

seaborn.regplot

```
seaborn.regplot(data=None, *, x=None, y=None, x_estimator=None, x_bins=None, x_ci='ci', scatter=True, fit_reg=True, ci=95,
n_boot=1000, units=None, seed=None, order=1, logistic=False, lowess=False, robust=False, logx=False, x_partial=None,
y_partial=None, truncate=True, dropna=True, x_jitter=None, y_jitter=None, label=None, color=None, marker='o',
scatter_kws=None, line_kws=None, ax=None)
```

Plot data and a linear regression model fit.

There are a number of mutually exclusive options for estimating the regression model. See the [tutorial](#) for more information.

Parameters: x, y: string, series, or vector array

Input variables. If strings, these should correspond with column names in `data`. When pandas objects are used, axes will be labeled with the series name.

data : DataFrame

Tidy (“long-form”) dataframe where each column is a variable and each row is an observation.

x_estimator : callable that maps vector -> scalar, optional

Apply this function to each unique value of `x` and plot the resulting estimate. This is useful when `x` is a discrete variable. If `x_ci` is given, this estimate will be bootstrapped and a confidence interval will be drawn.

x_bins : int or vector, optional

Bin the `x` variable into discrete bins and then estimate the central tendency and a confidence interval. This binning only influences how the scatterplot is drawn; the regression is still fit to the original data. This parameter is interpreted either as the number of evenly-sized (not necessary spaced) bins or the positions of the bin centers. When this parameter is used, it implies that the default of `x_estimator` is `numpy.mean`.

x_ci : “ci”, “sd”, int in [0, 100] or None, optional

Size of the confidence interval used when plotting a central tendency for discrete values of `x`. If `"ci"`, defer to the value of the `ci` parameter. If `"sd"`, skip bootstrapping and show the standard deviation of the observations in each bin.

scatter : bool, optional

If `True`, draw a scatterplot with the underlying observations (or the `x_estimator` values).

fit_reg : bool, optional

If `True`, estimate and plot a regression model relating the `x` and `y` variables.

ci : int in [0, 100] or None, optional

Size of the confidence interval for the regression estimate. This will be drawn using translucent bands around the regression line. The confidence interval is estimated using a bootstrap; for large datasets, it may be advisable to avoid that computation by setting this parameter to None.

n_boot : int, optional

Number of bootstrap resamples used to estimate the `ci`. The default value attempts to balance time and stability; you may want to increase this value for “final” versions of plots.

units : variable name in data, optional

If the `x` and `y` observations are nested within sampling units, those can be specified here. This will be taken into account when computing the confidence intervals by performing a multilevel bootstrap that resamples both units and observations (within unit). This does not otherwise influence how the regression is estimated or drawn.

seed : int, numpy.random.Generator, or numpy.random.RandomState, optional

Seed or random number generator for reproducible bootstrapping.

order : int, optional

If `order` is greater than 1, use `numpy.polyfit` to estimate a polynomial regression.

logistic : bool, optional

If `True`, assume that `y` is a binary variable and use `statsmodels` to estimate a logistic regression model. Note that this is substantially more computationally intensive than linear regression, so you may wish to decrease the number of bootstrap resamples (`n_boot`) or set `ci` to None.

lowess : bool, optional

If `True`, use `statsmodels` to estimate a nonparametric lowess model (locally weighted linear regression). Note that confidence intervals cannot currently be drawn for this kind of model.

robust : *bool, optional*

If `True`, use `statsmodels` to estimate a robust regression. This will de-weight outliers. Note that this is substantially more computationally intensive than standard linear regression, so you may wish to decrease the number of bootstrap resamples (`n_boot`) or set `ci` to `None`.

logx : *bool, optional*

If `True`, estimate a linear regression of the form $y \sim \log(x)$, but plot the scatterplot and regression model in the input space. Note that `x` must be positive for this to work.

