Modeling with Linear Regression

Min-Te Sun, Ph.D.

1

Linear Model

- Linear model is the building block of many other models
  - Such as simple and multiple linear regression, logistic regression, ANOVA, ANCOVA, and so on.
- In this chapter, we will cover:
  - Simple linear regression
  - Robust linear regression
  - Hierarchical linear regression
  - Polynomial regression
  - Multiple linear regression
  - Interactions
  - Variable variance

2

Simple Linear Regression

- Many problems we find in science, engineering, and business are of the following form. We have a variable *x* and we want to model/predict a variable *y*. Importantly, these variables are paired like $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_n, y_n)\}$.
  - In the most simple scenario, known as **simple linear regression**, both *x* and *y* are uni-dimensional continuous random variables.
  - We call the *y* variables the **dependent**, **predicted**, or **outcome** variables, and the *x* variables the **independent**, **predictor**, or **input** variables.
  - When *X* is a matrix, we have what is known as **multiple linear regression**.

3

Typical Situations for Linear Regression Model

- Some typical situations where linear regression models can be used are:
  - Model the relationship between factors like rain, soil salinity, and the presence/absence of fertilizer in crop productivity. Then, answer questions such as: is the relationship linear? How strong is this relationship? Which factors have the strongest effect?
  - Find a relationship between average chocolate consumption by country and the number of Nobel laureates in that country, and then understand why this relationship could be spurious.
  - Predict the gas bill (used for heating and cooking) of your house by using the sun radiation from the local weather report. How accurate is this prediction?

4

Connection to Machine Learning

- Machine learning is an umbrella term for a collection of methods to automatically learn patterns in data, and then use what we learn to predict future data or to take decisions under uncertainty.
- ML and statistics are really intertwined subjects, and the connection becomes clear if you took a probabilistic perspective.
  - Using the ML terminology, we say a regression problem is an example of supervised learning. Under the machine learning framework, we have a regression problem when we want to learn a mapping from *X* to *Y*, with *Y* being a continuous variable. (Machine learning usually talks about features instead of variables.)
  - We say that the learning process is supervised because we know the values of the pairs; in some sense, we know the correct answer, and all the remaining questions are about how to generalize these observations (or this dataset) to any possible future observation, that is, to a situation when we know *X* but not *Y*.

5

Core of Linear Regression Models

- The chances are high that you are already familiar with the following equation:

$$y_i = \alpha + x_i\beta$$

- This equation says that there is a linear relation between the variable *x* and the variable *y*. The parameter β controls the **slope** of the linear relationship and thus is interpreted as the change in the variable *y* per unit change in the variable *x*. The other parameter, α, is known as the **intercept**, and tells us the value of y_i when $x_i = 0$. Graphically, the intercept is the value y_i of the point where the line intercepts the y axis.

6

Least Squares Fitting

- There are several ways to find the parameters for a linear model; one method is known as **least squares fitting**.
  - Least squares returns the values of α and β yielding the lowest average quadratic error between the observed *y* and the predicted *y*. Expressed in that way, the problem of estimating α and β is an optimization problem.
- An alternative to optimization is to generate a fully probabilistic model. Thinking probabilistically gives us several advantages:
  1. We can obtain the best values of α and β together with an estimation of the uncertainty we have about the parameter's values. Optimization methods require extra work to provide this information.
  2. Additionally, the probabilistic approach, especially when done using tools such as PyMC3, will give us the flexibility to adapt models to particular problems.

7

Linear Regression Model

- Probabilistically, a linear regression model can be expressed as follows:
$$y \sim N(\mu = \alpha + \beta x, \epsilon)$$
  - That is, the data vector *y* is assumed to be distributed as a Gaussian with a mean of $\alpha + \beta x$ and with a standard deviation of ϵ.
- A linear regression model is an extension of the Gaussian model where the mean is not directly estimated but rather computed as a linear function of a predictor variable and some additional parameters.
- Since we do not know the values of α, β, or ϵ, we have to set prior distributions for them. A reasonable generic choice would be:
$$\alpha \sim N(\mu_\alpha, \sigma_\alpha)$$
$$\beta \sim N(\mu_\beta, \sigma_\beta)$$
$$\epsilon \sim |N(0, \sigma_\epsilon)|$$

8

Priors

- For the prior over α, we can use a very flat Gaussian by setting the value σ_α to a relatively high value for the scale of the data. In general, we do not know where the intercept can be, and its value can vary a lot from one problem to another and for different domain knowledge. For many problems the author has worked on, α is usually centered around zero and with a σ_α no larger than 10, but this is just his experience with a small subset of problems and not something easy to transfer to other problems.
- Regarding the slope, it may be is easier to have a general idea of what to expect than for the intercept. For many problems, we can at least know the sign of the slope *a priori*; for example, we expect the variable weight to increase, on average, with the variable height.
- For ϵ, we can set σ_ϵ to a large value on the scale of the variable *y*, for example, ten times the value for its standard deviation.
- These very vague priors guarantee a very small effect of the prior on the posterior, which is easily overcome by the data.

9

Alternatives for ϵ

- A couple of alternatives to the Half-Gaussian distribution for ϵ are the Uniform or the half-Cauchy distributions.
  - The half-Cauchy distribution generally works well as a good regularizing prior and the Uniform distributions are generally not a very good choice unless you know that the parameter is truly restricted by hard boundaries.
- If we want to use really strong priors around some specific value for the standard deviation, we can use the gamma distribution.
  - The default parameterization of the gamma distribution can be a little bit confusing at first, but fortunately PyMC3 allows us to define it using the shape and rate (the most common parameterization) or the mean and standard deviation (a more intuitive parameterization, at least for newcomers).

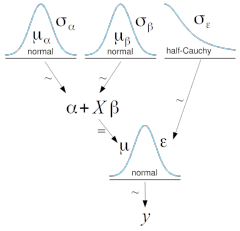
10

Connection between Point-Estimate and Bayesian Analysis

- To see how the gamma and other distributions look like, you can check out the PyMC3 documentation here: https://docs.pymc.io/api/distributions/continuous.html.
- The point-estimate obtained using the least squares method will agree with the **maximum a posteriori** (**MAP**) (the mode of the posterior) from a Bayesian simple linear regression with flat priors.

11

Kruschke Diagrams for Linear Regression Model



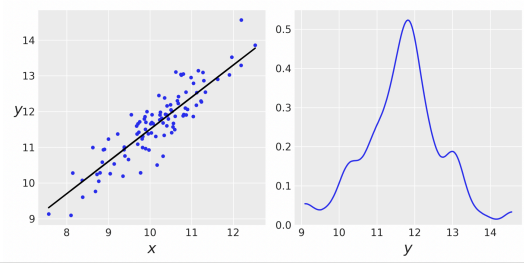
12

Synthetic Dataset

```
np.random.seed(1)
N = 100
alpha_real = 2.5
beta_real = 0.9
eps_real = np.random.normal(0, 0.5, size=N)
x = np.random.normal(10, 1, N) y_real = alpha_real + beta_real * x
y = y_real + eps_real
_, ax = plt.subplots(1,2, figsize=(8, 4))
ax[0].plot(x, y, 'C0.')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y', rotation=0)
ax[0].plot(x, y_real, 'k')
az.plot_kde(y, ax=ax[1])
ax[1].set_xlabel('y')
plt.tight_layout()
```

13

Figures for Synthetic Dataset



14

Build and Fit Model

- Note that μ is expressed in the model as a deterministic variable.

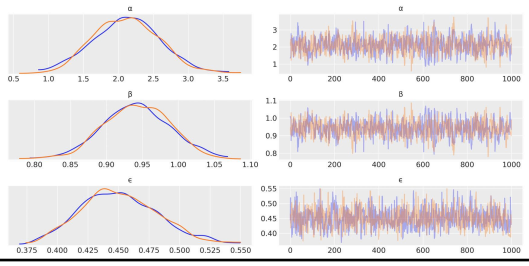
```
with pm.Model() as model_g:
        α = pm.Normal('α', mu=0, sd=10)
        β = pm.Normal('β', mu=0, sd=1)
        ε = pm.HalfCauchy('ε', 5)

        μ = pm.Deterministic('μ', α + β * x)
        y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)
trace_g = pm.sample(2000, tune=1000)
```

- Alternatively, instead of including a deterministic variable in the model, we can omit it. In such a case, the variable will still be computed but not saved in the trace. For example, we could have written the following:

```
y_pred = pm.Normal('y_pred', mu= α + β * x, sd=ε, observed=y)
```

15

az.plot_trace(trace_g, var_names=['α', 'β', 'ε'])

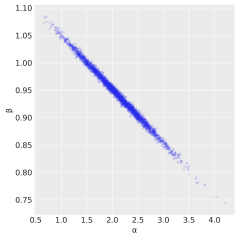


16

Linear Models and High Autocorrelation

- Linear models lead to posterior distribution where α and β are highly correlated. See the following code and the right figure for an example:

```
az.plot_pair(trace_g, var_names=['α', 'β'],
                      plot_kwargs={'alpha': 0.1})
```



17

Wheel of Fortune

- The correlation we are seeing in the figure in the previous slide is a direct consequence of our assumptions.
  - No matter which line we fit to our data, all of them should pass for one point, that is, the mean of the *x* variable and the mean of the *y* variable.
- Hence, the line fitting process is somehow equivalent to spinning a straight line fixed at the center of the data, like a wheel of fortune. An increase in the slope means a decrease of the intercept and vice versa. Both parameters are going to be correlated by definition of the model. Hence, the shape of the posterior (excluding ϵ) is a very diagonal space.
  - This can be problematic for samplers such as Metropolis-Hastings and to a lesser extent NUTS. For details of why this is true, see Chapter 8, Inference Engines.

