Hierarchical Modeling

Tim Frasier

General Idea

- One benefit of a Bayesian approach is that you have full control of your models and what assumptions they make about the data
- Hierarchical modelling provides a more efficient use of the data, where you can explicitly model how different estimates are related to one another
- Will start simple now, but get more complicated later

Rationale

- Current models have amnesia:
  - Each category has completely new prior distribution
  - As if model has never seen data like it before

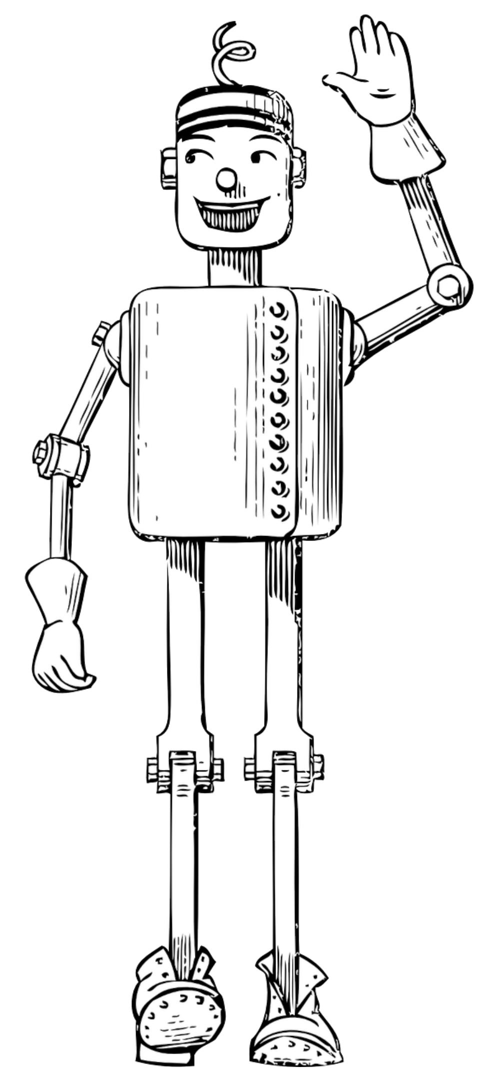
- Is this best use of data?
  - Surely different categories of the same variable provide *some* information about the other categories

  - Height coefficient of one sex must provide some information about the coefficient for the other sex (can’t differ by 0, or by 1,000,000)

Example

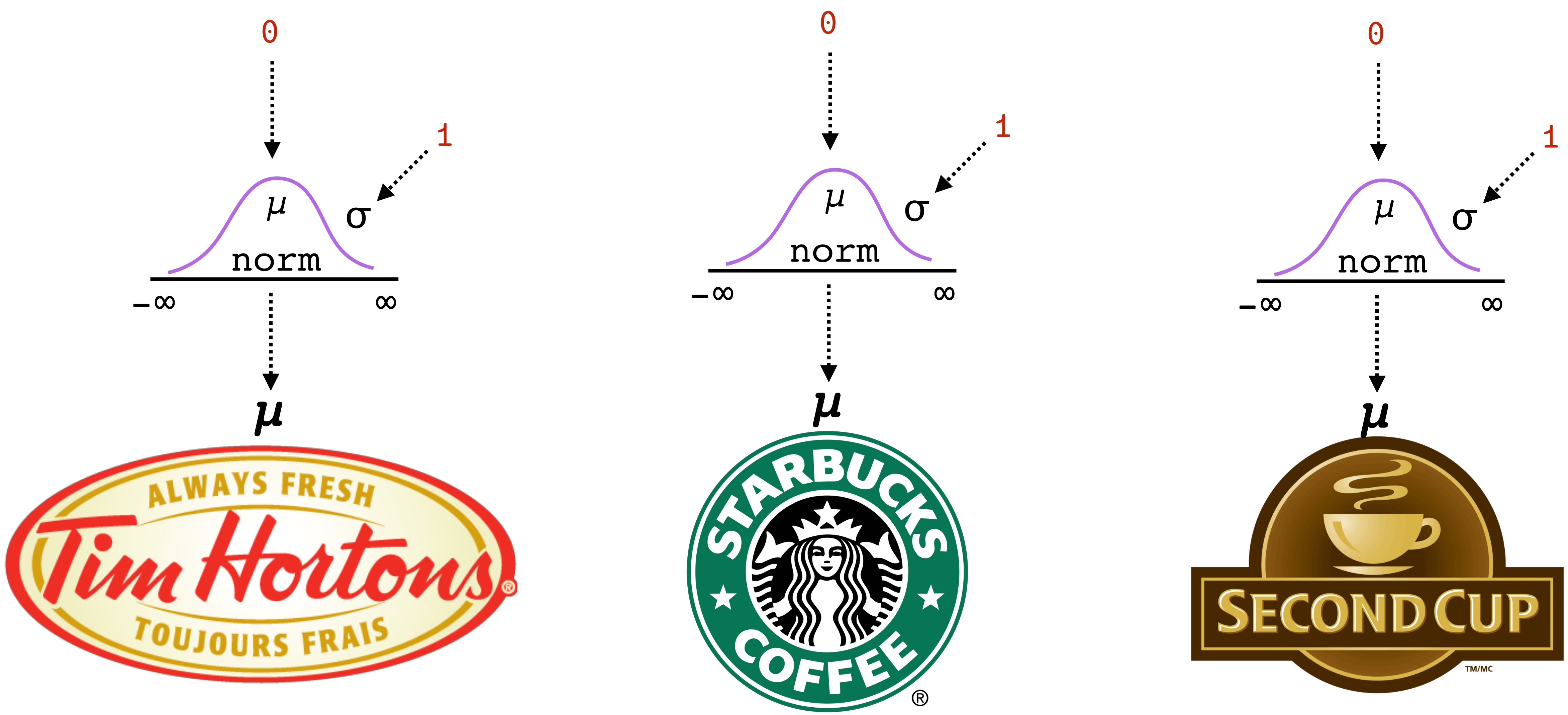
- Programming a robot to estimate waiting times at coffee shops
  - Tim Hortons
  - Starbucks
  - Second Cup

- When switching from Tim Hortons to Starbucks, like its never been to a coffee shop before

