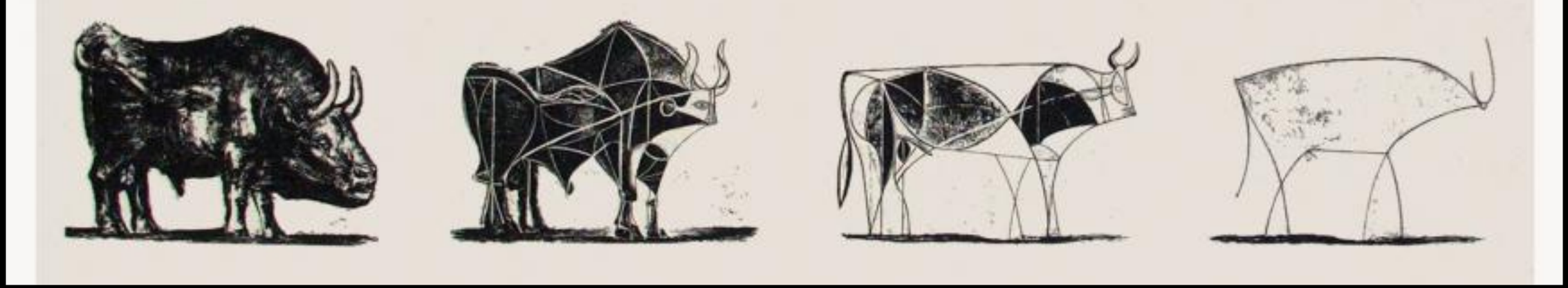
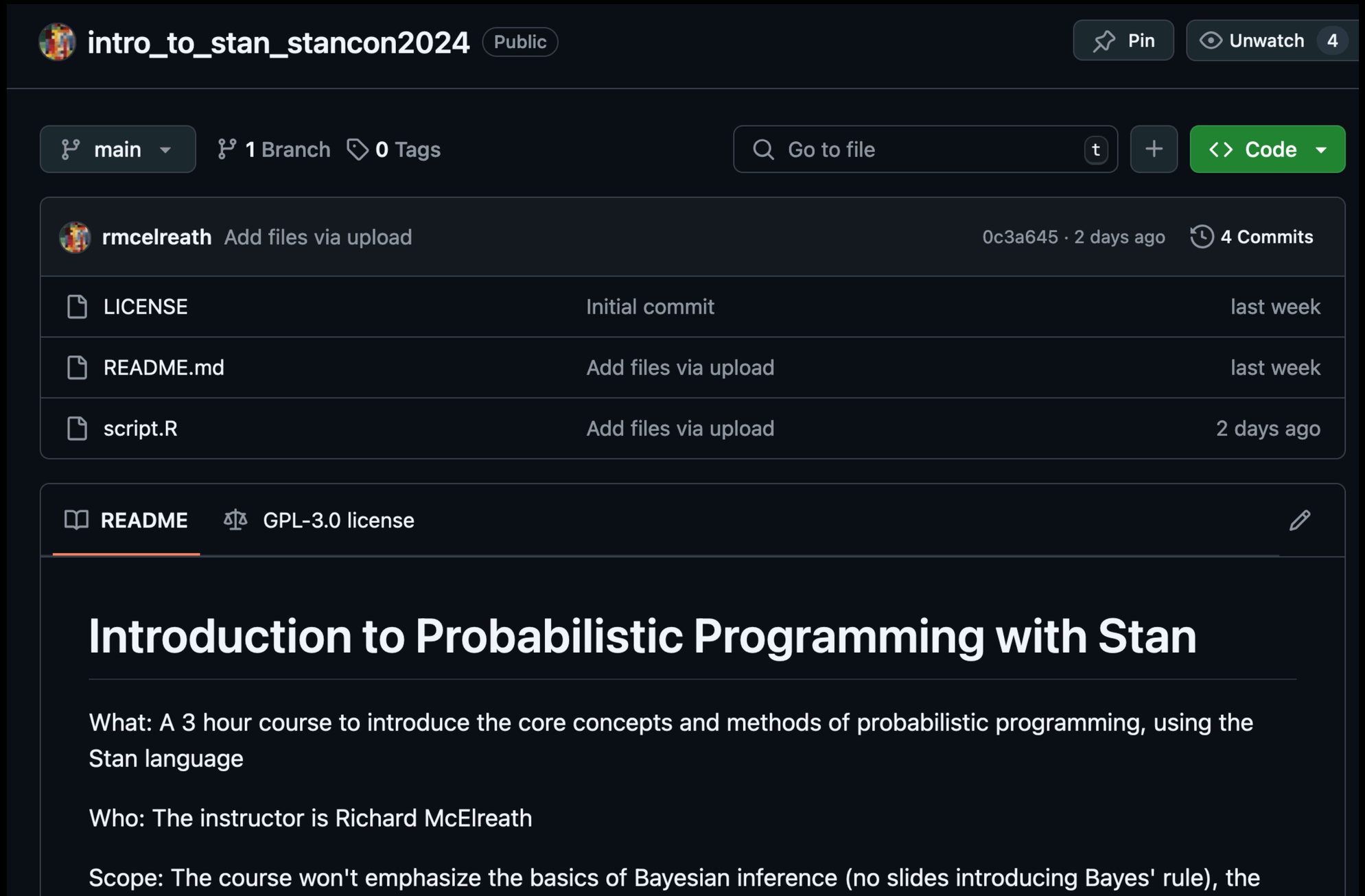


Introduction to probabilistic programming with *Stan*

Richard McElreath

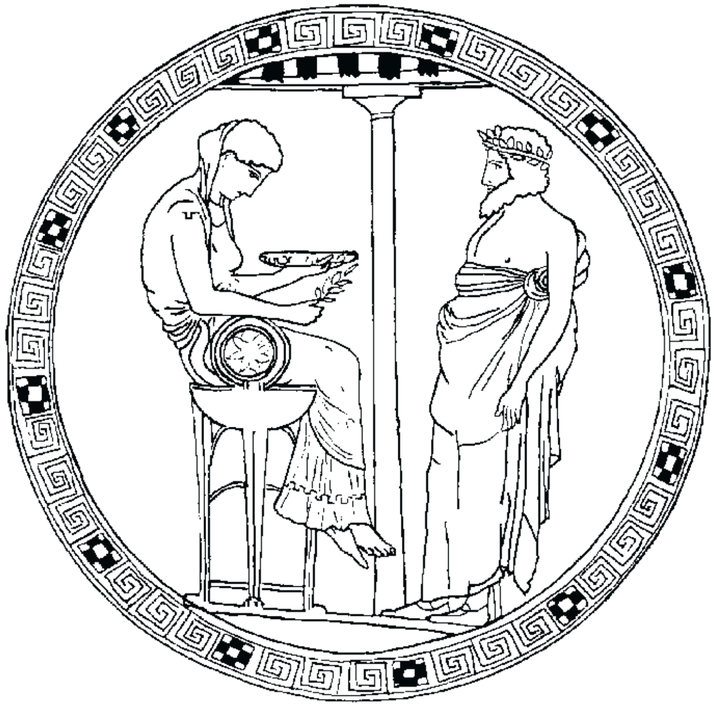
StanCon Oxford 2024

https://github.com/rmcelreath/intro_to_stan_stancon2024



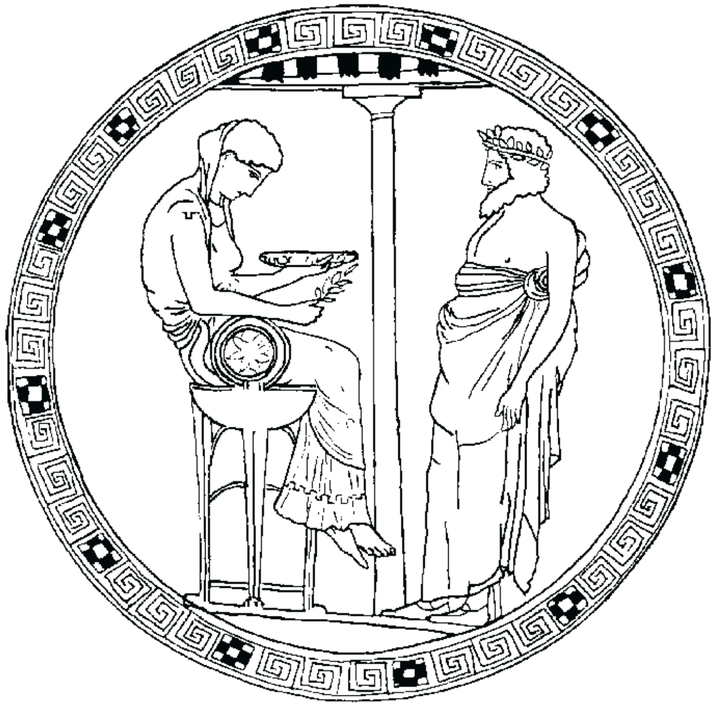
Probabilistic Programming

- What if there were an oracle you could feed your assumptions and it would reveal the implications?



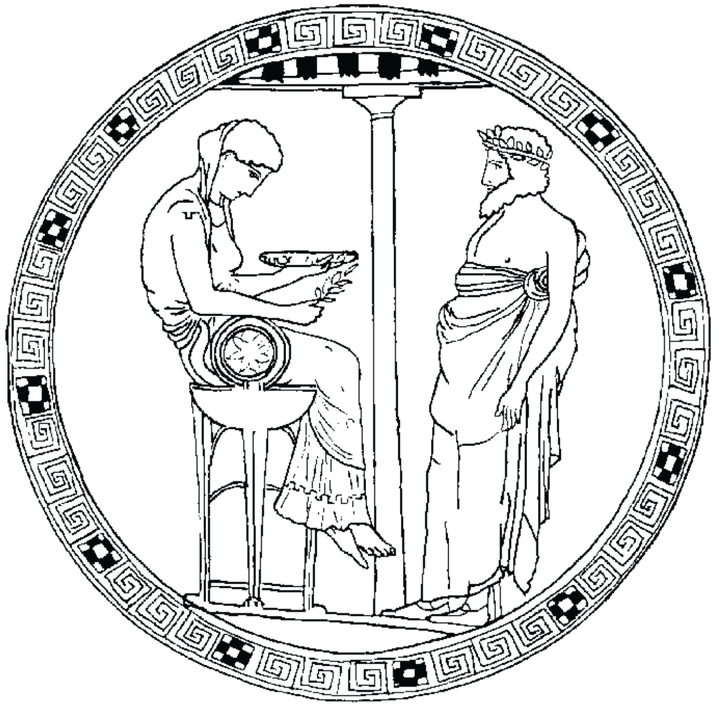
Probabilistic Programming

- What if there were an oracle you could feed your assumptions and it would reveal the implications?
- What if the oracle required information to be phrased in a very specific and non-obvious way?



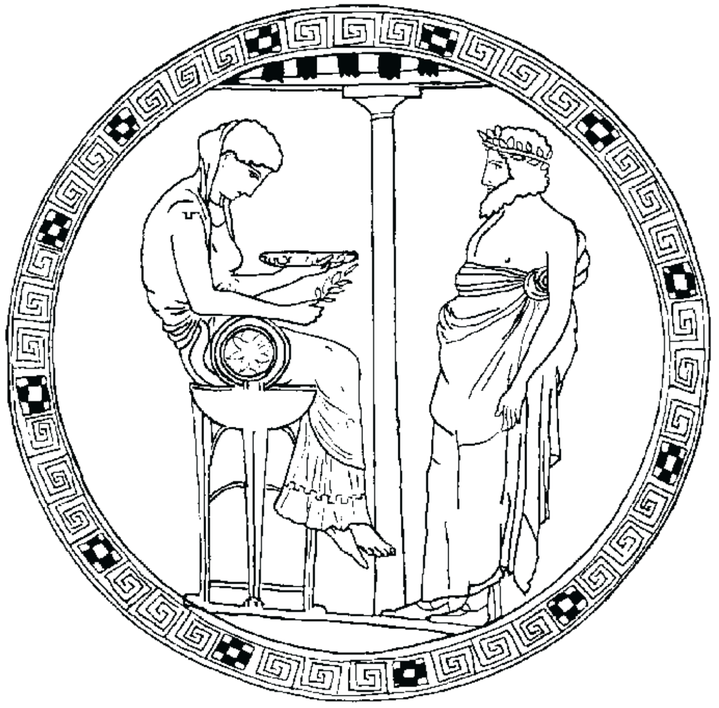
Probabilistic Programming

- What if there were an oracle you could feed your assumptions and it would reveal the implications?
- What if the oracle required information to be phrased in a very specific and non-obvious way?
- What if the oracle encrypted the answer?



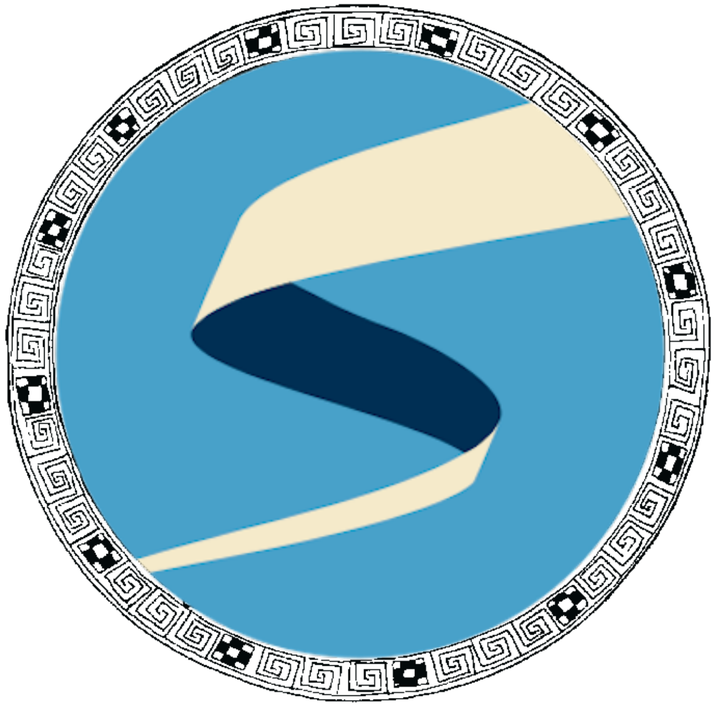
Probabilistic Programming

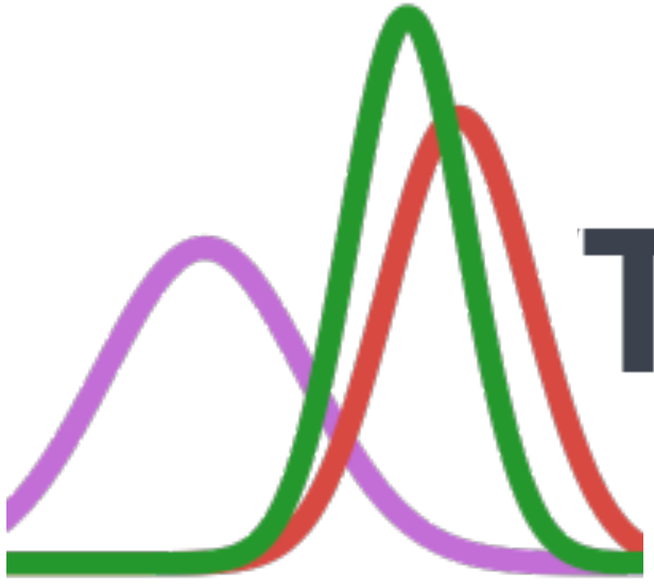
- What if there were an oracle you could feed your assumptions and it would reveal the implications?
- What if the oracle required information to be phrased in a very specific and non-obvious way?
- What if the oracle encrypted the answer?
- What if the oracle were indifferent to human suffering?



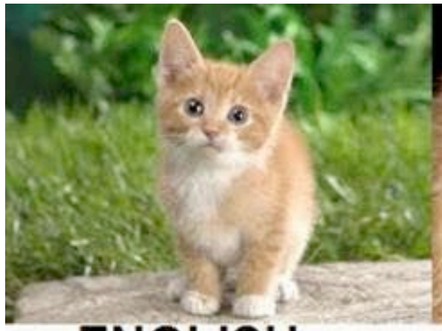
Stan is an indifferent oracle

- Stan is a language and framework for defining and sampling from probabilistic models
- Expressive, optimized language for statistical computing
- Variety of sampling algorithms using gradients
- Analytical gradients via automatic differentiation
- Diagnostics, profiling, and more

