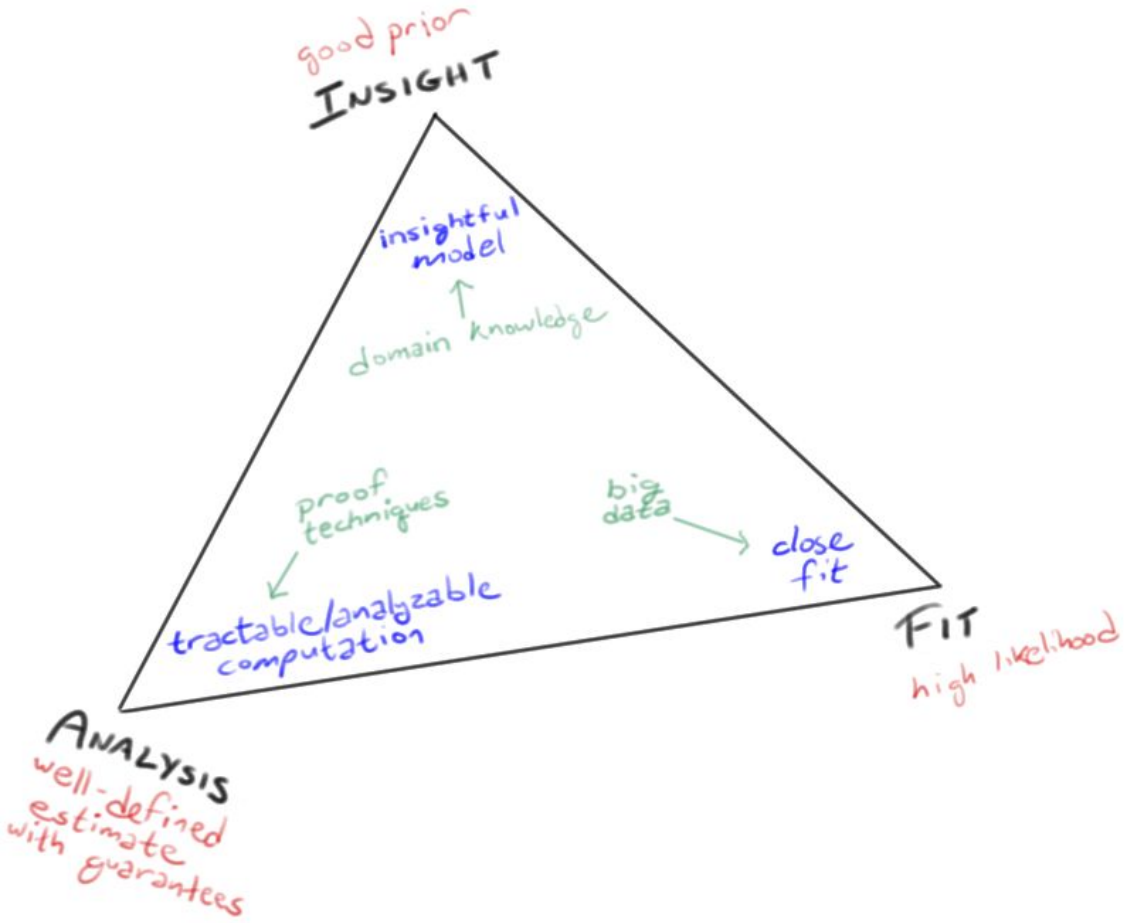
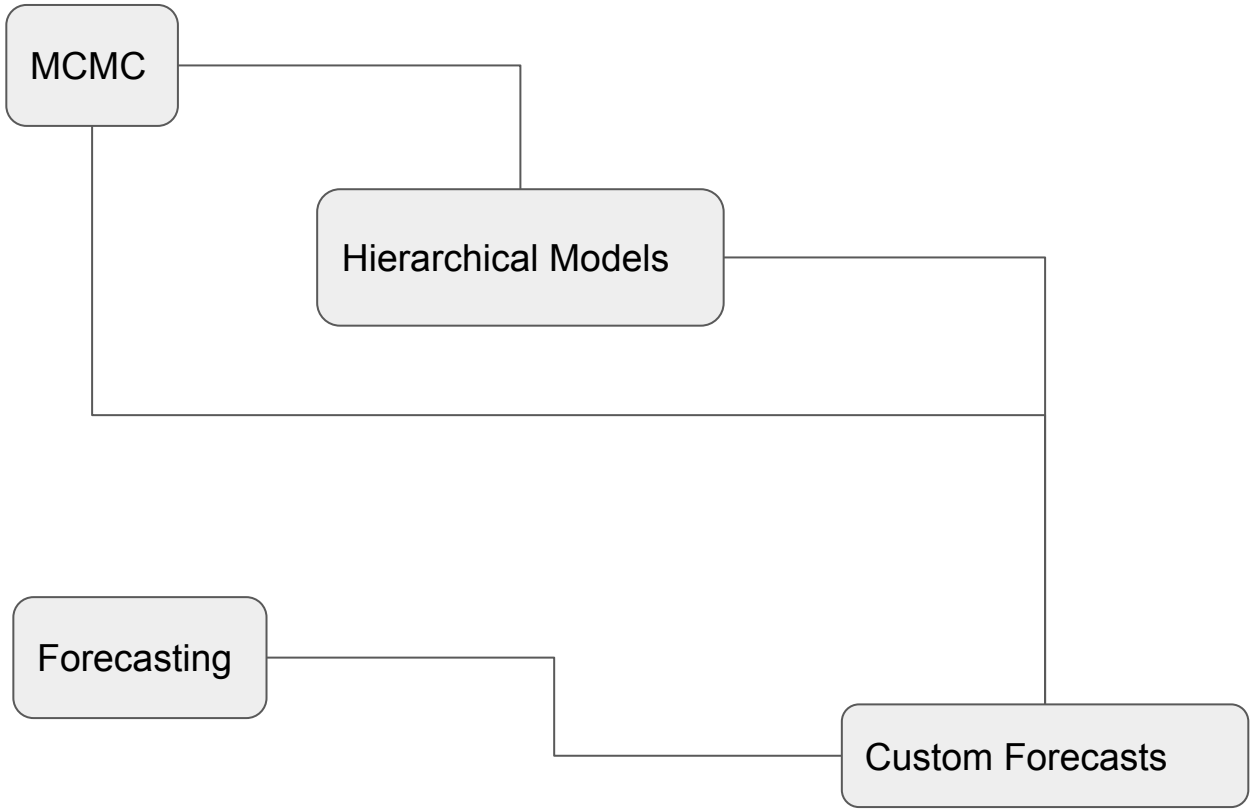
Applied Machine Learning