{x,y}_partial : *strings in `data` or matrices*

Confounding variables to regress out of the `x` or `y` variables before plotting.

truncate : *bool, optional*

If `True`, the regression line is bounded by the data limits. If `False`, it extends to the `x` axis limits.

{x,y}_jitter : *floats, optional*

Add uniform random noise of this size to either the `x` or `y` variables. The noise is added to a copy of the data after fitting the regression, and only influences the look of the scatterplot. This can be helpful when plotting variables that take discrete values.

label : *string*

Label to apply to either the scatterplot or regression line (if `scatter` is `False`) for use in a legend.

color : *matplotlib color*

Color to apply to all plot elements; will be superseded by colors passed in `scatter_kws` or `line_kws`.

marker : *matplotlib marker code*

Marker to use for the scatterplot glyphs.

{scatter,line}_kws : *dictionaries*

Additional keyword arguments to pass to `plt.scatter` and `plt.plot`.

ax : *matplotlib Axes, optional*

Axes object to draw the plot onto, otherwise uses the current Axes.

Returns: **ax** : *matplotlib Axes*

The Axes object containing the plot.

See also

`lplot`

Combine `regplot()` and `FacetGrid` to plot multiple linear relationships in a dataset.

`jointplot`

Combine `regplot()` and `JointGrid` (when used with `kind="reg"`).

`pairplot`

Combine `regplot()` and `PairGrid` (when used with `kind="reg"`).

`residplot`

Plot the residuals of a linear regression model.

Notes

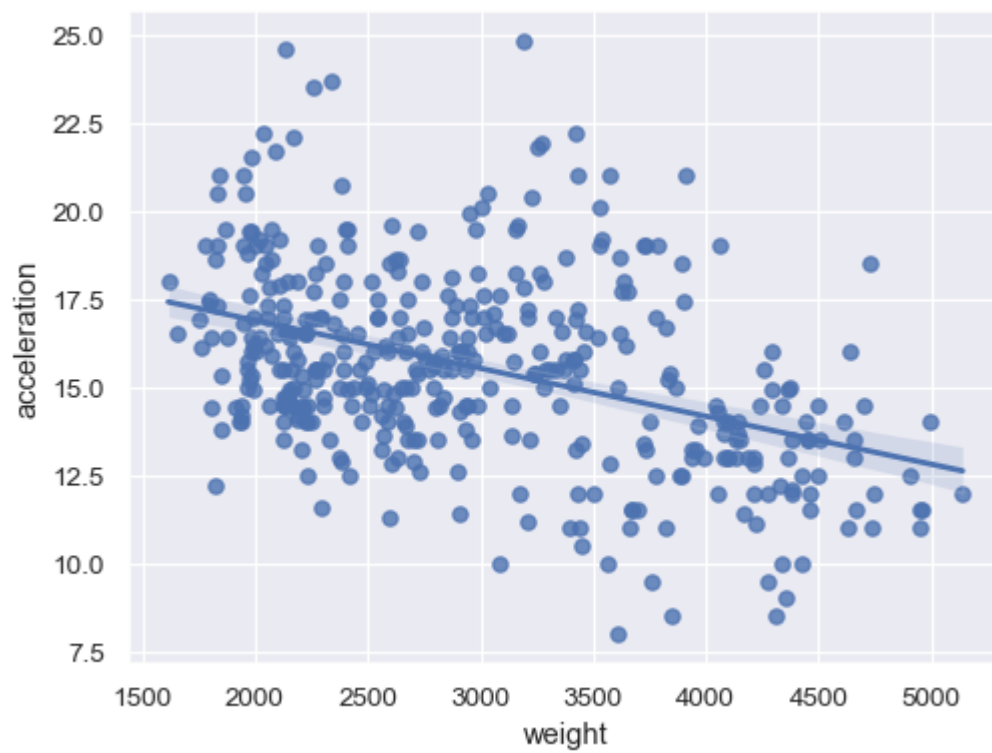
The `regplot()` and `lplot()` functions are closely related, but the former is an axes-level function while the latter is a figure-level function that combines `regplot()` and `FacetGrid`.

