# Programming Probabilistically

Min-Te Sun, Ph.D.

1

## Probabilistic Programming with PyMC3

- The basic idea is to specify models using code and then solve them in a more or less automatic way. Reasons to do probabilistic programming are:
  1. Many models do not lead to an analytic closed form, and thus we can only solve those models using numerical techniques.
  2. Modern Bayesian statistics is mainly done by writing code, and Python is the most popular programming language.
- PyMC3 – a very flexible Python library for probabilistic programming.
- ArviZ – a new Python library that will help us interpret the results of probabilistic models.

2

## Bayesian Statistics

- Bayesian statistics is conceptually very simple: *the knowns* and *the unknowns*
  - We use Bayes' theorem to condition the latter on the former.
- Generally, we refer to the *knowns* as **data** and treat it like a constant, and the *unknowns* as **parameters** and treat them as probability distributions.
  - In more formal terms, we assign probability distributions to unknown quantities. Then, we use Bayes' theorem to transform the prior probability distribution into a posterior distribution .

3

## Probabilistic Programming Language Framework

- Users specify a full probabilistic model by writing a few lines of code, and then inference follows automatically.
- It enables practitioners to build complex probabilistic models in a less time-consuming and less error- prone way.
  - Its impact on data science is like Fortran programming on scientific computing
- PPL hides details on how probabilities are manipulated and how the inference is performed from users, allowing users to focus on model specification and the analysis of results.

4

## PyMC3 Primer

- A Python library for probabilistic programming
- PyMC3's base code is written using Python, and the computationally demanding parts are written using NumPy and Theano.
- **Theano** is a Python library that was originally developed for deep learning and allows us to define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently.
- The reasons PyMC3 uses Theano are
  - Some of the sampling methods, such as No-U-Turn Sampler (NUTS), need gradients to be computed, and Theano knows how to compute gradients using what is known as **automatic differentiation**.
  - Theano compiles Python code to C code, and hence PyMC3 is really fast.

5

## Flipping Coins PyMC3 Way

- Generating data

```
np.random.seed(123)
trials = 4
theta_real = 0.35 # unknown value in a real experiment
data = stats.bernoulli.rvs(p=theta_real, size=trials)
```

- Model specification
  - For the likelihood, we will use the binomial distribution with $n = 1$ and $p = \theta$.
  - For the prior, a beta distribution with the parameters (1, 1), which is equivalent to a uniform distribution in the interval [0, 1].

$$\theta \sim Beta(1,1)$$
$$y \sim Bern(n = 1, p = \theta)$$

6

## Model Translation to PyMC3

```
with pm.Model() as our_first_model:
        θ = pm.Beta('θ', alpha=1., beta=1.)
        y = pm.Bernoulli('y', p=θ, observed=data)
        trace = pm.sample(1000, random_seed=123)
```

- Note that we use the name θ twice, first as a Python variable and then as the first argument of the Beta function; using the same name is a good practice to avoid confusion.
  - The θ variable is a random variable; it is not a number, but an object representing a probability distribution from which we can compute random numbers and probability densities.
- We pass the data using the observed argument.
  - This is the way in which we tell PyMC3 that we want to condition for the unknown on the knows (data).
  - The observed values can be passed as a Python list, a tuple, a NumPy array, or a pandas DataFrame.

7

## Pushing Inference Button

- The last line is the *inference button*. We are asking for 1,000 samples from the posterior and will store them in the trace object.
  - If you run the code, you will get a message like this:

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [θ]
100%|[redacted]| 3000/3000 [00:00<00:00, 3695.42it/s]
```

- The first and second lines tell us that PyMC3 has automatically assigned the NUTS sampler, and has used a method to initialize that sampler. The third line says that PyMC3 will run two chains in parallel, thus we will get two independent samples from the posterior.
  - The number of chains depends on the number of processors in the machine. you can change it using the chains argument for the sample function.

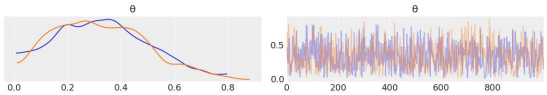
8

## Pushing Inference Button (Cont.)

- The next line is telling us which variables are being sampled by which sampler.
  - For this particular case, NUTS is used to sample θ. PyMC3 assign different samplers to different variables automatically based on properties of the variables that ensures that the best possible sampler is used for each variable.
  - Users can manually assign samplers using the step argument of the sample function.
- The last line is a progress bar, with several related metrics indicating how fast the sampler is working, including the number of iterations per second.
  - PyMC3 generates 3,000 samples because we have 500 additional samples per chain to auto-tune the sampling algorithm. This sample will be discarded by default. We also have 1,000 productive draws per-chain, thus a total of 3,000 samples are generated. We can change the number of tuning steps with the tune argument of the sample function.

9

## plot_trace from ArviZ to Summarize Posterior




- By using az.plot_trace, we get two subplots for each unobserved variable. The only unobserved variable in our model is θ.
  - Notice that **y** is an observed variable representing the data; we do not need to sample that because we already know those values. Thus, in the above figure, we have two subplots.
- On the left, we have a **Kernel Density Estimation** (**KDE**) plot; this is like the smooth version of the histogram. On the right, we get the individual sampled values at each step during the sampling. From the trace plot, we can visually get the plausible values from the posterior.

10

## Numerical Summary of Trace

- We can get that using az.summary, which will return a pandas DataFrame:

| | Mean | Sd | mc error | hpd 3% | hpd 97% | Eff_n | r_hat |
|---|---|---|---|---|---|---|---|
| *θ* | 0.33 | 0.18 | 0.0 | 0.02 | 0.64 | 847.0 | 1.0 |

- We get the mean, standard deviation (sd), and 94% HPD interval (hpd 3% and hpd 97%). As we discussed in *Thinking Probabilistically*, we can use these numbers to interpret and report the results of a Bayesian inference.
  - The last two metrics are related to diagnosing samples. For details, see Chapter 8, *Inference Engines*.

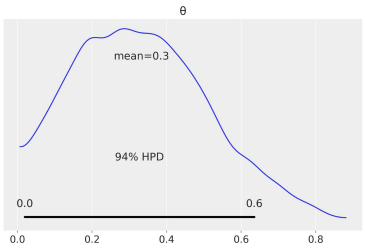
11

## plot_posterior Function in ArviZ

- By default, plot_posterior shows a histogram for discrete variables and KDEs for continuous variables. We also get the mean of the distribution and the 94% HPD as a black line at the bottom of the plot.
  - We can ask for the median or mode using the point_estimate argument.
  - Different interval values can be set for the HPD with the credible_interval argument.
  - This type of plot was introduced by John K. Kruschke in his great book *Doing Bayesian Data Analysis.*

12

## az.plot_posterior(trace)




13

## Posterior-based Decisions

- Sometimes, we need to make decisions based on our inferences. We have to reduce a continuous estimation to a dichotomous one: yes-no, health-sick, contaminated-safe, and so on.
  - We may need to decide if the coin is fair or not. A fair coin is one with a $\theta$ value of exactly 0.5.
  - We can compare the value of 0.5 against the HPD interval. In the figure on slide #13, we can see that the HPD goes from ≈0.02 to ≈0.71 and hence 0.5 is included in the HPD. According to our posterior, the coin seems to be tail-biased, but we cannot completely rule out the possibility that the coin is fair.
- If we want a sharper decision, we will need to collect more data to reduce the spread of the posterior or maybe we need to find out how to define a more informative prior.

14

## Region Of Practical Equivalence (ROPE)

- We generally do not care about exact results, but results within a certain interval. We call this interval a Region Of Practical Equivalence (ROPE).
  - Accordingly, in practice, we can relax the definition of fairness and we can say that a fair coin is one with a value of $\theta$ around 0.5. For example, we could say that any value in the interval [0.45, 0.55] will be, for our purposes, practically equivalent to 0.5.
- Once the ROPE is defined, we compare it against the Highest-Posterior Density (HPD). We can get at least three scenarios:

1. The ROPE does not overlap with the HPD; we can say the coin is not fair.
2. The ROPE contains the entire HPD; we can say the coin is fair.
3. The ROPE partially overlaps with HPD; we cannot say the coin is fair or unfair.

15

## Choice of ROPE

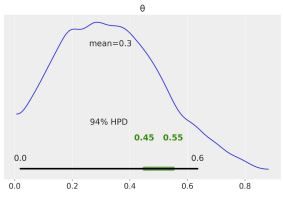
- A ROPE is an arbitrary interval we choose based on background knowledge. Any value inside this interval is assumed to be of practical equivalence.
- If we choose a ROPE in the interval [0, 1], we will always say we have a fair coin. Notice that we do not need to collect data to perform any type of inference. Of course, this is a trivial, unreasonable, and dishonest choice and probably nobody is going to agree with our ROPE definition. We mention it just to highlight the fact that the definition of the ROPE is context-dependent; there is no auto-magic rule that will fit everyone's intentions. Decisions are inherently subjective and our mission is to take the most informed possible decisions according to our goals.

16

## Plot Posterior with HPD

- We can use the plot_posterior function to plot the posterior with the HPD interval and the ROPE. The ROPE appears as a semi-transparent thick (green) line:

