

PLS 900/803 Bayesian Analysis

Shahryar Minhas [s7minhas.com]

Readings to go with this lecture

Two books that I highly recommend, McElreath's Statistical Rethinking (linked below) provides a great and friendly introduction to Bayesian analysis. A much more comprehensive book comes is Bayesian Data Analysis from Gelman et al (also linked below). For those of you interested in Bayes, I recommend delving into each. For our purposes the below is all that goes with this lecture:

- [Statistical Rethinking Ch 4 and 5](#)
- Advanced:
 - [Bayesian Data Analysis Ch 10 and 11](#)
 - [Conceptual Introduction to Hamiltonian Monte Carlo](#)
- Other:
 - [High level overview](#)
 - [Bayes and Qual Approaches I](#)
 - [Bayes and Qual Approaches II](#)

MLE Synopsis: From random variables to models

Suppose we are studying outcome y and we decide it is distributed f

- Stochastic component: $y \sim f(\mu, \alpha)$
- Systematic component: $\mu = g(X, \beta)$

This formulation encompasses all the models in this course.

For example, you are used to seeing linear regression written this way:

$$y_i = X_i\beta + \epsilon_i$$
$$\epsilon_i \sim f_N(0, \sigma^2)$$

In our notation, this is equivalent to assuming $f(\cdot)$ is the Normal distribution:

$$y_i \sim f_N(\mu_i, \sigma^2)$$
$$\mu_i = X_i\beta$$

From models to inference

- It's not possible to infer $P(\theta|y)$ from y alone
- If we assume a distribution for y , we can solve for $P(y|\theta)$...

$$\begin{array}{lcl} \text{conditional probability} & = & \frac{\text{joint probability}}{\text{marginal probability}} \\ P(\theta|y) = \frac{P(\theta \cap y)}{P(y)} & & P(y|\theta) = \frac{P(\theta \cap y)}{P(\theta)} \end{array}$$

$$\begin{array}{lcl} P(y)P(\theta|y) = P(\theta \cap y) & & P(\theta)P(y|\theta) = P(\theta \cap y) \\ P(y)P(\theta|y) & = & P(\theta)P(y|\theta) \end{array}$$

$$P(\theta|y) = \frac{P(\theta)P(y|\theta)}{P(y)}$$

This famous result is known as Bayes rule, it shows how to write a conditional probability $P(a|b)$ in terms of its inverse $P(b|a)$

From models to inference

$$P(\theta|y) = \frac{P(\theta)P(y|\theta)}{P(y)} \text{ (Bayes Rule)}$$

So to infer the probability of the parameters θ given the data y , we need to know $P(\theta)$ and $P(y)$ a priori

We can rewrite Bayes rule to replace $P(y)$ with other quantities:

$$\begin{aligned} P(\mathbf{y}) &= \int_{\Theta} P(\boldsymbol{\theta} \cap \mathbf{y}) d\boldsymbol{\theta} \\ &= \int_{\Theta} P(\boldsymbol{\theta}) P(\mathbf{y}|\boldsymbol{\theta}) d\boldsymbol{\theta} \end{aligned}$$

But $P(\theta)$ is not known *objectively*

From models to inference

$$P(\theta|y) = \frac{P(\theta)P(y|\theta)}{P(y)} \text{ (Bayes Rule)}$$

$P(\theta)$ is not known *objectively*, but we need it to compute $P(\theta|y)$

This creates a fork in the road of inference, with two major schools of thought:

Bayesian inference:

- Make a subjective guess of the a priori $P(\theta)$
- Then use $P(y|\theta)$ to calculate $P(\theta|y)$

Likelihood inference:

- Give up on calculating $P(\theta|y)$ to avoid making subjective guesses of $P(\theta)$
- Instead focus on making inferences directly from $P(y|\theta)$

What to do?

One option ...



Or ...

Lets just go with Fisher (1922) and say: $P(y|\theta) \rightarrow L(\theta|x)$



Brilliant guy ... ugly views ... on your own time:
<https://njoselson.github.io/Fisher-Pearson/> (maybe reading about him will make you want to be Bayesian :))

Fisher's justification

Fisher (1922): $P(y|\theta) \rightarrow L(\theta|x)$

- $P(y) = 1$ since the data has already occurred
- $P(\theta) = 1$, which is equivalent to putting a finite uniform prior on θ over its support

Thus:

$$P(\theta|y) = \frac{P(\theta)P(y|\theta)}{P(y)} \text{ (Bayes Rule)}$$

$$P(\theta|y) = \frac{1}{1} P(y|\theta) \text{ (Bayes Rule)}$$

Implications of this choice?

$$\mathcal{L}(\theta|x) \propto P(y|\theta)$$

- Likelihood of the parameters given the data is proportional to the probability of the data given the parameters
- We can't objectively state the probability of a particular $\hat{\theta}$ given y , we can objectively state the relative likelihood of $\hat{\theta}$ over some other $\hat{\theta}'$
 - \mathcal{L} is a surface in θ space showing which parameter values are more likely than others
 - We can look at the profile of the likelihood function against each parameter in θ to see which $\hat{\theta}$ s are likely

General steps for finding MLE

Assuming a statistical model parameterized by a fixed and unknown θ , the $L(\theta)$ is the probability of the observed data considered as a function of θ .

The process goes as follows:

- Identify the PMF or PDF.
- Create the likelihood function from the joint distribution of the observed data.
- Change to the log for convenience.
- Take the first derivative with respect to the parameter of interest.
- Set equal to zero.
- Solve for the MLE.

Measuring uncertainty of the MLE

- The first derivative measures slope and the second derivative measures *curvature* or *concavity* of the function at a given point.
- If the second derivate is *negative* the slope of the tangent line must be decreasing \rightarrow MLE is a maximum. 0 The second derivative also give you information on the *acceleration* or the rate of change with respect to your estimate.
- The more peaked the function (the greater rate of change) at the MLE, the more "certain" the data are about this estimator.
- Poisson example:

$$\frac{d}{d\theta} \ell(\theta|\mathbf{x}) = -n + \frac{1}{\theta} \sum_{i=1}^n x_i$$

$$\frac{d^2}{d\theta^2} \ell(\theta|\mathbf{x}) = \frac{d}{d\theta} \left(\frac{d}{d\theta} \ell(\theta|\mathbf{x}) \right) = -\theta^{-2} \sum_{i=1}^n x_i$$

Measuring uncertainty of the MLE

- The negative inverse of the expected value of the second derivative is the variance of the MLE.
- If we take the square root, we get the SE of the MLE.

$$Var(\hat{\theta}) = \left(- E \left[\frac{d^2}{d\theta^2} \ell(\theta|\mathbf{x}) \right] \right)^{-1}$$

- The more peaked the function (the greater rate of change) at the MLE, the more "certain" we are about this estimator. Notice the inverse relationship between the second derivate and the variance.

To summarize

The score function is the first derivative of the log-likelihood:

$$\text{Score function} \rightarrow S(\theta) = \frac{d}{d\theta} \ell(\theta)$$

At the maximum, the second derivative of the log-likelihood is negative, so we define the curvature of at $\hat{\theta}$ as $I(\theta)$, where,

$$(\text{observed}) \text{ Fisher Information} \rightarrow I(\theta) = -\frac{d^2}{d\theta^2} \ell(\theta)$$

A large curvature of $I(\hat{\theta})$, the *observed* Fisher Information, is associated with a tight or strong peak, intuitively indicating *less* uncertainty about θ .

Because $I(\theta)$ evaluates the MLE, it is a *number* rather than a function.

Notice that the standard error for $\hat{\theta}$ is calculated using the Fisher Information.

A reading list to keep in mind

- General MLE (apart from the book)
 - Myung (2003)
- Interaction
 - Ai and Norton (2003)
 - Berry et al. (2009)
- Separation
 - Zorn (2005)
 - Rainey (2016)
- Simulation
 - Tomz et al. (2000)
 - Hanmer & Kalkan (2013)
- Model Assessment
 - Perils of policy by p-value
 - Cross-Validation
 - Separation Plot

Bayesian Inference

Switching over to Bayes ...

- In the **frequentist** paradigm, we took a particular approach to thinking about *randomness* when trying to learn about an unknown parameter θ using observed data \mathbf{Y}
 - θ is an unknown **constant**
 - \mathbf{Y} are random variables
 - $\hat{\theta}$ is our estimator of θ - it's a function of random random variables
 - Probabilities reflect behavior in *repeated samples*

Bayesian inference

- **Bayesian** inference takes a different approach to the problem. Rather than treating θ as a constant, we consider it to be **random** as well!
 - Probabilities reflect **beliefs** about a particular quantity
 - $f(\theta)$ denotes our *prior* beliefs about the value of θ
 - $f(\theta|\mathbf{Y})$ is the *posterior* distribution given the observed data. This is our target of inference
 - \mathbf{Y} are still random variables...but the posterior distribution *conditions* on them
 - We derive the posterior using **Bayes rule**

$$f(\theta|\mathbf{Y}) = \frac{f(\mathbf{Y}|\theta)f(\theta)}{f(\mathbf{Y})}$$

Bayesian inference

- The denominator can be written as an integral over all of the possible values of θ (marginalizing over θ)

$$f(\mathbf{Y}) = \int f(\mathbf{Y}, \theta) d\theta = \int f(\mathbf{Y}|\theta) f(\theta) d\theta$$

- So often we'll write the posterior distribution *up to a proportionality constant* as

$$\underbrace{f(\theta|\mathbf{Y})}_{\text{posterior}} \propto \underbrace{f(\mathbf{Y}|\theta)}_{\text{likelihood}} \times \underbrace{f(\theta)}_{\text{prior}}$$

Bayesian inference

- In addition to the posterior distribution of the parameters θ , we'll often want to generate "forecasts" of an out-of-sample \tilde{Y} conditional on what we have observed
 - This involves integrating over the posterior values of the parameter θ

$$\begin{aligned}f(\tilde{Y}|\mathbf{Y}) &= \int f(\tilde{Y}, \theta|\mathbf{Y})d\theta \\&= \int f(\tilde{Y}|\theta, \mathbf{Y})f(\theta|\mathbf{Y})d\theta \\&= \int f(\tilde{Y}|\theta)f(\theta|\mathbf{Y})d\theta\end{aligned}$$

- Sound scary? Nah ... all this means is that after forming our beliefs (posterior distribution) about some parameters based on the data we've seen, we don't just make a single prediction using the "most likely" parameter value.
- Instead, we make a bunch of predictions considering all the plausible parameter values (based on our belief) and then average these predictions to get a more informed, well-rounded forecast.