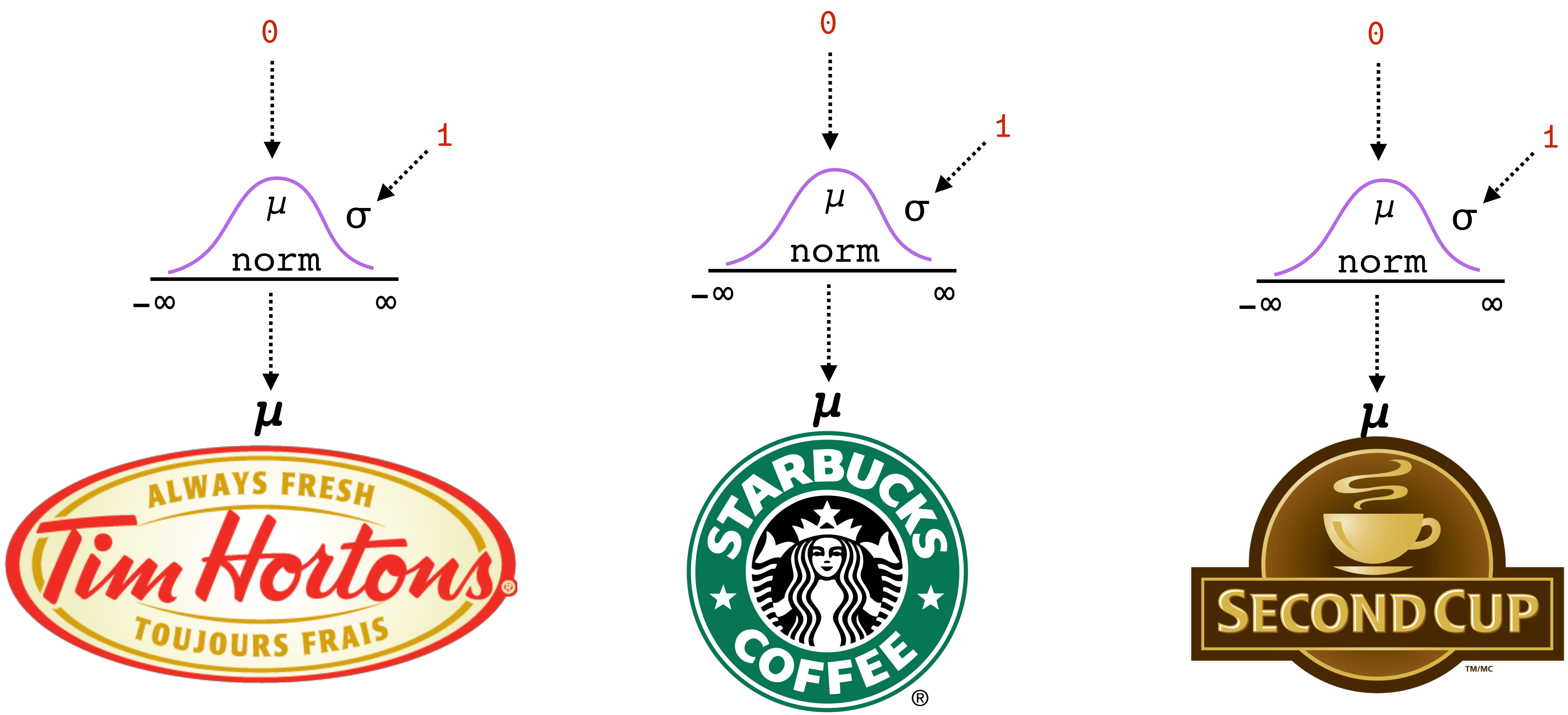


Example idea from McElreath (2016) *Statistical Rethinking*. CRC Press, Boca Raton, FL.
Images from Pixabay, in the public domain

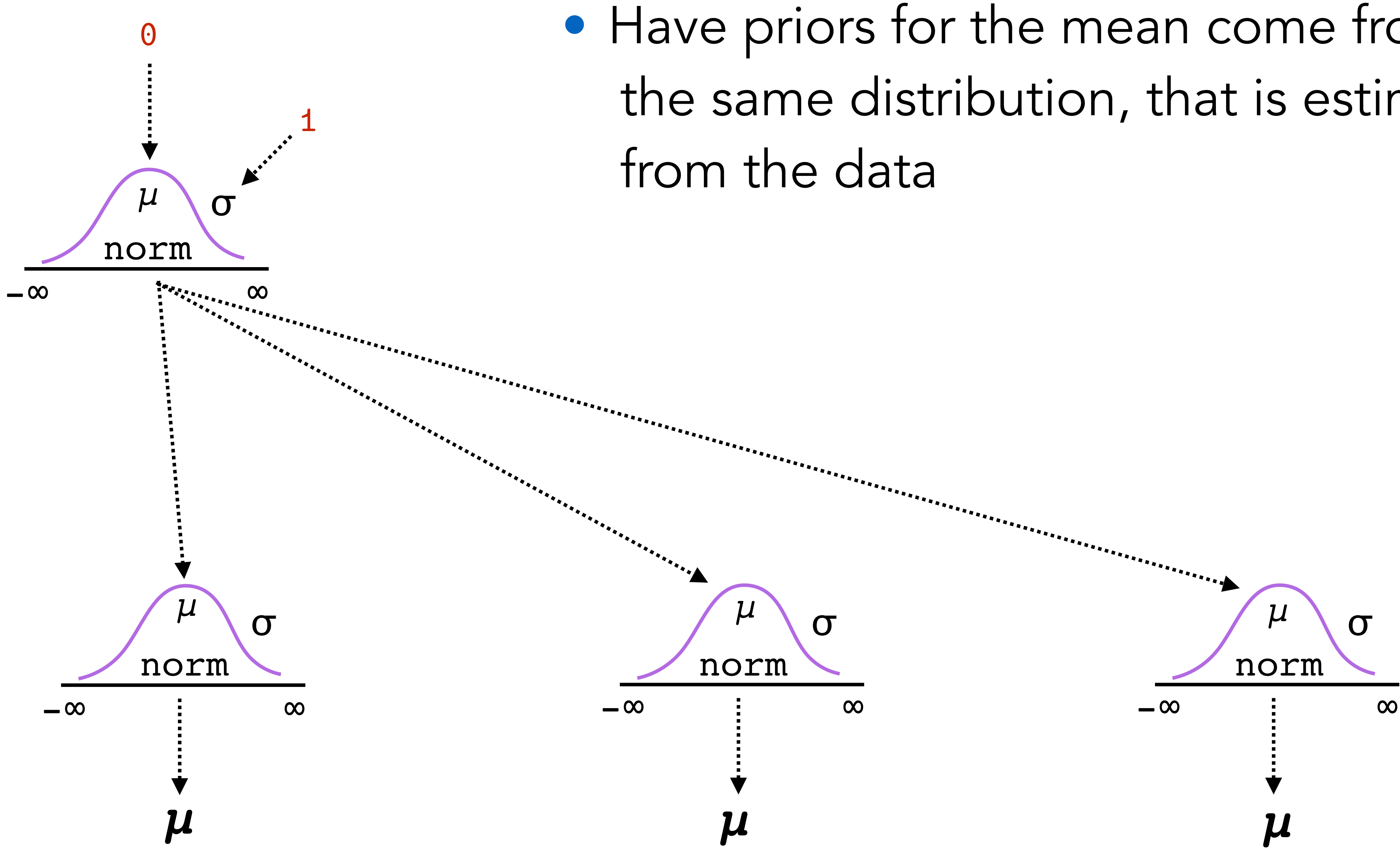
```
// Likelihood
     for (i in 1:N) {
       mu[i] = b1[x[i]];
       y[i] ~ normal(mu[i], sigma[x[i]]);
     }

// Priors
     for (j in 1:nxLevels) {
       b1[j] ~ normal(0, 1);
       sigma[j] ~ cauchy(1, 1);
     }
```



