Probabilistic Programming

Marius Popescu

popescunmarius@gmail.com

2019 - 2020

Bayesian Regression with PyMC

Logistic | Linear | Ridge | LASSO

The Bayesian Perspective

Unlike in the ordinary regression case, we don't get a single regression line - we get a probability distribution on the space of all such lines. The width of this posterior represents the uncertainty in our estimate.

Why to use it?

Although computationally it's a LOT harder than ordinary least squares, one can easily formulate and solve a very flexible model that addresses most of the problems with ordinary least squares: extreme sensitivity to outliers, inability to incorporate priors, and little ability to quantify uncertainty.

When to use it?

- The distribution of errors is not normal or close to it
- Don’t have enough data

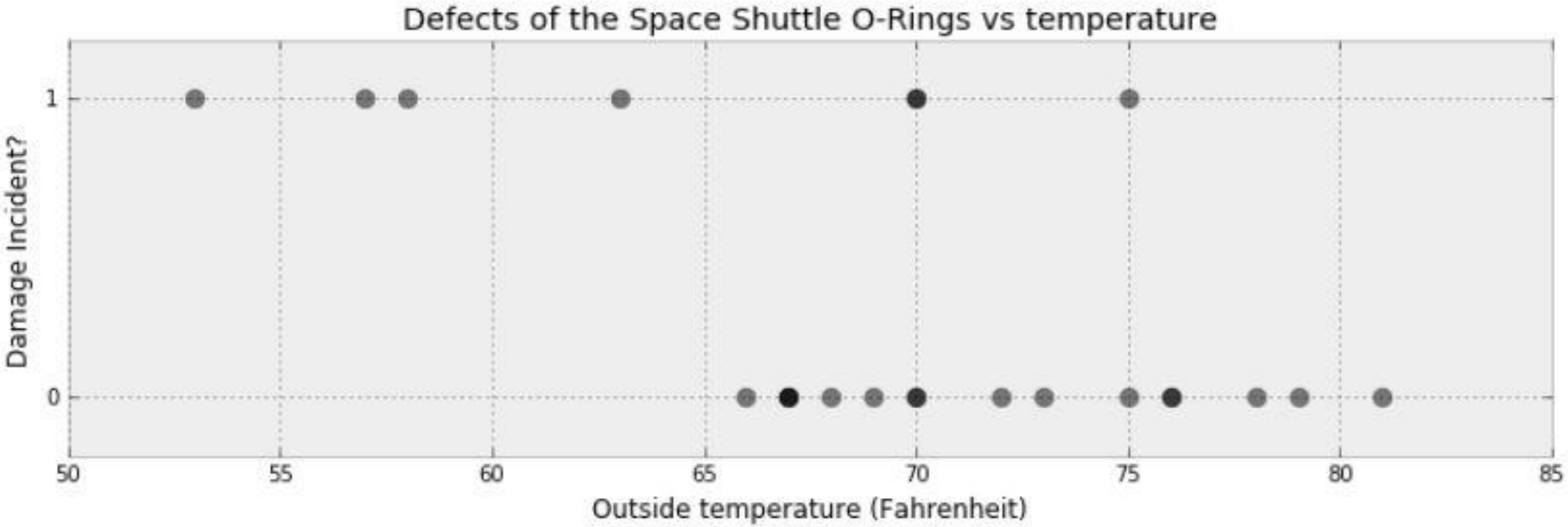
Bayesian Logistic Regression

Challenger Space Shuttle Disaster

Space Shuttle Challenger disaster
28 January 1986
Francis R. Scobee, Commander
Michael J. Smith, Pilot
Ronald McNair, Mission Specialist
Ellison Onizuka, Mission Specialist
Judith Resnik, Mission Specialist
Greg Jarvis, Payload Specialist
Christa McAuliffe, Payload Specialist
BUTROUS FOUNDATION

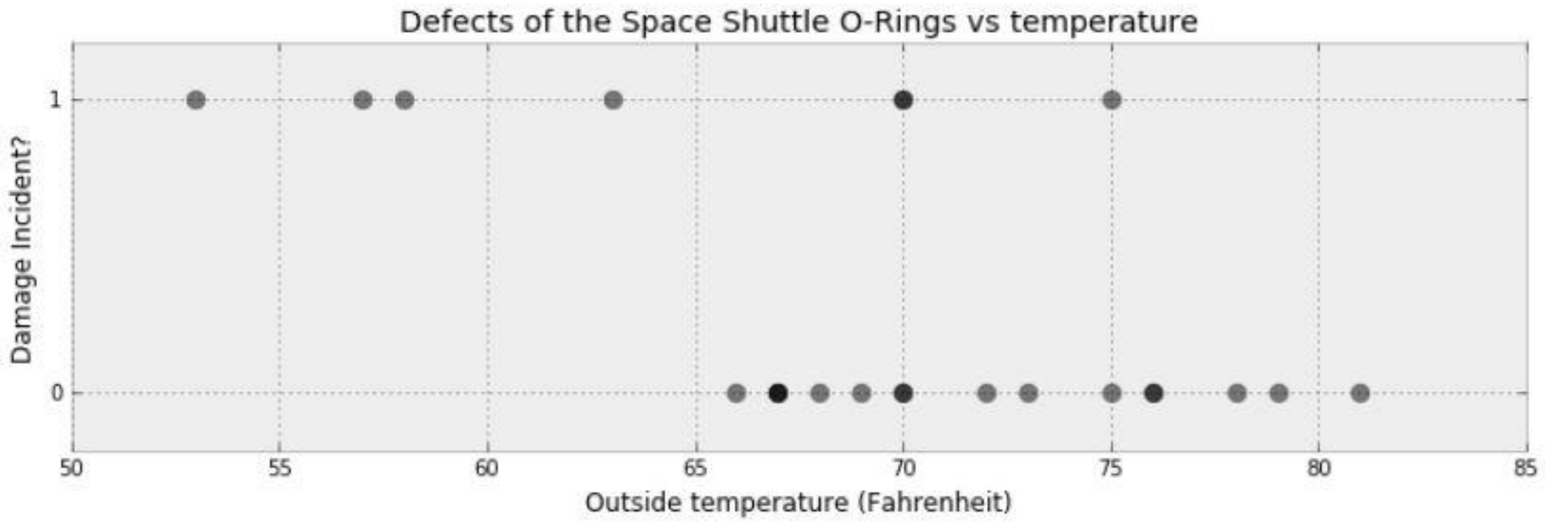
On January 28, 1986, the twenty-fifth flight of the U.S. space shuttle program ended in disaster when one of the rocket boosters of the Shuttle Challenger exploded shortly after lift-off, killing all seven crew members. The presidential commission on the accident concluded that it was caused by the failure of an O-ring in a field joint on the rocket booster, and that this failure was due to a faulty design that made the O-ring unacceptably sensitive to a number of factors including outside temperature. Of the previous 24 flights, data were available on failures of O-rings on 23, (one was lost at sea), and these data were discussed on the evening preceding the Challenger launch, but unfortunately only the data corresponding to the 7 flights on which there was a damage incident were considered important and these were thought to show no obvious trend.

The Data


Defects of the Space Shuttle O-Rings vs temperature
Damage Incident?
1
0
50
55
60
65
70
75
80
85
Outside temperature (Fahrenheit)

The Problem

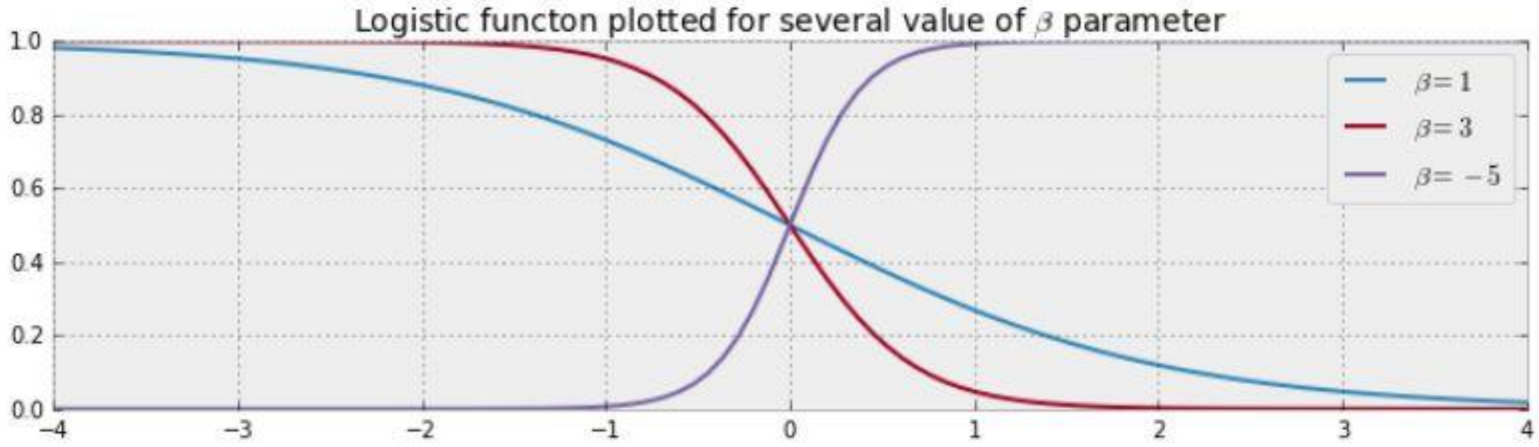
It looks clear that the probability of damage incidents occurring increases as the outside temperature decreases. We are interested in modeling the probability here because it does not look like there is a strict cutoff point between temperature and a damage incident occurring. The best we can do is ask "At temperature t, what is the probability of a damage incident?".