18

Line is Constrained Due to Least Square Method

- The fact that the line is constrained to pass through the mean of the data is only true for the least square method (and its assumptions). Using Bayesian methods, this constrain is relaxed.
- Later in the examples, we will see that, in general, we get lines around the mean values of *x* and *y*, and not exactly through the exact mean. Moreover, if we use strong priors, we could end up with lines far away from the mean of *x* and *y*.
  - Nevertheless, the idea that the autocorrelation is related to the line spinning around a more or less defined point remains true, and that is all we need to understand regarding the correlation of the α and β parameters.

19

Modifying Data Before Running

- One simple way to remove the correlation α and β is to center the x variable. For each data point, we subtract the mean of the x variable ($\bar{x}$):
$$x' = x - \bar{x}$$
- As a result, x' will be centered at 0, and hence the pivot point when changing the slope is exactly the intercept, and thus the plausible parameter space is now more circular and less correlated.
- Centering data is not only a computational trick; it can also help in interpreting the results. The intercept is the value of y_i when $x_i = 0$. For many problems, this interpretation has no real meaning. For example, for quantities such as the height or weight, values of zero are meaningless and hence the intercept has no value in helping to make sense of the data. Instead, when centering the variables, the intercept is always the value of y_i for the mean value of x.

20

Estimate Precise Intercept

- For some problems, it may be useful to estimate the intercept precisely because it is not feasible to experimentally measure the value of $x_i = 0$ and so the estimated intercept can provide us with valuable information. However, extrapolations can be problematic, so be careful when doing this!

21

Centered Data or Uncentered Data?

- We may want to report the estimated parameters in terms of the centered data or in terms of the uncentered data, depending on our problem and audience. If we need to report the parameters as if they were determined in the original scale, we can do the following to put them back into the original scale:
$$\alpha = \alpha' - \beta' \bar{x}$$
- This correction is the result of the following algebraic reasoning:
$$y = \alpha' + \beta' x' + \epsilon$$
$$y = \alpha' + \beta'(x - \bar{x}) + \epsilon$$
$$y = \alpha' - \beta' \bar{x} + \beta' x + \epsilon$$
- Therefore, it follows that the first equation is true and also that:
$$\beta = \beta'$$

22

Standardizing Data

- We can even go further than centering *x* and transforming the data by **standardizing** it before running models. Standardizing is a common practice for linear regression models in statistics and ML since many algorithms behave better when the data is standardized. This transformation is achieved by centering the data and dividing it by the standard deviation. Mathematically we have:

$$x' = \frac{x - \bar{x}}{x_{sd}}$$

$$y' = \frac{y - \bar{y}}{y_{sd}}$$

23

Advantages of Standardizing Data

1. One advantage of standardizing the data is that we can always use the same weakly informative priors without having to think about the scale of the data. For standardized data, the intercept will always be around 0 and the slope will be restricted to the interval [-1, 1].
2. Additionally, standardizing the data allow us to talk in terms of Z-scores, that is, in units of standard deviations. If someone says the value of a parameter is -1.3 in Z-score units, we automatically know that the value in question is 1.3 standard deviations below the mean, irrespective of the actual value of the mean or the actual value of the standard deviation of the data. A change in one Z-score unit is a change in one standard deviation, whatever the scale of the original data is. This can be very useful when working with several variables; having all of the variables in the same scale can simplify the interpretation of the data.

24

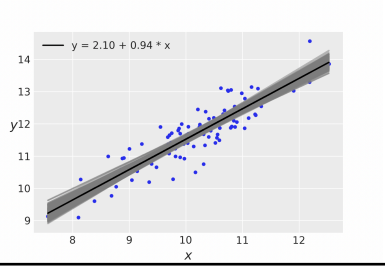
Interpreting and Visualizing Posterior

- We can the explore the posterior using ArviZ functions like plot_trace and summary, or we can use our own functions.
- For a linear regression, it could be useful to plot the average line that fits the data, together with the average mean values of α and β. To reflect the posterior's uncertainty, we can use semitransparent lines that have been sampled from the posterior:

```
plt.plot(x, y, 'C0.')
alpha_m = trace_g['α'].mean()
beta_m = trace_g['β'].mean()

draws = range(0, len(trace_g['α']), 10)
plt.plot(x, trace_g['α'][draws] + trace_g['β'][draws] * x[:, np.newaxis], c='gray', alpha=0.5)
plt.plot(x, alpha_m + beta_m * x, c='k', label=f'y = {alpha_m:.2f} + {beta_m:.2f} * x')

plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.legend()
```

25

Posterior Predictive Check I



26

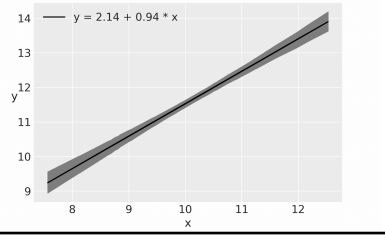
More Visual Hints

- Notice that uncertainty is lower in the middle, although it is not reduced to a single point, that is, the posterior is compatible with lines not passing exactly through the mean of the data, as we have already mentioned.
- Having the semitransparent lines is perfectly fine, but we may want to add a *cool-factor* to the plot and instead use a semitransparent band to illustrate the **Highest-Posterior Density** (**HPD**) interval of μ. In fact, this was the main reason we defined the deterministic variable μ in the model. Having this variable simplifies the following code:

```
plt.plot(x, alpha_m + beta_m * x, c='k', label=f'y = {alpha_m:.2f} + {beta_m:.2f} * x')
sig = az.plot_hpd(x, trace_g['μ'], credible_interval=0.98, color='k') plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.legend()
```

27

Posterior Predictive Check II



28

Even More Visual Hints

- One more option is to plot the HPD (for example, 94% and 50%) of the predicted data $\hat{y}$, that is, where we expect to see the 94% and 50% of future data, according to our model. For the figure in the next slide, we are going to use a darker gray for the HPD 50 and a lighter gray for the HPD 95.
- Getting the posterior predictive samples is easy in PyMC3 using the sample_posterior_predictive() function:

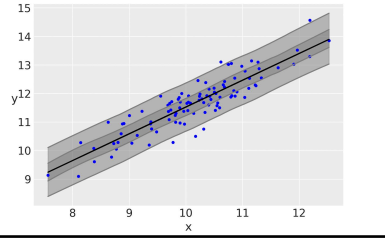
```
ppc = pm.sample_posterior_predictive(trace_g, samples=2000, model=model_g)
```

- Now, we can plot the results:

```
plt.plot(x, y, 'b.')
plt.plot(x, alpha_m + beta_m * x, c='k', label=f'y = {alpha_m:.2f} + {beta_m:.2f} * x')

az.plot_hpd(x, ppc['y_pred'], credible_interval=0.5, color='gray')
az.plot_hpd(x, ppc['y_pred'], color='gray')

plt.xlabel('x')
plt.ylabel('y', rotation=0)
```

29

Posterior Predictive Check III



30

az.plot_hpd

- The function az.plot_hpd is a helper function that we can use to plot a HPD interval for linear regressions. By default this function smooth the interval. Try passing the argument smooth=False to see what happens.

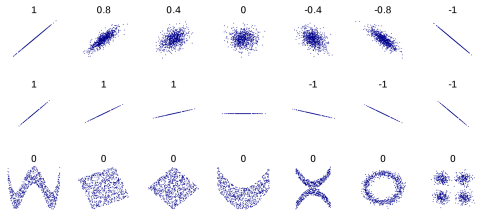
31

Pearson correlation coefficient

- Sometimes, we want to measure the degree of (linear) dependence between two variables. The most common measure of the linear correlation between two variables is the Pearson correlation coefficient, often identified with a lowercase r. When $r = +1$, we have a perfect positive linear correlation, that is, an increase of one variable predicts an increase of the other. When we have $r = -1$, we have a perfect negative linear correlation and the increase of one variable predicts a decrease of the other. When $r = 0$, we have no linear correlation. As a general rule, we will get intermediate values. It's important to always have two aspects in mind: the Pearson correlation coefficient says nothing about non-linear correlations. We should not confuse r with the slope of a regression. The image in the next slide (from Wikipedia) is a good one to have at hand:

32

Different Datasets and Their Corresponding r



33

Relationship Between r and β

- Part of the confusion between r and the slope of a regression may be explained by the following relationship:

$$r = \beta \frac{\sigma_x}{\sigma_y}$$

- That is, the slope (β) and the Pearson correlation coefficient (r) have the same value, but only when the standard deviation of *x* and *y* are equal. Notice that it is true, for example, when we standardize the data. To further clarify:
  - The Pearson correlation coefficient (r) is a measure of the degree of correlation between two variables and is always restricted to the interval [-1, 1]. It does not matter about the scale of the data.
  - The slope of a linear regression (β) indicates how much *y* change per unit change of *x*, and can take any real value.