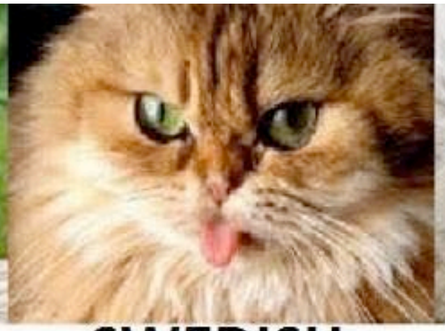




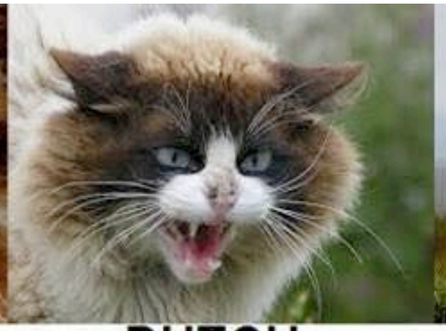
Turing.jl



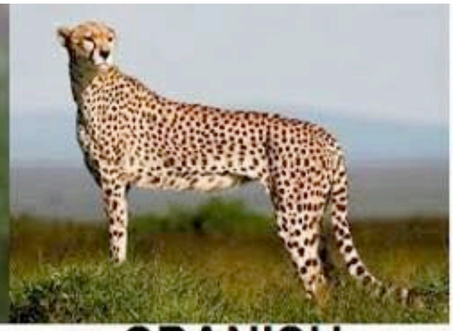
ENGLISH
Grammar



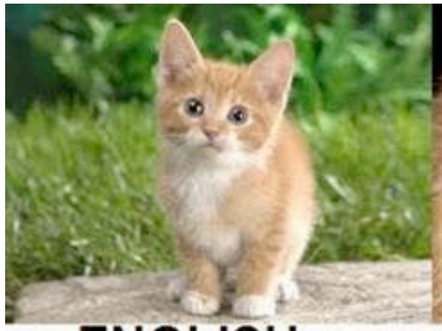
SWEDISH
Grammar



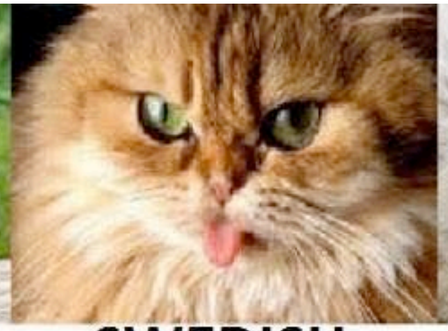
DUTCH
Grammar



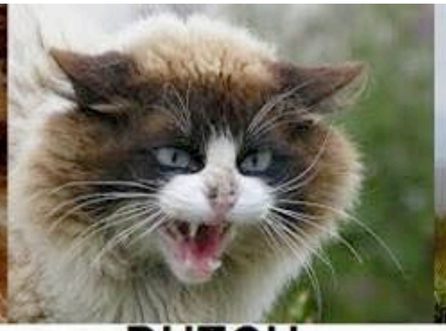
SPANISH
Grammar



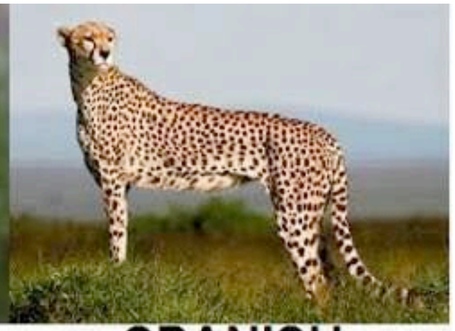
ENGLISH
Grammar



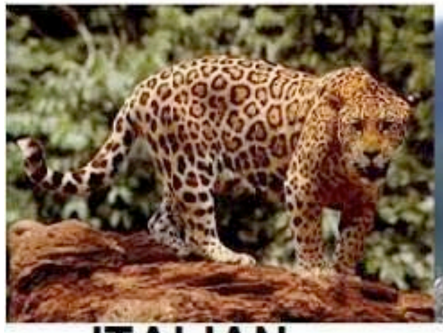
SWEDISH
Grammar



DUTCH
Grammar

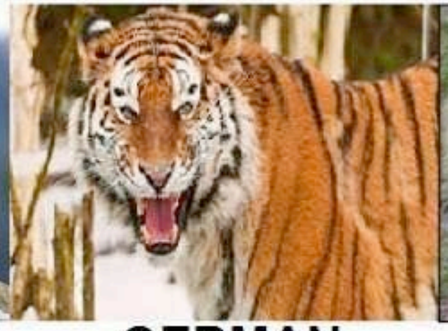


SPANISH
Grammar



ITALIAN
Grammar

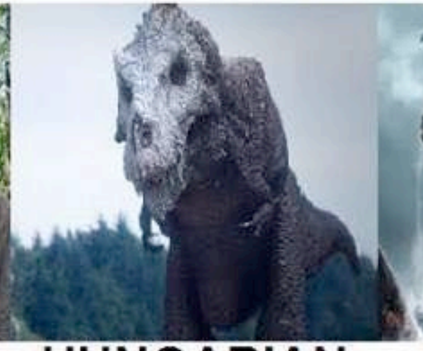
FRENCH
Grammar



GERMAN
Grammar

TURKISH
Grammar

POLISH
Grammar



HUNGARIAN
Grammar

ICELANDIC
Grammar

FINNISH
Grammar

ARABIC
Grammar

Modest Goals

- Learn the basic concepts of probabilistic programming
- Learn how to express ideas as probability models
- Learn conventions of Stan language
- Learn a reasonable, reliable workflow

Not-Goals

- Introduce Bayesian inference
- Introduce Markov chain Monte Carlo
- Explain the details of automatic differentiation
- Code complex statistical models

Part 1: Probability as Logic

Part 2: Generative Models

Part 3: Reasonable Workflow

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

Two trains leave their stations at the same time, traveling towards one another along the same 300 km long track. The first train is traveling at 100 km/h. The second 120 km/h. How long before the two trains meet?

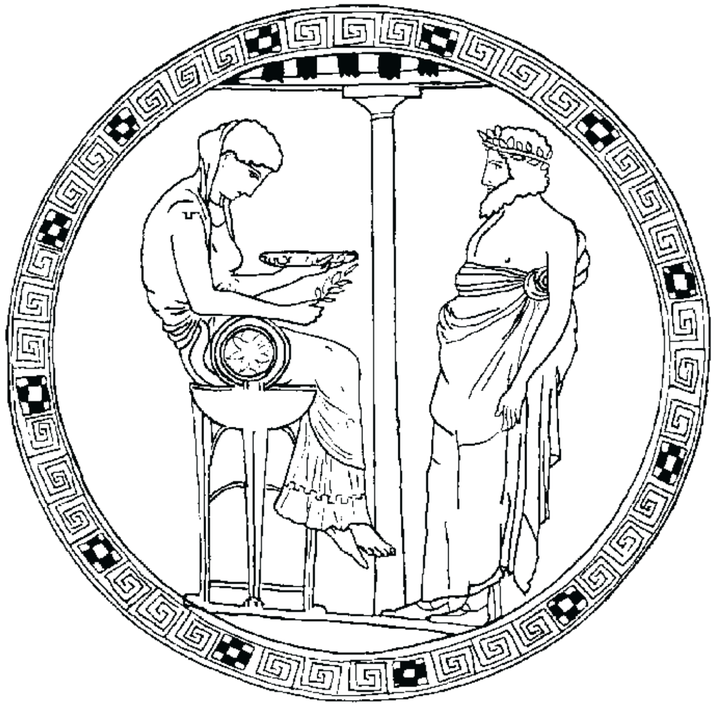
Two trains leave their stations at the same time, traveling towards one another along the same 300 km long track. The first train is traveling at 100 km/h. The second train is faster than the first and has a maximum speed of 120 km/h. How long before the two trains meet?

Given national-level data on the number of heavy metal bands per capita and the average self-reported happiness, what is the best-fit line measuring the association between these two variables?

300,000 people opted in to respond to an online survey on the Xbox gaming platform. Data is available on respondent gender and age. 90% of the sample is men, and 60% is 18–29 years old. How can we use these data to predict what the full population thinks?

Probabilistic Programming

- What if there were an oracle you could feed your assumptions and it would reveal the implications?
- What if the oracle required information to be phrased in a very specific and non-obvious way?



Probabilistic Programming Workflow

- The oracle knows probability theory and understands only variables, constraints, distributions, and relationships (VCDR)
  - 1) Translate problem into variables, constraints, distributions, relationships
  - 2) Express VCDR as a program
  - 3) ????
  - 4) Profit

Information: VCDR

- Variables (measurements, rates, probabilities, states)
- Constraints on variables (min, max, discrete, continuous)
- Distributions of variables
- Relationships among variables (and their distributions)

Variables

- Observed (data)
  - Measurements, counts, presence/absence, categories
- Unobserved (parameters)
  - Rates, expectations, probabilities, latent measurements etc

5

Translate this sentence

Good evening.

vas | a | buenas | mi

padre | dias | noches

CHECK

5

Translate this sentence

Y is positive

real | Y | upper | =

<lower | array | 0>

CHECK

because this yields a lower-dimensional parameter space over which to do inference. We'll fit both models in Stan. The former model will be referred to as the latent variable GP, while the latter will be called the marginal likelihood GP.

The hyperparameters controlling the covariance function of a Gaussian process can be fit by assigning them priors, like we have in the generative models above, and then computing the posterior distribution of the hyperparameters given observed data. The priors on the parameters should be defined based on prior knowledge of the scale of the output values (α), the scale of the output noise (σ), and the scale at which distances are measured among inputs (ρ). See the Gaussian process priors section for more information about how to specify appropriate priors for the hyperparameters.

The Stan program implementing the marginal likelihood GP is shown below. The program is similar to the Stan programs that implement the simulation GPs above, but because we are doing inference on the hyperparameters, we need to calculate the covariance matrix `K` in the model block, rather than the transformed data block.

```
data {
  int<lower=1> N;
  array[N] real x;
  vector[N] y;
}
transformed data {
  vector[N] mu = rep_vector(0, N);
}
parameters {
  real<lower=0> rho;
  real<lower=0> alpha;
  real<lower=0> sigma;
```

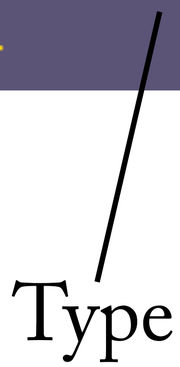
“The duration is between 5 and 25 minutes.”

One variable: duration
Constraints: minimum 5, maximum 25

```
1   // Stan model 01 - "The duration is between 5 and 25 minutes"
2   parameters{
3       real< lower=5 , upper=25 > duration;
4   }
```

```
// Stan model 01 - "The duration is between 5 and 25 minutes"
parameters{
    real< lower=5 , upper=25 > duration;
}
```

```
1   // Stan model 01 - "The duration is between 5 and 25 minutes"
2   parameters{
3       real< lower=5 , upper=25 > duration;
4   }
```

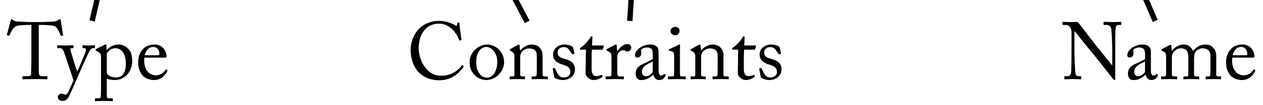


real = continuous

```
1   // Stan model 01 - "The duration is between 5 and 25 minutes"
2   parameters{
3       real< lower=5 , upper=25 > duration;
4   }
```

real = continuous

```stan
// Stan model 01 - "The duration is between 5 and 25 minutes"
parameters{
    real< lower=5 , upper=25 > duration;
}
```



real = continuous

Block *Unobserved variables = parameters*

```
1   // Stan model 01 - "The duration is between 5 and 25 minutes"
2   parameters{
3       real< lower=5 , upper=25 > duration;
4   }
```

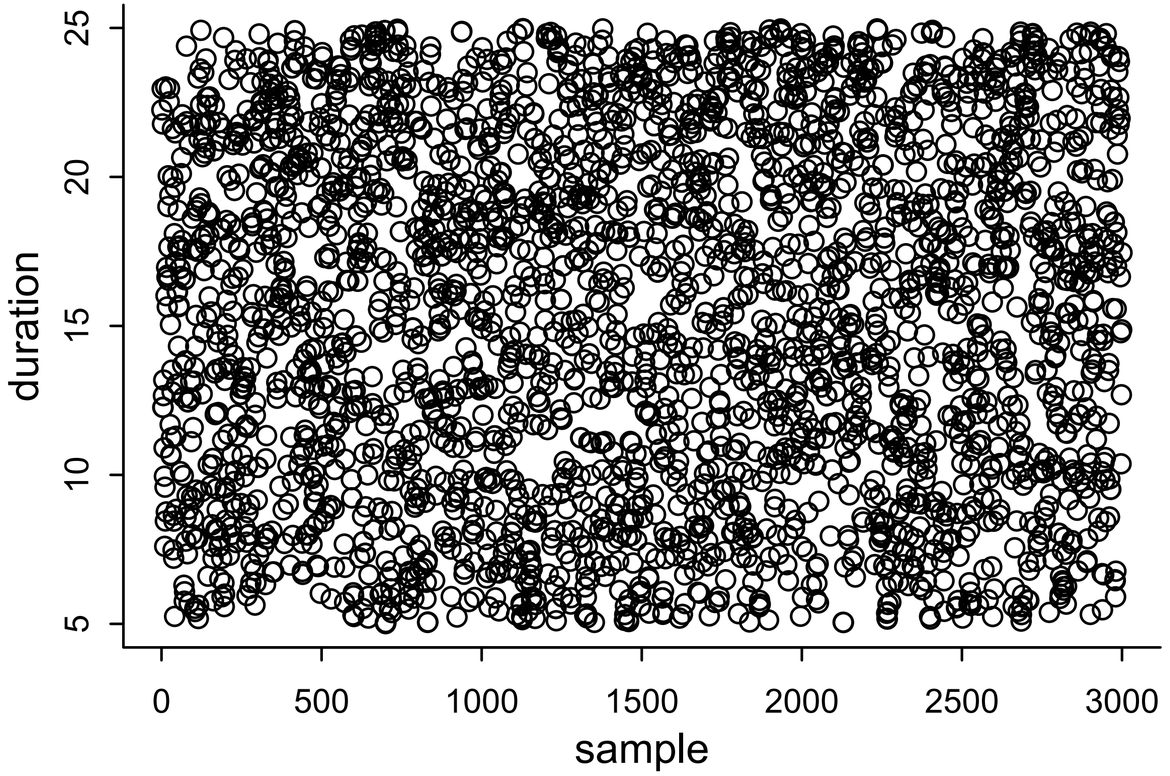
Type Constraints Name

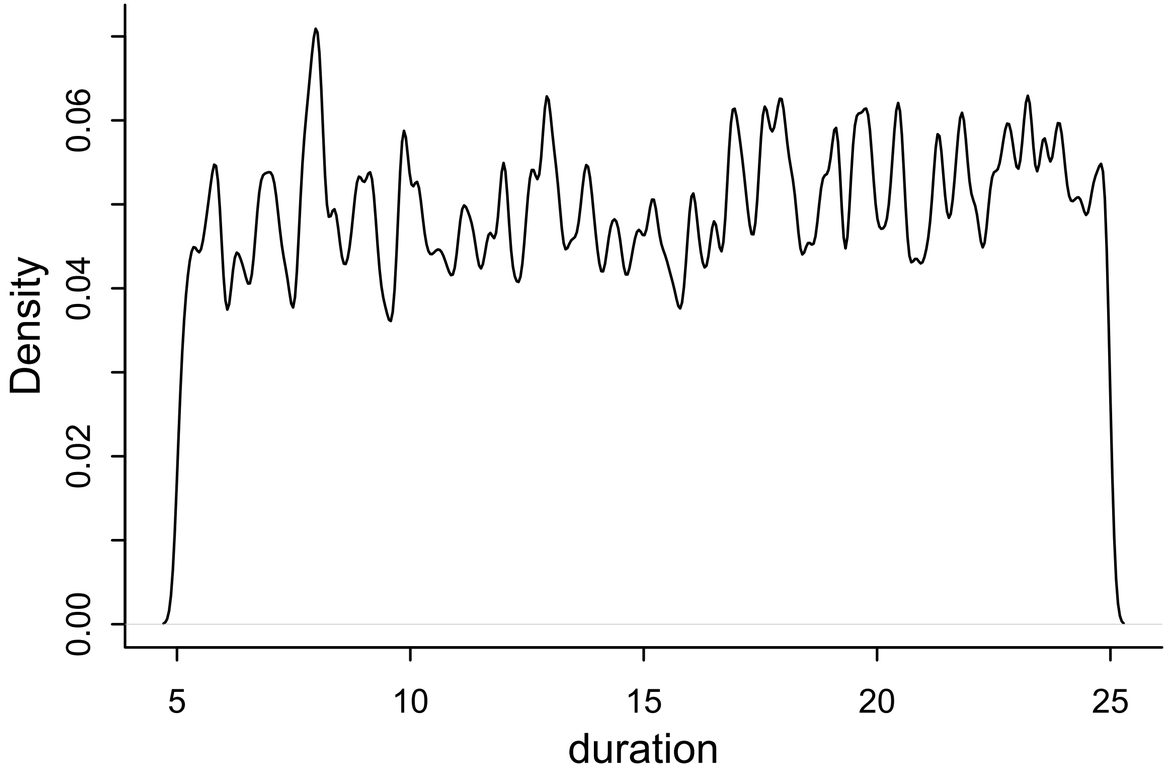
real = continuous

```
model_code_1 <- "
parameters{
    real< lower=5 , upper=25 > duration;
}
"

f <- write_stan_file(model_code_1)
model_1 <- cmdstan_model(f)

samples_1 <- model_1$sample(
    data = list(),
    seed = 123,
    chains = 4,
    parallel_chains = 4
)

samples_1$summary()
```

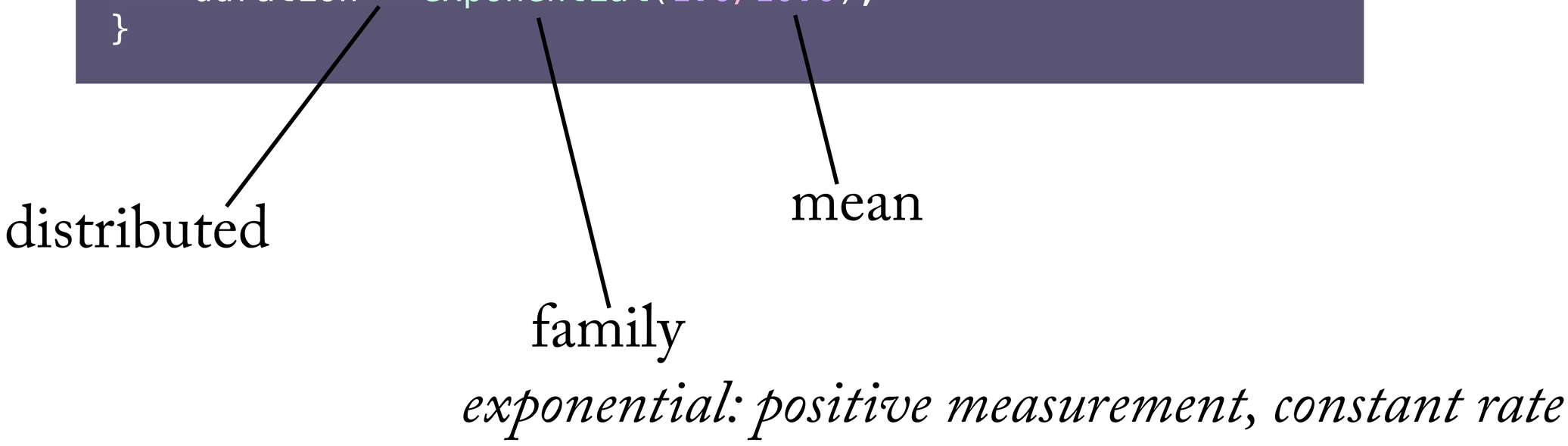
```
> samples_1$summary()
# A tibble: 2 × 10
  variable  mean median    sd   mad     q5   q95  rhat ess_bulk ess_tail
  <chr>    <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 lp__      1.00   1.31 0.803 0.400 -0.660  1.61  1.00    1651.    1516.
2 duration 15.3   15.4  5.79  7.48   6.14  24.0   1.00    1689.    1521.
```

```
1    // Stan model 01 - "The duration is between 5 and 25 minutes"
2    parameters{
3        real< lower=5 , upper=25 > duration;
4    }
```



