# Regression workshop 2

Zane

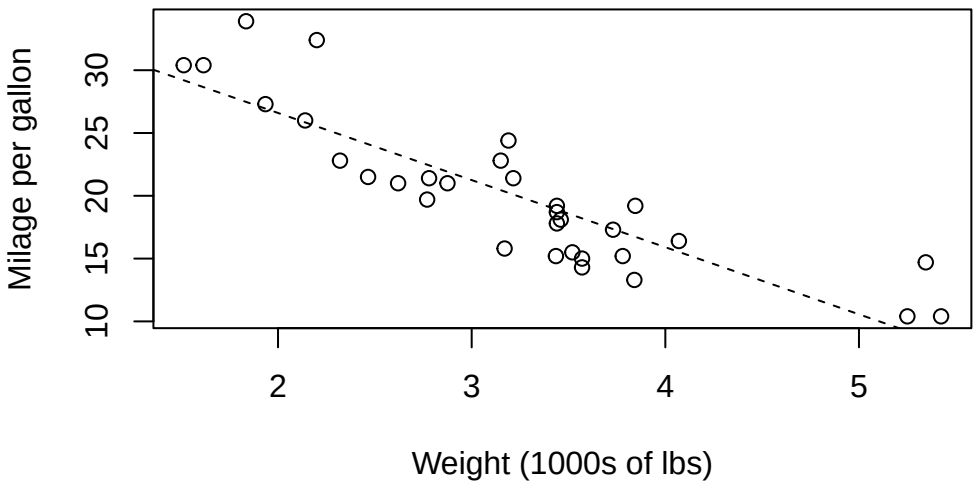
First we need to load the data. We'll use the `mtcars` data again, which comes with `R`.

```
data(mtcars)
```

### Brief intro to regression confidence intervals

To demonstrate CI's for regression models, we'll start with the same regression model we fitted last time, with `mpg` as the response variable and `wt` as the explanatory variable.

```
mod <- lm(mpg ~ wt, data = mtcars)
plot(
    x = mtcars$wt,
    y = mtcars$mpg,
    xlab = "Weight (1000s of lbs)",
    ylab = "Milage per gallon"
)
abline(mod, lty = 2, lwd = 1)
```




The dashed line in this plot is a visualization of the fitted regression line. Recall that the `summary()` of the model gives us information about the fitted coefficients and their confidence intervals.

```
summary(mod)
```

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
```

```
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Since we covered the basics of what a confidence interval means, let's try to interpret these confidence intervals in the context of our problem. First we'll look at the confidence intervals for the coefficients.

```
# Get confidence intervals for the regression parameters
confint(mod, level = 0.95)
```

```
                2.5 %    97.5 %
(Intercept) 33.450500 41.119753
wt          -6.486308 -4.202635
```

- Our estimated intercept is 37.29 (95% CI: 33.45, 41.12).
- Our estimate slope is -5.34 (95% CI: -6.49, -4.20).

If our confidence interval does not contain zero, that's the same as our p-value for the coefficient being significant. Our data are consistent (at a 95% confidence level) with any number in the CI.

Remember last time we said the intercept is not very useful for us practically (a car can't have weight zero). We can interpret the slope though. Our point estimate is -5.34, so if we increased the weight of a car by 1 unit (for this dataset, that is 1000 lbs) and didn't change anything else about the car, we would expect the MPG to decrease by -5.34. However, we know that this is not the only value consistent with our data. We can also say that we are 95% confident that the true amount of change is between -6.49 to -4.20 units in the MPG.

### Confidence intervals for predictions at specific x values

We can also get a CI for the prediction at each point – remember that this is a CI for the conditional mean of y ($E(y \mid x_i)$). Note that you need to do `?predict.lm` to get specific help for this function.

```
predict(mod, interval = "confidence", level = 0.95) |>
    head()
```

```
                    fit      lwr      upr
Mazda RX4         23.28261 21.98867 24.57655
Mazda RX4 Wag     21.91977 20.75275 23.08679
Datsun 710        24.88595 23.38301 26.38890
```

```
Hornet 4 Drive    20.10265 19.00300 21.20230
Hornet Sportabout 18.90014 17.77147 20.02882
Valiant           18.79325 17.65922 19.92729
```

These values are somewhat tricky to interpret – they take into account the uncertainty in our parameter estimates, but nothing else. If we look at the weight of a Mazda RX4 (2.62 thousand lbs) – the first observation in this data frame – we can see that the model predicts an MPG of 23.28. We can be 95% confident that the mean MPG of cars with a weight of 2.62 thousand lbs is between 21.99 and 24.58.