34

Determination Coefficient

- The Pearson coefficient is related to a quantity known as the **determination coefficient**, and, for a linear regression model, this is just the square of the Pearson coefficient, that is, r^2 (or sometimes R^2). This is pronounced as *r squared* and can be defined as the variance of the predicted values divided by the variance of the data. Therefore, it can be interpreted as the proportion of the variance in the dependent variable that is predicted from the independent variable. For Bayesian linear regression, the variance of the predicted values can be larger than the variance of the data and this will lead to an R^2 larger than 1, then a good solution is to define R^2 as follows:

$$R^2 = \frac{\mathbf{V}_{n=1}^{N} \mathbf{E}[\hat{y}^s]}{\mathbf{V}_{n=1}^{N} \mathbf{E}[\hat{y}^s] + \mathbf{V}_{n=1}^{S}(\hat{y}^s - y)}$$

35

Notation in Equation

- In the above equation, $E[\hat{y}^s]$ is the expected (or mean) value $\hat{y}$ over s posterior samples.
- This is the variance of the predicted values divided by the variance of the predicted values plus the errors (or residuals). This definition has the advantage of ensuring R^2 is restricted to the interval [0, 1].
- The easiest way to compute R^2 is to use the r2_score() function from ArviZ. We need the observed values of y and the predicted values of $\hat{y}$. Remember that we can get $\hat{y}$ from sample_posterior_predictive():

```
az.r2_score(y, ppc['y_pred'])
```

- By default, this function will return R^2 (0.8, for this example) and the standard deviation (0.03).

36

Pearson Coefficient from Multivariate Gaussian

- Another way to compute the Pearson coefficient is by estimating the covariance matrix of a multivariate Gaussian distribution. A multivariate Gaussian distribution is the generalization of the Gaussian distribution to more than one dimension. Let's focus on the case of two dimensions because that is what we are going to use right now. Generalizing to higher dimensions is almost trivial once we understand the bivariate case. To fully describe a bivariate Gaussian distribution, we need two means (or a vector with two elements), one per each marginal Gaussian. We also need two standard deviations, right? Well, not exactly; we need a 2 x 2 covariance matrix, which looks like this:

$$\Sigma = \begin{bmatrix} \sigma_{x_1}^2 & \rho\sigma_{x_1}\sigma_{x_2} \\ \rho\sigma_{x_1}\sigma_{x_2} & \sigma_{x_2}^2 \end{bmatrix}$$

37

Notation

- Here, Σ is the Greek capital sigma letter and it is common practice to use it to represent the covariance matrix. In the main diagonal, we have the variances of each variable, expressed as the square of their standard deviations σ_{x_1} and σ_{x_2}. The rest of the elements in the matrix are the covariances (the variance between variables), which are expressed in terms of the individual standard deviations and ρ, the Pearson correlation coefficient between variables. Notice that we have a single ρ because we have only two dimensions (or variables). For three variables, we would have three Pearson coefficients.
- The following code generates contour plots for bivariate Gaussian distributions with both means fixed at (0, 0). One of the standard deviations is fixed $\sigma_{x_1} = 1$, while the other takes the values 1 or 2 and different values for the Pearson correlation coefficient:

38

Code

```
sigma_x1 = 1
sigmas_x2 = [1, 2]
rhos = [-0.90, -0.5, 0, 0.5, 0.90]

k, l = np.mgrid[-5:5:.1, -5:5:.1]
pos = np.empty(k.shape + (2,))
pos[:, :, 0] = k
pos[:, :, 1] = l

f, ax = plt.subplots(len(sigmas_x2), len(rhos), sharex=True, sharey=True, figsize=(12, 6),
                     constrained_layout=True)
```

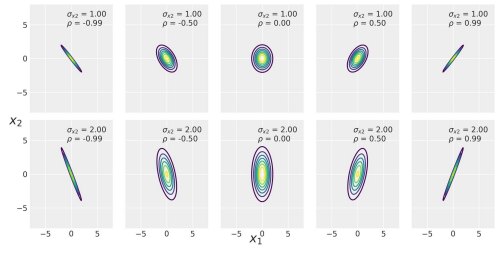
39

Code (Cont.)

```
for i in range(2):
  for j in range(5):
        sigma_x2 = sigmas_x2[i]
        rho = rhos[j]
        cov = [[sigma_x1**2, sigma_x1*sigma_x2*rho], [sigma_x1*sigma_x2*rho, sigma_x2**2]]
        rv = stats.multivariate_normal([0, 0], cov)
        ax[i, j].contour(k, l, rv.pdf(pos))
        ax[i, j].set_xlim(-8, 8)
        ax[i, j].set_ylim(-8, 8)
        ax[i, j].set_yticks([-5, 0, 5])
        ax[i, j].plot(0, 0, label=f'$\\sigma_{{x2}}$ = {sigma_x2:3.2f}\n$\\rho$ = {rho:3.2f}', alpha=0)
        ax[i, j].legend()
f.text(0.5, -0.05, 'x_1', ha='center', fontsize=18)
f.text(-0.05, 0.5, 'x_2', va='center', fontsize=18, rotation=0)
```

40

The Plot



41

Wishart Distribution and LKJ Prior

- Now that we know the multivariate Gaussian distribution, we can use it to estimate the Pearson correlation coefficient. Since we do not know the values of the covariance matrix, we have to put priors over it. One solution is to use the Wishart distribution, which is the conjugate prior of the inverse covariance matrix of a multivariate normal. The Wishart distribution can be considered as the generalization to higher dimensions of the gamma distribution we saw earlier or also the generalization of the X^2 distribution.
- A second option is to use the LKJ prior (see https://docs.pymc.io/notebooks/LKJ.html for details). This is a prior for the correlation matrix (and not the covariance matrix), which may be convenient, given that it is generally more useful to think in terms of correlations.
- We are going to explore a third option and we are going to put priors directly for σ_{x_1}, σ_{x_2}, and ρ, and then use those values to manually build the covariance matrix:

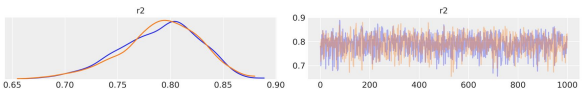
42

Code

```
data = np.stack((x, y)).T
with pm.Model() as pearson_model:
        μ = pm.Normal('μ', mu=data.mean(0), sd=10, shape=2)
        σ_1 = pm.HalfNormal('σ_1', 10)
        σ_2 = pm.HalfNormal('σ_2', 10)
        ρ = pm.Uniform('ρ', -1., 1.)
        r2 = pm.Deterministic('r2', ρ**2)
        cov = pm.math.stack(([σ_1**2, σ_1*σ_2*ρ], [σ_1*σ_2*ρ, σ_2**2]))
```

43

az.plot_trace(trace_p, var_names=['r2'])
(omit all variables other than r2)



44

Comparison of Distributions of r^2

- We can see that the distribution of *r2* values is around the value we got in the previous example using the r2_score function from ArviZ. Maybe an easier way to compare the value obtained from the multivariate Gaussian with the previous result is by using summary. As you can see, we got a pretty good match:

```
az.summary(trace_p, var_names=['r2'])
```

	mean	sd	mc error	hpd 3%	hpd 97%	eff_n	r_hat
r2	0.79	0.04	0.0	0.72	0.86	839.0	1.0

45

Robust Linear Regression

- Assuming that the data follows a Gaussian distribution, it is perfectly reasonable in many situations. By assuming Gaussianity, we are not necessarily saying data is really Gaussian; instead, we are saying that it is a reasonable approximation for a given problem. The same applies to other distributions. As we saw in the previous chapter, sometimes, this Gaussian assumption fails, for example, in the presence of outliers. We learned that using a Student's t-distribution is a way to effectively deal with outliers and get a more robust inference. The very same idea can be applied to linear regression.

46

Anscombe Quartet Dataset

- To exemplify the robustness that a Student's t-distribution brings to a linear regression, we are going to use a very simple and nice dataset: the third data group from the Anscombe quartet. If you do not know what the Anscombe quartet is, remember to check it later at Wikipedia (https://en.wikipedia.org/wiki/Anscombe%27s_quartet). We can upload it using pandas. We are going to center the data, just to make things easier for the sampler—even a cool sampler like NUTS needs a little help from time to time:

```
ans = pd.read_csv('../data/anscombe.csv')
x_3 = ans[ans.group == 'III']['x'].values
y_3 = ans[ans.group == 'III']['y'].values
x_3 = x_3 - x_3.mean()
```

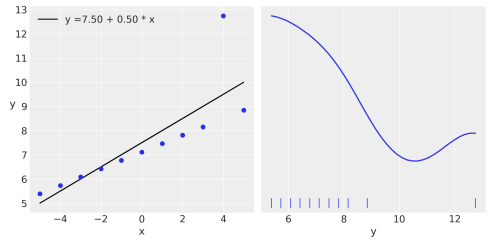
47

What This Little Tiny Dataset Looks Like?

```
_, ax = plt.subplots(1, 2, figsize=(10, 5))
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]
ax[0].plot(x_3, (alpha_c + beta_c * x_3), 'k', label=f'y ={alpha_c:.2f} + {beta_c:.2f} * x')
ax[0].plot(x_3, y_3, 'C0o')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y', rotation=0)
ax[0].legend(loc=0)
az.plot_kde(y_3, ax=ax[1], rug=True)
ax[1].set_xlabel('y')
ax[1].set_yticks([])
plt.tight_layout()
```

48

The Plot



49

Rewrite Previous Model

- Now, we are going to rewrite the previous model (model_g), but this time we are going to use a Student's t-distribution instead of a Gaussian. This change also introduces the need to specify the value of ν, the normality parameter. If you do not remember the role of this parameter, check the slides of Programming Probabilistically before continuing.

50

Shifted Exponential

- In the following model, we are using a shifted exponential to avoid values of ν close to zero. The non-shifted exponential puts too much weight on values close to zero. This may be fine for data with no to moderate outliers, but for data with extreme outliers (or data with a few bulk points), like in the Anscombe's third dataset, it is better to avoid such low values. Take this, as well as other priors recommendations, with a pinch of salt. The defaults are good starting points, but there's no need to stick to them. Other common priors for ν are gamma(2, 0.1) or gamma(mu=20, sd=15).