Application: Predicting Elections

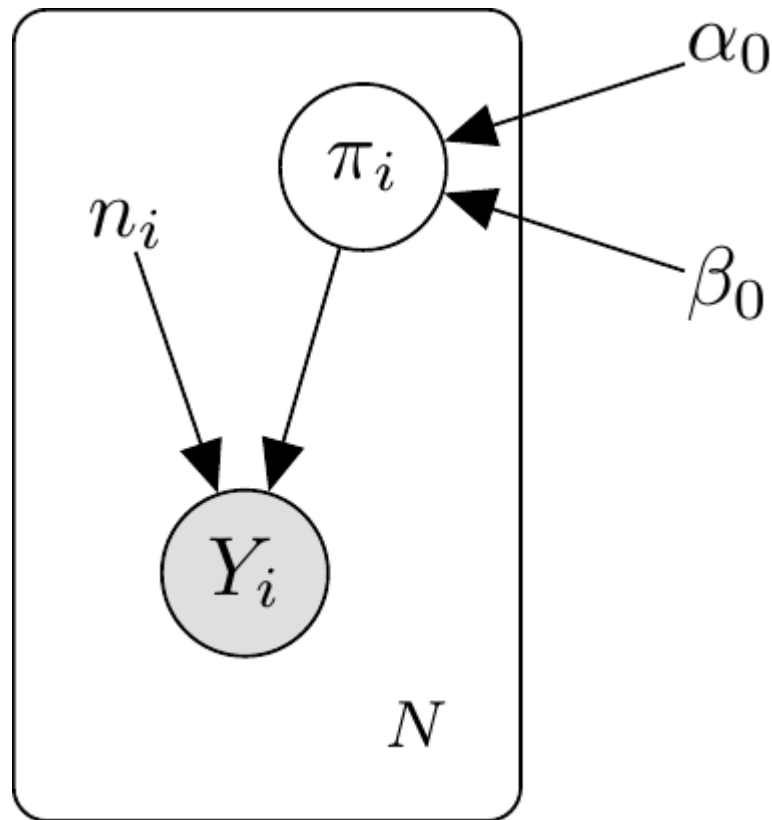
- We'd like to predict U.S. House of Representatives elections using data from the prior Presidential election at the county level
 - *Pettigrew (2018)* assembled the 2018 US House returns at the county level
 - Available as `us-house-wide.csv` on D2L
- For each county i , $i \in \{1, 2, \dots, N\}$, we observe:
 - Y_i : The number of votes for the Democratic candidate county
 - n_i : The number of votes cast in total in that county
- Let's start by building a simple model and going from there.
 - Assume Y_i is a draw from a binomial distribution with a total number of trials K_i and "success" probability π .
 - π_i comes from a **Beta** distribution with *hyperparameters* α_0 and β_0

$$Y_i \sim \text{Binomial}(n_i, \pi_i)$$

$$\pi_i \sim \text{Beta}(\alpha_0, \beta_0)$$

Plate notation

- A common method of writing statistical models is via **plate notation**
 - These concisely encode independence and dependence assumptions across parameters and data.



Features of a Bayesian model I

- There are four types of variables in a Bayesian model
 - **Observed Data:** Variables that have a probability distribution but on whose observed values we condition - Y_i
 - **Known Constants:** Fixed quantities that do not have probability distributions (e.g. regressors or other features of the nodes) - $n_i, N, \alpha_0, \beta_0$
 - **Deterministic quantities:** Transformations of other variables
 - **Latent parameters:** Variables that have a probability distribution that we do not observe - π_i

Features of a Bayesian model II

- Sometimes the known constants are *actually* known by us (e.g. N or n_i) and in other cases they are *assumed to be known*
 - These are typically parameters of the prior distribution which are called **hyperparameters** (here: α_0 and β_0)
 - The hyperparameters govern the distribution of the prior -- crucially, its mean and variance.
- In Bayesian inference, we are interested in obtaining either the posterior of the latent parameters *or* integrating them out (in the latter case, they are sometimes referred to as "nuisance" parameters)
 - Here, we want to obtain $f(\pi_i | \mathbf{Y})$
 - When writing the posterior, we'll often omit the implicit conditioning on the observed constants.

The Likelihood

- Remember the posterior distribution is proportional to the **likelihood** times the **prior**

$$\underbrace{f(\pi_i|\mathbf{Y})}_{\text{posterior}} \propto \underbrace{f(\mathbf{Y}|\pi_i)}_{\text{likelihood}} \times \underbrace{f(\pi_i)}_{\text{prior}}$$

- First, let's derive the **likelihood** $f(\mathbf{Y}|\pi_i)$
 - Since the Y_i are independent conditional on π_i , we can write:

$$f(\mathbf{Y}|\pi_i) = \prod_{j=1}^N f(Y_j|\pi_i)$$

- Next, for $j \neq i$, we have $f(Y_j|\pi_i) = f(Y_j)$ since Y_j only depends on π_j . That's just a constant term and drops out of the posterior distribution, leaving

$$f(\pi_i|\mathbf{Y}) = f(\pi_i|Y_i) \propto f(Y_i|\pi_i)f(\pi_i)$$

The Likelihood and Prior

- What's $f(Y_i|\pi_i)$? We've defined it as the binomial PMF

$$f(Y_i|\pi_i) = \binom{n_i}{Y_i} \pi_i^{Y_i} (1 - \pi_i)^{n_i - Y_i}$$

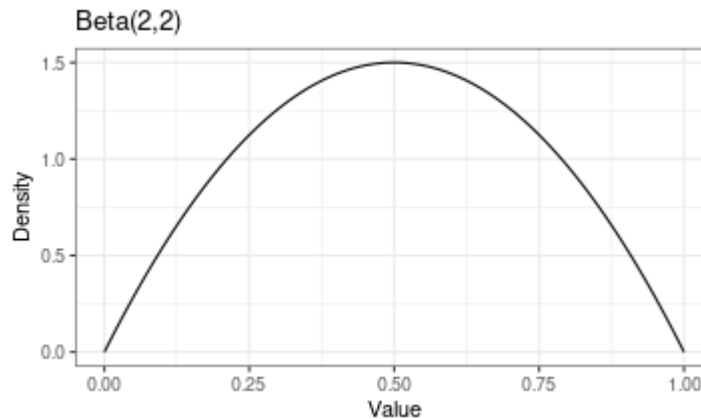
- What about the prior? $f(\pi_i)$? We've chosen the **beta** distribution

$$f(\pi_i) = \frac{\Gamma(\alpha_0)\Gamma(\beta_0)}{\Gamma(\alpha_0 + \beta_0)} \pi_i^{\alpha_0 - 1} (1 - \pi_i)^{\beta_0 - 1}$$

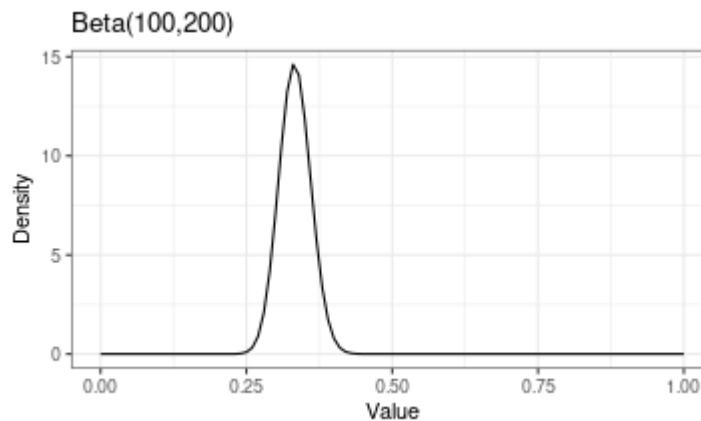
- The Beta distribution has some notable features.
 - Two parameters: α_0 and β_0 (can be thought of as pseudo counts of successes and failures, respectively)
 - It's mean: $E[\pi_i] = \frac{\alpha_0}{\alpha_0 + \beta_0}$
 - It's variance $Var(\pi_i) = \frac{\alpha_0 \beta_0}{(\alpha_0 + \beta_0)^2 (\alpha_0 + \beta_0 + 1)}$

The Beta Distribution

- Let's see how the parameters influence the shape of the beta. Beta(2, 2):



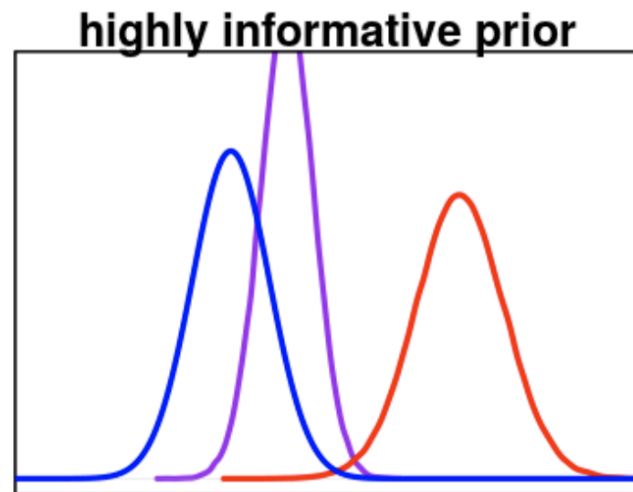
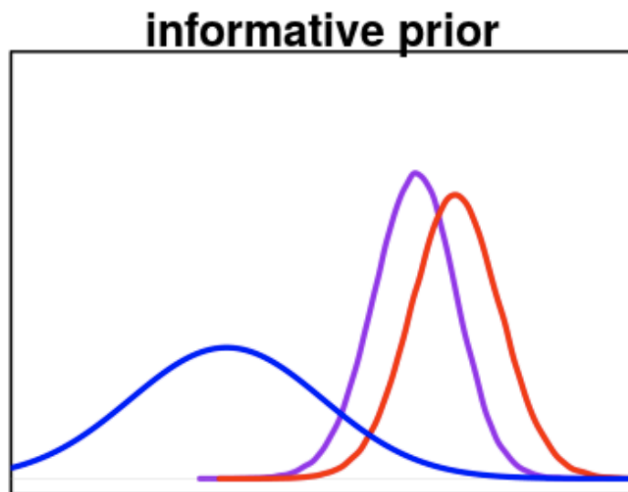
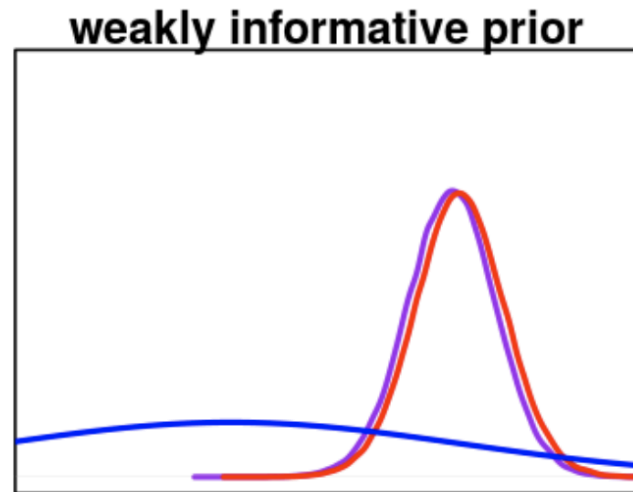
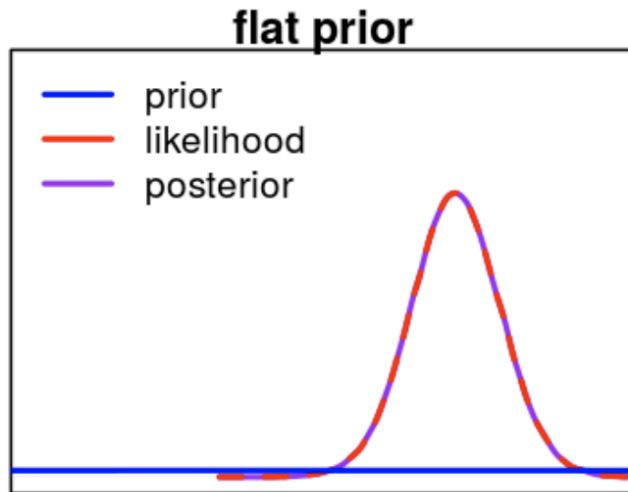
- Beta(100,200):



Importance of Prior Distributions

- Priors reflect **external knowledge** or beliefs.
- Can **stabilize estimates** in small samples.
- Can also **bias results** if chosen inappropriately.