www.eanalytica.com/files/AML.pdf



https://www.cs.jhu.edu/~jason/tutorials/ml-simplex.html

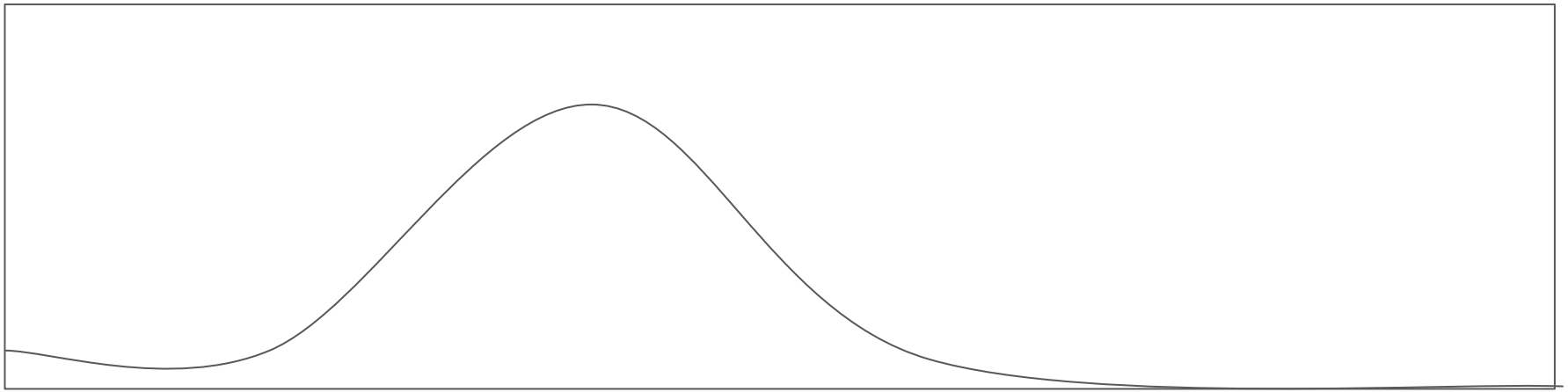

MCMC
Hierarchical Models
Forecasting
Custom Forecasts

Why MCMC?