It's also easy to combine `regplot()` and `JointGrid` or `PairGrid` through the `jointplot()` and `pairplot()` functions, although these do not directly accept all of `regplot()`'s parameters.

Examples

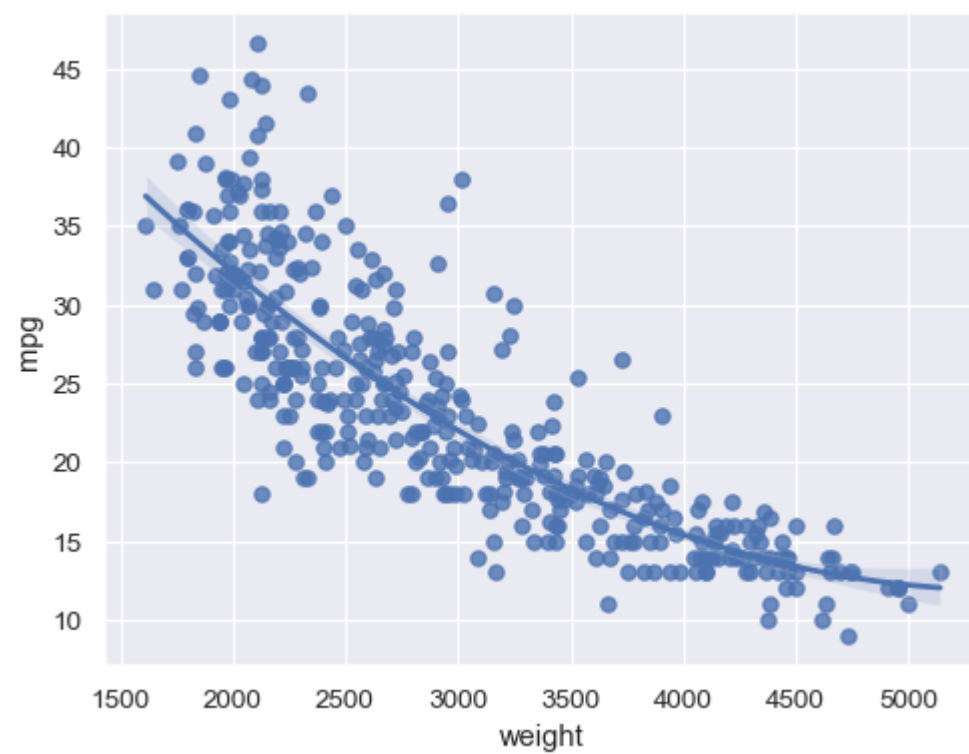
Plot the relationship between two variables in a DataFrame:

```
sns.regplot(data=mpg, x="weight", y="acceleration")
```