Influence of prior distribution



Choosing a prior

- There are two ways to think about choosing a prior distribution
 - **Informative** prior - Use the prior to encode our existing beliefs about the parameter
 - **Uninformative/Diffuse** prior - Pick a prior that will have the least impact on the posterior distribution
- We also need to consider the *shape* of the prior distribution $f(\pi_i)$
 - Why did we pick the Beta distribution?
 - Because it has a special property when combined with a binomial likelihood. It is a **conjugate** prior to the binomial likelihood.
- **Conjugate prior**: A prior distribution is *conjugate* to a particular likelihood if the posterior distribution is of the same form as the prior
 - $f(\theta)$ is beta; $f(\mathbf{Y}|\theta)$ is binomial; $f(\theta|\mathbf{Y})$ is beta

Conjugate Priors

- Advantages:
 - Analytical tractability (can find the posterior without numerical methods).
 - Computational efficiency.
- Common conjugate priors for Gaussian likelihood: Normal (for mean) and Inverse Gamma (for variance).
- Lots of other examples [here](#).

Hyperparameters

- Priors themselves have parameters (called hyperparameters).
- For example, a normal prior has mean and variance.
- Hyperparameters can be:
 - Fixed based on strong beliefs or external information.
 - Estimated from the data (empirical Bayes).

Choosing a prior

- In our application, π_i must be between 0 and 1, so we already have some information.
- One possible choice for an *uninformative* prior is the uniform distribution - each parameter value is equally likely:

$$f(\pi_i) \propto 1$$

- In settings where the parameter takes on values $(-\infty, \infty)$, we could still consider a "uniform" prior but it will be **improper** as it's not a density that integrates to 1
- For $\pi_i \in (0, 1)$, the uniform prior corresponds to the Beta(1, 1) distribution
 - Originally proposed by Bayes.
 - This flat prior assumes that all probabilities are equally plausible ... i.e., we should get a result similar to the MLE.

Bayesian updating

- Now that we have defined our model: variables, likelihood, and prior; we can feed it with our data to obtain the posterior distribution.
- The process of going from the prior to the posterior is called Bayesian updating.
- We can view the prior as our initial belief of the possible values that our parameter of interest (in this case probability of democrat winning) can take.
- Then we collect some data and update our prior using the likelihood to obtain the posterior.
- This process can be repeated iteratively; the posterior becomes a new prior, we collect more data and update our posterior.
- In practice, we feed the data only once to our statistical model, but it is important to think that Bayesian updating is an iterated learning process.
- So how do we do it?

The posterior density

- We've mentioned that $f(\pi_i|Y_i)$ is a beta distribution -- let's show that!

$$f(\pi_i|\mathbf{Y}) \propto f(Y_i|\pi_i)f(\pi_i)$$

- Plug in the densities (we'll drop any multiplicative constants that don't depend on π_i)

$$f(\pi_i|\mathbf{Y}) \propto \left[\pi_i^{Y_i} (1 - \pi_i)^{n_i - Y_i} \right] \times \left[\pi_i^{\alpha_0 - 1} (1 - \pi_i)^{\beta_0 - 1} \right]$$

- Adding the exponents

$$f(\pi_i|\mathbf{Y}) \propto \pi_i^{Y_i + \alpha_0 - 1} (1 - \pi_i)^{n_i - Y_i + \beta_0 - 1}$$

- And we can recognize (lol) this as the **kernel** of the beta distribution with parameters $\alpha = Y_i + \alpha_0$ and $\beta = n_i - Y_i + \beta_0$ (see [here](#)).

Calculating the posterior density

- The posterior distribution encodes updated plausibilities (or beliefs) for all parameter values conditioned on the data.
- Essentially, we really are just applying Bayes formula.
- The nice thing about working with conjugate priors is since we know the form of the posterior distribution the denominator will go to 1, since we're just integrating a PDF over its whole parameter space.
- Importantly, it is not always possible to compute the posterior analytically unless we constrain our prior to special forms (i.e., conjugate priors) that are easy to do some algebra with.
- But bear in mind that in many of the interesting models where we will really want to apply Bayes we will need to approximate the posterior using computational techniques such as Markov Chain Monte Carlo.
- The example we're working with so far is just one of the cases where we can solve for the posterior analytically.

Sampling to summarize

- In the theoretical world (when the posterior has a closed mathematical form), answering questions about point estimates and uncertainty intervals implies calculating yucky integrals BUT practically all we really have to do is basic summary statistics on samples from the posterior
- Once we have a posterior distribution, we typically will report **summaries** in the style of our typical frequentist point and interval estimates.
 - Note, however, that these have a different interpretation in the Bayesian framework.
- **Point** summaries
 - *Posterior Mean*: $\hat{\theta} = E[\theta|\mathbf{Y}]$
 - *Posterior Mode*: $\hat{\theta} = \arg \max_{\theta} p(\theta|\mathbf{Y})$
- **Credible interval**: A 95% credible interval (l_{95}, h_{95}) is a range of values that contains 95% of the posterior density

$$\int_{l_{95}}^{h_{95}} f(\theta|\mathbf{Y})d\theta = .95$$

Point estimates in a Bayesian setting

- The idea of point estimation in Bayesian settings is to summarize the posterior with a single value
- The three most common options are:
 - mode: which is the value with the highest posterior probability, also known as the maximum a posteriori (MAP) estimate
 - mean: just take the average of the samples from the posterior
 - median: just take the median of the samples from the posterior
- For many of the parameters that you will be typically estimating via Bayes each of these three options will yield very similar answers

Credible vs confidence intervals

- Credible intervals resemble very much the confidence intervals we saw in OLS and MLE
- The interpretations are very different though
 - A confidence interval is a region that after infinitely repeating the data sampling experiment will contain the true parameter with a certain frequency
 - A credible interval instead is simply a range of values that we believe our parameter can take with a certain probability

Quantities of interest

- There is no one unique credible interval! As a result, there are a few common choices for how to construct a credible interval
 - **Highest Density** interval (HDI) - no values outside of the interval have higher density than values *inside* the interval
 - **Equal-tailed** interval (ETI) - the probability of being below the lower limit is equal to the probability above the upper limit

Application

- Let's take a look at the elections data.

```
elections <- read_csv("us-house-wide.csv")
```

- One feature of this county-level House data is that because House districts don't perfectly overlap counties, we have some county-district combinations with very few voters!

```
allegan <- elections %>% filter(county == "Allegan")  
print(allegan)
```

```
## # A tibble: 2 × 10  
##   state county  fipscode fipscode2 office district total.votes  dem  rep  
##   <chr> <chr>    <chr>    <chr>    <chr>   <chr>      <dbl> <dbl> <dbl>  
## 1 MI     Allegan 26005     26005000... US Ho... 2          945    426    506  
## 2 MI     Allegan 26005     26005000... US Ho... 6         48295  17654  28257
```

- MI-2 only has 945 votes from Allegan County.

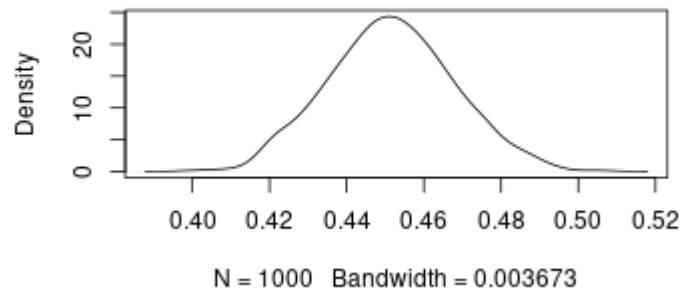
Application

- Let's get the posterior distribution for π_i for MI-2 in Allegan County under an uninformative uniform prior

```
mi2_allegan <- elections %>% filter(state=="MI"&county == "Allegan"&c
posterior_alpha <- mi2_allegan$dem + 1
posterior_beta <- mi2_allegan$total.votes - mi2_allegan$dem + 1

#
plot(density(rbeta(1000, posterior_alpha, posterior_beta)))
```

density(x = rbeta(1000, posterior_alpha, posterior_b



```
# Plot the posterior
mi2_posterior <- ggplot() + xlim(.3, .5) + geom_function(fun=dbeta,
```

Application

- We know the form of the beta mean, so our posterior mean estimate is

```
posterior_mean <- posterior_alpha/(posterior_alpha + posterior_beta)
posterior_mean
```

```
## [1] 0.451
```

- Note this corresponds to the MLE:

```
mi2_allegan$dem/mi2_allegan$total.votes
```

```
## [1] 0.451
```

- And we can obtain an equal-tailed 95% credible interval simply via the quantile function

```
posterior_ci <- c(qbeta(.025, shape1=posterior_alpha, shape2=posterior_beta),
                  qbeta(.975, shape1=posterior_alpha, shape2=posterior_beta))
posterior_ci
```

```
## [1] 0.419 0.483
```

Application

- But we have some more information from the other counties in the district

```
mi2 <- elections %>% filter(state=="MI" & district == 2)
mi2
```

```
## # A tibble: 8 × 10
##   state county fipscode fipscode2 office district total.votes  dem  rep
##   <chr> <chr>   <chr>   <chr>   <chr>   <chr>      <dbl> <dbl> <dbl>
## 1 MI     Allegan 26005   26005000... US Ho... 2          945    426    506
## 2 MI     Kent    26081   26081000... US Ho... 2        65089  31822  32182
## 3 MI     Lake    26085   26085000... US Ho... 2        4620   1767   2730
## 4 MI     Mason    26105   26105000... US Ho... 2        9880   4143   5514
## 5 MI     Muskego 26121   26121000... US Ho... 2       67627  35685  30603
## 6 MI     Newaygo 26123   26123000... US Ho... 2       20126   6798  12763
## 7 MI     Oceana   26127   26127000... US Ho... 2       10651   4205   6218
## 8 MI     Ottawa  26139   26139000... US Ho... 2      126525  46408  78454
```

- What if we instead constructed our prior such that its mean was centered on the average of all the other counties?
 - We can control the strength of the prior via the prior variance

Application

```
prior_mean <- sum(mi2 %>% filter(county != "Allegan") %>% pull(dem)),  
# factor of 1000 is arbitrary to make the distribution wide  
prior_variance <- (prior_mean*(1-prior_mean))/1000  
# Convert these to alpha/beta:  
# meaning we are computing the hyperparameters based on a  
# desired mean and variance  
prior_alpha <- (((1 - prior_mean)/prior_variance) - (1/prior_mean))*p  
prior_beta <- prior_alpha*(1/prior_mean - 1)
```