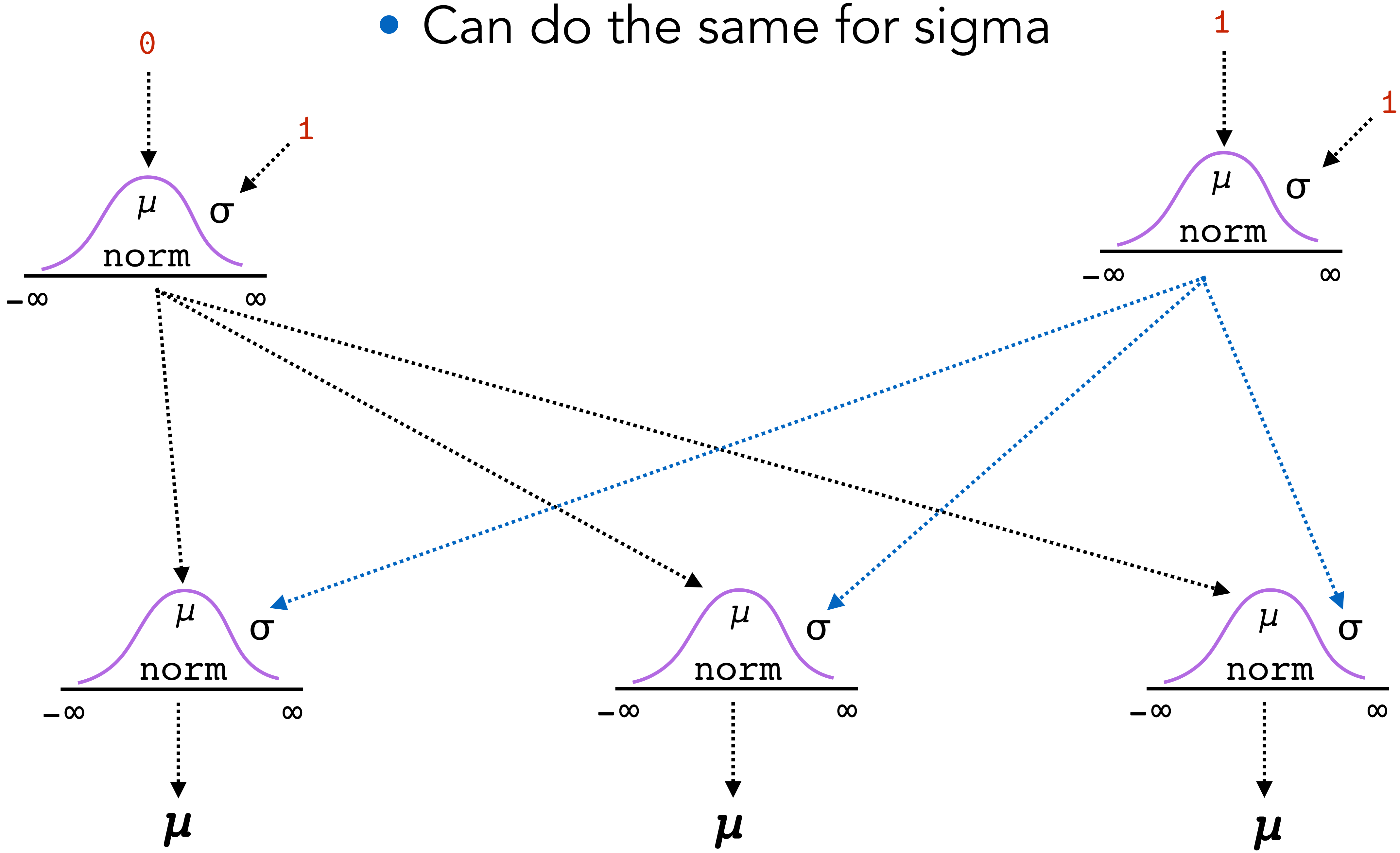
- Can have each data point be part of the estimate for that category, while also informing the overall pattern
  - Sort of like having the posterior from one category be the prior for the next



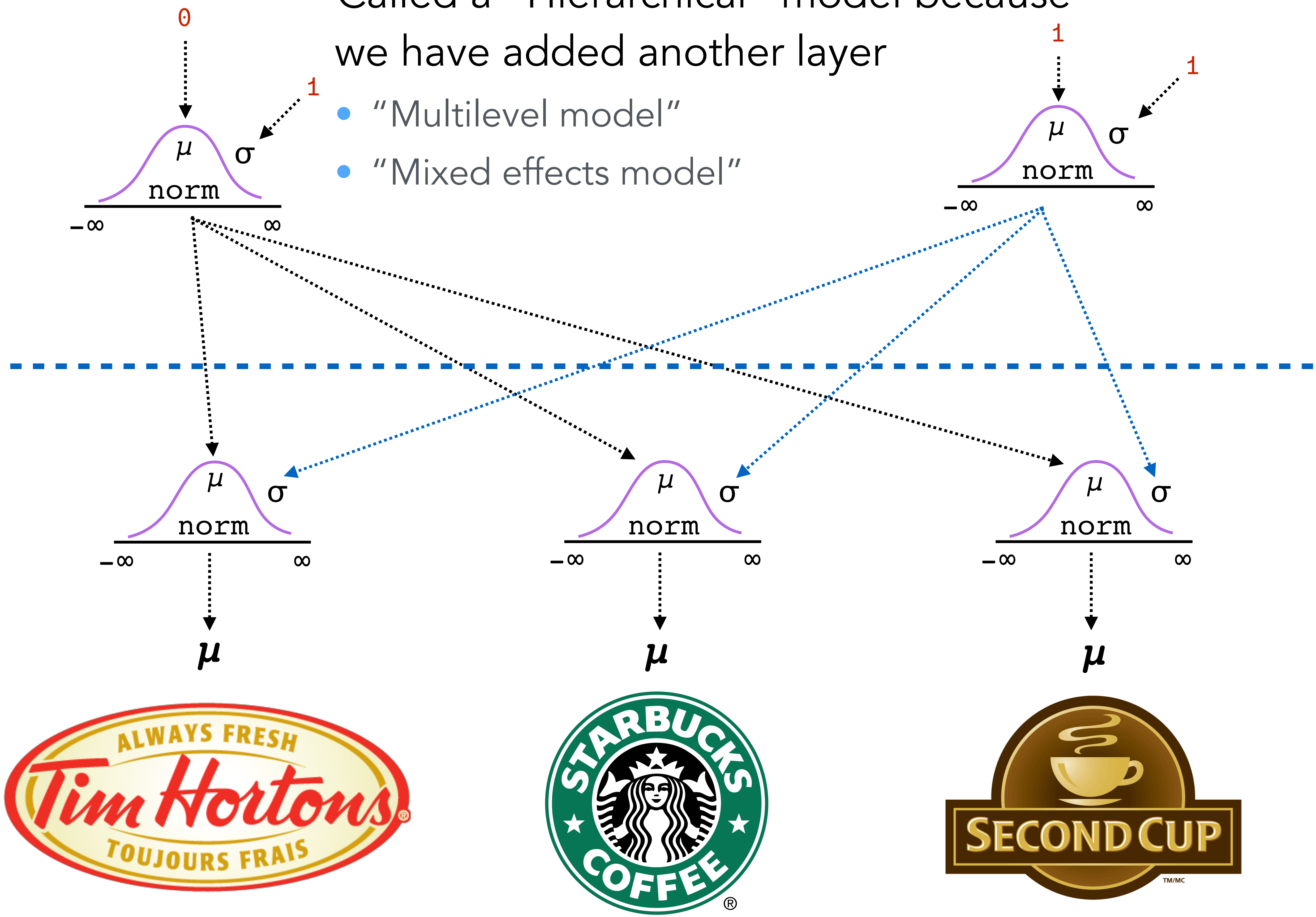
- Have priors for the mean come from the same distribution, that is estimated from the data



