa
```

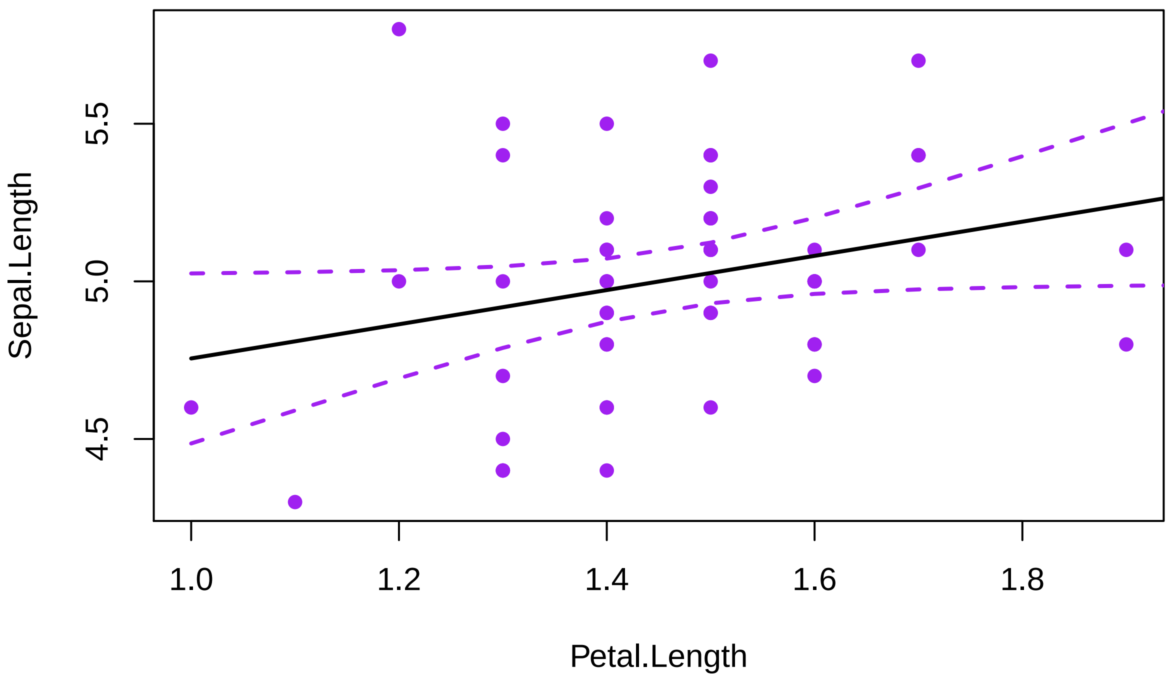
```
# STEP 4: predict y values (or petal length) for each row of your dataframe
# predict function will also give  confidence intervals
# This function uses your linear model to make the predictions
CI <- as.data.frame(predict(iris.lm, newdata = df, interval = "confidence"))
head(CI)
```

```
##        fit      lwr      upr
## 1 4.755461 4.485792 5.025129
## 2 4.809690 4.590460 5.028920
## 3 4.863919 4.692491 5.035348
## 4 4.918149 4.788924 5.047373
## 5 4.972378 4.872401 5.072355
## 6 5.026607 4.930289 5.122925
```

The `newdata` argument of the `predict` function needs to contain a dataframe that has *all of your predictors* from the linear model. So, if your linear model included petal length, species, AND sepal width as predictors, your dataframe (df) would need a column corresponding to sepal width as well. A good rule of thumb would be to use the average sepal width for the species you are using.

The CI dataframe gives the fitted (predicted) value for each entry in the dataframe, as well as the lower and upper confidence intervals. We'll use these for plotting.

