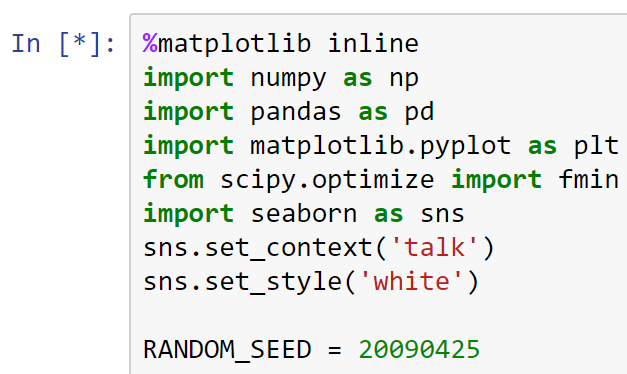
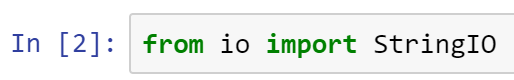
Let’s practice **Linear regression** . . .

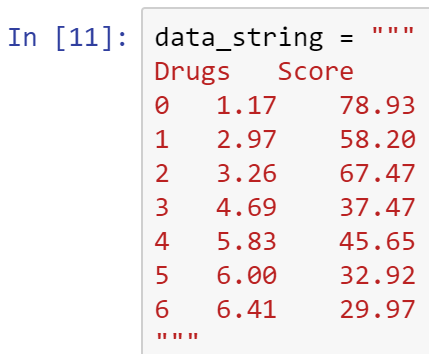
A general, primary goal of many statistical data analysis tasks is to relate the influence of one variable on another.

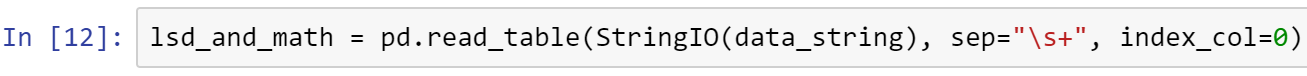
For example:

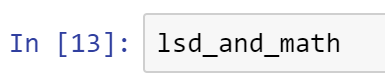
* How different medical interventions influence the incidence or duration of disease?
* How baseball player’s performance varies as a function of age?
* How test scores are correlated with tissue LSD concentration?

