# L9. Computational Bayesian Statistics (Applications)

## Purpose

- PyMC3 is a probabilistic programming module for Python that allows users to fit Bayesian models using a variety of numerical methods, most notably Markov chain Monte Carlo (MCMC) and variational inference (VI).
- Its flexibility and extensibility make it applicable to a large suite of problems.
- Along with core model specification and fitting functionality, PyMC3 includes functionality for summarizing output and for model diagnostics.

## Features

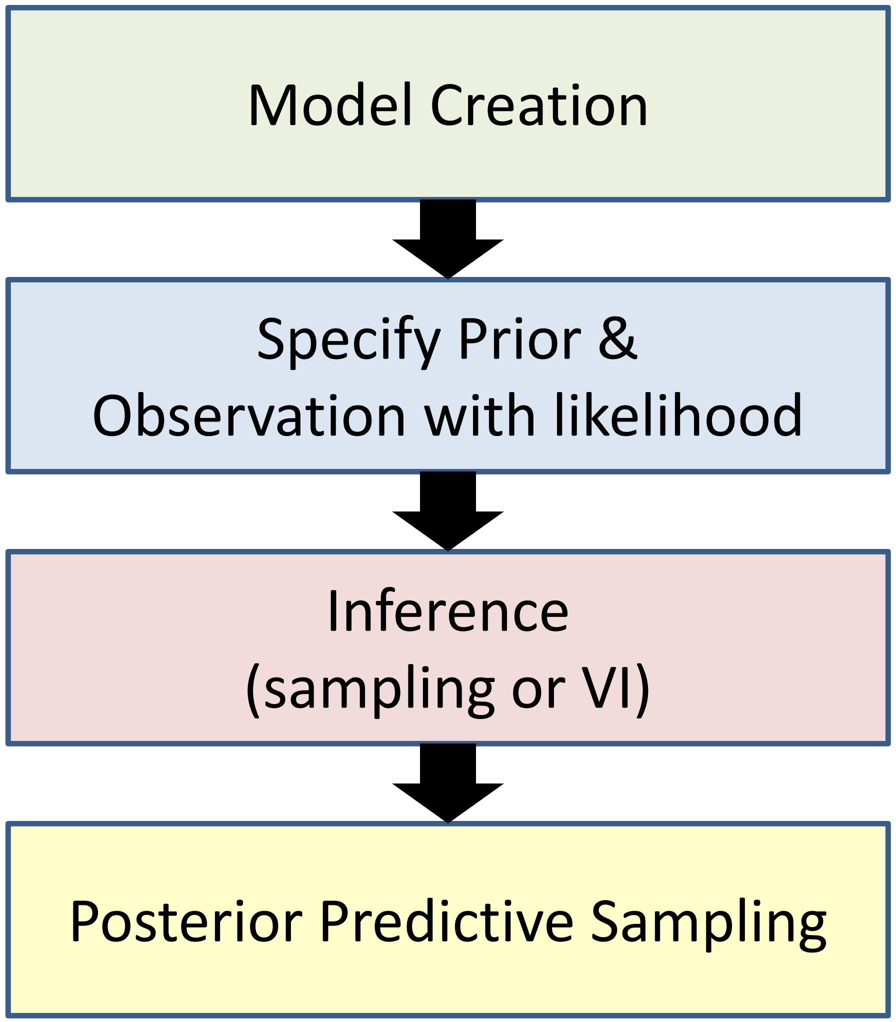
- PyMC3 strives to make Bayesian modeling as simple and painless as possible, allowing users to focus on their scientific problem, rather than on the methods used to solve it. Here is a partial list of its features:
  - ✓ Modern methods for fitting Bayesian models, including MCMC and VI.
  - ✓ Includes a large suite of well-documented statistical distributions.
  - ✓ Uses Theano as the computational backend, allowing for fast expression evaluation, automatic gradient calculation, and GPU computing.
  - ✓ Built-in support for Gaussian process modeling.
  - ✓ Model summarization and plotting.
  - ✓ Model checking and convergence detection.
  - ✓ Extensible: easily incorporates custom step methods and unusual probability distributions.
  - ✓ Bayesian models can be embedded in larger programs, and results can be analyzed with the full power of Python.

## Features

- PyMC3 strives to make Bayesian modeling as simple and painless as possible, allowing users to focus on their scientific problem, rather than on the methods used to solve it. Here is a partial list of its features:

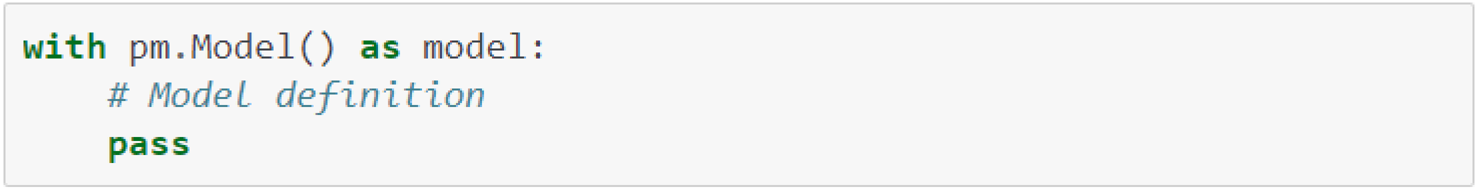
  - ✓ Modern methods for fitting Bayesian models, including MCMC and VI.
  - ✓ Includes a large suite of well-documented statistical distributions.
  - ✓ Uses Theano as the computational backend, allowing for fast expression evaluation, automatic gradient calculation, and GPU computing.
  - ✓ Built-in support for Gaussian process modeling.
  - ✓ Model summarization and plotting.
  - ✓ Model checking and convergence detection.
  - ✓ Extensible: easily incorporates custom step methods and unusual probability distributions.
  - ✓ Bayesian models can be embedded in larger programs, and results can be analyzed with the full power of Python.

## Procedure

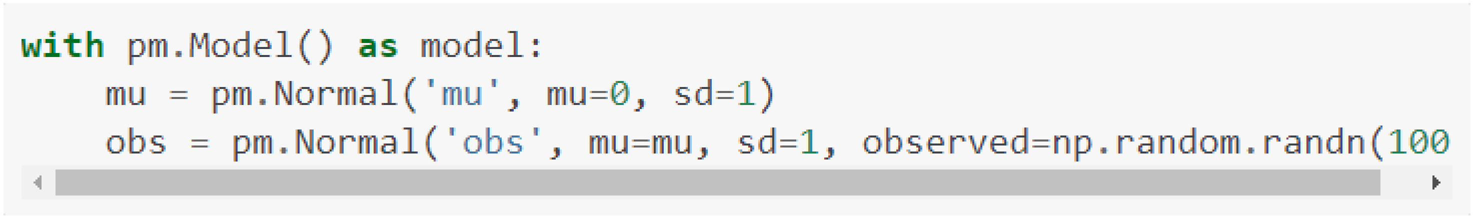

Model Creation
Specify Prior &
Observation with likelihood
Inference
(sampling or VI)
Posterior Predictive Sampling

## 1. Model Creation

- Models in PyMC3 are centered around the **model** class. It has references to all random variables (RVs) and computes the model logp and its gradients. Usually, you would instantiate it as part of a **with** context:



```
with pm.Model() as model:
    # Model definition
    pass
```



```
with pm.Model() as model:
    mu = pm.Normal('mu', mu=0, sd=1)
    obs = pm.Normal('obs', mu=mu, sd=1, observed=np.random.randn(100
```

## 2. Probability Distribution

- Every probabilistic program consists of observed and unobserved Random Variables (RVs).
  - ✓ Observed RVs are defined via likelihood distributions,
  - ✓ Unobserved RVs are defined via prior distributions.
  - ✓ In PyMC3, probability distributions are available from the main module space:

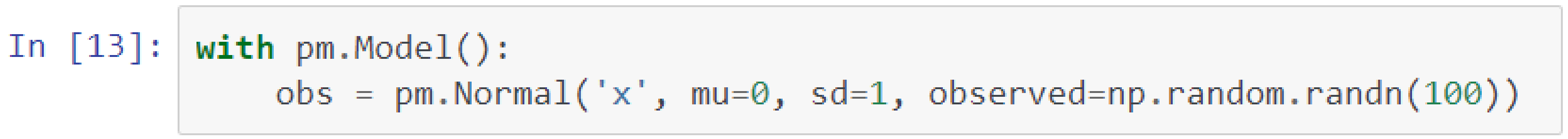
- In the PyMC3 module, the structure for probability distributions looks like this:

pymc3.distributions - continuous - discrete - timeseries - mixture

## 2. Probability Distribution

### 1. Observed Random Variable

- Observed RVs are defined just like unobserved RVs but require data to be passed into the **observed** keyword argument:



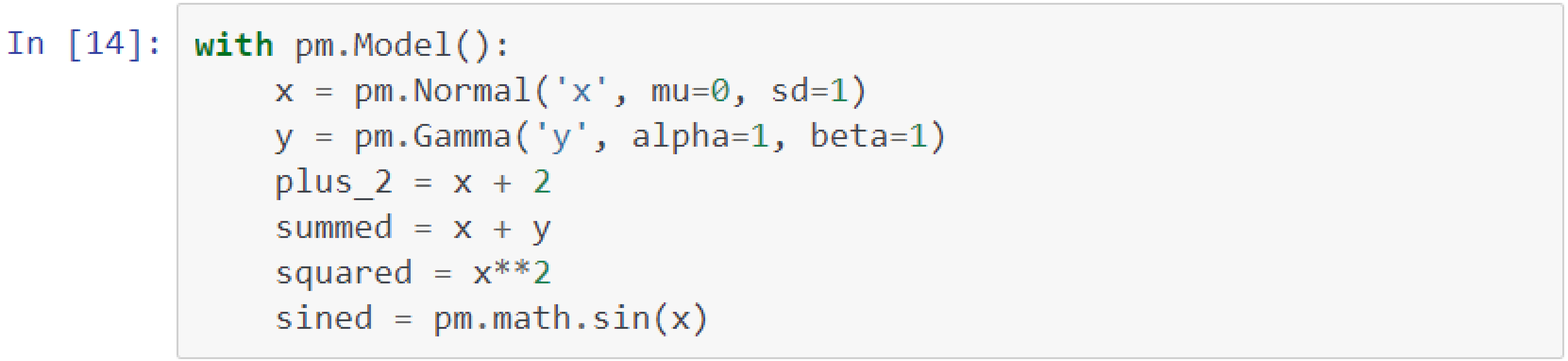
```
In [13]: with pm.Model():
             obs = pm.Normal('x', mu=0, sd=1, observed=np.random.randn(100))
```

✓ **observed** supports lists, numpy.ndarray, theano and pandas data structures.

## 2. Probability Distribution

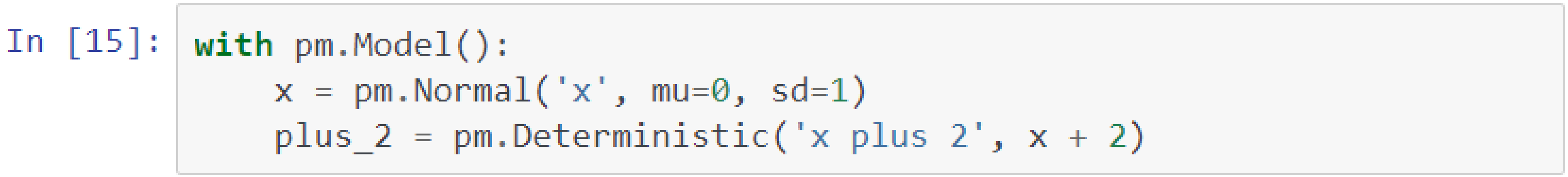
### 2. Deterministic Transform

- PyMC3 allows you to freely do algebra with RVs in all kinds of ways:



```
In [14]: with pm.Model():
             x = pm.Normal('x', mu=0, sd=1)
             y = pm.Gamma('y', alpha=1, beta=1)
             plus_2 = x + 2
             summed = x + y
             squared = x**2
             sined = pm.math.sin(x)
```

- While these transformations work seamlessly, their results are not stored automatically. Thus, if you want to keep track of a transformed variable, you have to use **pm.Determinisitc**

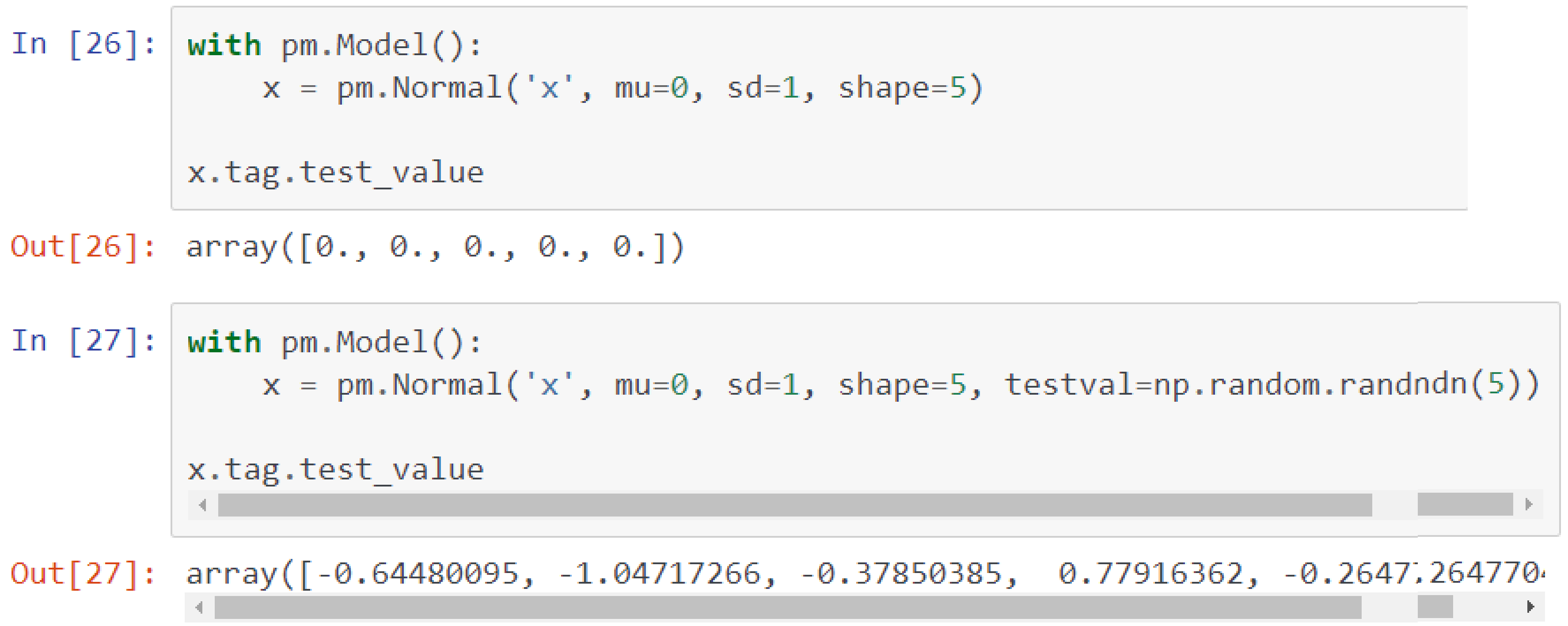


```
In [15]: with pm.Model():
             x = pm.Normal('x', mu=0, sd=1)
             plus_2 = pm.Deterministic('x plus 2', x + 2)
```