Application

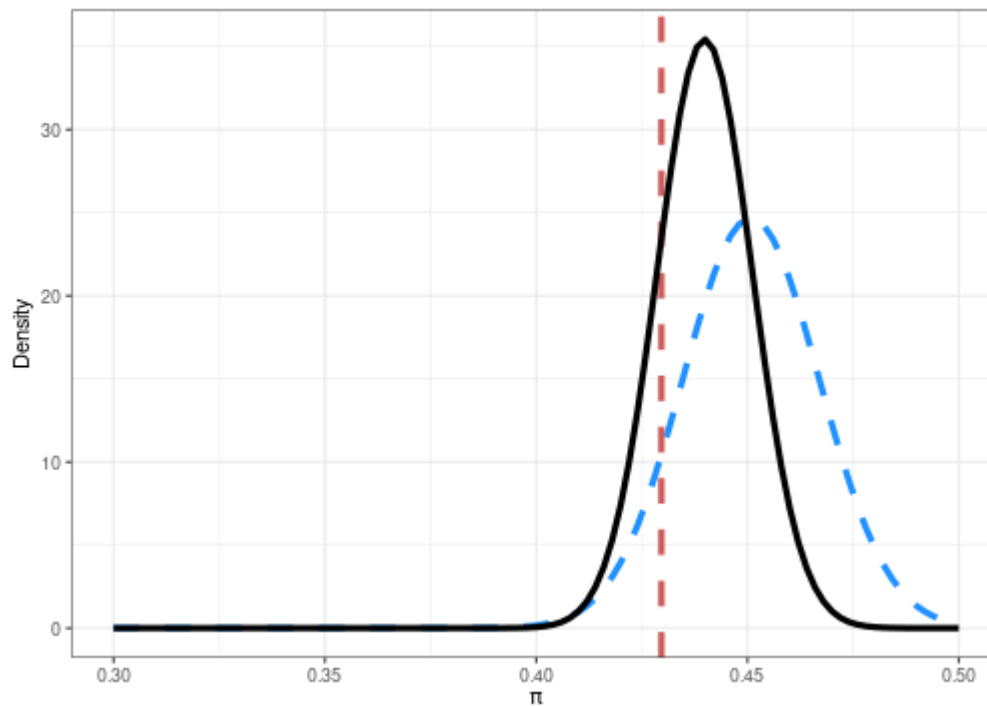
- Let's plug-in the new prior parameters

```
posterior2_alpha <- mi2_allegan$dem + prior_alpha
posterior2_beta <- mi2_allegan$total.votes - mi2_allegan$dem + prior_alpha
# Plot the posterior
mi2_posterior2 <- ggplot() + xlim(.3, .5) + geom_function(fun=dbeta,
  theme_bw() + xlab(expression(pi)) +
  ylab("Density") + geom_vline(xintercept=prior_mean, lty=2, lwd=1.5) +
  geom_function(fun=dbeta, args=list(shape1=posterior2_alpha, shape2=
```

Application

- Let's plug-in the new prior parameters

```
mi2_posterior2
```



Application

- Our new posterior mean

```
posterior2_mean <- posterior2_alpha/(posterior2_alpha + posterior2_beta)
posterior2_mean
```

```
## [1] 0.44
```

- And 95% credible interval

```
posterior2_ci <- c(qbeta(.025, shape1=posterior2_alpha, shape2=posterior2_beta),
                  qbeta(.975, shape1=posterior2_alpha, shape2=posterior2_beta))
posterior2_ci
```

```
## [1] 0.418 0.462
```


Application

- The posterior mean can be thought of as a "weighted average" of the prior mean and the MLE
 - The weights are controlled by the variance of the prior.
 - Can interpret the hyper-parameters for this case - α_0 and β_0 - as the number of "previously observed" counts.
- Stronger priors \rightsquigarrow narrower credible intervals
 - But it takes a *lot* more data to move the posterior distribution from the prior.
- Often the prior serves to **regularize** our estimates
 - We want our estimates to be "pulled" towards a particular value if there's very little data.
 - A common type of "regularizing" prior is designed to attenuate our estimates to 0 - we'll talk about this when we get to the last topic of the course!

Connections with Frequentism/MLE

- You'll notice that for the uninformative uniform prior, our posterior mode is equivalent to the MLE
 - If $f(\theta) \propto 1$, $f(\theta|\mathbf{Y}) = f(\mathbf{Y}|\theta)$
- More generally, for most well-behaved priors and likelihoods, the **Bernstein-von Mises** theorem states that as $n \rightarrow \infty$...
 - ...the posterior distribution $f(\theta|\mathbf{Y})$ will converge to a **normal** distribution
 - ...centered at the true parameter θ_0
 - ...with variance-covariance matrix equal to the inverse Fisher information
- Essentially: In large samples, posterior distributions converge to the sampling distribution of the MLEs

Markov-Chain Monte Carlo

Markov-Chain Monte Carlo (MCMC)

- In many settings, we have this problem in computing the posterior

$$f(\theta|\mathbf{Y}) = \frac{\overbrace{f(\mathbf{Y}|\theta)}^{\text{Easy!}} \times \overbrace{f(\theta)}^{\text{Easy!}}}{\underbrace{f(\mathbf{Y})}_{\text{HARD!}}}$$

- The **prior** $f(\theta)$ has a known distribution (sometimes up to a proportionality constant)
- The **likelihood** $f(\mathbf{Y}|\theta)$ has a known distribution specified by our data-generating process
- In general, we write our model so that the joint density of the data and the parameters $f(\mathbf{Y}, \theta) = f(\mathbf{Y}|\theta)f(\theta)$ factors very neatly

Markov-Chain Monte Carlo (MCMC)

- But $f(\mathbf{Y})$ is a tough to evaluate integral!

$$f(\mathbf{Y}) = \int f(\mathbf{Y}, \theta) d\theta$$

- We've used the trick of having a **conjugate prior** which lets us know the distribution of the posterior, in the application above we knew that the posterior followed a Beta

MCMC Benefits/Costs

- Benefits:
 - MCMC allows us to produce samples from the joint posterior without maximizing anything and we'll be able to sample directly from the posterior without assuming that it is coming from a specific distribution
 - MCMC is why people use Bayes for everything from multilevel/hierarchical modeling to text/network analysis
- Costs:
 - Cost of this power is that it may take much (much, much, much ...) longer for our estimation to complete

MCMC High level

- The essence of MCMC is to produce samples from the posterior using only the product of the likelihood and prior, which are always available to us since we're specifying them
- So by just evaluating the likelihood \times prior and without normalizing the denominator, MCMC is supposed to allow us to generate random representative values from the posterior distribution
- This is awesome because computing that evidence term (denominator of Bayes formula) is at times just not possible

MCMC ... a bit more formally

- MCMC methods allow us to generate a sample of observations from the target posterior **even when we don't know it** as long as we can evaluate a function that is **proportional** to the posterior
 - **Monte Carlo**: Use repeated samples to obtain numeric approximations of key quantities
 - **Markov-Chain**: The samples from the posterior are constructed via a "chain" that has the Markov property
- **Our Goal**: Generate a monte carlo sample $\{\theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \dots\}$ of arbitrary length such that the sequence of draws **converges** to a **stationary distribution** that is the target posterior
- This sample is a **markov chain** in that the distribution of the $(i + 1)$ th draw depends on the value of the i th, but only on that past value

$$f(\theta^{(i+1)} | \mathbf{Y}, \theta^{(i)}, \theta^{(i-1)}, \dots, \theta^{(1)}) = f(\theta^{(i+1)} | \mathbf{Y}, \theta^{(i)})$$

- Important note: our samples from the target posterior will be **dependent**

Metropolis Hastings Algorithm (a MCMC approach)

High Level I

- Suppose an elected politician lives on a long chain of islands.
- They are constantly traveling from island to island, wanting to stay in the public eye.
- At the end of a day they have to decide whether to:
 - stay on the current island
 - move to the adjacent island to the left
 - move to the adjacent island to the right
- Their goal is to visit all the islands **proportionally** to their **relative population**.
- But they don't know the total population of all the islands.
- They only know the population of the current island where they are located.
- They can also ask about the population of an adjacent island to which they plan to move.

Metropolis Hastings Algorithm (a MCMC approach)

High Level II

- Lets say this is a clever politician that paid attention in methods class and they decide to use the Metropolis algorithm as a heuristic for traveling across the islands (Metropolis et al 1953).
- First, they flip a coin to decide whether to propose the adjacent island to left or the adjacent island to the right.
- If the proposed island has a larger population than the current island ($P_{proposed} > P_{current}$), then they go to the proposed island.
- If the proposed island has a smaller population than the current island ($P_{proposed} < P_{current}$), then they go to the proposed island with probability $\frac{P_{proposed}}{P_{current}}$.
- In the long run, the probability that the politician is on any one of the islands exactly matches the relative population of the island!

Metropolis Hastings Algorithm (a MCMC approach)

High Level III

- Before talking about why this works lets reframe in teh context of our actual problem, which is to draw samples from an unknown posterior probability distribution
- The "islands" in our objective are parameter values θ (and they need not be discrete, but can instead take on a continuous range of values as usual)
- The "population sizes" in our objective are the posterior probabilities (or densities) at each parameter value: $f(\theta|data)$
- The "days" in our objective are samples taken from the posterior distribution.
- The coin flip represents the **proposal distribution**, $q(\theta)$
- The Metropolis algorithm will eventually give us a collection of samples from the posterior.
- We can then use these samples just the same as if we were sampling directly from a posterior distribution.

Metropolis Hastings Algorithm (a MCMC approach)

High Level IV

- Now, let's try to understand why the algorithm works.
- We are going to denote our target probability density from which we want to draw samples as $p(\theta)$.
 - Bear in mind that $p(\theta)$ is usually a posterior density $f(\theta|d)$.
- Consider two adjacent positions and the probabilities of moving from one to the other.
- We'll see that the relative transition probabilities, between adjacent positions, exactly match the relative values of the target density $p(\theta)$.
- Extrapolate that result across all the positions, and you can see that, in the long run, each position will be visited proportionally to its target value.
- Suppose we are at position θ .

Metropolis Hastings Algorithm (a MCMC approach)

High Level V (eek)

- The probability of moving to $\theta + 1$, denoted $\mathbb{P}(\theta \rightarrow \theta + 1)$, is the probability of proposing that move times the probability of accepting it if proposed, which is:

$$\mathbb{P}(\theta \rightarrow \theta + 1) = 0.5 \times \min(p(\theta + 1)/p(\theta), 1)$$

- On the other hand, if we are presently at position $\theta + 1$, the probability of moving to θ is:

$$\mathbb{P}(\theta + 1 \rightarrow \theta) = 0.5 \times \min(p(\theta)/p(\theta + 1), 1)$$

Metropolis Hastings Algorithm (a MCMC approach)

High Level V (eek)

- The ratio of the transition probabilities is:

$$\begin{aligned}\frac{\mathbb{P}(\theta \rightarrow \theta + 1)}{\mathbb{P}(\theta + 1 \rightarrow \theta)} &= \frac{0.5 \times \min(p(\theta + 1)/p(\theta), 1)}{0.5 \times \min(p(\theta)/p(\theta + 1), 1)} \\ &= \begin{cases} \frac{1}{p(\theta)/p(\theta+1)} & \text{if } p(\theta + 1) > p(\theta) \\ \frac{p(\theta+1)/p(\theta)}{1} & \text{if } p(\theta + 1) < p(\theta) \end{cases} \\ &= \frac{p(\theta + 1)}{p(\theta)}\end{aligned}$$

Metropolis Hastings Algorithm (a MCMC approach)

High Level VI

- The equation on the previous slide tells us that during transitions back and forth between adjacent positions, the relative probability of the transitions exactly matches the relative values of the target distribution.
- That might be enough to get the intuition that, in the long run, adjacent positions will be visited proportionally to their relative values in the target distribution.
- If that's true for adjacent positions, then, by extrapolating from one position to the next, it must be true for the whole range of positions.
- In more mathematical terms, this means that the transition probabilities form a Markov chain that has the target distribution as its equilibrium or stationary distribution.
- Hence, one can obtain a sample of the desired distribution by recording states from the chain.

