Model visualizations using predictions

PSTAT126

Lab 5

Objectives

This lab covers the model visualization technique introduced in lecture in greater depth. Doing so involves learning some slightly more advanced `ggplot` functionality with respect to layering and aesthetics. The following topics are emphasized:

- Generating prediction grids with `data_grid()`
- Plotting paths with `geom_path()`
- Adding uncertainty bands with `geom_ribbon()`
- Back-transformation

The basic idea behind these model visualizations is to plot a path through predictions computed along a sequence of predictor values. In fact, that's what `geom_smooth()` does for you 'under the hood' – so you've already been doing this without realizing it!

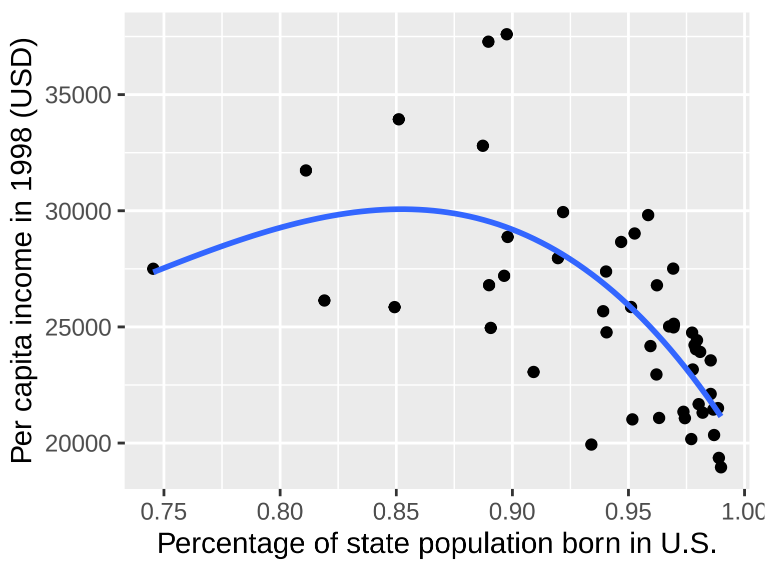
However, `geom_smooth()` only works in a limited range of settings. To develop more sophisticated graphics, we'll need to understand and then mimic the process.

The first section covers the basic calculations in the one-predictor setting; the second section extends the idea to the multi-predictor setting. Both sections work with familiar data. You will likely only cover the first section during your lab session; if so, please work through the second section on your own.

Basics: visualizations with one predictor

This segment works with the `eco` data from `faraway` on 1998 per capita income for each U.S. state and the proportion of residents of each state born in the U.S. as of the 1990 census. By now, this should be a familiar plot:

```r
p <- ggplot(eco, aes(x = usborn, y = income)) +
  geom_point() +
  labs(x = 'Percentage of state population born in U.S.',
       y = 'Per capita income in 1998 (USD)')

p + geom_smooth(method = 'lm', formula = 'y ~ poly(x, 3)', se = F)
```



We're going to replicate it by explicitly computing the path shown.

Prediction grids with `data_grid()`

The first step is to generate a sequence of values of `usborn` (the predictor shown on the horizontal axis). `data_grid()` provides a way of doing this flexibly and easily.

The most basic functionality, shown below, assigns the unique values of `usborn` to a column named `usborn`.

```r
# create a data grid with unique values from the usborn column of the eco dataset.
# by default, it sorts the unique values in ascending order
data_grid(eco, usborn = usborn) %>% head()
```

```
## # A tibble: 6 x 1
##   usborn
##    <dbl>
## 1  0.745
## 2  0.811
## 3  0.819
## 4  0.849
## 5  0.851
## 6  0.887
```

Take a moment to verify that the column above contains these values:

```r
# equivalent results
unique(eco$usborn) %>% sort()
```

```
##  [1] 0.74541 0.81116 0.81911 0.84932 0.85111 0.88732 0.88972 0.89000 0.89072
## [10] 0.89650 0.89761 0.89801 0.90918 0.91964 0.92187 0.93406 0.93914 0.94037
## [19] 0.94060 0.94688 0.95113 0.95172 0.95268 0.95850 0.95951 0.96208 0.96227
## [28] 0.96322 0.96751 0.96928 0.96944 0.96953 0.96958 0.97371 0.97432 0.97708
## [37] 0.97743 0.97770 0.97848 0.97908 0.97952 0.98026 0.98089 0.98197 0.98541
## [46] 0.98544 0.98656 0.98688 0.98856 0.98895 0.98987
```

```
data_grid(eco, usborn = usborn) %>% pull(usborn)
```

```
##  [1] 0.74541 0.81116 0.81911 0.84932 0.85111 0.88732 0.88972 0.89000 0.89072
## [10] 0.89650 0.89761 0.89801 0.90918 0.91964 0.92187 0.93406 0.93914 0.94037
## [19] 0.94060 0.94688 0.95113 0.95172 0.95268 0.95850 0.95951 0.96208 0.96227
## [28] 0.96322 0.96751 0.96928 0.96944 0.96953 0.96958 0.97371 0.97432 0.97708
## [37] 0.97743 0.97770 0.97848 0.97908 0.97952 0.98026 0.98089 0.98197 0.98541
## [46] 0.98544 0.98656 0.98688 0.98856 0.98895 0.98987
```

It's not strictly necessary, but often helpful to include a model object to ensure that the output of `data_grid()` contains all the columns needed to call `predict()` with the results and the desired model. For us, this will be the model we wish to visualize. Run the cell below to verify that the results are the same as above.

```
# model to visualize
fit_eco <- lm(income ~ poly(usborn, 3), data = eco)

# prediction grid
eco %>%
  data_grid(usborn = usborn,
            .model = fit_eco) %>%
  head()
```