## 2. Probability Distribution

### 3. Initialization with test_values

- While PyMC3 tries to automatically initialize models it is sometimes helpful to define initial values for RVs. This can be done via the **testval** kwarg:



```
In [26]: with pm.Model():
             x = pm.Normal('x', mu=0, sd=1, shape=5)

         x.tag.test_value

Out[26]: array([0., 0., 0., 0., 0.])

In [27]: with pm.Model():
             x = pm.Normal('x', mu=0, sd=1, shape=5, testval=np.random.randndn(5))

         x.tag.test_value

Out[27]: array([-0.64480095, -1.04717266, -0.37850385,  0.77916362, -0.2647:264770
```

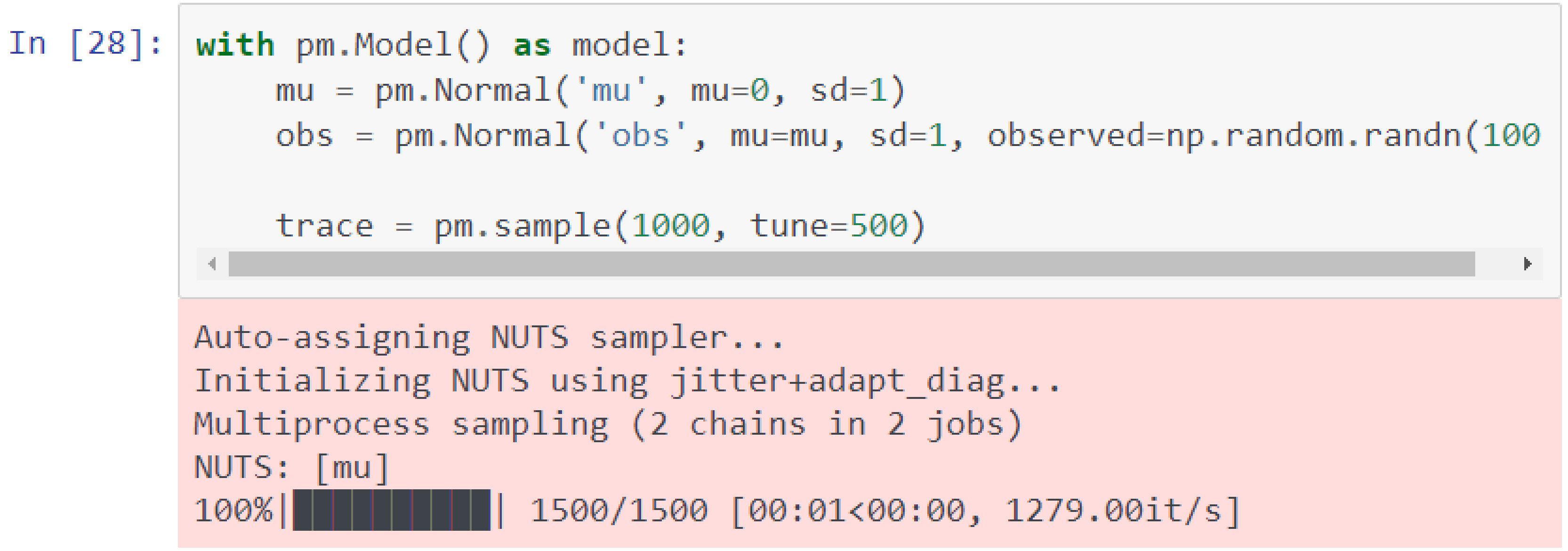
## 3. Inference

- Once we have defined our model, we have to perform inference to approximate the posterior distribution. PyMC3 supports two broad classes of inference: **sampling** and **variational inference**.

## 3. Inference

### 1. Sampling

- The main entry point to MCMC sampling algorithms is via the **pm.sample()** function. By default, this function tries to auto-assign the right sampler(s) and auto-initialize if you don’t pass anything.



```
In [28]: with pm.Model() as model:
             mu = pm.Normal('mu', mu=0, sd=1)
             obs = pm.Normal('obs', mu=mu, sd=1, observed=np.random.randn(100

             trace = pm.sample(1000, tune=500)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [mu]
100%|██████████| 1500/1500 [00:01<00:00, 1279.00it/s]
```

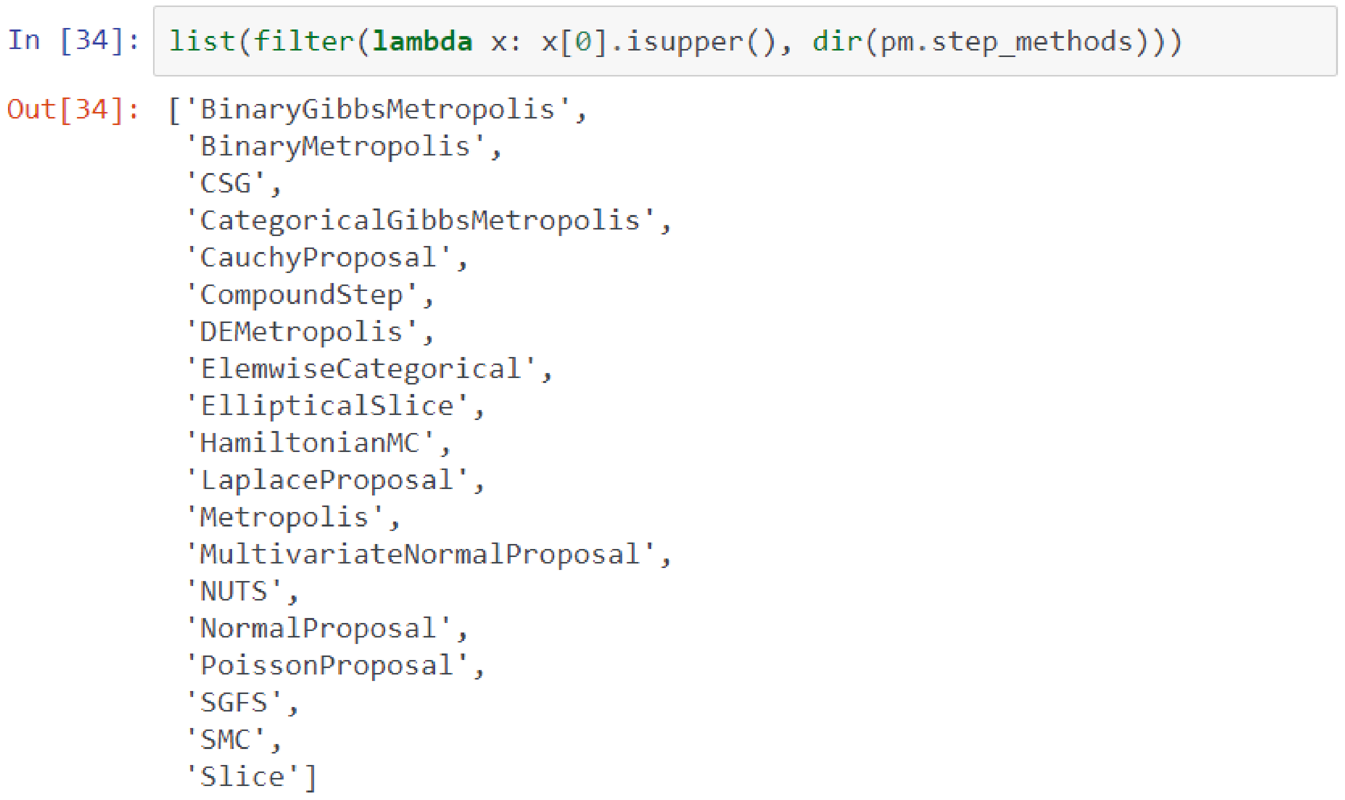
✓ PyMC3 assigns the NUTS sampler, which is very efficient even for complex models.

✓ Here we draw 1000 samples from the posterior and allow the sampler to adjust its parameters in an additional 500 iterations. These 500 samples are discarded by default:

## 3. Inference

### 1. Sampling

- PyMC3, offers a variety of other samplers, found in **pm.step_methods**



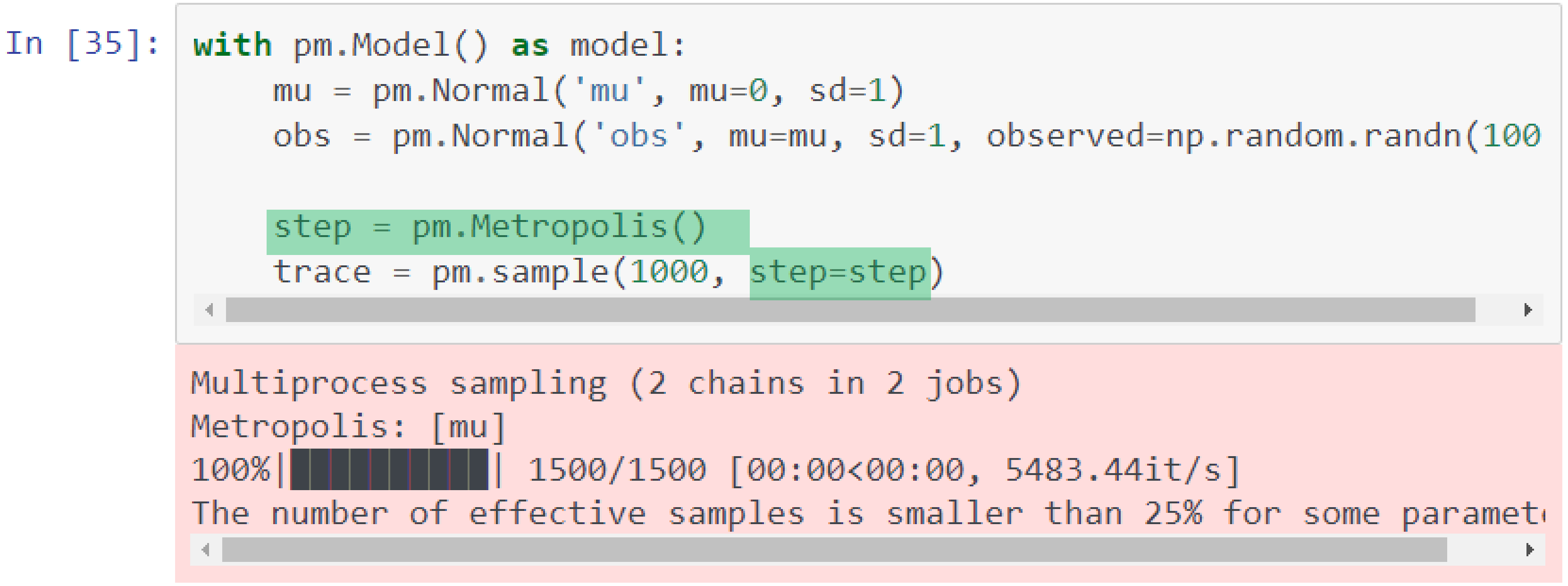
```
In [34]: list(filter(lambda x: x[0].isupper(), dir(pm.step_methods)))

Out[34]: ['BinaryGibbsMetropolis',
          'BinaryMetropolis',
          'CSG',
          'CategoricalGibbsMetropolis',
          'CauchyProposal',
          'CompoundStep',
          'DEMetropolis',
          'ElemwiseCategorical',
          'EllipticalSlice',
          'HamiltonianMC',
          'LaplaceProposal',
          'Metropolis',
          'MultivariateNormalProposal',
          'NUTS',
          'NormalProposal',
          'PoissonProposal',
          'SGFS',
          'SMC',
          'Slice']
```

- Commonly used step-methods besides **NUTS** are Metropolis and Slice. For almost all continuous models, “NUTS” should be preferred.
  - ✓ The **N**o-**U**-**T**urn **S**ampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo (Matthew D. Hoffman, Andrew Gelman, 2014)

## 3. Inference

### 1. Sampling

- other sampling methods can be passed to sample:



```
In [35]: with pm.Model() as model:
             mu = pm.Normal('mu', mu=0, sd=1)
             obs = pm.Normal('obs', mu=mu, sd=1, observed=np.random.randn(100

             step = pm.Metropolis()
             trace = pm.sample(1000, step=step)
```