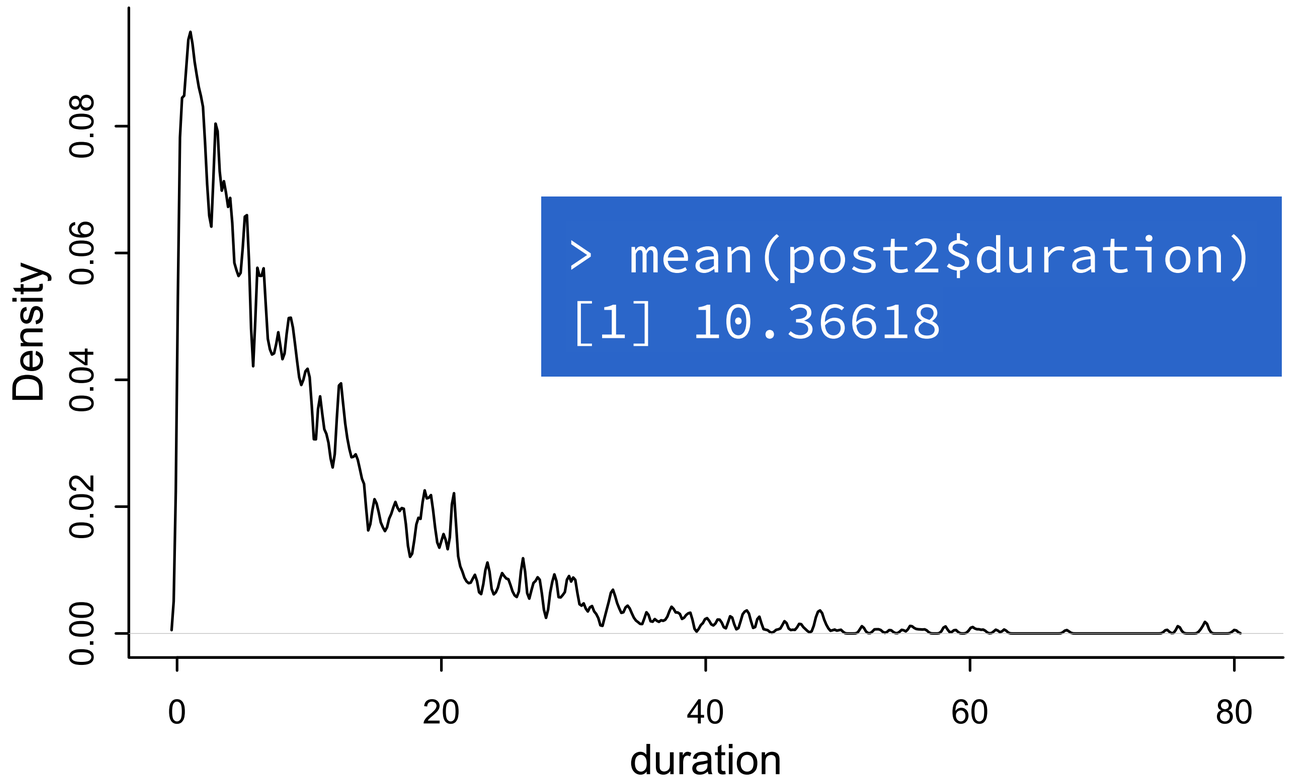


```
// Stan model 02 - "The average duration is 10 minutes"
parameters{
    real< lower=0 > duration;
}
model{
    duration ~ exponential(1.0/10.0);
}
```



exponential: positive measurement, constant rate

```
// Stan model 02 - "The average duration is 10 minutes"
parameters{
    real< lower=0 > duration;
}
model{
    duration ~ exponential(1.0/10.0);
}
```




```
> mean(post2$duration)
[1] 10.36618
```

```
// Stan model 3 - Add data - duration observed N times, what is average?
data{
    int N;
    array[N] real<lower=0> y; // observed durations
}
parameters{
    real< lower=0 > avg_duration;
}
model{
    y ~ exponential(1.0/avg_duration);
}
```

observed variables

data block

number of observations

array of observed durations

```
// Stan model 3 - Add data - duration observed N times, what is average?
data{
    int N;
    array[N] real<lower=0> y; // observed durations
}
parameters{
    real< lower=0 > avg_duration;
}
model{
    y ~ exponential(1.0/avg_duration);
}
```

observed variables

data block

number of observations

array of observed durations

```
// Stan model 3 - Add data - duration observed N times, what is average?
data{
    int N;
    array[N] real<lower=0> y; // observed durations
}
parameters{
    real< lower=0 > avg_duration;
}
model{
    y ~ exponential(1.0/avg_duration);
}
```

unobserved average

observed variables

data block

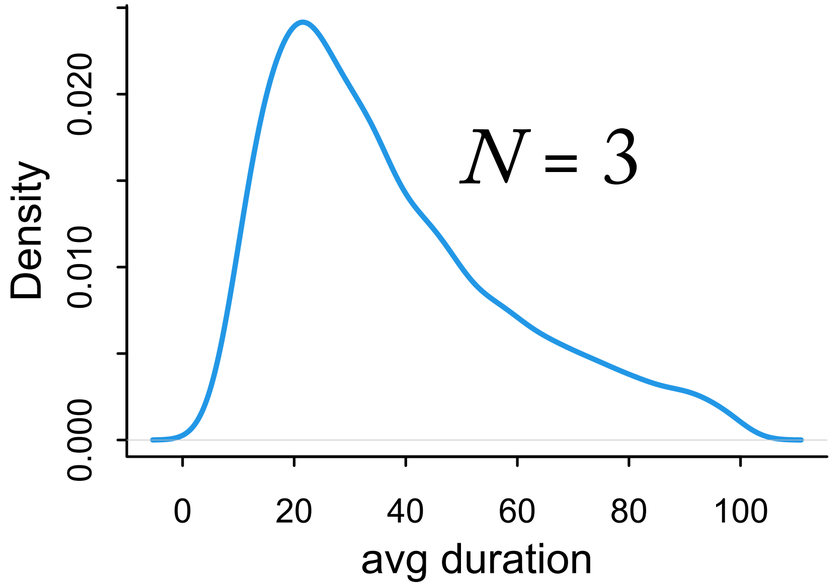
number of observations

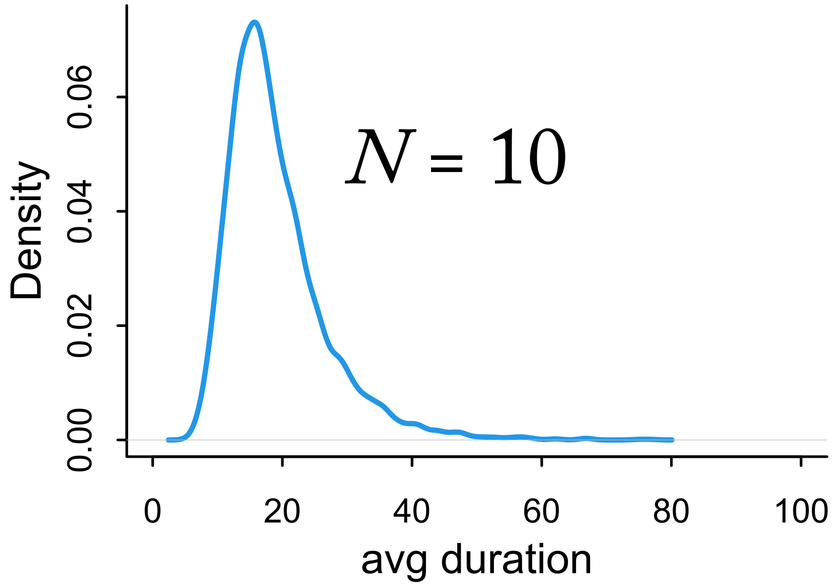
array of observed durations

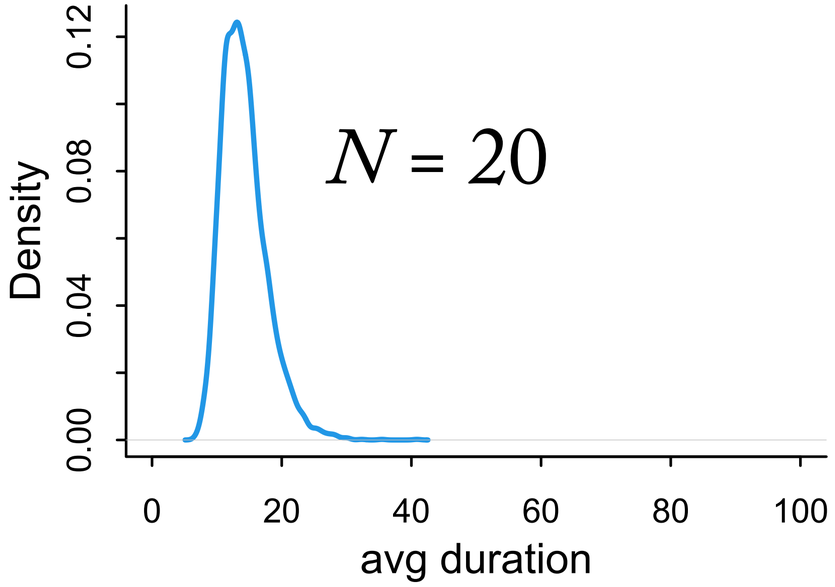
```
// Stan model 3 - Add data - duration observed N times, what is average?
data{
    int N;
    array[N] real<lower=0> y; // observed durations
}
parameters{
    real< lower=0 > avg_duration;
}
model{
    y ~ exponential(1.0/avg_duration);
}
```

unobserved average

```
// Stan model 3 - Add data - duration observed N times, what is average?
data{
    int N;
    array[N] real<lower=0> y; // observed durations
}
parameters{
    real< lower=0 > avg_duration;
}
model{
    y ~ exponential(1.0/avg_duration);
}
```







```
// Stan model 4 - relationship - duration associated with location x
data{
    int N;                      // number of observations
    int M;                      // number of locations
    array[N] real<lower=0> y; // observed durations
    array[N] int x;             // location (categorical)
}
parameters{
    array[M] real<lower=0> avg_duration;
}
model{
    y ~ exponential(avg_duration[x]^(-1));
}
```

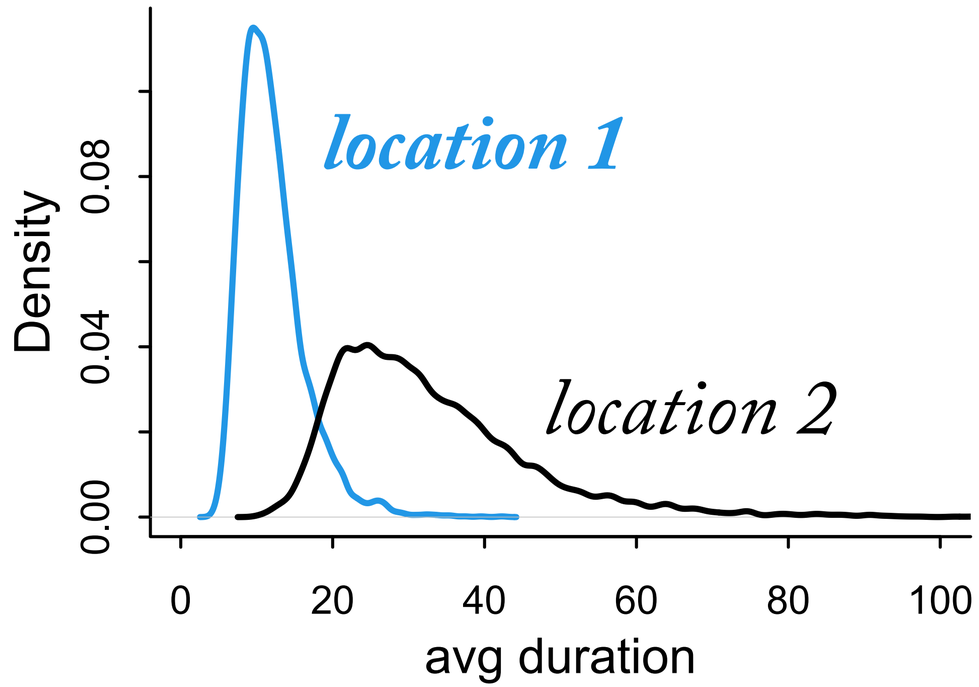
number of locations

average for each location

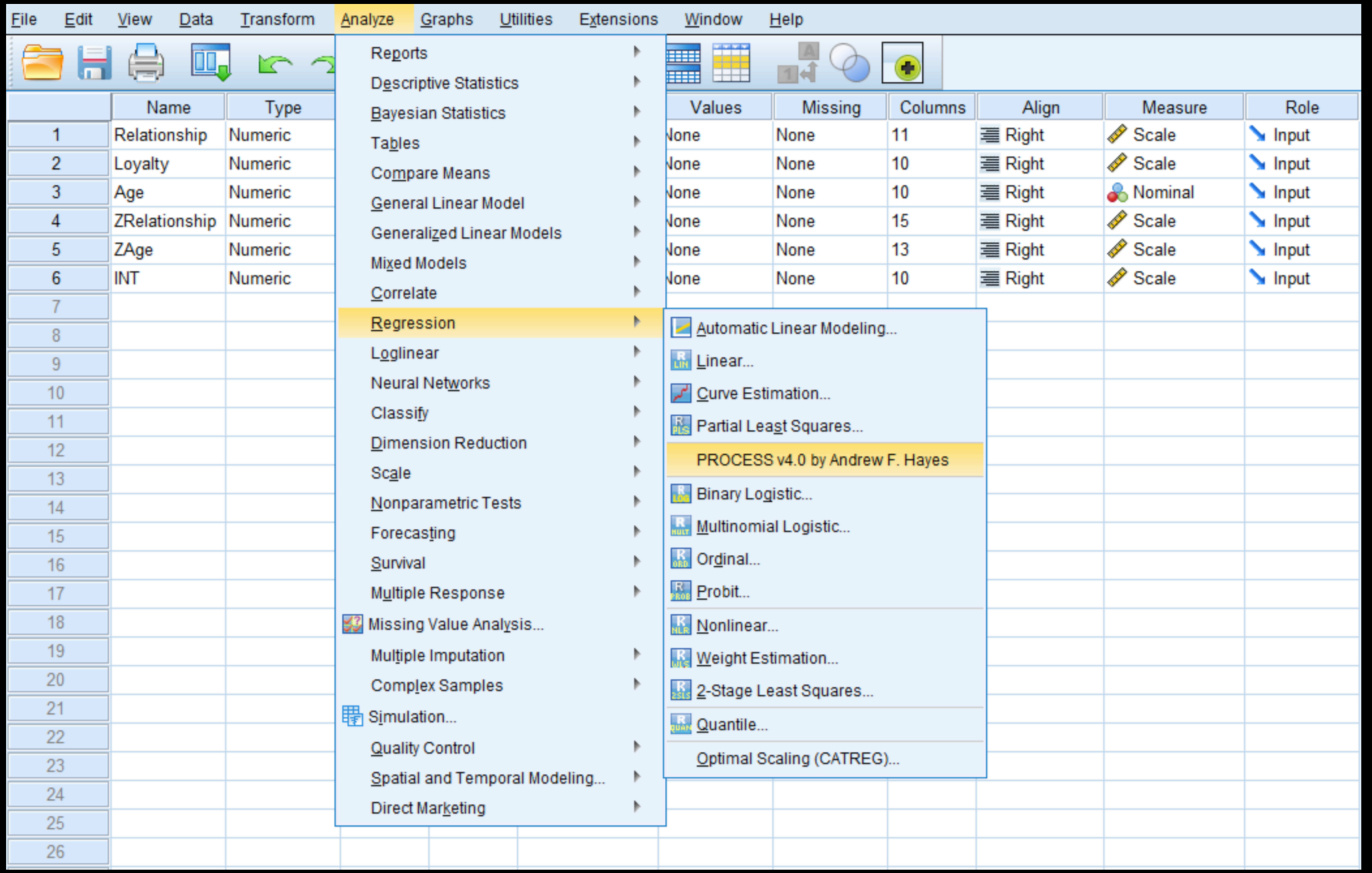
```
// Stan model 4 - relationship - duration associated with location x
data{
    int N;                      // number of observations
    int M;                      // number of locations
    array[N] real<lower=0> y; // observed durations
    array[N] int x;             // location (categorical)
}
parameters{
    array[M] real<lower=0> avg_duration;
}
model{
    y ~ exponential(avg_duration[x]^(-1));
}
```

relationship between duration and location

```
// Stan model 4 - relationship - duration associated with location x
data{
    int N;                      // number of observations
    int M;                      // number of locations
    array[N] real<lower=0> y; // observed durations
    array[N] int x;             // location (categorical)
}
parameters{
    array[M] real<lower=0> avg_duration;
}
model{
    y ~ exponential(avg_duration[x]^(-1));
}
```