```
{plot(Sepal.Length~Petal.Length, data = subset(iris, Species =="setosa"),
      col="purple", pch = 16)
lines(x = df$Petal.Length, y = CI$fit, lwd = 2)
lines(x = df$Petal.Length, y = CI$lwr, lwd=2, lty="dashed", col="purple")
lines(x = df$Petal.Length, y = CI$upr, lwd=2, lty="dashed", col="purple")
}
```



Scatterplot 4: Interaction plots

As you saw in previous examples from lecture and lab, the `abline` function can add a simple best fit line to a scatterplot. However, when we have multiple regression with interactions, we may want to show multiple best fit lines. The following example shows the relationship between summer precipitation and C3 abundance, at different levels of annual temperature.

This plotting will use a package called `interactions`. To install the package, delete the #'s and run the following code. Then comment the lines out with a # or delete the code chunk. You only have to run these lines of code ONCE on your computer.

```
#install.packages("interactions")
#install.packages("RColorBrewer")
```

This example works with the dataset from Lab 9 (plant abundance). We'll make a linear model using the interaction between summer precip (JJAMAP) and temperature (MAT) to predict C3 abundance.

```
plantabun <- read.csv("PlantAbund.csv")
lmplant <- lm(C3~JJAMAP*MAT, plantabun)
```

We will plot the relationship between longitude and C4, with a best fit line for high latitude and low latitude places.

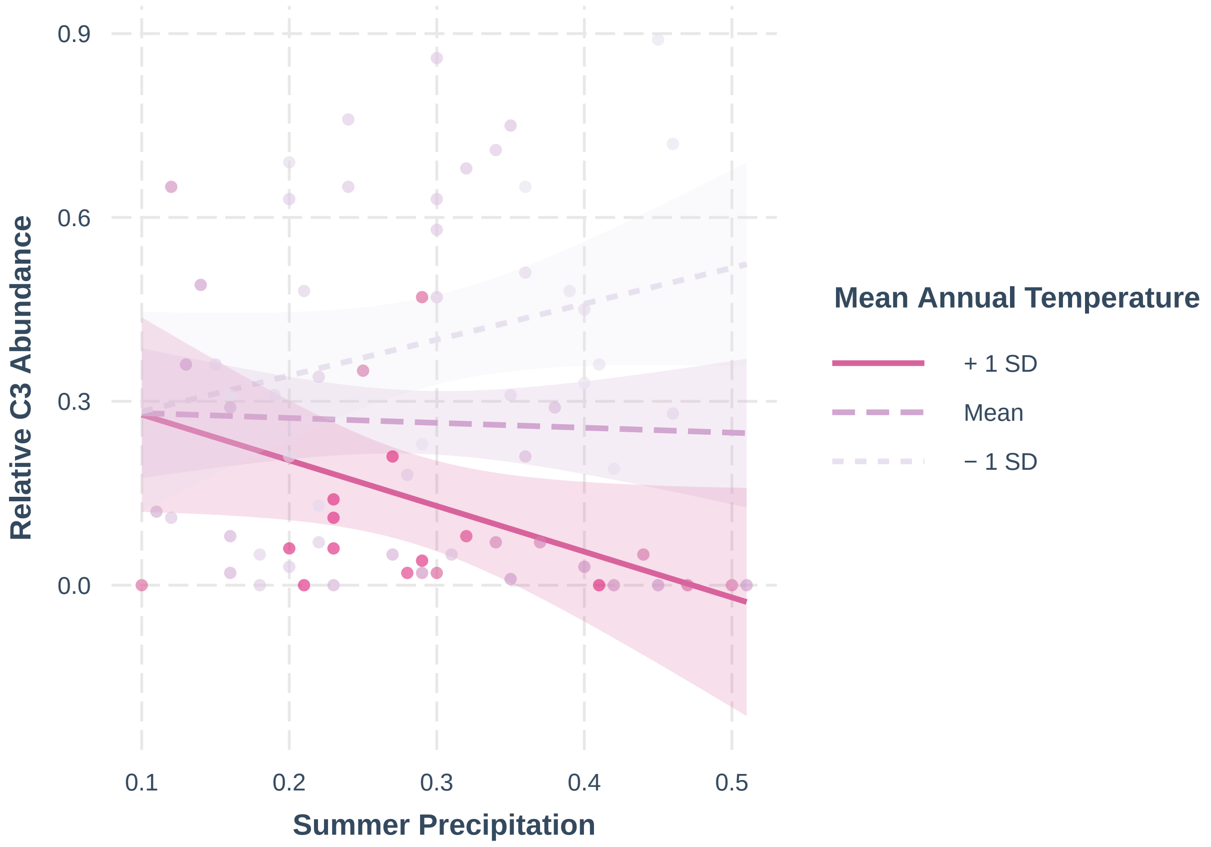
The command `interact_plot` in the `interactions` library is used. The arguments are:

- `pred=` gives the predictor variable we want on the X axis
- `modx=` gives the other predictor variable in the model that we want to vary
- `plot.points=TRUE` adds the raw data from the original dataset
- `interval = TRUE` adds a confidence interval around the best fit lines

- `int.width = 0.95` makes the confidence interval equal to a 95% CI
- `colors = "PuRd"` makes the colors vary from purple to red. See ?RColorBrewer for more types of sequential palettes.
- `x.label` X axis label
- `y.label` Y axis label
- `legend.main` Legend title

For more customization, see `?interact_plot`