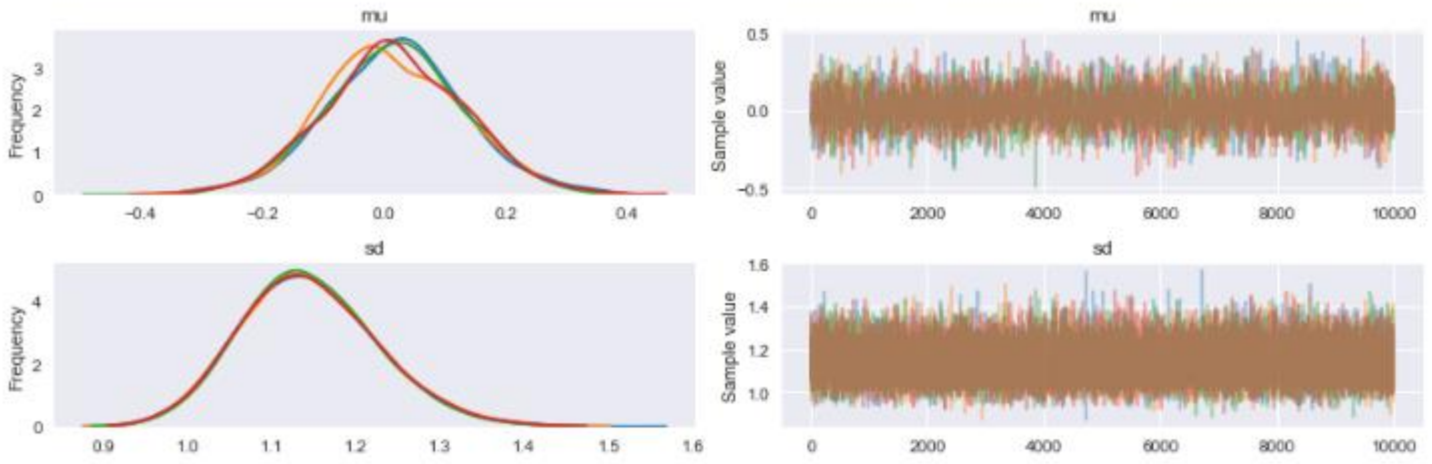
```
Multiprocess sampling (2 chains in 2 jobs)
Metropolis: [mu]
100%|██████████| 1500/1500 [00:00<00:00, 5483.44it/s]
The number of effective samples is smaller than 25% for some paramet
```

## 3. Inference

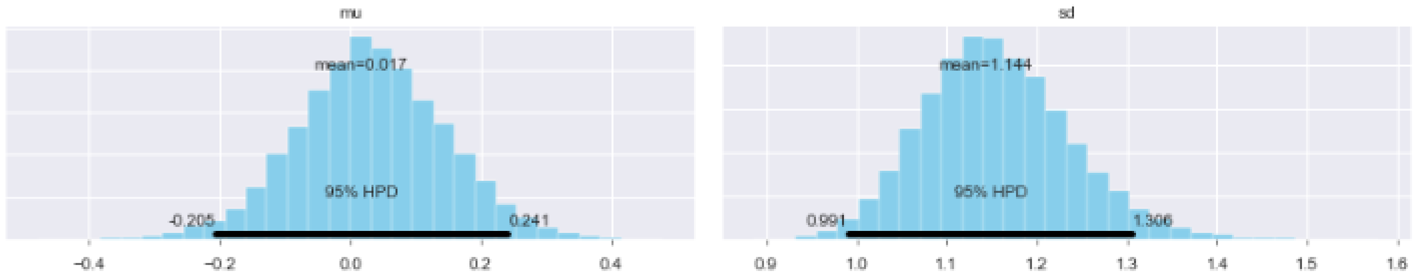
### 2. Analyzing Sampling Results

- The most common used plot to analyze sampling results is the so-called trace-plot.

```
In [37]: pm.traceplot(trace);
```



```
In [40]: pm.plot_posterior(trace);
```

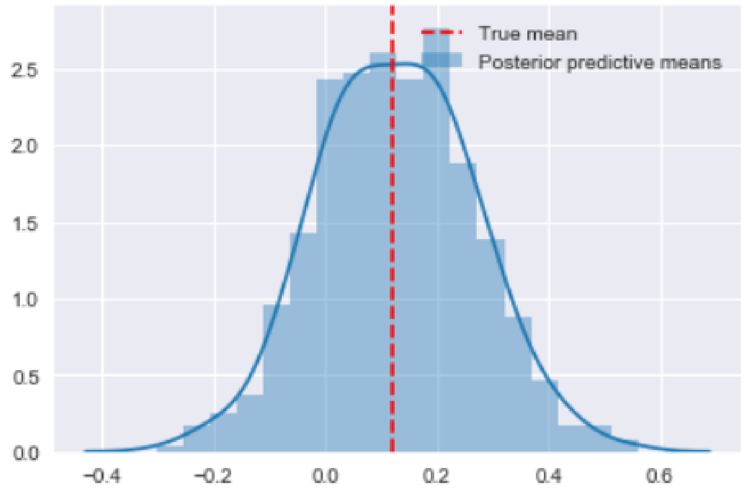


## 4. Posterior Predictive Sampling

- **sample_posterior_predictive()** returns a dict with a key for every observed node:

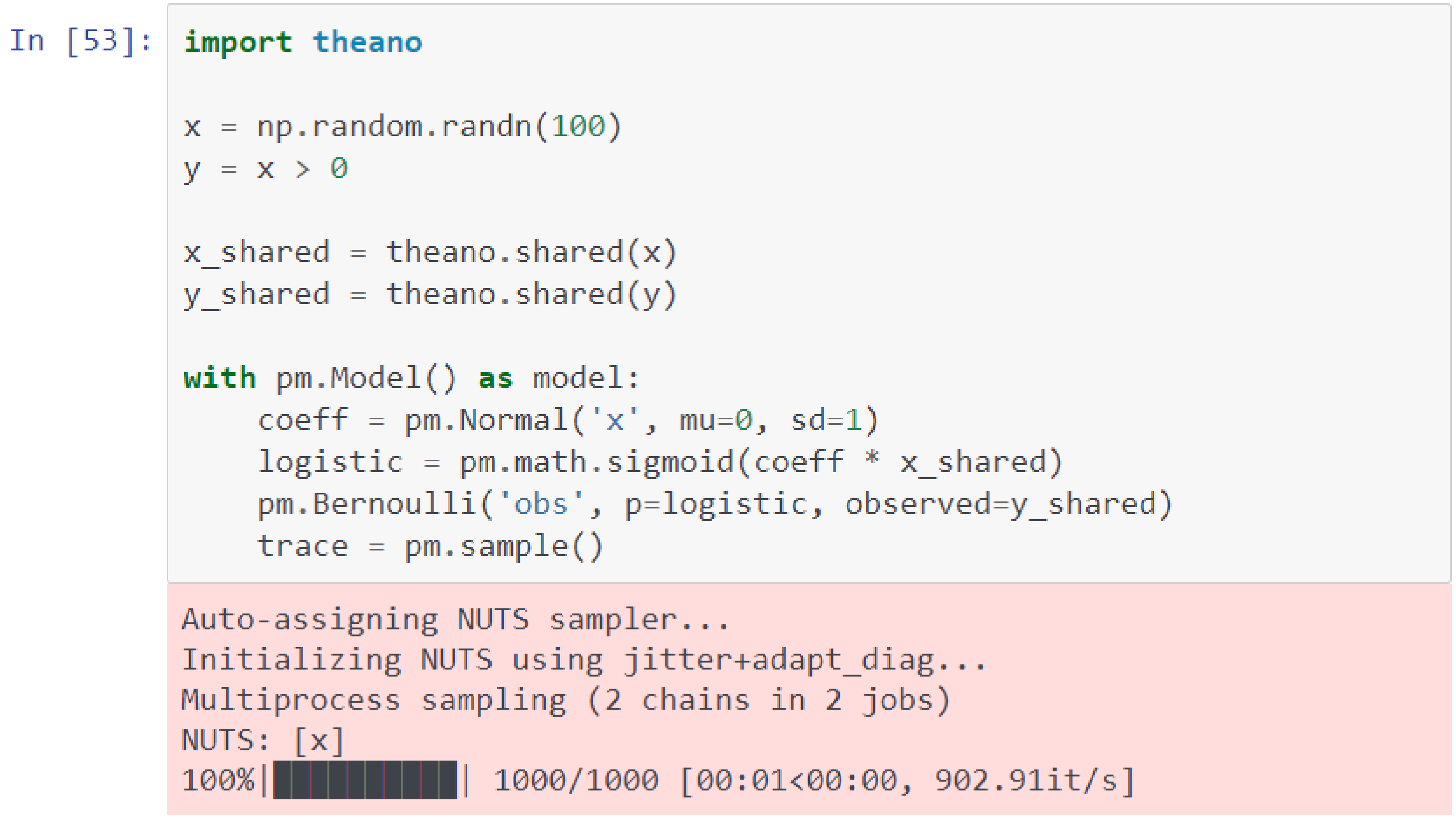
```
In [51]: post_pred['obs'].shape

Out[51]: (500, 100)
```

```
In [52]: plt.figure()
         ax = sns.distplot(post_pred['obs'].mean(axis=1), label='Posterior predictive means')
         ax.axvline(data.mean(), color='r', ls='--', label='True mean')
         ax.legend();
```



## 4. Posterior Predictive Sampling

- In many cases you want to predict on unseen / hold-out data. This is especially relevant in Probabilistic Machine Learning and Bayesian Deep Learning.
- When you pass data directly into a model, you are giving Theano permission to treat this data as a constant and optimize it away as it sees fit

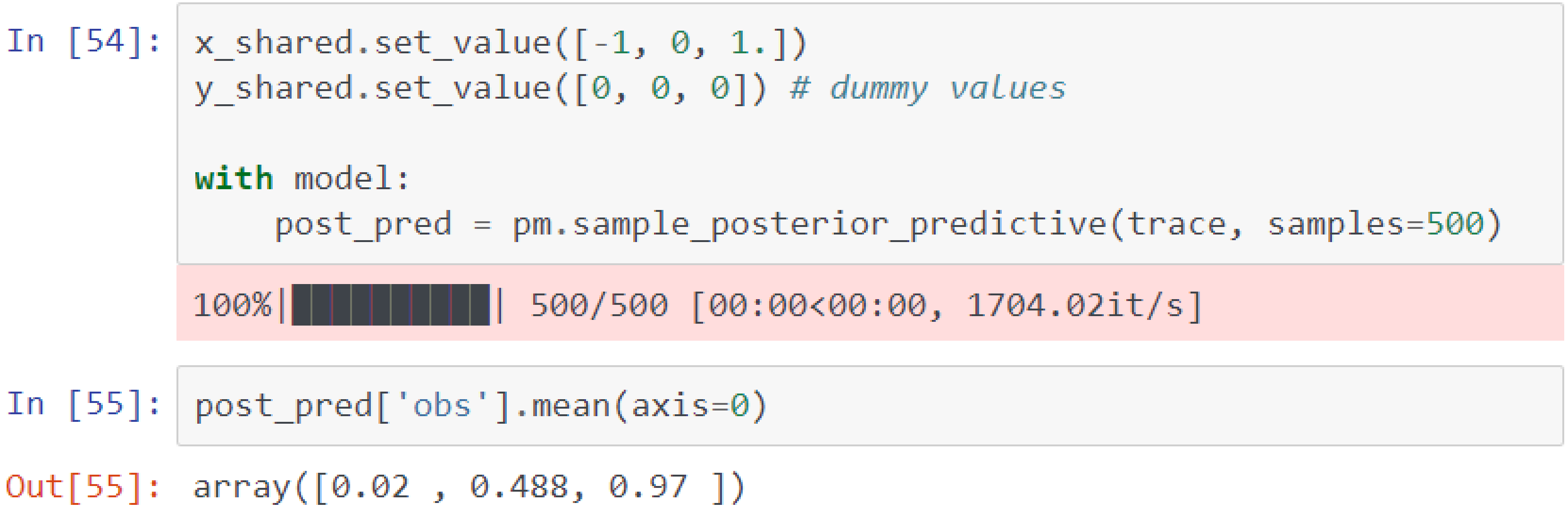


```
In [53]: import theano

         x = np.random.randn(100)
         y = x > 0

         x_shared = theano.shared(x)
         y_shared = theano.shared(y)

         with pm.Model() as model:
             coeff = pm.Normal('x', mu=0, sd=1)
             logistic = pm.math.sigmoid(coeff * x_shared)
             pm.Bernoulli('obs', p=logistic, observed=y_shared)
             trace = pm.sample()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [x]
100%|██████████| 1000/1000 [00:01<00:00, 902.91it/s]
```

## 4. Posterior Predictive Sampling

- In many cases you want to predict on unseen / hold-out data. This is especially relevant in Probabilistic Machine Learning and Bayesian Deep Learning.
- Now assume we want to predict on unseen data. For this we have to change the values of **x_shared** and **y_shared**



```
In [54]: x_shared.set_value([-1, 0, 1.])
         y_shared.set_value([0, 0, 0]) # dummy values

         with model:
             post_pred = pm.sample_posterior_predictive(trace, samples=500)

         100%|██████████| 500/500 [00:00<00:00, 1704.02it/s]

In [55]: post_pred['obs'].mean(axis=0)

Out[55]: array([0.02 , 0.488, 0.97 ])
```

## 3. PyMC3 Applications: GLMs

## 1. Bayesian Linear Regression Using PyMC3

- **Likelihood (observation model)**:

$$Y_i \sim N(\mu_i, \sigma^2)$$

where

$$\mu_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \cdots + \beta_n x_{in}$$

$$p(y_1, \dots, y_m | \beta_0, \dots, \beta_n) = \prod_{i=1}^{m} N(y_i; \mu_i, \sigma^2)$$