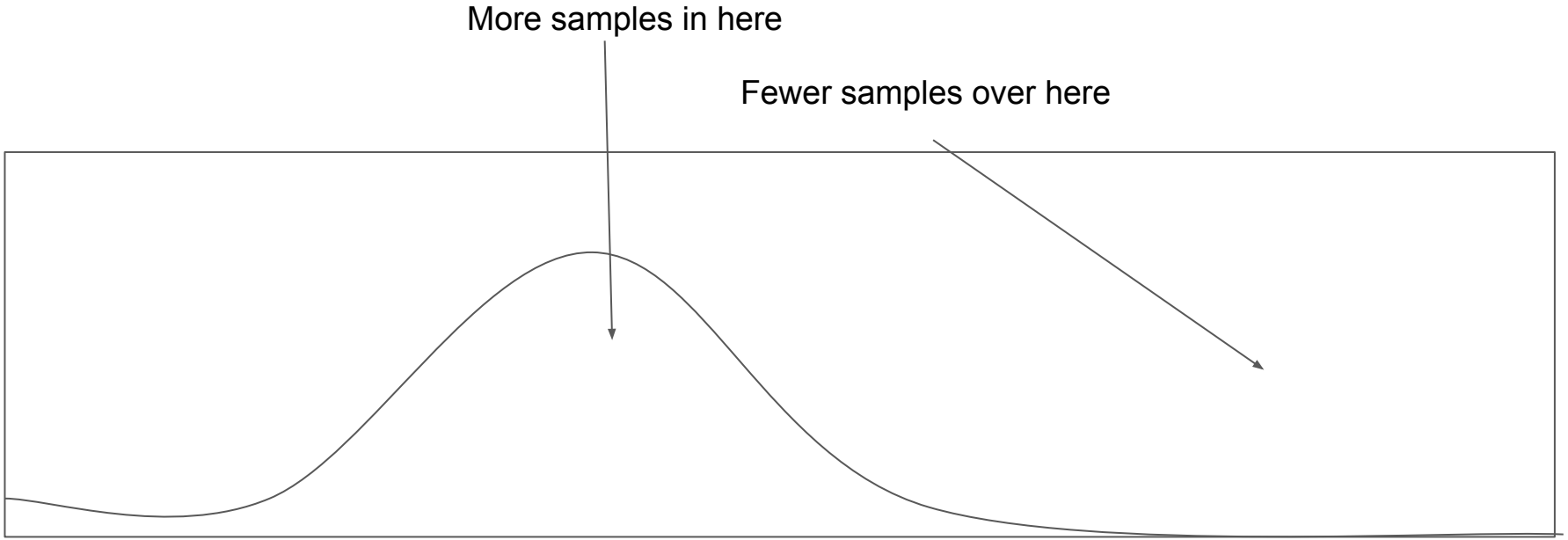
Because integration is hard.

It can be easy to specify a model but difficult/impossible to figure out equations for the parameters

Why MCMC?



Why MCMC?



= fewer rejected samples
= actually possible to sample from this distribution

Metropolis Hastings

Assume we have a likelihood function (or something proportional to a likelihood function) called f(x).

Choose an arbitrary x_0 to start with.

To generate x_{n+1}:

1. Generate a candidate x'. Usually this is picked from a Normal distribution centred on x_n
2. Calculate the acceptance ratio $a = f(x')/f(x_n)$
3. If the acceptance ratio >= 1 then $x_{n+1}=x'$
   If $a < 1$ then $x_{n+1}= x'$ with probability a and $x_{n+1}= x_n$ otherwise

Metropolis Hastings

Example:

You steal your friends lucky coin flip coin because you want to check if it is fair or not. You flip the coin 100 times and observe 56 heads. What is the posterior distribution of P(flip=H)?