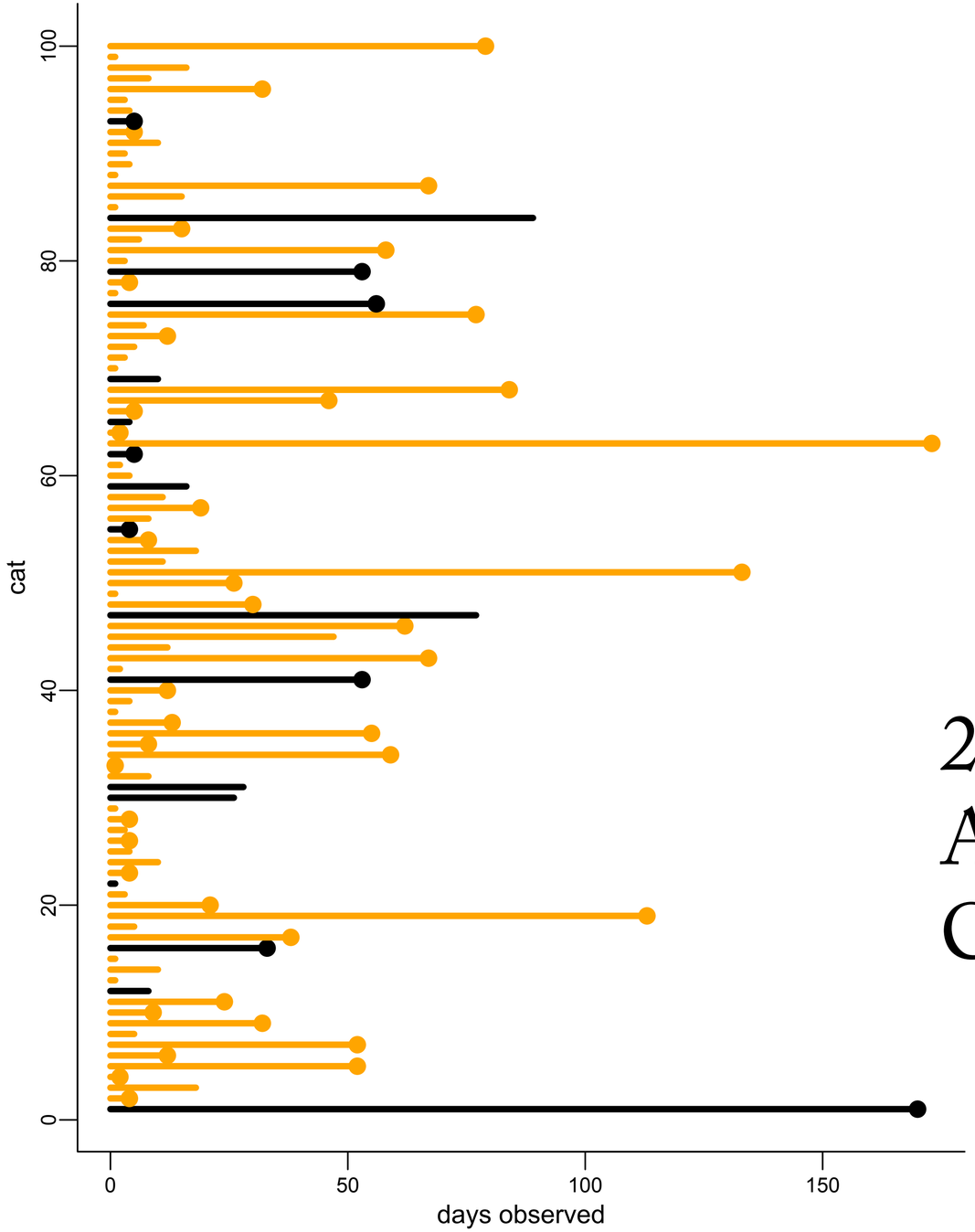


Pause for Questions


File
Edit
View
Data
Transform
Analyze
Graphs
Utilities
Extensions
Window
Help
Reports
Descriptive Statistics
Bayesian Statistics
Tables
Compare Means
General Linear Model
Generalized Linear Models
Mixed Models
Correlate
Regression
Loglinear
Neural Networks
Classify
Dimension Reduction
Scale
Nonparametric Tests
Forecasting
Survival
Multiple Response
Missing Value Analysis...
Multiple Imputation
Complex Samples
Simulation...
Quality Control
Spatial and Temporal Modeling...
Direct Marketing
Automatic Linear Modeling...
Linear...
Curve Estimation...
Partial Least Squares...
PROCESS v4.0 by Andrew F. Hayes
Binary Logistic...
Multinomial Logistic...
Ordinal...
Probit...
Nonlinear...
Weight Estimation...
2-Stage Least Squares...
Quantile...
Optimal Scaling (CATREG)...
Name
Type
Values
Missing
Columns
Align
Measure
Role
1 Relationship Numeric None None 11 Right Scale Input
2 Loyalty Numeric None None 10 Right Scale Input
3 Age Numeric None None 10 Right Nominal Input
4 ZRelationship Numeric None None 15 Right Scale Input
5 ZAge Numeric None None 13 Right Scale Input
6 INT Numeric None None 10 Right Scale Input

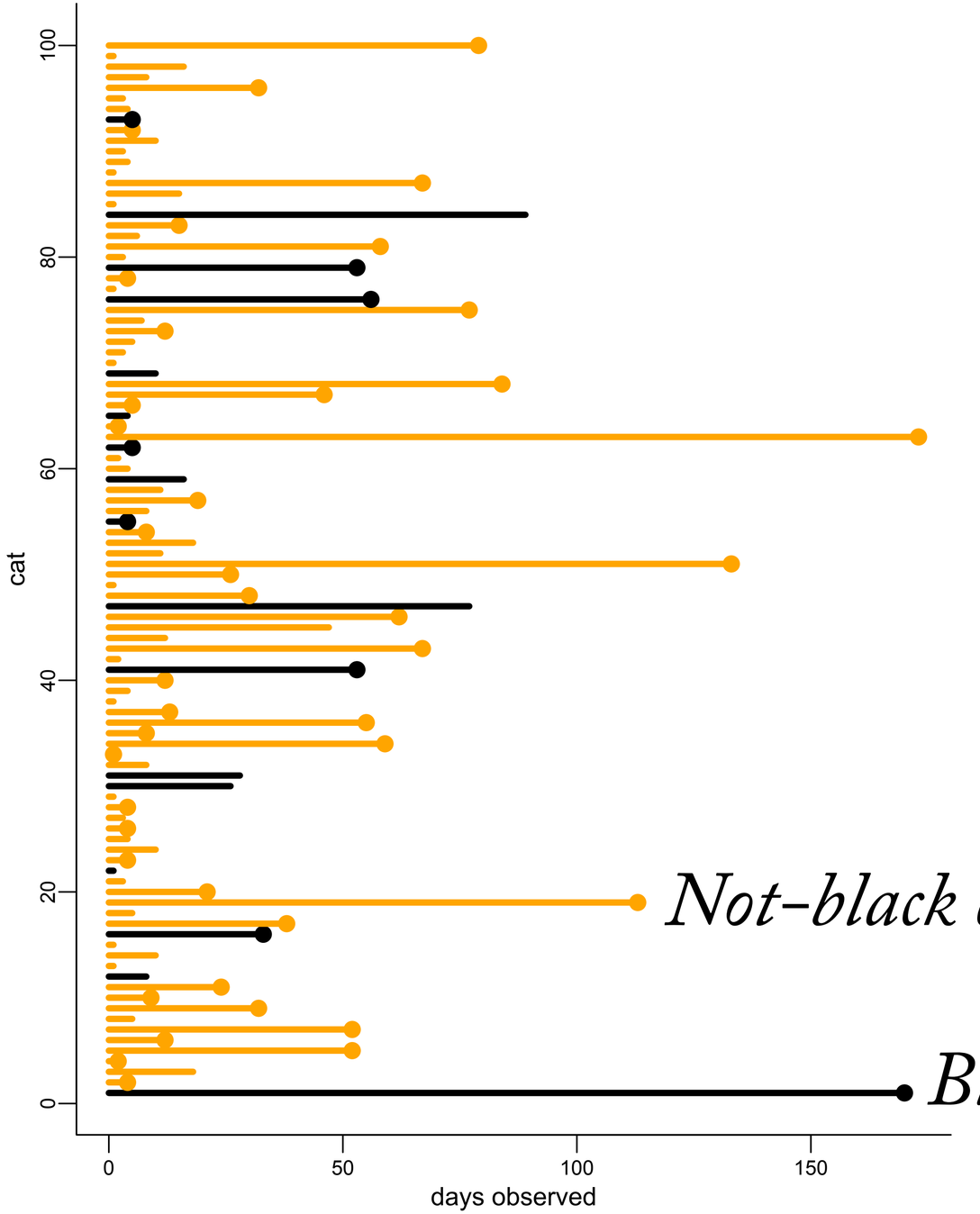
Generative modeling and scientific purpose

- The best way to write a probabilistic program is to stop and go write a different program
- Generative modeling: Explicitly connect scientific/mechanistic models of nature to inference algorithm
- Clarify goals, make logic of estimator transparent and testable



22,356 cats from the Austin, Texas Animal Center database

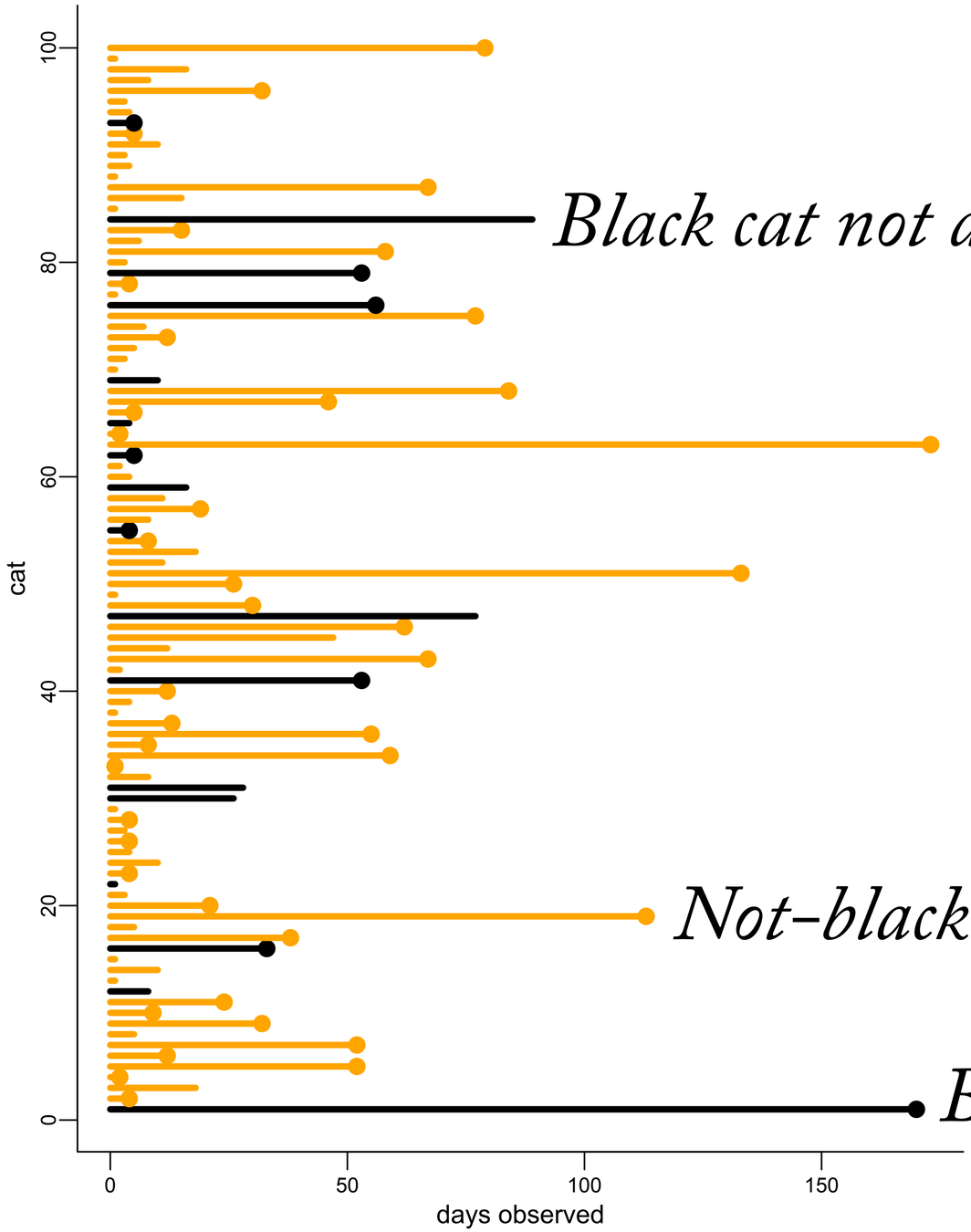
FRIENDS OF AUSTIN
— ANIMAL CENTER —



Not-black cat adopted

Black cat adopted

FRIENDS OF AUSTIN
— ANIMAL CENTER —



Black cat not adopted

Not-black cat adopted

Black cat adopted

FRIENDS OF AUSTIN
— ANIMAL CENTER —

Generative cat adoptions

- Assume daily probability of adoption depends upon color
- Probability wait D days until adoption: $p(1-p)^{D-1}$
- Use generative model to build statistical model and to test it

$$\Pr(D \mid p) = p(1-p)^{D-1}$$

$$\Pr(p) = \text{Beta}(1,10)$$

$$\Pr(p \mid D) \propto \Pr(D \mid p)\Pr(p)$$

Observed variables

```
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

Observed variables

Unobserved

```
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

Observed variables

Unobserved

Distributions and relationships

```
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

$\Pr(p)$

```
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

$\log \Pr(D|p)$

Defining the Joint Posterior: target

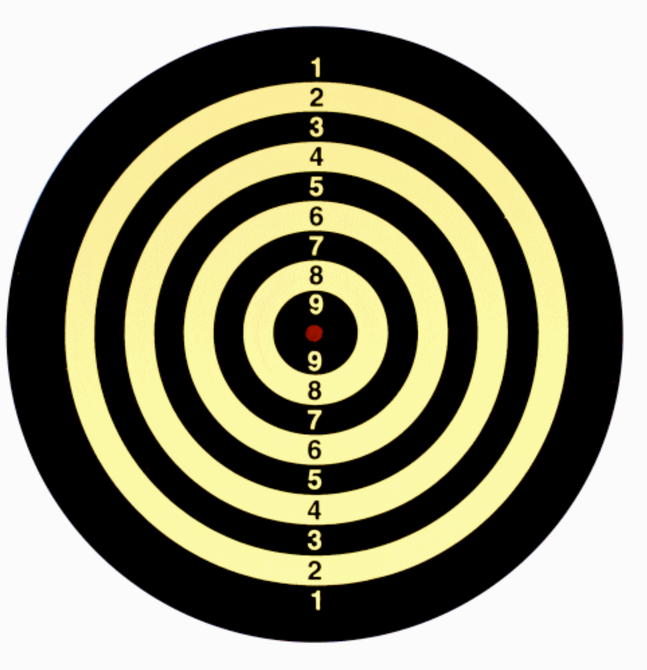
- Model block defines the numerator of Bayes formula:

$$\Pr(p \mid D, A) \propto \Pr(D, A \mid p) \Pr(p)$$

- We call the logarithm of this product the “target”

$$\text{target} = \log \Pr(D, A \mid p) + \log \Pr(p)$$

- Both target += and ~ add terms to the target



$\Pr(p)$

```
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

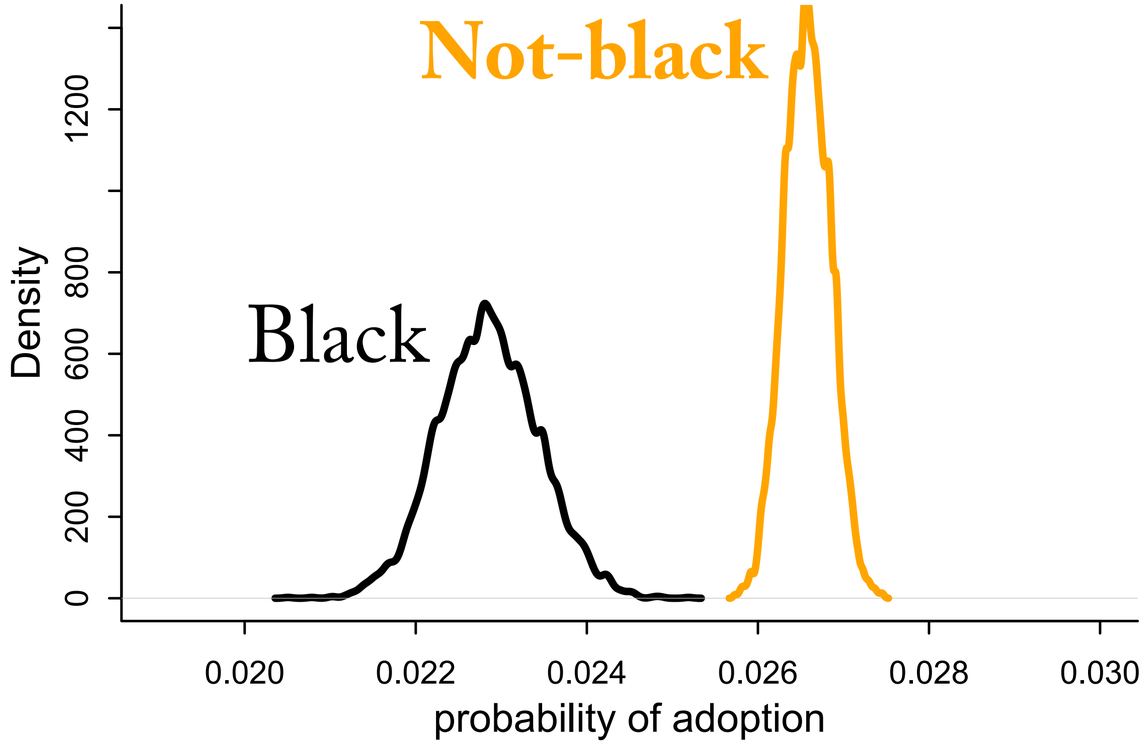
$\log \Pr(D|p)$

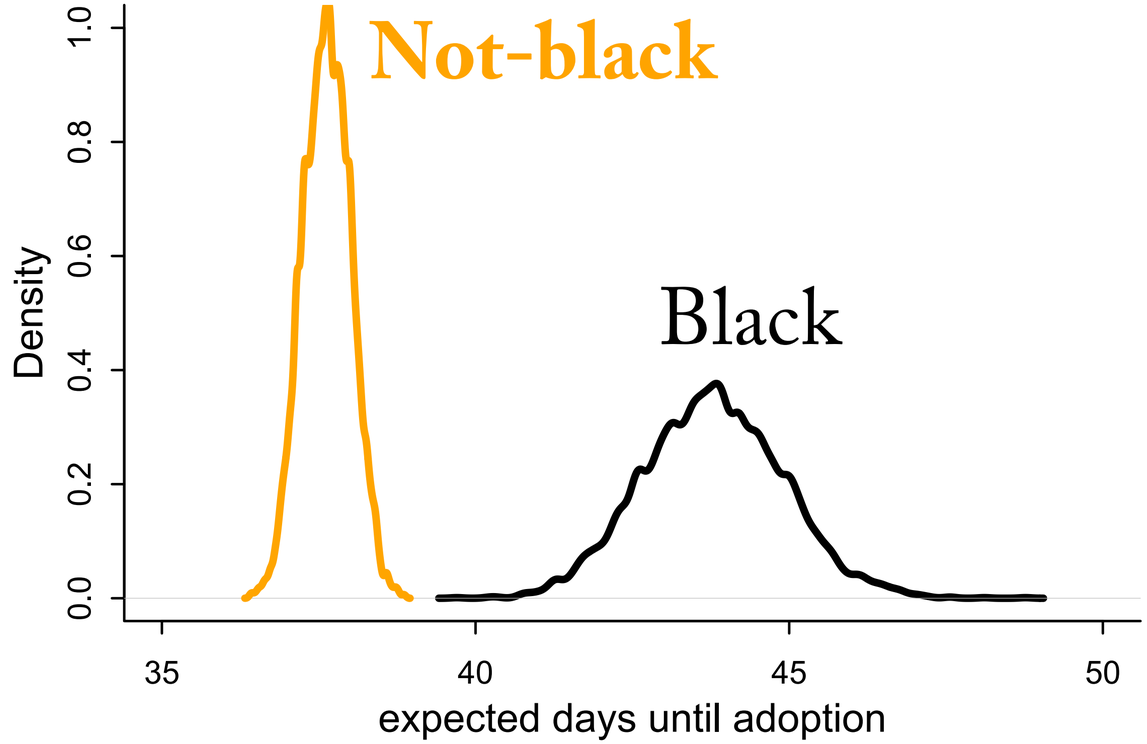
$\log \Pr(p)$