- **Prior**:

$$p(\beta_0, \dots, \beta_n) = \mathrm{N}(\mathbf{b}_o, \mathbf{V}_o)$$

where

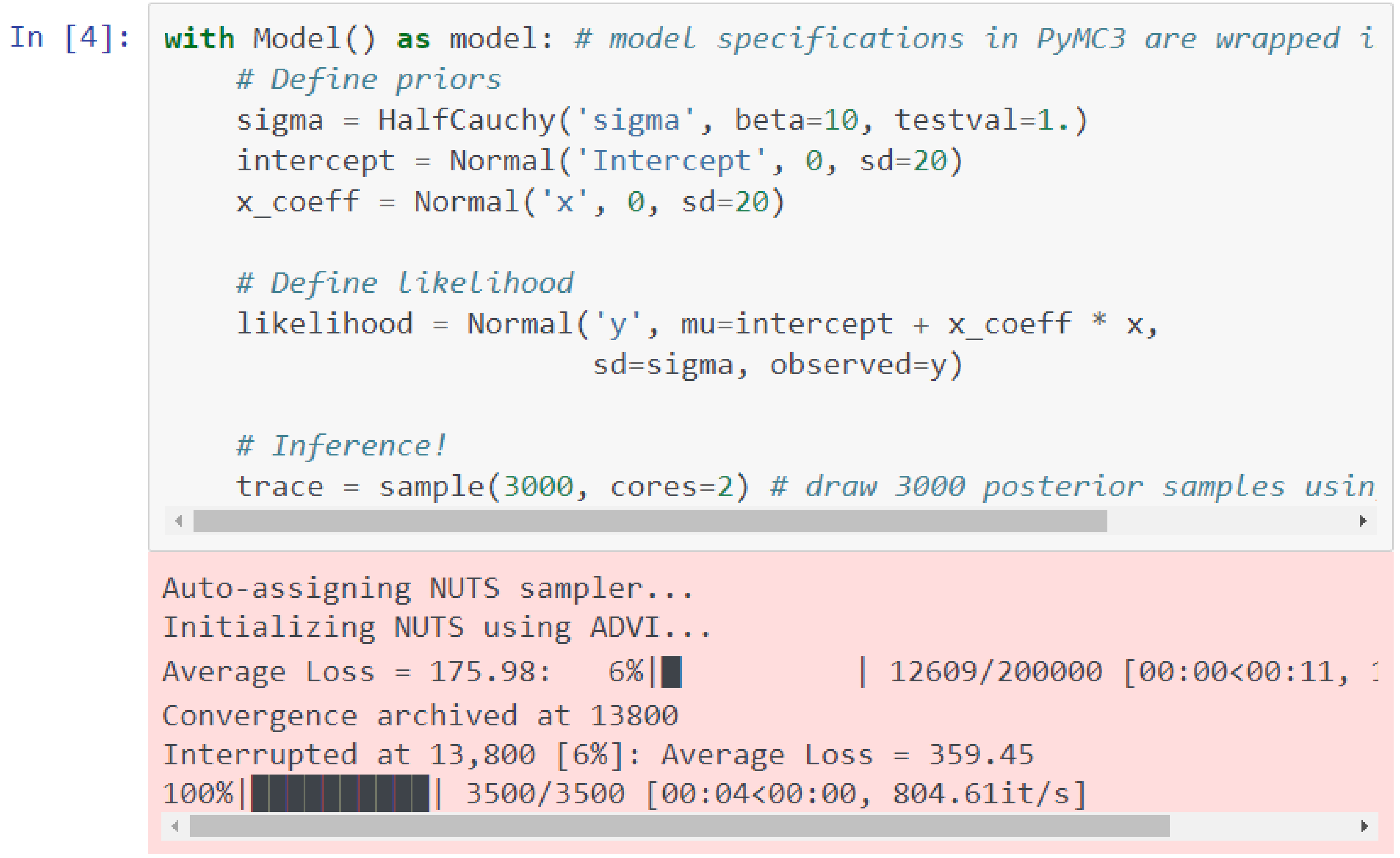
$$\mathbf{b}_o = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}, \mathbf{V}_o = \begin{pmatrix} s_0^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_n^2 \end{pmatrix}$$

- **Prior**:

$$p(\beta_0, \dots, \beta_n | y_1, \dots, y_m) \propto p(\beta_0, \dots, \beta_n) p(y_1, \dots, y_m | \beta_0, \dots, \beta_n)$$

## 1. Bayesian Linear Regression Using PyMC3

- Fit a Bayesian linear regression model: model specifications in PyMC3 are wrapped in a with statement.
- Use the awesome new NUTS sampler (our Inference Button) to draw 2000 posterior samples.

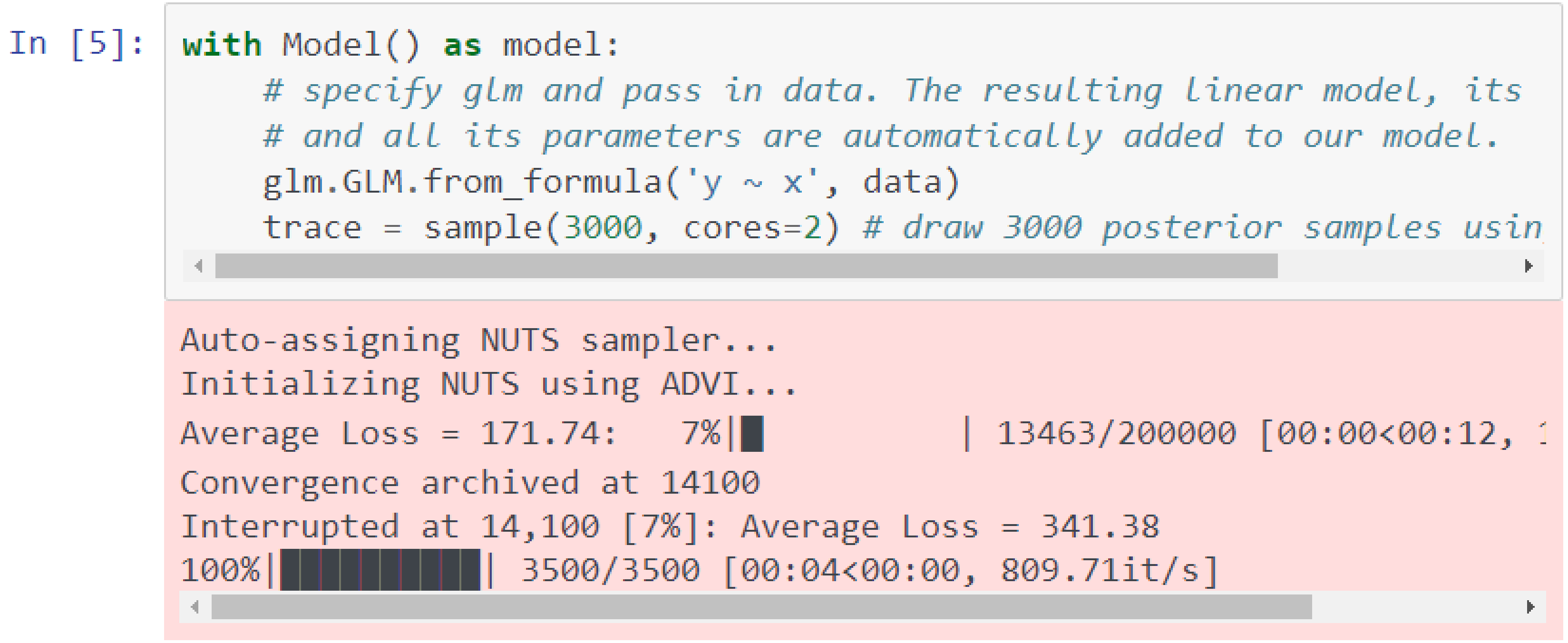


In [4]:
```
with Model() as model: # model specifications in PyMC3 are wrapped i
    # Define priors
    sigma = HalfCauchy('sigma', beta=10, testval=1.)
    intercept = Normal('Intercept', 0, sd=20)
    x_coeff = Normal('x', 0, sd=20)

    # Define likelihood
    likelihood = Normal('y', mu=intercept + x_coeff * x,
                        sd=sigma, observed=y)

    # Inference!
    trace = sample(3000, cores=2) # draw 3000 posterior samples usin
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 175.98:   6%|█         | 12609/200000 [00:00<00:11, 1
Convergence archived at 13800
Interrupted at 13,800 [6%]: Average Loss = 359.45
100%|██████████| 3500/3500 [00:04<00:00, 804.61it/s]
```

## 1. Bayesian Linear Regression Using PyMC3

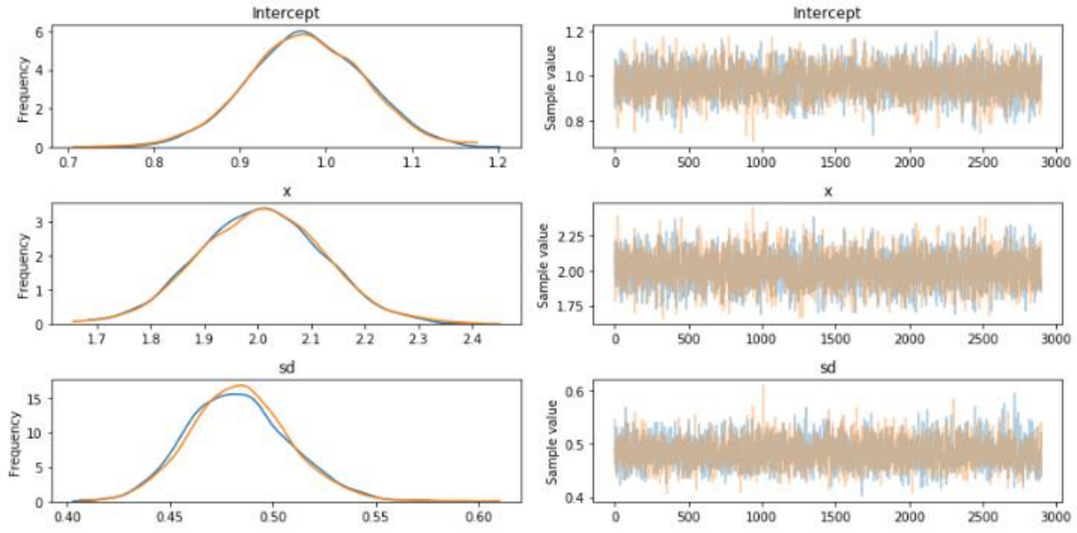
- The new **glm( )** function instead takes a Patsy linear model specifier from which it creates a design matrix. **glm( )** then adds random variables for each of the coefficients and an appropriate likelihood to the model.

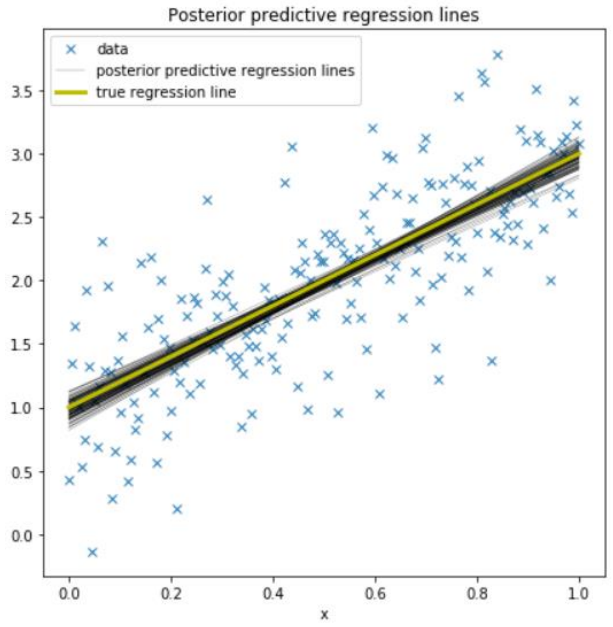


```
In [5]: with Model() as model:
            # specify glm and pass in data. The resulting linear model, its
            # and all its parameters are automatically added to our model.
            glm.GLM.from_formula('y ~ x', data)
            trace = sample(3000, cores=2) # draw 3000 posterior samples usin
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 171.74:   7%|█          | 13463/200000 [00:00<00:12, 1
Convergence archived at 14100
Interrupted at 14,100 [7%]: Average Loss = 341.38
100%|██████████| 3500/3500 [00:04<00:00, 809.71it/s]
```

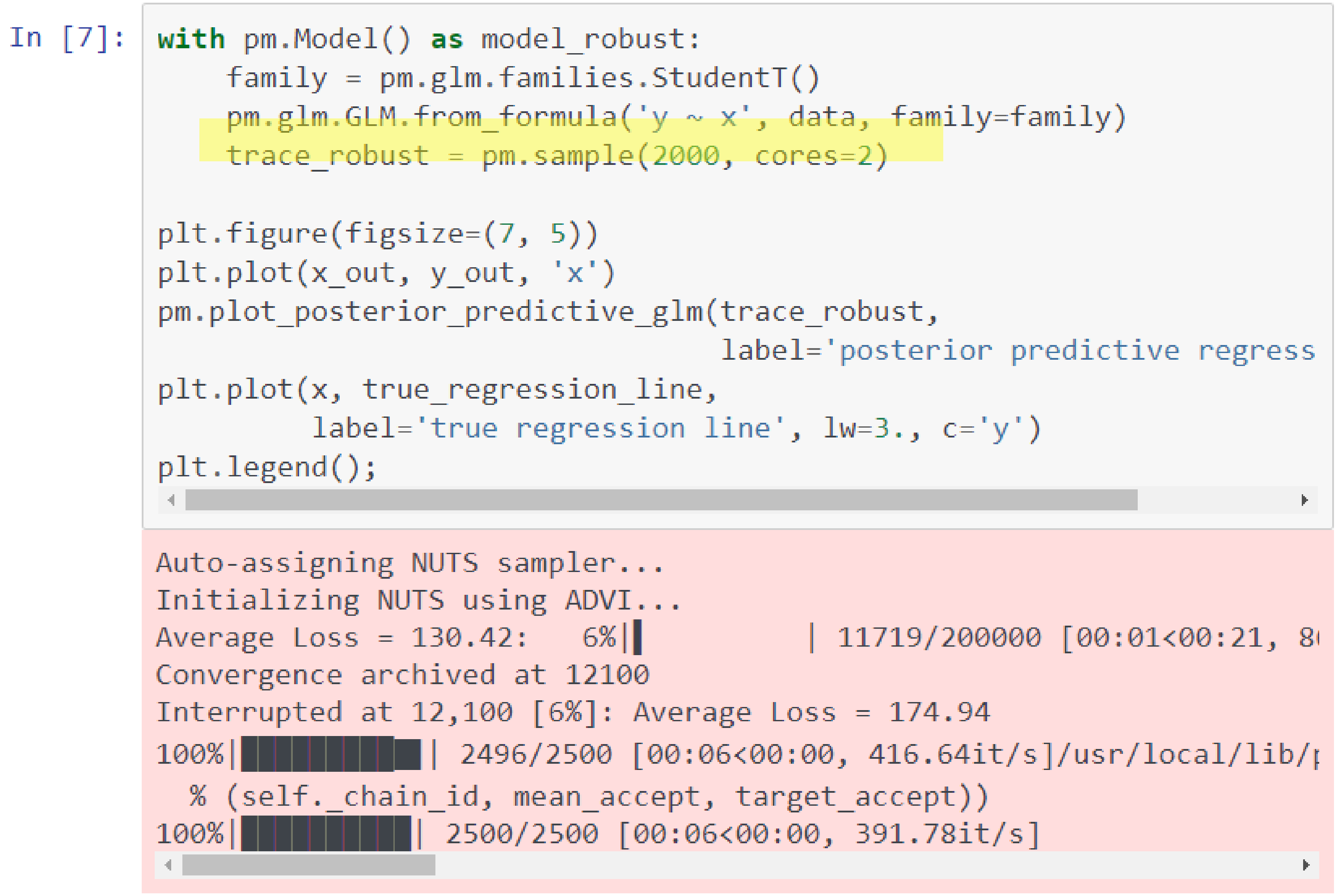
## 1. Bayesian Linear Regression Using PyMC3