```
az.plot_posterior(trace, rope=[0.45, .55])
```

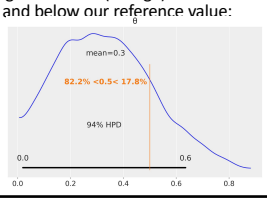



17

## Compare Posterior against Reference Value

- Another tool we can use to help us make a decision is to compare the posterior against a reference value. We can do this using plot_posterior. As you can see, we get a vertical (orange) line and the proportion of the posterior above and below our reference value:

```
az.plot_posterior(trace, ref_val=0.5)
```




18

## Loss Functions

- To make a good decision, it is important to have the highest possible level of precision for the estimated value of the relevant parameters, but it is also important to take into account the cost of making a mistake. The cost/benefit trade-off can be mathematically formalized using loss functions (also known as cost functions, objective functions, fitness functions, or utility functions).
- The key idea is to use a function that captures how different the true value and the estimated value of a parameter are. The larger the value of the loss function, the worse the estimation is (according to the loss function). Some common examples of loss functions are:
  - The quadratic loss $(\theta - \hat{\theta})^2$
  - The absolute loss $|\theta - \hat{\theta}|$
  - The 0-1 loss $I(\theta \neq \hat{\theta})$, where $I$ is the indicator function

19

## Expected Loss Function

- In practice, we generally do not have the value of the true parameter $\theta$ at hand. Instead, we have an estimation in the form of a posterior distribution. Thus, what we can do is find out the value of $\hat{\theta}$ that minimizes the expected loss function.
- By expected loss function, we mean the loss function averaged over the whole posterior distribution.

20

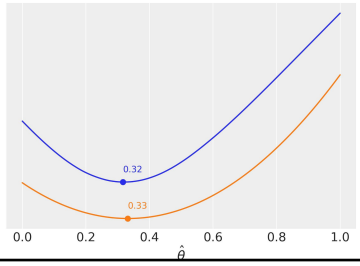
## Plotting Loss Functions

- In the following block of code, we have two loss functions: the absolute loss (lossf_a) and the quadratic loss (lossf_b). We will explore the value of $\hat{\theta}$ over a grid of 200 points. We will then plot those curves and we will also include the value of $\hat{\theta}$ that minimizes each loss function:

```
grid = np.linspace(0, 1, 200)
θ_pos = trace['θ']
lossf_a = [np.mean(abs(i - θ_pos)) for i in grid]
lossf_b = [np.mean((i - θ_pos)**2) for i in grid]
for lossf, c in zip([lossf_a, lossf_b], ['C0', 'C1']):
        mini = np.argmin(lossf)
        plt.plot(grid, lossf, c)
        plt.plot(grid[mini], lossf[mini], 'o', color=c)
        plt.annotate('{:.2f}'.format(grid[mini]), (grid[mini], lossf[mini] + 0.03), color=c)
        plt.yticks([])
        plt.xlabel(r'$\hat \theta$')
```

21

## Loss Functions




22

## Take Home Message of Loss Functions

- As we can see, the result looks somewhat similar for lossf_a is $\hat{\theta} = 0.32$ and lossf_b is $\hat{\theta} = 0.33$. What is interesting from this result is that the first value is equal to the median of the posterior and the last value is equal to the mean of the posterior.
  - You can check this for yourself by computing np.mean(θ_pos), np.median(θ_pos).
- This is no formal proof, but the take home message is this: different loss functions are related to different point-estimates.

23

## Should We Explicitly Choose a Loss Function?

- If we want to be formal and we want to compute a single point-estimate, we must decide which loss function we want, or in reverse, if we choose a given point-estimate, we are implicitly (and maybe even unconsciously) deciding on a loss function.
- The advantage of explicitly choosing a loss function is that we can tailor the function to our problem, instead of using some predefined rule that may not be suitable in our particular case.
  - For example, in many problems, the cost of making a decision is asymmetric; it is not the same to decide whether it is safe or not to administrate a vaccine to children under five and being right, that being wrong. Making a bad decision could cost thousands of lives and produce a health crisis that could have being avoided by administrating a cheap and safe vaccine.
  - Thus, if our problem demands it, we can construct an asymmetric loss function. It is also important to notice that, as the posterior is in the form of numerical samples, we can compute complex loss functions that don't need to be restricted by mathematical convenience or mere simplicity.

24

## In Practice …

- It is not true that every time people uses a point-estimate they are truly thinking in terms of loss functions. In fact, loss functions are not very common in many of the scientific fields.
- People often choose the median, just because it is more robust to outliers than the mean, or use the mean just because is a simple and familiar concept, or because they think their observable is truly the average of some process at some level.

25

## Gaussian Model

- We introduced the main Bayesian notions using the beta-binomial model mainly because of its simplicity. Another very simple model is the Gaussian or normal model.
- Gaussians are very appealing from a mathematical point of view because working with them is easy.
  1. The conjugate prior of the Gaussian mean is the Gaussian itself.
  2. There are many phenomena that can be nicely approximated using Gaussians; essentially, almost every time that we measure the average of something, using a *big enough* sample size, that average will be distributed as a Gaussian. The details of when this is true, when this is not true, and when this is more or less true, are elaborated in the **central limit theorem** (**CLT**). (You are advised to check out wiki page of CLT!)

26

## Why Are Gaussian Models Important?

- Many phenomena are indeed averages.
  - For example, the height (and almost any other trait of a person, for that matter) is the result of many environmental factors and many genetic factors, and hence we get a nice Gaussian distribution for the height of adult people.
- In summary, Gaussians are easy to work with and they are more or less abundant in nature, and hence many of the statistical methods you may already know, or have at least heard of, are based on normality assumptions. Thus, it is important to learn how to build these models, and then it is also equally important to learn how to relax the normality assumptions, something surprisingly easy in a Bayesian framework and with modern computational tools such as PyMC3.

27

## An Example – Nuclear Magnetic Resonance

- **Nuclear magnetic resonance** (**NMR**) is a powerful technique that's used to study molecules and also living things such as humans or yeast. NMR allows you to measure different kinds observable quantities that are related to unobservable and interesting molecular properties. One of these observables is known as **chemical shift**; we can only get chemical shifts for the nuclei of certain types of atoms.
  - All of this is in the domain of quantum chemistry and the details are irrelevant for this class.
- Chemical shifts – *shift*s because we measured a signal shift from a reference value, and *chemical* because the shift is related somehow to the chemical environment on the nuclei we are measuring.

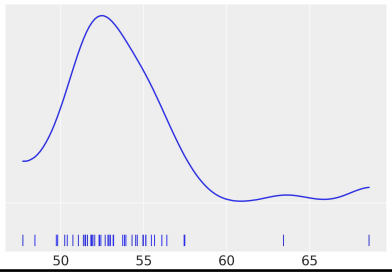
28

- The choice of this example is based on author's expertise. For all we care at the moment, we could have been measuring the height of a group of people, the average time to travel back home, the weight of bags of oranges, or even a discrete variable like the number of sexual partners of the Tokay gecko if we are willing to approximate this number as a continuous variable.
- Keep in mind that we do not need our data to be truly Gaussian (or any other distribution, for that matter): we only demand that a Gaussian is a *reasonable* approximation to our data. For this example, we have *48* chemical shifts values that we can load into a NumPy array and plot by using the following code:

```
data = np.loadtxt('../data/chemical_shifts.csv')
az.plot_kde(data, rug=True)
plt.yticks([0], alpha=0)
```

29

## KDE Plot of Dataset




30

## A Reasonable Model

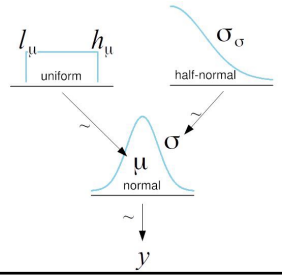
- Let's forget about those two outliers for a moment and assume that a Gaussian distribution is a proper description of the data. Since we do not know the mean or the standard deviation, we must set priors for both of them. Therefore, a reasonable model could be:

$$\mu \sim U(l, h)$$
$$\sigma \sim |N(0, \sigma_\sigma)|$$
$$y \sim N(\mu, \sigma)$$

- Thus, $\mu$ comes from a uniform distribution with boundaries $l$ and $h$, which are the lower and upper bounds, respectively, and $\sigma$ comes from a half-normal distribution with a standard deviation of $\sigma_\sigma$. A half-Normal distribution is like the regular Normal distribution but restricted to positive values (including zero). You can get samples from a half-normal by sampling from a normal distribution and then taking the absolute value of each sampled value. Finally, in our model, the data $y$ comes from a Normal distribution with the parameters $\mu$ and $\sigma$.

31

## Kruschke-style Diagram




32

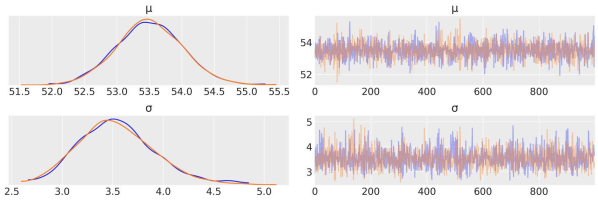
## Prior and PyMC3 Code for Gaussian Model

- If we do not know the possible values of $\mu$ and $\sigma$, we can set priors reflecting our ignorance. One option is to set the boundaries of the uniform distribution to be $l = 40$, $h = 75$, which is a range larger than the range of the data. Alternatively, we can choose a range based on our previous knowledge. We may know that this is not physically possible to have values below 0 or above 100 for this type of measurement. In such a case, we can set the prior for the mean as a uniform with parameters $l = 0$, $h = 100$. For the half normal, we can set $\sigma_\sigma = 10$, just a large value for the data. The PyMC3 code of the model is as follows:

```
with pm.Model() as model_g:
        μ = pm.Uniform('μ',lower=40, upper=70)
        σ = pm.HalfNormal('σ', sd=10)
        y = pm.Normal('y', mu=μ, sd=σ, observed=data)
        trace_g = pm.sample(1000)
az.plot_trace(trace_g)
```

33

## Trace Plot of Gaussian Model




34

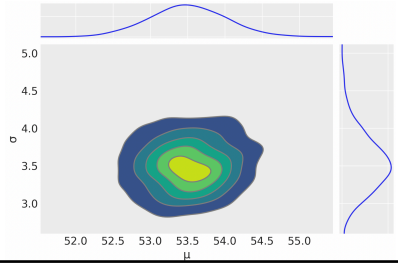
## Marginal Distributions

- As we may have noticed, in the figure in slide #34, it has one row for each parameter. For this model, the posterior is bi-dimensional, and so the figure is showing the marginal distributions of each parameter. We can use the plot_joint function from ArviZ to see what the bi-dimensional posterior looks like, together with the marginal distributions for μ and σ:

```
az.plot_joint(trace_g, kind='kde', fill_last=False)
```

35

## Use plot_joint function from ArviZ to See Bi-dimensional Posterior




36

## Numerical Summary of Trace

- If we want to get access to the values for any of the parameters stored in a trace object, we can index the trace with the name of the parameter in question. As a result, we will get a NumPy array. Try doing trace_g['σ'] or az.plot_kde(trace_g['σ']).
  - Using Jupyter Notebook/lab, you can get characters such as $\sigma$ by writing \sigma in a code cell and then hitting the Tab key.
- We are going to print the summary for later use:

```
az.summary(trace_g)
```

| | Mean | Sd | mc error | hpd 3% | hpd 97% | Eff_n | r_hat |
|---|---|---|---|---|---|---|---|
| $\mu$ | 53.49 | 0.50 | 0.00 | 52.5 | 54.39 | 2081.0 | 1.0 |
| $\sigma$ | 3.54 | 0.38 | 0.01 | 2.8 | 4.22 | 1823.0 | 1.0 |

37

## Posterior Predictive Checks

- Now that we have computed the posterior, we can use it to simulate data and check how consistent the simulated is data with respect to the observed data. If you remember from Chapter 1, *Thinking Probabilistically*, we generically call this type of comparisons **posterior predictive checks**, because we are using the posterior to make predictions, and are using those predictions to check the model.

38

## sample_posterior_predictive

- Using PyMC3 is really easy to get posterior predictive samples if you use the sample_posterior_predictive function. With the following code, we are generating 100 predictions from the posterior, each one of the same size as the data. Notice that we have to pass the trace and the model to sample_posterior_predictive, while the other arguments are optional:

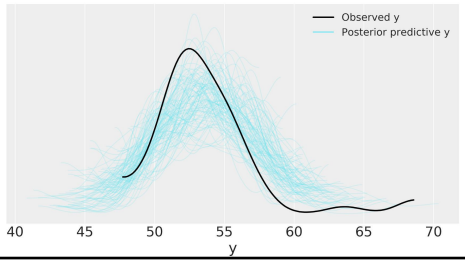
```
y_pred_g = pm.sample_posterior_predictive(trace_g, 100, model_g)
```

- The y_pred_g variable is a dictionary, with the keys being the name of the observed variable in our model and the values an array of shape (samples, size), in this case, (100, len(data)).
  - We have a dictionary because we could have models with more than one observed variable. We can use the plot_ppc function for a visual posterior predictive check:

```
data_ppc = az.from_pymc3(trace=trace_g, posterior_predictive=y_pred_g)
ax = az.plot_ppc(data_ppc, figsize=(12, 6), mean=False)
ax[0].legend(fontsize=15)
```

39

## Posterior Predictive Check of Gaussian Model




40

## Figure Explanation

- In the figure in slide #40, the single (black) line is a KDE of the data and the many semitransparent (cyan) lines are KDEs computed from each one of the 100 posterior predictive samples. The semitransparent (cyan) lines reflect the uncertainty we have about the inferred distribution of the predicted data. Sometimes, when you have very few data points. A plot like this one could show the predicted curves as *hairy* or *wonky*; this is due to the way the KDE is implemented in ArviZ. The density is estimated within the actual range of the data passed to the kde function, while outside this range the density is assumed to be zero.
  - While some could reckon this as a bug, we think it's a feature, since it's reflecting a property of the data instead of over-smoothing it.

41

## Is Our Model Faulty?

- From the figure in slide #40, we can see that the mean of the simulated data is slightly displaced to the right and that the variance seems to be larger for the simulated data than for the actual data. This is a direct consequence of the two observations that are separated from the bulk of the data. Can we use this plot to confidently say that the model is faulty and needs to be changed?
  - As always, the interpretation of the model and its evaluation is context-dependent. Based on the author's experience for these type of measures and the ways we generally use this data, we would say this model is a reasonable enough representation of the data and useful for most of our analysis.

42

## What Can We Do?

- One objection you may have with the model_g model is that we are assuming a normal distribution, but we have two data points on the tails of the distribution, making the *normally* assumption a little bit *forced*. Since the tails of the normal distribution falls quickly as we move away from the mean, the normal distribution is *surprised* by seeing those two points and *reacts* by moving itself toward those points and increasing the standard deviation.
- We can imagine those points as having an *excessive weight* determining the parameters of the normal distribution. So, what can we do?

43

## Solutions

- One option is to declare those points as outliers and remove them from the data. We may have a valid reason to discard those points – maybe a malfunction of the equipment or a human error while measuring those two data points.
  - Sometimes, we can even fix those data points, for example, if we realize they are just a result of bad coding while cleaning the data.
- On many occasions, we may also want to automatize the outlier elimination process by using one of the many *outlier rules*. Two of them are:
  1. Any data point below *1.5* times the interquartile range from the lower quartile or *1.5* times the interquartile range above the upper quartile is an outlier
  2. Any data point below or above two times the standard deviation of the data should be declared an outlier and banished from our data
- Bayesians prefer to encode assumptions directly into the model by using different priors and likelihoods rather than through *ad hoc* heuristics such as outlier removal rules.

44

## Student's t-distribution

- One very useful option when dealing with outliers and Gaussian distributions is to replace the Gaussian likelihood with a Student's t-distribution. This distribution has three parameters: the mean, the scale (analogous to the standard deviation), and the degrees of freedom, which is usually referred to using the Greek letter $\nu$, that can vary in the interval [0, ∞]. We call $\nu$ the normality parameter, since it is in charge of controlling how normal-like the Student's t-distribution is.
  - For a value of $\nu = 1$, we get a distribution with very heavy tails, which is also known as Cauchy or Lorentz distribution. By heavy tails, we mean that it is more probable to find values away from the mean compared to a Gaussian, or in other words values are not as concentrated around the mean as in a lighter tail distribution like the Gaussian. (For example, 95% of the values from a Cauchy distribution are found between -12.7 and 12.7. Instead, for a Gaussian (with a standard deviation of one), this occurs between -1.96 and 1.96.)
  - On the other side of the parameter space, when $\nu$ approaches infinity, we recover the Gaussian distribution.
  - A very curious feature of the Student's t- distribution is that it has no defined mean when $\nu \leq 1$. Of course, in practice, any finite sample from a Student's t-distribution is just a bunch of numbers from which it is always possible to compute an empirical mean. Intuitively, this can be understood as follows: the tails of the distribution are so heavy that at any moment it could happen that we get a sampled value from almost anywhere from the real line, so if we keep getting numbers, we will never approximate to a fixed value. Instead, the estimate will keep wandering around.

45

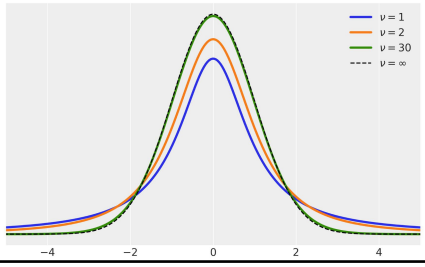
## Plotting Student's t-distributions

- In a similar fashion, the variance of this distribution is only defined for values of $\nu > 2$. Be careful that the scale of the Student's t-distribution is not the same as the standard deviation. For $\nu \leq 2$, the distribution has no defined variance and hence no defined standard deviation. The scale and the standard deviation become closer and closer as $\nu$ approaches infinity:

```
plt.figure(figsize=(10, 6))
x_values = np.linspace(-10, 10, 500)
for df in [1, 2, 30]:
        distri = stats.t(df)
        x_pdf = distri.pdf(x_values)
        plt.plot(x_values, x_pdf, label=fr'$\nu = {df}$', lw=3)
x_pdf = stats.norm.pdf(x_values)
plt.plot(x_values, x_pdf, 'k--', label=r'$\nu = \infty$')
plt.xlabel('x')
plt.yticks([])
plt.legend()
plt.xlim(-5, 5)
```

46

## Student's t-distribution w/ Different $\nu$ values




47

## Revised Model

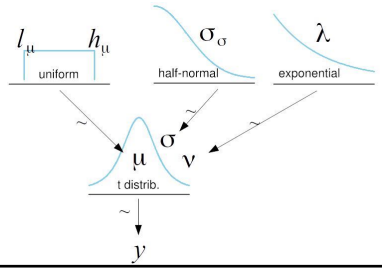
- We are going to rewrite the previous model by replacing the Gaussian distribution with the Student's t-distribution:

$$\mu \sim U(l, h)$$
$$\sigma \sim |N(0, \sigma_\sigma)|$$
$$\nu \sim Exp(\lambda)$$
$$y \sim T(\mu, \sigma, \nu)$$

- Because the Student's t-distribution has one more parameter ($\nu$) than the Gaussian, we need to specify one more prior. We are going to set $\nu$ as an exponential distribution with a mean of 30.
  - From the figure in slide #47, we can see that a Student's t-distribution with $\nu = 30$ looks pretty similar to a Gaussian (even when it is not). In fact, from the same diagram, we can seethat most of the action happens for relatively small values of $\nu$. Hence, we can say that the exponential prior with a mean of 30 is a weakly informative prior telling the model we more or less think $\nu$ should be around 30, but can move to smaller and larger values with ease.

48

## Kruschke-style Diagram



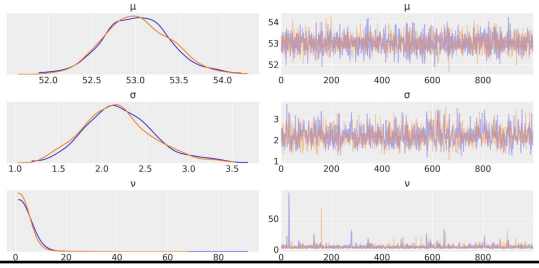

49

## PyMC3 Code for Revised Model

```
with pm.Model() as model_t:
        μ = pm.Uniform('μ', 40, 70)
        σ = pm.HalfNormal('σ', sd=10)
        ν = pm.Exponential('ν', 1/30)
        y = pm.StudentT('y', mu=μ, sd=σ, nu=ν, observed=data)
        trace_t = pm.sample(1000)
az.plot_trace(trace_t)
```

50

## Trace Plot of Revised Model




51

## Numerical Summary of Trace

- Compare the trace from model_g (the figure in slide #34) with the trace of model_t (the figure in slide #51).
- Now, print the summary of model_t and compare it with the one from model_g. Before you keep reading, take a moment to spot the difference between both results.