Modeling

We need a function of temperature, call it $p(t)$, that is bounded between 0 and 1 (so as to model a probability) and changes from 1 to 0 as we increase temperature. There are actually many such functions, but the most popular choice is the logistic function:

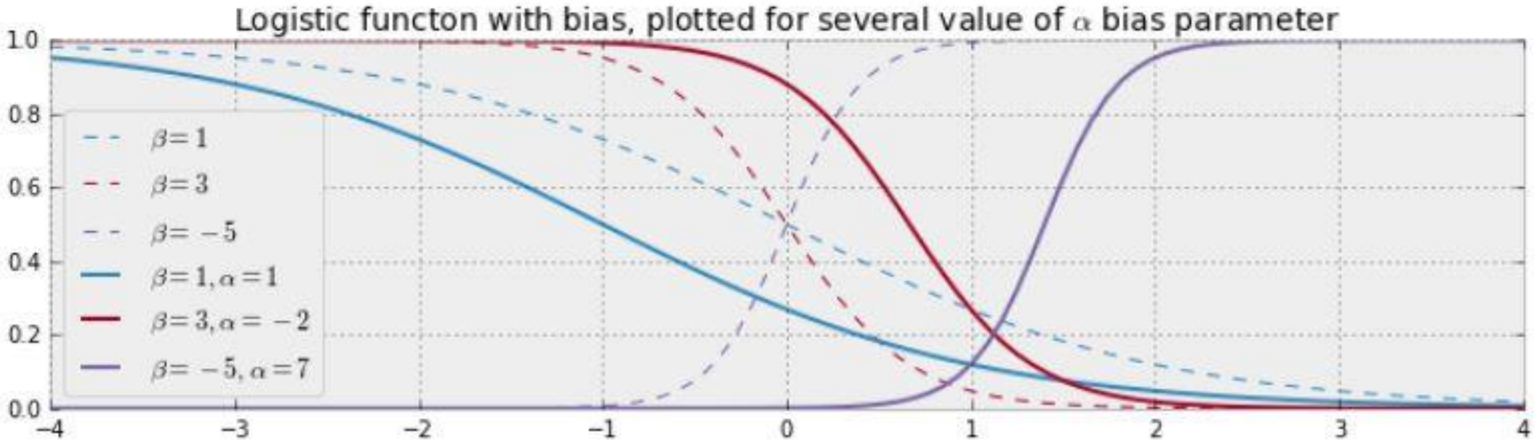
$$p(t) = \frac{1}{1 + e^{\beta t}}$$



Modeling

But something is missing. In the plot of the logistic function, the probability changes only near zero, but in our data above the probability changes around 65 to 70. We need to add a bias term to our logistic function:

$$p(t) = \frac{1}{1 + e^{\beta t + \alpha}}$$



Modeling

α and β are unknown parameters.

The α and β parameters have no reason to be positive, bounded or relatively large, so they are best modeled by a Normal random variable

```
challenger_data = np.genfromtxt("challenger_data.csv", skip_header=1,
                                usecols=[1, 2], missing_values="NA",
                                delimiter=",")
# drop the NA values
challenger_data = challenger_data[~np.isnan(challenger_data[:, 1])]

temperature = challenger_data[:, 0]
D = challenger_data[:, 1]  # defect or not?

def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

# notice the`value` here. We explain why below.
beta = pm.Normal("beta", 0, 0.001, value=0)
alpha = pm.Normal("alpha", 0, 0.001, value=0)
```

Modeling

We have our probabilities, but how do we connect them to our observed data?

$$D_i \sim Ber\big(p(t_i)\big), i = 1..N$$

```
@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta * t + alpha))

# connect the probabilities in `p` with our observations through a
# Bernoulli random variable.
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)

model = pm.Model([observed, beta, alpha])

# Mysterious code to be explained in Chapter 3
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)
```

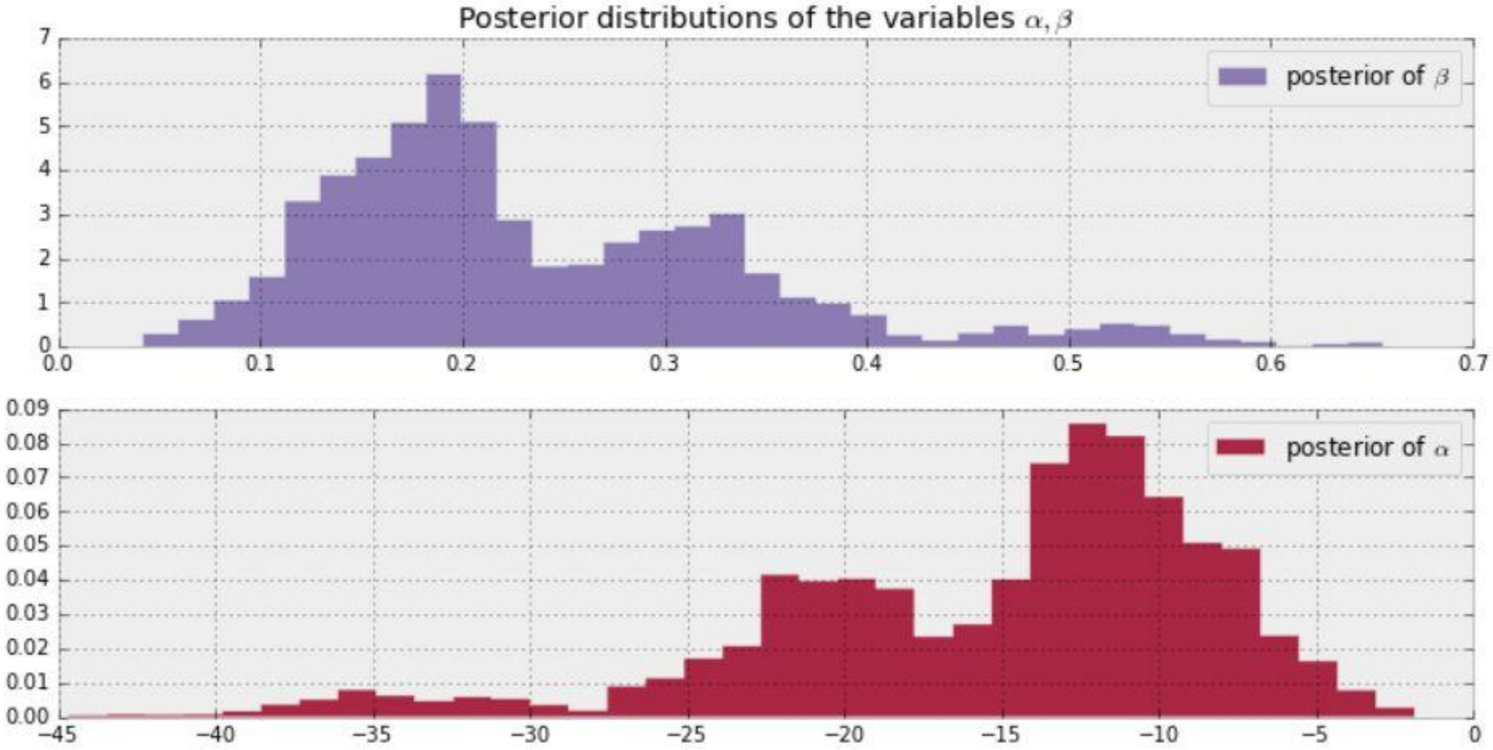
Notice in the above code we had to set the values of beta and alpha to 0. The reason for this is that if beta and alpha are very large, they make p equal to 1 or 0. Unfortunately, `pm.Bernoulli` does not like probabilities of exactly 0 or 1, though they are mathematically well-defined probabilities. So by setting the coefficient values to 0, we set the variable p to be a reasonable starting value. This has no effect on our results, nor does it mean we are including any additional information in our prior. It is simply a computational caveat in PyMC.

The Results

All samples of β are greater than 0. If instead the posterior was centered around 0, we may suspect that $\beta = 0$, implying that temperature has no effect on the probability of defect.

Similarly, all α posterior values are negative and far away from 0, implying that it is correct to believe that α is significantly less than 0.

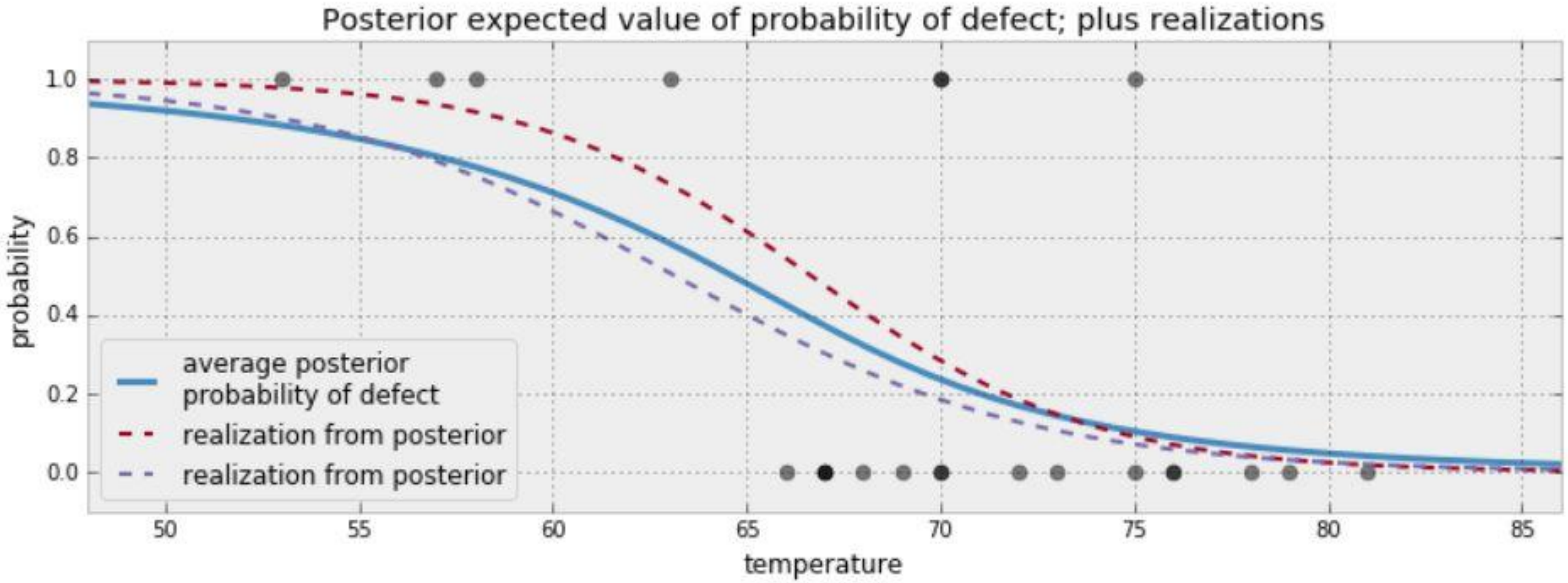
Regarding the spread of the data, we are very uncertain about what the true parameters might be (though considering the low sample size and the large overlap of defects-to-nondefects this behaviour is perhaps expected).