## 1. Bayesian Linear Regression Using PyMC3

### Robust Linear Regression



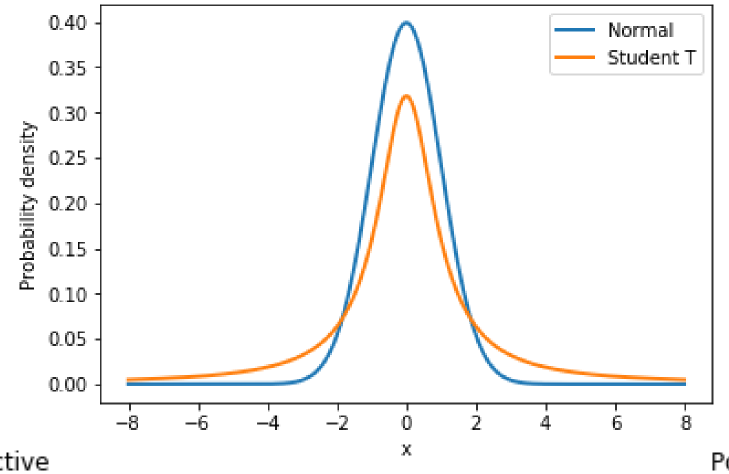
```
In [7]: with pm.Model() as model_robust:
            family = pm.glm.families.StudentT()
            pm.glm.GLM.from_formula('y ~ x', data, family=family)
            trace_robust = pm.sample(2000, cores=2)

        plt.figure(figsize=(7, 5))
        plt.plot(x_out, y_out, 'x')
        pm.plot_posterior_predictive_glm(trace_robust,
                                         label='posterior predictive regress
        plt.plot(x, true_regression_line,
                 label='true regression line', lw=3., c='y')
        plt.legend();
```

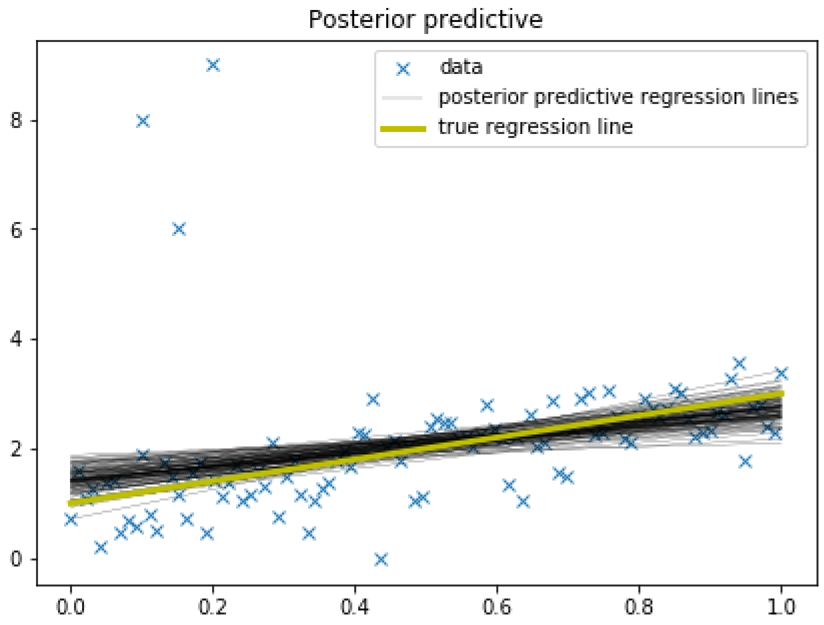
```
Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 130.42:   6%|▌         | 11719/200000 [00:01<00:21, 8
Convergence archived at 12100
Interrupted at 12,100 [6%]: Average Loss = 174.94
100%|█████████▉| 2496/2500 [00:06<00:00, 416.64it/s]/usr/local/lib/
  % (self._chain_id, mean_accept, target_accept))
100%|██████████| 2500/2500 [00:06<00:00, 391.78it/s]
```

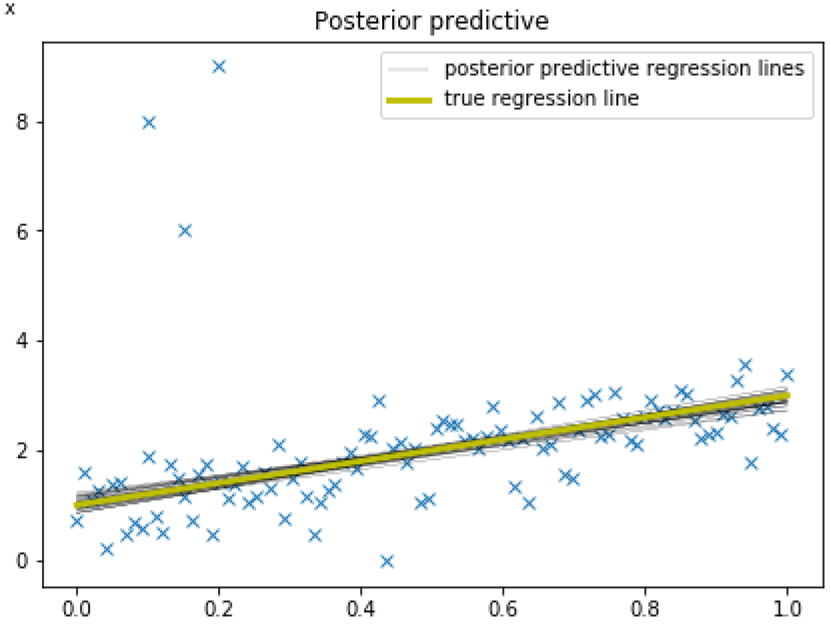
- PyMC3's **glm( )** function allows you to pass in a family object that contains information about the likelihood.

## 1. Bayesian Linear Regression Using PyMC3

### Robust Linear Regression







## 2. Bayesian Logistic Regression Using PyMC3

- The goal is to use demographic features, or variables, to predict whether an individual makes more than \$50,000 per year.

- age: continuous.
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- sex: Female, Male.
- capital-gain: continuous.
- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

## 2. Bayesian Logistic Regression Using PyMC3

- **Likelihood (observation model)**:

$$Y_i \sim \text{Bernulli}(\pi_i)$$

where

$$\text{logit}(\pi_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \cdots + \beta_n x_{in}$$

$$p(y_1, \dots, y_m | \beta_0, \dots, \beta_n) = \exp\left(\beta_0 \Sigma y_i + \Sigma \beta_j \Sigma x_{ij} y_i\right) \prod_{i=1}^{m} \left( \frac{1}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in})} \right)$$

$$Y_i = \begin{cases} 1 & \text{Income} > 50K \\ 0 & \text{otherwise} \end{cases}$$

$$\text{logit}(\pi_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4}$$

- $x_{i1} = (age)_i$
- $x_{i2} = (age)^2{}_i$
- $x_{i3} = (educ)_i$
- $x_{i4} = (hours)_i$

## 2. Bayesian Logistic Regression Using PyMC3

- **Likelihood (observation model)**:

$$Y_i \sim \text{Bernulli}(\pi_i)$$

where

$$\text{logit}(\pi_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \cdots + \beta_n x_{in}$$

$$p(y_1, \ldots, y_m | \beta_0, \ldots, \beta_n) = \exp\left(\beta_0 \Sigma y_i + \Sigma \beta_j \Sigma x_{ij} y_i\right) \prod_{i=1}^{m} \left( \frac{1}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in})} \right)$$

- **Prior**:

$$p(\beta_0, \ldots, \beta_n) = \mathrm{N}(\mathbf{b}_o, \mathbf{V}_o)$$

where

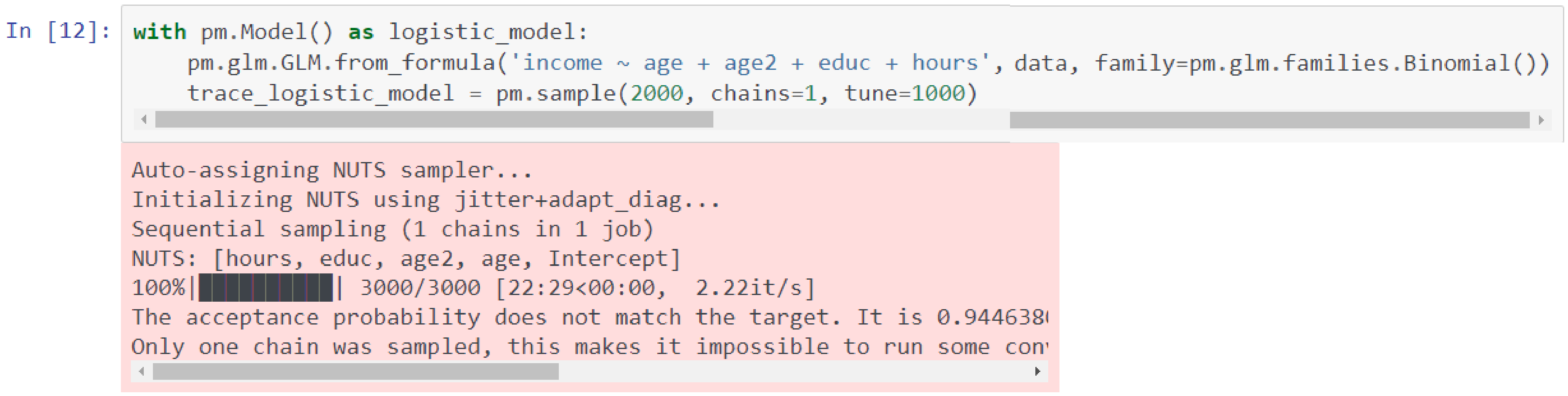
$$\mathbf{b}_o = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}, \mathbf{V}_o = \begin{pmatrix} s_0^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_n^2 \end{pmatrix}$$

- **Prior**:

$$p(\beta_0, \ldots, \beta_n | y_1, \ldots, y_m) \propto p(\beta_0, \ldots, \beta_n) p(y_1, \ldots, y_m | \beta_0, \ldots, \beta_n)$$

## 2. Bayesian Logistic Regression Using PyMC3

### Create Model and Samles



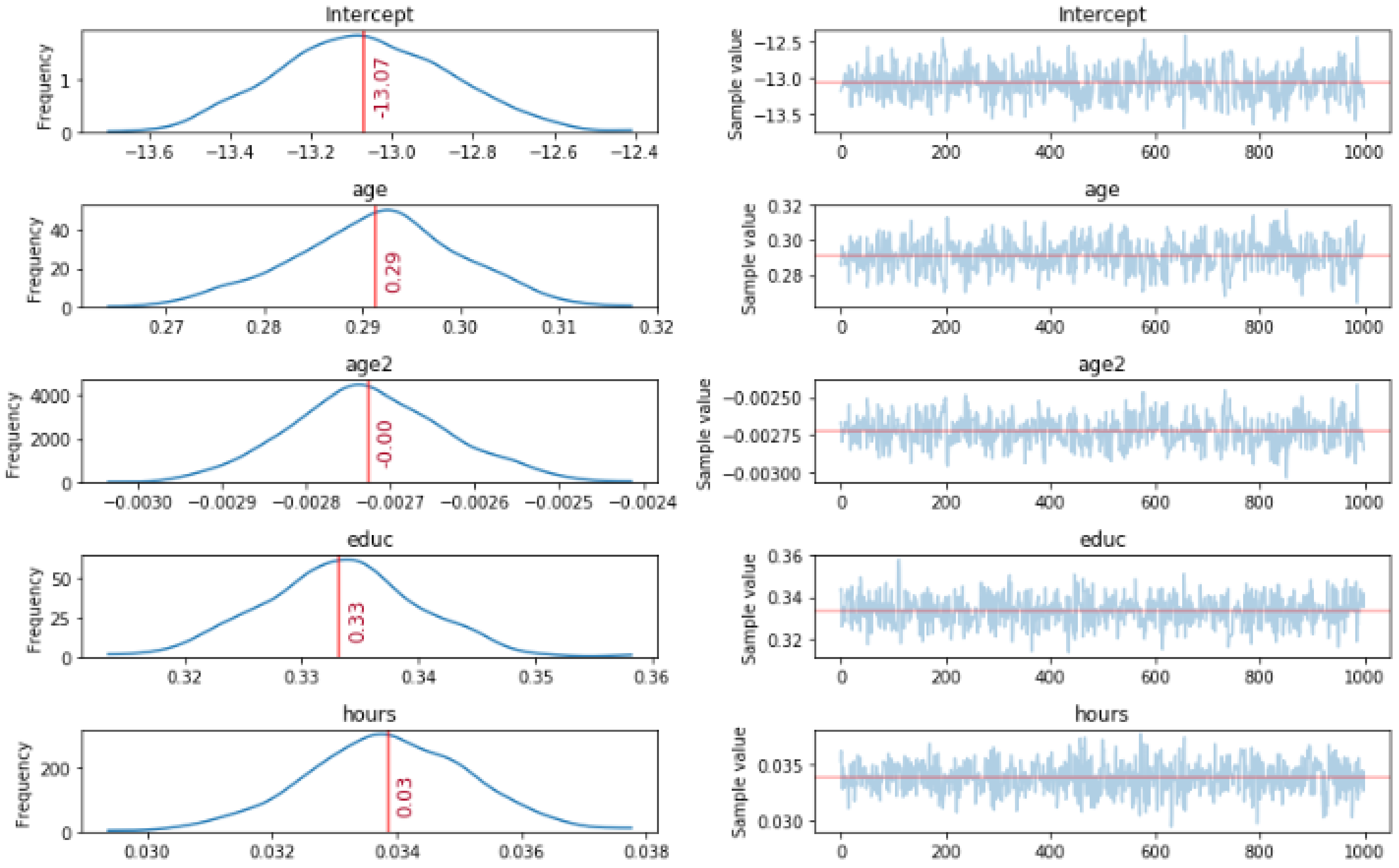
```
In [12]: with pm.Model() as logistic_model:
             pm.glm.GLM.from_formula('income ~ age + age2 + educ + hours', data, family=pm.glm.families.Binomial())
             trace_logistic_model = pm.sample(2000, chains=1, tune=1000)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (1 chains in 1 job)
NUTS: [hours, educ, age2, age, Intercept]
100%|██████████| 3000/3000 [22:29<00:00,  2.22it/s]
The acceptance probability does not match the target. It is 0.944638
Only one chain was sampled, this makes it impossible to run some con
```