Metropolis-Hastings Algorithm ... more formally I

- The foundational algorithm for generating MCMC samples is the **Metropolis-Hastings** algorithm.
 - **Crucially:** We can sample from almost *any* distribution (though some distributions are better than others)
- The algorithm relies on two key concepts to generate another sample $\theta^{(i+1)}$ given the past sample value $\theta^{(i)}$:
 - The **proposal distribution** - A probability distribution $f(\theta^{(i+1)}|\theta^{(i)})$ that generates our "proposal" value
 - The **acceptance probability** - A calculation for the probability of "accepting" the proposed value or rejecting it

Metropolis-Hastings Algorithm ... more formally II

- To generate the markov chain of M samples from the target posterior distribution $\{\theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \dots, \theta^{(M)}\}$, the **Metropolis-Hastings** algorithm iterates between two steps:
 1. **Proposal:** Conditional on the current value θ , generate a draw θ^* from $Q(\theta^*|\theta)$ where Q is the proposal distribution.
 2. **Accept/Reject:** With probability α , "accept" the proposal and set $\theta^{(i+1)} = \theta^*$ - otherwise "reject" and set $\theta^{(i+1)} = \theta$

$$\alpha = \min \left\{ 1, \frac{f(\mathbf{Y}|\theta^*)f(\theta^*)}{f(\mathbf{Y}|\theta)f(\theta)} \times \frac{Q(\theta|\theta^*)}{Q(\theta^*|\theta)} \right\}$$

- **Intuitively** - If the proposal distribution is symmetric...
 - ...if the unnormalized posterior density is higher at the proposed rather than the current location, **always accept**
 - ...if the unnormalized posterior density is lower at the proposed rather than current location, **maybe accept**

Proposal Distribution

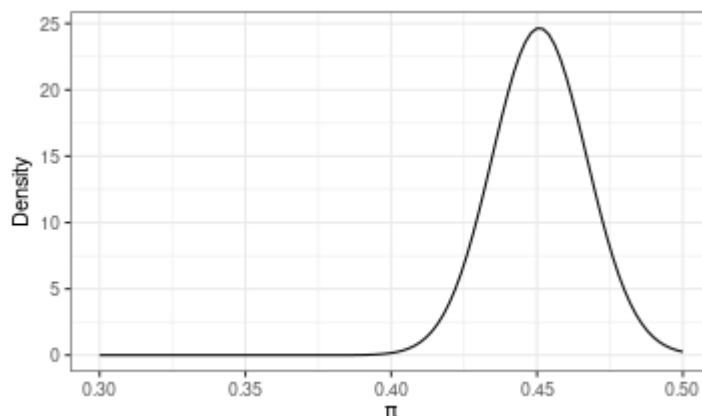
- Our choice of Q governs how quickly the Markov Chain will "converge" to the true posterior
 - We want to choose a proposal distribution that could "eventually" propose each value in the domain of θ
 - We also want a proposal distribution that generates a high acceptance probability α
- It is common to choose a proposal distribution that is **symmetric**:

$$\frac{Q(\theta|\theta^*)}{Q(\theta^*|\theta)} = 1$$

- **Symmetric** here just means that the suggestion for moving from one point to another point is the same as the suggestion for moving in the opposite direction.
- For $\theta \in (-\infty, \infty)$, common to propose θ^* from a $\text{Normal}(\theta, \Omega)$
 - That is, from a normal distribution centered on the "current" parameter with an arbitrarily chosen variance Ω
 - This is **symmetric** since the chances of proposing θ^* from θ are the same as the chances of proposing θ from θ^*

Back to the application

- Let's go back to our posterior distribution for the Democratic vote share in the part of MI-2 that is in Allegan County



- Previously, we calculated the posterior directly since it was beta, but suppose we couldn't?

Back to the application

- Let's construct our MCMC sampler. First, let's choose a proposal distribution
 - For $\pi_i \in (0, 1)$, we could use a Beta, but the Beta isn't symmetric
 - $\text{Uniform}(0, 1)$ would generate a lot of bad proposals
- Instead, let's transform π_i to be unbounded. Define $\theta_i = \log \left(\frac{\pi_i}{1-\pi_i} \right)$
 - We'll sample θ s and convert them back to π when we evaluate the likelihood and prior
 - An intuitive distribution $Q(\theta^*|\theta)$ would be the standard logistic distribution centered on θ . It's **symmetric**!

Back to the application ... this gets dirty

```
set.seed(6886)
logistic <- function(x) 1/(1 + exp(-x)) # Helper Function
logit <- function(x) log(x/(1-x))
M <- 100000 # Number of MCMC samples
pi_mcmc <- rep(NA, M) # Vector to store our samples
pi_mcmc[1] <- .5 # Pick a starting value
```

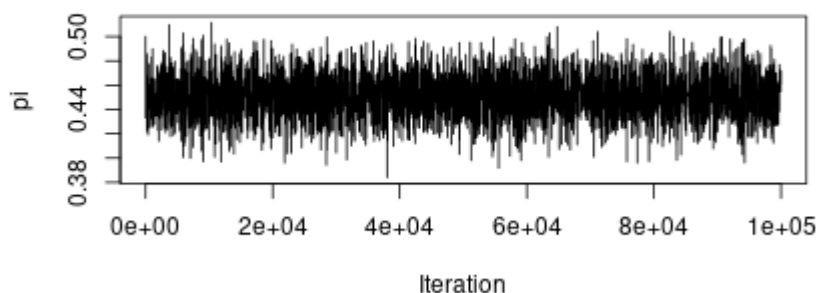
Back to the application ... this gets dirty

```
for (i in 1:(M-1)){ # For i in 1:(M-1)
  ## Step 1 - Proposal
  theta_i <- logit(pi_mcmc[i])
  theta_star <- rlogis(1, location = theta_i)
  ## Step 2 - Accept/Reject
  lik_star <- dbinom(mi2_allegan$dem, mi2_allegan$total.votes, prob =
  prior_star <- dbeta(logistic(theta_star), shape1 = 1, shape2=1)
  lik_current <- dbinom(mi2_allegan$dem, mi2_allegan$total.votes, pr
  prior_current <- dbeta(logistic(theta_i), shape1 = 1, shape2=1)
  Q_star_current <- dlogis(theta_star, location=theta_i) # This is a
  Q_current_star <- dlogis(theta_i, location=theta_star)
  #Assemble the acceptance ratio
  ar <- ((lik_star*prior_star)/((lik_current*prior_current))*(Q_curren
  # Choose to accept or reject
  accept <- rbinom(1,1,min(1,ar))
  # Store the next value
  pi_mcmc[i+1] <- logistic(theta_star)*accept + logistic(theta_i)*(1-
}
```

Back to the application ... evaluating the results

- Let's first see how the chain progresses by generating the **trace plot**

```
plot(y=pi_mcmc, x=1:length(pi_mcmc), xlab="Iteration", ylab="pi", type="l")
```

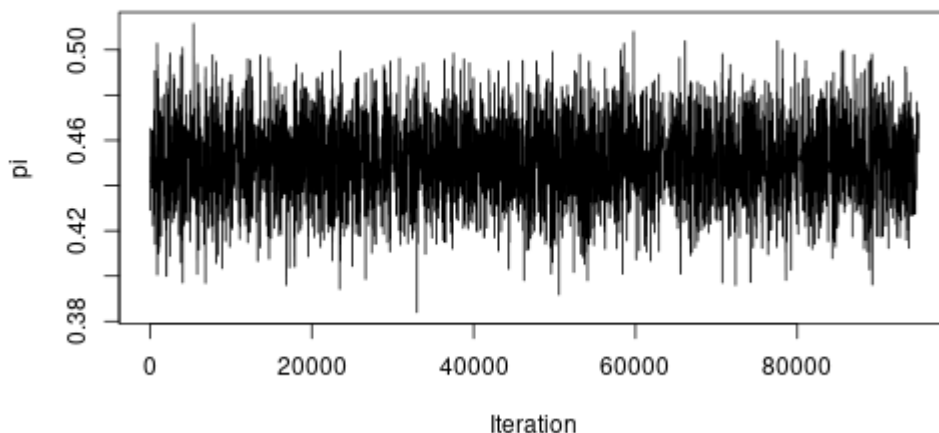


- You'll see that it takes a bit for the chain to reach the area of the posterior with the most mass
 - If we started closer to the "mass" of the distribution, we would have converged faster
- In practice, we will throw away some number of our initial samples as a **burn-in** period, since we know that the chain needs some time to reach the stationary distribution.
 - Here we'll drop the first 5000 samples and keep the rest

Back to the application

- Our ideal trace plots have no clear trends - in such a case, we consider the chain to have reached its stationary distribution

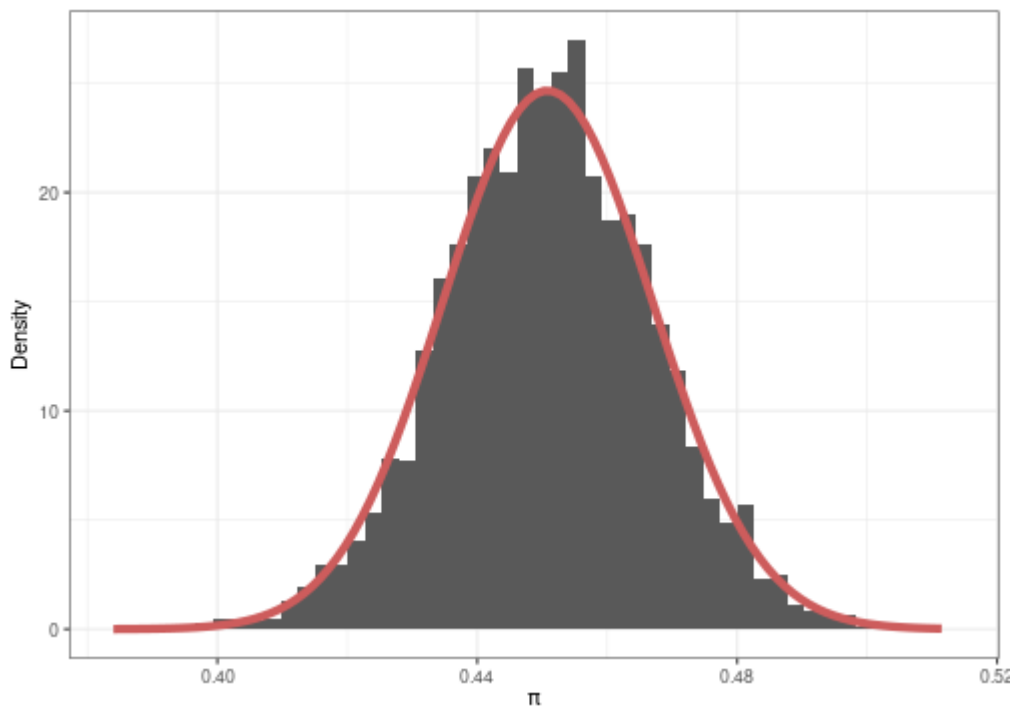
```
pi_mcmc_use <- pi_mcmc[5000:M] # Toss our burn-in period  
plot(y=pi_mcmc_use, x=1:length(pi_mcmc_use), xlab="Iteration", ylab=''
```