```
az.summary(trace_t)
```

| | Mean | Sd | mc error | hpd 3% | hpd 97% | Eff_n | r_hat |
|---|---|---|---|---|---|---|---|
| $\mu$ | 53.00 | 0.39 | 0.01 | 52.28 | 53.76 | 1254.0 | 1.0 |
| $\sigma$ | 2.20 | 0.39 | 0.01 | 1.47 | 2.94 | 1008.0 | 1.0 |
| $\nu$ | 4.51 | 3.35 | 0.11 | 1.10 | 9.27 | 898.0 | 1.0 |

52

## Conclusion

- The estimation of $\mu$ between both models is similar, with a difference of $\approx 0.5$.
- The estimation of $\sigma$ changes from ≈3.5 to ≈2.1. This is a consequence of the Student's t-distribution giving less weight by values away from the mean.
- We can also see that $\nu \approx 4.5$, that is, we have a not very Gaussian-like distribution and instead one with heavier tails.

53

## Posterior Predictive Check Again

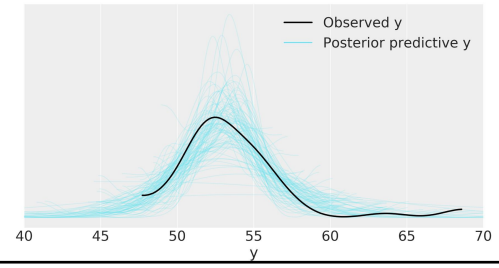
- Now, we are going to do a posterior predictive check of the Student's t model and we are going to compare it to the Gaussian model:

```
y_ppc_t = pm.sample_posterior_predictive( trace_t, 100, model_t, random_seed=123)
y_pred_t = az.from_pymc3(trace=trace_t, posterior_predictive=y_ppc_t)
az.plot_ppc(y_pred_t, figsize=(12, 6), mean=False)
ax[0].legend(fontsize=15)
plt.xlim(40, 70)
```

- Using the Student's t-distribution in our model leads to predictive samples that seem to better fit the data in terms of the location of the peak of the distribution and also its spread. Notice how the samples extend far away from the bulk of the data, and how a few of the predictive samples looks very flat. This is a direct consequence of the Student's t- distribution expecting to see data points far away from the mean or bulk of the data, and this is the reason we are setting xlim to [40, 70].

54

## Posterior Predictive Check of Revised Model




55

## Robust Estimation

- The Student's t-distribution allows us to have a more robust estimation because the outliers have the effect of decreasing $\nu$, instead of pulling the mean toward them and increasing the standard deviation. Thus, the mean and the scale are estimated by weighting more data points from the bulk of the data than those apart from it.
- Once again, it is important to remember that the scale is not the standard deviation. Nevertheless, the scale is related to the spread of the data; the lower its value, the more concentrated the distribution.
- Also, for values of $\nu \geq 2$, the value of the scale tends to be pretty close (at least for most practical purposes) to the values estimated after removing the outliers. So, as a rule of thumb, for values of $\nu$ not too small, and taking into account that it is not theoretically fully correct, we can consider the scale of a Student's t-distribution as a reasonable practical proxy for the standard deviation of the data after removing outliers.

56

## Groups Comparison

- One pretty common statistical analysis is group comparison.
  - We may be interested in how well patients respond to a certain drug, the reduction of car accidents by the introduction of a new traffic regulation, student performance under different teaching approaches, and so on.
- Sometimes, this type of question is framed under the hypothesis testing scenario with the goal of declaring a result statistically significant. Relying only on statistical significance can be problematic for many reasons:
  1. Statistical significance is not equivalent to practical significance;
  2. A really small effect can be declared significant just by collecting enough data.
  3. The idea of hypothesis testing is connected to the concept of p-values.
     - People are used to thinking that way mostly because that's what they learn in most introductory statistical courses. There is a long record of studies and essays showing that, more often than not, p-values are used and interpreted the wrong way, even by people who are using them on a daily basis.

57

## Effect Size

- Instead of doing hypothesis testing, we are going to take a different route and we are going to focus on estimating the **effect size**, that is, quantifying the difference between two groups.
  - One advantage of thinking in terms of effect size is that we move away from the yes-no questions like; Does it work?, Is there any effect? to the more nuance type of question like; How well does it work? How large/small is the effect? (The **effect size** is just a way to quantify the size of the difference between two groups.)

58

## Control Group and Treatment Groups

- Sometimes, when comparing groups, people talk about a control group and a treatment group (or maybe more than one control and treatment groups).
  - This makes sense, for example, when we want to test a new drug: because of the placebo effect and other reasons, we want to compare the new drug (the treatment) against a control group (a group not receiving the drug). In this case, we want to know how well one drug works compared to doing nothing (or, as is generally done, against the placebo effect).
  - One interesting alternative question will be to ask how good a new drug is compared with the (already approved) most popular drug to treat that illness. In such a case, the control group cannot be a placebo; it should be the other drug.

59

## An Example

- Bogus control groups are a splendid way to lie using statistics.
  - Imagine you work for a dairy product company that wants to sell overly sugared yogurts to kids by telling their dads and moms that this particular yogurt boosts the immune system or help their kids grown stronger. One way to cheat and falsely back up your claims with data is by using milk or even water as a control group, instead of another cheaper, less sugary, less marketed yogurt.
- When someone says something is harder, better, faster, stronger, remember to ask what the baseline used for the comparison was.

60

## Three Tools

- To compare groups, we must decide which feature (or features) we are going to use for the comparison. A very common feature is the mean of each group. Because we are Bayesian, we will work to obtain a posterior distribution of the differences of means between groups and not just a point-estimate of the differences.
- To help us see and interpret such a posterior, we are going to use three tools:
  - A posterior plot with a reference value (covered already)
  - The Cohen's d
  - The probability of superiority

61

## Cohen's d

- A common way to measure the effect size is **Cohen's d**, which is defined as follows:

$$\frac{\mu_2 - \mu_1}{\sqrt{\frac{\sigma_2^2 - \sigma_1^2}{2}}}$$

- According to this expression, the effect size is the difference of the means with respect to the pooled standard deviation of both groups.
  - Because we can get a posterior distribution of means and standard deviations, we can compute a posterior distribution of Cohen's d values. Of course, if we just need or want one single value, we can compute the mean of that posterior distribution and get a single Cohen's d value. Generally, when computing a pooled standard deviation, we take into account the sample size of each group explicitly, but the the above formula is omitting the sample size of both groups. The reason for this is that we are getting the values of the standard deviation from the posterior and thus we are already accounting for the standard deviations' uncertainty.

62

## Variability of Each Group by Their Standard Deviations

- Cohen's d is a way to measure the effect size, where the difference of the means are standardized by considering the pooled standard deviations of both groups.
- Cohen's d introduces the variability of each group by using their standard deviations. This is really important, as differences of one when you have a standard deviation of 0.1 seems large compared to the same difference when the standard deviation is 10. Also, a change of *x* units from one group to another could be explained by every individual data point changing exactly *x* units or by half of them not changing and the other half changing 2*x* units, and by many other combinations.
  - Thus, including the intrinsic variations of groups is a way to put the differences in context. Re-scaling (standardizing) the differences helps us make sense of the importance of the difference between groups, even when we are not very familiar with the scale used for the measurements.

63

## Z-Score

- A Cohen's d can be interpreted as a *Z*-score (a standard score). A Z-score is the signed number of standard deviations by which a value differs from the mean value of what is being observed or measured. Thus, a Cohen's d of 0.5 could be interpreted as a difference of 0.5 standard deviation of one group with respect to the other.

64

## Calibrate Based on Context

- Even when the differences of means are standardized, we may still need to calibrate ourselves based on the context of a given problem to be able to say if a given value is *big*, *small*, *medium*, and so on.
  - Fortunately, this calibration can be acquired with enough practice.
  - Just as an example, if we are used to performing several analyses for more or less the same type of problems, we can get used to a Cohen's d of say 1, so when we get a Cohen's d of say 2, we know we have something important (or someone made a mistake somewhere!).
  - If you do not have this practice yet, you can ask a domain expert for their valuable input. A very nice web page to explore what different values of Cohen's d look like is http://rpsychologist.com/d3/cohend.

65

## Probability of Superiority

- This is another way to report the effect size, and this is defined as the probability that a data point taken at random from one group has a larger value than one also taken at random from the other group. If we assume that the data we are using is distributed as a normal, we can compute the probability of superiority from the Cohen's d using the following expression:

$$ps = \phi\left(\frac{\delta}{\sqrt{2}}\right)$$

  - Here, $\phi$ is the cumulative normal distribution and $\delta$ is the Cohen's d. We can compute a point-estimate of the probability of superiority (what is usually reported), or we can compute the whole posterior distribution of values.
  - If we are OK with the normality assumption, we can use this formula to get the probability of superiority from the Cohen's d. Otherwise, as we have samples from the posterior, we can directly compute it.

66

## The Tips Dataset

- This data was reported for the first time by Bryant, P. G. and Smith, M (1995) in *Practical Data Analysis: Case Studies in Business Statistics*.
- We want to study the effect of the day of the week on the amount of tips at a restaurant. For this example, the different groups are the days.
  - Notice there is no control group or treatment group. If we wish, we can arbitrarily establish one day, for example, Thursday, as the reference or *control*. For now, let's start the analysis by loading the dataset as a pandas DataFrame using just one line of code.