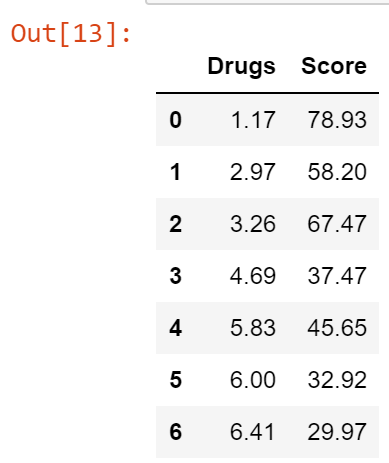


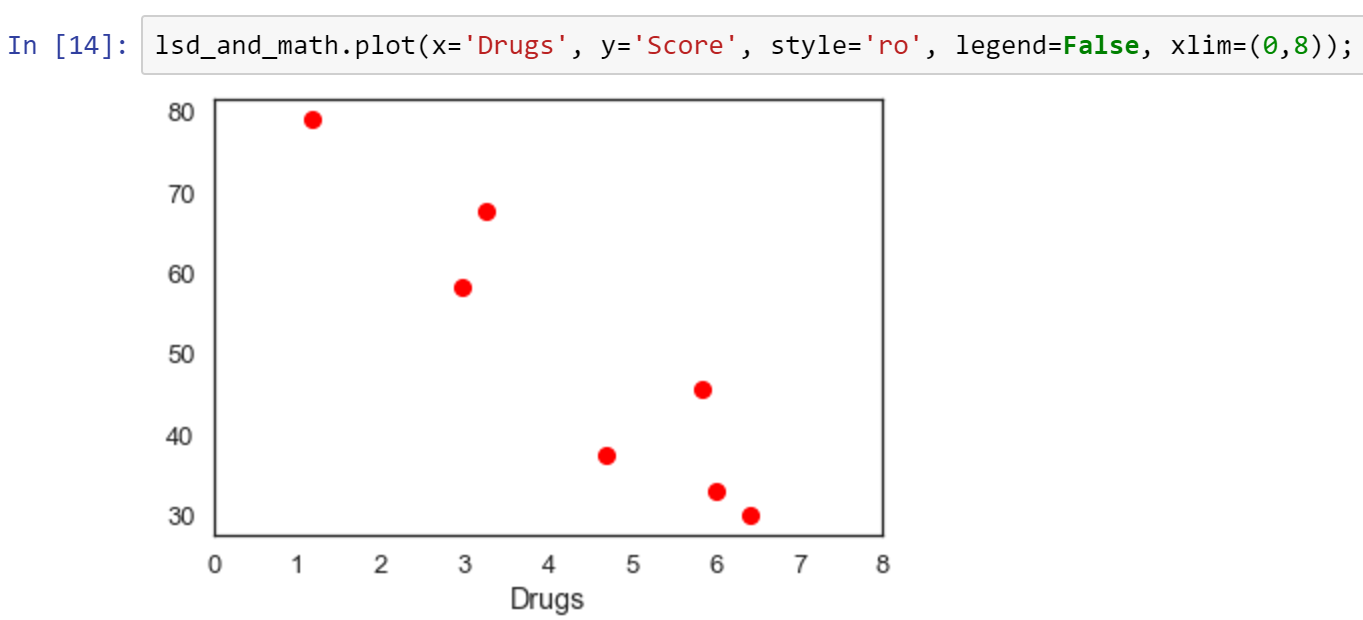










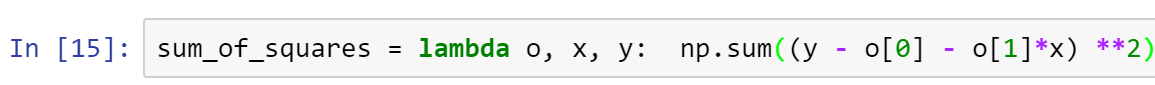


Grades drop as they use more drugs.

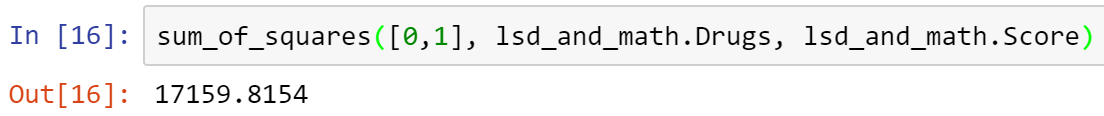
We can build a model to characterize the relationship between X and Y, recognizing that additional factors other than X (the ones we have measured or are interested in) may influence the response variable Y.

A model that fits the data above is a line that is inversely proportional or a negative correlation line.

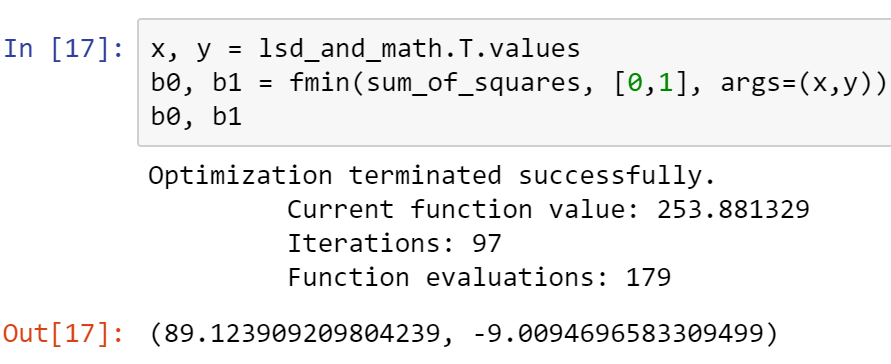
Regression is a weighted sum of independent predictors.



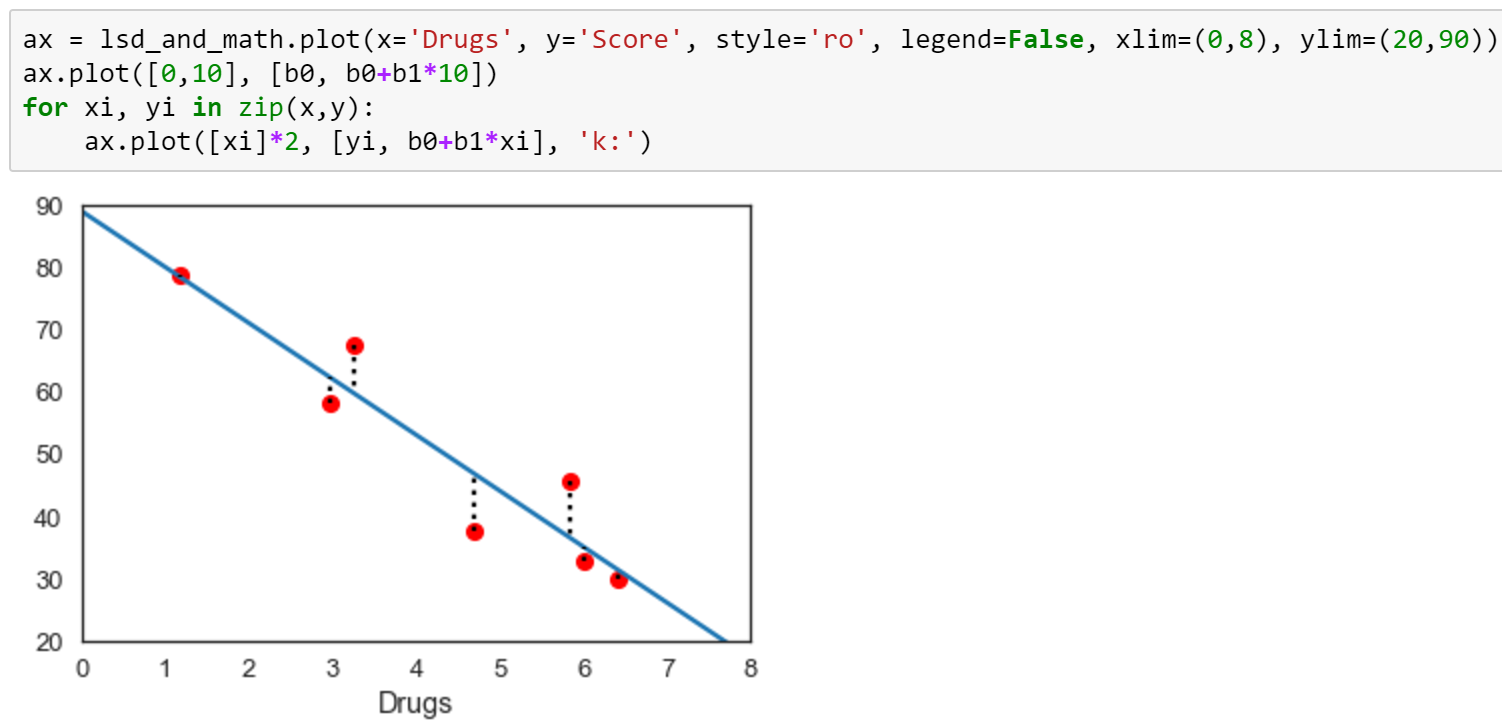
Here is a sample calculation, using arbitrary parameter values.