### Prediction intervals vs confidence intervals

We can also get PIs for each observed value of the explanatory variables. The PI is different from the CI because it gives us a plausible range of values that we might expect for a new observation with a specific x value, if our model is correct for that new observation.

So, for example, if Mazda released a new car that weighs the same as the RX4 ( again, 2.62 thousand lbs), what would we reasonably expect the MPG for that car to be, based on our model? This accounts for the error in our model estimation (this is all that the CI captures), and the individual variance across experimental units (across different car models of the same weight, for us).

Let's calculate the PIs for the values we observed in the data. So, the PI will ALWAYS be larger than the CI at the same level, because it accounts for an extra source of variation. Note the warning message that we get here.

```
predict(mod, interval = "prediction") |>
    head()
```

```
Warning in predict.lm(mod, interval = "prediction"): predictions on current data refer to _fu
```

```
                      fit      lwr      upr
Mazda RX4         23.28261 16.92894 29.63628
Mazda RX4 Wag     21.91977 15.59072 28.24882
Datsun 710        24.88595 18.48644 31.28546
Hornet 4 Drive    20.10265 13.78568 26.41962
Hornet Sportabout 18.90014 12.57806 25.22223
Valiant           18.79325 12.47021 25.11630
```

Recall that again, the weight of the Mazda RX4 is the first observation in this data frame. Based on this PI, we can say that, again, our model predicts an MPG of 23.28 for a car with

weight 2.62 thousand lbs – note that the point estimates don't change. What is new is that with this information, we can say that we are 95% confident that the MPG for an individual (new) car with weight 2.62 thousand lbs is between 16.92 and 29.64.

## Interpolation

We are likely also interested in getting CIs or PIs for a wide range of values, e.g., for all the values of the explanatory variable in the range that our model covers. Getting a prediction for a new value in the range of the model is called **interpolation** (as opposed to **extrapolation**, where you make predictions for values outside the range of the model). Note here that the "range of the model" is the range of the explanatory variable – our model is only "valid" (in a specific sense) for values in-between the x-values we already observed, and we have to make more assumptions to look outside of those bounds.

The process for getting interpolated predictions in `R` is simple: all we need to do is build a vector with all the values we want to get predictions on. To get the CI for every value in the predictor range, that means we need a vector that starts at the minimum predictor value and ends at the maximum predictor value, and increases by some step size each time. In `R`, you can easily create this vector with the `seq()` function, which allows you to decide how many points you should evaluate, or what the step size between each point should be. (Remember there are actually an uncountably infinite number of points in the interval, so we have to choose enough points in this vector to get a reasonable amount of precision for whatever we are doing.)

```
summary(mtcars$wt)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.513   2.581   3.325   3.217   3.610   5.424
```

From the summary of the `wt` values, we see the min and max of the observed weights. It's usually fine to extrapolate just a tiny amount, so based on these numbers, we'll get predictions between 1.5 and 5.5, because I like nice round numbers. We'll choose a step size of 0.1 each time, which should give us smooth enough predictions to be useful.

```
new_wts <- seq(1.5, 5.5, 0.1)
```

Note that because of how the `predict.lm()` function works, we need to make sure the function knows that these values are associated with the `wt` variable. Specifically, these values have to be in a column named `wt` inside of a data frame.

```
# Has to have the same name as the variable in the linear model
new_wts_df <- data.frame(wt = new_wts)
```

Now we can pass this data frame as the `newdata` argument to the `predict()` function, and it will get the PI (or CIs if we set `interval = "confidence"`) for each of those values, even though we did not observe them.

```
new_wts_preds <-
    predict(
        mod,
        newdata = new_wts_df,
        interval = "prediction",
        level = 0.95
    )

# Put the weight values together with the predictions
# You could save this as an object if you wanted to do more with it, like
# make plots.
# See the appendix for the broom::augment() function that does this
# automatically.
interp_preds <- cbind(new_wts_df, new_wts_preds)
head(interp_preds)
```

```
  wt      fit      lwr      upr
1 1.5 29.26842 22.65412 35.88271
2 1.6 28.73397 22.15262 35.31532
3 1.7 28.19952 21.64930 34.74975
4 1.8 27.66508 21.14412 34.18604
5 1.9 27.13063 20.63707 33.62419
6 2.0 26.59618 20.12811 33.06425
```