The function `add_predictions()` appends model predictions as a new column to any data frame (with the columns needed to compute the predictions – hence using the `.model = ...` argument with `data_grid()`):

```
# append predictions
eco %>%
  data_grid(usborn = usborn, .model = fit_eco) %>%
  add_predictions(model = fit_eco) %>%
  head()
```

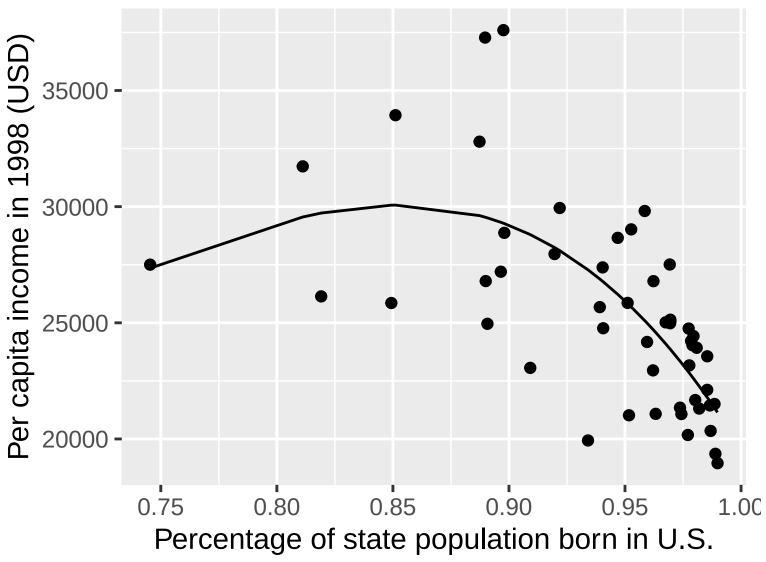
```
## # A tibble: 6 x 2
##   usborn   pred
##    <dbl>  <dbl>
## 1  0.745 27353.
## 2  0.811 29555.
## 3  0.819 29725.
## 4  0.849 30065.
## 5  0.851 30068.
## 6  0.887 29615.
```

Plotting paths

The foregoing data frame is all that's needed to plot a path through the predictions – values of `usborn` and values of the response. Store it and construct the plot:

```
# store prediction grid
pred_df_eco <- eco %>%
  data_grid(usborn = usborn, .model = fit_eco) %>%
  add_predictions(model = fit_eco)
```

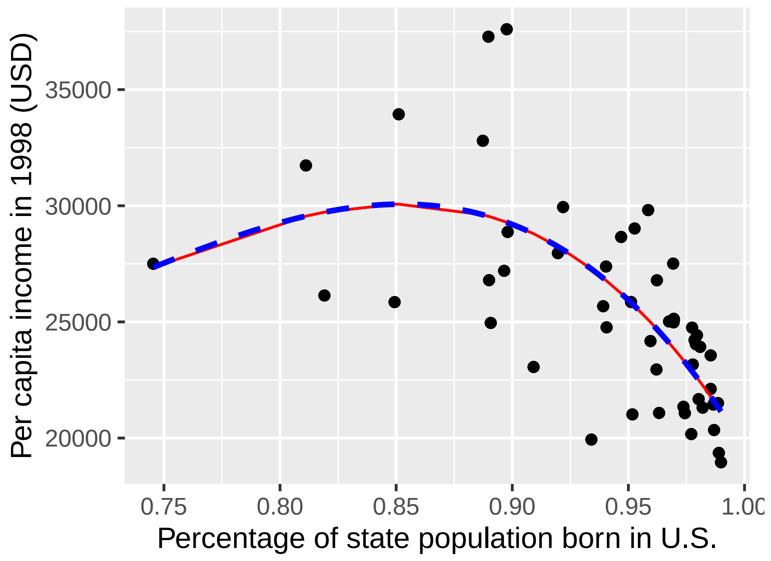
```
# plot path
p + geom_path(aes(y = pred), data = pred_df_eco)
```



Your turn Take a moment to compare the result with `geom_smooth()`. Add a layer to the plot below with the analogous `geom_smooth()` visualization. If need be, you can copy the command from the plot at the very top; but try to do it from scratch if you can. (*Hint*: play with the aesthetics `color`, `linetype`, and `alpha` (transparency) and use one or more of them to make it easier to visually distinguish the lines.)

```
# compare with geom_smooth()
```

```
# Solution
# plot with both geom_path() and geom_smooth()
p + geom_path(aes(y = pred), data = pred_df_eco,
              color = "red", linetype = "solid") +
  geom_smooth(aes(y = income), data = eco,
              method = 'lm', formula = 'y ~ poly(x, 3)', se = F,
              color = "blue", linetype = "dashed")
```



Notice how the `geom_path()` layer is a little jagged compared with the `geom_smooth()` layer. That's because for the path, the prediction grid comprises only the unique values *in the data set*; so especially where the data are sparse, one can see the path comprises straight line segments.