```
tips = pd.read_csv('../data/tips.csv')
tips.tail()
```

67

## The Last Few Entries of Tips Dataframe

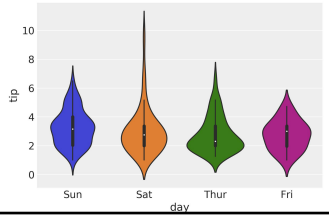
| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **239** | 29.03 | 5.92 | Male | No | Sat | Dinner | 3 |
| **240** | 27.18 | 2.00 | Female | Yes | Sat | Dinner | 2 |
| **241** | 22.67 | 2.00 | Male | Yes | Sat | Dinner | 2 |
| **242** | 17.82 | 1.75 | Male | No | Sat | Dinner | 2 |
| **243** | 18.78 | 3.00 | Female | No | Thurs | Dinner | 2 |

68

## violinplot Function

- From this DataFrame, we are only going to use the day and tip columns. We can plot our data using the violinplot function from seaborn:

```
sns.violinplot(x='day', y='tip', data=tips)
```




69

## Simplifying Things

- Just to simplify things, we are going to create three variables: the y variable, representing the tips, the idx variable, a categorical dummy variable to encode the days with numbers, that is, [0, 1, 2, 3] instead of [Thursday, Friday, Saturday, Sunday], and finally the groups variable, with the number of groups (4):

```
tip = tips['tip'].values
idx = pd.Categorical(tips['day'], categories=['Thur', 'Fri', 'Sat', 'Sun']).codes
groups = len(np.unique(idx))
```

70

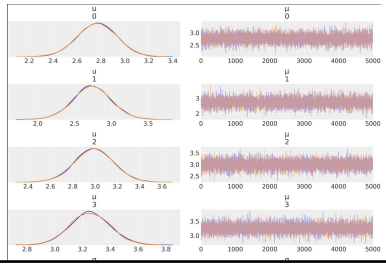
## Model for Tips Problem

- The model for this problem is almost the same as model_g; the only difference is that now $\mu$ and $\sigma$ are going to be vectors instead of scalar variables.
  - PyMC3 syntax is extremely helpful for this situation: instead of writing for loops, we can write our model in a vectorized way. This means that for the priors, we pass a shape argument and for the likelihood, we properly index the means and sds variables using the idx variable:

```
with pm.Model() as comparing_groups:
        μ = pm.Normal('μ', mu=0, sd=10, shape=groups)
        σ = pm.HalfNormal('σ', sd=10, shape=groups)
        y = pm.Normal('y', mu=μ[idx], sd=σ[idx], observed=tip)
        trace_cg = pm.sample(5000)
az.plot_trace(trace_cg)
```

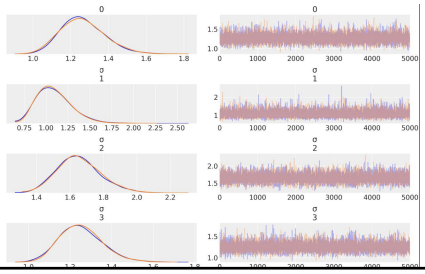
71

## Posteriors of $\mu$ (for Different Days)



72

## Posteriors of $\sigma$ (for Different Days)



73

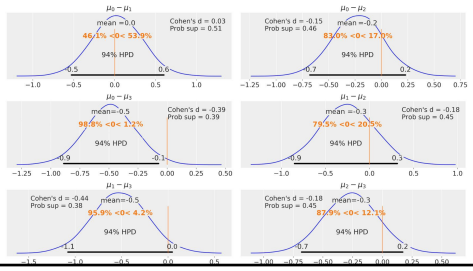
## Plotting the Difference

- The following code is just a way of plotting the difference without repeating the comparison. Instead of plotting the *all-against-all* matrix, we are just plotting the upper triangular portion:

```
dist = stats.norm()

_, ax = plt.subplots(3, 2, figsize=(14, 8), constrained_layout=True)

comparisons = [(i, j) for i in range(4) for j in range(i+1, 4)]
pos = [(k, l) for k in range(3) for l in (0, 1)]

for (i, j), (k, l) in zip(comparisons, pos):
        means_diff = trace_cg['μ'][:, i] - trace_cg['μ'][:, j]
        d_cohen = (means_diff / np.sqrt((trace_cg['σ'][:, i]**2 + trace_cg['σ'][:, j]**2) /
2)).mean()
        ps = dist.cdf(d_cohen/(2**0.5))
        az.plot_posterior(means_diff, ref_val=0,
        ax=ax[k, l]) ax[k, l].set_title(f'$\mu_{i}-\mu_{j}$')
        ax[k, l].plot(0, label=f"Cohen's d = {d_cohen:.2f}\nProb sup = {ps:.2f}", alpha=0)
        ax[k, l].legend()
```

74

## Posterior of Difference of $\mu$



75

## Result Interpretation

- One way to interpret these results is by comparing the reference value with the HPD interval. According to the previous diagram, we have only one case when the 94% HPD excludes the reference value of zero, that is, the difference in tips between Thursday and Sunday. For all the other examples, we cannot rule out a difference of zero (according to the HPD-reference-value-overlap criteria). But even for that case, is an average difference of ≈0.5 dollars large enough? Is that difference enough to accept working on Sunday and missing the opportunity to spend time with family or friends? Is that difference enough to justify averaging the tips over the four days and giving every waitress and waiter the same amount of tip money? Those kinds of questions cannot be answered by statistics; they can only be informed by statistics.

76

## Hierarchical Models

- Suppose we want to analyze the quality of water in a city, so we take samples by dividing the city into neighborhoods. We may think we have two options to analyze this data:
  - Study each neighborhood as a separated entity
  - Pool all the data together and estimate the water quality of the city as a single big group
- Both options could be reasonable, depending on what we want to know. We can justify the first option by saying we obtain a more detailed view of the problem, which otherwise could become invisible or less evident if we average the data. The second option can be justified by saying that if we pool the data, we obtain a bigger sample size and hence a more accurate estimation.
  - Both are good reasons, but we can do something else, something in-between. We can build a model to estimate the water quality of each neighborhood and, at the same time, estimate the quality of the whole city. This type of model is known as a **hierarchical model** or **multilevel model**, because we model the data using a hierarchical structure or one with multiple levels.

77

## How to Build Hierarchical Model?

- Instead of fixing the parameters of our priors to some constant numbers, we estimate them directly from the data by placing shared priors over them. These higher-level priors are often called **hyper-priors**, and their parameters **hyperparameters**; hyper means *over* in Greek. Of course, it is also possible to put priors over the hyper-priors and create as many levels as we want; the problem is that the model rapidly becomes difficult to understand.
  - Unless the problem really demands more structure, adding more levels *than necessary* does not help to make better inferences. On the contrary, we end up entangled in a web of hyper-priors and hyperparameters without the ability to assign any meaningful interpretation to them, partially spoiling the advantages of model-based statistics. After all, the main idea of building models is to make sense of data, and thus models should reflect (and take advantage of) the structure in the data.

78

## Water Quality Example

- To illustrate the main concepts of hierarchical models, we are going to use a toy model of the water quality example and we are going to use synthetic data.
  - Imagine we have collected water samples from three different regions of the same city and we have measured the lead content of water; samples with lead concentration above recommendations from the **World Health Organization** (**WHO**) are marked with zero and samples with values below the recommendations are marked with one.
  - This is just a pedagogic example; in a more realistic example, we would have a continuous measurement of lead concentration and probably many more groups. Nevertheless, for our current purposes, this example is good enough to uncover the details of hierarchical models.

79

## Generate Synthetic Data

- We can generate the synthetic data with the following code:

```
N_samples = [30, 30, 30]
G_samples = [18, 18, 18]

group_idx = np.repeat(np.arange(len(N_samples)), N_samples)
data = []
for i in range(0, len(N_samples)):
        data.extend(np.repeat([1, 0], [G_samples[i], N_samples[i] - G_samples[i]]))
```

- We are simulating an experiment where we have measured three groups, each one consisting of a certain number of samples; we store the total number of samples per group in the N_samples list. Using the G_samples list, we keep a record of the number of good quality samples per group. The rest of the code is there just to generate a list of the data, filled with zeros and ones.

80

## The Water Quality Model

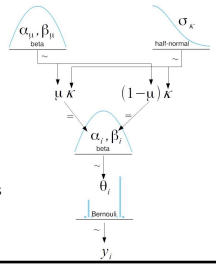
- The model is essentially the same one we used for the coin problem, except for two important features:
  - We have defined two hyper-priors that will influence the beta prior.
  - Instead of putting hyper-priors on the parameters $\alpha$ and $\beta$, we are indirectly defining them in terms of $\mu$, the mean of the beta distribution, and $\kappa$, the precision (the concentration) of the beta distribution. The precision is analog to the inverse of the standard deviation; the larger the value of $\kappa$, the more concentrated the beta distribution will be:

$$\mu \sim Beta(\alpha_\mu, \beta_\mu)$$
$$\kappa \sim |Normal(0, \sigma_\kappa)|$$
$$\alpha = \mu * \kappa$$
$$\beta = (1 - \mu) * \kappa$$
$$\theta_i \sim Beta(\alpha_i, \beta_i)$$
$$y_i \sim Bern(\theta_i)$$

81

## Kruschke Diagrams

- Notice we are using the subindex *i* to indicate that the model has groups with different values for some of the parameters.
  - That is, not all parameters are shared between the groups.
- Notice how in the right figure we have used the symbol = instead of $\sim$ to define $\alpha_i$ and $\beta_i$, once we know $\mu$ and $\kappa$ the values of $\alpha_i$ and $\beta_i$ becomes fully determined.
  - Accordingly, we call this type of variable deterministic in opposition to stochastic variables such as $\mu$, $\kappa$, or $\theta$.




82

## Parameterization

- Using the mean and precision is mathematically equivalent to using the $\alpha$ and $\beta$ parameterization, implying we should get the same results. So, why are we taking this detour instead of the more direct route? There are two reasons for this:
  1. The mean and precision parameterization, while mathematically equivalent, is numerically better suited for the sampler, and thus we are more confident of the results PyMC3 is returning, we will learn about the reason for and intuition behind this statement in Chapter 8, Inference Engines.
  2. This is a concrete example showing that there is potentially more than one way to express a model. Mathematically equivalent implementations could have nevertheless practical differences; the efficiency of the sampler is one aspect to consider, while another one could be the interpretability of the model. For some specific problems or particular audiences, it could be a better choice to report the mean and precision of the beta distribution than the $\alpha$ and $\beta$ parameters.

83

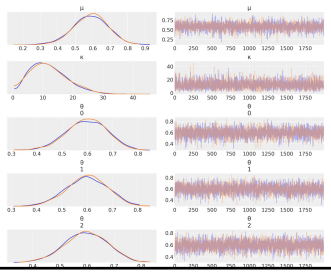
## Obtain Sample for Water Quality Model

- Let's implement and solve the model in PyMC3 so that we can keep discussing hierarchical models:

```
with pm.Model() as model_h:
        μ = pm.Beta('μ', 1., 1.)
        κ = pm.HalfNormal('κ', 10)
        θ = pm.Beta('θ', alpha=μ*κ, beta=(1.0-μ)*κ, shape=len(N_samples))
        y = pm.Bernoulli('y', p=θ[group_idx], observed=data)

        trace_h = pm.sample(2000)

az.plot_trace(trace_h)
```

84

## The Results



85

## Shrinkage

- To show you one of the main consequences of hierarchical models, we will require your assistance, so please join us in a brief experiment. We will need you to print and save the summary using az.summary(trace_h). Then, We want you to re-run the model two more times after making small changes to the synthetic data, and always keep a record of the summary. In total, we will have three runs:
  - One run setting all the elements of G_samples to 18
  - One run setting all the elements of G_samples to 3
  - One last run setting one element to 18 and the other two to 3

86

## Experiment Result

- Before continuing, please take a moment to think about the outcome of this experiment. Focus on the estimated mean value of $\theta_i$ in each experiment. Based on the first two runs of the model, could you predict the outcome for the third case?
- If we put the result in a table, we get something more or less like this; remember that small variations could occur due to the stochastic nature of the sampling process:

| G_samples | $\theta$ (mean) |
|---|---|
| 18, 18, 18 | 0.6, 0.6, 0.6 |
| 3, 3, 3 | 0.11, 0.11, 0.11 |
| 18, 3, 3 | 0.55, 0.13, 0.13 |

87

## What Happens?

- In the first row, we can see that for a dataset of 18 good samples out of 30 we get a mean value for $\theta$ of 0.6; remember that now the mean of $\theta$ is a vector of three elements, one per group. Then, on the second row, we have only 3 good samples out of 30 and the mean of $\theta$ is 0.11. At the end, on the last row, we get something interesting and probably unexpected. Instead of getting a mix of the mean estimates of $\theta$ from the other rows, such as 0.6, 0.11, and 0.11, we get different values, namely 0.55, 0.13, and 0.13. What happened? Maybe a convergence problem or some error with the model specification?
  - Nothing like that—we get our estimates shrunk toward the common mean. And this is totally OK, indeed this is just a consequence of our model: by using hyper-priors, we are estimating the parameters of the beta prior distribution from the data, and each group is informing the rest, which in turn is informed by the estimation of the others. Put in a more succinct way, the groups are effectively sharing information through the hyper-prior. As a result, we are observing what is known as shrinkage. This effect is the consequence of partially pooling the data; we are modeling the groups not as independent from each other, nor as a single big group. Instead, we have something in the middle.

88

## Why is Shrinkage Useful?

- Why is this useful? Because having shrinkage contributes to more stable inferences. This is, in many ways, similar to what we saw with the Student's t-distribution and the outliers, where we used heavy tail distribution results in a model that is more robust or less responsive to data points away from the mean. Introducing hyper-priors and hence inferences at a higher level results in a more conservative model, one that is less responsive to extreme values in individual groups.
  - To illustrate this, imagine that the samples size are different from each neighborhood, some smaller, some larger; the smaller the sample size, the easier it is to get bogus results. At an extreme, if you take only one sample in a given neighborhood, you may just hit the only really old lead pipe in the whole neighborhood or, on the contrary, the only one made out of PVC. In one case, you will overestimate the bad quality and in the other underestimate it.
  - Under a hierarchical model, the miss-estimation of one group will be ameliorated by the information provided by the other groups. Of course, a larger sample size will also do the trick but, more often than not, that is not an option.

89

## Amount of Shrinkage

- The amount of shrinkage depends, of course, on the data; a group with more data will pull the estimate of the other groups harder than a group with fewer data points. If several groups are similar and one group is different, the similar groups are going to inform the others of their similarity and reinforce a common estimation, while they are going to pull toward them the estimation for the less similar group; this is exactly what we saw in the previous example.
- The hyper-priors also have a part in modulating the amount of shrinkage. We can effectively use an informative prior to shrink our estimate to some reasonable value if we have trustworthy information about the group-level distribution. Nothing prevents us from building a hierarchical model with just two groups – but we would prefer to have several groups. Intuitively, the reason is that getting shrinkage is like assuming each group is a data point, and we are estimating the standard deviation at the group level. Generally, we do not trust an estimation with too few data points unless we have a strong prior to inform our estimation. Something similar is true for a hierarchical model.

90

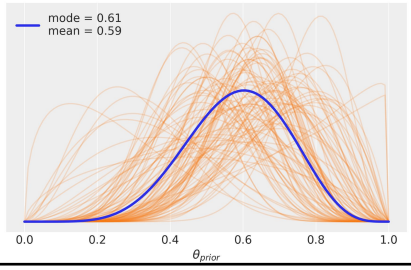
## Plotting Estimated Prior

- We may also be interested in seeing what the estimated prior looks like. One way to do this is as follows:

```
x = np.linspace(0, 1, 100)
for i in np.random.randint(0, len(trace_h), size=100):
        u = trace_h['μ'][i]
        k = trace_h['κ'][i]
        pdf = stats.beta(u*k, (1.0-u)*k).pdf(x)
        plt.plot(x, pdf, 'C1', alpha=0.2)
u_mean = trace_h['μ'].mean()
k_mean = trace_h['κ'].mean()
dist = stats.beta(u_mean*k_mean, (1.0-u_mean)*k_mean)
pdf = dist.pdf(x)
mode = x[np.argmax(pdf)]
mean = dist.moment(1)
plt.plot(x, pdf, lw=3, label=f'mode = {mode:.2f}\nmean = {mean:.2f}')
plt.yticks([])
plt.legend()
plt.xlabel('$θ_{prior}$')
plt.tight_layout()
```

91

## The Estimated Prior in Water Quality Model




92

## One More Example – Chemical Shift Data

- Once again, we have chemical shift data. This data comes from a set of protein molecules. To be precise, we should say the chemical shifts are from the $^{13}C_{\alpha}$ nuclei of proteins as this is an observable we measure only for certain types of atomic nuclei. Proteins are made from sequences of 20 building blocks known as amino acidic residues. Each amino acid can appear in the sequence zero or more times, and sequences can vary from a few amino acids to hundreds or even thousands of them. Each amino acid has one and only one $^{13}C_{\alpha}$, so we can confidently associate each chemical shift to a particular amino acidic residue in a protein. Furthermore, each one of these 20 amino acids has different chemical properties that contribute to the biological properties of the protein; some can have net electric charges and some can only be neutral; some like to be surrounded by water molecules while others prefer the company of the same type or similar type of amino acids; and so on. The key aspect is that they are similar but not equal, and hence it may sound reasonable or even natural to base any chemical shift-related inference in terms of 20 groups, as defined by the amino acids types. You can learn more about proteins with this excellent video: https://www.youtube.com/watch?v=wvTv8TqWC48.

93

## Simplification

- For the sake of this example, let's simplifying things a little bit here. In practice, experiments are messy, and there is always a chance that we do not get a complete record of chemical shifts. One common problem is signal overlapping, that is, the experiment does not have enough resolution to distinguish two or more close signals. For this example, I just removed those cases, so we will assume that the dataset is complete.
- In the following code block, we load the data into a DataFrame; please take a moment to inspect the DataFrame. You will see four columns: the first one is the ID of a protein – if you feel curious, you can access a lot of information about a protein by using the ID in this page: https://www.rcsb.org/. The second column includes the name of the amino acid, using a standard three-letter code, while the following columns correspond to theoretical computed chemical shift values (using quantum chemical computations) and the experimentally measured chemical shifts.

94

## How Well Theoretical Computations Match Experimental Measures

- The motivation for this example is to compare the differences to access, among other reasons, how well the theoretical computations are reproducing the experimental measures. For that reason, we are computing the pandas' series diff:

```
cs_data = pd.read_csv('../data/chemical_shifts_theo_exp.csv')
diff = cs_data.theo.values - cs_data.exp.values
idx = pd.Categorical(cs_data['aa']).codes
groups = len(np.unique(idx))
```

- To see the difference between a hierarchical and non-hierarchical model, we are going to build two models. The first one is basically the same as the comparing_groups model:

```
with pm.Model() as cs_nh:
        μ = pm.Normal('μ', mu=0, sd=10, shape=groups)
        σ = pm.HalfNormal('σ', sd=10, shape=groups)
        y = pm.Normal('y', mu=μ[idx], sd=σ[idx], observed=diff)
        trace_cs_nh = pm.sample(1000)
```

95

## Hierarchical Model and Non-Hierarchical Model

- Now, we will build the hierarchical version of the model. We are adding two hyper-priors: one for the mean of $\mu$ and one for the standard deviation of $\mu$. We are leaving $\sigma$ without hyper-priors. This is just a model choice; I am deciding on a simpler model just for pedagogical purposes. You may face a problem where this seems unacceptable and you may consider it necessary to add a hyper-prior for ; feel free to do that:

```
with pm.Model() as cs_h:
        # hyper_priors
        μ_μ = pm.Normal('μ_μ', mu=0, sd=10)
        σ_μ = pm.HalfNormal('σ_μ', 10)

        # priors
        μ = pm.Normal('μ', mu=μ_μ, sd=σ_μ, shape=groups)
        σ = pm.HalfNormal('σ', sd=10, shape=groups)

        y = pm.Normal('y', mu=μ[idx], sd=σ[idx], observed=diff)
        trace_cs_h = pm.sample(1000)
```

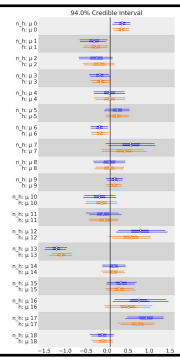
96

## plot_forest Function

- We are going to compare the results using ArviZ's plot_forest function. We can pass more than one model to this function. This is useful when we want to compare the values of parameters from different models such as with the present example. Notice that we are passing several arguments to plot_forest to get the plot that we want, like combined=True to merge results from all the chains. Please explore the rest of the arguments:

```
_, axes = az.plot_forest([trace_cs_nh, trace_cs_h], model_names=['n_h', 'h'],
                         var_names='μ', combined=False, colors='cycle')
y_lims = axes[0].get_ylim()
axes[0].vlines(trace_cs_h['μ_μ'].mean(), *y_lims)
```

97




98

## Figure Explanation

- We have a plot for the 40 estimated means, one per amino acid (20) multiplied by two as we have two models. We also have their 94% credible intervals and the interquantile range (the central 50% of the distribution). The vertical (black) line is the global mean according to the hierarchical model. This value is close to zero, as expected for theoretical values reproducing experimental ones.
  - The most relevant part of this plot is that the estimates from the hierarchical model are *pulled* toward the partially-pooled mean, or equivalently they are *shrunken* with respect to the un-pooled estimates. You will also notice that the effect is more notorious for those groups farther away from the mean (such as 13) and that the uncertainty is on par or smaller than that from the non-hierarchical model. The estimates are partially pooled because we have one estimate for each group, but estimates for individual groups restrict each other through the hyper-prior.

99

## Beauty of Hierarchical Model and Looking Forward

- Therefore, we get an intermediate situation between having a single group, all chemical shifts together, and having 20 separated groups, one per amino acid. That is the beauty of hierarchical models.
  - We can certainly say that *hierarchical models are one great idea!*
- In the following chapters, we will keep building hierarchical models and learn how to use them to build better models. We will also discuss how hierarchical models are related to the pervasive overfitting/underfitting issue in statistics and machine learning in Chapter 5, *Model Comparison*. In Chapter 8, *Inference Engines*, we will discuss some technical problems that we may find when sampling from hierarchical models and how to diagnose and fix those problems.

100