However, we have the stated objective of minimizing the sum of squares, so we can pass this function to one of several optimizers in SciPy.

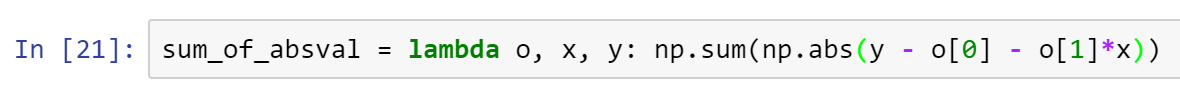


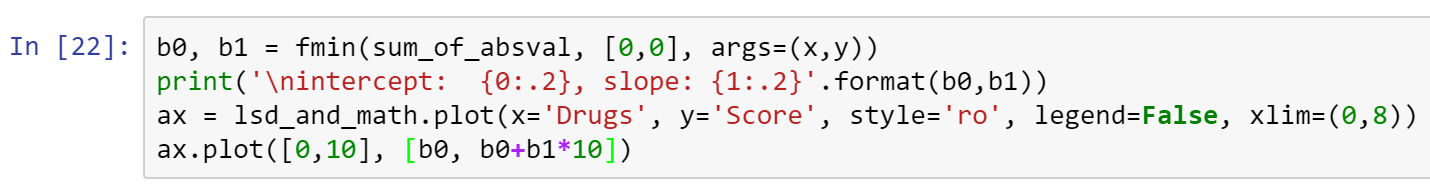


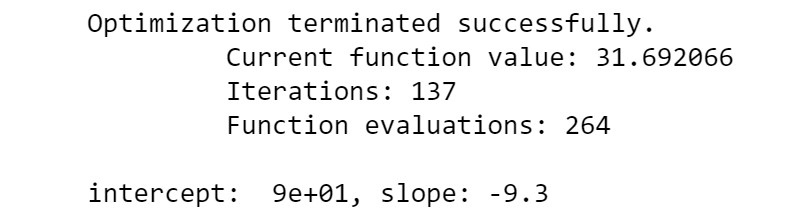


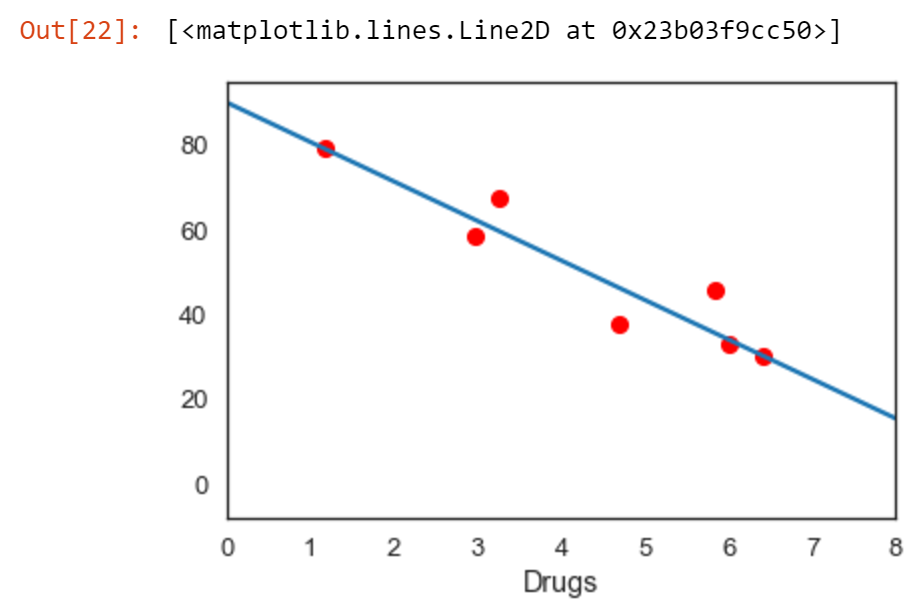
Alternative loss functions

Minimizing the sum of squares is not the only criterion we can use; it is just a very popular (and successful) one. For example, we can try to minimize the sum of absolute differences:







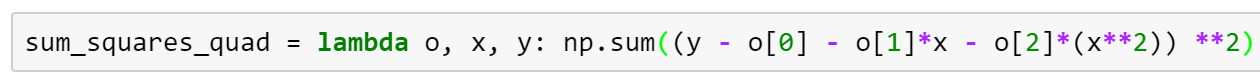


**Polynomial regression**

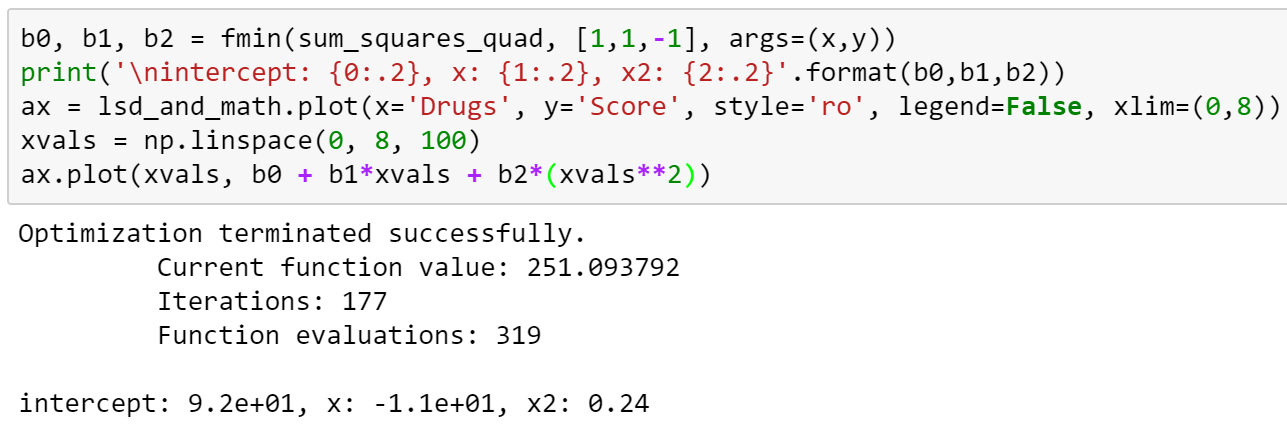
We are not restricted to a straight-line regression model; we can request a curved relationship between our variables by introducing polynomial terms. For example, a cubic model:

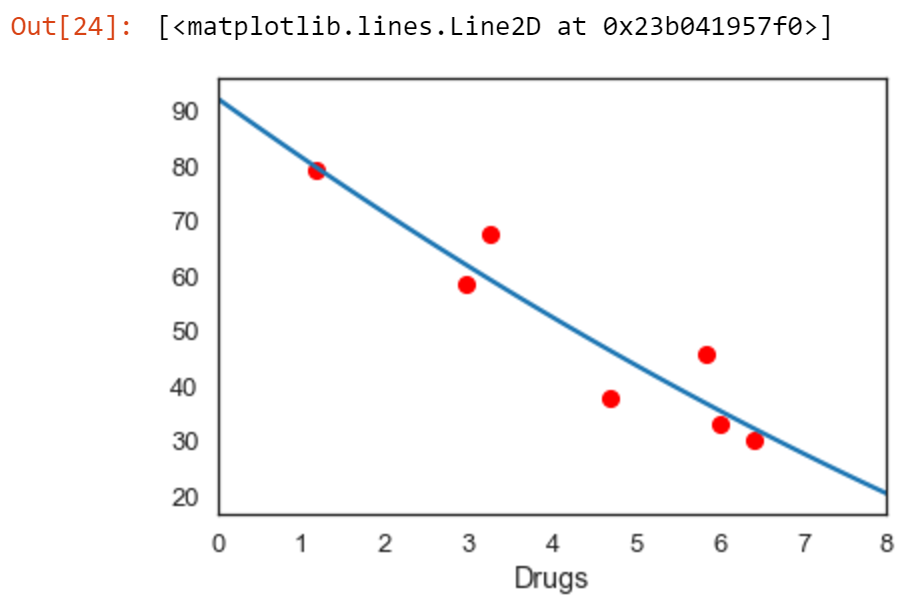
Y = b0 + b1xi + b2x2





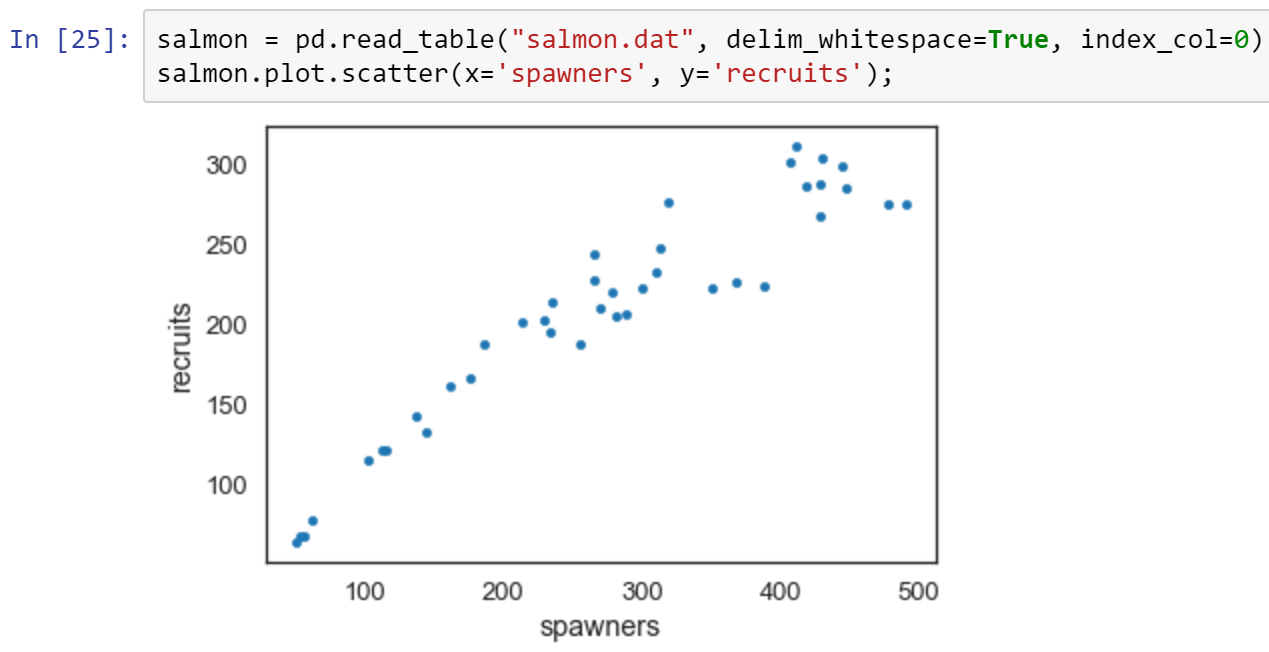






Although a polynomial model characterizes a nonlinear relationship, it is a linear problem in terms of estimation. That is, the regression model f(ylx) is linear in the parameters.

For some data, it may be reasonable to consider polynomials of order>2. For example, consider the relationship between the number of spawning salmon and the number of juveniles recruited into the population the following year; one would expect the relationship to be positive, but not necessarily linear.



Bayesian Linear Regression with PyMC3

In practice, we need not fit least squares models by hand because they are implemented generally in packages such as scikit-learn and statsmodels. Moreover, we are interested not only in obtaining a line of best fit, but also estimates of uncertainty in the line and the parameters used to calculate the line.

Priors

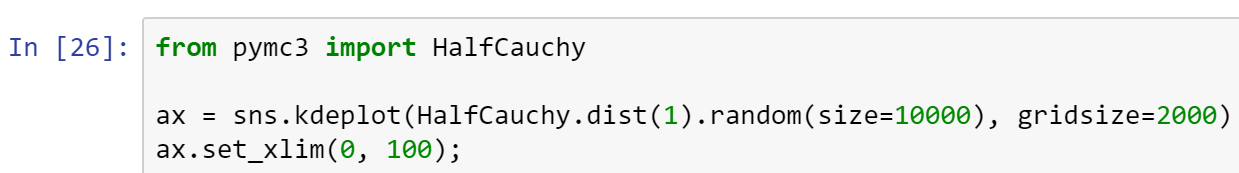
The first step in specifying our model is to specify priors four our models. Since regression parameters are continuous, and potentially positive or negative, we can use a Normal distribution with a variance set to an appropriate value that reflects our prior knowledge of the parameter.