51

Code

```
with pm.Model() as model_t:
        α = pm.Normal('α', mu=y_3.mean(), sd=1)
        β = pm.Normal('β', mu=0, sd=1)
        ε = pm.HalfNormal('ε', 5)
        ν_ = pm.Exponential('ν_', 1/29)
        ν = pm.Deterministic('ν', ν_ + 1)

        y_pred = pm.StudentT('y_pred', mu=α + β * x_3, sd=ε, nu=ν, observed=y_3)
    trace_t = pm.sample(2000)
```

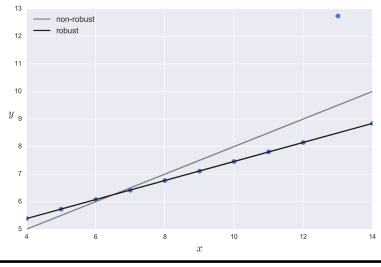
52

Robust Fit According to model_t and Non-Robust Fit According to SciPy's Linregress

```
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]

plt.plot(x_3, (alpha_c + beta_c * x_3), 'k', label='non-robust', alpha=0.5)
plt.plot(x_3, y_3, 'C0o')
alpha_m = trace_t['α'].mean()
beta_m = trace_t['β'].mean()
plt.plot(x_3, alpha_m + beta_m * x_3, c='k', label='robust')
plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.legend(loc=2)
plt.tight_layout()
```

53

The Plot



54

Plot Explanation

- While the non-robust fit tries to compromise and include all points, the *robust* Bayesian model, model_t, automatically *discards* one point and fits a line that passes exactly through all the remaining points. Although this is a very peculiar dataset, but the message remains for *more real* and complex ones. A Student's t-distribution, due to its heavier tails, is able to give *less importance* to points that are far away from the bulk of the data.

55

Contemplate Values of Parameters

- Before moving on, take a moment to contemplate the values of the parameters (We omit the intermediate $\nu\_$ parameter as it is not of direct interest):

```
az.summary(trace_t, var_names=varnames)
```

	mean	sd	mc error	hpd 3%	hpd 97%	eff_n	r_hat
α	7.11	0.00	0.0	7.11	7.12	2216.0	1.0
β	0.35	0.00	0.0	0.34	0.35	2156.0	1.0
ϵ	0.00	0.00	0.0	0.00	0.01	1257.0	1.0
ν	1.21	0.21	0.0	1.00	1.58	3138.0	1.0

56

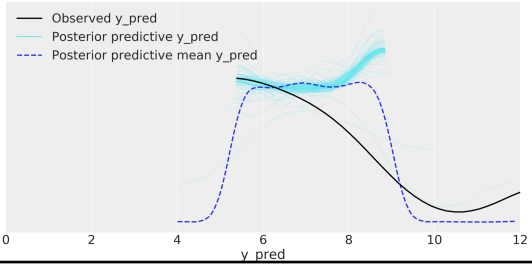
Narrow Parameter Range

- As you can see, the values of α, β, and ϵ are very narrowly defined and even more so with ϵ, which is basically 0. This is totally reasonable, given that we are fitting a line to a perfectly aligned set of points (if we ignore the outlier point).
- Let's run a posterior predictive check to explore how well our model captures the data. We can let PyMC3 do the hard work of sampling from the posterior for us:

```
ppc = pm.sample_posterior_predictive(trace_t, samples=200, model=model_t,
                                     random_seed=2)
data_ppc = az.from_pymc3(trace=trace_t, posterior_predictive=ppc)
ax = az.plot_ppc(data_ppc, figsize=(12, 6), mean=True)
plt.xlim(0, 12)
```

57

Posterior Predictive Check



58

Plot Explanation

- For the bulk of the data, we get a very good match. Also notice that our model predicts values away from the bulk to both sides and not just above the bulk. For our current purposes, this model is performing just fine and it does not need further changes. Nevertheless, notice that for some problems, we may want to avoid this. In such a case, we should probably go back and change the model to restrict the possible values of y_pred to positive values.

59

Hierarchical Linear Regression

- In the previous discussion, we learned about the rudiments of hierarchical models. We can apply this concept to linear regression as well. This allows models to deal with inferences at the group level and estimations above the group level. As we already saw, this is done by including **hyperpriors**.

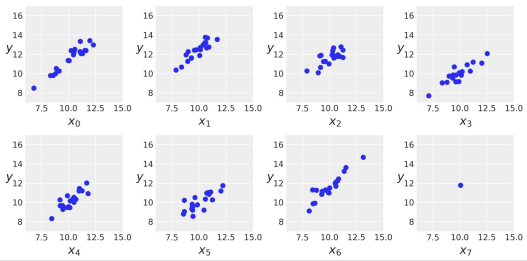
60

Creating 8 Related Data Groups (Including One Group with Single Data Point)

```
N = 20
M= 8
idx = np.repeat(range(M-1), N)
idx = np.append(idx, 7)
np.random.seed(314)

alpha_real = np.random.normal(2.5, 0.5, size=M)
beta_real = np.random.beta(6, 1, size=M)
eps_real = np.random.normal(0, 0.5, size=len(idx))

y_m = np.zeros(len(idx))
x_m = np.random.normal(10, 1, len(idx))
y_m = alpha_real[idx] + beta_real[idx] * x_m + eps_real

_, ax = plt.subplots(2, 4, figsize=(10, 5), sharex=True, sharey=True)
ax = np.ravel(ax)
j, k = 0, N
for i in range(M):
        ax[i].scatter(x_m[j:k], y_m[j:k])
        ax[i].set_xlabel('x_{i}')
        ax[i].set_ylabel('y_{i}', rotation=0, labelpad=15)
        ax[i].set_xlim(6, 15)
        ax[i].set_ylim(7, 17)
        j += N
        k += N
plt.tight_layout()
```

61

The Plot of 8 Groups



62

Non-Hierarchical Model

- Now, we are going to center the data before feeding it to the model:

```
x_centered = x_m - x_m.mean()
```

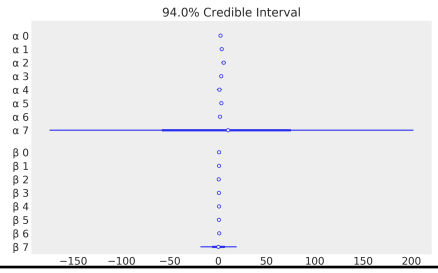
- First we are going to fit a non-hierarchical model, just as we have already seen. The only difference is that we are now going to include code to re-scale α to the original scale:

```
with pm.Model() as unpooled_model:
        α_tmp = pm.Normal('α_tmp', mu=0, sd=10, shape=M)
        β = pm.Normal('β', mu=0, sd=10, shape=M)
        ε = pm.HalfCauchy('ε', 5)
        ν = pm.Exponential('ν', 1/30)

        y_pred = pm.StudentT('y_pred', mu=α_tmp[idx] + β[idx] * x_centered, sd=ε, nu=ν,
                             observed=y_m)
        α = pm.Deterministic('α', α_tmp - β * x_m.mean())
trace_up = pm.sample(2000)
```

- As we can see in the following figure, the estimations for the α_7 and β_7 parameters are very very wide compared to the rest of the $\alpha_{0:6}$ and $\beta_{0:6}$ parameters:

63

az.plot_forest(trace_up, var_names=['α', 'β'], combined=True)



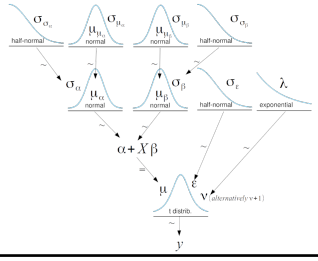
64

What Is Going On?

- You may have already guessed what is going on—it does not make sense to try to fit a line through a single point. We need at least two points, otherwise the parameters α and β are unbounded. That is totally true unless we provide some more information; we can do this by using priors. Putting a strong prior for α can lead to a well-defined set of lines, even for one data point. Another way to convey information is by defining hierarchical models, since hierarchical models allow information to be shared between groups, shrinking the plausible values of the estimated parameters. This becomes very useful in cases where we have groups with sparse data. In this example, we have taken that sparsity of the data to the extreme—a group with a single data point!

65

Kruschke Diagram of Our Hierarchical Model



66

Differences Compared to Previous Models

- In the following PyMC3 model, the main differences compared to previous models are:
  - We add hyperpriors.
  - We also add a few lines to transform the parameters back to the original un-centered scale. Remember that this is not mandatory; we can keep the parameters in the transformed scale as long as we keep that in mind when interpreting the results.