```
model{
    target += beta_lpdf(p|1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```

$\log \Pr(D|p)$

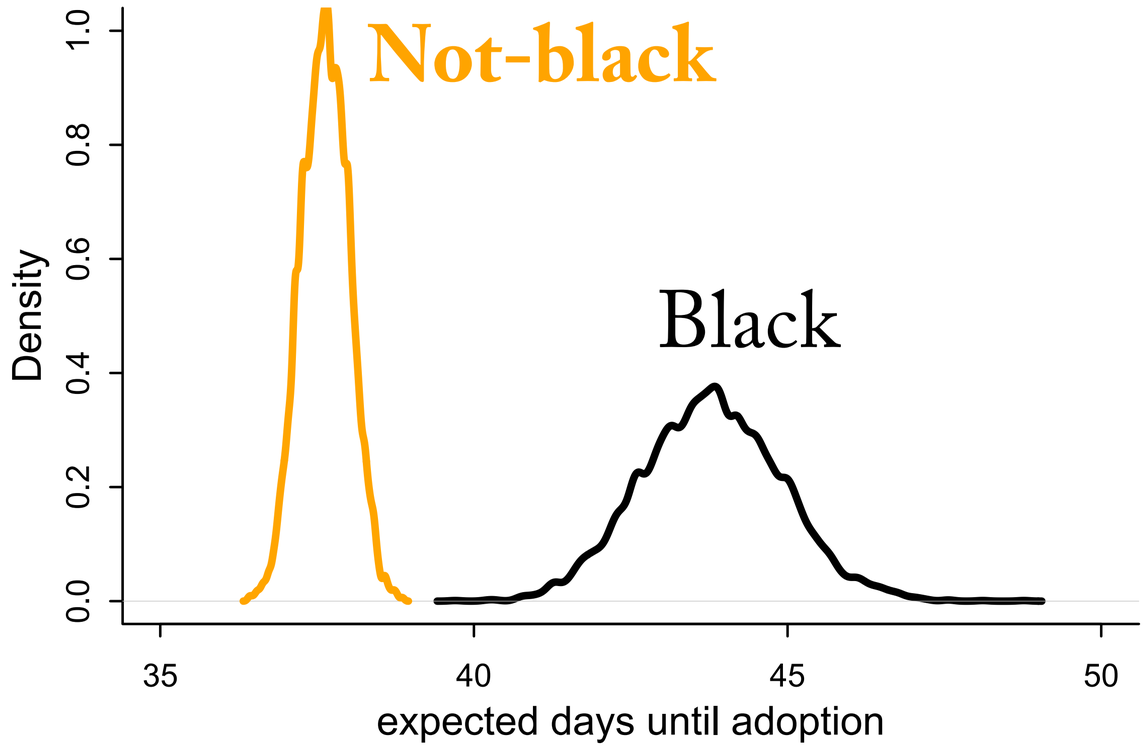
```
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            // something here
        }
    }
}
```



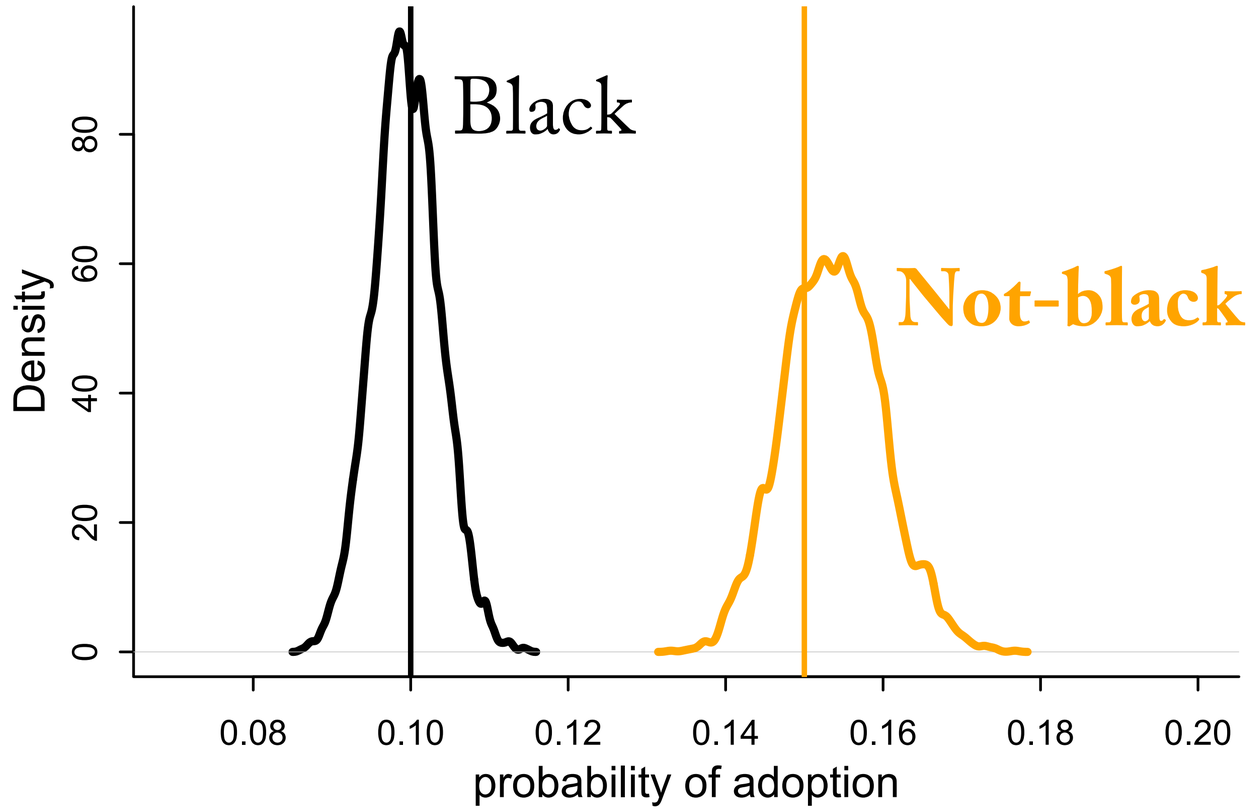


Black cat posterior

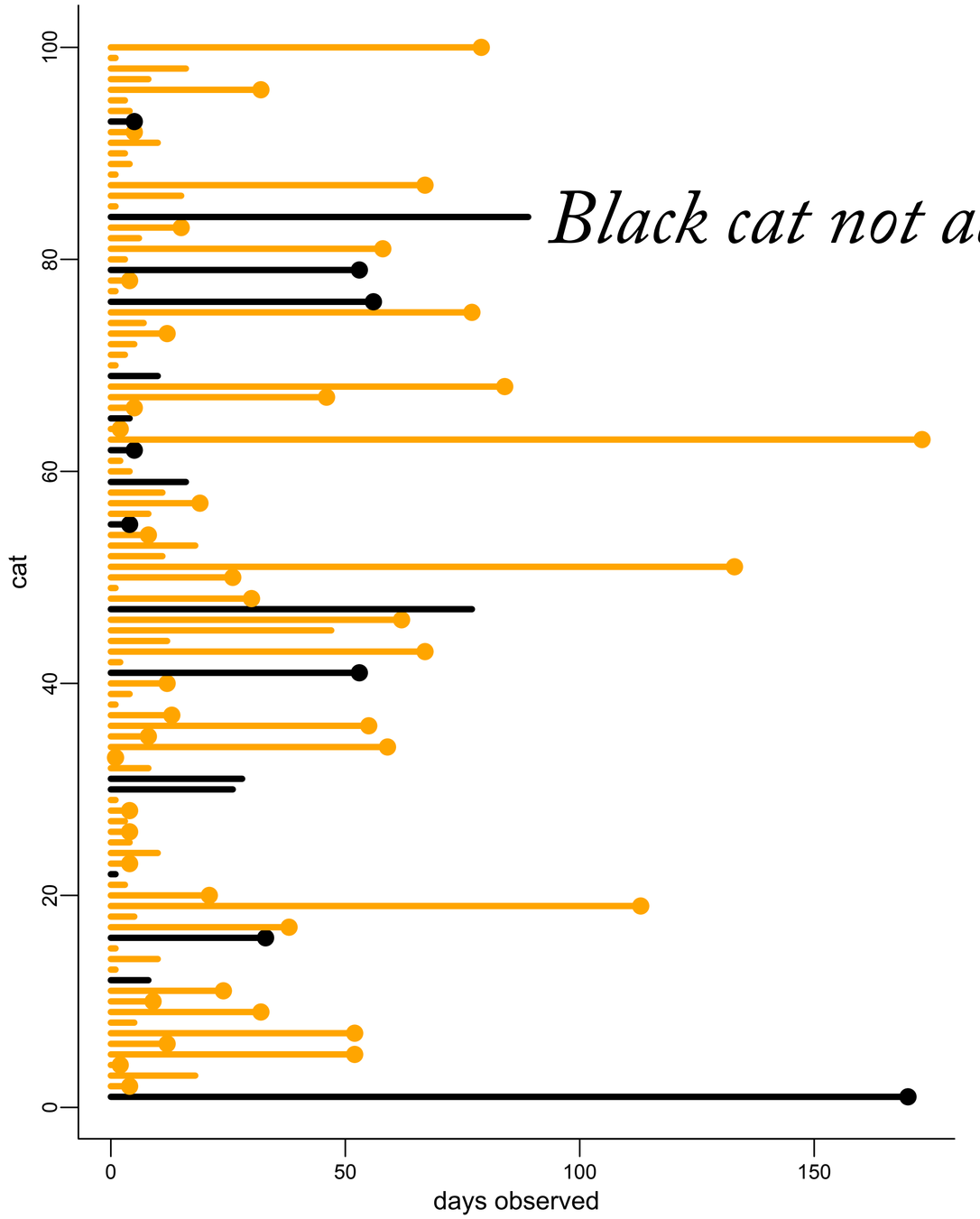
- We have estimates, but are they correct?
- Use generative model to simulate data for testing stat model
- Generative model: parameters in, data out
- Statistical model: Data in, parameters out



```
sim_cats1 <- function(n=1e3,p=c(0.1,0.2)) {
    color <- sample(c(1,2),size=n,replace=TRUE)
    days <- rgeom( n , p[color] ) + 1
    return(list(N=n,days=days,color=color,adopted=rep(1,n)))
}
```



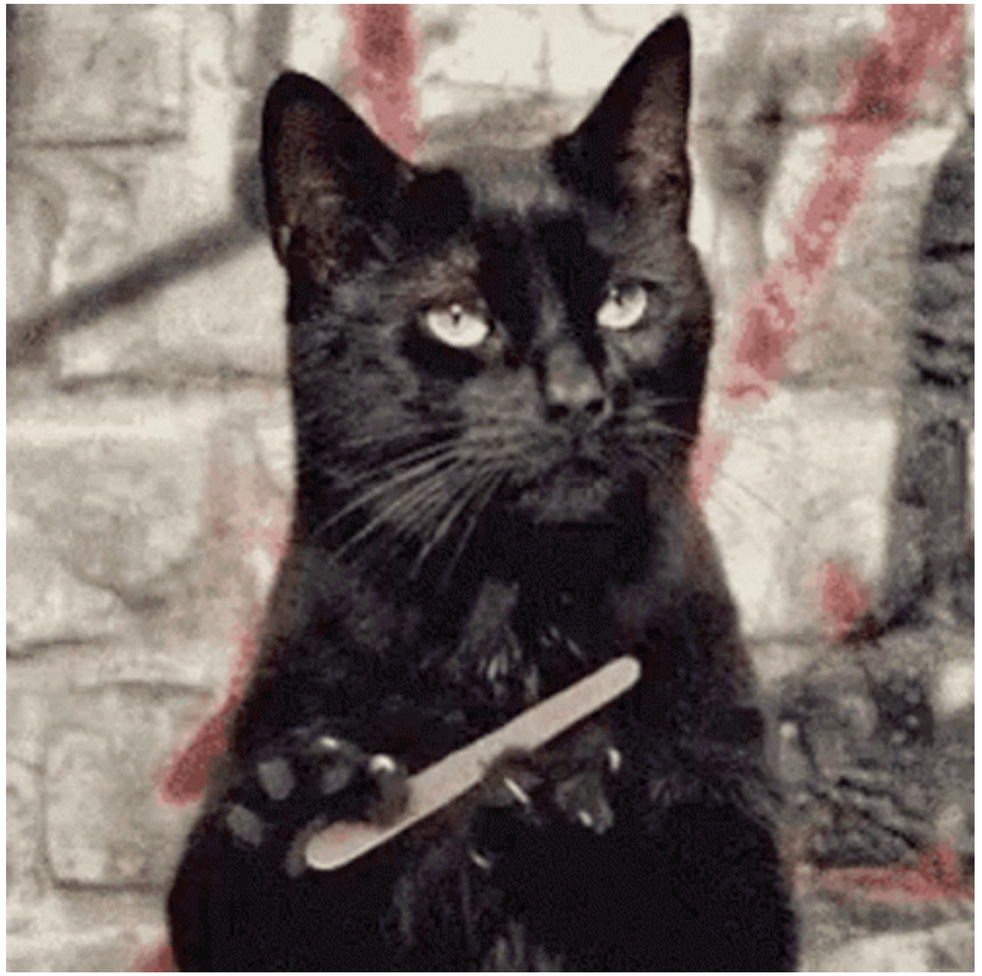
Modeling observation



Days waiting is information

Modeling Observation: Censoring

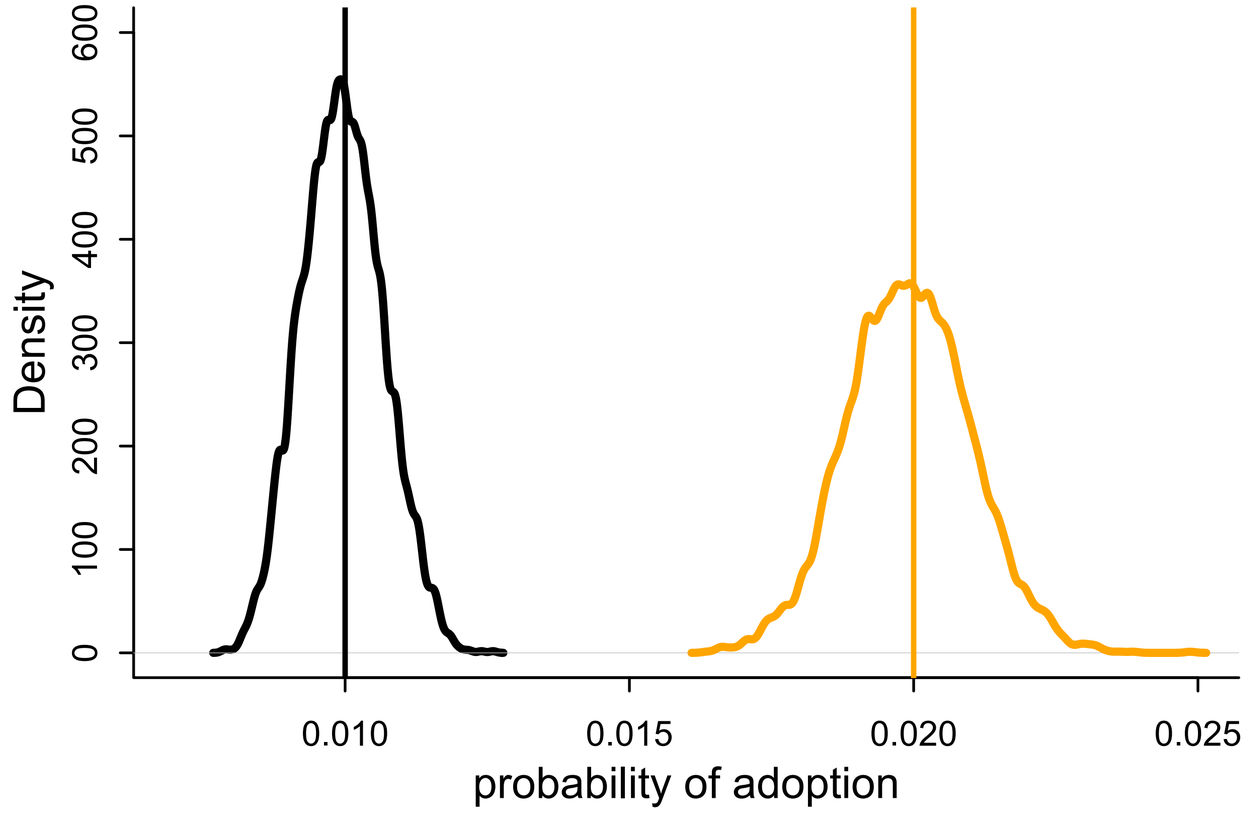
- Half the cats were not (yet) adopted
- How to use these records?
- Waited D days so far and no one has adopted you? Don't give up hope!
- $\Pr(D, A = 0 \mid p) = (1 - p)^D$



```
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            target += log( (1-P)^days[i] );
        }
    }
}
```