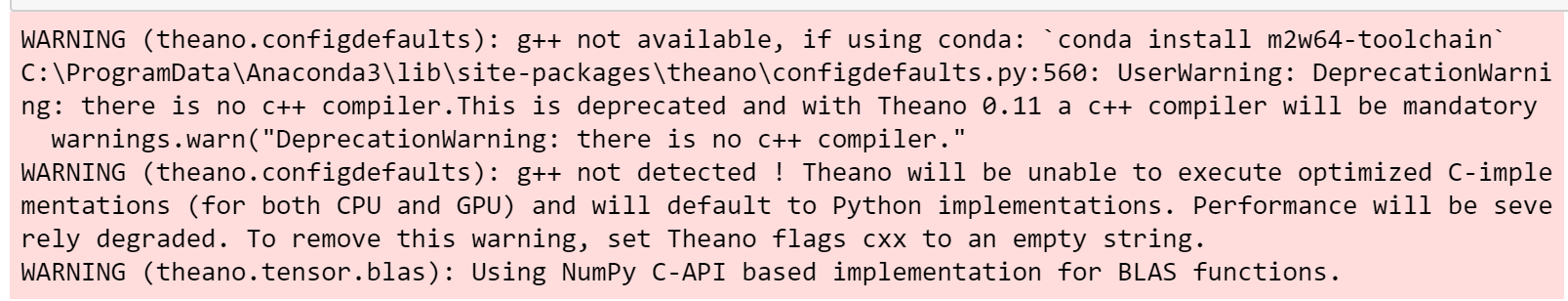
B = Normal(0,100)

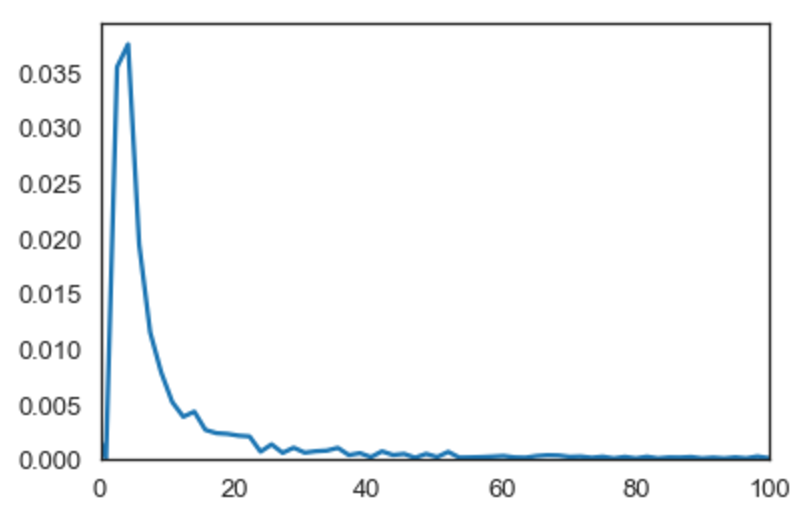
The other latent variable is the residual variance of the observations after applying our model. This is the process uncertainty that we identified earlier.

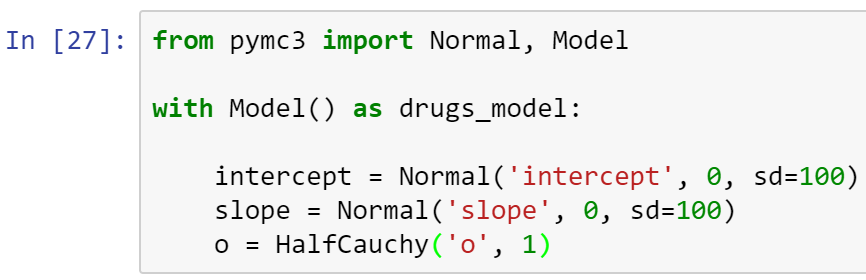
O = HalfCauchy(1)

The half-Cauchy distribution used here provides support over positive continuous values,







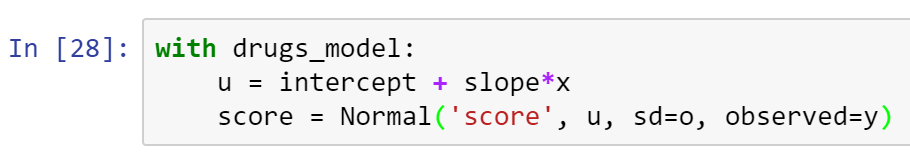


Likelihood

The sampling distribution of the data for a regression model is a normal distribution, and we specified the standard deviation for this sampling distribution in theta above.

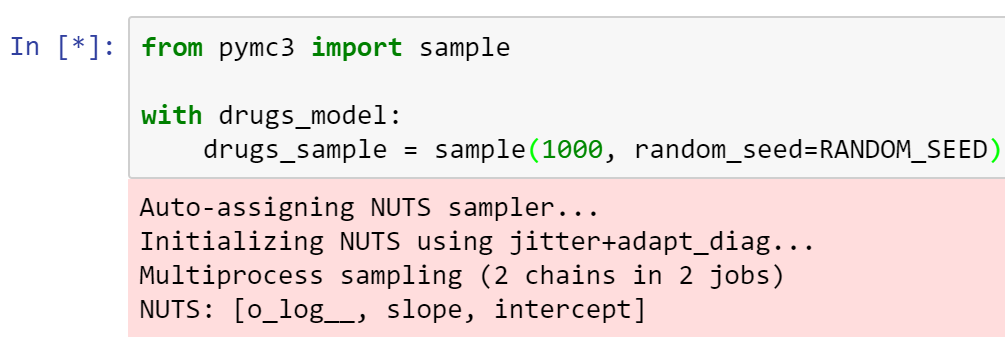
Y = Normal(u,o)

Here, u is the expected value of the ith observation, which is generated by the regression model at the corresponding value of x. We can calculate this expected value as a function of the regression parameters and the data, and pass it to the normal likelihood.