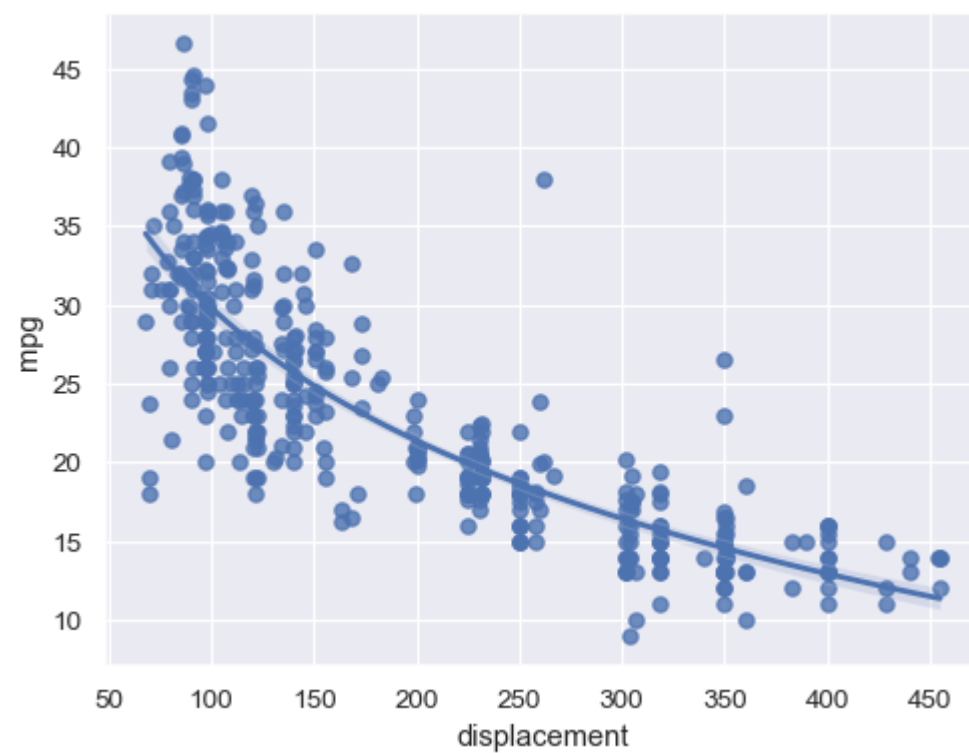
Fit a higher-order polynomial regression to capture nonlinear trends:

```
sns.regplot(data=mpg, x="weight", y="mpg", order=2)
```



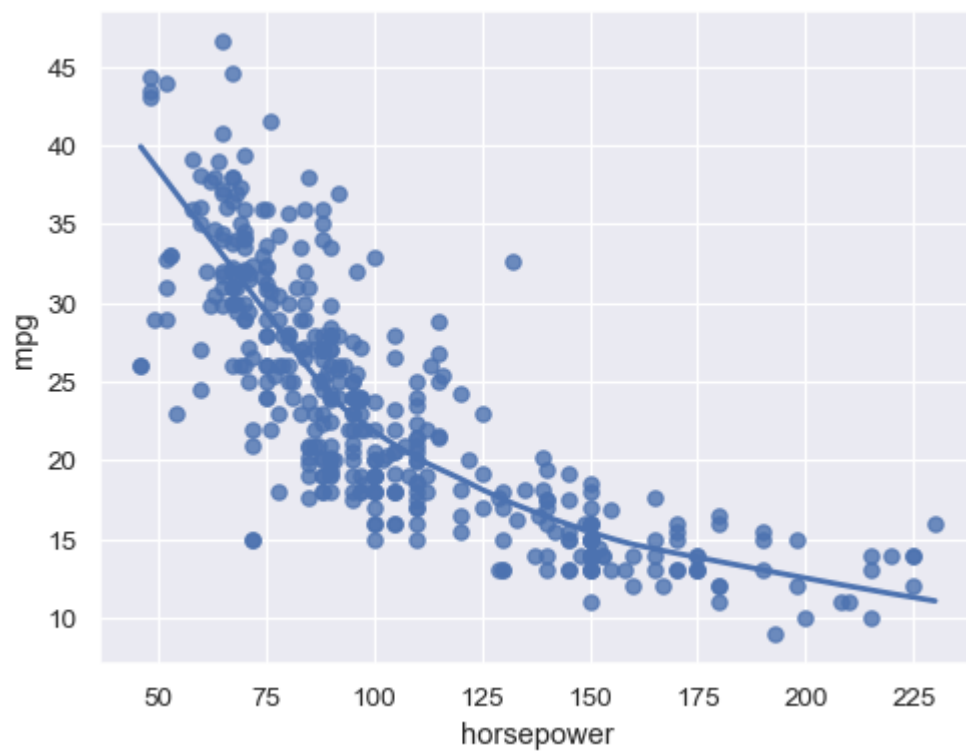
Alternatively, fit a log-linear regression:

```
sns.regplot(data=mpg, x="displacement", y="mpg", logx=True)
```