- Note that trace plots are **diagnostics** -- they don't prove that the chain has "mixed," but a lot of jumps or stagnant periods would suggest that our chain needs to run longer

Back to the application

```
# Plot the histogram on top of the "true" distribution  
mi2_mcmc <- data.frame(x=pi_mcmc_use) %>% ggplot(aes(x=x)) + theme_k  
  ylab("Density")  
mi2_mcmc
```



Back to the application

- We can compute summaries like the posterior mean or a 95% credible interval by taking summaries of the MCMC sample

```
posterior_mcmc_mean <- mean(pi_mcmc_use)
posterior_mcmc_mean
```

```
## [1] 0.451
```

```
posterior_mean
```

```
## [1] 0.451
```

Back to the application

- We get very close to the true posterior (and can get closer with more iterations)

```
posterior_mcmc_ci <- quantile(pi_mcmc_use, c(.025, .975))  
posterior_mcmc_ci
```

```
## 2.5% 97.5%
```

```
## 0.418 0.482
```

```
posterior_ci
```

```
## [1] 0.419 0.483
```

Multiple parameters

- In this example, we've focused on estimating a single parameter. Suppose instead we have K parameters: $\theta = \{\theta_1, \theta_2, \theta_3, \dots, \theta_K\}$
- With multiple parameters, we'd need a multivariate proposal distribution (like a multivariate normal)
 - But this can be hard and mixing is often very slow with multivariate proposals.
- Instead, we can use **Gibbs sampling** -- generating each update $\theta_k^{(i+1)}$ conditional on the other current values of the other parameters $\theta_{-k}^{(i)}$
 - Importantly, if we **can** sample from the conditional distribution $\theta_k | \theta_{k-1}, \mathbf{Y}$ we can get proposals with acceptance probability 1
 - Even if we can't, we can use a M-H step for that parameter

Gibbs Sampling

- The **Gibbs sampling** algorithm generates updates

$\theta^{(i+1)} = \{\theta_1^{(i+1)}, \theta_2^{(i+1)}, \theta_3^{(i+1)}, \dots, \theta_K^{(i+1)}\}$ by sampling in sequence:

$$\theta_1^{(i+1)} := \theta_1^* \sim f(\theta_1 | \mathbf{Y}, \theta_2^{(i)}, \theta_3^{(i)}, \dots, \theta_K^{(i)})$$

$$\theta_2^{(i+1)} := \theta_2^* \sim f(\theta_2 | \mathbf{Y}, \theta_1^{(i+1)}, \theta_3^{(i)}, \dots, \theta_K^{(i)})$$

\vdots

$$\theta_k^{(i+1)} := \theta_k^* \sim f(\theta_k | \mathbf{Y}, \theta_1^{(i+1)}, \theta_2^{(i+1)}, \dots, \theta_{k-1}^{(i+1)}, \theta_{k+1}^{(i)}, \dots, \theta_K^{(i)})$$

\vdots

$$\theta_K^{(i+1)} := \theta_K^* \sim f(\theta_K | \mathbf{Y}, \theta_1^{(i+1)}, \theta_2^{(i+1)}, \dots, \theta_{K-1}^{(i+1)})$$

- **Important:** Sampling from the conditionals $f(\theta_k | \theta_{-k}, \mathbf{Y})$ is often easier to do than sampling from the marginal distribution $f(\theta_k | \mathbf{Y})$.
 - In fact, we'll often introduce latent variables to make a Gibbs sampling algorithm work where it otherwise wouldn't. This is called **data augmentation**

Gibbs Sampling

- To see why sampling from the conditionals works, consider the implied MH acceptance probability when we treat the conditional distribution $f(\theta_k | \theta_{-k}^{(i)}, \mathbf{Y})$ as the proposal distribution Q

$$\alpha = \frac{f(\mathbf{Y} | \theta_k^*, \theta_{-k}^{(i)}) f(\theta_k^*, \theta_{-k}^{(i)})}{f(\mathbf{Y} | \theta_k^{(i)}, \theta_{-k}^{(i)}) f(\theta_k^{(i)}, \theta_{-k}^{(i)})} \times \frac{f(\theta_k^{(i)} | \mathbf{Y}, \theta_{-k}^{(i)})}{f(\theta_k^* | \mathbf{Y}, \theta_{-k}^{(i)})}$$

- Rewrite the numerator/denominator in terms of the posterior and the data (and recall that $f(\mathbf{Y})$ cancels)

$$\alpha = \frac{f(\theta_k^*, \theta_{-k}^{(i)} | \mathbf{Y})}{f(\theta_k^{(i)}, \theta_{-k}^{(i)} | \mathbf{Y})} \times \frac{f(\theta_k^{(i)} | \mathbf{Y}, \theta_{-k}^{(i)})}{f(\theta_k^* | \mathbf{Y}, \theta_{-k}^{(i)})}$$

Gibbs Sampling

- Factor the posteriors

$$\alpha = \frac{f(\theta_k^* | \theta_{-k}^{(i)}, \mathbf{Y}) \times f(\theta_{-k}^{(i)} | \mathbf{Y})}{f(\theta_k^{(i)} | \theta_{-k}^{(i)}, \mathbf{Y}) \times f(\theta_{-k}^{(i)} | \mathbf{Y})} \times \frac{f(\theta_k^{(i)} | \mathbf{Y}, \theta_{-k}^{(i)})}{f(\theta_k^* | \mathbf{Y}, \theta_{-k}^{(i)})}$$

- Everything cancels so that $\alpha = 1$
 - So Gibbs sampling is a special case of Metropolis-Hastings where the proposal distribution choice **guarantees** acceptance.

Bayesian Regression

Application 2: Predicting Elections

- Now suppose that instead of observing counts, we look at the democratic party vote share in county i in the 2018 House elections (imagine we summed over all of the votes in all of the districts in that county).
 - Let Y_i denote the share of votes going to the democratic party candidate in 2018 in county i .
- We want to generate a predictive model based on X_i , (here, we'll just be using the democratic presidential vote share in 2016).

Bayesian Normal Model

- The conventional "normal" Bayesian regression model assumes

$$Y_i | X_i, \beta, \sigma^2 \sim \text{Normal}(X_i' \beta, \sigma^2)$$

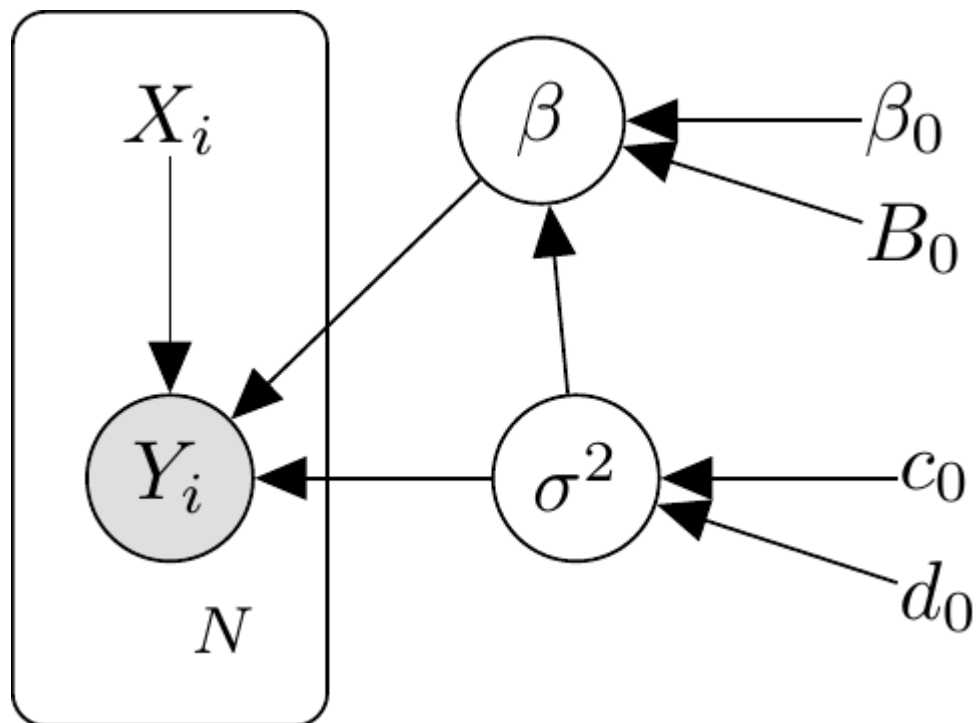
$$\beta | \sigma^2 \sim \text{Normal}(\beta_0, \sigma^2 B_0^{-1})$$

$$\sigma^2 \sim \text{Inverse-Gamma}(\frac{c_0}{2}, \frac{d_0}{2})$$

- In this case, β_0 , B_0 , c_0 and d_0 are all hyper-parameters
- This is referred to as the "Normal-Inverse-Gamma" model as the joint prior distribution on β, σ^2 is Normal-Inverse Gamma and it happens to also be the **conjugate prior** for the case with unknown β and σ^2
 - So the posterior is known in closed form to be **Normal-Inverse-Gamma**
 - And conditional on σ^2 , $\beta | \mathbf{X}, \mathbf{Y}, \sigma^2$ is also normally distributed!
- A common simplification is to make β and σ^2 marginally independent

$$\beta \sim \text{Normal}(\beta_0, B_0^{-1})$$

Plate notation



Application: Predicting Elections

```
# Aggregate the house data to counties
elections_county <- elections %>%
  group_by(fipscode) %>%
  summarize(
    state=state[1],
    county=county[1],
    total.votes = sum(total.votes),
    dem = sum(dem))

# Merge in 2015 Presidential
pres_2016 <- read_csv("clinton_2016_vote.csv")
elections_county <- elections_county %>%
  left_join(
    pres_2016 %>% dplyr::select(county_fips, candidatevotes, totalvotes)
    by=c(fipscode="county_fips"))

# Generate vote shares
elections_county$dem2018 <- elections_county$dem/elections_county$total.votes
elections_county$dem2016 <- elections_county$candidatevotes/elections_county$total.votes

# Drop missing
elections_county <- elections_county %>%
  filter(!is.na(dem2018)&!is.na(dem2016))
```

Metropolis-Hastings

- Set up the regression

```
X_mat <- model.matrix(dem2018 ~ dem2016, data=elections_county)
Y <- elections_county$dem2018
K <- ncol(X_mat) # Number of beta parameters
```

- Set up a diffuse prior

```
beta_0 <- rep(0, K)
B_inv_0 <- solve(diag(rep(1/9, K)))
c_0 = 0.001
d_0 = 0.001
```

- Set up the MCMC

```
M <- 40000 # Number of MCMC samples
burnin <- 5000
beta_mcmc <- matrix(nrow = M, ncol=K) # Vector to store our samples
beta_mcmc[1,] <- c(0,1) # Pick a starting value
sigma_mcmc <- rep(NA, M)
sigma_mcmc[1] <- 1
```

Metropolis-Hastings

- Write some functions to evaluate the likelihood and priors

```
log_lik_norm <- function(b, sigma, Y, X){  
  linpred <- X%*%b  
  sum(dnorm(Y, mean=linpred, sd=sigma, log=T))  
}
```