The Results

The expected probability for a specific value of the temperature. That is, we average over all samples from the posterior to get a likely value $p(t_i)$.

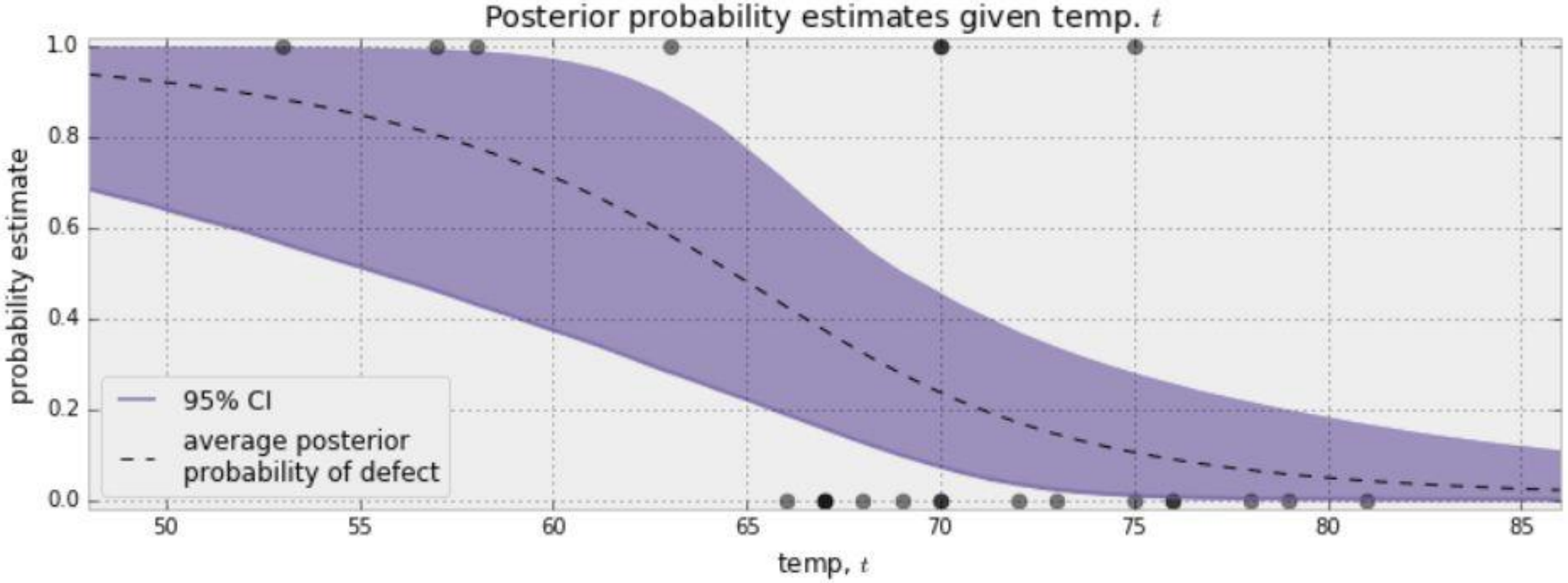
We also plotted two possible realizations of what the actual underlying system might be. Both are equally likely as any other draw. The blue line is what occurs when we average all the 20000 possible dotted lines together.



The Results

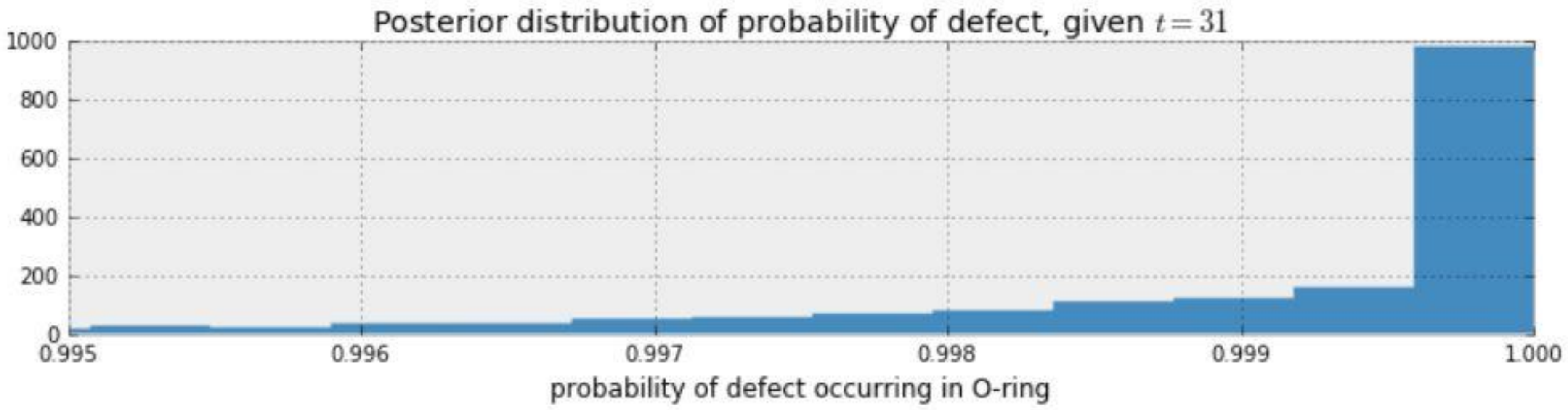
An interesting question to ask is for what temperatures are we most uncertain about the defect-probability? Below we plot the expected value line and the associated 95% intervals for each temperature.

The 95% credible interval, or 95% CI, painted in purple, represents the interval, for each temperature, that contains 95% of the distribution. For example, at 65 degrees, we can be 95% sure that the probability of defect lies between 0.25 and 0.75.



What about the day of the Challenger disaster?

```
prob_31 = logistic(31, beta_samples, alpha_samples)


plt.xlim(0.995, 1)
plt.hist(prob_31, bins=1000, normed=True, histtype='stepfilled')
plt.title("Posterior distribution of probability of defect, given $t= 31$")
plt.xlabel("probability of defect occurring in O-ring")
plt.show()
```



Bayesian Linear Regression

Simple Linear Regression

Let's assume we have a one-dimensional dataset:

$$(x_1, y_1), \dots, (x_n, y_n)$$

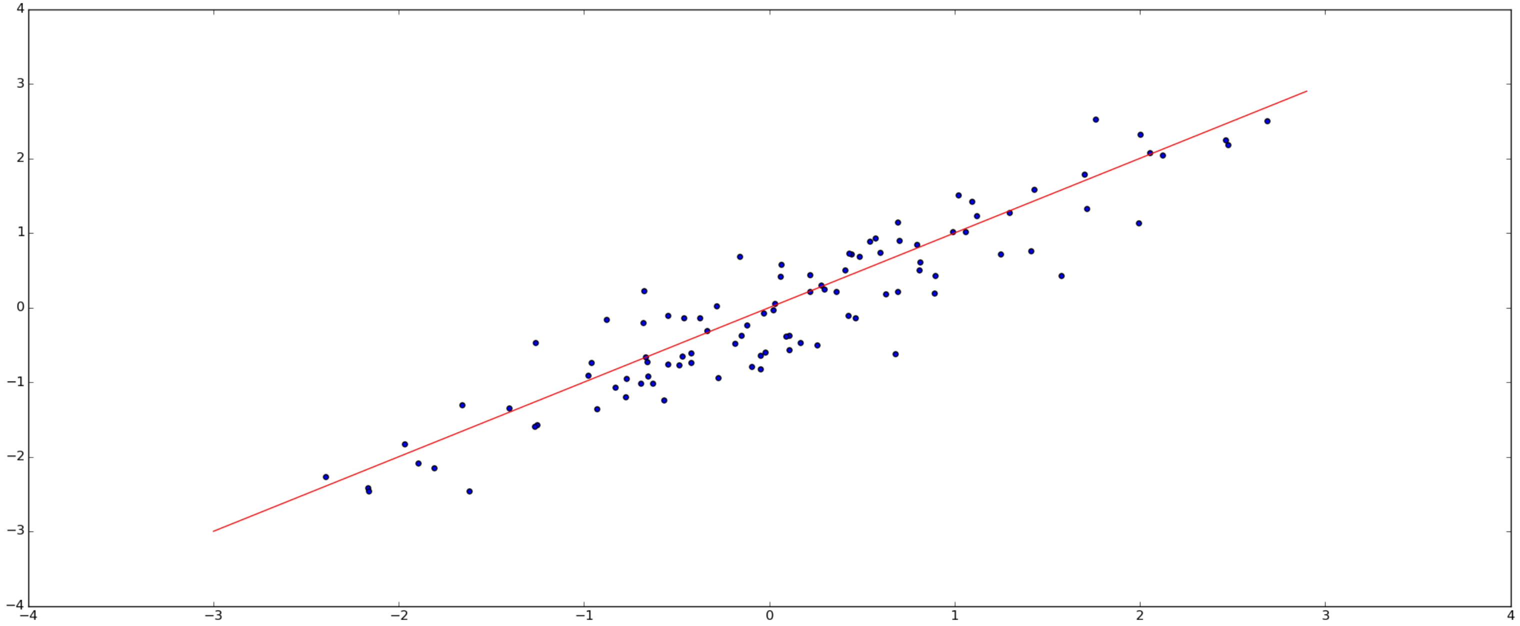
The goal is to predict y_i as a function of x_i