## 2. Bayesian Logistic Regression Using PyMC3

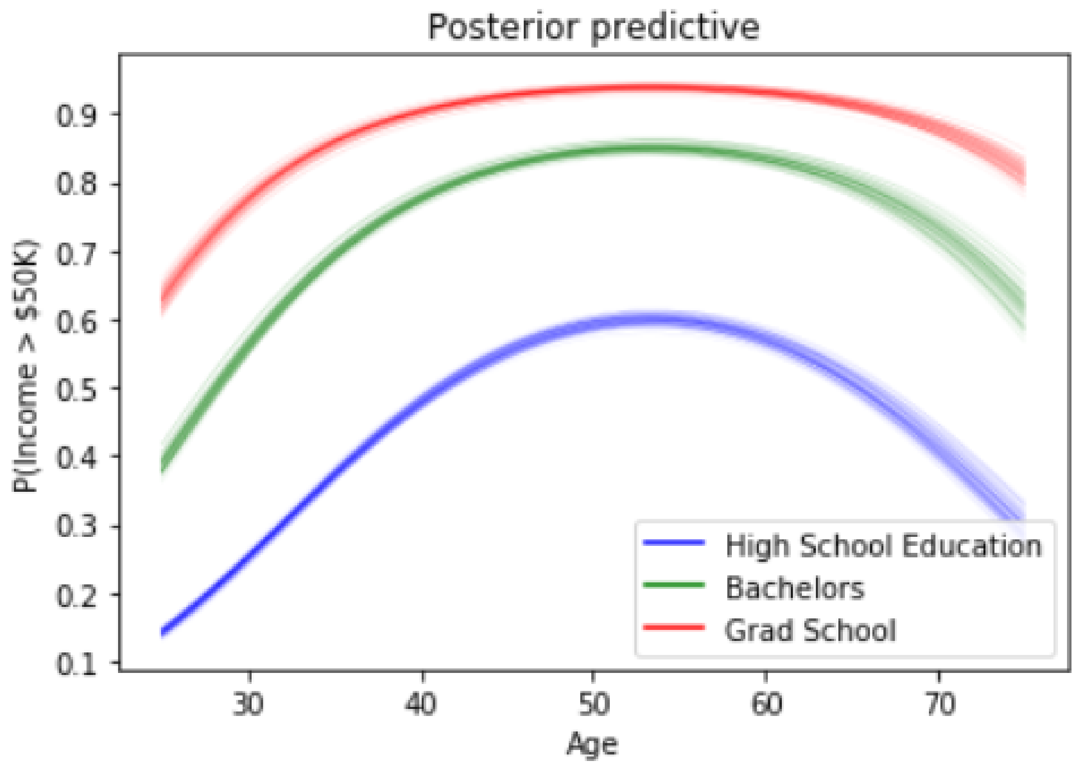
### Posterior Distribution on Parameters

```
In [13]: plot_traces(trace_logistic_model, retain=1000)
```



## 2. Bayesian Logistic Regression Using PyMC3

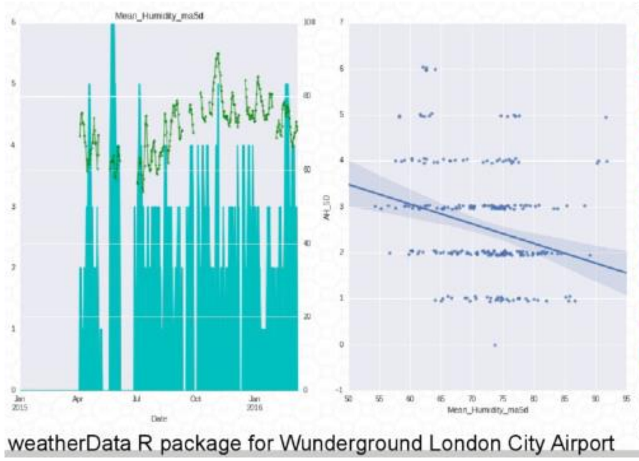
### Posterior Distribution on Parameters



## 3. Bayesian Poisson Regression Using PyMC3

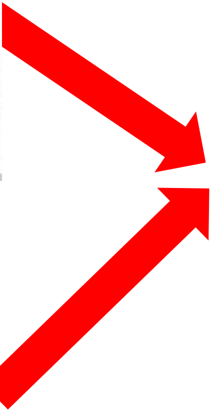
- Understanding the various effects of external environmental factors upon the allergic sneezing of a test subject.

**Environmental factors**

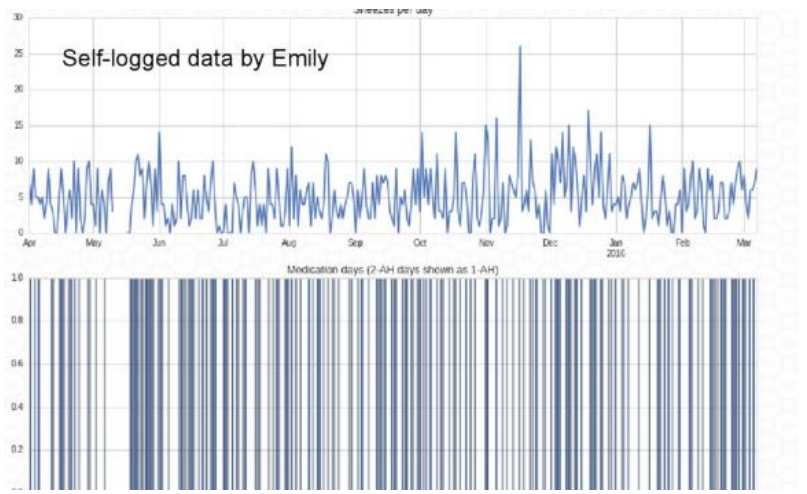


weatherData R package for Wunderground London City Airport

Annual NO2 pollution via LondonAir.org.uk



**# of observed Allergic symptoms**



**Drug consumptions?**

## 3. Bayesian Poisson Regression Using PyMC3

- Input features:

  - ✓ $y_1$: The subject sneezes N times per day, recorded as **nsneeze (int)**
  - ✓ $x_1$: The subject may or may not drink alcohol during that day, recorded as **alcohol (boolean)**
  - ✓ $x_2$: The subject may or may not take an antihistamine medication during that day, recorded as the negative action **nomeds (boolean)**
  - ✓ $x_3$: **alcohol (boolean)$\times$nomeds (boolean)**

- It is postulated (probably incorrectly) that sneezing occurs at some baseline rate, which increases if an antihistamine is not taken, and further increased after alcohol is consumed.

- The data is aggregated per day, to yield a total count of sneezes on that day, with

## 3. Bayesian Poisson Regression Using PyMC3

- **Likelihood (observation model)**:

$$Y_i \sim \text{Poisson}(\lambda_i)$$

where

$$\log(\lambda_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{ip}$$

$$p(y_1, \ldots, y_m | \beta_0, \beta_1, \beta_2) \propto \prod_{i=1}^{m} \lambda_i^{y_i} \exp(-\lambda_i)$$

$$\propto \exp\left(-\Sigma \exp\left(\Sigma x_{ij}\beta_j\right)\right) \exp\left(\Sigma y_i \Sigma x_{ij}\beta_j\right)$$

- **Prior:**

$$p(\beta_0, \beta_1, \beta_2) = \mathrm{N}(\mathbf{b}_o, \mathbf{V}_o)$$

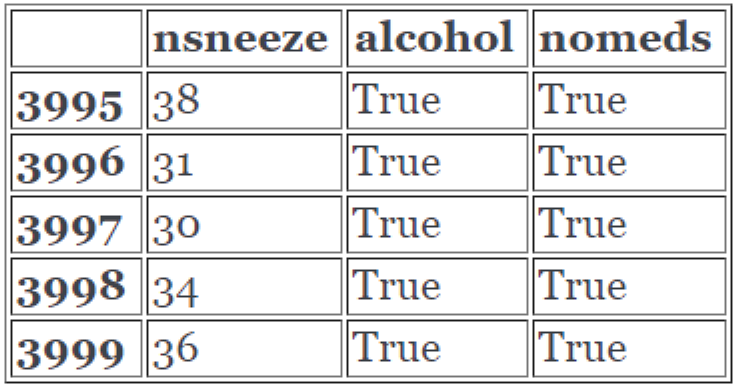
where

$$\mathbf{b}_o = \begin{pmatrix} b_0 \\ \vdots \\ b_2 \end{pmatrix}, \mathbf{V}_o = \begin{pmatrix} s_0^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_2^2 \end{pmatrix}$$

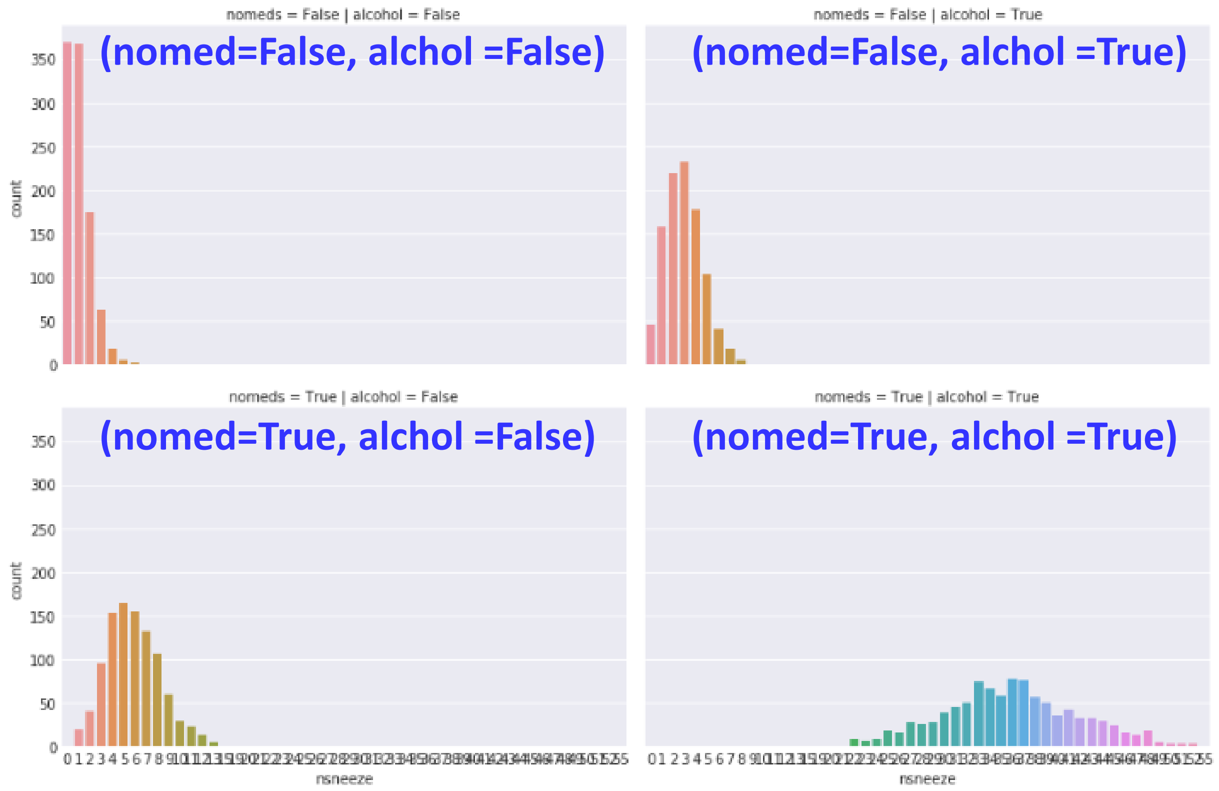
- **Prior:**

$$p(\beta_0, \beta_1, \beta_2 | y_1, \ldots, y_m) \propto p(\beta_0, \beta_1, \beta_2) p(y_1, \ldots, y_m | \beta_0, \beta_1, \beta_2)$$