Metropolis Hastings

Write a function that gives the probability of observing this data given that the probability of flipping a head is *p*.

Metropolis Hastings

```
library(mcmc) # A useful library!

# log-likelihood function

llhood <- function(parameters) {

  p <- parameters[1]

  if(p>1 || p<0) { return(-Inf) }

  ll <- dbinom(56,100,p,log=TRUE)

  return(ll)

}
```

Metropolis Hastings

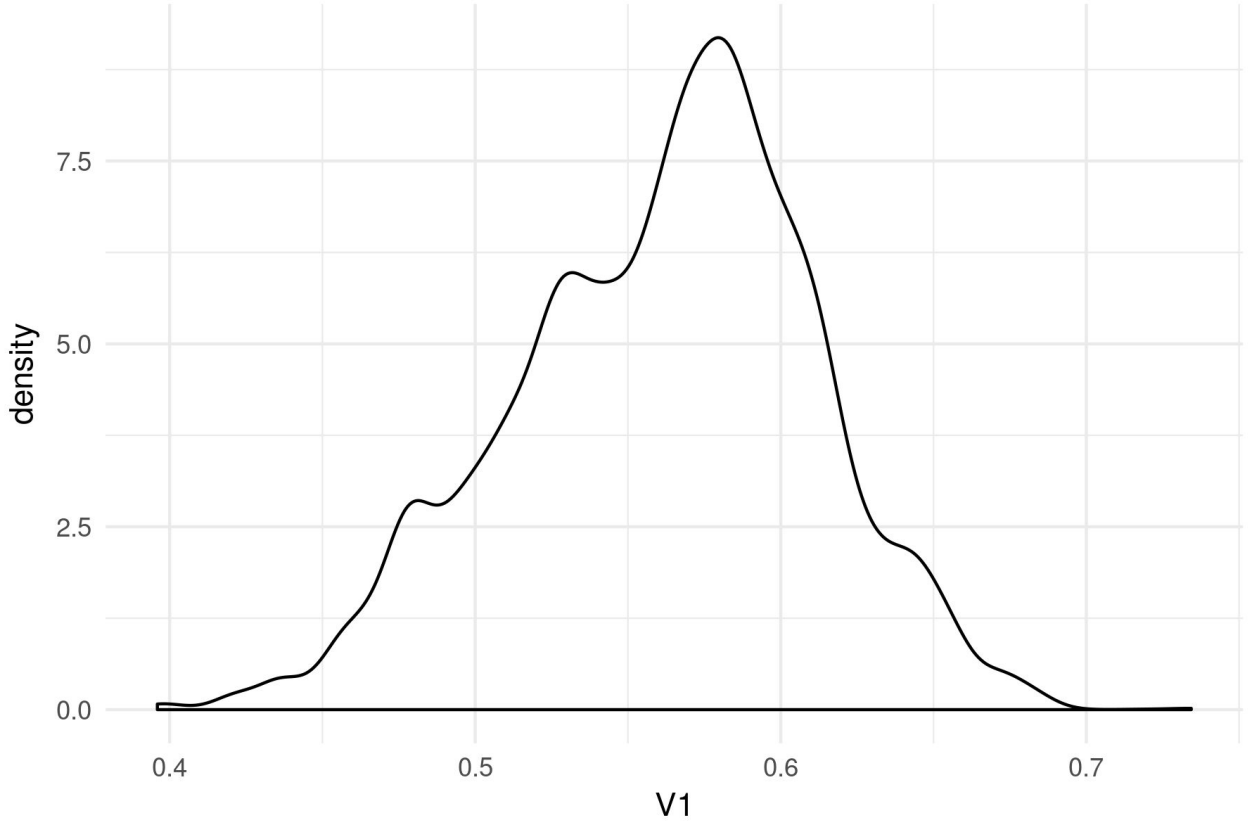
Continued…

```
out <- metrop(llhood,c(0.5),10000)

trace <- as.data.frame(out$batch)

ggplot(trace,aes(x=V1)) + geom_density() + theme_minimal()
```