```
# load the libraries
library(interactions)
library(RColorBrewer)

# make the plot
interact_plot(lmplant, pred = "JJAMAP", modx = "MAT", plot.points = TRUE,
              interval = TRUE, int.width = .95,
              colors = "PuRd",x.label = "Summer Precipitation",
              y.label = "Relative C3 Abundance", legend.main = "Mean Annual Temperature")
```



To look at the plot click the little rectangle with an arrow that appears above the upper-right corner of the plot. This will open the plot in a new window.

The plot tells us a few important things:

1) The pink dotted line shows the relationship between temperature and C3 relative abundance at mean precipitation. We can see that C3 abundance doesn't really change with increasing precipitation.

2) The red line shows the relationship between precipitation and C3 relative abundance at sites that are +1 SD away from the mean in temperature ("higher temperature" sites). At hot sites, precipitation decreases C3 abundances.

3) The purple dotted line shows the relationship between Precipitation and C3 at cold sites (-1 SD in temperature). Precipitation increases C3 plants at cold sites.

Scatterplot 5: Poisson regression

This dataset shows the relationship between mammal species richness and island area.

```
island <- read.csv("MammalIsland.csv")
head(island)
```

```
##   X MammalRich Area Latitude
## 1 1         10   40    42.99
## 2 2          0    3    41.25
## 3 3        107 2707    41.18
## 4 4          8   61    41.42
## 5 5         62 1190    41.27
## 6 6         85 1350    41.08
```

```
mammal.glm <- glm(MammalRich~log(Area)+Latitude,
                  data = island, family = "poisson")
summary(mammal.glm)
```

```
## 
## Call:
## glm(formula = MammalRich ~ log(Area) + Latitude, family = "poisson",
##     data = island)
## 
## Deviance Residuals: 
##      Min        1Q    Median        3Q       Max  
## -1.50117  -0.70097   0.07677   0.51721   1.51270  
## 
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -0.09650    0.58638  -0.165    0.869    
## log(Area)    0.70789    0.01697  41.724   <2e-16 ***
## Latitude    -0.01703    0.01462  -1.164    0.244    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for poisson family taken to be 1)
## 
##     Null deviance: 4268.322  on 21  degrees of freedom
## Residual deviance:   16.785  on 19  degrees of freedom
## AIC: 126.1
## 
## Number of Fisher Scoring iterations: 4
```

We want to plot `area` on the x-axis and `mammal richness` on the y axis. First, let's get a range of areas to use in our plot. To avoid extrapolating we'll go from the minimum area in our dataset to the maximum area in our dataset, in increments of 10.

