ggplot2: 3 ways to plot estimated lines

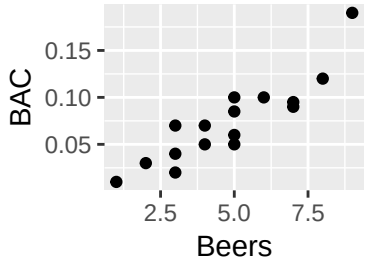
Recall our example modeling the relationship between the number of beers someone has had and their blood alcohol content:

```
library(ggplot2)

beer <- read.csv("http://www.evanlray.com/data/openintro/bac.csv")
lm_fit <- lm(BAC ~ Beers, data = beer)
summary(lm_fit)
```

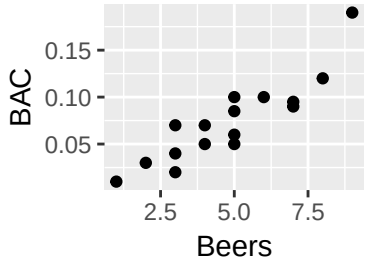
```
##
## Call:
## lm(formula = BAC ~ Beers, data = beer)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.027118 -0.017350  0.001773  0.008623  0.041027
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.012701   0.012638  -1.005    0.332
## Beers        0.017964   0.002402   7.480 2.97e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02044 on 14 degrees of freedom
## Multiple R-squared:  0.7998, Adjusted R-squared:  0.7855
## F-statistic: 55.94 on 1 and 14 DF,  p-value: 2.969e-06
```

Here are two ways to make our scatter plot, both of which work equally well:

```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point()
```




```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC))
```



We demonstrate 3 ways to add the estimated function $\widehat{f}(X) = \widehat{\beta}_0 + \widehat{\beta}_1 X$ to the scatter plot. These are (in order from least flexible to most flexible):

1. `geom_abline`: add a line with intercept `a` and slope `b`
2. `geom_smooth`: add a smooth obtained via a specified method and formula
3. `stat_function`: add a user-defined function

Method 1: `geom_abline`

In addition to the standard `mapping` and `data` arguments, the `geom_abline` function requires us to specify:

- an `intercept` for the line
- a `slope` for the line

These will be obtained from the estimated `coef`ficients from `lm`:

```
coef(lm_fit)
```

```
## (Intercept)       Beers
## -0.01270060  0.01796376
```

```
coef(lm_fit)[1]
```

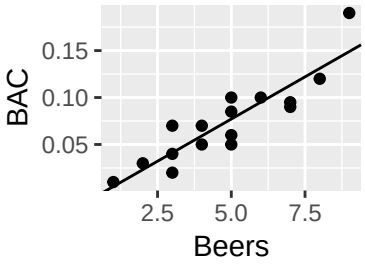
```
## (Intercept)
##  -0.0127006
```

```
coef(lm_fit)[2]
```

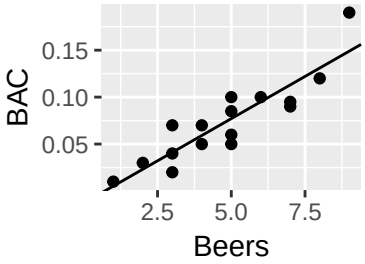
```
##      Beers
## 0.01796376
```

Our final plot can be made in either of two ways:

```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point() +
  geom_abline(intercept = coef(lm_fit)[1], slope = coef(lm_fit)[2])
```




```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_abline(intercept = coef(lm_fit)[1], slope = coef(lm_fit)[2])
```



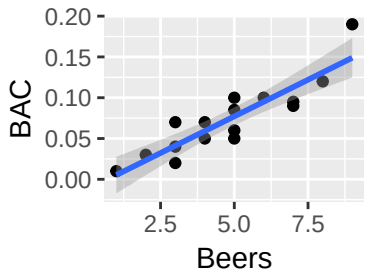
Method 2: `geom_smooth`

In addition to the standard `mapping` and `data` arguments, the `geom_smooth` function requires us to specify:

- a `method` for obtaining the smooth. Common choices are `lm`, `glm`, `gam`, and `loess`. `auto` picks a flexible method depending on the sample size.
- a `formula` to use in obtaining the smooth. Note that this should be specified in terms of `y` and `x`, not the names of variables in your data set.