This can be improved by generating predictions along an evenly-spaced sequence, rather than for each unique predictor value in the dataset. `seq_range()` will do that in a less verbose way than the usual `seq()`:

```
# these are equivalent
seq_range(eco$usborn, 5)
```

```
## [1] 0.745410 0.806525 0.867640 0.928755 0.989870
```

```
seq(from = min(eco$usborn),
    to = max(eco$usborn),
    length = 5)
```

```
## [1] 0.745410 0.806525 0.867640 0.928755 0.989870
```

To construct a prediction grid using a sequence, simply substitute a call to `seq_range()` for the variable name in the arguments to `data_grid()`:

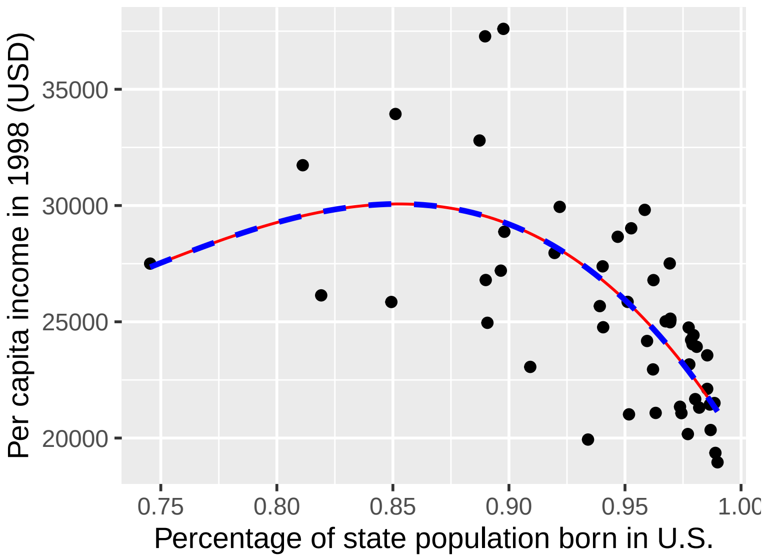
```
# note for the curious: try removing the 'usborn = ' part
# and check the column names in the prediction grid
eco %>%
  data_grid(usborn = seq_range(usborn, 100),
            .model = fit_eco) %>%
  head()
```

```
## # A tibble: 6 x 1
##   usborn
##    <dbl>
## 1  0.745
## 2  0.748
## 3  0.750
## 4  0.753
## 5  0.755
## 6  0.758
```

Your turn Now repeat the process: add predictions, store the result, plot the path, and overlay the `geom_smooth()` layer to compare to your last plot.

```
# prediction grid

# path compared with geom_smooth
```

```
# Solution
# generate a prediction data frame with an evenly-spaced usborn sequence
even_pred_df_eco <- eco %>%
  data_grid(usborn = seq_range(usborn, 100), .model = fit_eco) %>%
  add_predictions(model = fit_eco)

# plot with both geom_path() and geom_smooth()
p + geom_path(aes(y = pred), data = even_pred_df_eco,
              color = "red", linetype = "solid") +
  geom_smooth(aes(y = income), data = eco,
              method = 'lm', formula = 'y ~ poly(x, 3)',
              se = F, color = "blue", linetype = "dashed")
```



Adding confidence (or prediction) bands

To visualize uncertainty, one can compute upper and lower confidence (or prediction) limits for each prediction on the grid, and add a shaded area between those bounds to the plot.

As a starting point, the upper and lower limits are needed. These, if you recall, can be computed using `predict.lm()`:

```
# compute confidence limits
predict(fit_eco,
        newdata = pred_df_eco,
        interval = 'confidence',
        level = 0.95) %>%
  head()
```

```
##        fit      lwr      upr
## 1 27352.88 21172.31 33533.44
## 2 29555.47 26342.46 32768.47
## 3 29725.25 26618.78 32831.72
## 4 30064.88 27581.00 32548.77
## 5 30067.61 27628.60 32506.63
## 6 29614.57 27955.49 31273.65
```

So simply binding these three columns to the prediction grid will do the trick:

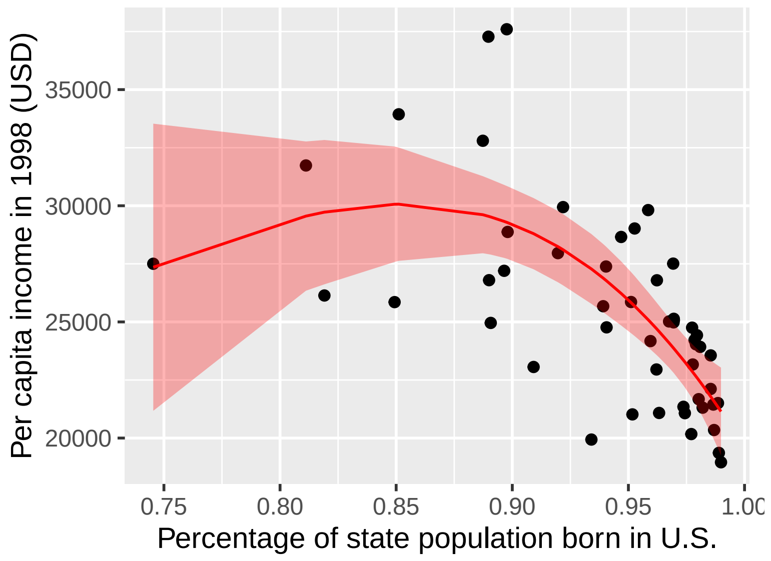
```
# add confidence limits
pred_df_eco_ci <- pred_df_eco %>%
  cbind(ci = predict(fit_eco,
                     newdata = pred_df_eco,
                     interval = 'confidence',
                     level = 0.95))

pred_df_eco_ci %>% head()
```

```
##    usborn     pred   ci.fit   ci.lwr   ci.upr
## 1 0.74541 27352.88 27352.88 21172.31 33533.44
## 2 0.81116 29555.47 29555.47 26342.46 32768.47
## 3 0.81911 29725.25 29725.25 26618.78 32831.72
## 4 0.84932 30064.88 30064.88 27581.00 32548.77
## 5 0.85111 30067.61 30067.61 27628.60 32506.63
## 6 0.88732 29614.57 29614.57 27955.49 31273.65
```

Notice that by specifying `ci = ...`, the string `ci` is appended as a prefix to the column names from the output of `predict()` when they are added to the data frame.