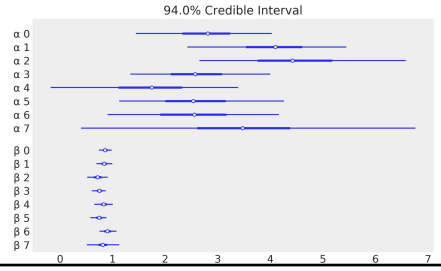
67

Code for Hierarchical Model

```
with pm.Model() as hierarchical_model:
        # hyper-priors
        α_μ_tmp = pm.Normal('α_μ_tmp', mu=0, sd=10)
        α_σ_tmp = pm.HalfNormal('α_σ_tmp', 10)
        β_μ = pm.Normal('β_μ', mu=0, sd=10)
        β_σ = pm.HalfNormal('β_σ', sd=10)

        # priors
        α_tmp = pm.Normal('α_tmp', mu=α_μ_tmp, sd=α_σ_tmp, shape=M)
        β = pm.Normal('β', mu=β_μ, sd=β_σ, shape=M)
        ε = pm.HalfCauchy('ε', 5)
        ν = pm.Exponential('ν', 1/30)
        y_pred = pm.StudentT('y_pred', mu=α_tmp[idx] + β[idx] * x_centered, sd=ε, nu=ν,
observed=y_m)
        α = pm.Deterministic('α', α_tmp - β * x_m.mean())
        α_μ = pm.Deterministic('α_μ', α_μ_tmp - β_μ * x_m.mean())
        α_σ = pm.Deterministic('α_sd', α_σ_tmp - β_μ * x_m.mean())
trace_hm = pm.sample(1000)
```

68

az.plot_forest(trace_hm, var_names=['α', 'β'], combined=True)



69

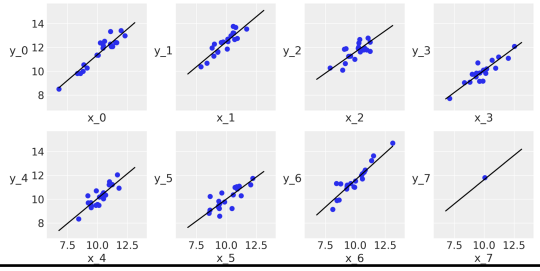
Show Parameters of Both Models

- A good way to compare models using az.plot_forest() is to show the parameters of both models (unpooled_model, hierarhical_model) simultaneously in the same plot. To do this, you just need to pass a list of traces.
- To better understand what the model is capturing about the data, let's plot the fitted lines for each of the eight groups:

```
_, ax = plt.subplots(2, 4, figsize=(10, 5), sharex=True, sharey=True, constrained_layout=True)
ax = np.ravel(ax)
j, k = 0, N
x_range = np.linspace(x_m.min(), x_m.max(), 10)
for i in range(M):
        ax[i].scatter(x_m[j:k], y_m[j:k])
        ax[i].set_xlabel(f'x_{i}')
        ax[i].set_ylabel(f'y_{i}', labelpad=17, rotation=0)
        alpha_m = trace_hm['α'][:, i].mean()
        beta_m = trace_hm['β'][:, i].mean()
        ax[i].plot(x_range, alpha_m + beta_m * x_range, c='k', label=f'y = {alpha_m:.2f} + {beta_m:.2f} * x')
        plt.xlim(x_m.min()-1, x_m.max()+1)
        plt.ylim(y_m.min()-1, y_m.max()+1)
        j += N
        k += N
```

70

The Plot



71

Plot Explanation

- Using a hierarchical model, we were able to fit a line to a single data point, as you can see in the figure in previous slide. At first, this may sound weird or even fishy, but this is just a consequence of the structure of the hierarchical model. Each line is informed by the lines of the other groups, thus we are not truly adjusting a line to a single point. Instead, we are adjusting a line to a single point that's been informed by the points in the other groups.

72

Correlation, Causation, and Messiness of Life

- Suppose we want to predict how much we are going to pay for gas to heat our home during winter and suppose we know the amount of sun radiation in the area we live. In this example, the sun radiation is going to be the independent variable, x, and the bill is the dependent variable, y. It is very important to note that there is nothing forbidding us to invert the question and ask about the amount of sun radiation, given the bill. If we establish a linear relationship (or any other relation, for that matter), we can go from x to y or vice versa. We call a variable independent because its value cannot be predicted by the model; instead, it is an input of the model and the dependent variable is the output. We say that the value of one variable depends on the value of the other because we build a model specifying such a dependency. We are not establishing a causal relationship between variables and we are not saying x causes y. Always remember the following mantra, correlation does not imply causation. Let's develop this idea a little bit more. Even if we are able to predict the gas bill of a home from the sun radiation and the sun radiation from the gas bill, we can agree that it isn't true that we can control the amount of radiation emitted by the sun by changing the thermostat of our house! However, it is true that higher sun radiation can be related to a lower gas bill.

73

Causation is More Than Correlation

- It is therefore important to remember that the statistical model we are building is one thing and the physical mechanism relating the variables is another. To establish that a correlation can be interpreted in terms of causation, we need to add a plausible physical mechanism to the description of the problem; a correlation is simply not enough. A very nice and amusing page showing clear examples of correlated variables with no causal relationship can be found at http://www.tylervigen.com/spurious-correlations.

74

Correlation Useful in Establishing Causation

- Is a correlation useless when it comes to establishing a causal link? Not at all—a correlation can, in fact, support a causal link if we perform a carefully designed experiment. For example, we know that global warming is highly correlated to the increasing levels of atmospheric CO_2. From this observation alone, we cannot conclude whether higher temperatures are causing an increase in the levels of CO_2 or if the higher levels of the carbonic gas are increasing the temperature.

75

Experiments For Varification

- Even more, it could happen that there is a third variable that we are not taking into account, and this variable is producing both higher temperatures and higher levels of CO_2. However—and pay attention to this—we can do an experiment to gain insight into this problem. One possible experiment could be the following; we build a set of glass tanks filled with different quantities of CO_2. We can have one with regular air (that contains ~0.04% of CO_2) and the others with and increasing amount of CO_2. We then let these tanks receive sun light for, let's say, three hours. If we do this, we will verify that tanks with higher levels of CO_2 have higher final temperatures. Hence, we will conclude that indeed CO_2 is a greenhouse effect gas. Using the same experiment, we can also measure the concentration of CO_2 at the end of the experiment to check that temperature does not cause the CO_2 level to increase, at least not from air. It is this experimental setting together with statistical models that gives evidence in favor of CO_2 emissions contributing to global warming.

76

Reality is Complicated!

- Another important aspect we will discuss following this example is that, even when the sun radiation and the gas bill are connected and maybe the sun radiation can be used to predict the gas bill, the relationship is more complicated, and other variables are involved. In fact, higher temperature can contribute to higher levels of CO_2 because oceans are a reservoir of CO_2, and CO_2 is less soluble in water when temperatures increase. Also, a higher sun radiation means that more energy is delivered to a home. Part of that energy is reflected and part is turned into heat, part of the heat is absorbed by the house, and part is lost to the environment. The amount of heat lost depends upon several factors, such as the outside temperature and the speed of the wind. Then, we have the fact that the gas bill could also be affected by other factors, such as the international price of oil and gas, the costs/profits for the company (and its level of *greediness*), and also how tightly the government regulates the company.

77

Statistical Models Simplify Messy Reality

- In summary, life is messy, problems are not generally simple to understand, and context is always important. Statistical models can help us achieve better interpretations, reduce the risk of making nonsensical statements, and get better predictions, but none of this is automatic.

78

Polynomial Regression

- Now, we are going to learn how to fit curves using linear regression. One way to fit curves using a linear regression model is by building a polynomial, like this:

$$\mu = \beta_0 x^0 + \beta_1 x^1 + \cdots + \beta_m x^m$$

- If we pay attention, we can see that the simple linear model is hidden in this polynomial. To uncover it, all we need to do is to make all the β_n coefficients higher than one exactly zero. Then, we will get:

$$\mu = \beta_0 + \beta_1 x^1$$

- Polynomial regression is still linear regression; the linearity in the model is related to how the parameters enter the model, not the variables. Let's try building a polynomial regression of degree 2:

$$\mu = \beta_0 + \beta_1 x^1 + \beta_2 x^2$$

- The third term controls the curvature of the relationship.

79

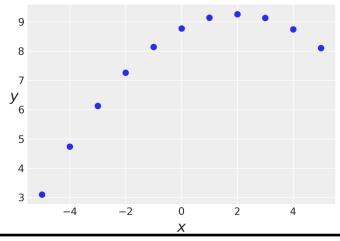
The 2nd Anscombe Quartet Dataset

- As a dataset, we are going to use the second group of the Anscombe quartet:

```
x_2 = ans[ans.group == 'II']['x'].values
y_2 = ans[ans.group == 'II']['y'].values
x_2 = x_2 - x_2.mean()
plt.scatter(x_2, y_2)
plt.xlabel('x')
plt.ylabel('y', rotation=0)
```

80

The Plot



81

Build PyMC3 Model

```
with pm.Model() as model_poly:
        α = pm.Normal('α', mu=y_2.mean(), sd=1)
        β1 = pm.Normal('β1', mu=0, sd=1)
        β2 = pm.Normal('β2', mu=0, sd=1)
        ε = pm.HalfCauchy('ε', 5)

        mu = α + β1 * x_2 + β2 * x_2**2
        y_pred = pm.Normal('y_pred', mu=mu, sd=ε, observed=y_2)
trace_poly = pm.sample(2000)
```

82

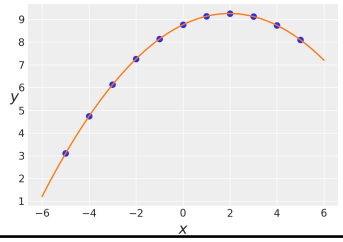
Plot Results

- Once again, we are going to omit some checks and summaries and just plot the results, which will be a nice curved line fitting the data almost with no errors. Take into account the minimalist nature of the dataset:

```
x_p = np.linspace(-6, 6)
y_p = trace_poly['α'].mean() + trace_poly['β1'].mean() * x_p + trace_poly['β2'].mean() * x_p**2
plt.scatter(x_2, y_2)
plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.plot(x_p, y_p, c='C1')
```

83

The Plot



84

Interpreting Parameters of Polynomial Regression

- One of the problems of polynomial regression is the interpretation of the parameters. If we want to know how *y* changes per unit change of *x*; we cannot just check the value of β_1, because β_2 and higher coefficients, if present, have an effect on such a quantity. Therefore, the β coefficients are no longer slopes, they are something else. In the previous example, β_1 is positive and hence the curve begins with a positive slope, but β_2 is negative and hence, after a while, the line begins to curve downwards. So, it is like we have two forces at play, one pushing the line up and the other down. The interplay depends on the value of *x*. When $x \leq 11$ (on the original scale or 2 on the centered scale), the dominating contribution comes from β_1, and when $x \geq 11$ β_2 dominates.