Our model describing y_i is:

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

where α and β are unknown parameters, and ε is the statistical noise (a random variable)

Our goal will be to compute a *posterior* on (α, β) that represents our degree of belief that any particular (α, β) is the "correct" one



Simple Linear Regression

Our model:

$$y = \alpha x + \beta + \varepsilon$$

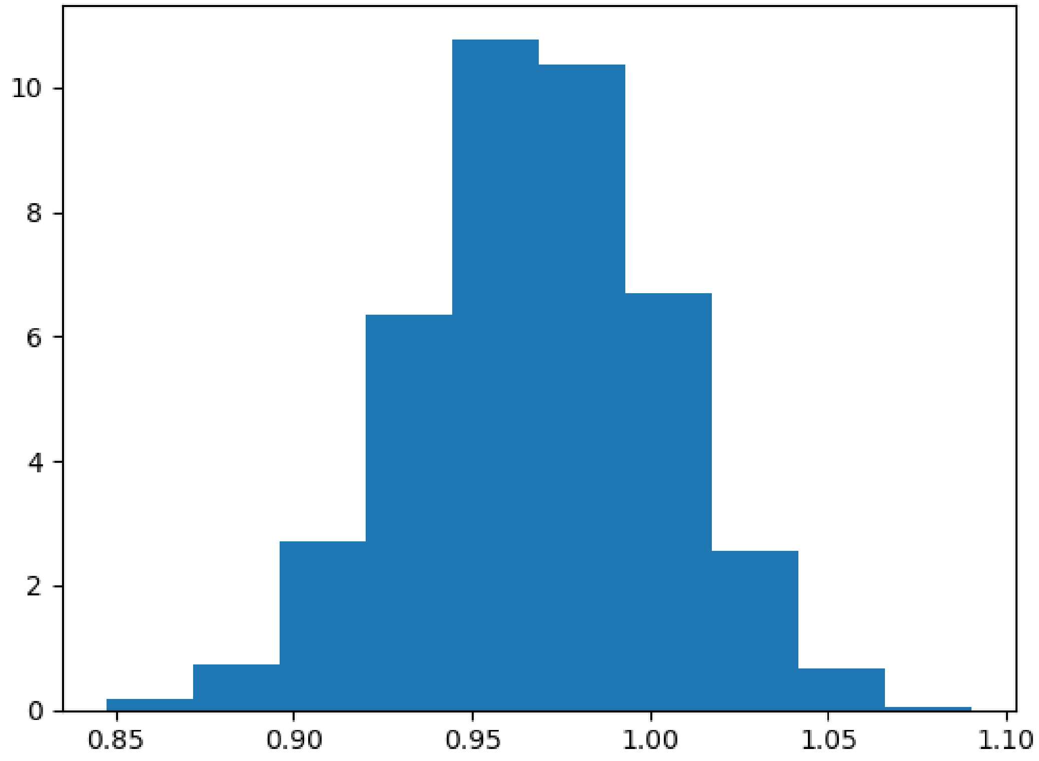
where α and β are unknown parameters, and $\varepsilon \sim \mathrm{N}(0, \frac{1}{\tau})$.

The most common priors on β and α are Normal priors.

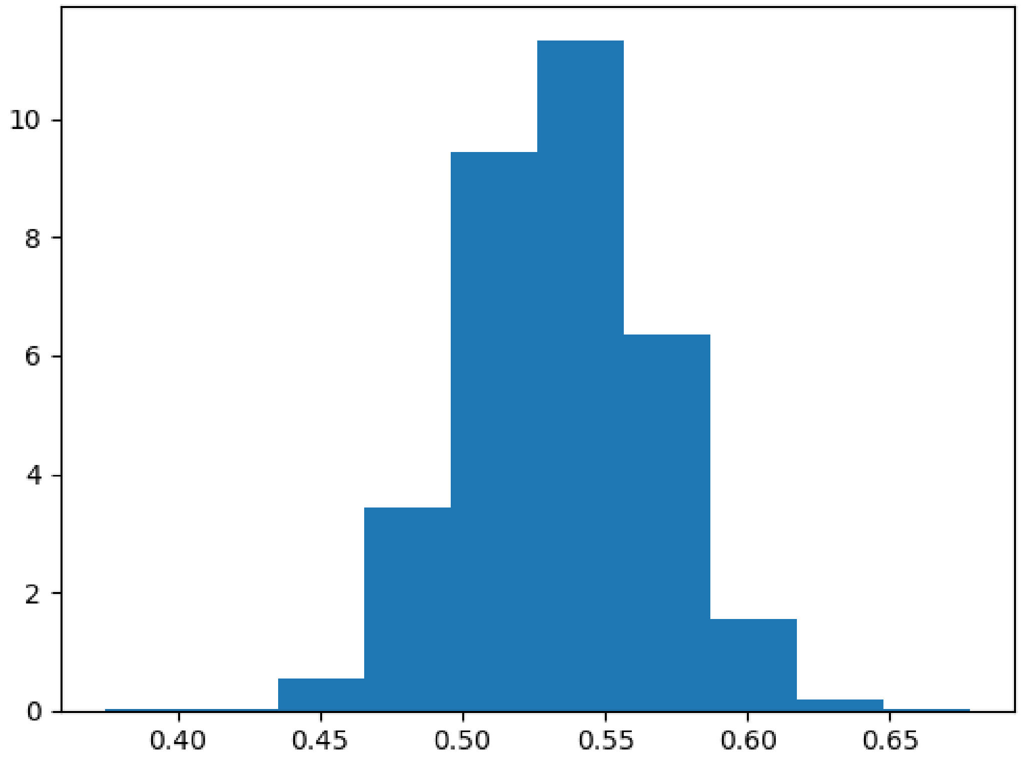
We will also assign a prior on τ, so that $\sigma = \frac{1}{\sqrt{\tau}}$ is uniform over 0 to 100 (equivalently then $\tau = \frac{1}{U(0,100)^2}$).

```
n = 100 # number of data points
x_data = np.random.normal(0, 1, n)
y_data = x_data + 0.5 + np.random.normal(0, 0.35, n)

std = pm.Uniform("std", 0, 100)

@pm.deterministic
def prec(U=std):
    return 1.0 / (U) ** 2

beta = pm.Normal("beta", 0, 0.0001)
alpha = pm.Normal("alpha", 0, 0.0001)

@pm.deterministic
def linear_regress(x=x_data, alpha=alpha, beta=beta):
    return x*alpha+beta

y = pm.Normal('y', linear_regress, prec, value=y_data, observed=True)

model = pm.Model([y, std, prec, alpha, beta])
mcmc = pm.MCMC(model)
mcmc.sample(iter=100000, burn=50000, thin=10)

alpha_samples = mcmc.trace('alpha')[:]
beta_samples = mcmc.trace('beta')[:]
std_samples = mcmc.trace('std')[:]
```