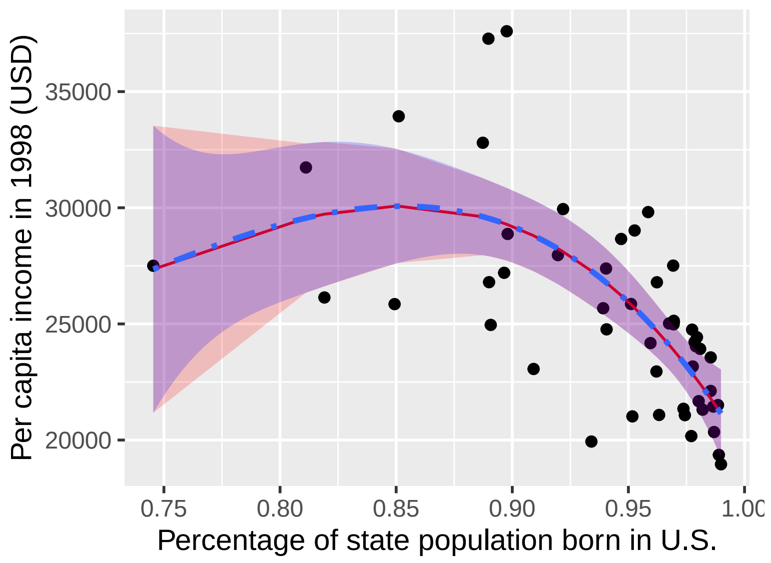
Shading the area between the two limits is accomplished by `geom_ribbon()`. This requires some new aesthetics: `ymin = ...` will be the lower limit, and `ymax = ...` will be the upper limit. (Additionally, notice that the fill color and transparency (`alpha`) are adjusted away from the default settings, but *outside* of the `aes()` call.)

```
# add uncertainty bands
p + geom_path(aes(y = pred),
              data = pred_df_eco,
              color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red',
              alpha = 0.3)
```



And this matches *exactly* the standard error returned with `geom_smooth(..., se = T, ...)`:

```
p + geom_path(aes(y = pred),
              data = pred_df_eco,
              color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red',
              alpha = 0.2) +
  geom_smooth(method = 'lm',
              formula = 'y ~ poly(x, 3)',
              se = T,
              fill = 'blue',
              alpha = 0.2,
              linetype = 'dotdash')
```



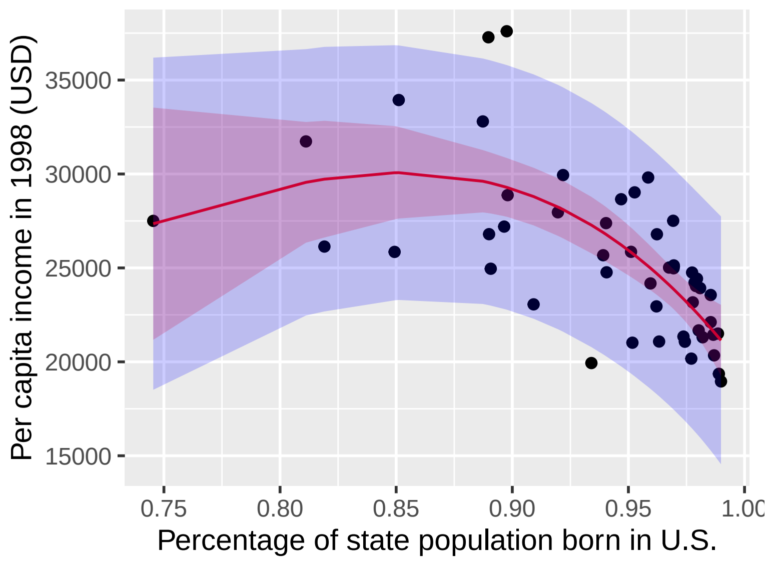
So by default, `geom_smooth()` plots a 95% confidence interval.

Your turn Modify the example above to show 95% *prediction* limits along the path. As a starting point, you'll need to compute prediction intervals along the prediction grid and append those to `pred_df_eco`; if you can, use the prefix `pi` (instead of `ci`).

(If you want a slight challenge, try showing both the prediction and the confidence limits in distinct colors by combining: the base and scatter layers (`p`); the path layer (`geom_path()`) in red; the 95% confidence limits (`geom_ribbon()`) in red; and the 95% prediction limits (`geom_ribbon()`) in blue.)

```
# compute prediction limits and append to grid


# plot
```

```
# Compute 95% prediction intervals
pred_df_eco_pi <- pred_df_eco %>%
  cbind(pi = predict(fit_eco, newdata = pred_df_eco, interval = 'prediction', level = 0.95))

# Create the plot
p + geom_path(aes(y = pred), data = pred_df_eco, color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr, ymax = ci.upr, y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red', alpha = 0.2) +
  geom_ribbon(aes(ymin = pi.lwr, ymax = pi.upr, y = pi.fit),
              data = pred_df_eco_pi,
              fill = 'blue', alpha = 0.2)
```



Advanced model visualizations: going beyond `geom_smooth()`

Why not always just use `geom_smooth()`? Well, it has limitations – it's great for single-predictor settings, but can't handle multiple variables or transformations especially well. Here we'll look at how to do something similar for both of these settings.

The solution file for Practice 4 proposed the following model for the `leafburn` data:

```
# model from hw3
fit_leafburn <- lm(log(burntime) ~ log(nitrogen) + log(chlorine) + log(potassium),
                   data = leafburn)
```

Our goal will be to visualize the relationship between `burntime` and `nitrogen` on the original scale with uncertainty. We'll start with generating appropriate prediction grids.