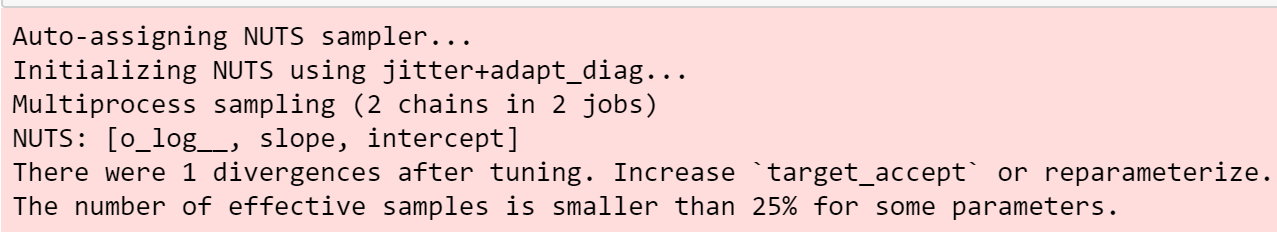


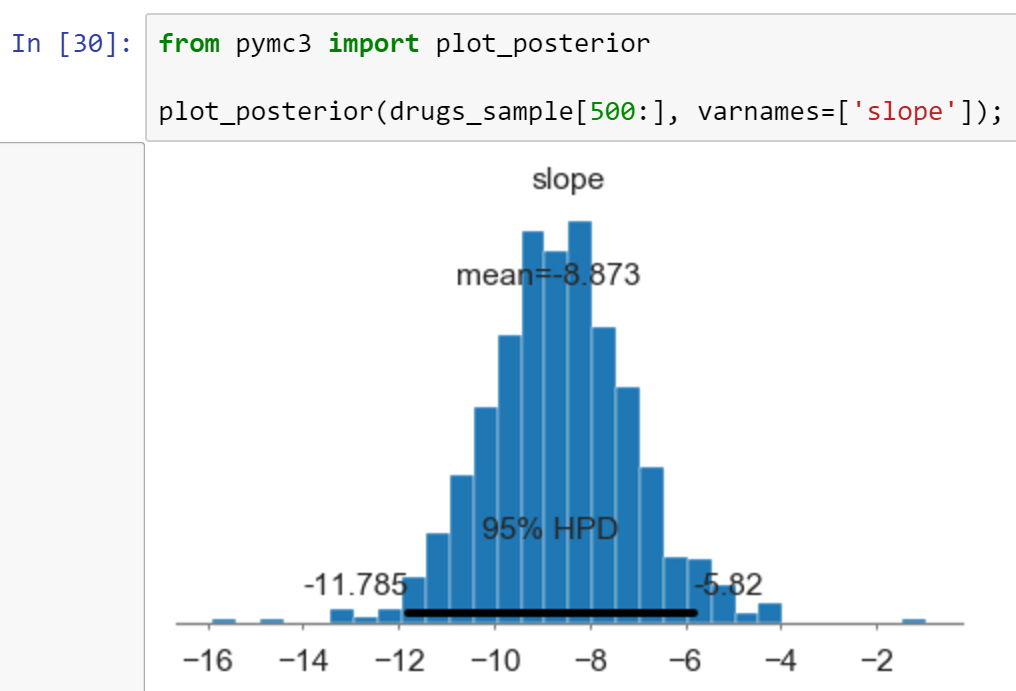
That’s it!

The regression model is fully specified with these 5 lines of Python code. We can now use the fitting method of our choice to estimate a posterior distribution. In the previous module, we used variational inference, here we will use a Markov chain Monte Carlo algorithm, called NUTS (the No U-Turn Sampler).



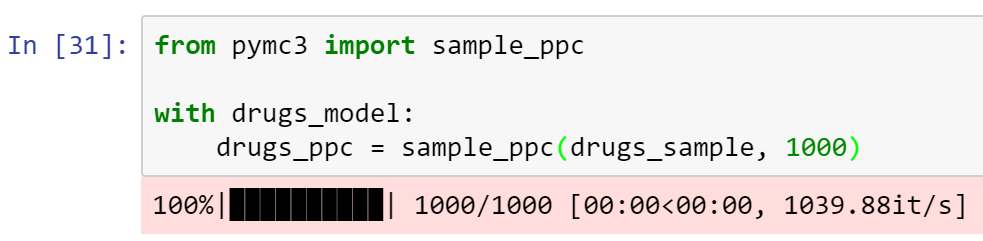
Wait a few seconds until this process completes.