```
AreaRange <- seq(min(island$Area), max(island$Area), by = 10)
AreaRange[1:10]
```

```
##  [1]  3 13 23 33 43 53 63 73 83 93
```

Now let's predict the mean species richness on islands that have a particular area. The first thing we need to do is make a fake dataframe that has the data we want to predict with (for plotting). We want to predict across a range of areas, but since our model includes another predictor (Latitude) we need to give the dataframe some value for Latitude. Let's do the mean.

```r
newdata.mammal <- data.frame("Area" = AreaRange,
                             "Latitude" = mean(island$Latitude))
```

To get the predicted value we will use our fitted glm model and our fake dataframe. The function we'll use is `predict`. In this case we add two extra arguments:

- `type = "response"` means that the predicted value will be on the scale of species richness instead of log-species richness.
- `se.fit = TRUE` means that we will get standard errors around our estimate.

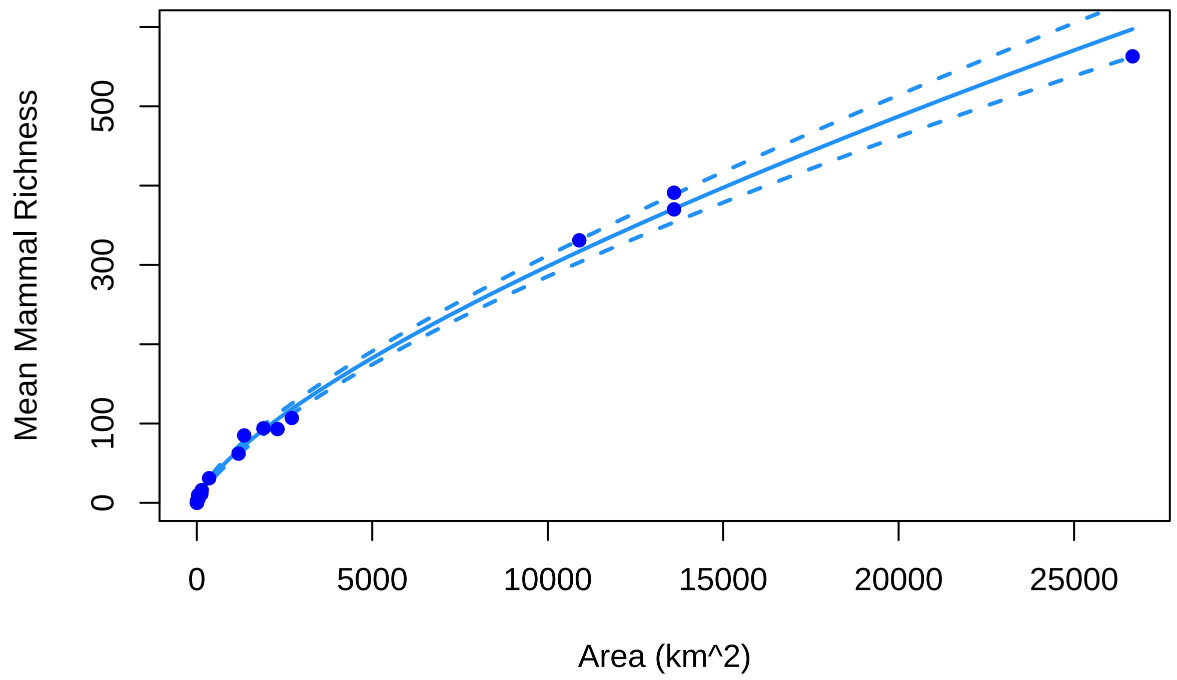
```r
predicted.mammal <- predict(mammal.glm, newdata = newdata.mammal,
                            type = "link", se.fit = TRUE)
str(predicted.mammal)
```

```
## List of 3
##  $ fit           : Named num [1:2667] -0.0441 0.9939 1.3977 1.6533 1.8407 ...
##   ..- attr(*, "names")= chr [1:2667] "1" "2" "3" "4" ...
##  $ se.fit        : Named num [1:2667] 0.1352 0.1107 0.1012 0.0952 0.0909 ...
##   ..- attr(*, "names")= chr [1:2667] "1" "2" "3" "4" ...
##  $ residual.scale: num 1
```

The output from the predict function has two datasets: `predicted.mammal$fit` has the predicted values, and `predicted.mammal$se` has the standard error estimates. Note that both of these are on the scale of the link function, meaning they are log(species) rather than species. We will use `exp` to un-do the log (back-transform back into raw counts).

Now we will plot the data, overlaying lower and upper confidence intervals (+ and - two standard errors.). `type = "l"` makes a line instead of points.