Metropolis Hastings



Metropolis Hastings

Example 2:

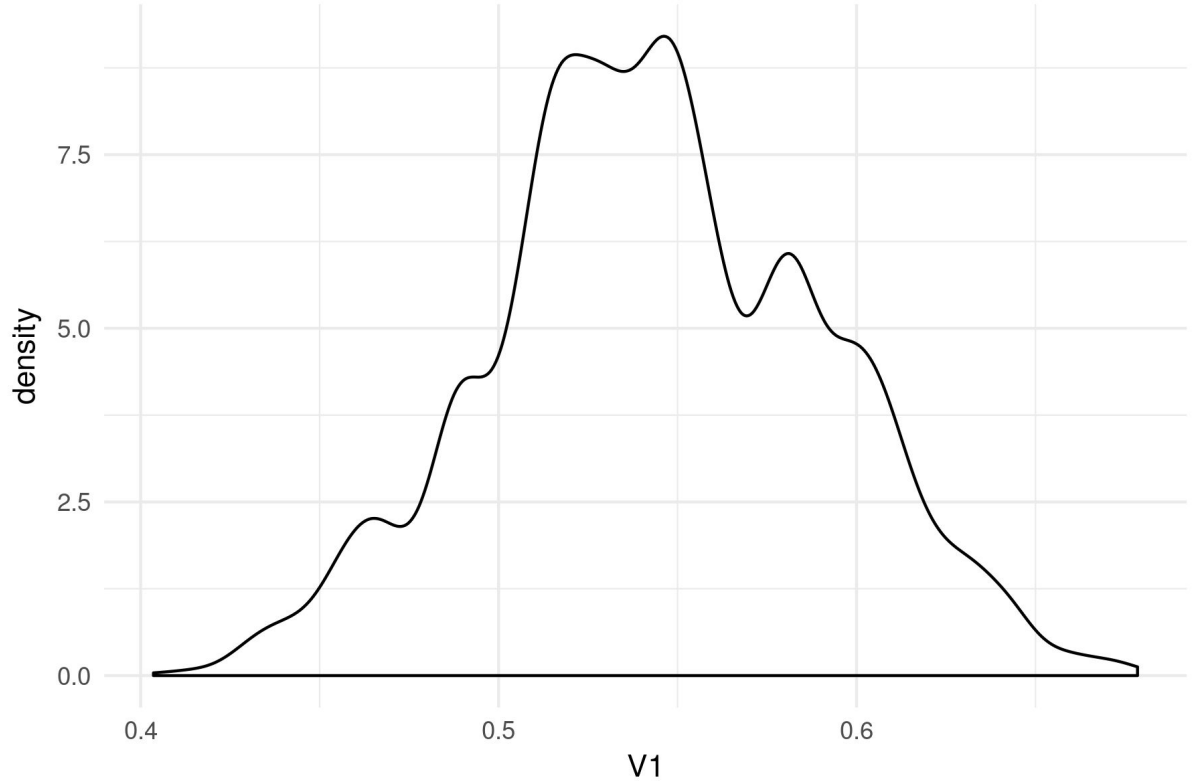
Your friend **always** chooses tails in a coin flip. So, if his coin is biased, it probably isn’t biased towards heads.

Include this information in the model and come up with a new estimate.

Metropolis Hastings

```
llhood <- function(parameters) {
  p <- parameters[1]
  if(p>1 || p<0) { return(-Inf) }
  lp <- dbeta(p,2,5,log=TRUE)
  ll <- dbinom(56,100,p,log=TRUE)
  return(ll+lp)
}
```

Metropolis Hastings



Metropolis Hastings

Write a function for the likelihood of observing the following data given that it is drawn from a normal distribution with mean *mu* and standard deviation 1:

6
6
8
11
4

Normal Distribution pdf:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Stan

Metropolis Hastings is simple and it runs quite quickly.