85

Domain Knowledge Matters

- The problem of interpreting the parameters is not just a mathematical problem. If this were the case, we could solve it by careful inspection and understanding of the model. The problem is that, in many cases, the parameters are not translated to meaningful quantities in our domain knowledge. We cannot relate them with the metabolic rate of a cell or the energy emitted by a distant galaxy or the number of bedrooms in a house. They are just nobs we can tweak to improve the fit but without a clear physical meaning. In practice, most people will agree that polynomials of order higher than 2 or 3 are not generally very useful models and alternatives are preferred, such as Gaussian processes, which is the main subject of Chapter 7, Gaussian Processes.

86

Is Polynomial Regression The Ultimate Model?

- As we saw, we can think of the line as a sub-model of the parabola when β_2 is equal to zero, and a line is also a sub-model of a cubic model when β_2 and β_3 are equal to zero. Of course, the parabola is a sub-model of the cubic one when β_3 is zero. OK, we will stop here, but I think you already notice the pattern. This suggests that we can, in principle, use polynomial regression to fit an arbitrary complex model. We just built a polynomial with the right order. We could do this by increasing the order one by one until we do not observe any improvement on the fit, or we could build an infinite order polynomial and then somehow make all the irrelevant coefficients zero until we get a perfect fit of our data. To test this idea, we can start with a very simple example. Let's use the quadratic model to fit the third group of the Anscombe dataset.

87

Overfitting

- OK, if you really did the exercise, you will have observed by yourself that it is possible to use a quadratic model to fit a line. While it may seem that the previous simple experiment validates the idea of building an infinite order polynomial to the fit data, we should curb our enthusiasm. In general, using a polynomial to fit data is not the best idea. Why? Because it does not matter which data we have. In principle, it is always possible to find a polynomial to fit the data perfectly! In fact, it is pretty easy to compute the exact order the polynomial should have. Why is fitting data perfectly problematic? Well that's the subject of Chapter 6, *Model Comparison*, but spoiler alert! A model that fits your current data perfectly will, in general, do a very poor job at fitting/describing unobserved data. The reason for this is that any real dataset will contain noise and sometimes (hopefully) an interesting pattern. An arbitrary over-complex model will fit the noise, leading to poor predictions. This is known as **overfitting**, and is a pervasive phenomena in statistics and ML. Polynomial regression makes for a convenient straw man when it comes to overfitting because it is easy to see the problem. This generates intuition that we can translate to more complex models, which can lead to overfitting without us really noticing. Part of the job, when analyzing data, is to be sure that models are not overfitting it. We will discuss this topic in detail in Chapter 6, *Model Comparison*.

88

Multiple Linear Regression

- So far, we have been working with one dependent variable and one independent variable. Nevertheless, it is not unusual to have several independent variables that we want to include in our model. Some examples could be:
  - Perceived quality of wine (dependent) and acidity, density, alcohol level, residual sugar, and sulphates content (independent variables)
  - A student's average grades (dependent) and family income, distance from home to school, and mother's education (categorical variable)
- We can easily extend the simple linear regression model to deal with more than one independent variable. We call this model multiple linear regression or less often multivariable linear regression (not to be confused with multivariate linear regression, the case where we have multiple dependent variables).

89

Multiple Linear Regression Model

- In a multiple linear regression model, we model the mean of the dependent variable as follows:

$$\mu = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$

- Notice that this looks similar to a polynomial regression, but is not exactly the same. For multiple linear regression, we have different variables instead of successive powers of the same variable. From the point of view of multiple linear regression, we can say that a polynomial regression is like a multiple linear regression but with made-up variables.

90

Linear Algebra Notation

- Using linear algebra notation, we can write a shorter version:

$$\mu = \alpha + X\beta$$

- Here, β is a vector of coefficients of length m, that is, the number of dependent variables. The variable X is a matrix of size $m \times n$ if n is the number of observations and m is the number of independent variables. If you are a little rusty with your linear algebra, you may want to check the Wikipedia article about the dot product between two vectors and its generalization to matrix multiplication. Basically what you need to know is that we are just using a shorter and more convenient way to write our model:

$$X\beta = \sum_{i=1}^{m} \beta_i x_i = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$

- Using the simple linear regression model, we find a straight line that (hopefully) explains our data. Under the multiple linear regression model we find, instead, a hyperplane of dimension m. Thus, the multiple linear regression model is essentially the same as the simple linear regression model, the only difference being that now β is a vector and X is a matrix.

91

Define Our Data

```
np.random.seed(314)
N = 100
alpha_real = 2.5
beta_real = [0.9, 1.5]
eps_real = np.random.normal(0, 0.5, size=N)

X = np.array([np.random.normal(i, j, N) for i, j in zip([10, 2], [1, 1.5])]).T
X_mean = X.mean(axis=0, keepdims=True)
X_centered = X - X_mean
y = alpha_real + np.dot(X, beta_real) + eps_real
```

92

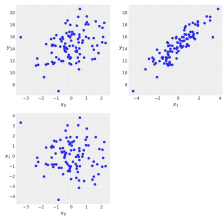
Convenient Function to Plot Scatter Plots

- Three plots: two between each independent variable, and the dependent variable, and the last one between both dependent variables.

```
def scatter_plot(x, y):
    plt.figure(figsize=(10, 10))
    for idx, x_i in enumerate(x.T):
        plt.subplot(2, 2, idx+1)
        plt.scatter(x_i, y)
        plt.xlabel(f'x_{idx+1}')
        plt.ylabel(f'y', rotation=0)

    plt.subplot(2, 2, idx+2)
    plt.scatter(x[:, 0], x[:, 1])
    plt.xlabel(f'x_{idx}')
    plt.ylabel(f'x_{idx+1}', rotation=0)
```

93

scatter_plot(X_centered, y)



94

Define Model Suitable for Multiple Linear Regression

- Now, let's use PyMC3 to define a model that's suitable for multiple linear regression. As expected, the code looks pretty similar to what we used for simple linear regression. The main differences are:
  - The variable beta is a Gaussian with shape=2, one slope per each independent variable
  - We define the variable μ using the dot product function pm.math.dot()
- If you are familiar with NumPy, you probably know that NumPy includes a dot function, and from Python 3.5 (and from NumPy 1.10), a new matrix operator, @, is included. Nevertheless, here, we use the dot function from PyMC3, which is just an alias for a Theano matrix multiplication operator. We are doing so because the variable β is a Theano tensor and not a NumPy array:

95

Code for Multiple Linear Regression Model

```
with pm.Model() as model_mlr:
    α_tmp = pm.Normal('α_tmp', mu=0, sd=10)
    β = pm.Normal('β', mu=0, sd=1, shape=2)
    ε = pm.HalfCauchy('ε', 5)

    μ = α_tmp + pm.math.dot(X_centered, β)
    α = pm.Deterministic('α', α_tmp - pm.math.dot(X_mean, β))
    y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)
trace_mlr = pm.sample(2000)
```

96

Summarize Inferred Parameters

```
varnames = ['α', 'β', 'ε']
az.summary(trace_mlr, var_names=varnames)
```

	mean	sd	mc error	hpd 3%	hpd 97%	eff_n	r_hat
$\alpha[0]$	1.86	0.46	0.0	0.95	2.69	5251.0	1.0
$\beta[0]$	0.97	0.04	0.0	0.89	1.05	5467.0	1.0
$\beta[1]$	1.47	0.03	0.0	1.40	1.53	5464.0	1.0
ϵ	0.47	0.03	0.0	0.41	0.54	4159.0	1.0

- As we can see, our model is capable of recovering the correct values (check the values used to generate the synthetic data).

97

Missing Factor

- Imagine the following situation. We have a variable *z* correlated with the predictor variable *x* and, at the same time, with the predicted variable *y*. Suppose that the variable *z* is the one responsible for causing *x* and *y*. For example, *z* could be the industrial revolution (a really complex variable!), *x* the number of pirates, and *y* the concentration of CO_2. This example should be very familiar to the Pastafarian reader. If we omit *z* from our analysis, we might end up with a nice linear relation between *x* and *y*, and we may even be able to predict *y* from *x*. However, if our interest lies in understating global warming, we could totally miss what is really going on with the actual mechanism relating these variables.

98

Confounding Variables

- We already discussed that correlation does not imply causation. One reason this is not necessarily true is that we may be omitting the variable *z* from our analysis. When this happens, *z* is named as a confounding variable or confounding factor. In many real scenarios, *z* is easy to miss. Maybe we did not measure it or it was not present in the dataset that was sent to us, or we we did not even think it could possibly be related to our problem. Not taking into account confounding variables in an analysis could lead us to establish spurious correlations. This is always a problem when we try to explain something and could also be problematic when we try to predict something without caring about understanding the underlying mechanism. Understanding the mechanism helps us translate what we have learned to new situations; blind predictions do not always have good transferability. For example, the number of sneakers produced in one country could be used as an easy-to-measure indicator of the strength of its economy, but this could be a terrible predictor for other countries with a different production matrix or cultural background.