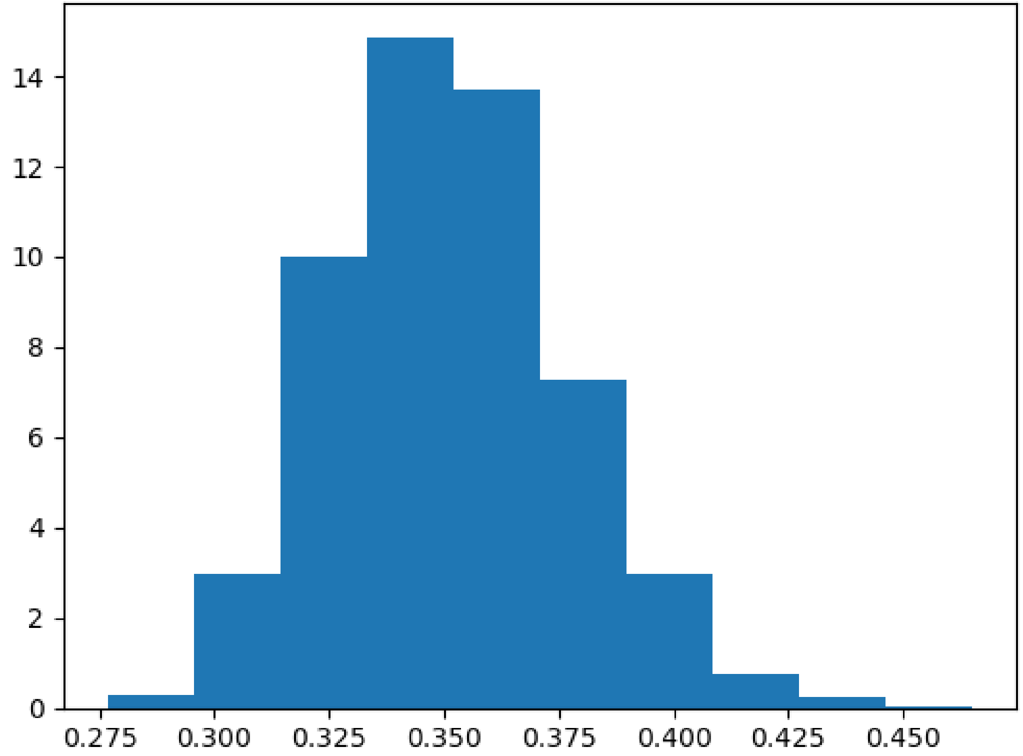
Posterior of α



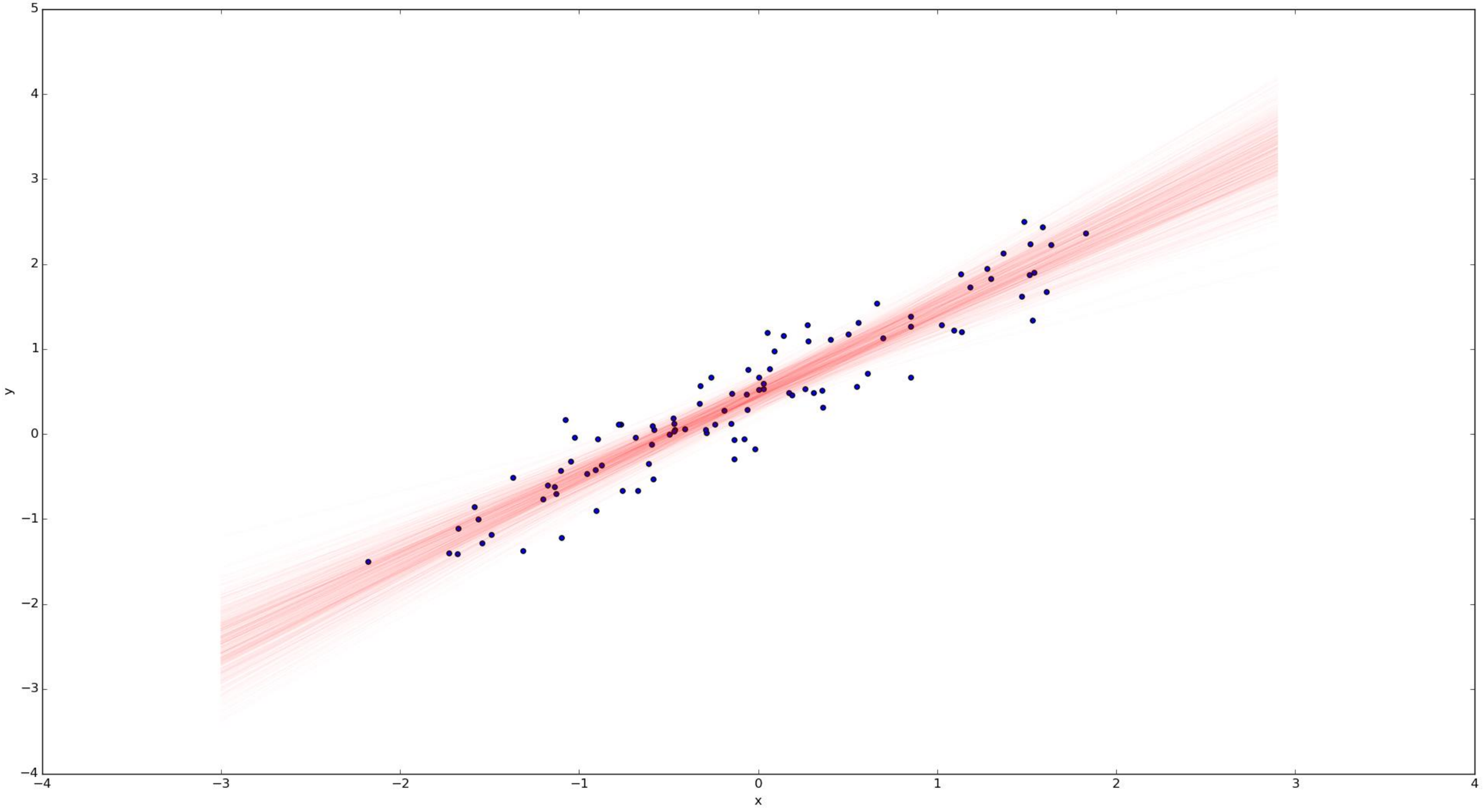
Posterior of β



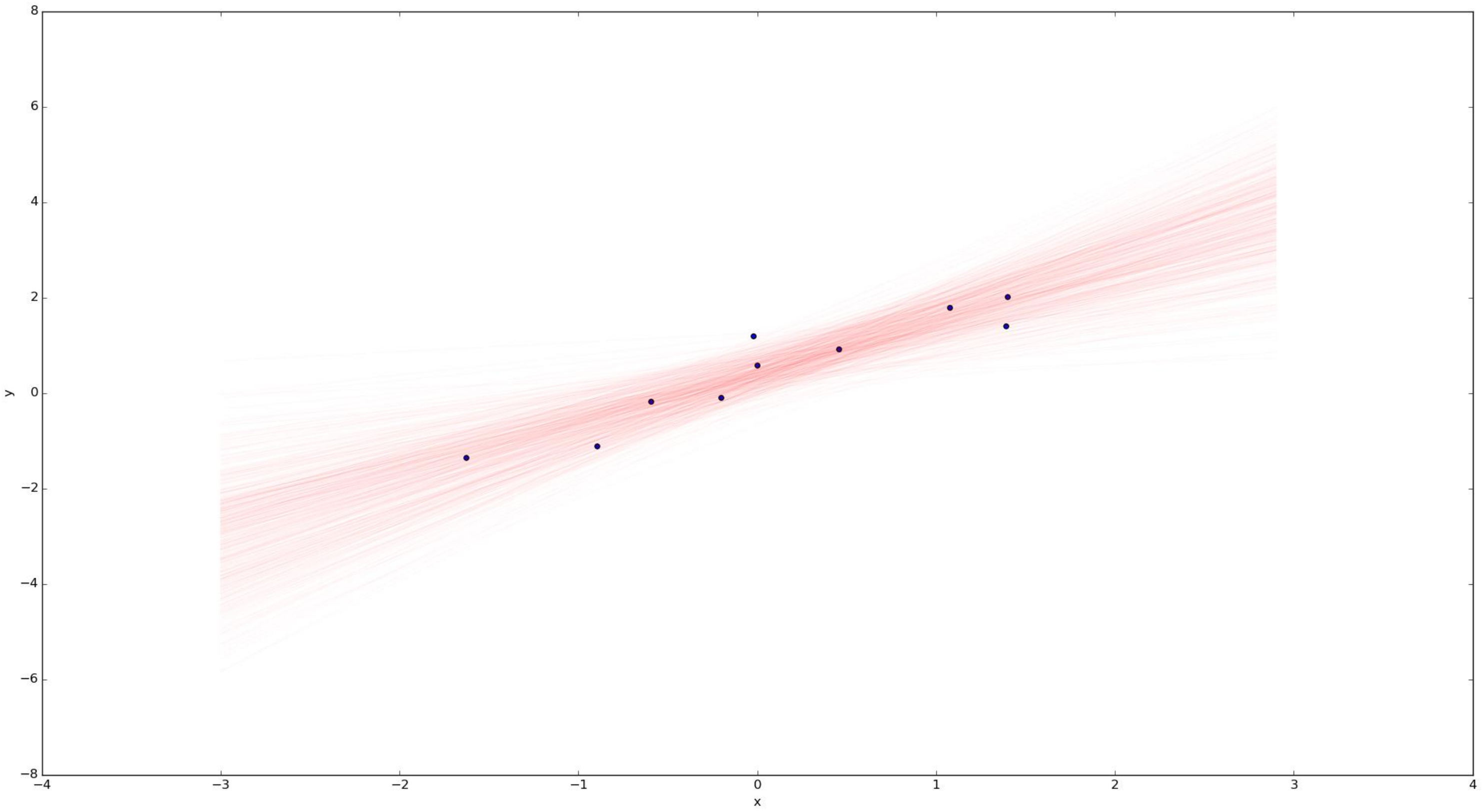
Posterior of σ



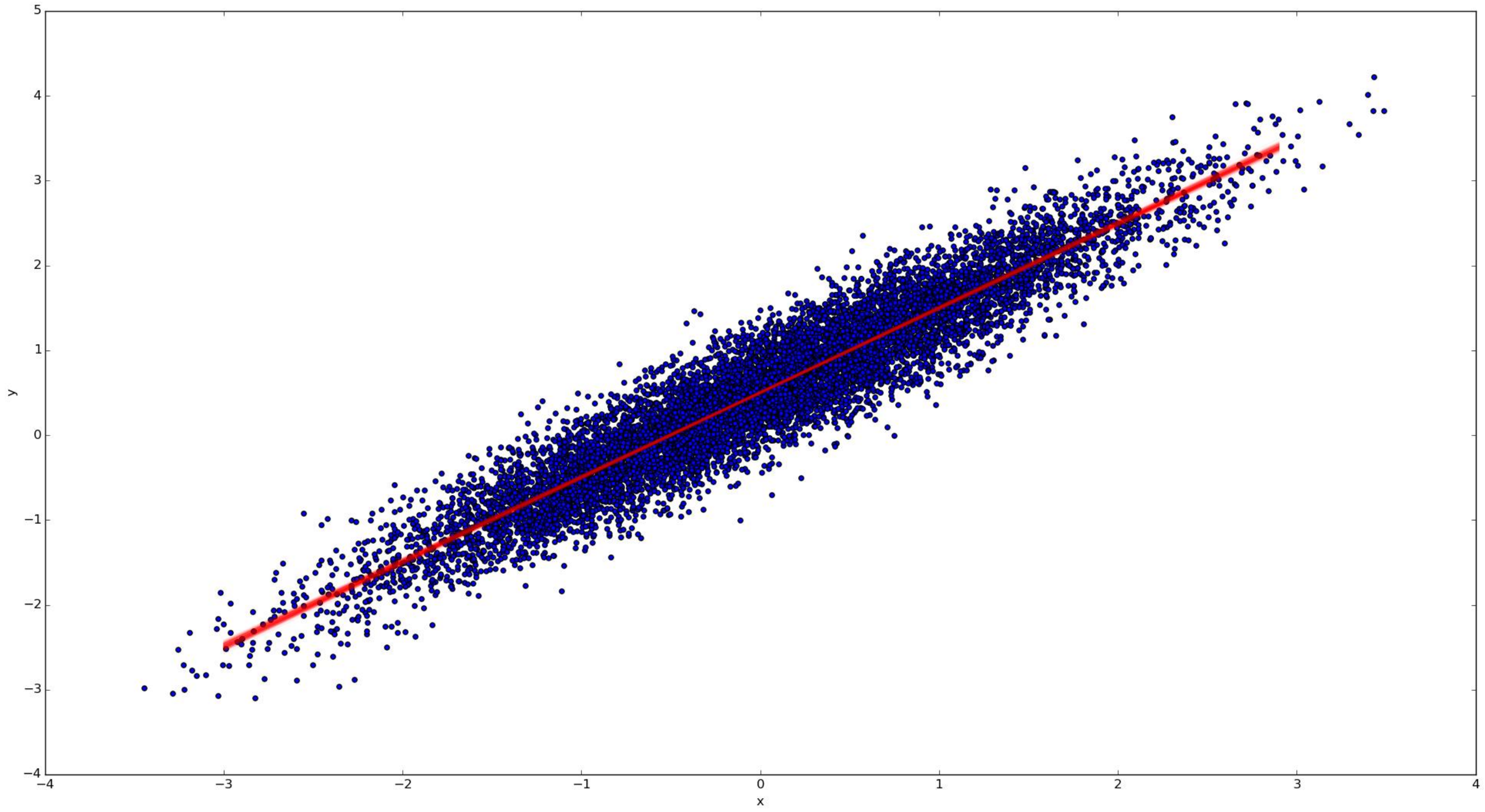
A Sample of Regression Lines



A Sample of Regression Lines (n = 10)

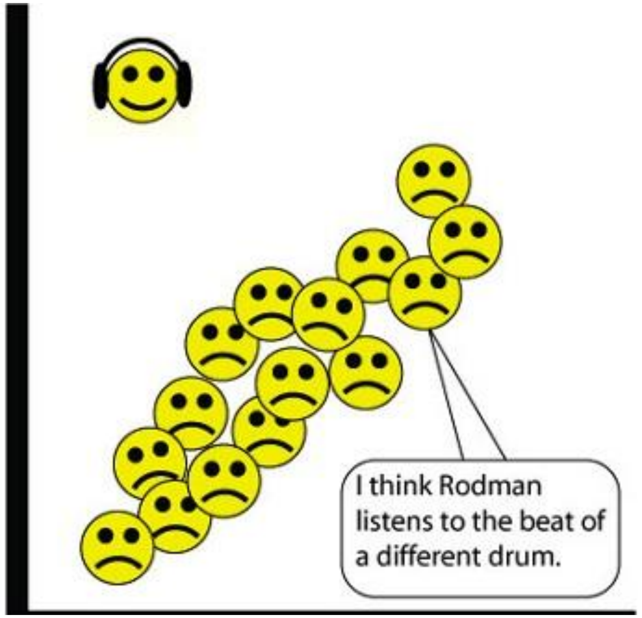


A Sample of Regression Lines (n = 1000)

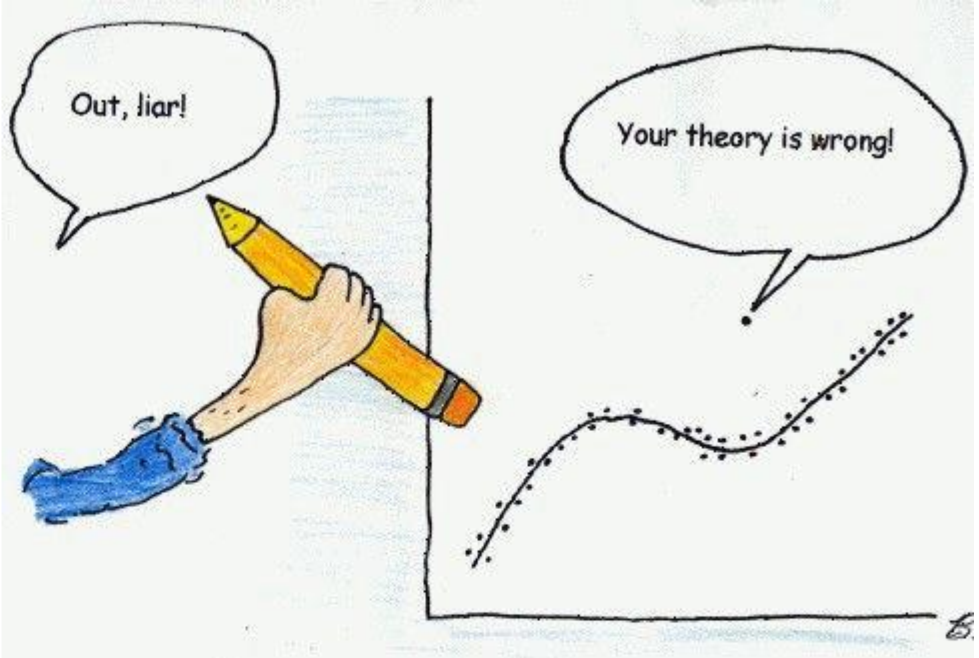


Incorporating Priors - Handling Outliers