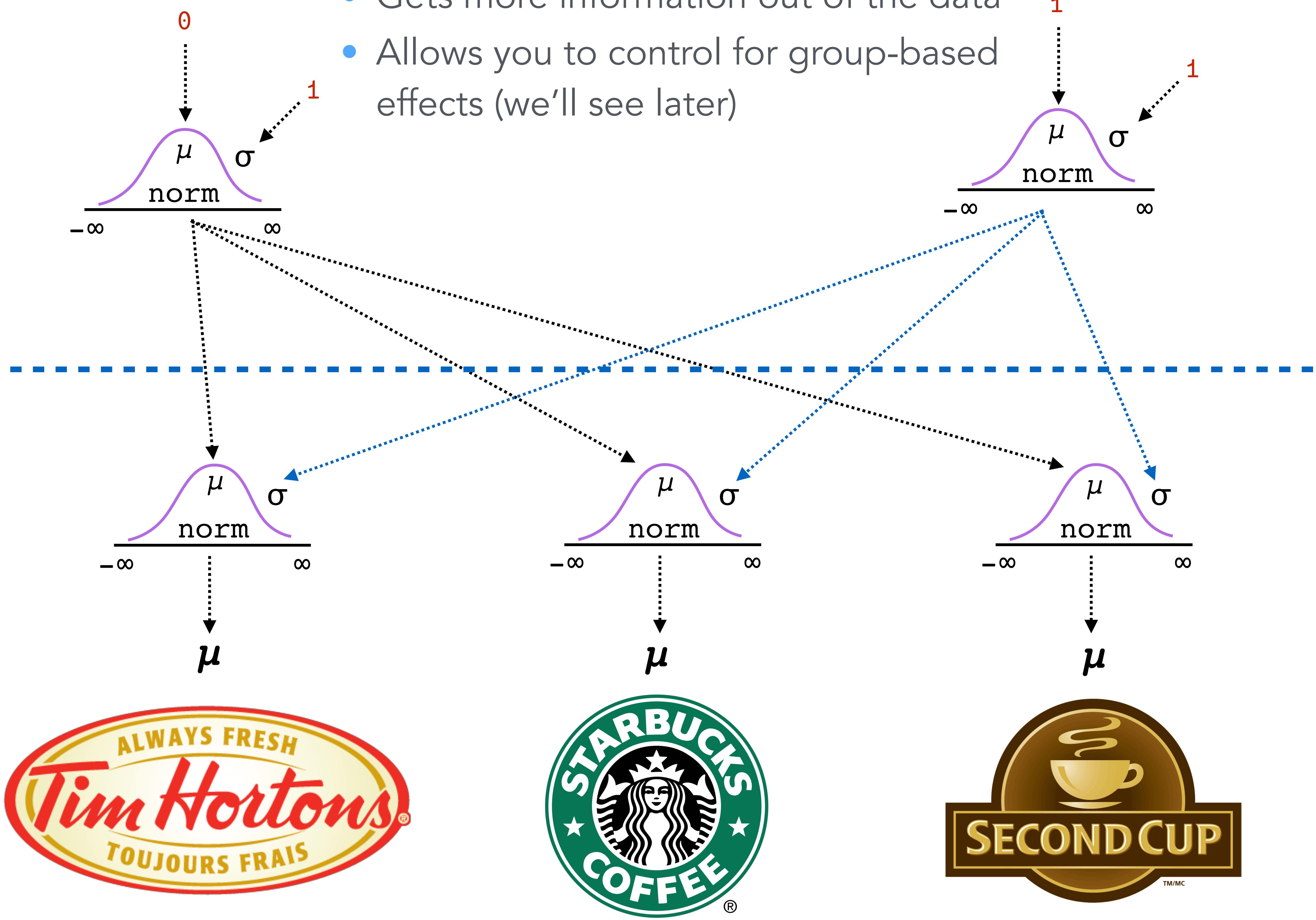
- Can do the same for sigma



- Called a "Hierarchical" model because we have added another layer
  - "Multilevel model"
  - "Mixed effects model"



- Gets more information out of the data
- Allows you to control for group-based effects (we'll see later)



The Data

Data

- `coffee.csv`

30 trips, true average = 15 minutes

5 trips, true average = 5 minutes

15 trips, true average = 10 minutes

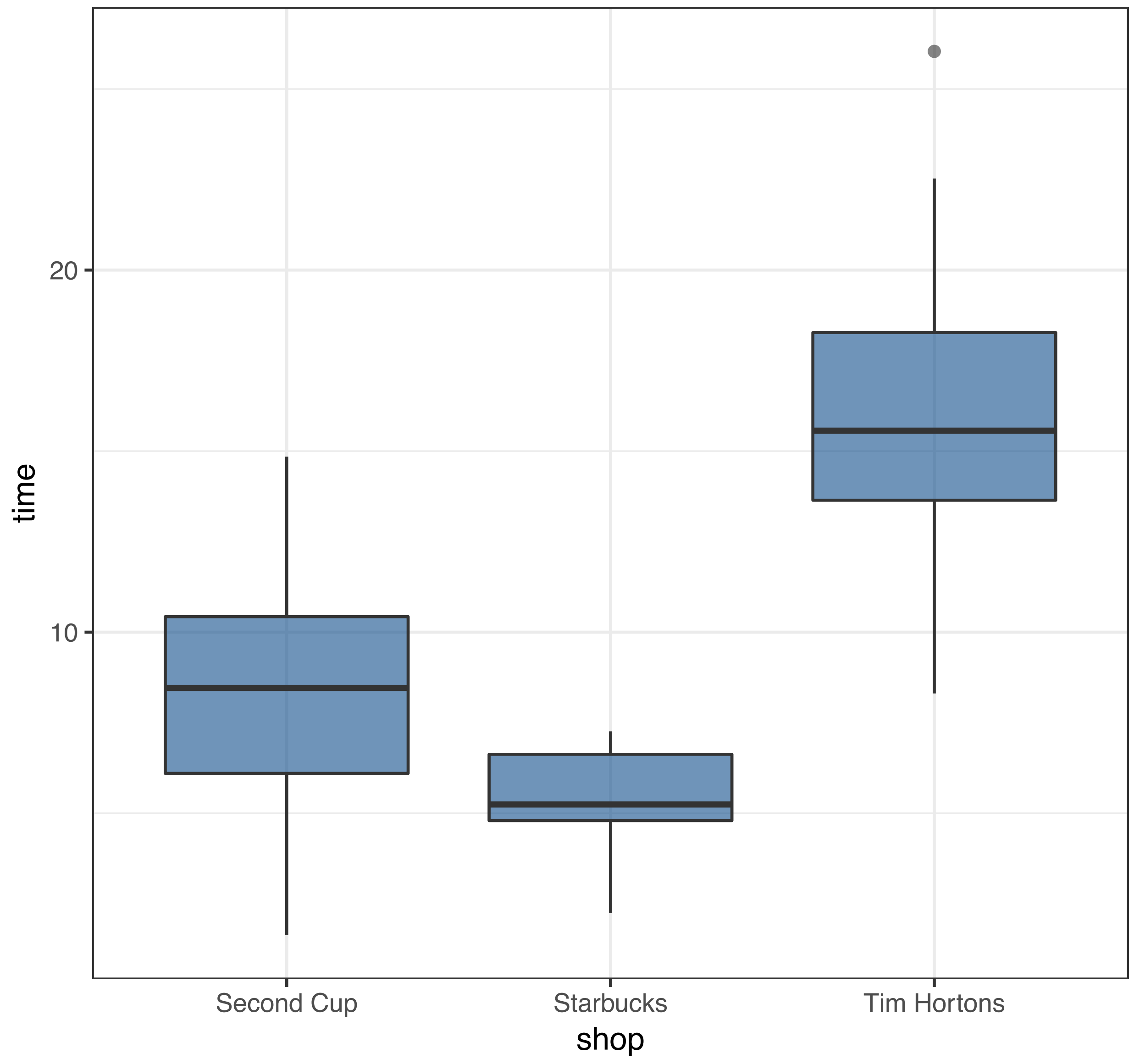
coffee

Filter

	trip	shop	times
1	1	Tim Hortons	10.0991578
2	2	Tim Hortons	4.1515614
3	3	Tim Hortons	12.4475285
4	4	Tim Hortons	9.6105939
5	5	Tim Hortons	13.5664162
6	6	Tim Hortons	7.0864281
7	7	Tim Hortons	16.6772393
8	8	Tim Hortons	7.8437422
9	9	Tim Hortons	11.8697399
10	10	Tim Hortons	7.5585060
11	11	Tim Hortons	19.1699169