Metropolis-within-Gibbs Algorithm Code

Doesn't fit on the slide :(

```
set.seed(6886)
for (i in 1:(M-1)){ # For i in 1:(M-1)
  ## Beta
  ## Step 1 - Proposal
  beta_star <- as.vector(mvtnorm::rmvnorm(1, mean = beta_mcmc[i,],
  ## Step 2 - Accept/Reject
  lik_star_beta <- log_lik_norm(beta_star, sigma_mcmc[i], Y, X_mat)
  lik_current_beta <- log_lik_norm(beta_mcmc[i,], sigma_mcmc[i], Y,

  prior_star_beta <- mvtnorm::dmvnorm(beta_star, mean = beta_0, sig
  prior_current_beta <- mvtnorm::dmvnorm(beta_mcmc[i,], mean = beta

  ## Accept/reject
  ar_beta <- exp( lik_star_beta + prior_star_beta - lik_current_beta
  accept_beta <- rbinom(1,1,min(1,ar_beta))
  beta_mcmc[i+1,] <- beta_star*accept_beta + beta_mcmc[i,]*(1-accept

  ## Sigma
  ## Step 1 - Proposal
  sigma_log <- rnorm(1, mean = log(sigma_mcmc[i]), sd=.01)
  sigma_star <- exp(sigma_log)
```

Metropolis-within-Gibbs Algorithm Explanation I

- Metropolis-Hastings for Beta:
 - Generate a proposal for the vector of parameters β (β_{star}) from a multivariate normal distribution centered at the current value of β ($\beta_{\text{mcmc}[i,]}$).
 - Compute the likelihood of the proposed value lik_star_beta and the likelihood of the current value lik_current_beta .
 - Compute the prior for the proposed value prior_star_beta and the prior for the current value $\text{prior_current_beta}$.
 - Compute the acceptance ratio ar_beta and decides whether to accept the proposed value or stick with the current value.

Metropolis-within-Gibbs Algorithm Explanation II

- Metropolis-Hastings for Sigma:
 - Similarly, a proposal for the parameter sigma (σ_{star}) is generated from a log-normal distribution centered at the current value of sigma ($\sigma_{\text{mcmc}[i]}$).
 - Compute the likelihood and prior values for the proposed and current values of sigma.
 - An acceptance ratio ar_sigma is then calculated, and a decision is made to accept the proposed value or remain with the current one.

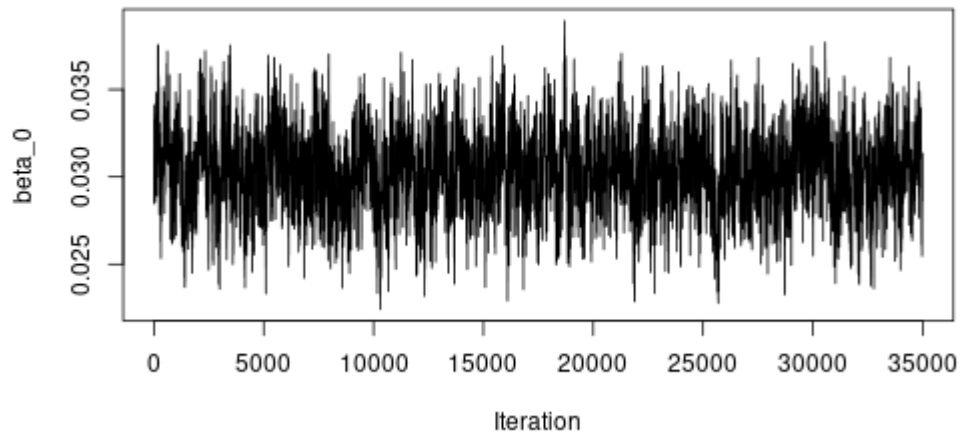
Metropolis-within-Gibbs Algorithm Explanation III

- The code alternates between updating the beta parameters and the sigma parameter.
- This is a Gibbs sampling structure (updating parameters one at a time or block by block).
- However, since we are using Metropolis-Hastings steps for each of the parameter sets instead of directly sampling from full conditionals, it's termed "Metropolis-within-Gibbs."

Convergence

- β_0

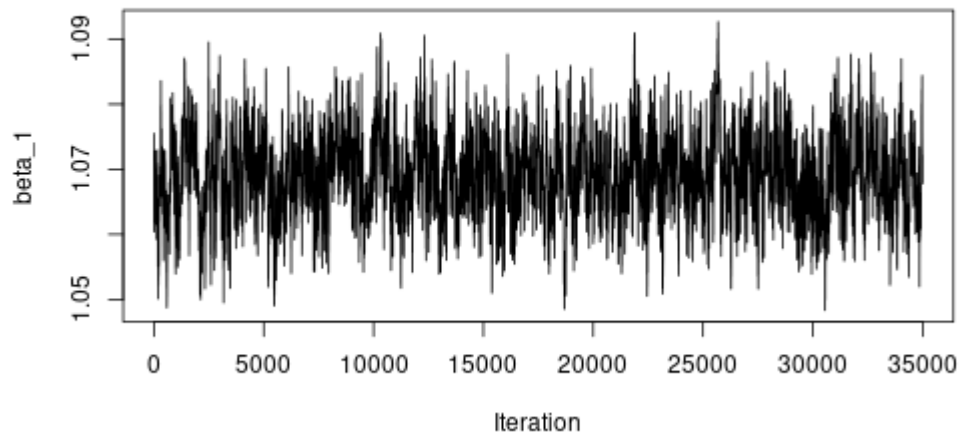
```
beta_mcmc_use <- beta_mcmc[burnin:M,] # Toss our burn-in period  
plot(y=beta_mcmc_use[,1], x=1:length(beta_mcmc_use[,1]), xlab="Iteration")
```



Convergence

- β_1

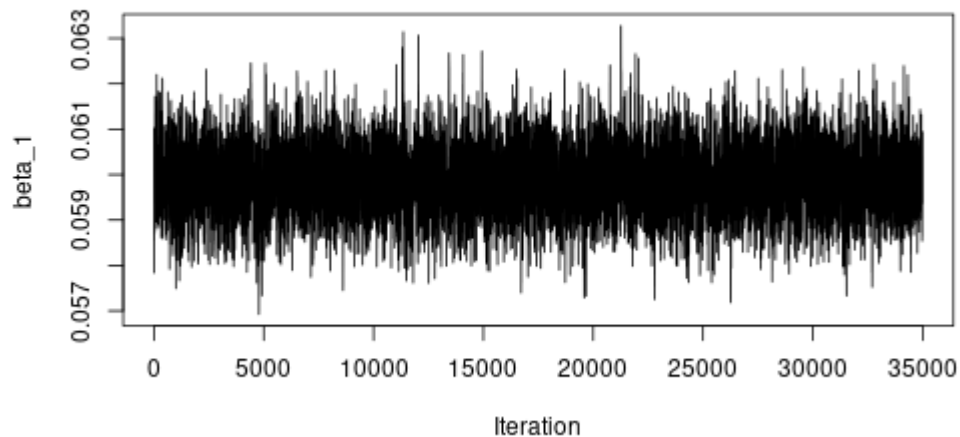
```
plot(y=beta_mcmc_use[,2], x=1:length(beta_mcmc_use[,2]), xlab="Iterati
```



Convergence

- σ

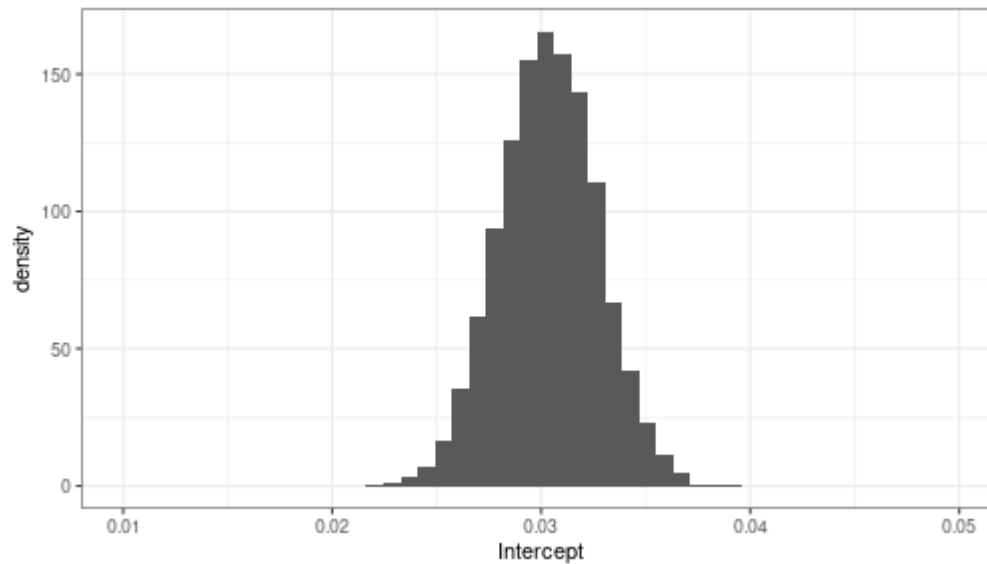
```
sigma_mcmc_use <- sigma_mcmc[burnin:M]  
plot(y=sigma_mcmc_use, x=1:length(sigma_mcmc_use), xlab="Iteration",
```



Summaries

β_0

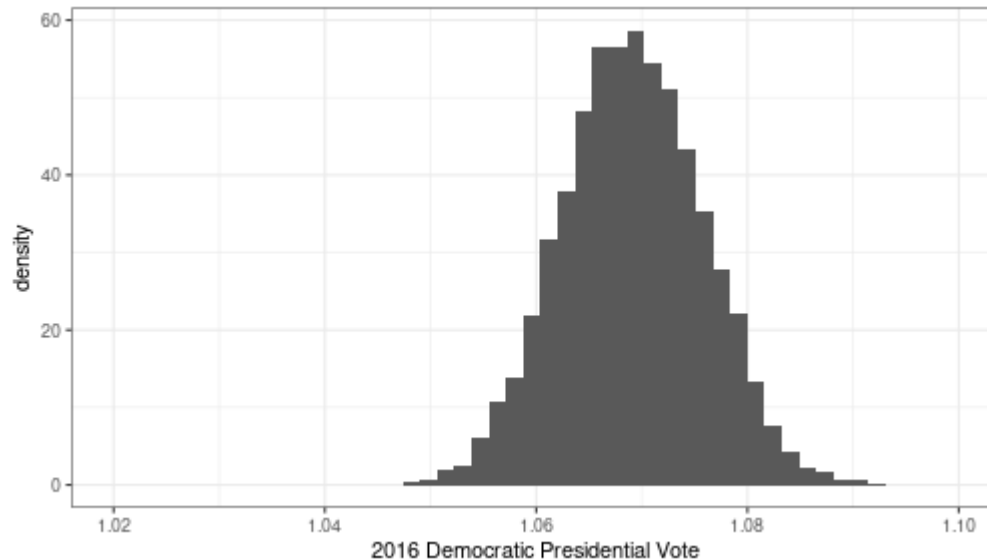
```
beta_mcmc_out <- as.data.frame(beta_mcmc_use)
colnames(beta_mcmc_out) <- c("Intercept", "dem2016")
beta_mcmc_out %>% ggplot(aes(x=Intercept)) + xlim(.01, .05) + theme_
```