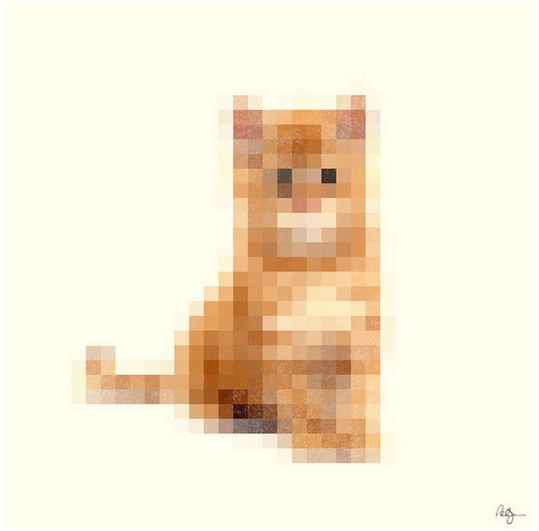
$\Pr(D, A=0 \mid p)$

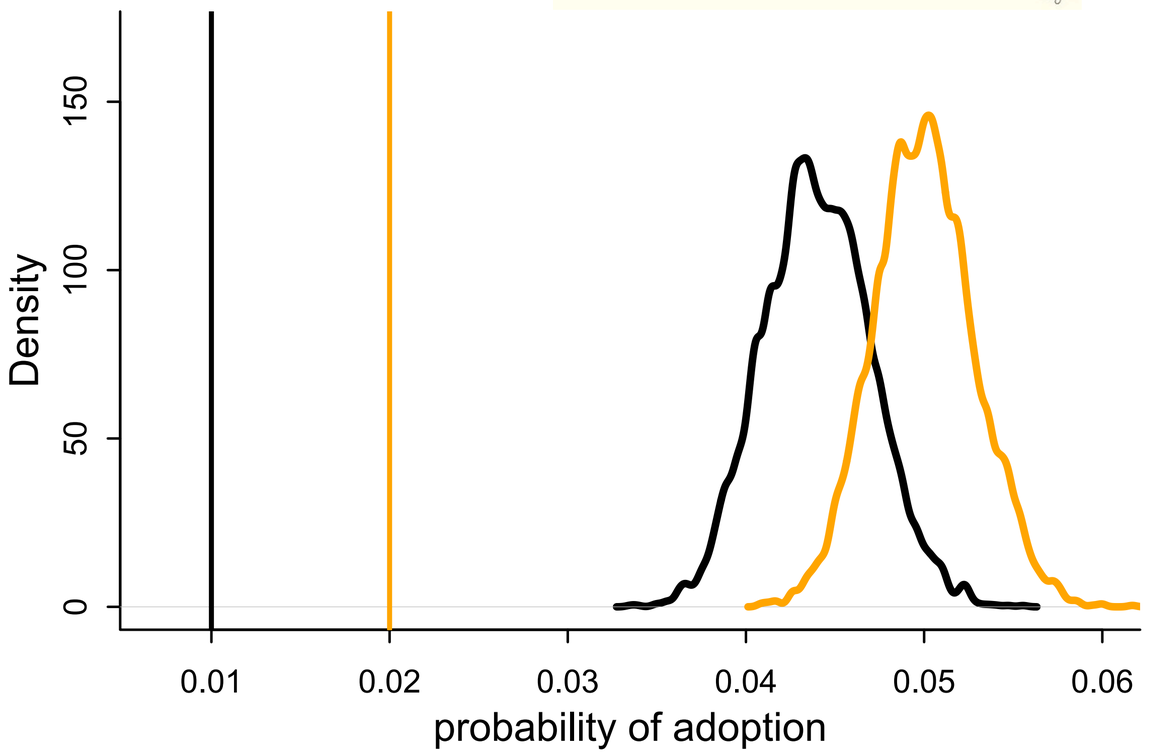
```
sim_cats2 <- function(n=1e3,p=c(0.01,0.02),cens=50) {
    color <- sample(c(1,2),size=n,replace=TRUE)
    days <- rgeom( n , p[color] ) + 1
    adopted <- ifelse( days < cens , 1 , 0 )
    days <- ifelse( adopted==1 , days , cens )
    return(list(N=n,days=days,color=color,adopted=adopted))
}
```



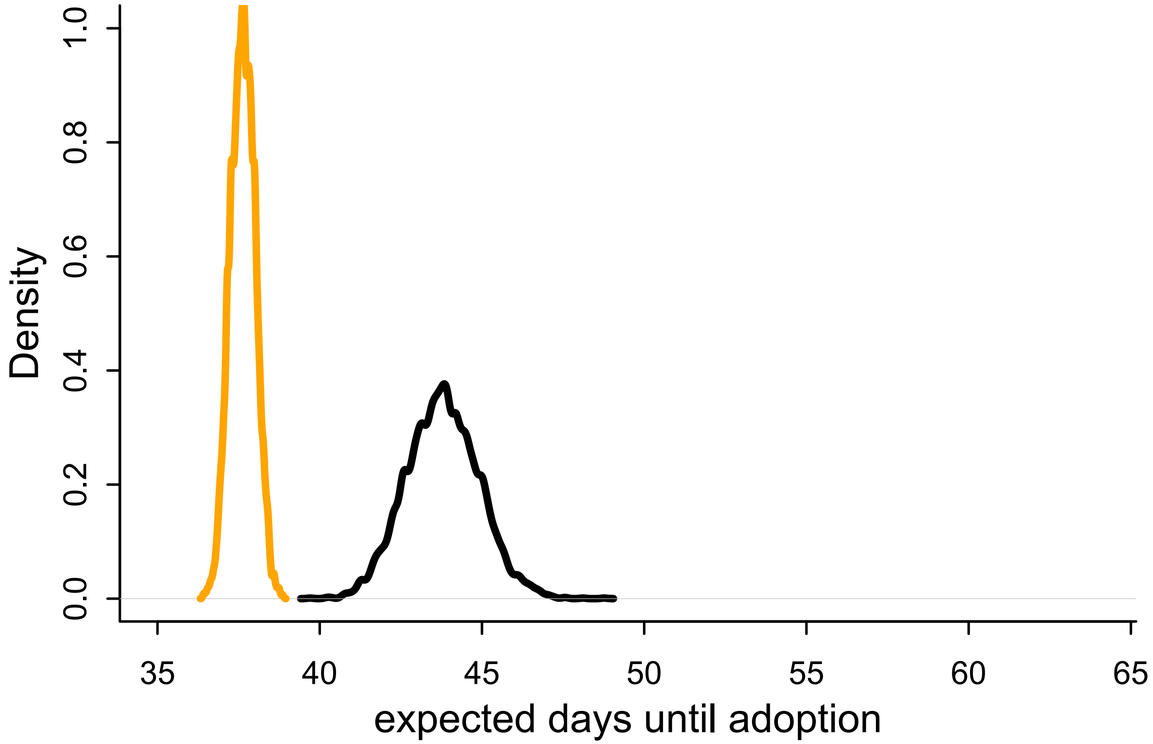
Model mis-specification

- Simulated data with censoring
- Model that ignores censored cats does a very bad job
- Why? Ignoring cats who aren't (yet) adopted biases estimate upwards — like counting successes and ignoring failures

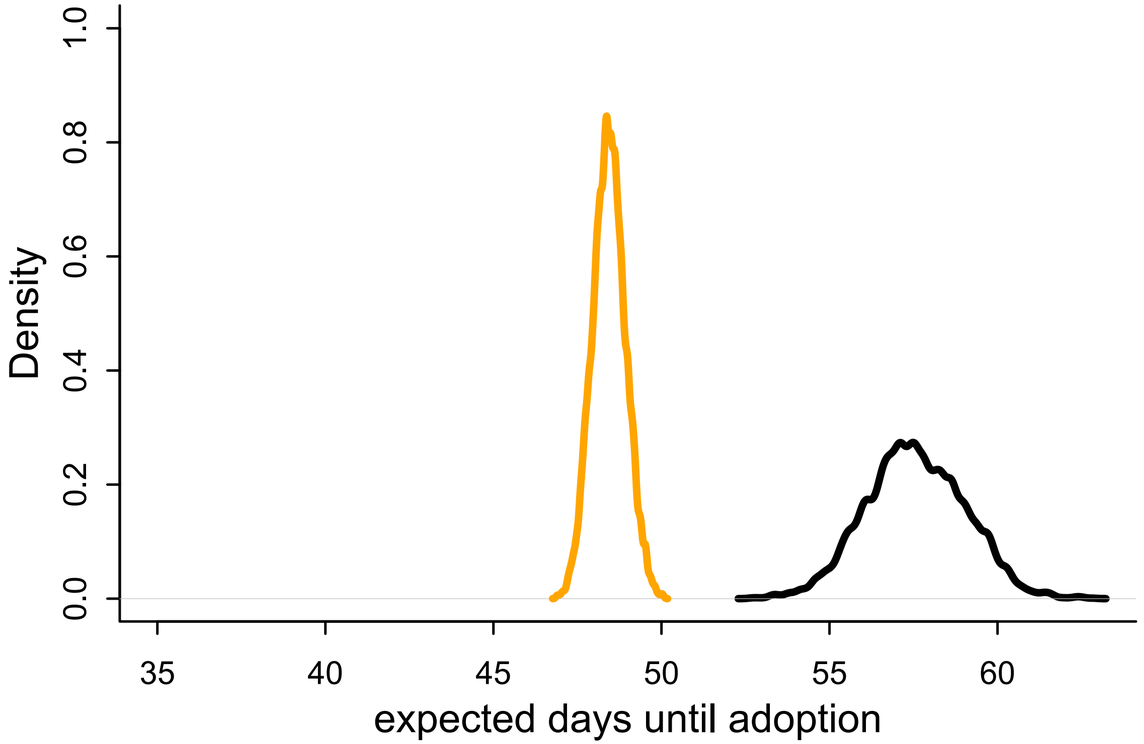




Ignore censoring



Model censoring



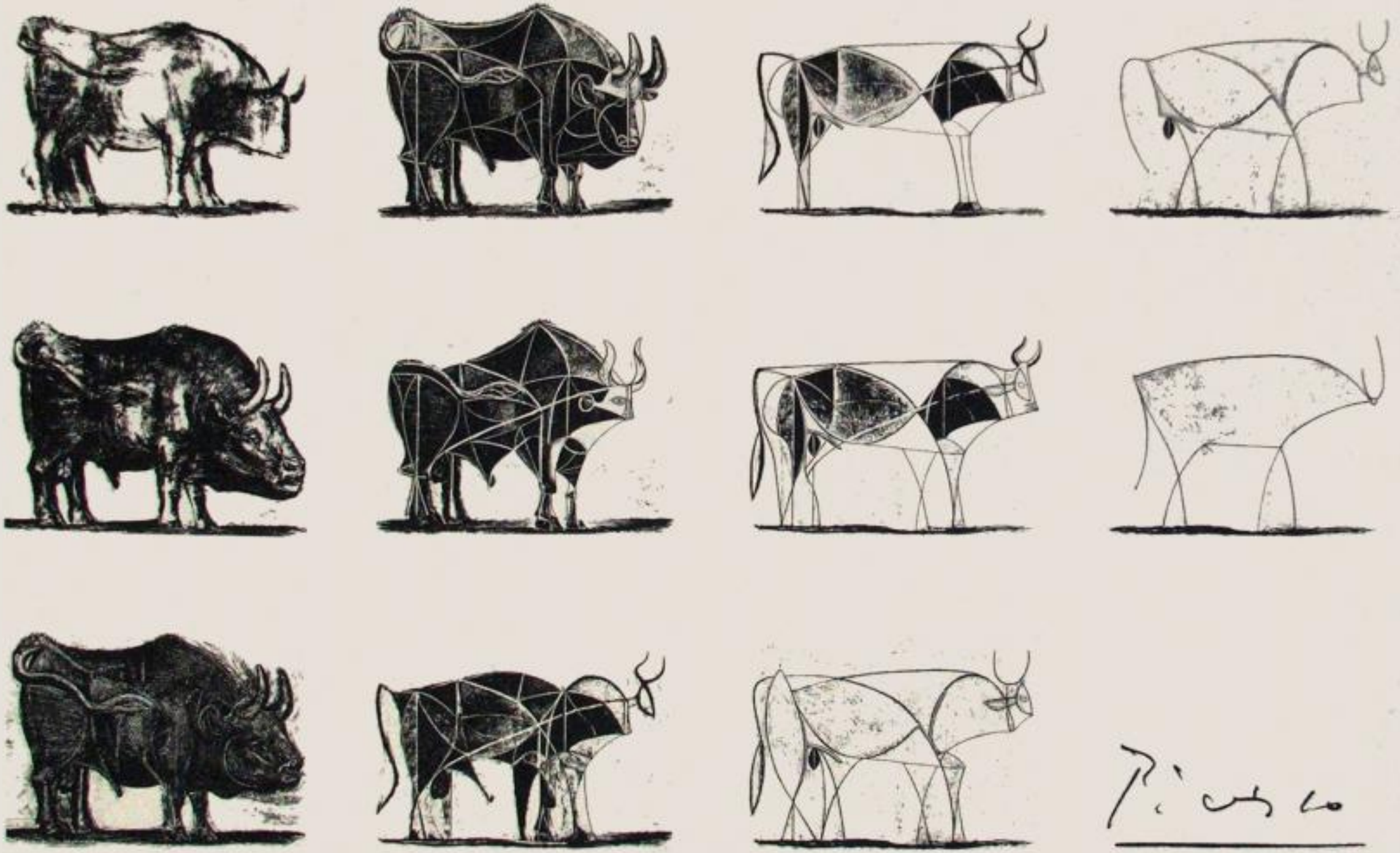
Observed variables

Unobserved

Distributions and relationships

```
// all events, including censored
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            target += log( (1-P)^days[i] );
        }
    }
}
```

Pause for Questions

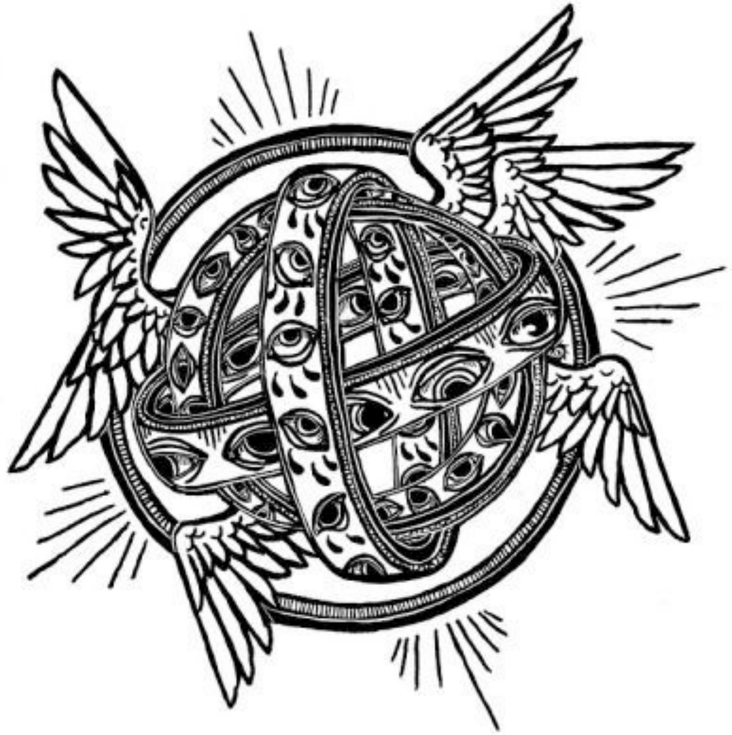


Bayes is Not Perfect (Either)

Idealized Bayesian

Bayes is Not Perfect (Either)

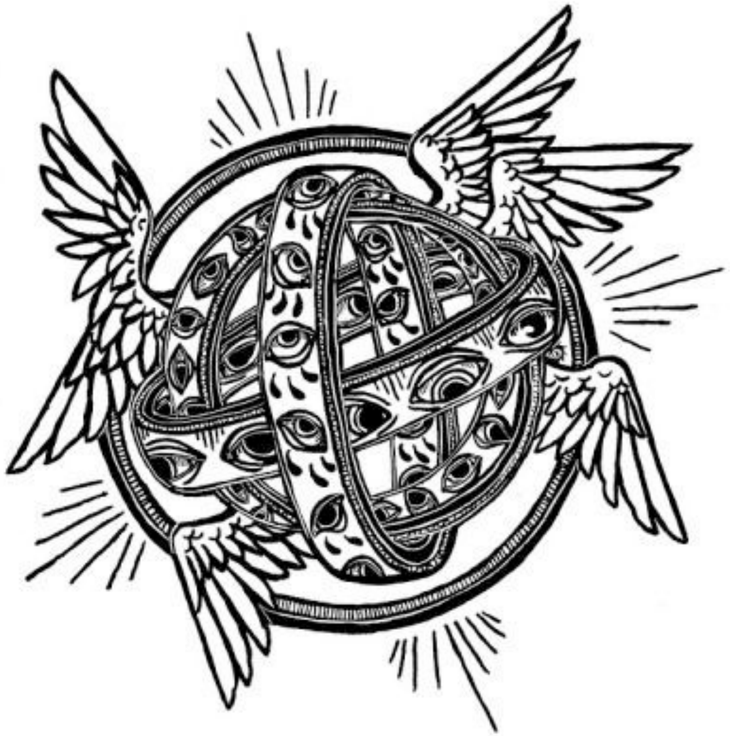
Idealized Bayesian

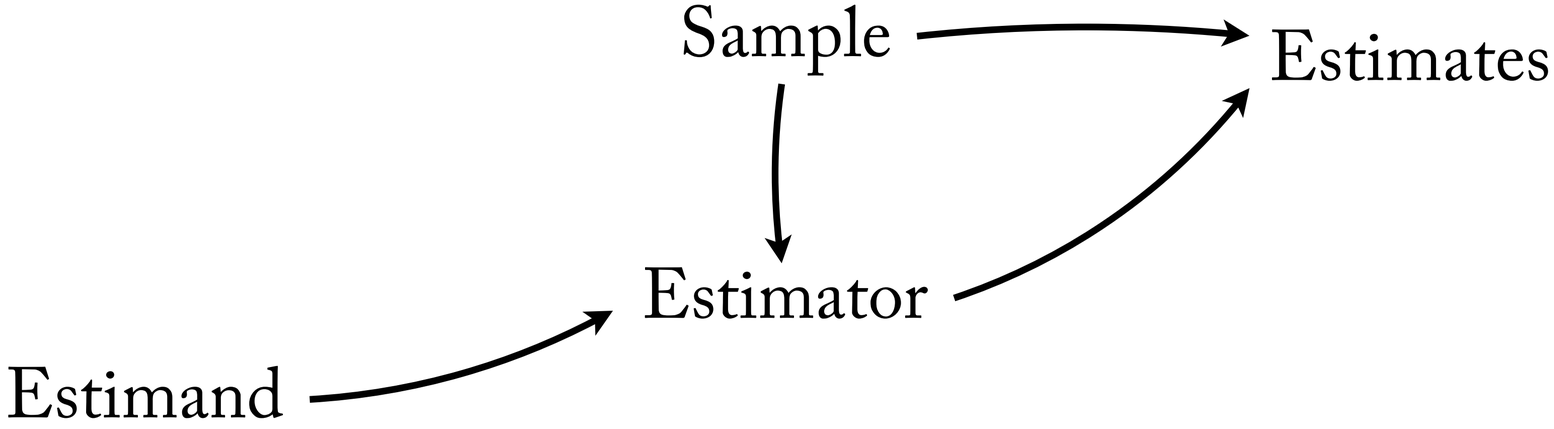


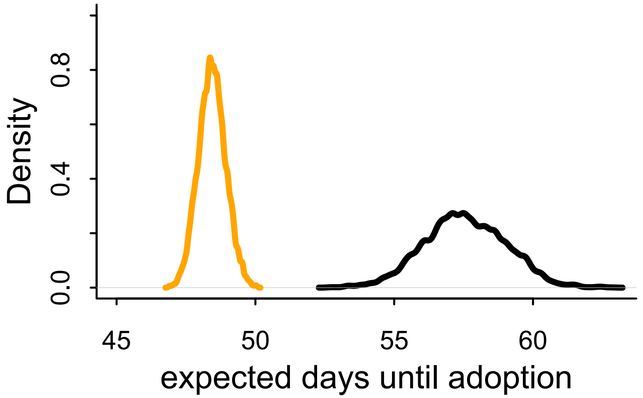
Biblically accurate Bayesian

Workflow: Reasonable & Reliable

- **Being smart requires working smart**
- Probabilistic programming workflow
  - Transparent components
  - Logical connections
  - Incremental steps
  - Incremental testing