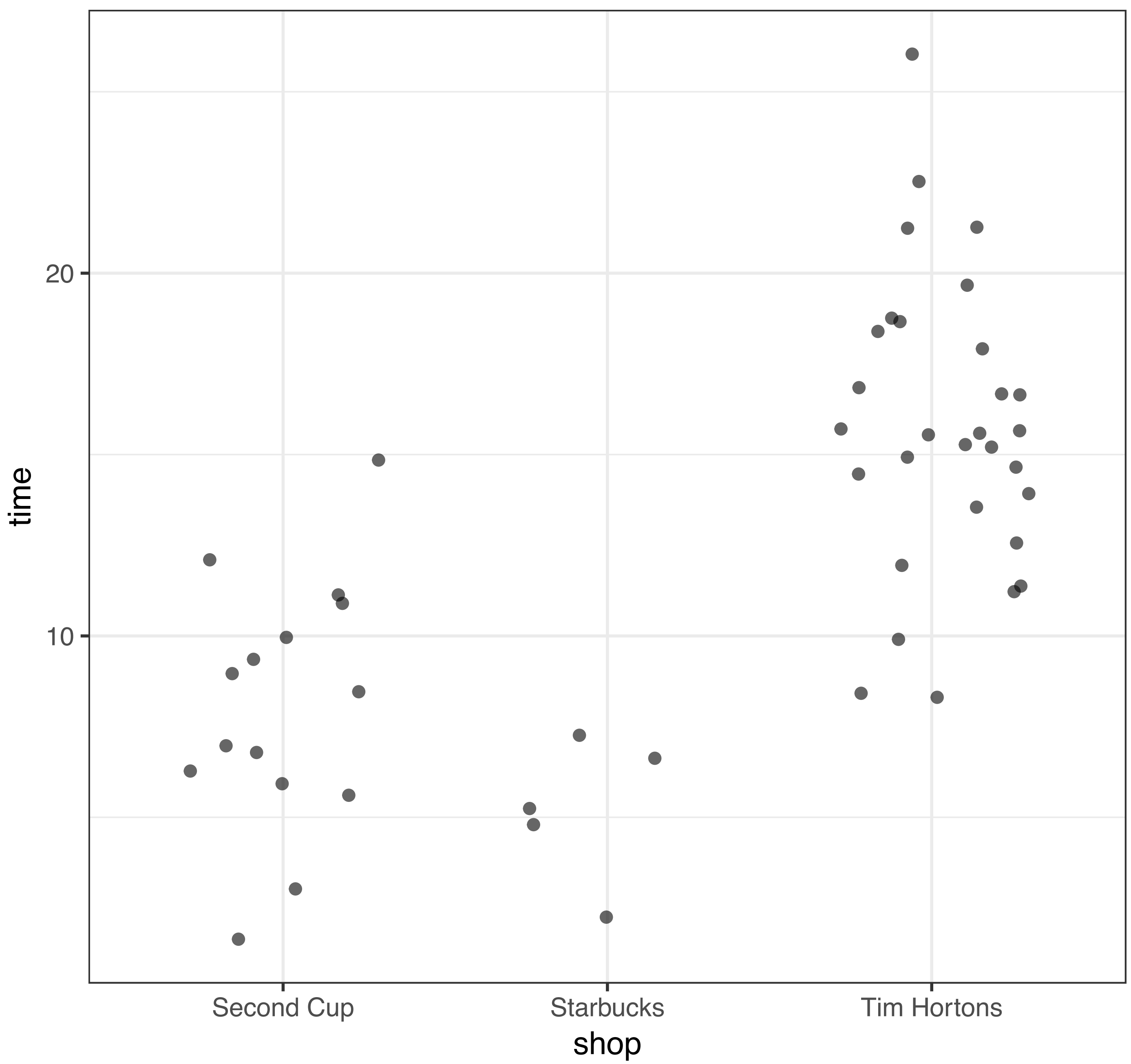
Data

```
coffee = read.table("coffee.csv", header = TRUE, sep = “,")

ggplot(coffee) +
  theme_bw() +
  geom_boxplot(aes(x = shop, y = time), fill = "dodgerblue4", alpha = 0.6)
```