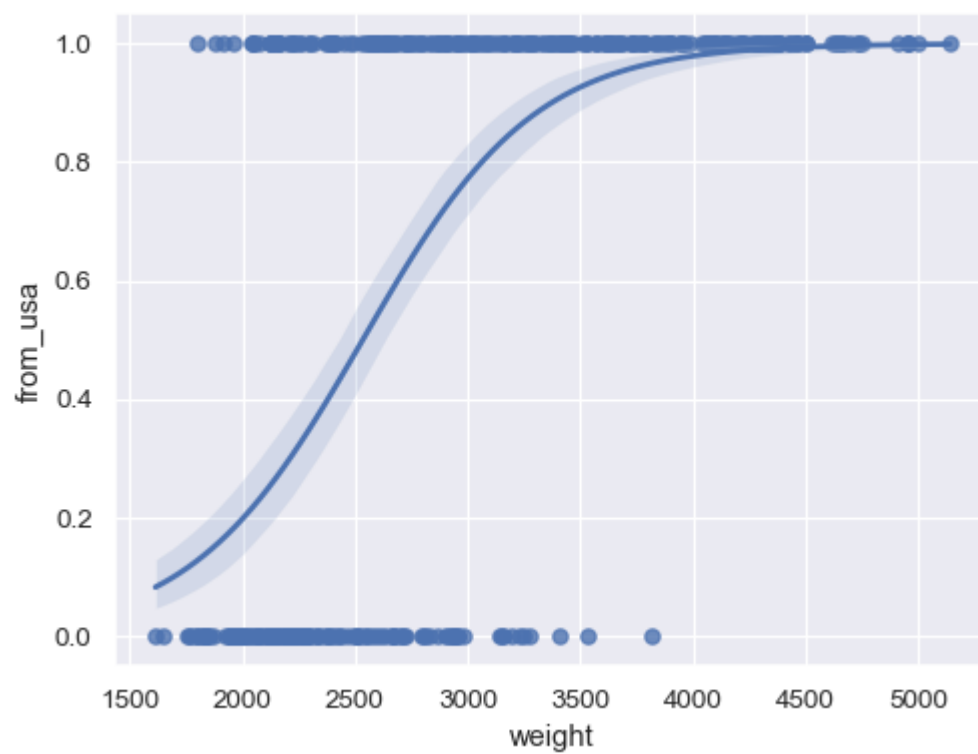
Or use a locally-weighted (LOWESS) smoother:

```
sns.regplot(data=mpg, x="horsepower", y="mpg", lowess=True)
```



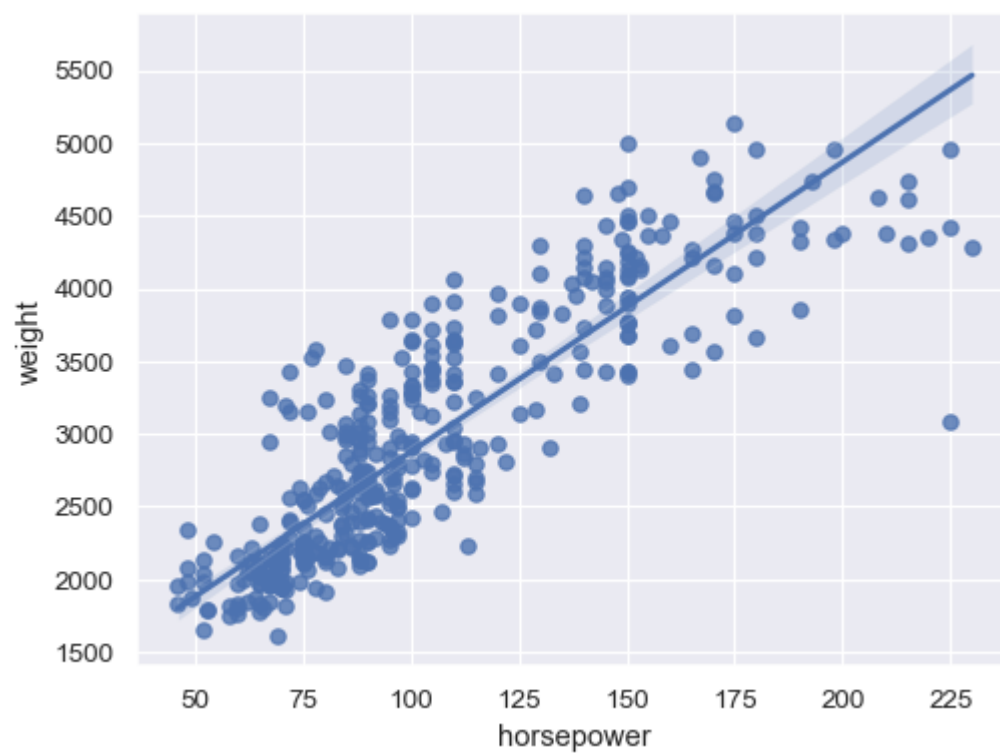
Fit a logistic regression when the response variable is binary:

```
sns.regplot(x=mpg["weight"], y=mpg["origin"].eq("usa").rename("from_usa"), logistic=True)
```