In statistics, an outlier is an observation point that is distant from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error;. An outlier can cause serious problems in statistical analyses.



Outliers are often treated as data points which need to be specially handled - often ignored - because they don't fit the model and can heavily skew results.



Cauchy Distribution

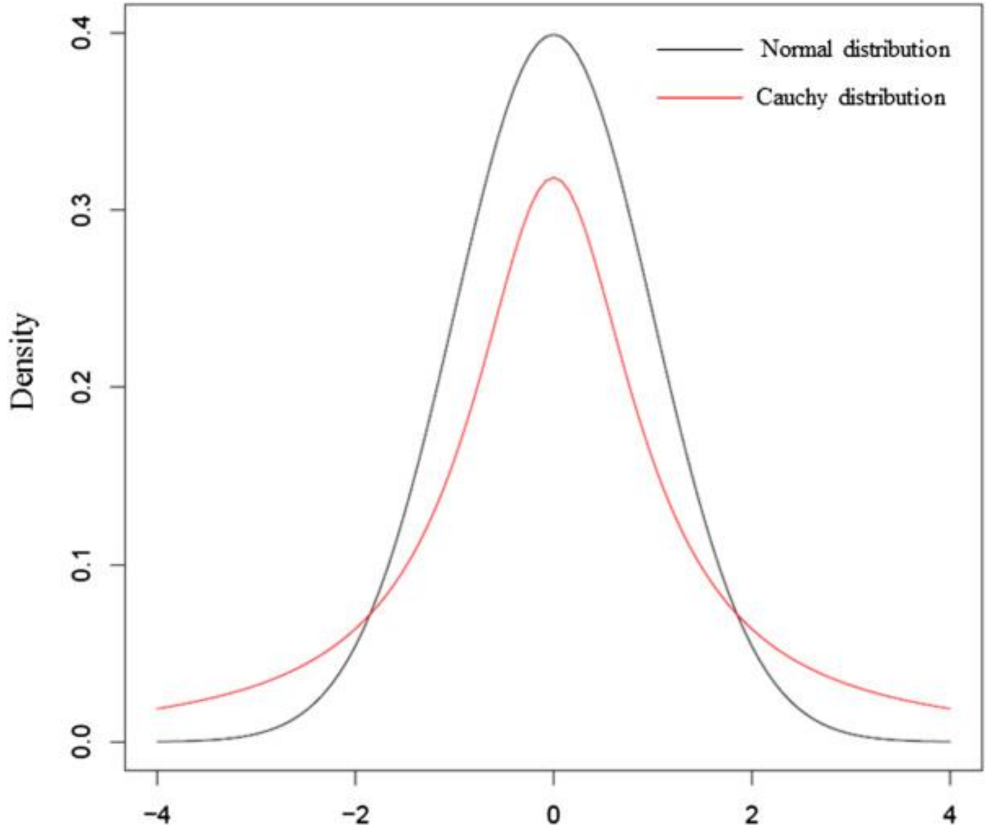
$$Z \sim \text{Cauchy}(\alpha, \beta)$$

$$f_Z(z|\alpha,\beta) = \frac{1}{\pi\beta\left[1+\left(\frac{x-\alpha}{\beta}\right)^2\right]}, \qquad \beta > 0,$$