```r
{
# plot predicted species richness at each value of area
plot(exp(predicted.mammal$fit)~AreaRange,
     type = "l", col = "dodgerblue", lwd = 2,
     xlab = "Area (km^2)", ylab = "Mean Mammal Richness")
# add a lower confidence interval
lines(exp(predicted.mammal$fit - 2*predicted.mammal$se.fit) ~ AreaRange,
      col = "dodgerblue", lwd = 2, lty = 2)
# add an upper confidence interval
lines(exp(predicted.mammal$fit + 2*predicted.mammal$se.fit) ~ AreaRange,
      col = "dodgerblue", lwd = 2, lty = 2)
# overlay raw data
points(MammalRich~Area, island, pch = 16, col = "blue")
}
```



Scatterplot 6: Binomial regression

These data come from a dataset investigating whether seeding clouds with Silver Iodine increases the chance of precipitation. Predictors include: cloud seeding (1 or 0) and cloud cover % (0 to 100). Response variable was whether it snowed (1) or not (0).

```
clouds <- read.csv("CloudSeeding.csv")
head(clouds)
```

```
##   X Seeded CloudCover Snowed
## 1 1      1         97      1
## 2 2      0         66      0
## 3 3      1         50      1
## 4 4      1         84      1
## 5 5      0         77      1
## 6 6      0         69      0
```

```
cloud.glm <- glm(Snowed~Seeded + CloudCover, data = clouds, family = "binomial")
summary(cloud.glm)
```

```
##
## Call:
## glm(formula = Snowed ~ Seeded + CloudCover, family = "binomial",
##     data = clouds)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7220  -0.9954  -0.5514   1.0924   1.8189
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.07433    0.86782  -2.390   0.0168 *
## Seeded       1.13440    0.59224   1.915   0.0554 .
## CloudCover   0.02433    0.01230   1.978   0.0479 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 76.082  on 54  degrees of freedom
## Residual deviance: 68.520  on 52  degrees of freedom
## AIC: 74.52
##
## Number of Fisher Scoring iterations: 4
```

We want to plot how cloud cover impacts chance of precipitation, with a different line for seeded clouds and not seeded clouds.

First let's get a range of cloud cover over which we will plot probability of snow. Cloud cover can vary from 0 to 100%, and to get a smooth line we'll do it in increments of 1.

```
CloudCoverRange <- seq(0,100,1)
```

Now let's predict the chance of precipitation for un-seeded clouds:

```
Pred.Precip.UnSeed <- predict(cloud.glm, newdata = data.frame(
  "Seeded"= 0, "CloudCover" = CloudCoverRange),
  type = "link", se.fit = TRUE
)
```

Next we'll predict the chance of precipitation for seeded clouds, changing the "Seeded" column to read 1 instead of 0:

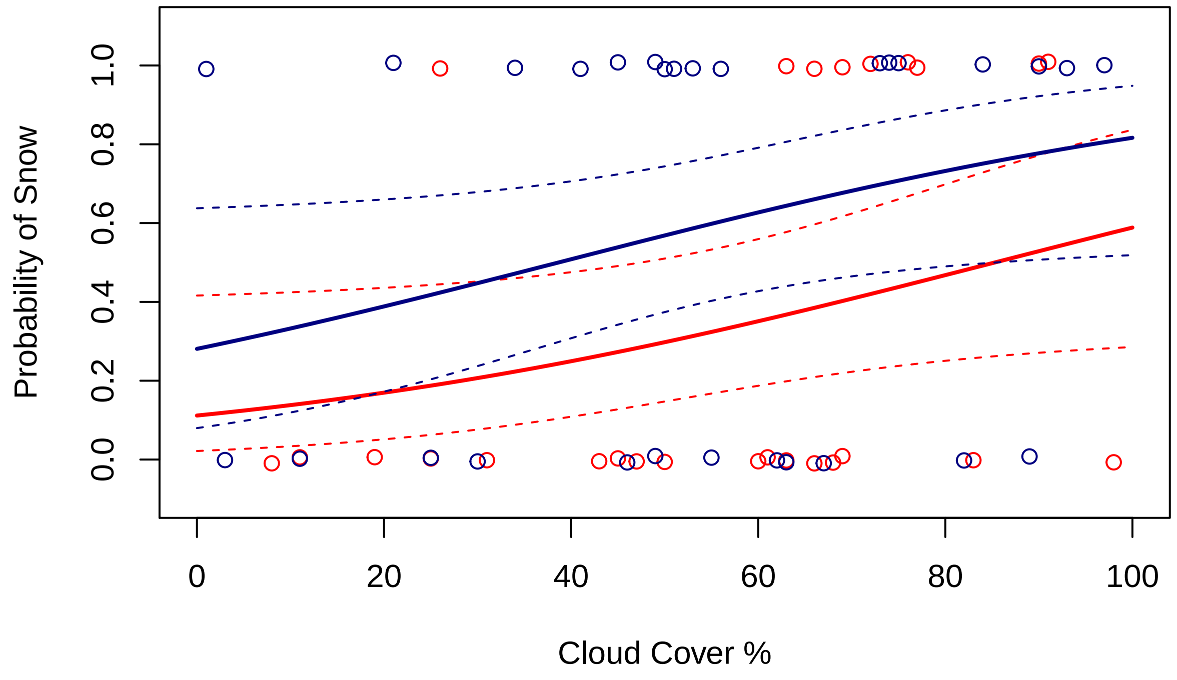
```
Pred.Precip.Seed <- predict(cloud.glm, newdata = data.frame(
  "Seeded"= 1, "CloudCover" = CloudCoverRange),
  type = "link", se.fit = TRUE
)
```

Now make an inverse logit function to back-transform the predictions into probabilities

```
invlogit <- function(x) {exp(x)/(1+exp(x))}
```

Let's plot the data:

```
{
  # probability of snow at each cloud cover, for un-seeded clouds
  plot(invlogit(Pred.Precip.UnSeed$fit) ~ CloudCoverRange, lwd = 2,
       type = "l", ylim = c(-.1, 1.1), ylab = "Probability of Snow",
       xlab = "Cloud Cover %", col = "red")
  # lower CI
  lines(invlogit(Pred.Precip.UnSeed$fit-2*Pred.Precip.UnSeed$se.fit) ~ CloudCoverRange,
        lty = 2, col = "red")
  # upper CI
    lines(invlogit(Pred.Precip.UnSeed$fit+2*Pred.Precip.UnSeed$se.fit) ~ CloudCoverRange,
          lty = 2, col = "red")