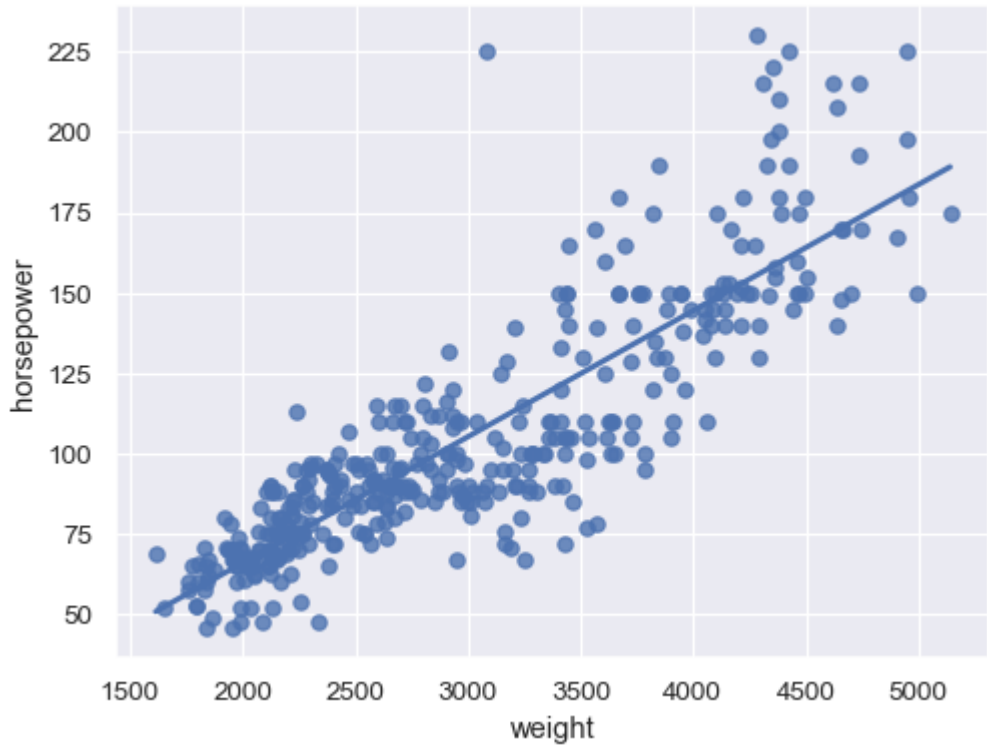
Fit a robust regression to downweight the influence of outliers:

```
sns.regplot(data=mpg, x="horsepower", y="weight", robust=True)
```