$$\frac{1}{\pi\beta} = \text{mod and median at } \alpha$$

Pathological distribution

Both its expected value and its variance are undefined



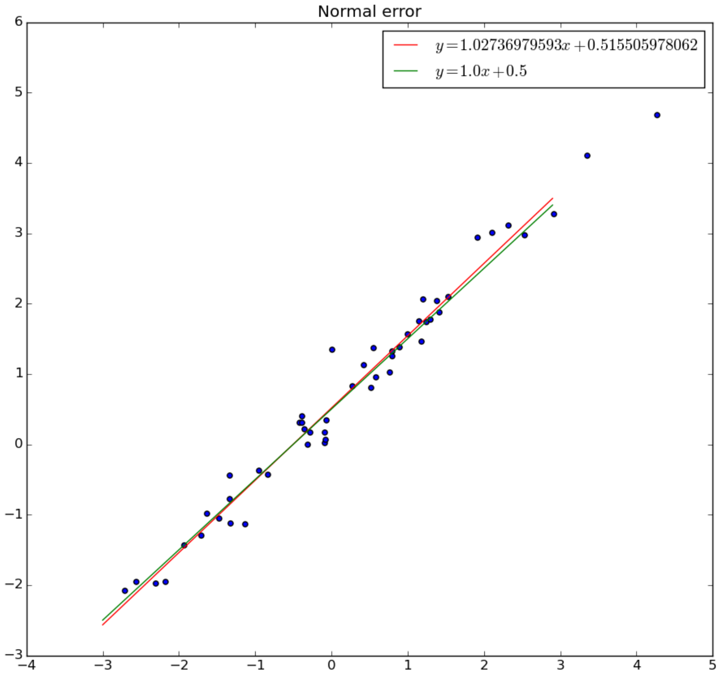
```
Z=pymc.Cauchy("Z", alpha, beta)
```

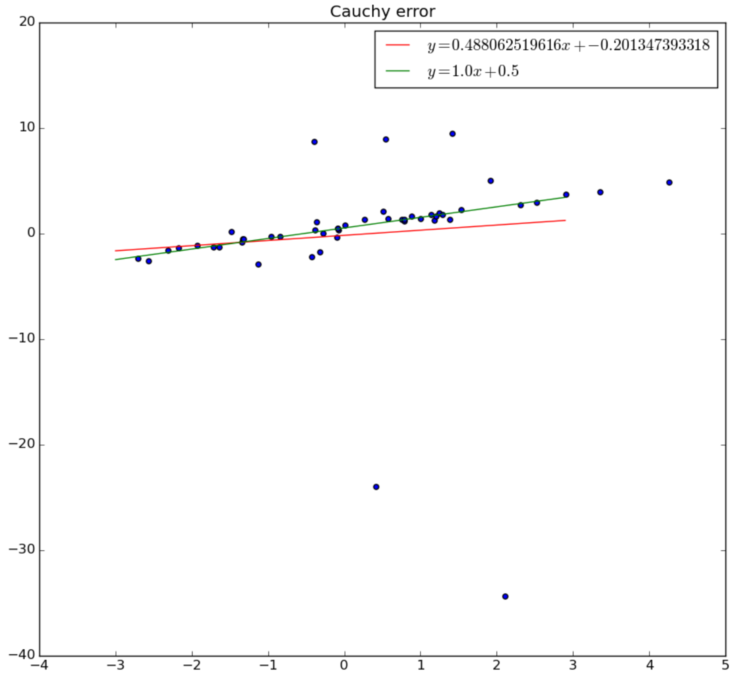
Changing the distribution of ε

50 data points were generated. In the left graph, the y-values were chosen according to the rule $y = x + 0.5 + \epsilon$ where ϵ is drawn from a normal distribution. In the right graph, the y-values were chosen according to the same rule, but with ϵ drawn from a Cauchy distribution.

The Cauchy distribution is actually pretty pathological - it's variance is infinite. The result of this is that "outliers" are not actually uncommon at all - extremely large deviations from the mean are perfectly normal.

The red line in the graph the best possible least squares fit line, while the green line is the true relation.





Incorporating Priors

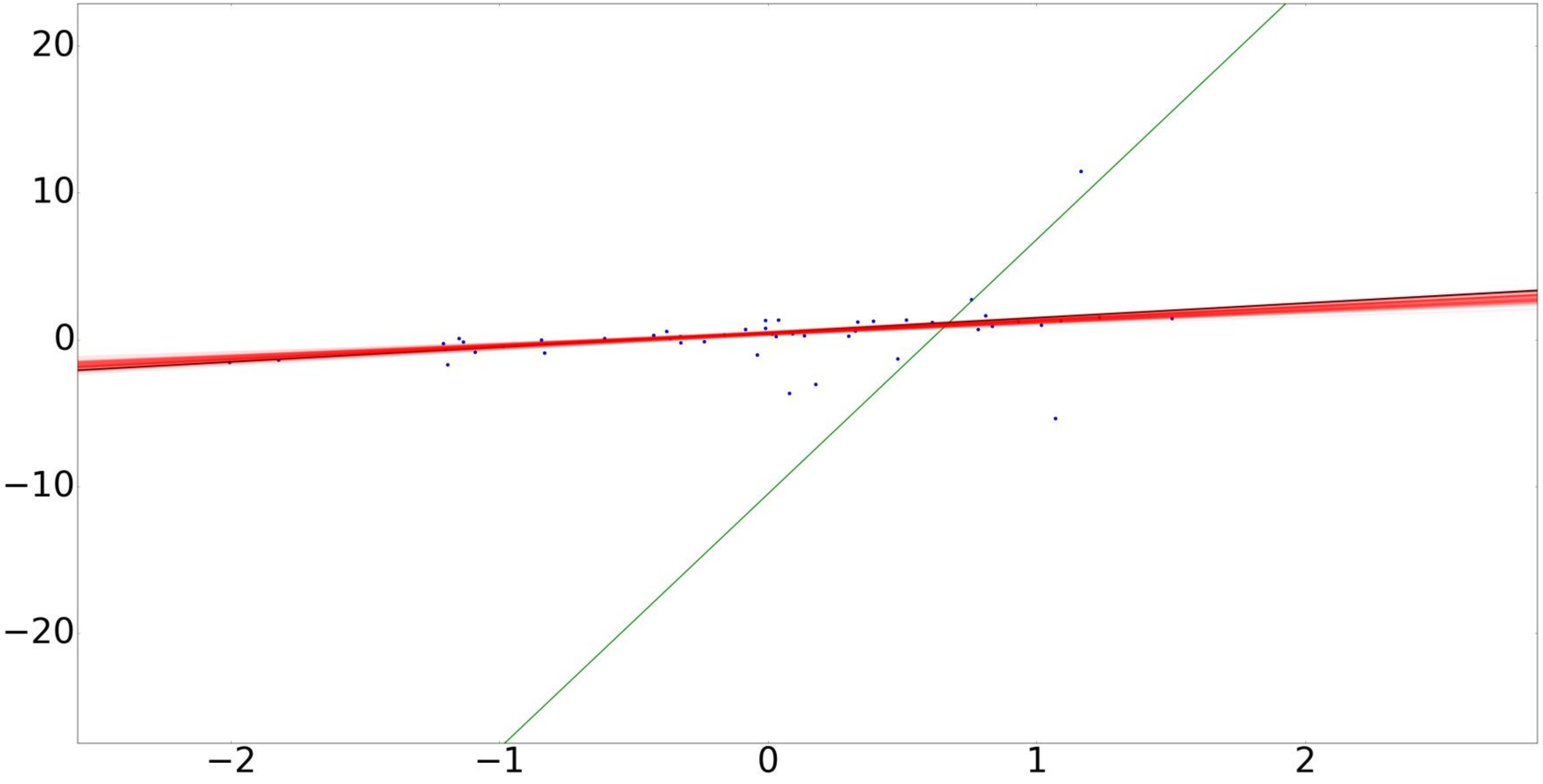
All we have to do

If we use Bayesian linear regression and simply change the distribution on the error in y to a Cauchy distribution, Bayesian linear regression adapts perfectly

```
…

std = pm.Uniform("std", 0, 100)

@pm.deterministic
def prec(U=std):
    return 1.0 / (U) ** 2

beta = pm.Normal("beta", 0, 0.0001)
alpha = pm.Normal("alpha", 0, 0.0001)

@pm.deterministic
def linear_regress(x=x_data, alpha=alpha, beta=beta):
    return x*alpha+beta

y = pm.Normal('y', linear_regress, prec, value=y_data, observed=True)
y = pm.Cauchy('y', linear_regress, 0.35, value=y_data, observed=True)
```

The Results

In this picture, we have 50 data points with Cauchy-distributed errors. The black line represents the true line used to generate the data. The green line represents the best possible least squares fit, which is driven primarily by the data point at (1, 10). The red lines represent samples drawn from the Bayesian posterior on (α, β).



Bayesian Multivariate Linear Regression

Poverty, Unemployment, Murder rate

I - the index

A1 - the inhabitants

A2 - the percentage of families incomes below $5000

A3 - the percentage unemployed

B - the number of murders per 1,000,000 inhabitants per annum.