But it has a few weaknesses:

- The “step size” needs tuning
- The “random” way it moves around can be inefficient
- Struggles with multi-modal distributions where there are wide valleys between peaks (but to a certain extent, every method struggles with these)

Stan

Stan is two things:

1. A special programming language that makes it easier to specify models
2. A very sophisticated set algorithms for sampling these models

https://github.com/stan-dev/stan/releases/download/v2.14.0/stan-reference-2.14.0.pdf

Stan

```
data {
  int<lower=1> flips;
  int<lower=0> heads;
}
parameters {
  real<lower=0,upper=1> p;
}
model {
  heads ~ binomial(flips,p)
}
```

Stan

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

d <- list(flips=100, heads=56)
fit <- stan("model.stan",data=d, iter=10000, chains=4)

library(shinystan)
launch_shinystan(fit)
```

Stan

```
data {
  int<lower=1> flips;
  int<lower=0> heads;
}
parameters {
  real<lower=0,upper=1> p;
}
model {
  p ~ beta(2,5)
  heads ~ binomial(flips,p)
}
```

Stan

```
fit <- stan("model.stan",data=d, iter=10000, chains=4)

launch_shinystan(fit)
```

Exercises

1. Update the biased coin example after your friend tells you “60% of the time it comes up tails”

2. Make a Stan model to estimate the conversion rate of a landing page

3. Make a model to compare the conversion rates of two landing pages
   HINT: you can access the trace using
   ```
   samples <- extract(fit)
   ```

4. Can you do the same but for revenue per click?

Stan

Let’s look at the last question in more detail:

http://www.eanalytica.com/different-test-models/

Evaluating Model Fit

This is complicated.

Rules of thumb:

1. Generate data using known parameters and the same random process. Can stan recover the parameters?
2. Check the diagnosis panel in shinystan
3. Lots of errors or poor convergence might mean your model is not a good fit for the data

Forecasting

Key idea:

Performance can be split into separate components. e.g.

1. Underlying trend
2. Seasonal variation
3. Randomness

Forecasting

```
data {
 int<lower=1> days;
 vector[days] day;
 vector[days] sessions;
}
parameters {
 real intercept;
 real slope;
 real<lower=0> error_sd;
}
model {
  sessions ~ normal(day*slope + intercept, error_sd);
}
```

Forecasting

```
data {
 int<lower=1> days;
 vector[days] day;
 vector[days] sessions;
}
parameters {
 real intercept;
 real slope;
 real<lower=0> error_sd;
 real<lower=0> period;
 real<lower=0,upper=7> offset;
 real<lower=0> seasonality_size;
}
model {
  sessions ~ normal(day * slope + intercept + seasonality_size * sin(period*day + offset),
error_sd);
}
```

Exponential Smoothing

```
data {
      int<lower=0> N;
      real<lower=0> y[N];
}
parameters {
      real trend[N];
      real s_trend;
      real s_q;
      real d;
}
model {
      real q[N];
      real cum_trend[N];
      for (i in 3:N)
            trend[i]~normal(2*trend[i-1]-trend[i-2],s_trend);

      cum_trend[1]<-trend[1];
      for (i in 2:N)
            cum_trend[i]<-cum_trend[i-1]+trend[i];

      for (i in 1:N)
            q[i]<-y[i]-cum_trend[i];
      for (i in 1:N)
            q[i]~normal(d,s_q);
}
```

Exercise

1. Modify the previous examples to include media spend as a dependent variable in the forecast

2. What if there is a non-linear relationship between sessions and media spend (e.g. power law)?

3. Can you add more seasonal components? E.g. weekly, monthly, yearly

4. What if the growth is logistic?
   i.e. *growth(t) = C/exp(-k(t-offset))*

Holidays

Some holidays occur at the same time every year and can be forecast by the usual seasonality methods with frequency 365.

Other holidays are not so convenient:

- Easter
- Sporting events
- Sales

Holidays

Two things needed here:

1. Train model parameters on past events
2. Use these parameters when forecasting future events

Holidays