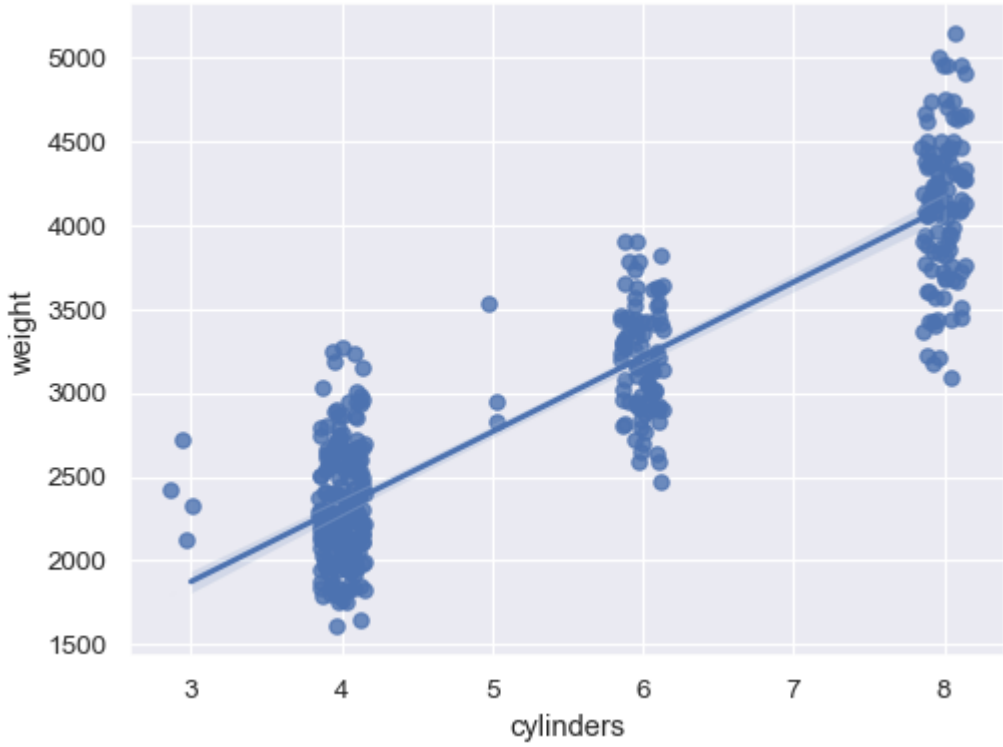
Disable the confidence interval for faster plotting:

```
sns.regplot(data=mpg, x="weight", y="horsepower", ci=None)
```



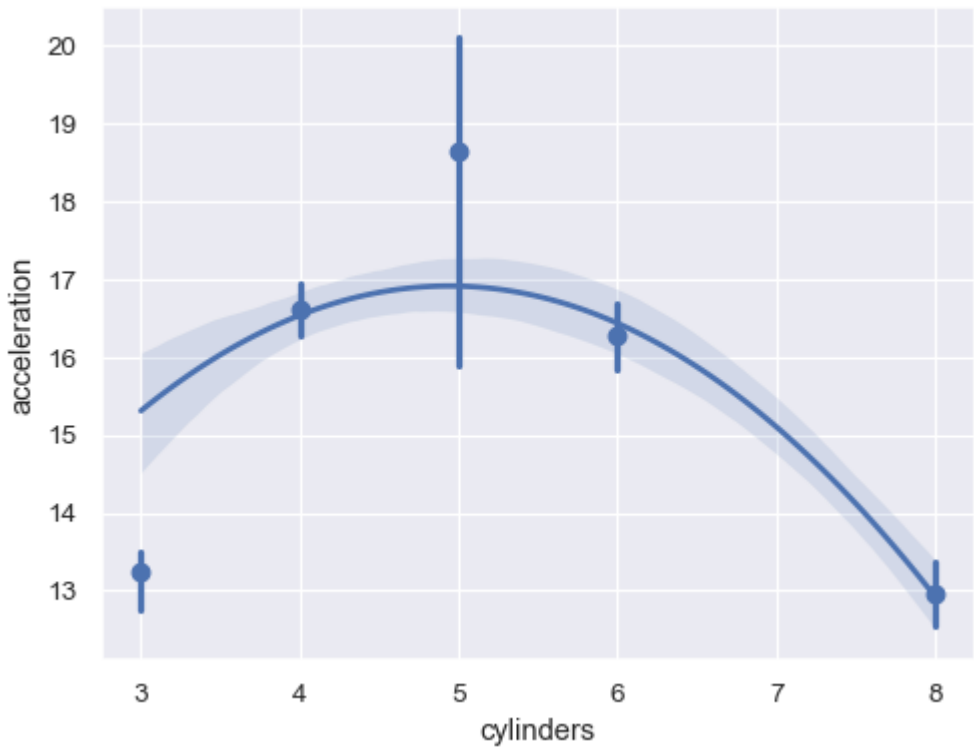
Jitter the scatterplot when the `x` variable is discrete:

```
sns.regplot(data=mpg, x="cylinders", y="weight", x_jitter=.15)
```