Helmut Spaeth, Mathematical Algorithms for Linear Regression, Academic Press, 1991,

I	A1	A2	A3	B
1	587000	16.5	6.2	11.2
2	643000	20.5	6.4	13.4
3	635000	26.3	9.3	40.7
4	692000	16.5	5.3	5.3
5	1248000	19.2	7.3	24.8
6	643000	16.5	5.9	12.7
7	1964000	20.2	6.4	20.9
8	1531000	21.3	7.6	35.7
9	713000	17.2	4.9	8.7
10	749000	14.3	6.4	9.6
11	7895000	18.1	6.0	14.5
12	762000	23.1	7.4	26.9
13	2793000	19.1	5.8	15.7
14	741000	24.7	8.6	36.2
15	625000	18.6	6.5	18.1
16	854000	24.9	8.3	28.9
17	716000	17.9	6.7	14.9
18	921000	22.4	8.6	25.8
19	595000	20.2	8.4	21.7
20	3353000	16.9	6.7	25.7

Bayesian Multivariate Linear Regression

```
std = pm.Uniform("std", 0, 100)

@pm.deterministic
def prec(U=std):
    return 1.0 / (U) ** 2

beta = pm.Normal("beta", 0, 0.0001)

d = x_data.shape[1]
alpha = np.empty(d, dtype=object)
for i in range(d):
    alpha[i] = pm.Normal('alpha_%i' % i, 0, 0.0001)

@pm.deterministic
def linear_regress(x=x_data, alpha=alpha, beta=beta):
    return x.dot(alpha) + beta

y = pm.Normal('y', linear_regress, prec, value=y_data,
              observed=True)

model = pm.Model([y, std, prec, pm.Container(alpha), beta])
mcmc = pm.MCMC(model)
mcmc.sample(iter=100000, burn=50000, thin=10)
```

```
ae = np.empty(d)
for i in range(d):
      ae[i] = np.mean(mcmc.trace('alpha_%i' % i)[:], axis = 0)
be = np.mean(mcmc.trace('beta')[:], axis = 0)
yh = xt.dot(ae) + be
print('Yh             Yt    MSE’)
for i in range(yt.shape[0]):
    print(yh[i], yt[i], (yh[i] - yt[i]) ** 2)
for i in range(d):
      alpha_samples = mcmc.trace('alpha_%i' % i)[:]
      plt.hist(alpha_samples)
      plt.show()


beta_samples = mcmc.trace('beta')[:]


plt.hist(beta_samples)
plt.show()
```

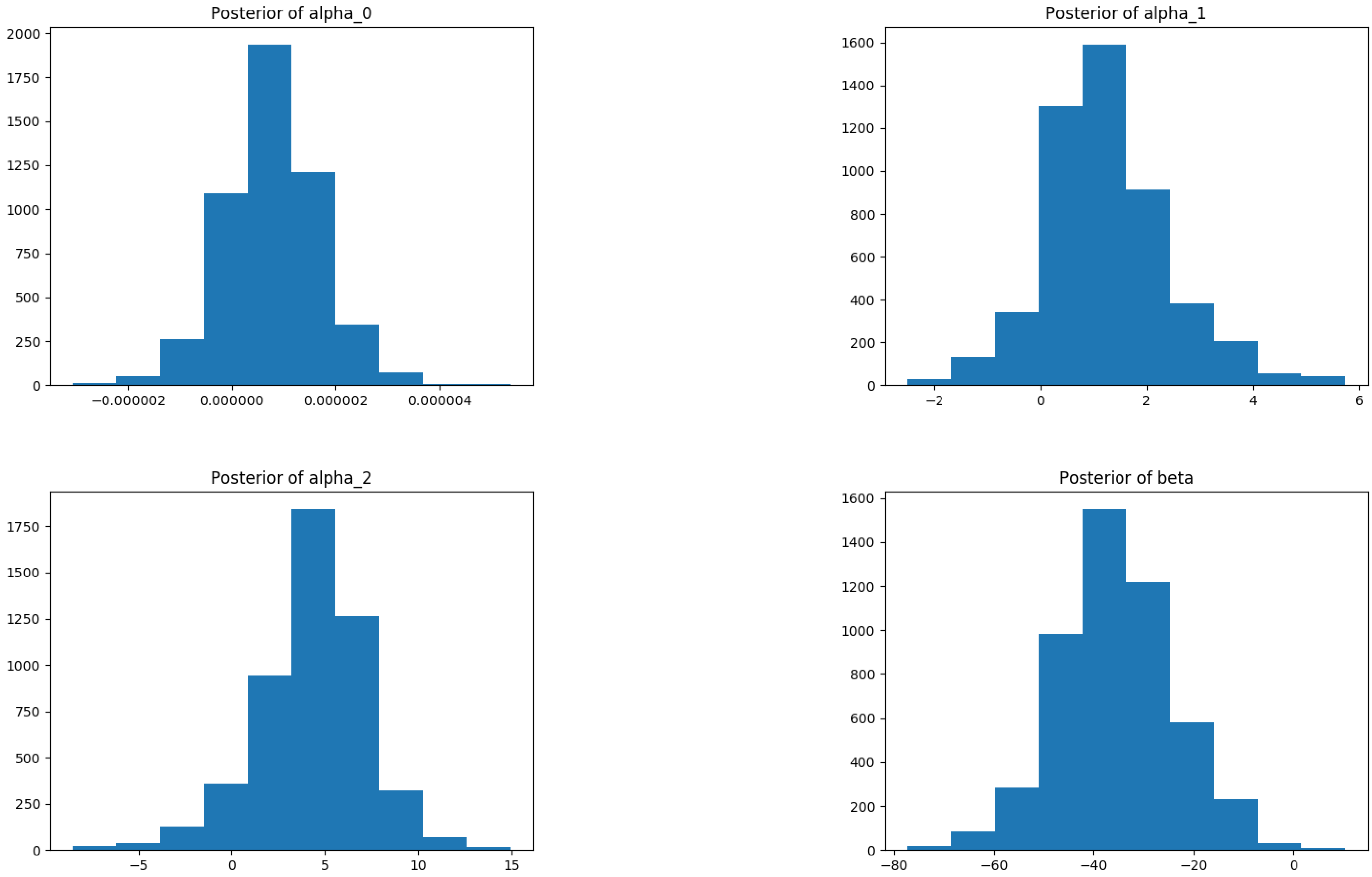
The Results

Bayesian

```
Yh              Yt    MSE
38.2743755645 40.7 5.88365390225
21.0844168026 24.8 13.8055584971
10.6573259035 9.6 1.11793806621
15.4250721385 15.7 0.0755853290386
16.2591949298 18.1 3.38856330644

        MSE = 4.8542
```

Least Squares

```
Yh         Yt          MSE
37.8029    40.7000     8.3930
21.2216    24.8000    12.8049
11.3111     9.6000     2.9278
15.2975    15.7000     0.1620
16.3539    18.1000     3.0490

      MSE = 5.4673
```


Posterior of alpha_0

Posterior of alpha_1

Posterior of alpha_2

Posterior of beta

Bayesian Perspective of Penalized Linear Regressions

The Probabilistic Interpretation of Least-Squares Linear Regression

Denote our response variable Y, and features are contained in the data matrix X. The standard linear model is:

$$Y = X\beta + \epsilon$$

where $\varepsilon \sim N(\mathbf{0}, \sigma\mathbf{I})$. The observed Y is a linear function of X (with coefficients β plus some noise term. Our unknown to be determined is β.

We use the following property of Normal random variables:

$$\mu' + N(\mu, \sigma) \sim N(\mu' + \mu, \sigma)$$

to rewrite the above linear model as:

$$Y = X\beta + N(\mathbf{0}, \sigma\mathbf{I}) \sim N(X\beta, \sigma\mathbf{I})$$

The Probabilistic Interpretation of Least-Squares Linear Regression

$$Y \sim N(X\beta, \sigma\mathbf{I})$$

The probability distribution of $Y|\beta, X$ (which is also the likelihood of $\beta|Y, X$)is:

$$f_Y(Y|\beta, X) = L(\beta|Y, X) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2\sigma^2}(Y - X\beta)^T(Y - X\beta)}$$

The log likelihood is:

$$l(\beta) = K - c(Y - X\beta)^T(Y - X\beta)$$

where K and $c > 0$ are constants

Maximum likelihood techniques wish to maximize this for β:

$$\hat{\beta} = \mathrm{argmax}_\beta - (Y - X\beta)^T(Y - X\beta)$$

Equivalently we can minimize the negative of the above:

$$\hat{\beta} = \mathrm{argmin}_\beta(Y - X\beta)^T(Y - X\beta)$$

This is the familiar least-squares linear regression equation

The Bayesian Perspective

Once we have the likelihood, we can include a prior distribution on β to derive the equation for the posterior distribution:

$$P(\beta|Y, X) = L(\beta|Y, X)p(\beta)$$

where $p(\beta)$ is a prior on the elements of β.

The Bayesian equivalent of maximum likelihood is maximum a posterior (MAP) that wish to maximize the posterior distribution.

If we include no explicit prior term, we are actually including an uninformative prior, $p(\beta) \propto 1$ (think of it as uniform over all numbers) and MAP is equal to maximum likelihood and so with least-squares linear regression.

Ridge Regression

If we have reason to believe the elements of β are not too large, we can suppose that a priori:

$$\beta \sim N(\mathbf{0}, \boldsymbol{\lambda}\mathbf{I})$$

The resulting posterior density function for β is proportional to:

$$e^{-\frac{1}{2\sigma^2}(Y-X\beta)^T(Y-X\beta)}e^{-\frac{1}{2\lambda^2}\beta^T\beta}$$

and taking the log of this, and combining and redefining constants, we arrive at:

$$l(\beta) = K - c(Y - X\beta)^T(Y - X\beta) - \frac{\tau}{2}\beta^T\beta$$

$$\hat{\beta} = \mathrm{argmax}_\beta - (Y - X\beta)^T(Y - X\beta) - \frac{\tau}{2}\beta^T\beta$$

Equivalently we can minimize the negative of the above, and rewriting $\beta^T\beta = ||\beta||_2^2$:

$$\hat{\beta} = \mathrm{argmin}_\beta(Y - X\beta)^T(Y - X\beta) + \frac{\tau}{2}||\beta||_2^2$$

This term is exactly Ridge Regression

LASSO

Similarly, if we assume a Laplace prior on β:

$$f_\beta(\beta) \propto e^{-\lambda\|\beta\|_1}$$

and following the same steps as above, we recover:

$$\hat{\beta} = \operatorname{argmin}_\beta (Y - X\beta)^T (Y - X\beta) + \lambda\|\beta\|_1$$

which is LASSO regression