99

Synthetic Data

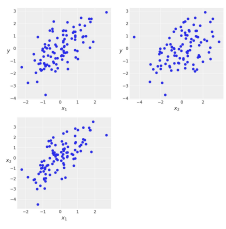
- We are going to use synthetic data to explore the concept of a confounding variable. The following code simulates a confounding variable as x_1. Notice how this variable has influences on x_2 and *y*:

```
np.random.seed(42)
N = 100
x_1 = np.random.normal(size=N)
x_2 = x_1 + np.random.normal(size=N, scale=1)
#x_2 = x_1 + np.random.normal(size=N, scale=0.01)
y = x_1 + np.random.normal(size=N)
X = np.vstack((x_1, x_2)).T
```

- Notice that by virtue of the way we create these variables, they are already centered.

100

scatter_plot(X, y)



101

First Model m_x1x2 (Linear Regression w/ Two Independent Variables x_1 and x_2)

```
with pm.Model() as m_x1x2:
    α = pm.Normal('α', mu=0, sd=10)
    β1 = pm.Normal('β1', mu=0, sd=10)
    β2 = pm.Normal('β2', mu=0, sd=10)
    ε = pm.HalfCauchy('ε', 5)

    μ = α + β1 * X[:, 0] + β2 * X[:, 1]
    y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)

trace_x1x2 = pm.sample(2000)
```

102

Second Model m_x1 (Simple Linear Regression for x_1)

```
with pm.Model() as m_x1:
        α = pm.Normal('α', mu=0, sd=10)
        β1 = pm.Normal('β1', mu=0, sd=10)
        ε = pm.HalfCauchy('ε', 5)
        μ = α + β1 * X[:, 0]
        y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)
trace_x1 = pm.sample(2000)
```

103

Third Model m_x2 (Simple Linear Regression for x_2)

```
with pm.Model() as m_x2:
        α = pm.Normal('α', mu=0, sd=10)
        β2 = pm.Normal('β2', mu=0, sd=10)
        ε = pm.HalfCauchy('ε', 5)
        μ = α + β2 * X[:, 1]
        y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)
trace_x2 = pm.sample(2000)
```

104

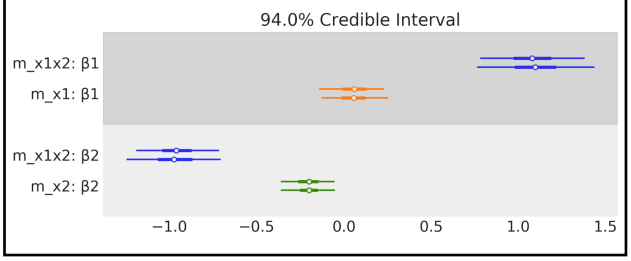
plot_forest()

- Take a look at the β parameters for these models. Using plot_forest(), we can compare them in a single plot:

```
az.plot_forest([trace_x1x2, trace_x1, trace_x2], model_names=['m_x1x2', 'm_x1', 'm_x2'],
        var_names=['β1', 'β2'], combined=False, colors='cycle', figsize=(8, 3))
```

105

The Plot



106

Plot Explanation (Redundant Variable)

- As we can see, β_2 for model m_x1x2 is around zero, indicating an almost null contribution of the x_2 variable to explain y. This is interesting because we already know (check the synthetic data) that the really important variable is x_1. Also notice—and this is really important—that β_2 for model m_x2 is around 0.55. This is larger than for model m_x1x2. The power of x_2 to predict y is reduced when we take into account x_1; the information in x_2 is redundant given x_1.

107

Multicollinearity or When Correlation is Too High

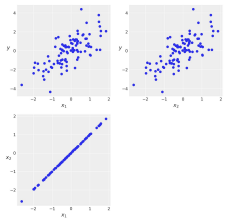
- In the previous example, we saw how a multiple linear regression model reacts to redundant variables, and we saw the importance of considering possible confounding variables. Now, we will take the previous example to an extreme and see what happens when two variables are highly correlated. To study this problem and its consequences for inference, we will use the same synthetic data and model as before, but now we will increase the degree of correlation between x_1 and x_2 by reducing the amount of Gaussian noise we add to x_1 to obtain x_2:

```
np.random.seed(42)
N = 100
x_1 = np.random.normal(size=N)
x_2 = x_1 + np.random.normal(size=N, scale=0.01)
y = x_1 + np.random.normal(size=N)
X = np.vstack((x_1, x_2)).T
```

- This change in the data-generating code is practically equivalent to summing zero to x_1, and hence both variables are, for all practical purposes, equal. You can then try varying the values of the scale and using less extreme values, but for now we want to make things crystal clear.

108

scatter_plot(X, y)



109

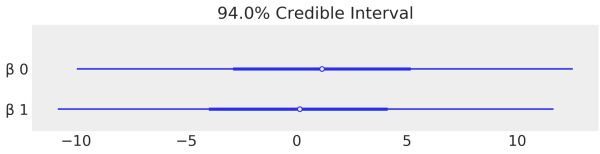
Tracing Multiple Linear Regression Model (Again)

- You should see something like the figure in the previous slide, the scatter plot for x_1 and x_2 is virtually a straight line with a slope around 1.
- We then run a multiple linear regression:

```
with pm.Model() as model_red:
    α = pm.Normal('α', mu=0, sd=10)
    β = pm.Normal('β', mu=0, sd=10, shape=2)
    ε = pm.HalfCauchy('ε', 5)
    μ = α + pm.math.dot(X, β)
    y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=y)
trace_red = pm.sample(2000)
```

110

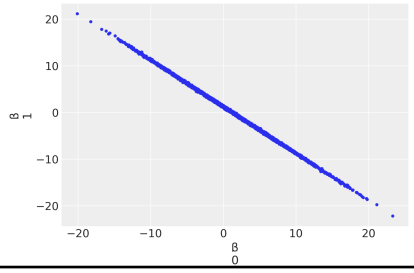
az.plot_forest(trace_red, var_names=['β'], combined=True, figsize=(8, 2))



- The HPD for β coefficients is suspiciously wide.

111

az.plot_pair(trace_red, var_names=['β'])



112

Plot Explanation (1/3)

- The marginal posterior for β is a really narrow diagonal. When one β coefficient goes up, the other must go down. Both are effectively correlated. This is just a consequence of the model and the data. According to our model, the mean μ is:

$$\mu = \alpha + \beta_1 x_1 + \beta_2 x_2$$

- If we assume x_1 and x_2 are not just practically equivalent, but mathematically identical, we can rewrite the model as:

$$\mu = \alpha + (\beta_1 + \beta_2) x$$

113

Plot Explanation (2/3)

- It turns out that it is the sum of β_1 and β_2, and not their separated values, that affects μ. We can make β_1 smaller and smaller as long as we get β_2. We practically do not have two variables and thus practically we do not have two x parameters. We say that the model is **indeterminate** (or equivalently the data is unable to restrict the parameters in the model). In our example, there are two reasons why β cannot freely move over the $[-\infty, \infty]$ interval. First, both variables are almost the same, but they are not exactly equal, and second, and the most important point, is that we have a prior restricting the plausible values that β can take.

114

Plot Explanation (3/3)

- There are a couple of things to notice from this example. First of all, the posterior is just the logical consequence of our data and model, and hence there is nothing wrong with obtaining such wide distributions for β; C'est la vie. Second, we can rely on this model to make predictions. Try, for example, making posterior predictive checks; the values predicted by the model are in agreement with the data; the model is capturing the data very well. Third, this may not be a very good model to understand our problem. It may be more clever just to remove one of the variables from the model. We will end up having a model that predicted the data as well as before, but with an easier (and simpler) interpretation.

115

How Large Correlation of Two Variables Will Become a Problem?

- In any real dataset, correlations are going to exist to some degree. How strong should two or more variables be correlated to become a problem? Unfortunately, statistics is a discipline with very few magic numbers. It is always possible to do a correlation matrix before running any Bayesian model and check for variables with a high correlation of, let's say, above 0.9 or so. Nevertheless, the problem with this approach is that what really matters is not the pairwise correlations we can observe in a correlation matrix, but the correlation of the variables inside a model, and as we already saw, variables behave differently in isolation than when they are put together in a model. Two or more variables can increase or decrease their correlation when put in the context of other variables in a multiple regression model. As always, careful inspection of the posterior together with an iterative critical approach to model building are highly recommended and can help us to spot problems and understand the data and models.

116

A Quick Guide

- If the correlation is really high, we can eliminate one of the variables from the analysis; given that both variables have similar information, which one we eliminate is often irrelevant. We can eliminate variables based on pure convenience, such as removing the least known variable in our discipline or one that is harder to interpret or measure.
- Another possibility is to create a new variable averaging the redundant variables. A more sophisticated version is to use a variable reduction algorithm such as **Principal Component Analysis** (**PCA**). The problem with PCA is that the resulting variables are linear combinations of the original ones obfuscating, in general, the interpretability of the results.
- Yet another solution is to put stronger priors to restrict the plausible values the coefficient can take. In *"Model Comparison"* we briefly discuss some choices for such priors, known as **regularizing priors**.