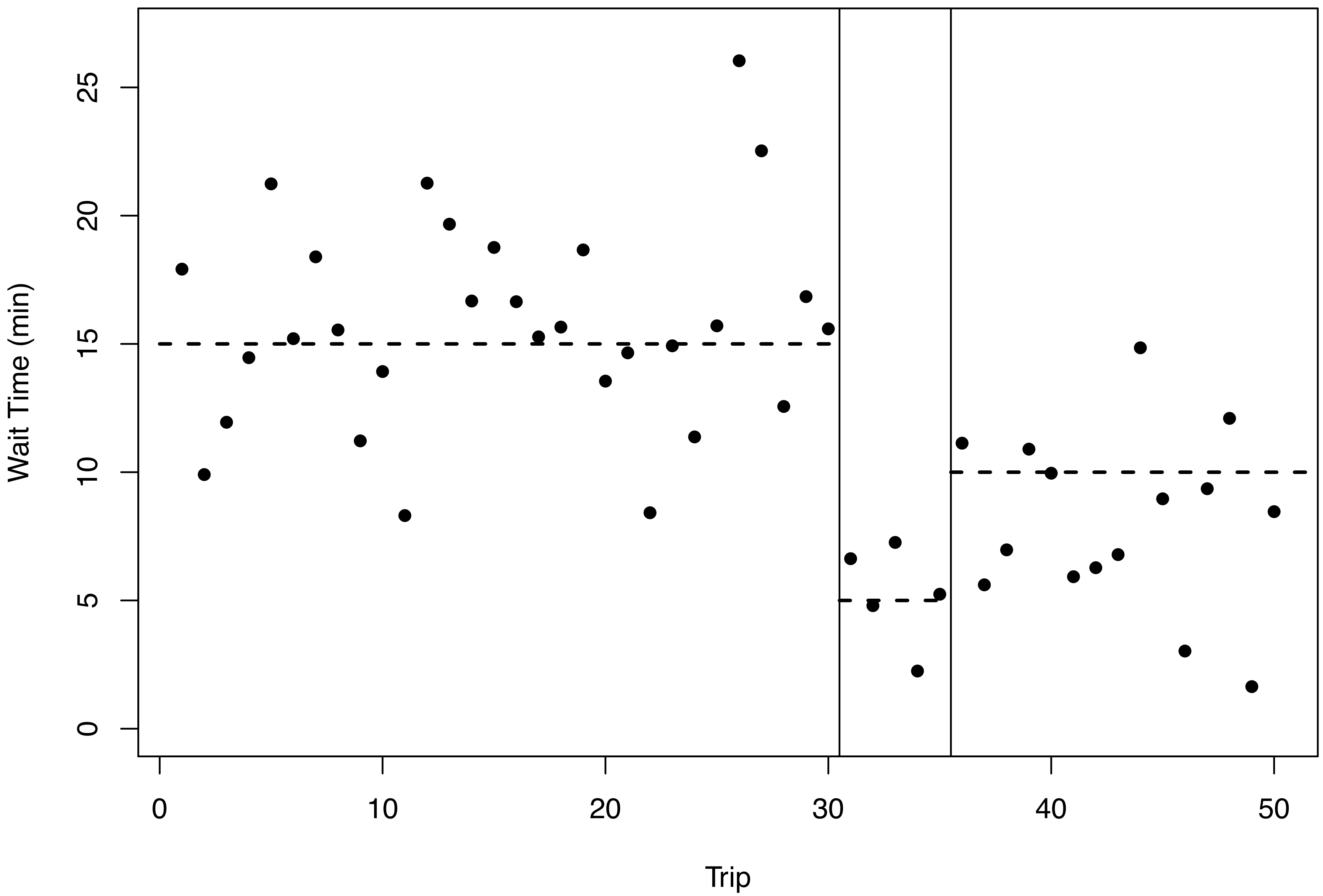


Data

```
ggplot(coffee) +
  theme_bw() +
  geom_jitter(aes(x = shop, y = time), height = 0, width = 0.3, alpha = 0.6)
```



Data



Load Libraries & Functions

```
library(rstan)
library(ggplot2)
source("plotPost.R")
```

Prepare the Data

Standardize metric (y) data

```
y = coffee$time
yMean = mean(y)
ySD = sd(y)
zy = (y - yMean) / ySD

N = length(zy)
```

Prepare the Data

Organize the nominal (x) data

```
x = as.numeric(coffee$shop)

xNames = levels(as.factor(coffee$shop))

nxLevels = length(unique(coffee$shop))
```

Create a data list for Stan

```
dataList = list(
  y = zy,
  x = x,
  N = N,
  nxLevels = nxLevels
)
```

Build the Model

- The **data block** stays the same

```
data {
    int N;            // Sample size
    int nxLevels;     // Number of shop types (levels in our nominal variable)
    vector[N] y;      // Vector of wait times
    int x[N];         // Vector of indicators of which level each sample is in
  }
```

Build the Model

- Let's look at the **model block** next
  - Definitions and likelihood remain the same

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    for (i in 1:N) {
      mu[i] = b1[x[i]];
      y[i] ~ normal(mu[i], sigma[x[i]]);

   ...
```

Build the Model

- Let's look at the **model block** next
  - Priors are where things get interesting

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j]    ~ normal(shopMean, shopMeanSD);
      sigma[j] ~ cauchy(1, 1);
    }

    // Hyperpriors
    shopMean    ~ normal(0, 1);
    shopMeanSD ~ normal(1, 1);
  }
```

- Old model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j] ~ normal(0, 1);
      sigma[j] ~ cauchy(1, 1);
    }
```

- New model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j]    ~ normal(shopMean, shopMeanSD);
      sigma[j] ~ cauchy(1, 1);
    }

    // Hyperpriors
    shopMean   ~ normal(0, 1);
    shopMeanSD ~ normal(1, 1);
  }
```

- Old model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j] ~ normal(0, 1);
      sigma[j] ~ cauchy(1, 1);
    }
```

Going to estimate a normal distribution from which **all** shop effects arise (will be informed by the distribution of each). Thus we need to estimate the mean and s.d. of this overall distribution.

- New model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j]    ~ normal(shopMean, shopMeanSD);
      sigma[j] ~ cauchy(1, 1);
    }

    // Hyperpriors
    shopMean   ~ normal(0, 1);
    shopMeanSD ~ normal(1, 1);
  }
```

- Old model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j] ~ normal(0, 1);
      sigma[j] ~ cauchy(1, 1);
    }
```

Going to estimate a normal distribution from which **all** shop effects arise (will be informed by the distribution of each). Thus we need to estimate the mean and s.d. of this overall distribution.

- New model

```
...
// Priors
    for (j in 1:nxLevels) {
      b1[j]    ~ normal(shopMean, shopMeanSD);
      sigma[j] ~ cauchy(1, 1);
    }

    // Hyperpriors
    shopMean    ~ normal(0, 1);
    shopMeanSD ~ normal(1, 1);
  }
```

Because we are estimating the mean and s.d. of this distribution, they need priors (yes…the priors need priors)

Build the Model

- Go back to the **parameters block**

```
parameters {
    real b1[nxLevels];                // Coefficients for effects of being in each level
    real<lower=0> sigma[nxLevels];    // Coefficients for sd of being in each level
    real shopMean;                    // Mean wait time across all shops
    real shopMeanSD;                  // sd for mean wait time across all shops
  }
```

Build the Model

- Lastly, the **generated quantities block** (stays the same)

```
generated quantities {
    // Definitions
    vector[N] y_pred;

    for (i in 1:N) {
      y_pred[i] = normal_rng(b1[x[i]], sigma[x[i]]);
    }
  }
```

Run the Model

- Remember to include all parameters you are interested in!!!!

```
stanFit <- stan(file = "model.stan",
                data = dataList,
                pars = c("b1", "sigma", "shopMean", "shopMeanSD", "y_pred"),
                warmup = 2000,
                iter = 7000,
                chains = 3)
```

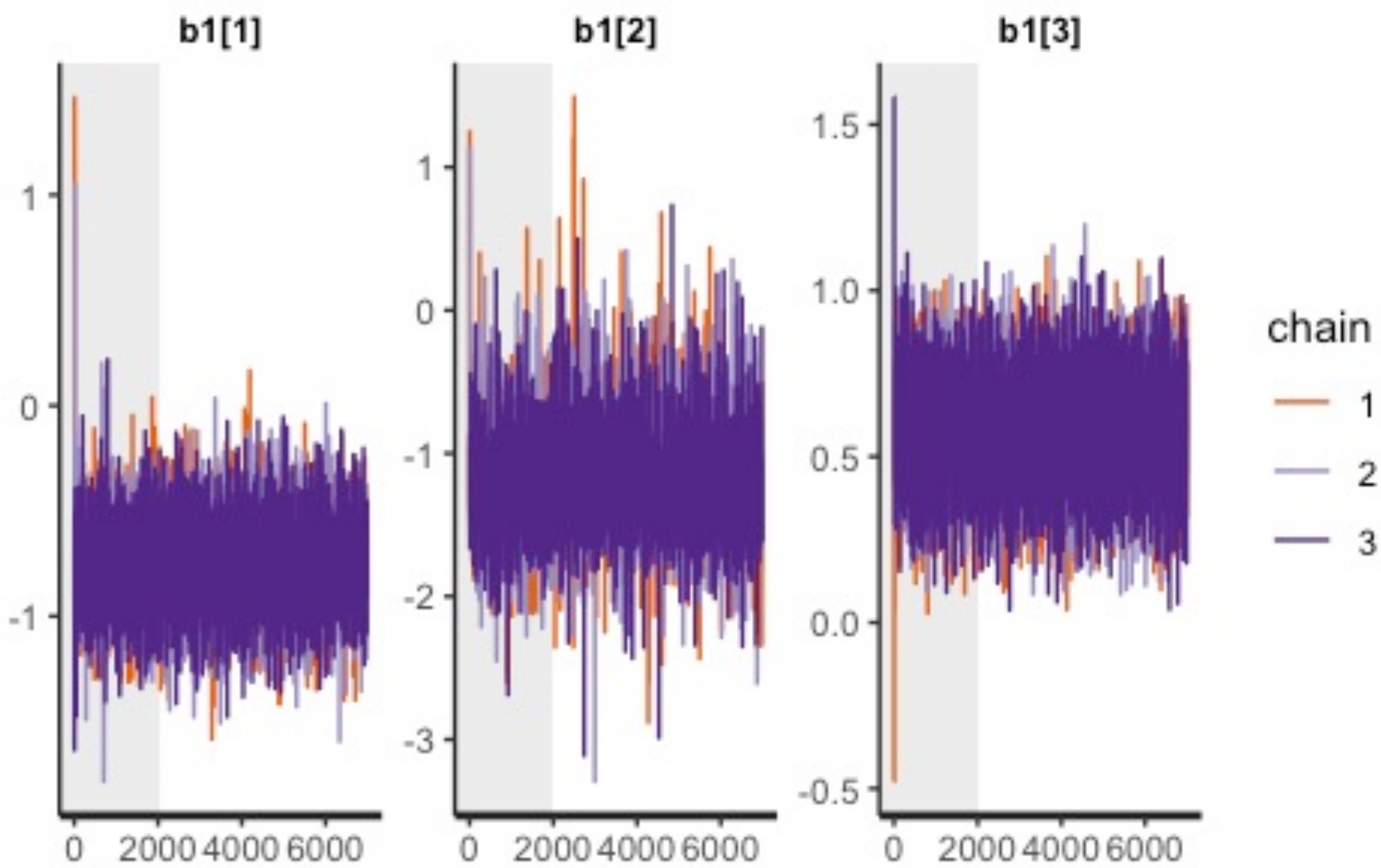
Check MCMC Performance

Rhat