Sample
Estimates
Estimator
Estimand



Sample → Estimates

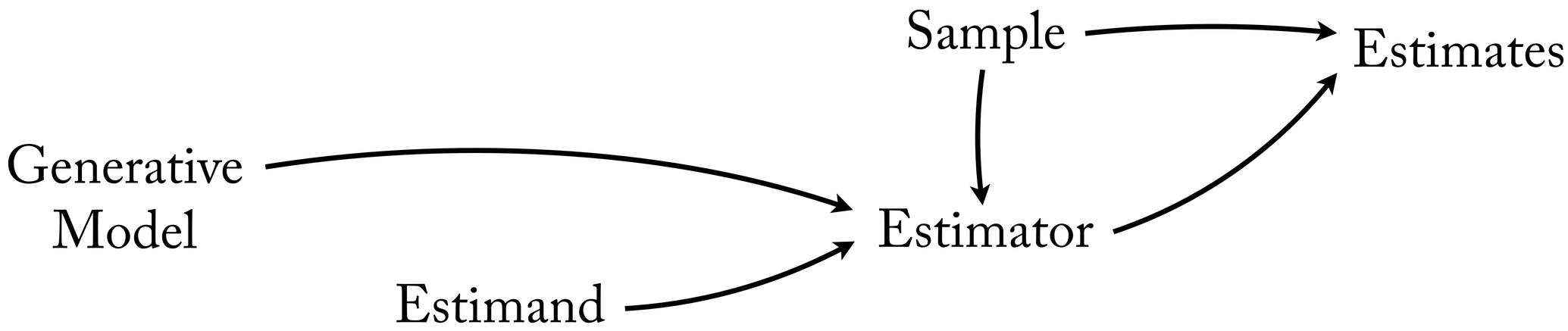
Sample → Estimator → Estimates

Estimand → Estimator

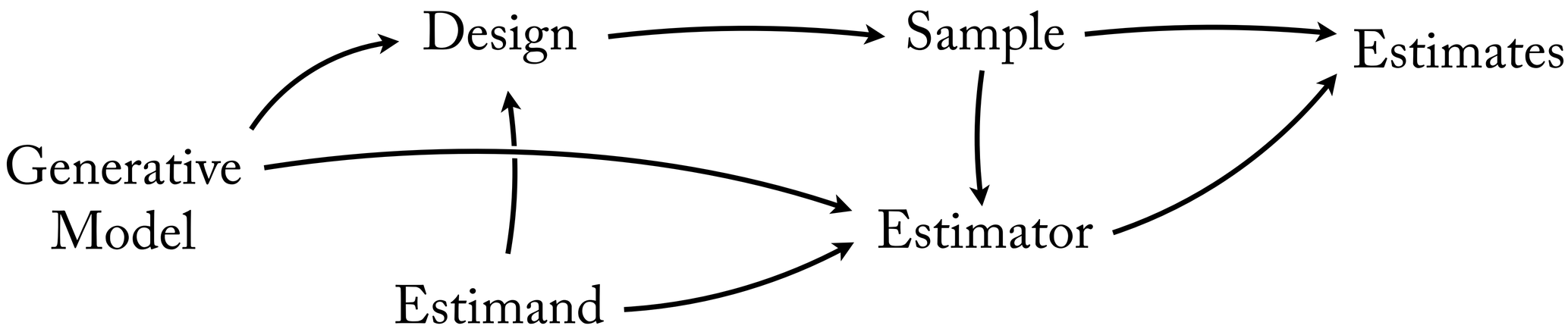
"Are black cats adopted slower than other cat colors?

```
// all events, including censored
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            target += log( (1-P)^days[i] );
        }
    }
}
```

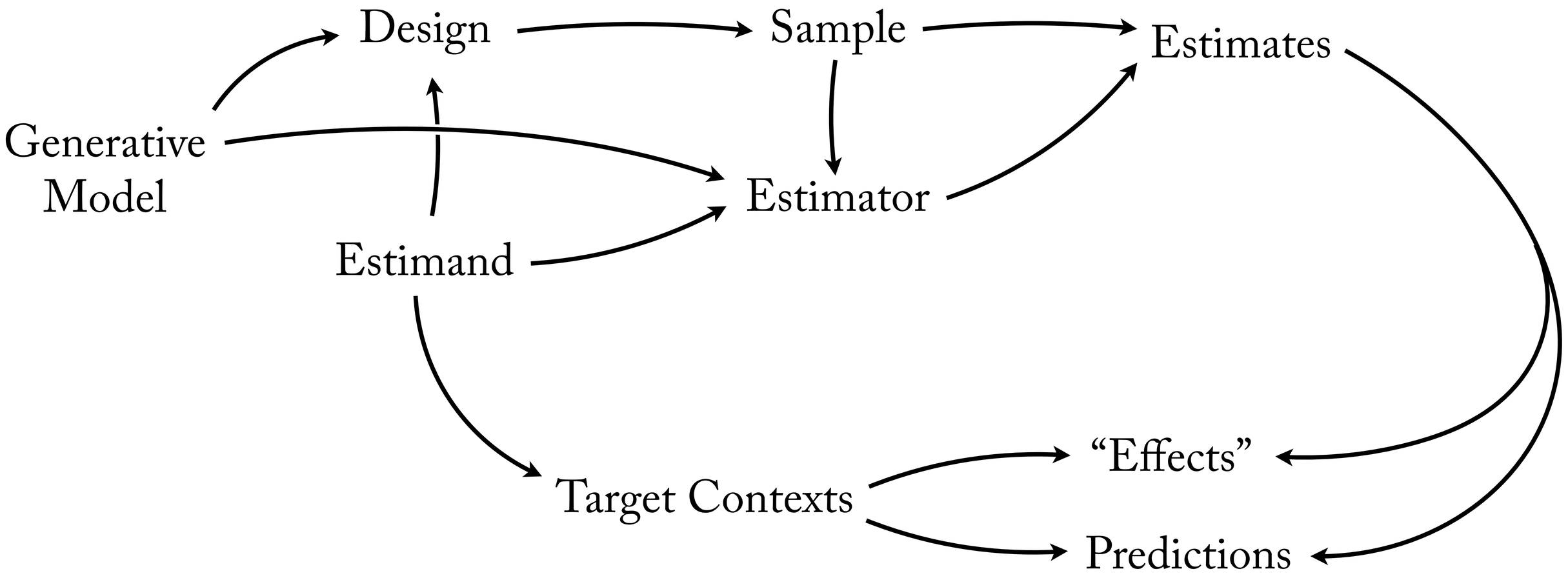
Better Workflow



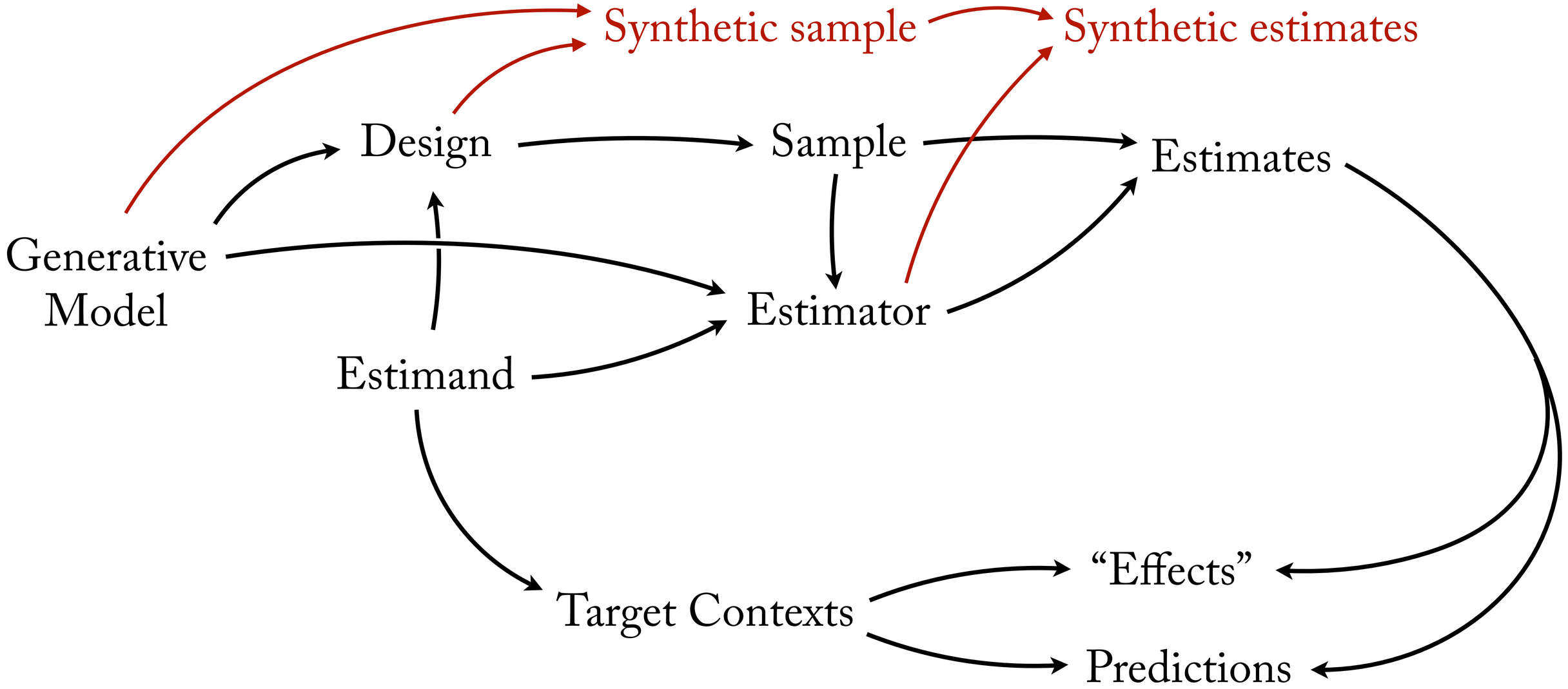
Better Workflow



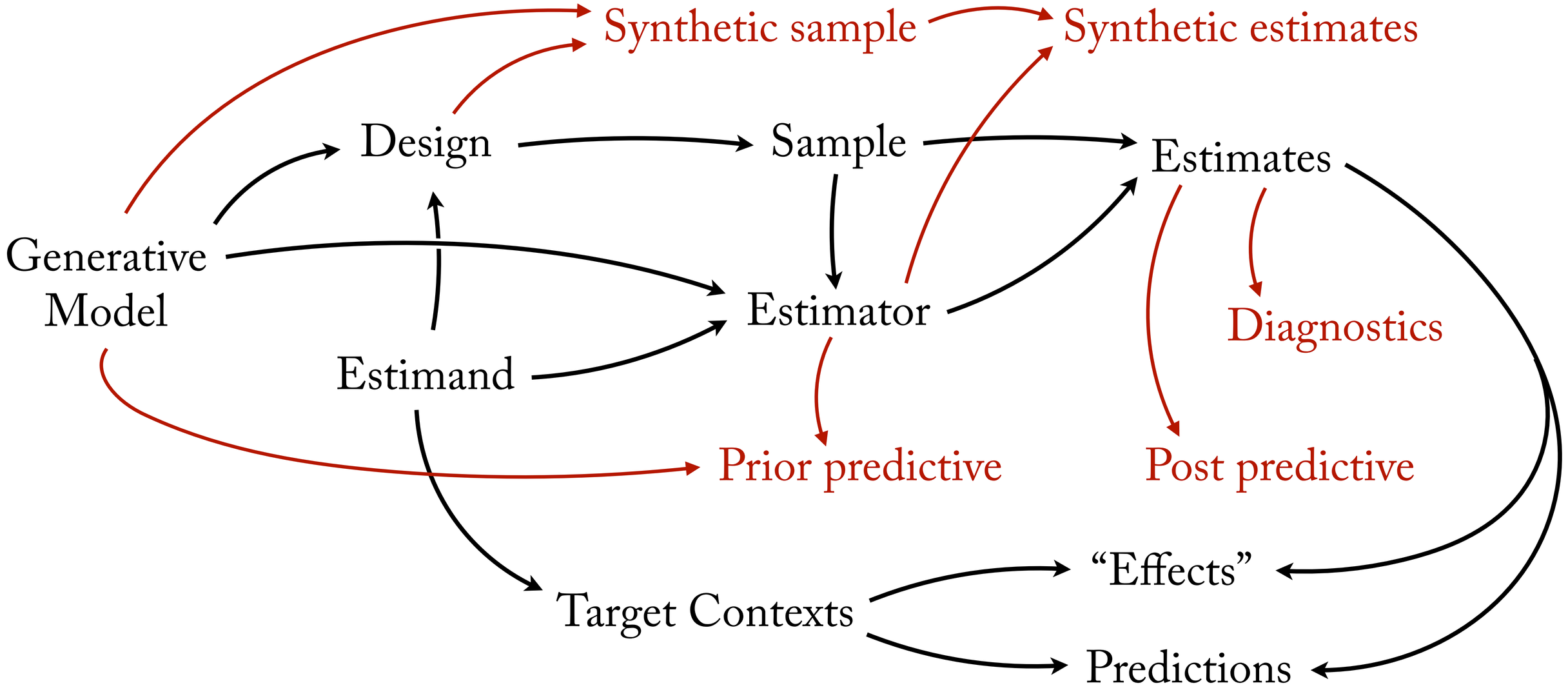
Better Workflow



Better Workflow

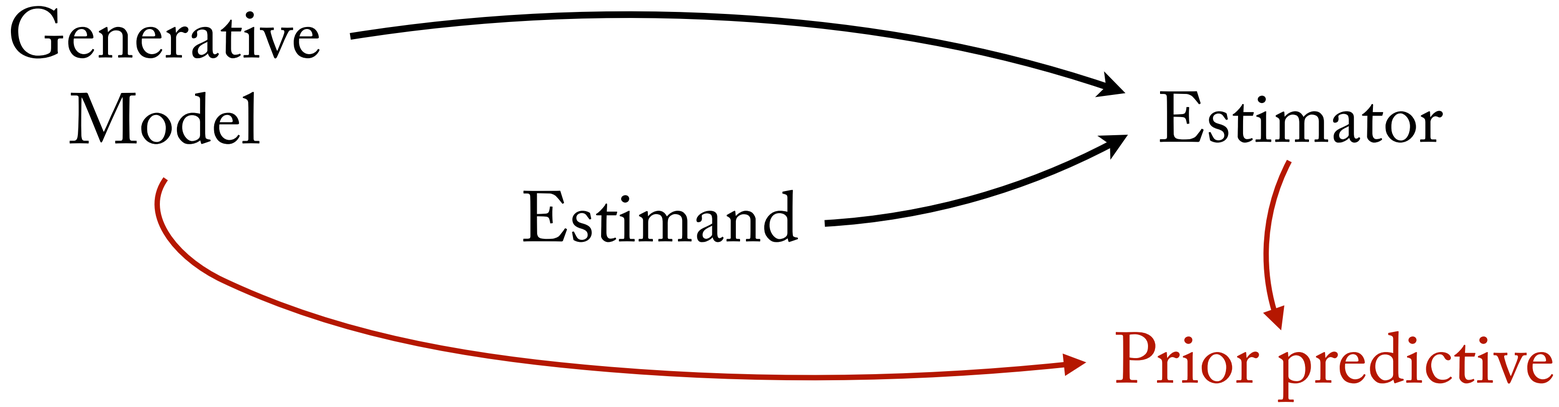


Better Workflow



Prior predictive distribution

"What does the model think, before it sees the data?"



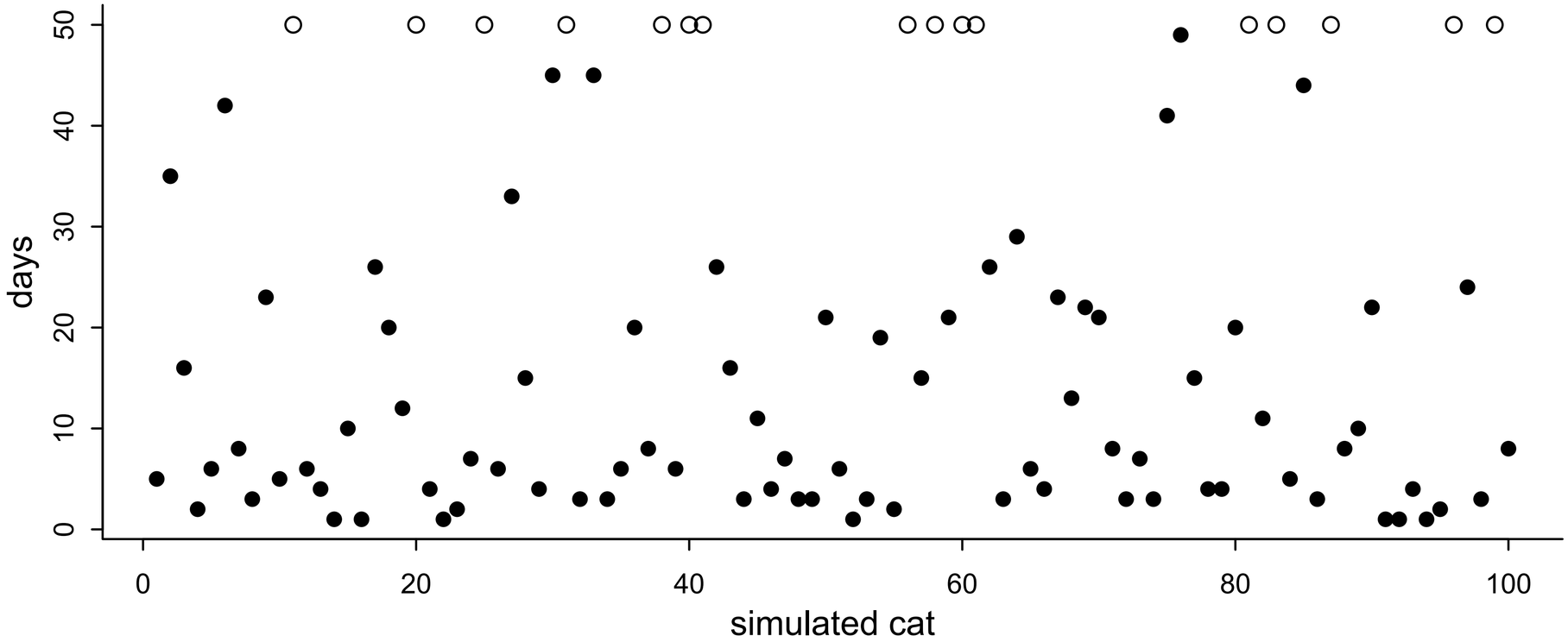
Prior predictive distribution

"What does the model think, before it sees the data?"