## 3. Bayesian Poisson Regression Using PyMC3

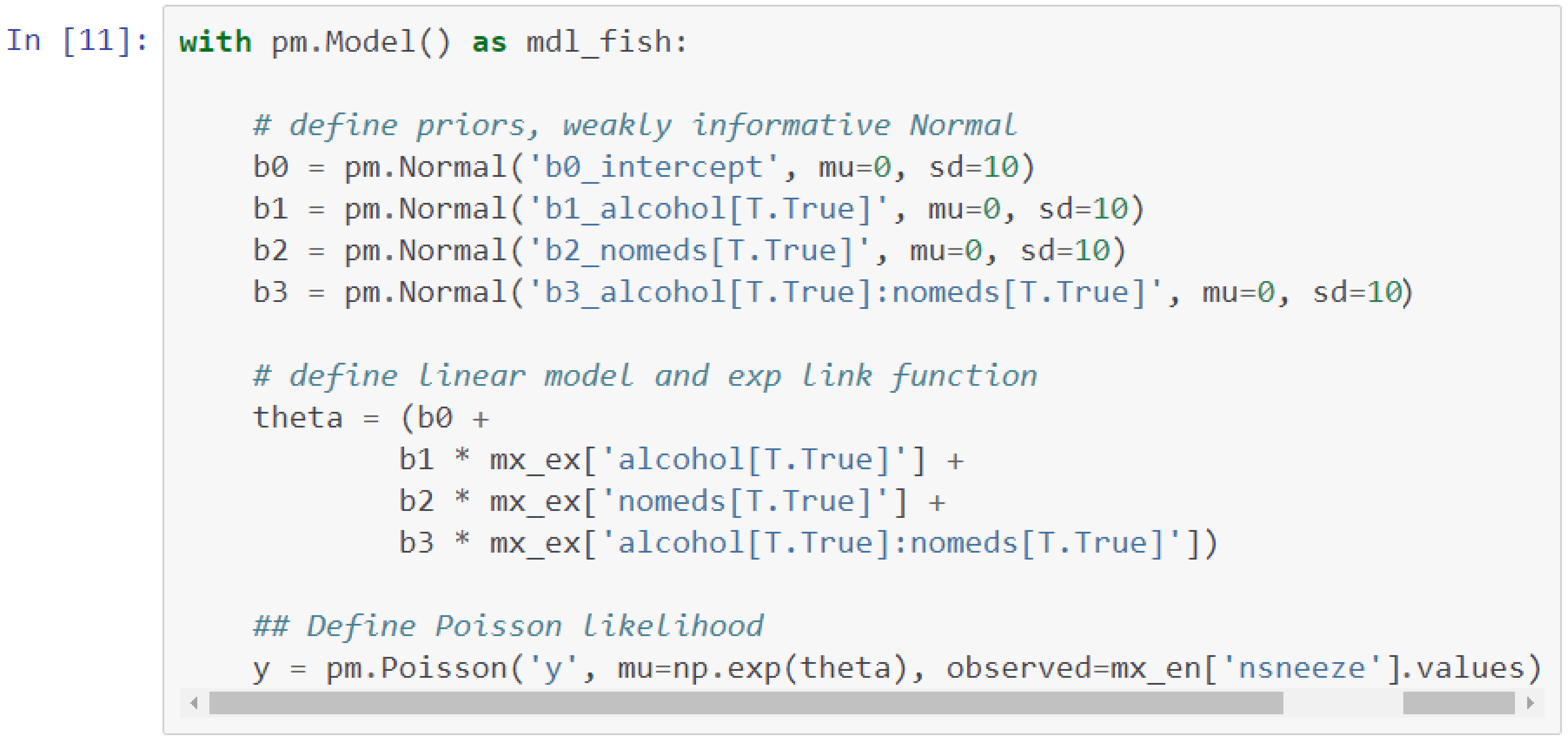


| | nsneeze | alcohol | nomeds |
|---|---|---|---|
| 3995 | 38 | True | True |
| 3996 | 31 | True | True |
| 3997 | 30 | True | True |
| 3998 | 34 | True | True |
| 3999 | 36 | True | True |



## 3. Bayesian Poisson Regression Using PyMC3

### Create Model



```
In [11]: with pm.Model() as mdl_fish:

             # define priors, weakly informative Normal
             b0 = pm.Normal('b0_intercept', mu=0, sd=10)
             b1 = pm.Normal('b1_alcohol[T.True]', mu=0, sd=10)
             b2 = pm.Normal('b2_nomeds[T.True]', mu=0, sd=10)
             b3 = pm.Normal('b3_alcohol[T.True]:nomeds[T.True]', mu=0, sd=10)

             # define linear model and exp link function
             theta = (b0 +
                     b1 * mx_ex['alcohol[T.True]'] +
                     b2 * mx_ex['nomeds[T.True]'] +
                     b3 * mx_ex['alcohol[T.True]:nomeds[T.True]'])

             ## Define Poisson likelihood
             y = pm.Poisson('y', mu=np.exp(theta), observed=mx_en['nsneeze'].values)
```

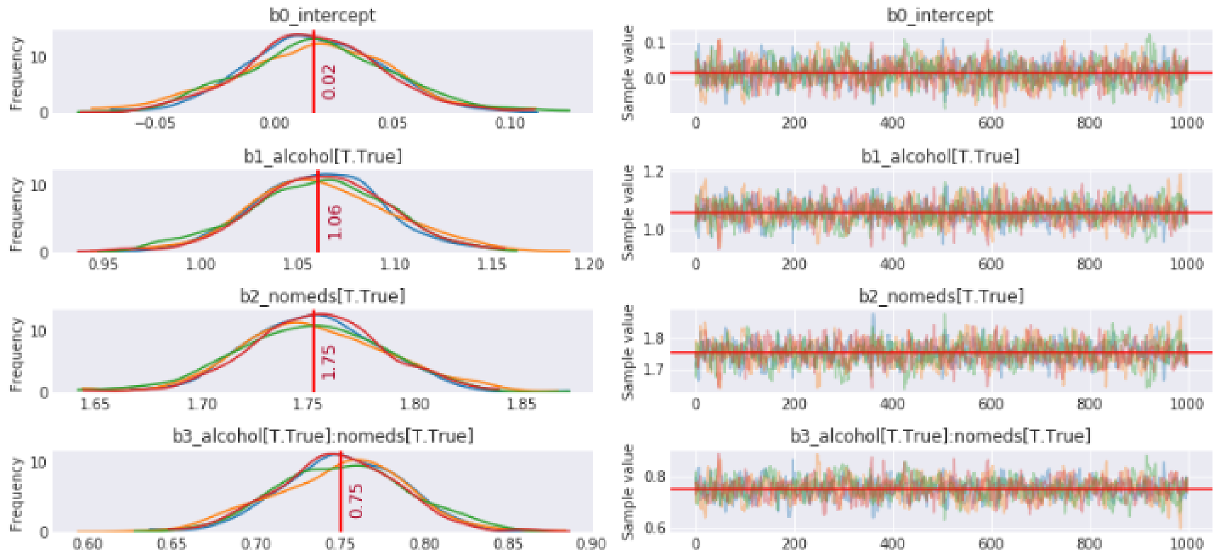
## 3. Bayesian Poisson Regression Using PyMC3