Summaries

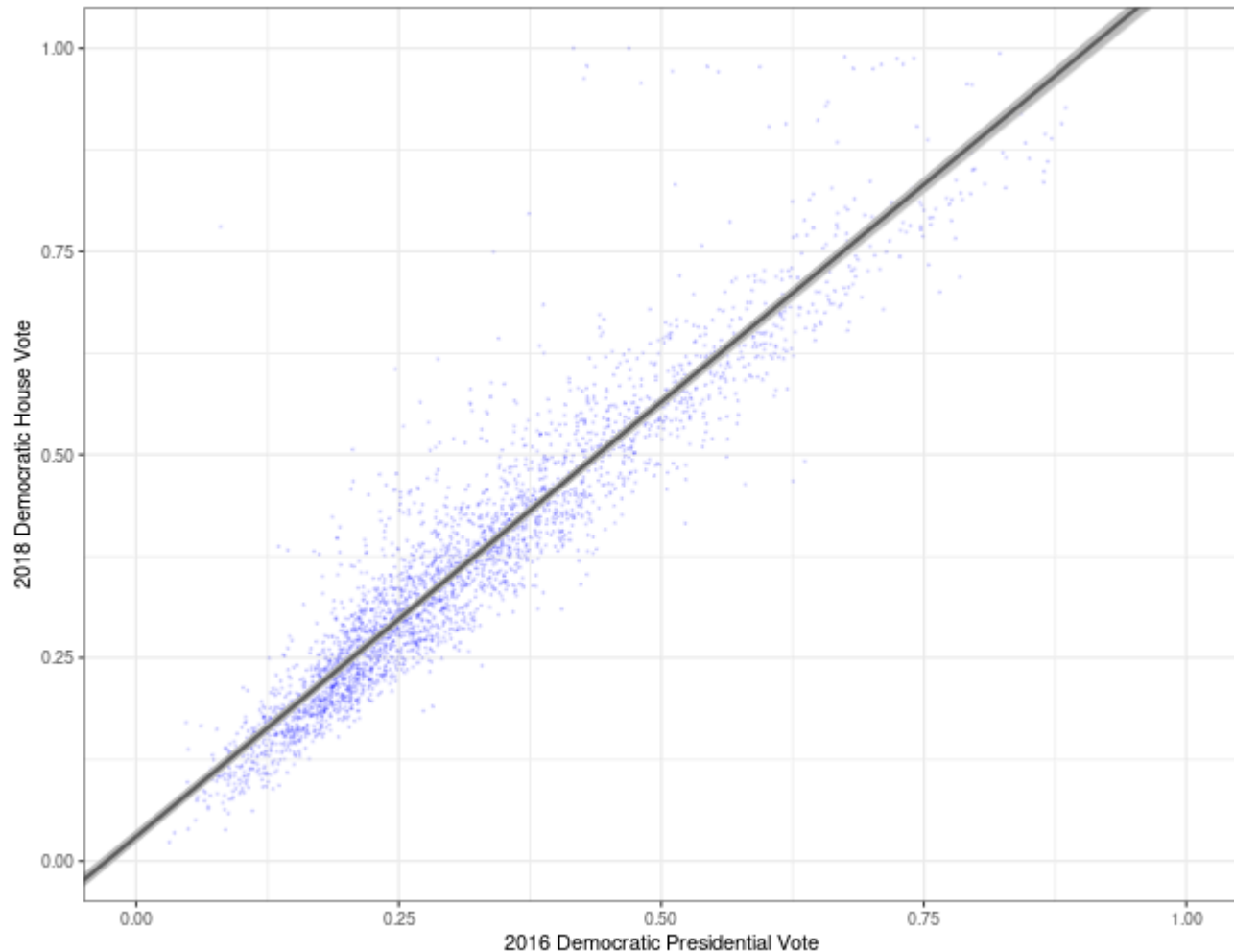
β_1

```
beta_mcmc_out %>% ggplot(aes(x=dem2016)) + theme_bw() + xlim(1.02, 1.10)
```



Summaries

- Overlay the data and regression



Summaries

- Posterior means and 95% credible intervals

```
summary_results <- data.frame(variable = c("Intercept", "dem2016"),  
                               ci95_lower = apply(beta_mcmc_out, 2, fun = ci95_lower),  
                               ci95_upper = apply(beta_mcmc_out, 2, fun = ci95_upper))
```

```
summary_results
```

##	variable	pm	ci95_lower	ci95_upper
##	Intercept	Intercept	0.0304	0.0258 0.035
##	dem2016	dem2016	1.0691	1.0562 1.082

brms

- Let's check against an existing Gibbs sampling implementation from brms

```
set.seed(6886)
library(brms)
brms_fit <- brm(
  formula=dem2018 ~ dem2016,
  data=elections_county,
  family=gaussian(),
  chains = 1,
  iter=M,
  warmup=burnin,
  seed=6886
)
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: 'gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0'
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/sminhas/R/x86_64-pc-linux-gnu-library/4.3/RcppEigen/include" -fopenmp -std=c++11 -DRCPP_EIGEN_USE_OPENMP -c
## In file included from /home/sminhas/R/x86_64-pc-linux-gnu-library/4.3/RcppEigen/include/Eigen/src/Core/Matrix.h:27:
##       from /home/sminhas/R/x86_64-pc-linux-gnu-library/4.3/RcppEigen/include/Eigen/Core:1:
##       from /home/sminhas/R/x86_64-pc-linux-gnu-library/4.3/RcppEigen/include/Eigen/StdVector:1:
##       from <command-line>:
## /home/sminhas/R/x86_64-pc-linux-gnu-library/4.3/RcppEigen/include/Eigen/src/Core/Matrix.h:27: fatal error: Eigen/SparseCore: No such file or directory
## 628 | namespace Eigen {
```

Gibbs sampler

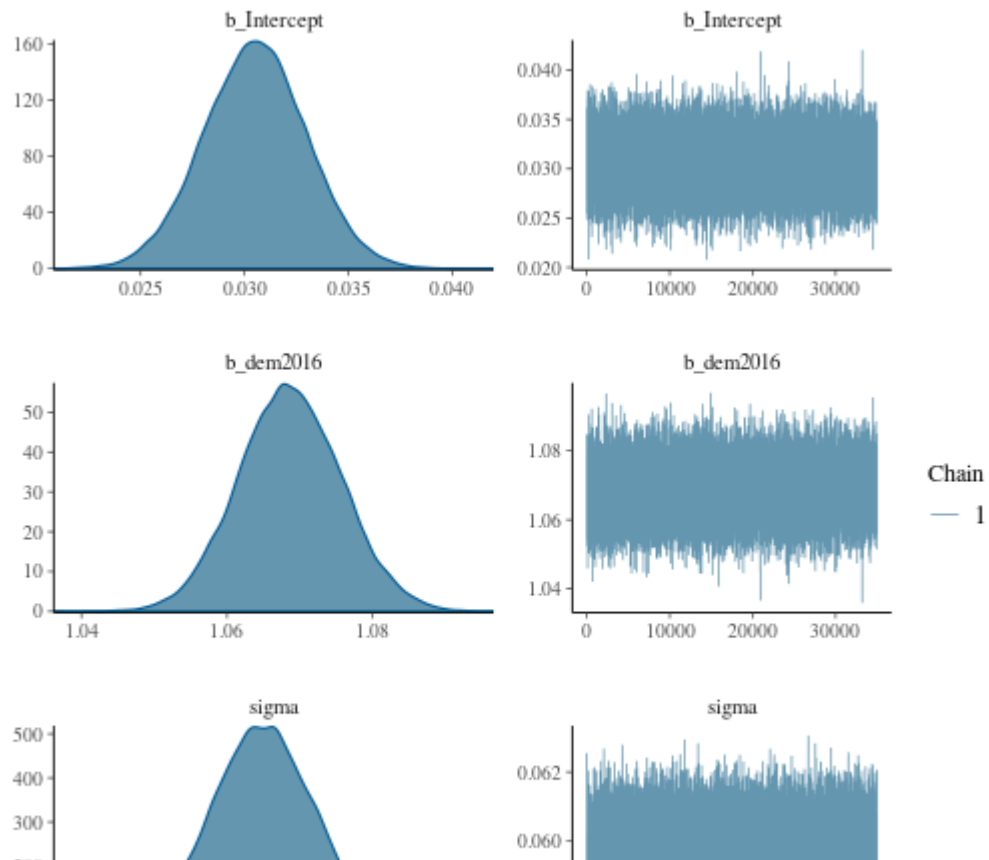
```
summary(brms_fit)
```

```
## Family: gaussian
## Links: mu = identity; sigma = identity
## Formula: dem2018 ~ dem2016
## Data: elections_county (Number of observations: 3061)
## Draws: 1 chains, each with iter = 40000; warmup = 5000; thin = 1;
## total post-warmup draws = 35000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept      0.03      0.00    0.03    0.04 1.00    27708    25872
## dem2016        1.07      0.01    1.06    1.08 1.00    25458    24355
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma          0.06      0.00    0.06    0.06 1.00    24494    24102
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Diagnostics

- Prettier trace plots (i.e., less visible dependence)!

```
plot(brms_fit)
```



Under the hood of brms

- brms is a front-end for an amazing piece of software called Stan
- Stan actually does not use a Metropolis and/or Gibbs algorithm but instead uses an approach called Hamiltonian Monte Carlo (HMC)
- HMC is another sampling method that is computationally costly but its proposals are much more efficient and so it doesn't need as many samples to describe a posterior ... as models become more complex HMC is the way to go

HMC I

- HMC is complex, but we can try to understand it at a high level
- Let's try to understand it in a very superficial way by using again the politician's tale.
- Suppose the politician has moved to the mainland now.
- Now, instead of moving over a set of discrete islands, it has to move through a continuous territory stretched out along a narrow valley, running north-south.
- The obligations are the same: to visit his citizens in proportion to their local density.
- And again, the politician doesn't know the population of each area in advance.

HMC II

- The strategy of the politician is the following:
- He drives his car across the narrow valley back and forth along its length.
- In order to spend more time in densely settled areas, he slows down his vehicle when houses grow more dense.
- Likewise, he speeds up when houses grow more sparse.
- This strategy requires knowing how quickly population density is changing, at their current location.
- But it doesn't require remembering where they've been or knowing the population distribution anywhere else.
- This story is analogous to how Hamiltonian Monte Carlo works.

HMC III

- In statistical applications, the politician's vehicle is the current vector of parameter values.
- HMC really does run a physics simulation, pretending the vector of parameters gives the position of a little frictionless particle.
- The log-posterior provides a surface for this particle to glide across.
- Then the job is to sweep across this surface, adjusting speed (momentum) in proportion to how high up we are.
- When the log-posterior is very flat, then the particle can glide for a long time before the slope (gradient) makes it turn around.
- When instead the log-posterior is very steep, then the particle doesn't get far before turning around.

Next time

- We'll practice more with brms
 - testing out the affect of different priors on the posterior
 - running models
 - interpreting results (brms works with `marginalEffects`)
 - making predictions using the posterior predictive
 - making tables (brms works with `modelsummary`)
 - talk about computational strategies for running things
 - and then we'll shift towards advanced applications such as multilevel modeling and measurement