Prediction grids with multiple variables

The `data_grid()` function works more or less the same no matter how many variables one has; if more variables are input to the call, it will generate a grid of all *combinations* of the named variables:

```
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5)) %>%
  head()
```

```
## # A tibble: 6 x 2
##   nitrogen potassium
##      <dbl>     <dbl>
## 1     2.22      2.76
## 2     2.22      3.72
## 3     2.22      4.68
## 4     2.22      5.64
## 5     2.22      6.6
## 6     2.27      2.76
```

Now the `.model` argument is important, because the output above has no `chlorine` column. So if one tries to add predictions, R will throw an error:

```
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5)) %>%
  add_predictions(model = fit_leafburn)
```

```
## Error in eval(predvars, data, env): object 'chlorine' not found
```

The problem is that there's no `chlorine` column in the ouptut from `data_grid()`, but the model `fit_leafburn` expects one. Adding the `.model` argument ensures that the output has a chlorine column.

```
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5),
            .model = fit_leafburn) %>%
  head()
```

```
## # A tibble: 6 x 3
##   nitrogen potassium chlorine
##      <dbl>     <dbl>    <dbl>
## 1     2.22      2.76     0.64
## 2     2.22      3.72     0.64
## 3     2.22      4.68     0.64
## 4     2.22      5.64     0.64
## 5     2.22      6.6      0.64
## 6     2.27      2.76     0.64
```

This grid has a sequence of 50 evenly-spaced values of `nitrogen` spanning the range in the data, repeated for each of 5 evenly-spaced values of `potassium` spanning the range in the data. In other words, all possible combinations of a 50-value sequence in one variable and a 5-value sequence in another. For `chlorine`, the median is repeated.

Your turn

i. Construct a prediction grid generated from a sequence of 100 nitrogen values and 3 potassium values. Print the first few rows.

```
# solution
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 100),
            potassium = seq_range(potassium, 3),
            .model = fit_leafburn)%>%
  head()
```

```
## # A tibble: 6 x 3
##   nitrogen potassium chlorine
##      <dbl>     <dbl>    <dbl>
## 1     2.22      2.76     0.64
## 2     2.22      4.68     0.64
## 3     2.22      6.6      0.64
## 4     2.24      2.76     0.64
## 5     2.24      4.68     0.64
## 6     2.24      6.6      0.64
```

ii. Construct a prediction grid generated from the same sequence as in (i), but fill in the 75th percentile for `chlorine` (instead of the median). (*Hint*: use `quantile(...)` in a similar way to how `seq_range()` is used in the call for `data_grid()`.) Print the first few rows.

```
# solution
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 100),
            potassium = seq_range(potassium, 3),
            chlorine = quantile(chlorine, 0.75),
            .model = fit_leafburn) %>%
  head()
```

```
## # A tibble: 6 x 3
##   nitrogen potassium chlorine
##      <dbl>     <dbl>    <dbl>
## 1     2.22      2.76     1.07
## 2     2.22      4.68     1.07
## 3     2.22      6.6      1.07
## 4     2.24      2.76     1.07
## 5     2.24      4.68     1.07
## 6     2.24      6.6      1.07
```

Adding predictions works exactly the same as before:

```
# store prediction grid
pred_df_leaf <- leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5),
            .model = fit_leafburn) %>%
  add_predictions(model = fit_leafburn)
```

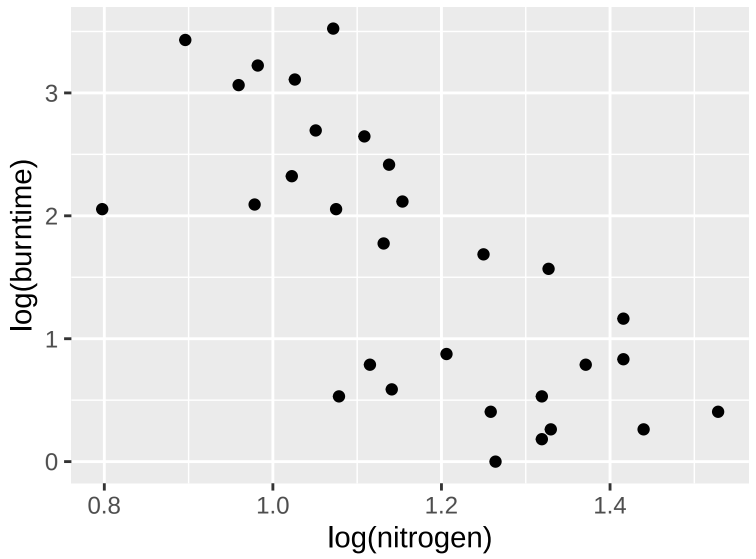
And ditto confidence limits:

```
pred_df_leaf_ci <- pred_df_leaf %>%
  cbind(ci = predict(fit_leafburn,
                     newdata = pred_df_leaf,
                     interval = 'confidence',
                     level = 0.9))
```