117

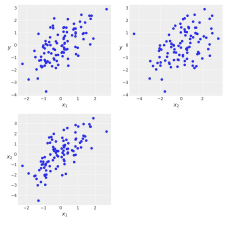
Masking Effect Variables

- One tricky example of how variables contribute to an outcome is the case of masking effect variables. Let's create a toy dataset to exemplify this phenomena. Basically, we are creating two independent variables (x_1 and x_2). They are positively correlated to each other and they are correlated to y, but in opposite directions; x_1 is positively correlated and x_2 negatively correlated:

```
np.random.seed(42)
N = 126
r = 0.8
x_1 = np.random.normal(size=N)
x_2 = np.random.normal(x_1, scale=(1 - r ** 2) ** 0.5)
y = np.random.normal(x_1 - x_2)
X = np.vstack((x_1, x_2)).T
scatter_plot(X, y)
```

118

Scatter Plots of Dataset



119

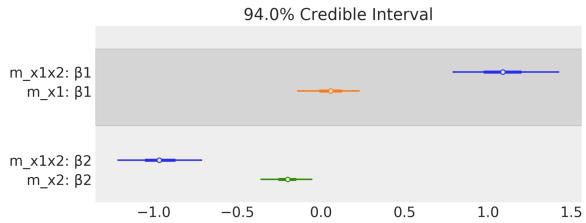
Three Linear Regression Models

- As we did before, we are going to build three related models. The first one, m_x1x2, is a linear regression model with two independent variables, x_1 and x_2 (stacked together in the variable *X*). The second model, m_x1, is a simple linear regression for x_1 and the third one, m_x2, is a simple linear regression for x_2. After sampling from these models, take a look at the β parameters using a forest plot to compare them in a single plot:

```
az.plot_forest([trace_x1x2, trace_x1, trace_x2], model_names=['m_x1x2', 'm_x1', 'm_x2'],
               var_names=['β1', 'β2'], combined=True, colors='cycle', figsize=(8, 3))
```

120

The Plot



121

Plot Explanation

- According to the posterior, the values of β for m_x1x2 are close to 1 and -1 (as expected, according to the way we generate the data). For the simple linear regression model, that is when we study each variable on its own, we can see that the values for β are instead closer to zero, indicating a weaker effect.
- Notice x_1 is correlated to x_2. In fact, when x_1 increases, x_2 also increases. Also notice that when y increases, x_1 also increases, but x_2 decreases. As a result of this particular arrangement, we get a partial cancellation of effects unless we include both variables in the same linear regression. The linear regression model is able to untable these effects because the model is learning for each data point what the contribution of x_1 to y is given a value of x_2, and the other way around for x_2.

122

Adding Interactions

- So far in the definition of the multiple regression model, it is declared (implicitly) that a change in x_i results in a constant change in y, while keeping fixed the values for the rest of the predictor variables. But of course, this is not necessarily true. It could happen that changes in x_i affects y, which is modulated by changes in x_j. A classic example of this behavior is the interaction between drugs. For example, increasing the dose of drug A results in a positive effect on a patient. This is true in the absence of drug B (or for a low dose of B) while the effect of A is negative (even lethal) for increasing doses of B.

123

Interactions

- In all of the examples we have seen so far, the dependent variables contribute additively to the predicted variable. We just add variables (each one multiplied by a coefficient). If we wish to capture effects, like in the drug example, we need to include terms in our model that are not additive. One common option is to multiply variables, for example:

$$\mu = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$$

- Notice that the β_3 coefficient is multiplying a third variable that is the product of x_1 and x_2. This non-additive term is an example of what is know in statistics as **interaction**. There are other ways to introduce **interactions**, but we are going to restrict the discussion to the multiplicative case, as it is the most common expression for interactions.

124

Rewrite Expression

- Interpreting linear models with interactions is not as easy as interpreting linear models without them. Let's rewrite the expression below:

$$\mu = \alpha + \underbrace{(\beta_1 + \beta_3 x_2)}_{\text{slope of } x_1} x_1 + \beta_2 x_2$$

$$\mu = \alpha + \beta_1 x_1 + \underbrace{(\beta_2 + \beta_3 x_1)}_{\text{slope of } x_2} x_2$$

125

Interpreting Linear Models with Interactions

- The interaction term can be understood as a linear model. Thus, the expression for the mean, μ, is a linear model with a linear model inside of it!
- The interaction is symmetric; we can think of it as the slope of x_1 as a function of x_2 and at the same time as the slope of x_2 as a function of x_1.
- In a multiple regression model without interactions, we get a hyperplane, that is, a flat hypersurface. An interaction term introduces a curvature in such a hypersurface. This is because slopes are not constant anymore but functions of another variable.
- The coefficient β_1 describes the influence of predictor x_1 only at $x_2 = 0$. This is true because for that value $\beta_3 x_2 = 0$, and then the slope of x_1 reduces to $\beta_1 x_1$. By symmetry, the same reasoning can be applied to β_2.

126

Variable Variance

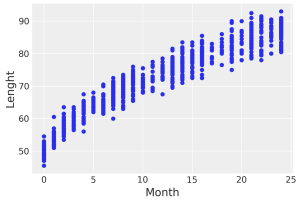
- We have been using the *linear motif* to model the mean of a distribution and, in the previous section, we used it to model interactions. We can also use it to model the variance (or standard deviation) when the assumptions of constant variance do not make sense. For those cases, we may want to consider the variance as a (linear) function of the dependent variable.
- The **World Health Organization** (**WHO**) and other health institutions around the world collect data for newborns and toddlers and design growth charts standards. These charts are an essential component of the paediatric toolkit and also a measure of the general well-being of populations in order to formulate health-related policies, plan interventions, and monitor their effectiveness (http://www.who.int/childgrowth/en/).

127

An Example

- An example of such data is the lengths (heights) of newborn/toddlers girls as a function of the age (in months):

```
data = pd.read_csv('../data/babies.csv')
data.plot.scatter('Month', 'Lenght')
```



128

Shared Variable

- To model this data, we are going to introduce three new elements compared to the previous models:
  - ε is now a linear function of x. To do this, we add two new parameters, γ and δ. These are direct analogs of α and β.
  - The linear model for the mean is a function of $\sqrt{x}$. This is just a simple trick to fit a linear model to a curve.
  - We define a **shared variable**, x_shared. We will use it to change the values of the x variable (Month, in this example), after model fitting without the need to refit the model.

129

Code

```
with pm.Model() as model_vv:
    α = pm.Normal('α', sd=10)
    β = pm.Normal('β', sd=10)
    γ = pm.HalfNormal('γ', sd=10)
    δ = pm.HalfNormal('δ', sd=10)

    x_shared = shared(data.Month.values * 1.)

    μ = pm.Deterministic('μ', α + β * x_shared**0.5)
    ε = pm.Deterministic('ε', γ + δ * x_shared)

    y_pred = pm.Normal('y_pred', mu=μ, sd=ε, observed=data.Lenght)
trace_vv = pm.sample(1000, tune=1000)
```

130

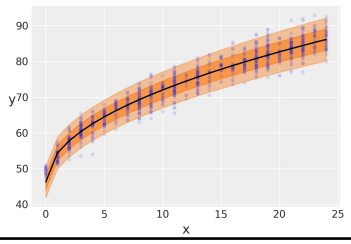
Code to Plot Figure

- The figure in the next slide shows the result of our model. The mean of μ is represented with a black curve, and two semitransparent orange bands represent 1 and 2 standard deviations, respectively:

```
plt.plot(data.Month, data.Lenght, 'C0.', alpha=0.1)

μ_m = trace_vv['μ'].mean(0)
ε_m = trace_vv['ε'].mean(0)
plt.plot(data.Month, μ_m, c='k')
plt.fill_between(data.Month, μ_m + 1 * ε_m, μ_m - 1 * ε_m, alpha=0.6, color='C1')
plt.fill_between(data.Month, μ_m + 2 * ε_m, μ_m - 2 * ε_m, alpha=0.4, color='C1')
plt.xlabel('x')
plt.ylabel('y', rotation=0)
```

131

The Plot



132

Author's Question

- At the time of writing the textbook, the author's daughter is two weeks old (about 0.5 months), and thus the author wonder how her length compares to the growth chart we have just created. One way to answer this question is to ask the model for the distribution of the variable length for babies of 0.5 months. Using PyMC3, we can ask this question with the sample_posterior_predictive function.

133

0.5 Not an Observed Value of x

- The output of this function will be samples of y conditioned on the observed data and the estimated distribution of parameters—that includes uncertainties. The only problem is that, by definition, this function will return predictions for y for the observed values of x, and 0.5 months (the value we care about) has not been observed; all measures are reported for integer months. The easier way to get predictions for non-observed values of x is to define a shared variable (as part of the model) and then update the value of the shared variable right before sampling from the posterior predictive distribution:

```
x_shared.set_value([0.5])
ppc = pm.sample_posterior_predictive(trace_vv, 2000, model=model_vv)
y_ppc = ppc['y_pred'][:, 0]
```

134

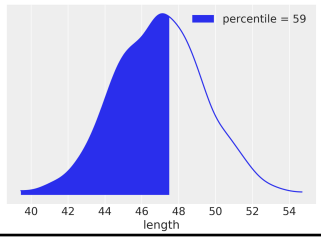
Plotting Expected Distribution of Lengths for Two Weeks Babies

- Now, we can plot the expected distribution of lengths for two weeks old babies and compute additional quantities, for example, the percentile of a child, given her length. Check the following block of code and the figure in the next slide for such an example:

```
ref = 47.5
density, l, u = az._fast_kde(y_ppc)
x_ = np.linspace(l, u, 200)
plt.plot(x_, density)
percentile = int(sum(y_ppc <= ref) / len(y_ppc) * 100)
plt.fill_between(x_[x_ < ref], density[x_ < ref], label='percentile = {:2d}'.format(percentile))
plt.xlabel('length')
plt.yticks([])
plt.legend()
```

135

The Plot



136