### Sample Model

```
In [12]: with mdl_fish:
             trc_fish = pm.sample(1000, tune=1000, cores=4)
```

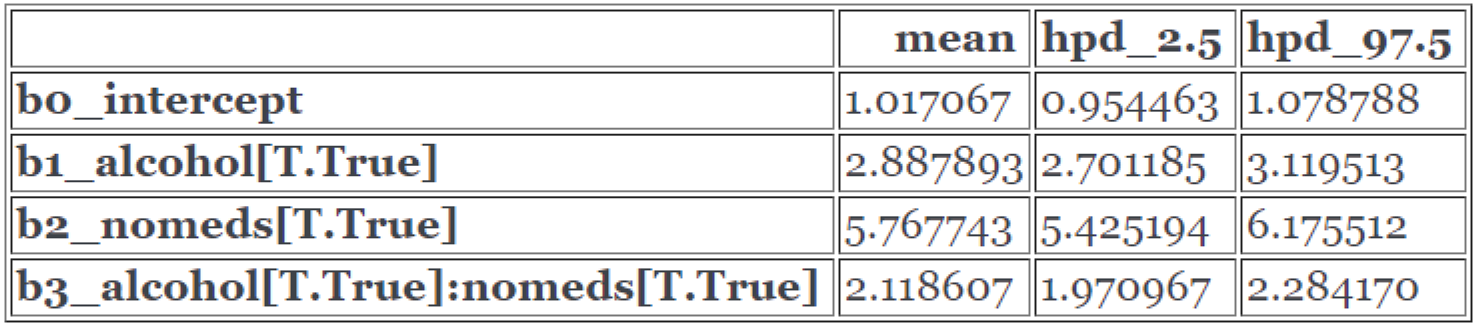
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [b3_alcohol[T.True]:nomeds[T.True], b2_nomeds[T.True], b1_alcol
Sampling 4 chains: 100%|██████████| 8000/8000 [01:25<00:00, 93.34dra
The number of effective samples is smaller than 25% for some paramet
```

### Visualize the Posterior Distribution of the Coefficients



## 3. Bayesian Poisson Regression Using PyMC3

### Results



| | mean | hpd_2.5 | hpd_97.5 |
|---|---|---|---|
| b0_intercept | 1.017067 | 0.954463 | 1.078788 |
| b1_alcohol[T.True] | 2.887893 | 2.701185 | 3.119513 |
| b2_nomeds[T.True] | 5.767743 | 5.425194 | 6.175512 |
| b3_alcohol[T.True]:nomeds[T.True] | 2.118607 | 1.970967 | 2.284170 |

## 4. Bayesian Survival Analysis Using PyMC3

- Survival analysis studies the distribution of the time to an event.
- Its applications span many fields across medicine, biology, engineering, and social science.
- This tutorial shows how to fit and analyze a Bayesian survival model in Python using PyMC3.

## 4. Bayesian Survival Analysis Using PyMC3

- **Likelihood (observation model)**:

$$T_i \sim \mathrm{Exp}(\lambda_i):\ \ p(t_i|\lambda_i) = \lambda e^{-\lambda t}$$

where

$$\lambda_i = \lambda_0 \exp\left(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \cdots + \beta_n x_{in}\right)$$

$$L(t_1, \dots, t_n, w_1, \dots, w_n|\beta_0, \dots, \beta_n) \propto e^{-t_i \Sigma e^{x_{ij}\beta_j}} \prod_{i=1}^{n} \left(t_i \Sigma e^{x_{ij}\beta_j}\right)^{w_i}$$

- **Prior**:

$$p(\beta_0, \dots, \beta_n) = \mathrm{N}(\mathbf{b}_o, \mathbf{V}_o)$$

where

$$\mathbf{b}_o = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}, \mathbf{V}_o = \begin{pmatrix} s_0^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_n^2 \end{pmatrix}$$

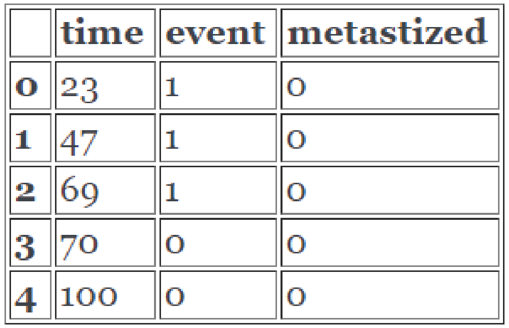
- **Prior**:

$$p(\beta_0, \dots, \beta_n|y_1, \dots, y_m) \propto p(\beta_0, \dots, \beta_n) p(y_1, \dots, y_m|\beta_0, \dots, \beta_n)$$

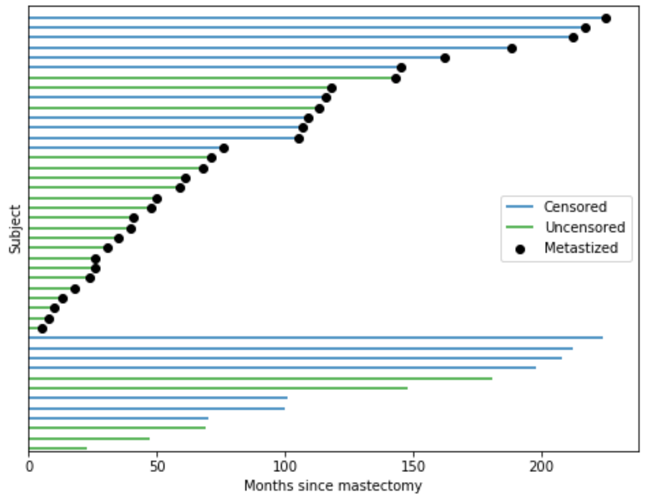
## 4. Bayesian Survival Analysis Using PyMC3

### Data

- Description
  - ✓ Survival times in months after chemotherapy (항암치료) of patient with a cancer. The cancers are classified as having metastized (전이) or not based on a medical test
- Format
  - ✓ A data frame with 42 observations on the following 3 variables.
    - ✓ Target response $Y$: survival times in months.
    - ✓ $W$: logical indicating if the event was observed (TRUE) or if the survival time was censored (FALSE).
    - ✓ Input Features $X$: metastized or not (전이되었는지 아닌지) (Boolean)



| | time | event | metastized |
|---|---|---|---|
| 0 | 23 | 1 | 0 |
| 1 | 47 | 1 | 0 |
| 2 | 69 | 1 | 0 |
| 3 | 70 | 0 | 0 |
| 4 | 100 | 0 | 0 |



(Data Source : B. S. Everitt and S. Rabe-Hesketh (2001), Analysing Medical Data using S-PLUS, Springer, New York, USA.)

## 4. Bayesian Survival Analysis Using PyMC3

### Model Description

- The most commonly used risk regression model is Cox's Proportional Hazard model

$$\lambda(t) = \lambda_0(t)\exp(\boldsymbol{x}\beta)$$

  - ✓ $\lambda_0(t)$ is the baseline hazard, which is independent of the covariate
  - ✓ ***Pricewise constant proportional hazard model*** can be used for the $i$th subject and $j$th interval as

$$\lambda_{ij} = \lambda_j \exp(\boldsymbol{x_i}\beta)$$

- In order to perform Bayesian inference with the Cox model, we must specify priors on $\beta$ and $\lambda_0(t)$:
  - ✓ We place a normal prior on $\beta$,

$$\beta \sim N\left(\mu_\beta, \sigma_\beta^2\right)$$

  - ✓ We place a normal hipper-prior on $\mu_\beta$ and $\sigma_\beta^2$

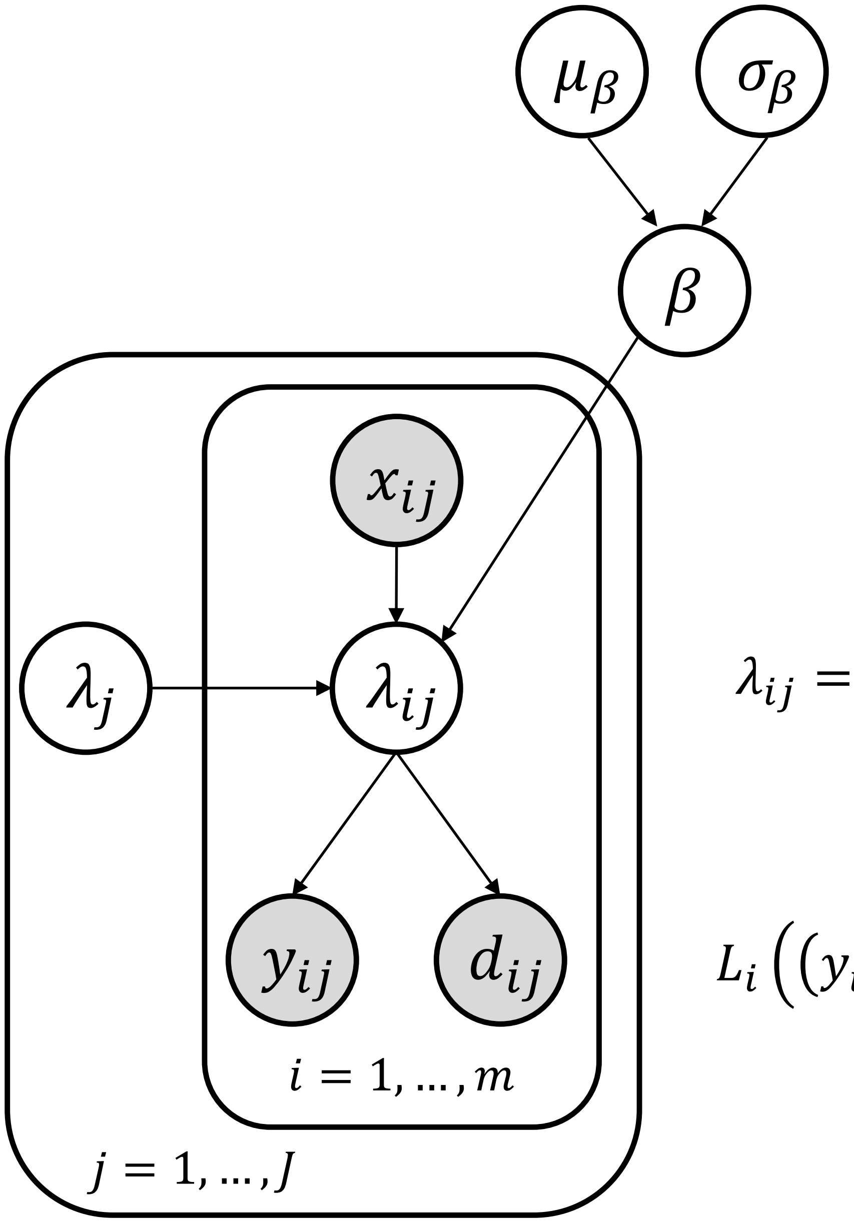
$$\mu_\beta \sim N(0, 10^2), \qquad \sigma_\beta^2 \sim N(0, 10)$$

  - ✓ We choose a semiparametric prior **(piecewise constant hazard model)**,

$$\lambda_0(t) = \lambda_j \text{ if } s_j \le t < s_{j+1}, \qquad \lambda_j \sim \text{Gamma}(10^{-2}, 10^{-2})$$

## 4. Bayesian Survival Analysis Using PyMC3

### Model Description



$$\mu_\beta \sim N(0, 10^2), \qquad \sigma_\beta^2 \sim N(0, 10)$$

$$\beta \sim N\left(\mu_\beta, \sigma_\beta^2\right)$$

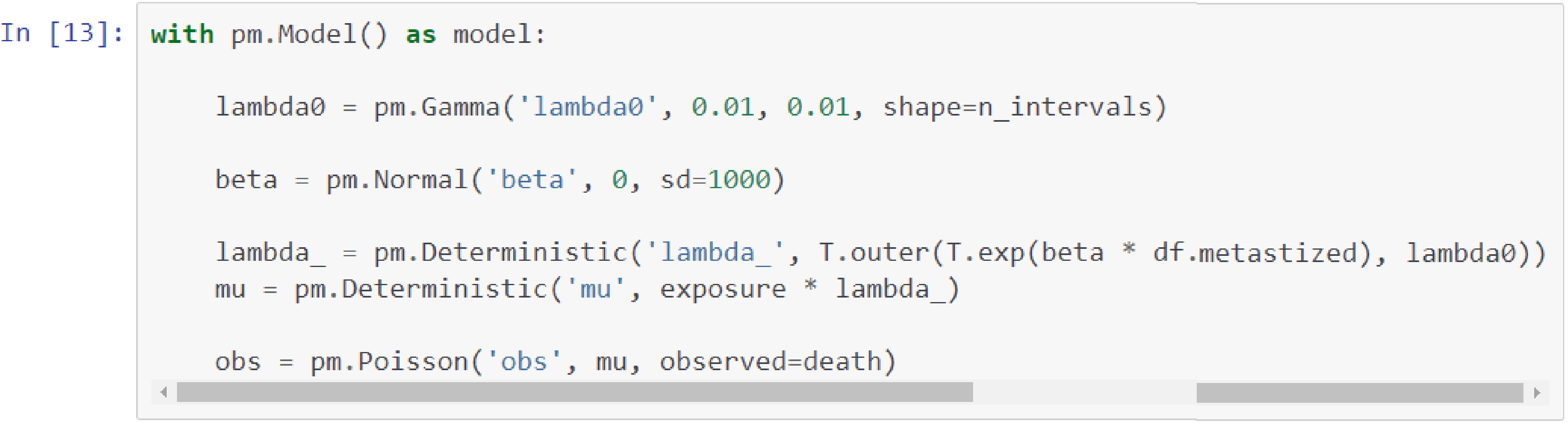
$$\lambda_{ij} = \lambda_j \exp(\beta x_i)$$

$$L_i\left(\left(y_{ij}, d_{ij}\right) | \lambda_{ij}\right) \propto \left(\lambda_{ij} e^{-\lambda_{ij} y_{ij}}\right)^{d_{ij}} \left(e^{-\lambda_{ij} y_{ij}}\right)^{1-d_{ij}}$$

## 4. Bayesian Survival Analysis Using PyMC3

### Create Model

- We may approximate $d_{ij}$ with a Possion random variable with mean $y_{ij}\lambda_{ij}$. This approximation leads to the following pymc3 model.



```
In [13]: with pm.Model() as model:

             lambda0 = pm.Gamma('lambda0', 0.01, 0.01, shape=n_intervals)

             beta = pm.Normal('beta', 0, sd=1000)

             lambda_ = pm.Deterministic('lambda_', T.outer(T.exp(beta * df.metastized), lambda0))
             mu = pm.Deterministic('mu', exposure * lambda_)

             obs = pm.Poisson('obs', mu, observed=death)
```

## 4. Bayesian Survival Analysis Using PyMC3

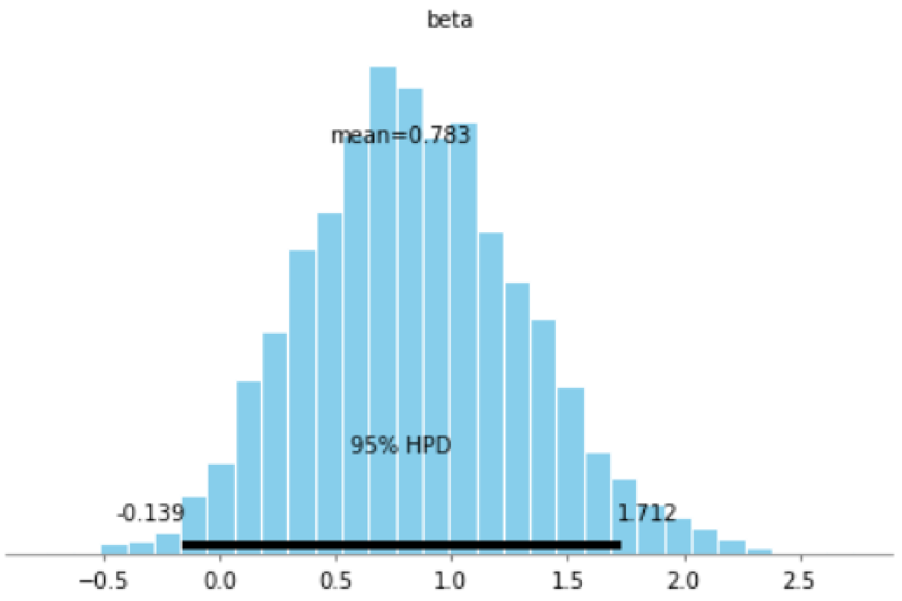
### Sampling

```
In [14]: n_samples = 1000
         n_tune = 1000
```

```
In [15]: with model:
             trace = pm.sample(n_samples, tune=n_tune, random_seed=SEED)

         100%|██████████| 2000/2000 [15:44<00:00,  2.31it/s]
```

```
In [16]: np.exp(trace['beta'].mean())

Out[16]: 2.1879479618944364
```

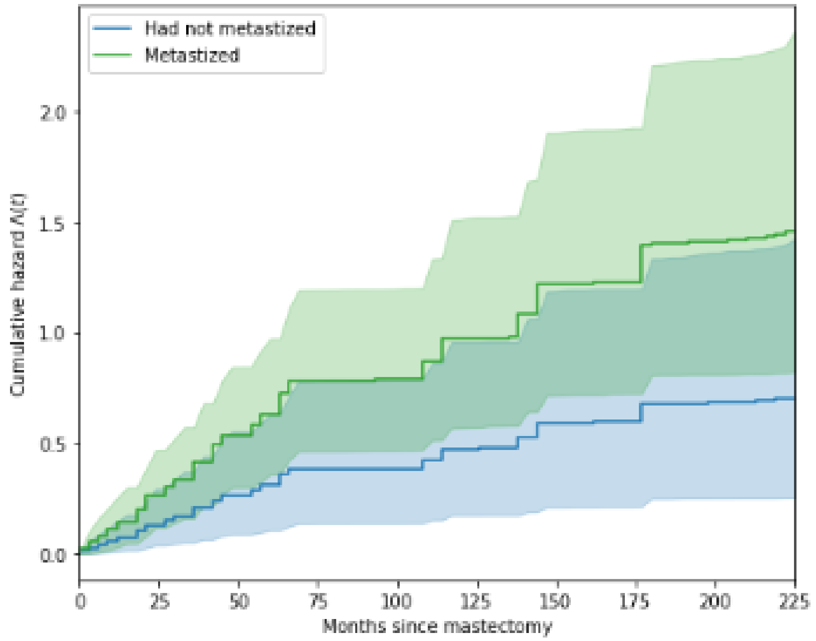
```
In [17]: pm.plot_posterior(trace, varnames=['beta'], color='#87ceeb');
```

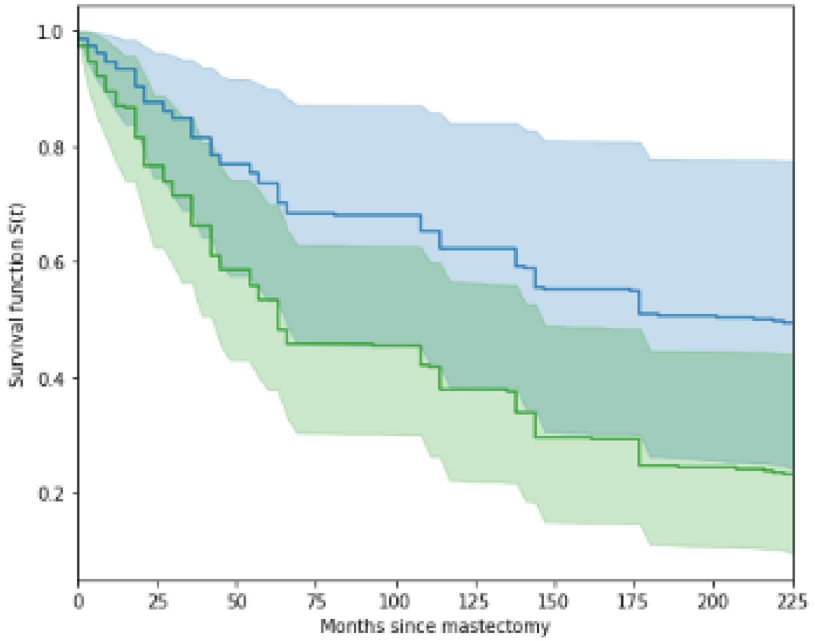


## 4. Bayesian Survival Analysis Using PyMC3

### Interpretations

- We now examine the effect of 항암치료 on both the cumulative hazard and on the survival function.





the cumulative hazard function

$$\Lambda(t) = \int_0^t \lambda(t)dt$$

Survival function

$$S(t) = P(T > t) = 1 - F(t) = e^{-\lambda t}$$

- The cumulative hazard for metastized subjects increases more rapidly initially, after which it increases roughly in parallel with the baseline cumulative hazard.