Visualizing model structure in multiple variables

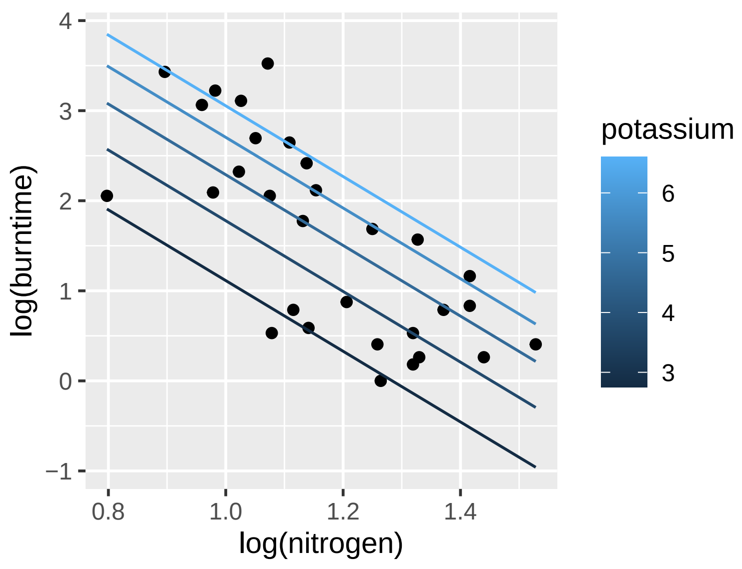
The most dense sequence in the prediction grid is in `nitrogen`; so the steps here display this relationship primarily. (If you were more interested in burn time and potassium, it would make more sense to have a dense sequence in potassium and a sparse sequence in nitrogen or chlorine and start with the burntime-potassium scatterplot.) So the steps below superimpose the model structure visualizations on this plot:

```r
# base scatterplot
g <- ggplot(leafburn,
            aes(x = log(nitrogen),
                y = log(burntime))) +
  geom_point()

g
```



Plot one line – the sequence of `nitrogen` values and the predictions – for each unique value of `potassium`. The basic layering pattern is the same as before, but notice there is a `group = ...` argument to `geom_path()`. Try removing that and see what happens (it was mentioned in lecture).

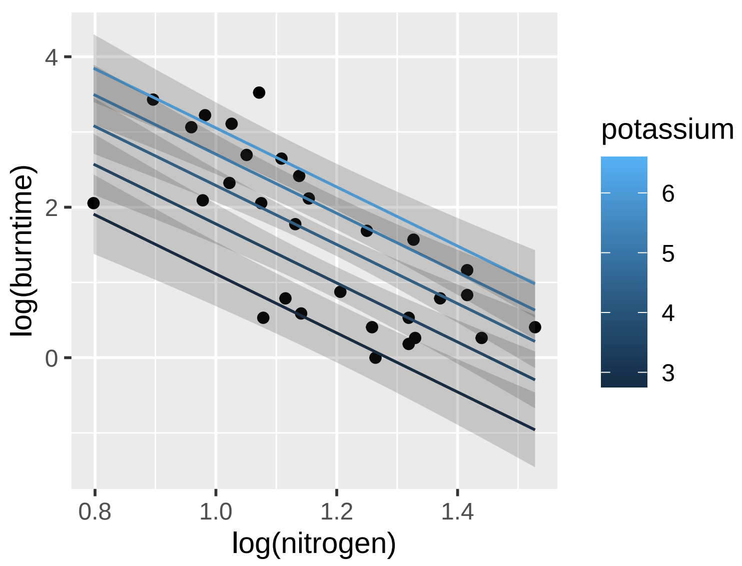
```r
# notice the group argument; try without
g + geom_path(aes(y = pred,
                  color = potassium,
                  group = potassium),
              data = pred_df_leaf_ci)
```



```
# use "group" to create separate lines for each unique value of potassium
```

Your turn Add confidence limits to each line by following the example in the previous section. All the calculations have been done for you – `pred_df_leaf_ci` contains the upper and lower limits – and the `geom_ribbon()` layer should look *almost* exactly the same; just add the proper `group` argument.

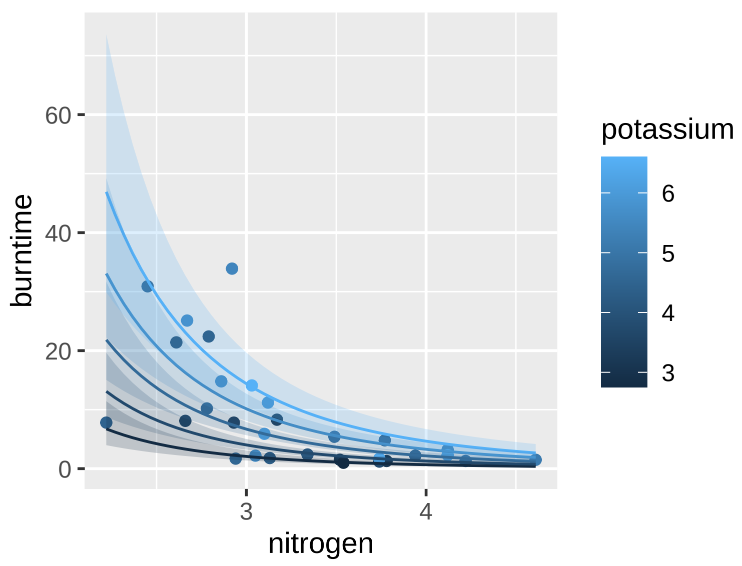
```
# solution
g + geom_path(aes(y = pred,
                  color = potassium,
                  group = potassium),
              data = pred_df_leaf_ci) +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  group = potassium,
                  y = ci.fit),
              data = pred_df_leaf_ci,
              alpha = 0.2)
```