```
data {
 int<lower=1> days;
 vector[days] day;
 vector[days] sessions;
}
parameters {
 real intercept;
 real slope;
 real<lower=0> sale_boost;
 real<lower=0> error_sd;
}
model {
  for (i in 1:20)
    sessions[i] ~ normal(day[i]*slope + intercept, error_sd);
  for (i in 21:25)
    sessions[i] ~ normal(day[i]*slope + intercept + sale_boost, error_sd);
  for (i in 26:days)
    sessions[i] ~ normal(day[i]*slope + intercept, error_sd);
}
```

Prophet

Facebook have recently released an R library that uses some of these ideas wrapped in a simpler user interface.

https://facebookincubator.github.io/prophet/

Hierarchical Models

Hierarchical models are a way to share information between different model parameters.

For example, with a landing page test I would be surprised if one version had a 99% conversion rate and the other version had a 1% conversion rate. I don’t know what the conversion rates are, but I expect them to be similar in some ways.

Or, if I’m forecasting for different cities then I don’t expect the trend to be exactly the same but I expect them to be similar

Hyperparameters

```
data {
 int<lower=1> days;
 vector[days] day;
 vector[days] sessions1;
 vector[days] sessions2;
}
parameters {
 real intercept1;
 real intercept2;
 real slope1;
 real slope2;
 real<lower=0> error_sd;
}
model {
  sessions1 ~ normal(day*slope1 + intercept1, error_sd);
  sessions2 ~ normal(day*slope2 + intercept2, error_sd);
}
```

Hyperparameters