- (1) Sample variables from prior
- (2) Simulate observed variables
- (3) Repeat from (1)

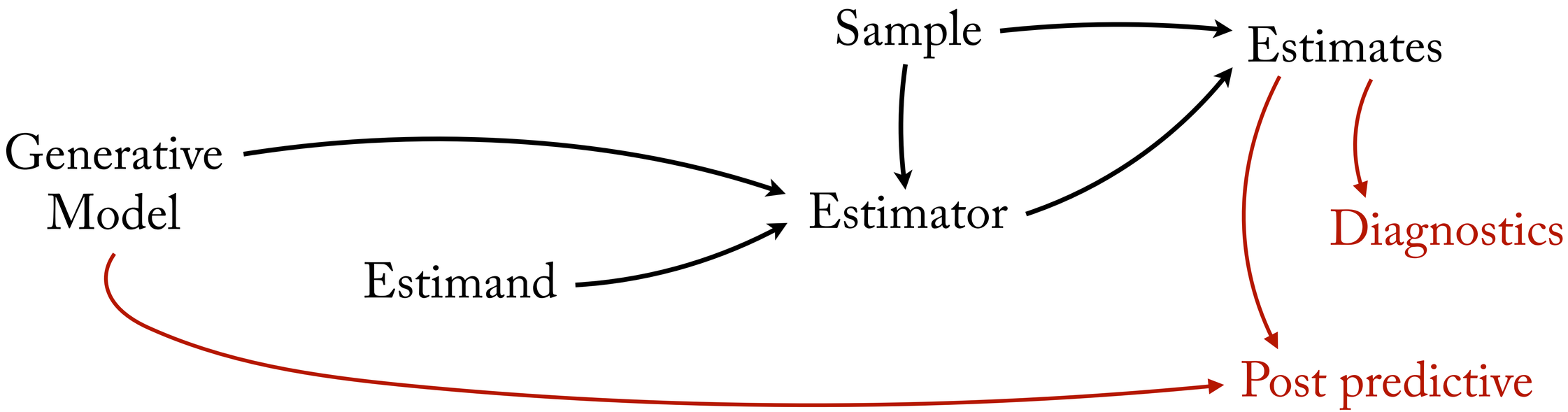
```
// all events, including censored
data{
    int N;
    array[N] int adopted; // 1/0 indicator
    array[N] int days;    // days until event
    array[N] int color;   // 1=black, 2=other
}
parameters{
    vector<lower=0,upper=1>[2] p;
}
model{
    p ~ beta(1,10);
    for ( i in 1:N ) {
        real P = p[color[i]];
        if ( adopted[i]==1 ) {
            target += log( (1-P)^(days[i]-1) * P );
        } else {
            target += log( (1-P)^days[i] );
        }
    }
}
```

```r
## prior samples
n <- 100
p1 <- rbeta(n,1,10)
p2 <- rbeta(n,1,10)

sim_cats2 <- function(n=1e3,p=c(0.01,0.02),cens=50) {
    color <- sample(c(1,2),size=n,replace=TRUE)
    days <- rgeom( n , p[color] ) + 1
    adopted <- ifelse( days < cens , 1 , 0 )
    days <- ifelse( adopted==1 , days , cens )
    return(list(N=n,days=days,color=color,adopted=adopted))
}

prior_days <- sapply( 1:n , function(i) sim_cats2(1,p=c(p1[i],p2[i]))$days )
```



Posterior predictive distribution

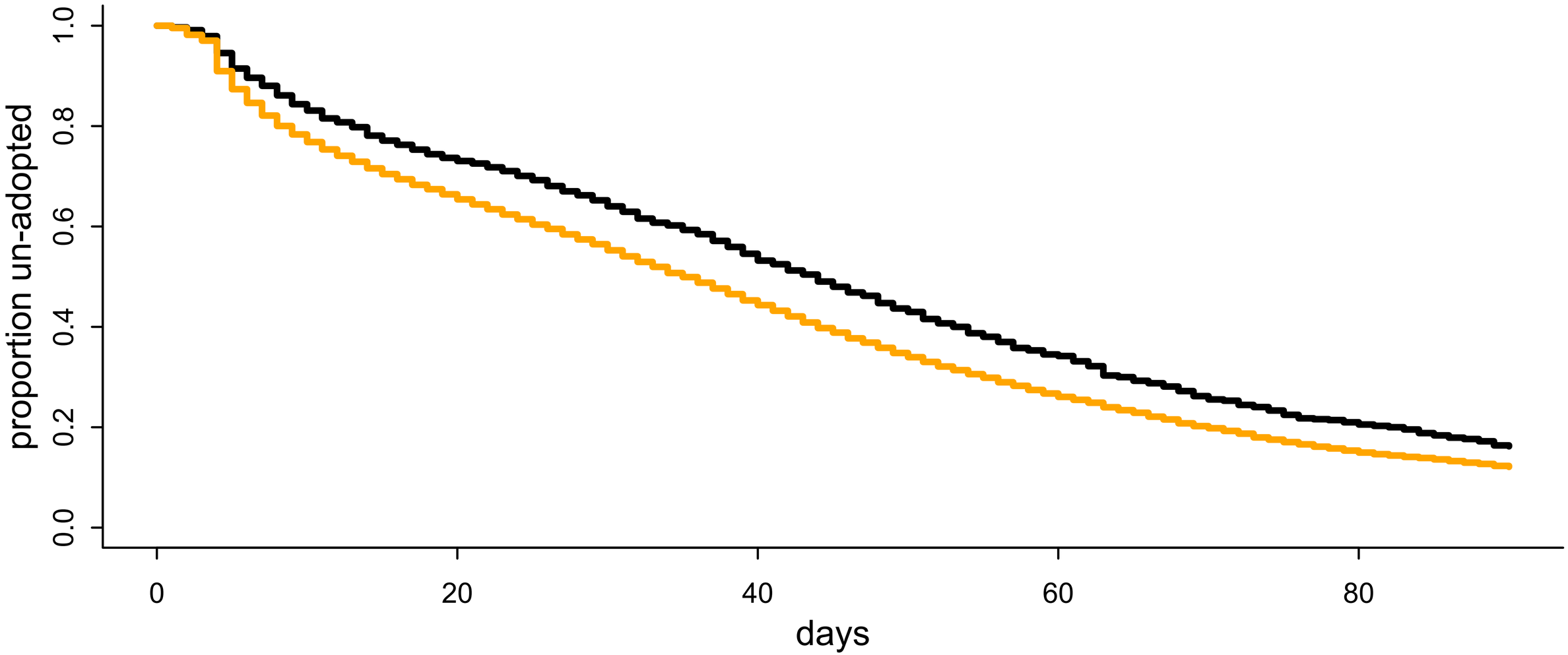
"What does the model think, after it sees the data?"

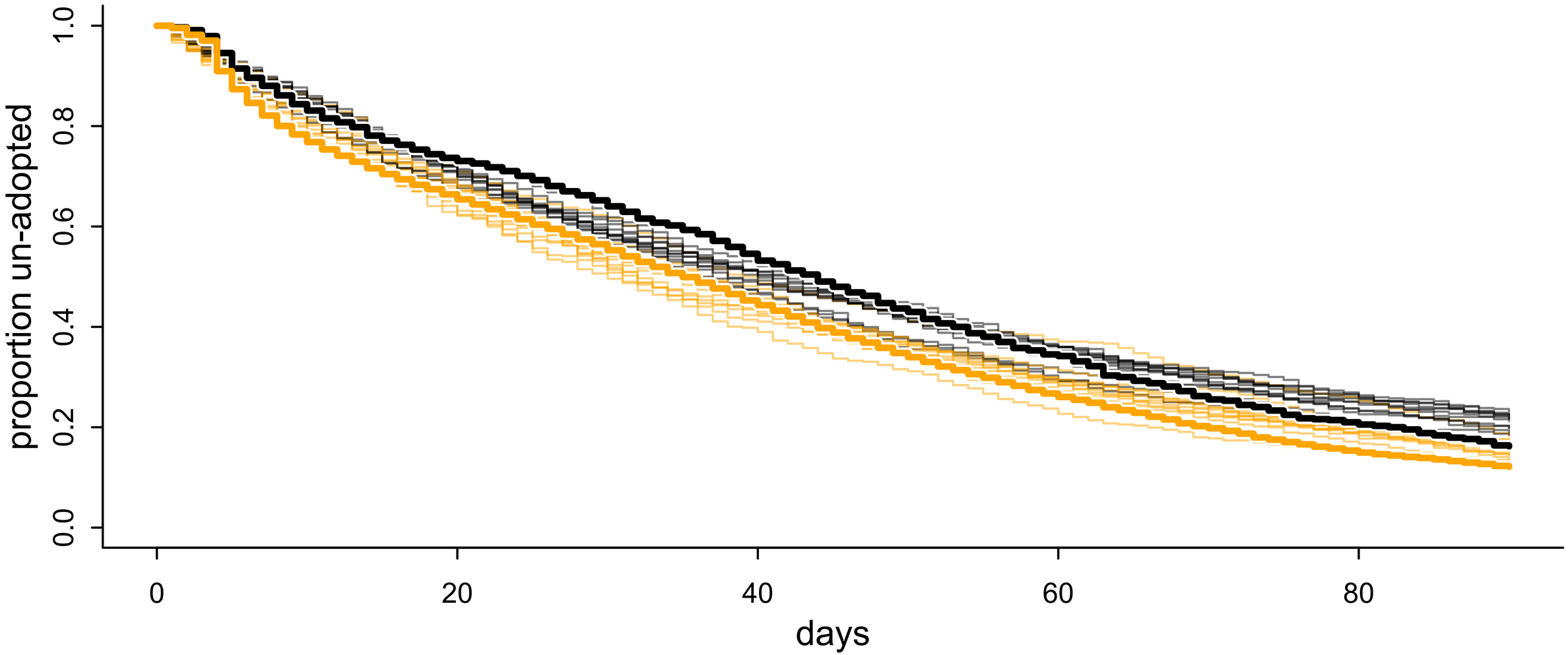


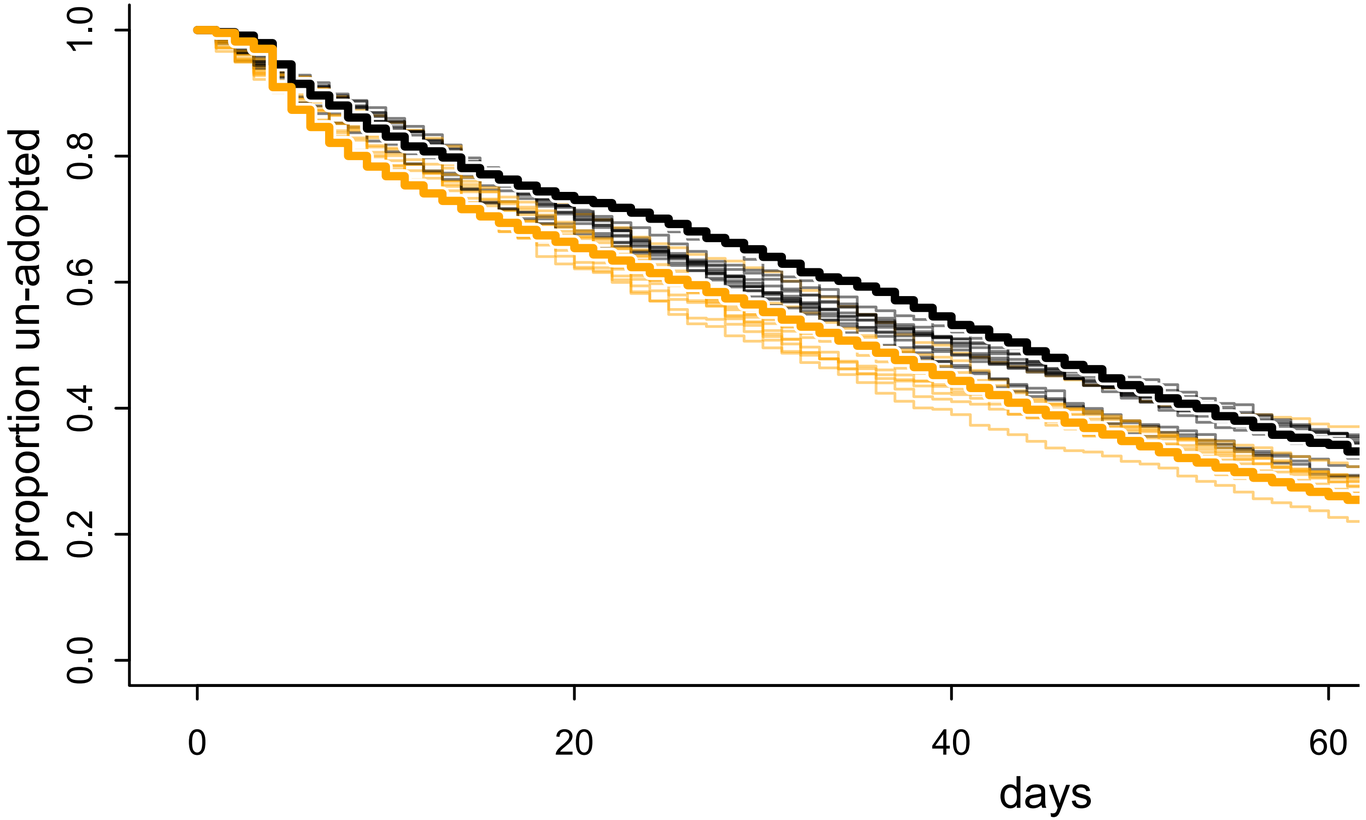
Posterior predictive distribution

"What does the model think, after it sees the data?"

- (1) Sample from posterior distribution
- (2) Simulate outcomes (for each observed case)
- (3) Assess fit, compare with other estimators
- Problem in this example: Censoring
- Can compare Kaplan-Meier curves to simulated

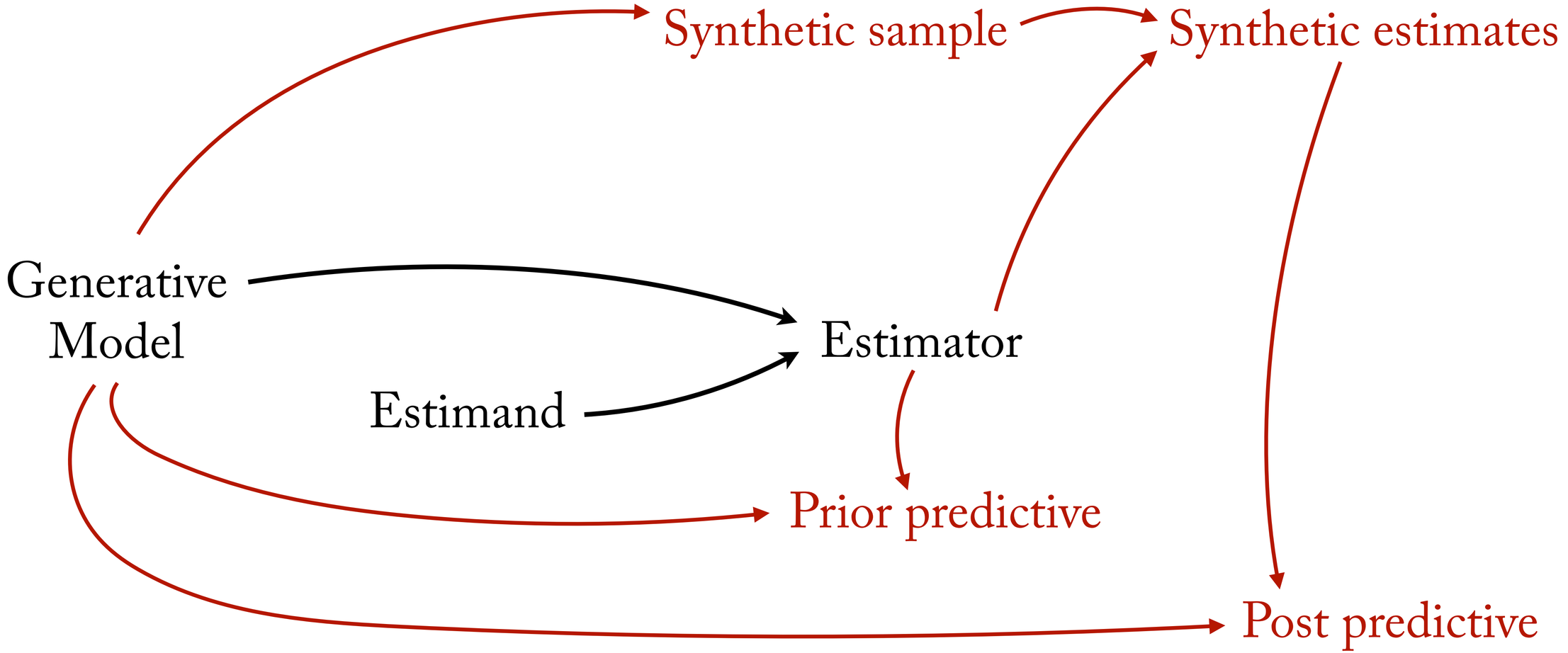

proportion un-adopted
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
days


proportion un-adopted
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
days



Cat age, breed
Competing events (death, escape)
Day of week
Month of year
Type of intake (stray, surrender)

Incremental building and testing



Incremental building and testing

- (1) Add new feature to simulation function
- (2) Update statistical model
- (3) Test update with simulated data
- (4) Check prior and posterior predictive distributions
- (5) Compare posterior predictive to previous models
- (6) Continue from (1) when useful

Modest Goals

- Learn the basic concepts of probabilistic programming
- Learn how to express ideas as probability models
- Learn conventions of Stan language
- Learn a reasonable, reliable workflow

Stop for Questions