Back-transformation

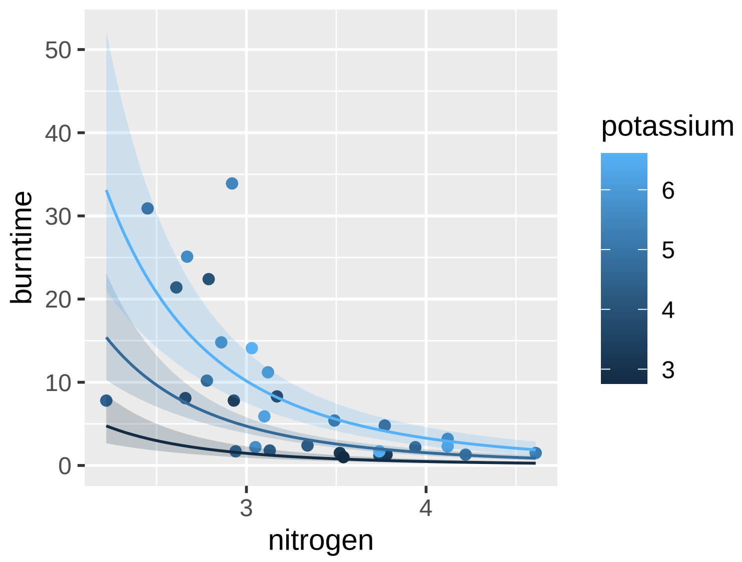
You may recall that interpretation of model coefficients was tricky because of the log transformations of all variables. Well, visualization on the original scale is easy – just drop the `log()` functions from the variables in the base scatter layer and exponentiate the predictions!

```
# visualization on original scale
ggplot(leafburn,
       aes(x = nitrogen,
           y = burntime)) + # note change here - no log
  geom_point(aes(color = potassium)) +
  geom_path(aes(y = exp(pred), # note change here - exp
                color = potassium,
                group = potassium),
            data = pred_df_leaf_ci) +
  geom_ribbon(aes(ymin = exp(ci.lwr), # note change here - exp
                  ymax = exp(ci.upr), # note change here - exp
                  y = NULL,
                  group = potassium,
                  fill = potassium),
              data = pred_df_leaf_ci,
              alpha = 0.2)
```



Your turn Remember that the plots above show the relationships between burntime and nitrogen for several values of potassium *given that chlorine is fixed at the median value in the dataset.* If the value of chlorine is changed, the relationships will shift. Try this: tinker with the code chunk below and change the value of chlorine to see the effect on the plot.

```
# solution

plot_leafburn <- function(chlorine_value) {
  # Create the prediction grid with the specified chlorine value
  pred_grid_leafburn <- leafburn %>%
    data_grid(nitrogen = seq_range(nitrogen, 100),
              potassium = seq_range(potassium, 3),
              chlorine = chlorine_value,
              .model = fit_leafburn) %>%
    add_predictions(fit_leafburn)

  pred_df_leaf_ci <- pred_grid_leafburn %>%
    cbind(ci = predict(fit_leafburn,
                       newdata = pred_grid_leafburn,
                       interval = 'confidence',
                       level = 0.9))

  # Create the plot
  ggplot(leafburn,
         aes(x = nitrogen,
             y = burntime)) +
    geom_point(aes(color = potassium)) +
    geom_path(aes(y = exp(pred),
                  color = potassium,
                  group = potassium),
              data = pred_grid_leafburn) +
    geom_ribbon(aes(ymin = exp(ci.lwr),
                    ymax = exp(ci.upr),
                    y = NULL,
                    group = potassium,
                    fill = potassium),
                data = pred_df_leaf_ci,
                alpha = 0.2)
}

# Call the function with different chlorine values
plot_leafburn(1)
```