# probability of snow at each cloud cover, for seeded clouds
  points(invlogit(Pred.Precip.Seed$fit) ~ CloudCoverRange, lwd = 2,
         type = "l", col = "navyblue")
  # lower CI
  lines(invlogit(Pred.Precip.Seed$fit-2*Pred.Precip.Seed$se.fit) ~ CloudCoverRange,
        lty = 2, col = "navyblue")
  # upper CI
    lines(invlogit(Pred.Precip.Seed$fit+2*Pred.Precip.Seed$se.fit) ~ CloudCoverRange,
          lty = 2, col = "navyblue")
  # add raw data for unseeded clouds
    points(jitter(Snowed,.05)~CloudCover, data = subset(clouds, Seeded == 0), col = "red")
    points(jitter(Snowed, .05)~CloudCover, data = subset(clouds, Seeded==1), col = "navyblue")
}
```



Mosaic plots

Mosaic plots are a good way of plotting data when:

- Your predictor (independent) variable is CATEGORICAL
- Your response (dependent) variable is CATEGORICAL

The basic steps for a mosaic plot are:

1) Create a table giving the categories of the response variable and the counts in each category
2) Plot the points, and overlay the means and error bars

Important arguments for modifying the default mosaic plot are:

- `t(TableName)` swaps the columns and rows in case you want to switch your x and y axis (try with and without!)
- `border = "white"` changes the border in between blocks
- `sub = "Name of Independent Variable"` changes the sub-title on the x axis (independent variable)
- `ylab = "y label axis"` changes the title of the dependent variable
- `col = c("red", "orange")` changes the colors of each block

Mosaic plot example 1 (two groups)

Experiment tested whether birds were more likely to get malaria if they were stressed (by having an egg removed from their nest). Predictor variable is the treatment (control or egg removed) and response is whether they got malaria or not.

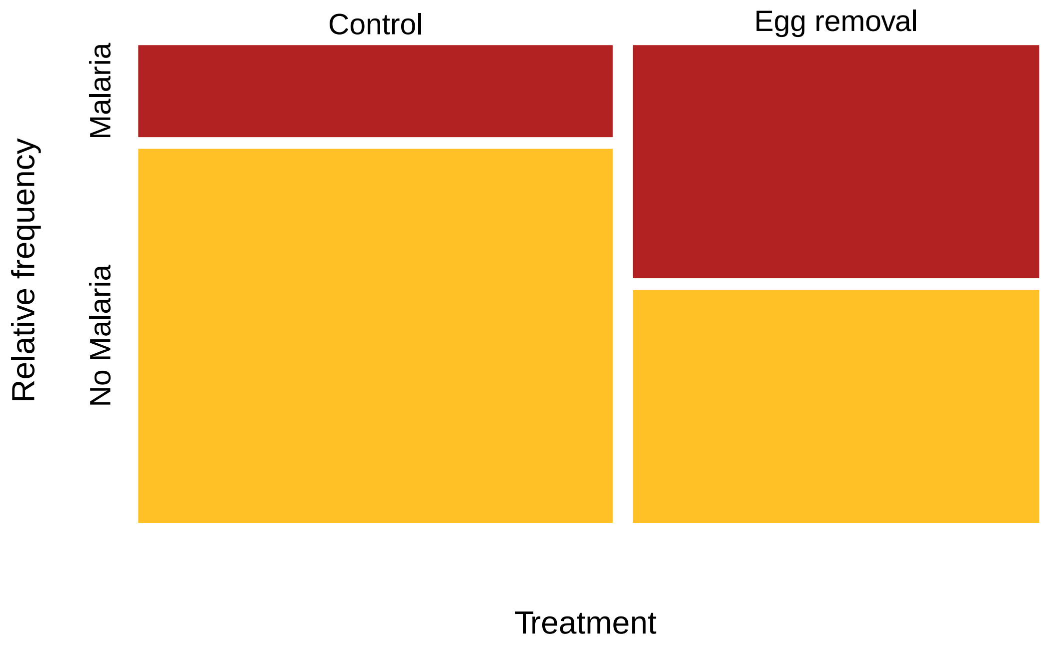
```
# read in the data from the web
Birds <-read.csv("http://whitlockschluter.zoology.ubc.ca/wp-content/data/chapter02/chap02e3aBirdMalaria
head(Birds)
```

```
##   bird treatment response
## 1    1   Control  Malaria
## 2    2   Control  Malaria
## 3    3   Control  Malaria
## 4    4   Control  Malaria
## 5    5   Control  Malaria
## 6    6   Control  Malaria
```

```
# create a table
# gives the number of birds who got malaria or
# didn't get malaria in the treatment and control group
BirdTable <- table(Birds$response, Birds$treatment)
BirdTable
```

```
##
##               Control Egg removal
##   Malaria           7          15
##   No Malaria       28          15
```