```
data {
 int<lower=1> days;
 vector[days] day;
 vector[days] sessions1;
 vector[days] sessions2;
}
parameters {
 real slope_mean;
 real<lower=0> slope_sd;
 real intercept1;
 real intercept2;
 real slope1;
 real slope2;
 real<lower=0> error_sd;
}
model {
  slope1 ~ normal(slope_mean,slope_sd);
  slope2 ~ normal(slope_mean,slope_sd);
  sessions1 ~ normal(day*slope1 + intercept1, error_sd);
  sessions2 ~ normal(day*slope2 + intercept2, error_sd);
}
```

My Favourite Example

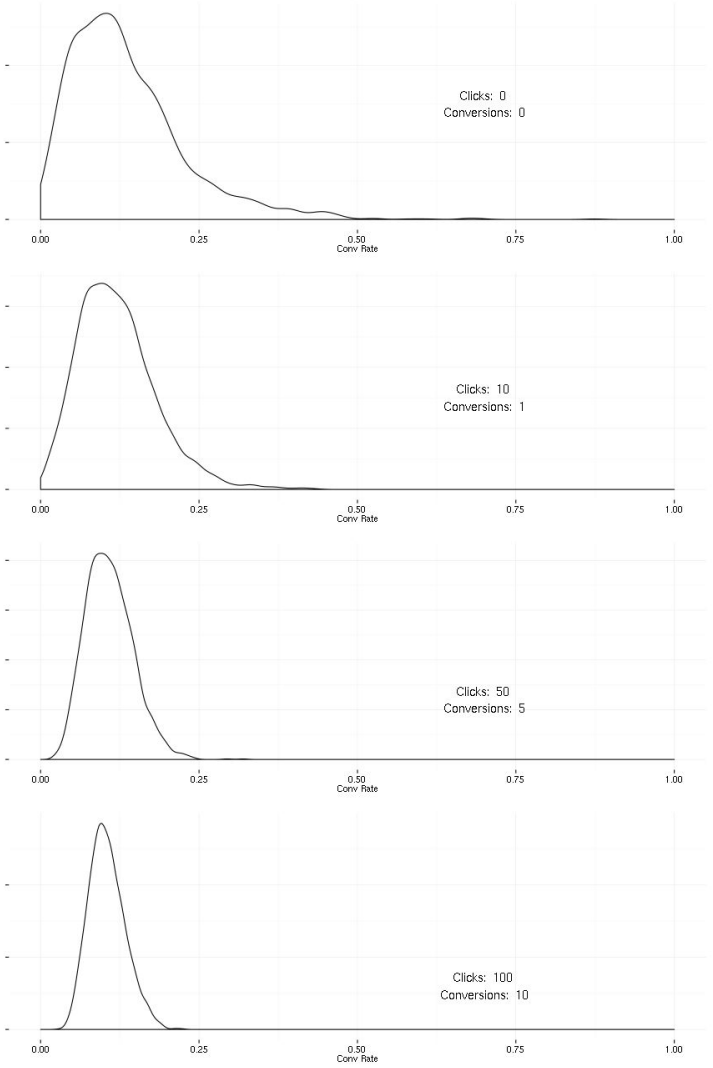
```
data {
 int<lower=0> J; // number of keywords
 int<lower=0> conv[J]; // number of conv for kwd j
 int<lower=0> click[J]; // number of clicks for j
}
parameters {
 real<lower=0,upper=1> theta[J]; // true conv rate for j
 real<lower=0> alpha; // prior conv count
 real<lower=0> beta; // prior fail count
}
model {
 // distribution parameters for the hyperpriors
 // are chosen arbitrarily (this matters less here)
 alpha ~ exponential(0.1); // hyperprior
 beta ~ exponential(0.1); // hyperprior
 theta ~ beta(alpha+1,beta+1); // prior
 conv ~ binomial(click,theta); // likelihood
}
```

My Favourite Example

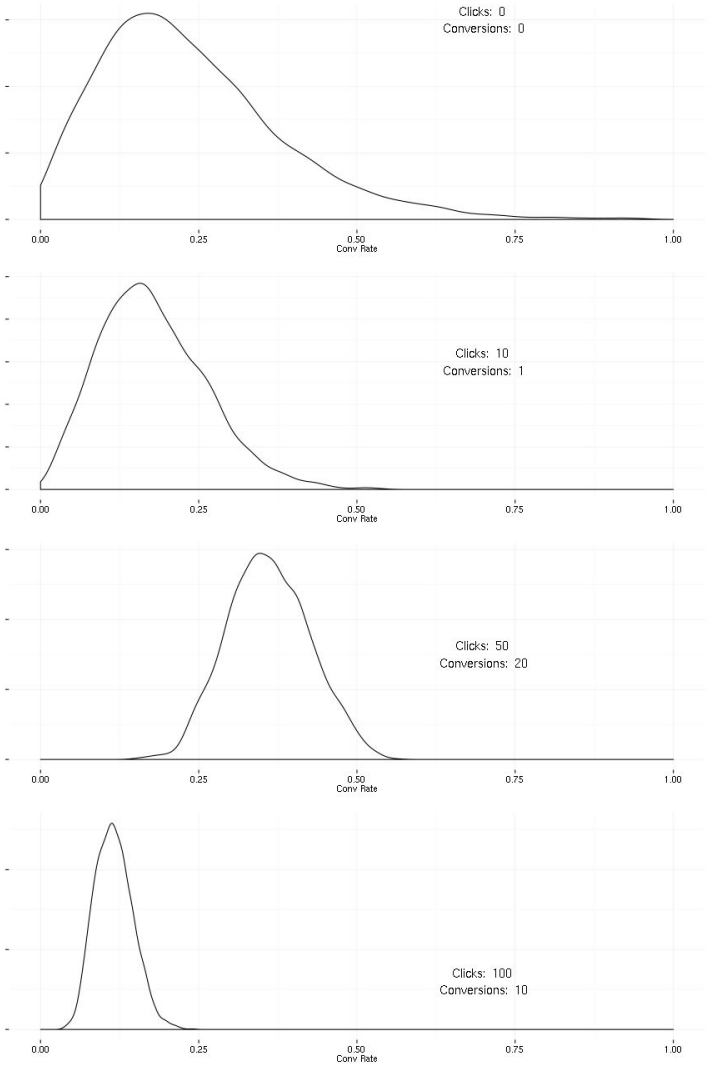
```
keyword_data <- list(J = 4,
                     conv = c(0,1,5,10),
                     click = c(0,10,50,100)
                     )

fit <- stan("model.stan",
            data=keyword_data,
            iter=1000,
            chains=4)
```

My Favourite Example



My Favourite Example



Final Exercises

1. Fit a hierarchical forecast for two product ranges.
   You expect one of them to have higher sales during a holiday period

2. Compare how many clicks/conversions are needed to reach 95% certainty on a split test in the following circumstances:
   a. Naive prior, no information sharing
   b. With a prior where you expect the conversion rate to be around 10%
   c. Where previous landing page tests have the following results:

A	B
5%	6%
8%	3%
15%	6%