## Appendix: tidier / easier to use code for regression models.

To get the CIs, etc., we can also use the `broom` package, which will output all of our results as a well-formatted data frame, which can often be easier to use than the output from base `R` functions. To get the CIs for the coefficients, we can use `broom::tidy()`. To get the specific documentation for this function and its options, you can use `?broom::tidy.lm`.

```
broom::tidy(mod)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
```

```
1 (Intercept)    37.3      1.88      19.9  8.24e-19
2 wt             -5.34     0.559     -9.56 1.29e-10
```

To get CIs or PIs, you can use `broom::augment.lm()`. This is similar to the `predict()` function, but it attaches the prediction columns directly to the inputted data frame, instead of returning only the predictions.

Note that by default, if you don't specify the `newdata` argument, the predictions will be returned for all of the explanatory variable combinations that were observed in the data, if you want others you have to create a data frame of explanatory variables that has all the observations you want, like with the `predict()` function.

```
broom::augment(mod, interval = "confidence")
```

```
# A tibble: 32 x 11
   .rownames        mpg    wt .fitted .lower .upper .resid   .hat .sigma .cooksd
   <chr>          <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
 1 Mazda RX4       21    2.62    23.3   22.0   24.6 -2.28  0.0433   3.07 1.33e-2
 2 Mazda RX4 Wag   21    2.88    21.9   20.8   23.1 -0.920 0.0352   3.09 1.72e-3
 3 Datsun 710      22.8  2.32    24.9   23.4   26.4 -2.09  0.0584   3.07 1.54e-2
 4 Hornet 4 Drive  21.4  3.22    20.1   19.0   21.2  1.30  0.0313   3.09 3.02e-3
 5 Hornet Sporta~  18.7  3.44    18.9   17.8   20.0 -0.200 0.0329   3.10 7.60e-5
 6 Valiant         18.1  3.46    18.8   17.7   19.9 -0.693 0.0332   3.10 9.21e-4
 7 Duster 360      14.3  3.57    18.2   17.0   19.4 -3.91  0.0354   3.01 3.13e-2
 8 Merc 240D       24.4  3.19    20.2   19.1   21.3  4.16  0.0313   3.00 3.11e-2
 9 Merc 230        22.8  3.15    20.5   19.3   21.6  2.35  0.0314   3.07 9.96e-3
10 Merc 280        19.2  3.44    18.9   17.8   20.0  0.300 0.0329   3.10 1.71e-4
# i 22 more rows
# i 1 more variable: .std.resid <dbl>
```

We can also get the prediction intervals (and the SEs).

```
broom::augment(mod, interval = "prediction", se_fit = TRUE)
```

```
# A tibble: 32 x 12
   .rownames        mpg    wt .fitted .lower .upper .se.fit .resid   .hat .sigma
   <chr>          <dbl> <dbl>   <dbl>  <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
 1 Mazda RX4       21    2.62    23.3   16.9   29.6   0.634 -2.28  0.0433   3.07
 2 Mazda RX4 Wag   21    2.88    21.9   15.6   28.2   0.571 -0.920 0.0352   3.09
 3 Datsun 710      22.8  2.32    24.9   18.5   31.3   0.736 -2.09  0.0584   3.07
 4 Hornet 4 Drive  21.4  3.22    20.1   13.8   26.4   0.538  1.30  0.0313   3.09
```

```
 5 Hornet Sporta~  18.7  3.44    18.9   12.6   25.2   0.553 -0.200 0.0329   3.10
 6 Valiant         18.1  3.46    18.8   12.5   25.1   0.555 -0.693 0.0332   3.10
 7 Duster 360      14.3  3.57    18.2   11.9   24.5   0.573 -3.91  0.0354   3.01
 8 Merc 240D       24.4  3.19    20.2   13.9   26.6   0.539  4.16  0.0313   3.00
 9 Merc 230        22.8  3.15    20.5   14.1   26.8   0.540  2.35  0.0314   3.07
10 Merc 280        19.2  3.44    18.9   12.6   25.2   0.553  0.300 0.0329   3.10
# i 22 more rows
# i 2 more variables: .cooksd <dbl>, .std.resid <dbl>
```

And finally, remember that we can "interpolate" to get CI values for all possible explanatory variable values.