```
print(stanFit)

               mean se_mean   sd   2.5%    25%   50%   75% 97.5% n_eff Rhat
b1[1]         -0.75    0.00 0.18  -1.11  -0.87 -0.75 -0.64 -0.39 12047    1
b1[2]         -1.21    0.00 0.29  -1.72  -1.38 -1.24 -1.08 -0.53  7296    1
b1[3]          0.58    0.00 0.14   0.30   0.49  0.58  0.67  0.86 13671    1
sigma[1]       0.69    0.00 0.15   0.47   0.59  0.67  0.77  1.05 10814    1
sigma[2]       0.57    0.00 0.31   0.24   0.36  0.49  0.69  1.40  6769    1
sigma[3]       0.76    0.00 0.11   0.59   0.69  0.75  0.83  1.01 12039    1
shopMean      -0.31    0.01 0.59  -1.45  -0.69 -0.32  0.05  0.90 11907    1
shopMeanSD     1.24    0.01 0.51   0.51   0.85  1.14  1.52  2.49  7635    1
y_pred[1]      0.57    0.01 0.79  -0.98   0.05  0.58  1.10  2.15 15149    1
y_pred[2]      0.58    0.01 0.78  -0.97   0.07  0.58  1.08  2.12 14542    1
...
```

Check MCMC Performance
traceplots

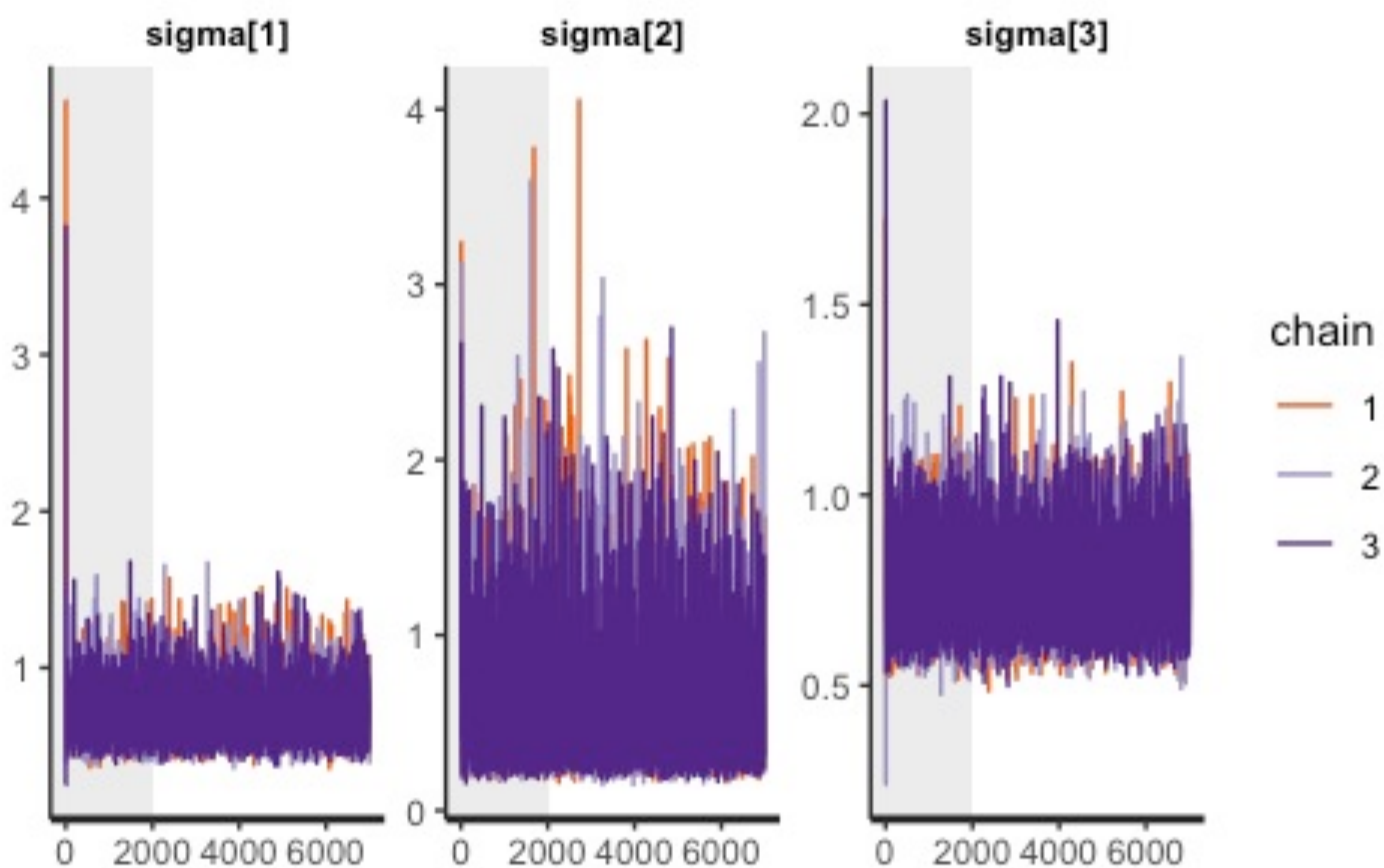
```
stan_trace(stanFit, pars = "b1", inc_warmup = TRUE)
```



Check MCMC Performance

traceplots

```
stan_trace(stanFit, pars = “sigma", inc_warmup = TRUE)
```



Evaluate Results

Extract the Data

```
mcmcChains = as.data.frame(stanFit)

zshopMean = mcmcChains$shopMean
zshopMeanSD = mcmcChains$shopMeanSD

chainLength = length(mcmcChains[, 1])

zsigma = matrix(0, ncol = nxLevels, nrow = chainLength)
for (i in 1:nxLevels) {
  zsigma[, i] = mcmcChains[, paste("sigma[", i, "]", sep = "")]
}
```

Evaluate Results

Extract the Data