Here we are fitting a line using the `lm` function:

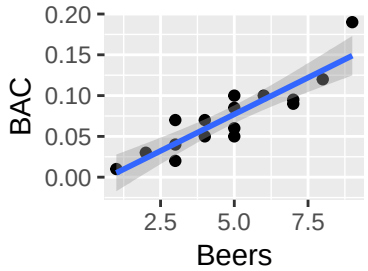
```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x)
```



Note that if you didn't specify the `data` and `mapping` universally in the call to `ggplot`, you'll have to specify them again for `geom_smooth`!

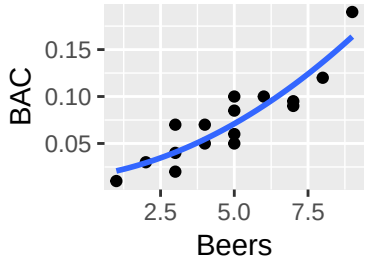
```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_smooth(method = "lm", formula = y ~ x)
```


```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_smooth(data = beer, mapping = aes(x = Beers, y = BAC), method = "lm", formula = y ~ x)
```



For the sake of demonstration, here's how you could plot a quadratic polynomial fit using `geom_smooth`. I'm also illustrating that you can turn off the confidence interval around the smooth by setting `se = FALSE`:

```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x + I(x^2), se = FALSE)
```



Method 3: `stat_function`

In addition to the standard `mapping` and `data` arguments, the `stat_function` function requires us to specify:

- a `function` to plot

There are also some other useful optional arguments:

- `args`: a named list of arguments for the function
- `xlim`: range of values to plot the function over (otherwise, will be minimum and maximum of data values on horizontal axis)

To demonstrate this, here are two possible ways to define an R function that calculates the predicted values from our line:

```
lm_fit_predictions <- function(x) {
  -0.0127006 + 0.0179638 * x
}

lm_fit_predictions(c(1, 2, 3, 4, 5, 6, 7, 8))
```

```
## [1] 0.0052632 0.0232270 0.0411908 0.0591546 0.0771184 0.0950822 0.1130460
## [8] 0.1310098
```

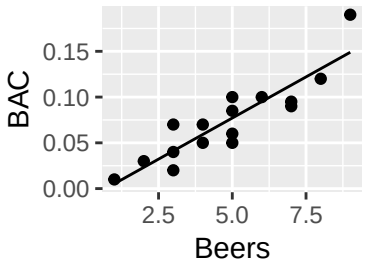
```
lm_fit_predictions <- function(x) {
  predict(lm_fit, data.frame(Beers = x))
}

lm_fit_predictions(c(1, 2, 3, 4, 5, 6, 7, 8))
```

```
##           1           2           3           4           5           6
## 0.005263158 0.023226920 0.041190682 0.059154443 0.077118205 0.095081967
##           7           8
## 0.113045729 0.131009491
```

Note that the function `lm_fit_predictions` that we have defined takes a vector `x` of values for the number of beers, and returns the corresponding vector of predicted values obtained from our linear model fit.

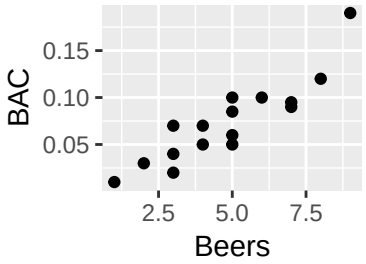
We can now add a plot of this function to our scatterplot as follows:

```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point() +
  stat_function(fun = lm_fit_predictions)
```

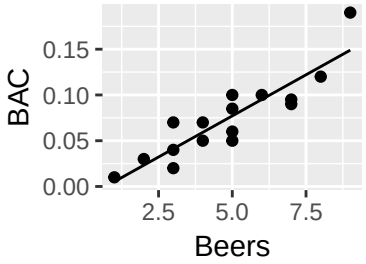


Note that if you didn't specify the `data` and `mapping` universally in the call to `ggplot`, you'll have to specify them again for `geom_smooth`!

```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC)) +
  stat_function(fun = lm_fit_predictions)
```




```
ggplot() +
  geom_point(data = beer, mapping = aes(x = Beers, y = BAC)) +
  stat_function(data = beer, mapping = aes(x = Beers, y = BAC), fun = lm_fit_predictions)
```



Here's a demonstration of `xlim`:

```
ggplot(data = beer, mapping = aes(x = Beers, y = BAC)) +
  geom_point() +
  stat_function(fun = lm_fit_predictions, xlim = c(5, 8))
```