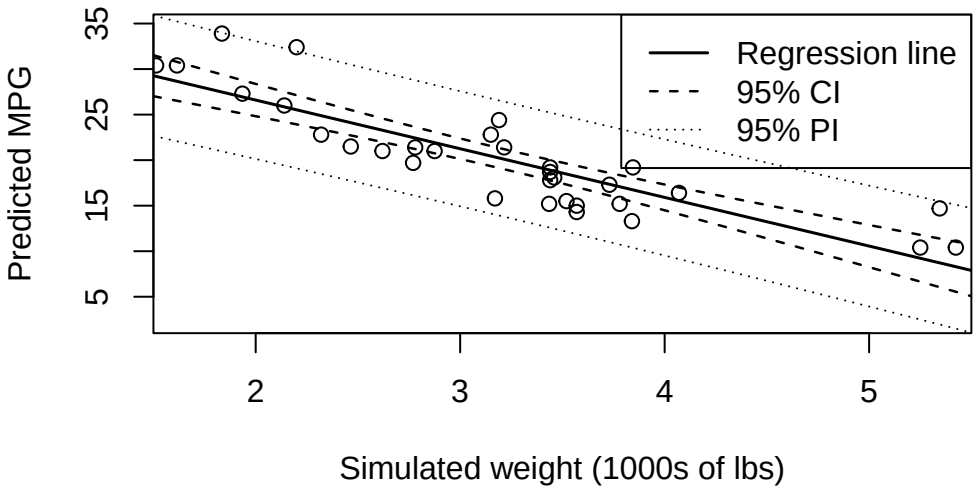
```
newdata_preds <-
    broom::augment(
        mod,
        newdata = data.frame(wt = seq(1.5, 5.5, 0.1)),
        interval = "prediction"
    )

newdata_ci <-
    broom::augment(
        mod,
        newdata = data.frame(wt = seq(1.5, 5.5, 0.1)),
        interval = "confidence"
    )
```

And here's a quick example of plotting these data easily using the output returned from `broom::augment()`.

```
# Create the plot with the regression line on it
plot(
    # Specify x and y data vectors
    newdata_preds$wt,
    newdata_preds$.fitted,
    # Specify axis titles
    xlab = "Simulated weight (1000s of lbs)",
    ylab = "Predicted MPG",
    # Make a line plot
    type = "l",
    # Change the line thickness (lwd = line width)
    lwd = 1.5,
    # Set the axis limits
```

```
    xlim = c(1.5, 5.5),
    ylim = c(1, 36),
    # These are graphical parameters necessary to take the xlim/ylim literally,
    # if you don't have them you will get a little extra space around each limit.
    xaxs = "i",
    yaxs = "i"
)
# Add lines for the CI and PI -- each of these commands adds a line to the plot
lines(
    newdata_ci$wt,
    newdata_ci$.lower,
    # Lty = 2 gives a dashed line (lty = line type)
    lty = 2,
    lwd = 1.25
)
lines(
    newdata_ci$wt,
    newdata_ci$.upper,
    lty = 2,
    lwd = 1.25
)
lines(
    newdata_preds$wt,
    newdata_preds$.lower,
    # Lty = 3 gives a dotted line
    lty = 3
)
lines(
    newdata_preds$wt,
    newdata_preds$.upper,
    lty = 3
)
# Add a legend for the different lty/lwd values
legend(
    "topright",
    c("Regression line", "95% CI", "95% PI"),
    lty = c(1, 2, 3),
    lwd = c(1.5, 1.25, 1)
)
# Add the original data points to the plot
points(mtcars$wt, mtcars$mpg)
```




As a final note, `broom::glance()` is the third and final function in the main `broom` trio which can give you the model fit statistics from a model.

```
broom::glance(mod)
```

```
# A tibble: 1 x 12
  r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
1     0.753         0.745  3.05      91.4 1.29e-10     1  -80.0  166.  170.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```