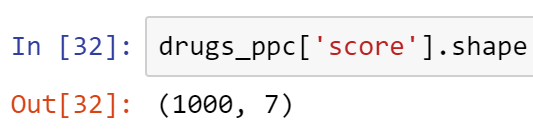


Because we have a vector of samples from the estimated posterior, it is easy to calculate means, medians, standard deviations, and probability intervals. Above, we see the mean and the 95% posterior credible interval reported.

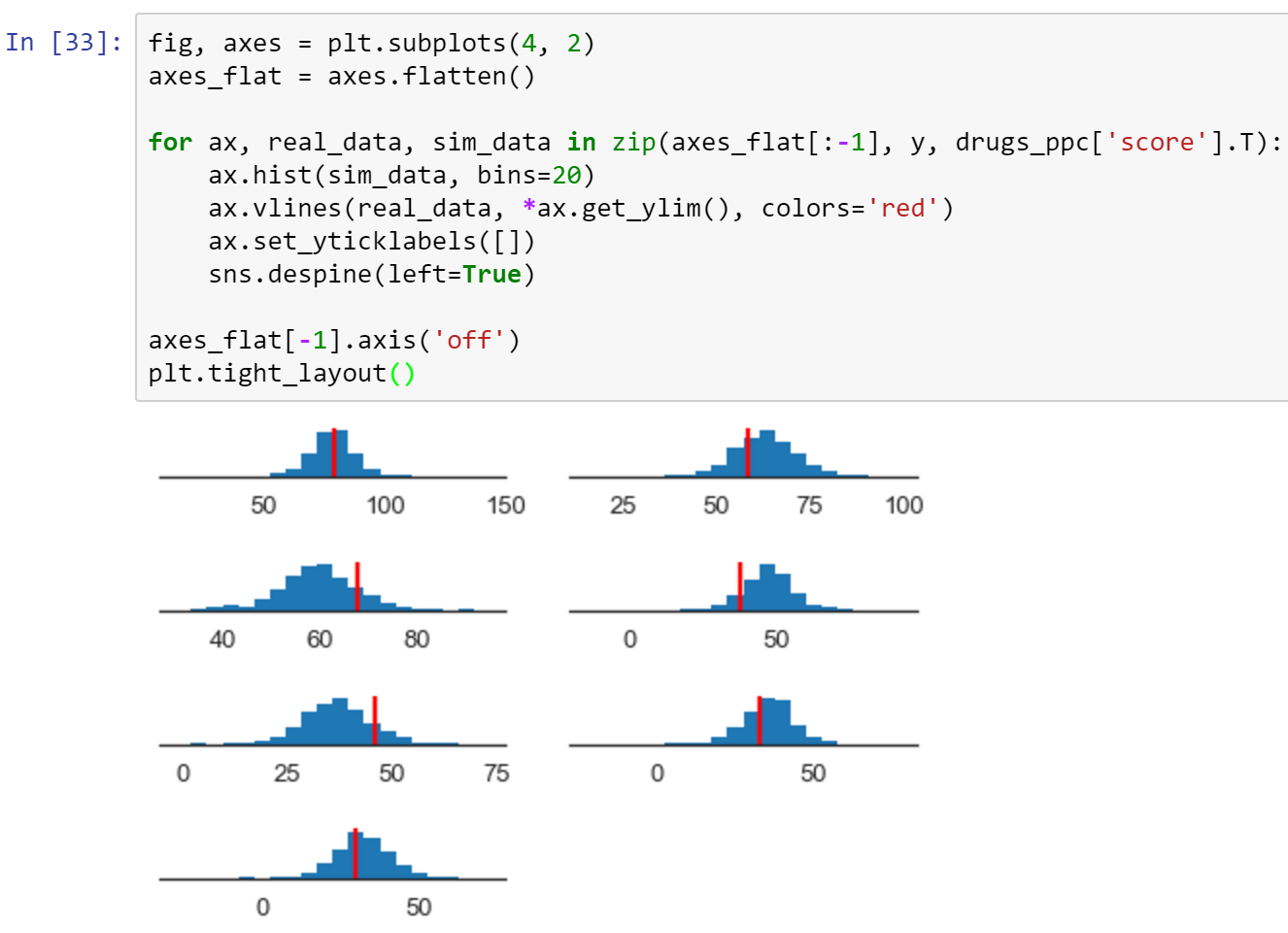
Sampling from the posterior predictive distribution is straightforward in PyMC; the sample\_ppc function draws posterior predictive checks from all of the data likelihoods.



This yields 1000 samples corresponding to each of the seven data points in our observation vector.



We can then compare these simulated data to the data we used to fit the model. Our claim is that the model might have plausibly been used to generate the data that we observed.



Below is a highly recommended resources to expand your knowledge in statistical modeling through Python.

For more on this topic, go to an open-source github repository:

<https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers>

You can also go to an open-community developed textbook:

<http://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/>

PyMC3 was rewritten from Fortran to Theano deep learning library.

● Copy all your code into a Word doc, place your name on it, and submit in Canvas.