```
# create a mosaic plot
mosaicplot(t(BirdTable), col = c("firebrick", "goldenrod1"),
           border = "white", sub = "Treatment",
           ylab = "Relative frequency", cex.axis = 0.9, main = "")
```



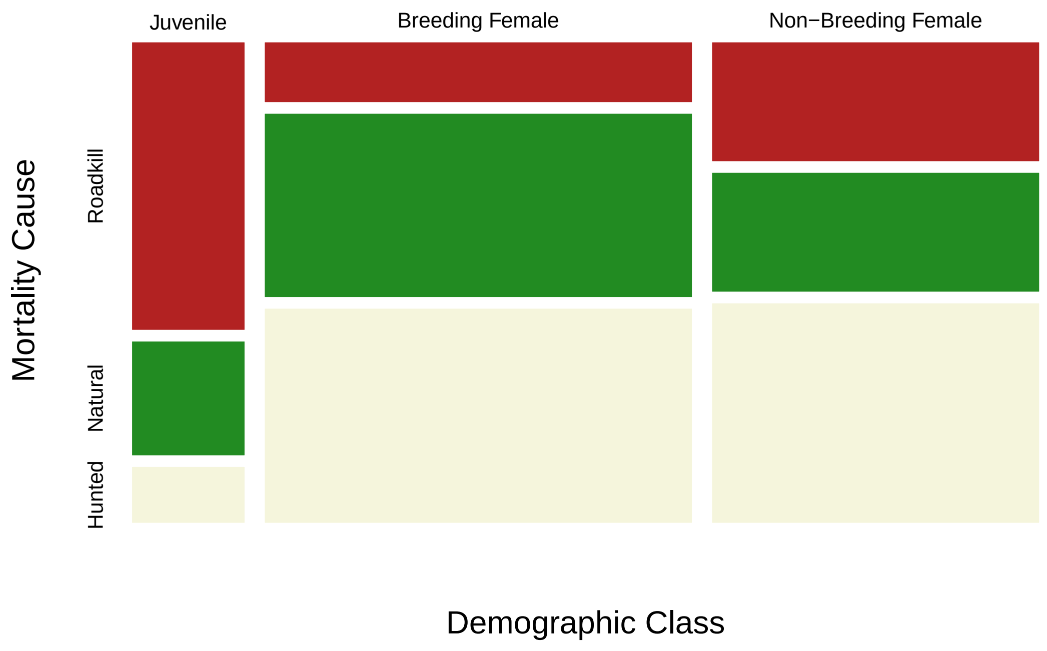
Mosaic plot example 2 (three groups)

In this example, we will input the data by hand. The independent variable is the demographic classes of bears (juvenile, breeding female, and non-breeding female) in a study. The dependent variable is the number of bears who died from each type of mortality (roadkill, natural causes, or hunting). This kind of data would be appropriately analyzed with a contingency test.

```r
juvenile.deaths <- c(5, 2, 1)
breed.fem.deaths <- c(4,12,14)
nb.fem.deaths <- c(6,6,11)
# combine into one data frame
bear.data <- data.frame(juvenile.deaths, breed.fem.deaths, nb.fem.deaths)
rownames(bear.data) <- c("Roadkill", "Natural", "Hunted")
# change column names so they are nicer for plot
colnames(bear.data) <- c("Juvenile", "Breeding Female", "Non-Breeding Female")

# view dataset
bear.data
```

```
##          Juvenile Breeding Female Non-Breeding Female
## Roadkill        5               4                   6
## Natural         2              12                   6
## Hunted          1              14                  11
```

```r
# create a mosaic plot
mosaicplot(t(bear.data), col = c("firebrick", "forestgreen", "beige"),
           border = "white", sub = "Demographic Class",
           ylab = "Mortality Cause", cex.axis = 0.7, main = "")
```


Juvenile
Breeding Female
Non−Breeding Female
Mortality Cause
Roadkill
Natural
Hunted
Demographic Class