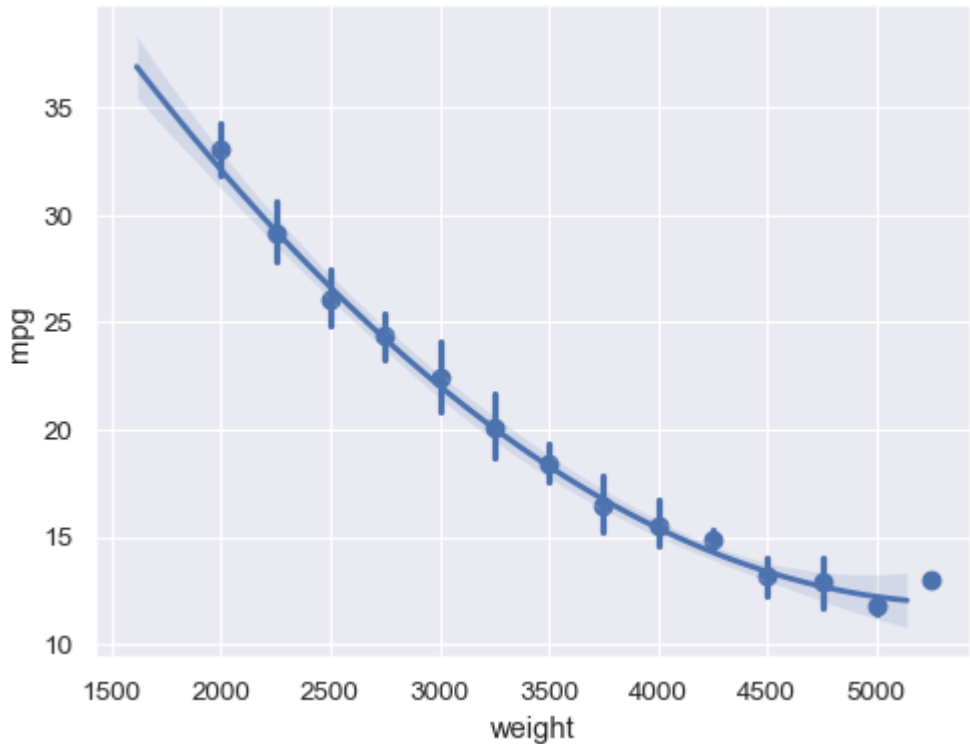
Or aggregate over the distinct `x` values:

```
sns.regplot(data=mpg, x="cylinders", y="acceleration", x_estimator=np.mean, order=2)
```



With a continuous `x` variable, bin and then aggregate:

```
sns.regplot(data=mpg, x="weight", y="mpg", x_bins=np.arange(2000, 5500, 250), order=2)
```



Customize the appearance of various elements:

```
sns.regplot(
    data=mpg, x="weight", y="horsepower",
    ci=99, marker="x", color=".3", line_kws=dict(color="r"),
)
```