Code appendix

```
# default code chunk options
knitr::opts_chunk$set(echo = T,
                      results = 'markup',
                      message = F,
                      warning = F,
                      fig.width = 4,
                      fig.height = 3,
                      fig.align = 'center')

# load packages
library(faraway)
library(tidyverse)
library(modelr)
p <- ggplot(eco, aes(x = usborn, y = income)) +
  geom_point() +
  labs(x = 'Percentage of state population born in U.S.',
       y = 'Per capita income in 1998 (USD)')

p + geom_smooth(method = 'lm', formula = 'y ~ poly(x, 3)', se = F)
# create a data grid with unique values from the usborn column of the eco dataset.
# by default, it sorts the unique values in ascending order
```

```r
data_grid(eco, usborn = usborn) %>% head()
# equivalent results
unique(eco$usborn) %>% sort()
data_grid(eco, usborn = usborn) %>% pull(usborn)
# model to visualize
fit_eco <- lm(income ~ poly(usborn, 3), data = eco)

# prediction grid
eco %>%
  data_grid(usborn = usborn,
            .model = fit_eco) %>%
  head()
# append predictions
eco %>%
  data_grid(usborn = usborn, .model = fit_eco) %>%
  add_predictions(model = fit_eco) %>%
  head()
# store prediction grid
pred_df_eco <- eco %>%
  data_grid(usborn = usborn, .model = fit_eco) %>%
  add_predictions(model = fit_eco)

# plot path
p + geom_path(aes(y = pred), data = pred_df_eco)
# compare with geom_smooth()

# Solution
# plot with both geom_path() and geom_smooth()
p + geom_path(aes(y = pred), data = pred_df_eco,
              color = "red", linetype = "solid") +
  geom_smooth(aes(y = income), data = eco,
              method = 'lm', formula = 'y ~ poly(x, 3)', se = F,
              color = "blue", linetype = "dashed")

# these are equivalent
seq_range(eco$usborn, 5)
seq(from = min(eco$usborn),
    to = max(eco$usborn),
    length = 5)
# note for the curious: try removing the 'usborn = ' part
# and check the column names in the prediction grid
eco %>%
  data_grid(usborn = seq_range(usborn, 100),
            .model = fit_eco) %>%
  head()
# prediction grid

# path compared with geom_smooth

# Solution
# generate a prediction data frame with an evenly-spaced usborn sequence
even_pred_df_eco <- eco %>%
  data_grid(usborn = seq_range(usborn, 100), .model = fit_eco) %>%
  add_predictions(model = fit_eco)

# plot with both geom_path() and geom_smooth()
p + geom_path(aes(y = pred), data = even_pred_df_eco,
              color = "red", linetype = "solid") +
  geom_smooth(aes(y = income), data = eco,
              method = 'lm', formula = 'y ~ poly(x, 3)',
              se = F, color = "blue", linetype = "dashed")
# compute confidence limits
predict(fit_eco,
        newdata = pred_df_eco,
        interval = 'confidence',
        level = 0.95) %>%
  head()
# add confidence limits
pred_df_eco_ci <- pred_df_eco %>%
  cbind(ci = predict(fit_eco,
                     newdata = pred_df_eco,
                     interval = 'confidence',
                     level = 0.95))

pred_df_eco_ci %>% head()
# add uncertainty bands
p + geom_path(aes(y = pred),
              data = pred_df_eco,
              color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red',
              alpha = 0.3)
p + geom_path(aes(y = pred),
              data = pred_df_eco,
              color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red',
              alpha = 0.2) +
  geom_smooth(method = 'lm',
              formula = 'y ~ poly(x, 3)',
              se = T,
              fill = 'blue',
              alpha = 0.2,
              linetype = 'dotdash')
# compute prediction limits and append to grid


# plot

# Compute 95% prediction intervals
```

```r
pred_df_eco_pi <- pred_df_eco %>%
  cbind(pi = predict(fit_eco, newdata = pred_df_eco, interval = 'prediction', level = 0.95))

# Create the plot
p + geom_path(aes(y = pred), data = pred_df_eco, color = 'red') +
  geom_ribbon(aes(ymin = ci.lwr, ymax = ci.upr, y = ci.fit),
              data = pred_df_eco_ci,
              fill = 'red', alpha = 0.2) +
  geom_ribbon(aes(ymin = pi.lwr, ymax = pi.upr, y = pi.fit),
              data = pred_df_eco_pi,
              fill = 'blue', alpha = 0.2)
# model from hw3
fit_leafburn <- lm(log(burntime) ~ log(nitrogen) + log(chlorine) + log(potassium),
                   data = leafburn)
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5)) %>%
  head()
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5)) %>%
  add_predictions(model = fit_leafburn)
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5),
            .model = fit_leafburn) %>%
  head()
# solution
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 100),
            potassium = seq_range(potassium, 3),
            .model = fit_leafburn)%>%
  head()
# solution
leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 100),
            potassium = seq_range(potassium, 3),
            chlorine = quantile(chlorine, 0.75),
            .model = fit_leafburn) %>%
  head()
# store prediction grid
pred_df_leaf <- leafburn %>%
  data_grid(nitrogen = seq_range(nitrogen, 50),
            potassium = seq_range(potassium, 5),
            .model = fit_leafburn) %>%
  add_predictions(model = fit_leafburn)
pred_df_leaf_ci <- pred_df_leaf %>%
  cbind(ci = predict(fit_leafburn,
                     newdata = pred_df_leaf,
                     interval = 'confidence',
                     level = 0.9))
# base scatterplot
g <- ggplot(leafburn,
            aes(x = log(nitrogen),
                y = log(burntime))) +
  geom_point()

g
# notice the group argument; try without
g + geom_path(aes(y = pred,
                  color = potassium,
                  group = potassium),
              data = pred_df_leaf_ci)
# use "group" to create separate lines for each unique value of potassium
# solution
g + geom_path(aes(y = pred,
                  color = potassium,
                  group = potassium),
              data = pred_df_leaf_ci) +
  geom_ribbon(aes(ymin = ci.lwr,
                  ymax = ci.upr,
                  group = potassium,
                  y = ci.fit),
              data = pred_df_leaf_ci,
              alpha = 0.2)
# visualization on original scale
ggplot(leafburn,
       aes(x = nitrogen,
           y = burntime)) + # note change here - no log
  geom_point(aes(color = potassium)) +
  geom_path(aes(y = exp(pred), # note change here - exp
                color = potassium,
                group = potassium),
            data = pred_df_leaf_ci) +
  geom_ribbon(aes(ymin = exp(ci.lwr), # note change here - exp
                  ymax = exp(ci.upr), # note change here - exp
                  y = NULL,
                  group = potassium,
                  fill = potassium),
              data = pred_df_leaf_ci,
              alpha = 0.2)
# solution

plot_leafburn <- function(chlorine_value) {
  # Create the prediction grid with the specified chlorine value
  pred_grid_leafburn <- leafburn %>%
    data_grid(nitrogen = seq_range(nitrogen, 100),
              potassium = seq_range(potassium, 3),
              chlorine = chlorine_value,
              .model = fit_leafburn) %>%
    add_predictions(fit_leafburn)

  pred_df_leaf_ci <- pred_grid_leafburn %>%
    cbind(ci = predict(fit_leafburn,
                       newdata = pred_grid_leafburn,
                       interval = 'confidence',
                       level = 0.9))

  # Create the plot
  ggplot(leafburn,
         aes(x = nitrogen,
             y = burntime)) +
    geom_point(aes(color = potassium)) +
    geom_path(aes(y = exp(pred),
                  color = potassium,
                  group = potassium),
              data = pred_grid_leafburn) +
    geom_ribbon(aes(ymin = exp(ci.lwr),
                    ymax = exp(ci.upr),
                    y = NULL,
                    group = potassium,
                    fill = potassium),
                data = pred_df_leaf_ci,
                alpha = 0.2)
}

# Call the function with different chlorine values
plot_leafburn(1)
```