```
zb1 = matrix(0, ncol = nxLevels, nrow = chainLength)
for (i in 1:nxLevels) {
  zb1[, i] = mcmcChains[, paste("b1[", i, "]", sep = "")]
}

zypred = matrix(0, ncol = N, nrow = chainLength)
for (i in 1:N) {
  zypred[, i] = mcmcChains[, paste("y_pred[", i, "]", sep = "")]
}
```

Evaluate Results

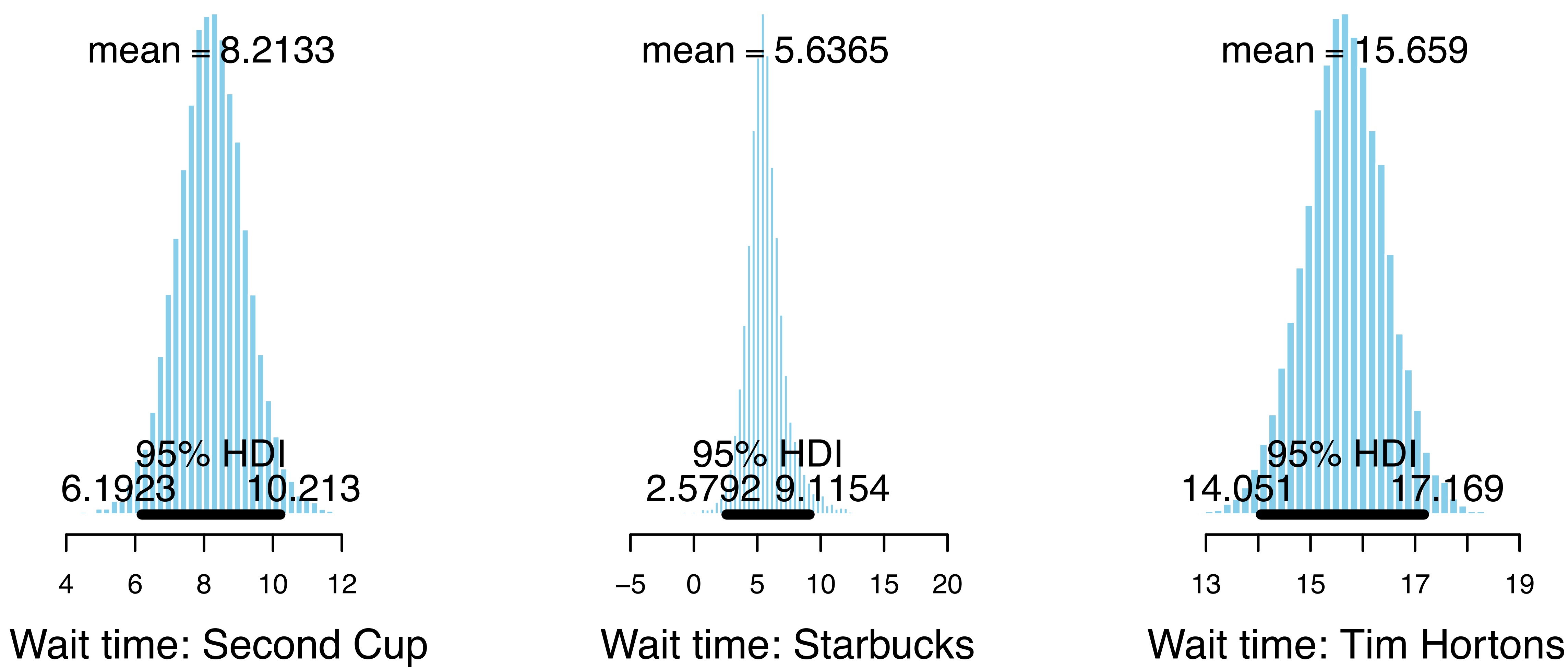
Convert back to original scale

```
sigma = zsigma * ySD
b1 = (zb1 * ySD) + yMean
ypred = (zypred * ySD) + yMean
shopMean = (zshopMean * ySD) + yMean
shopMeanSD = zshopMeanSD * ySD
```

Evaluate Results

Plot estimates

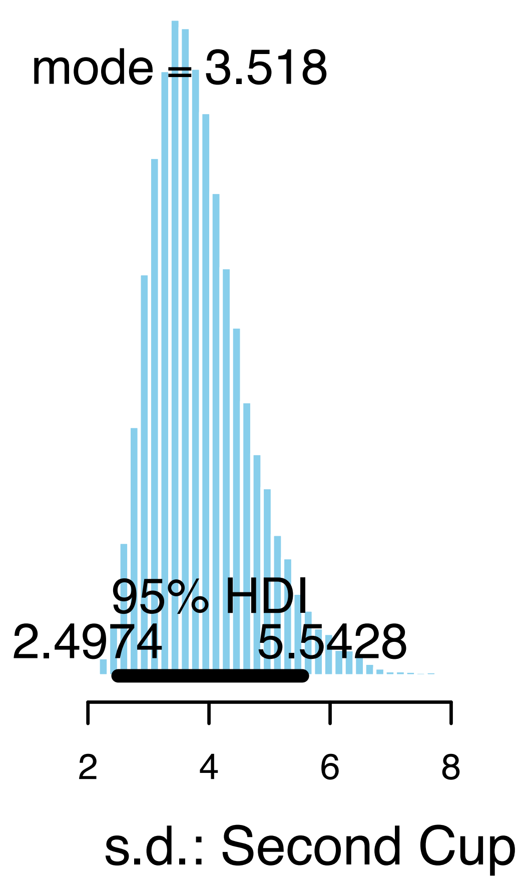
```
par(mfrow = c(1, 3))
histInfo = plotPost(b1[, 1], xlab = "Wait time: Second Cup")
histInfo = plotPost(b1[, 2], xlab = "Wait time: Starbucks")
histInfo = plotPost(b1[, 3], xlab = "Wait time: Tim Hortons")
```

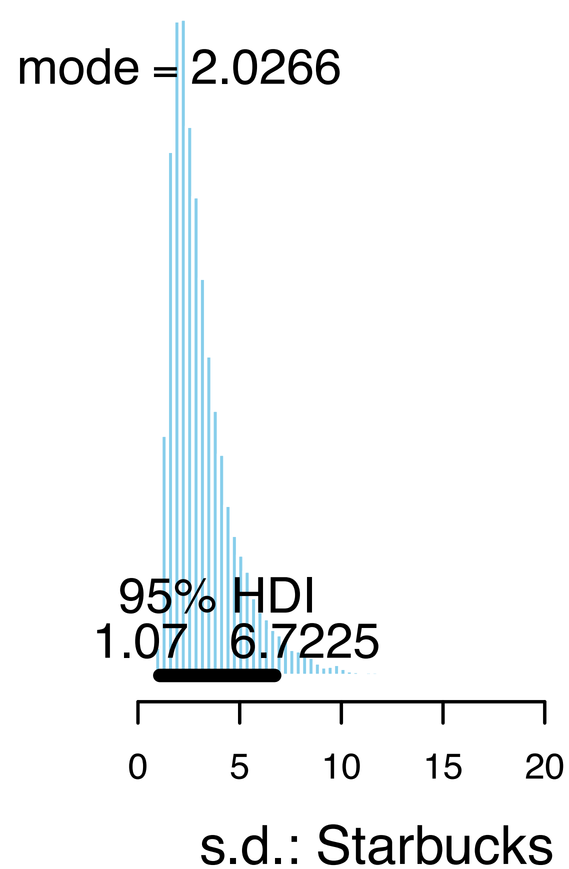


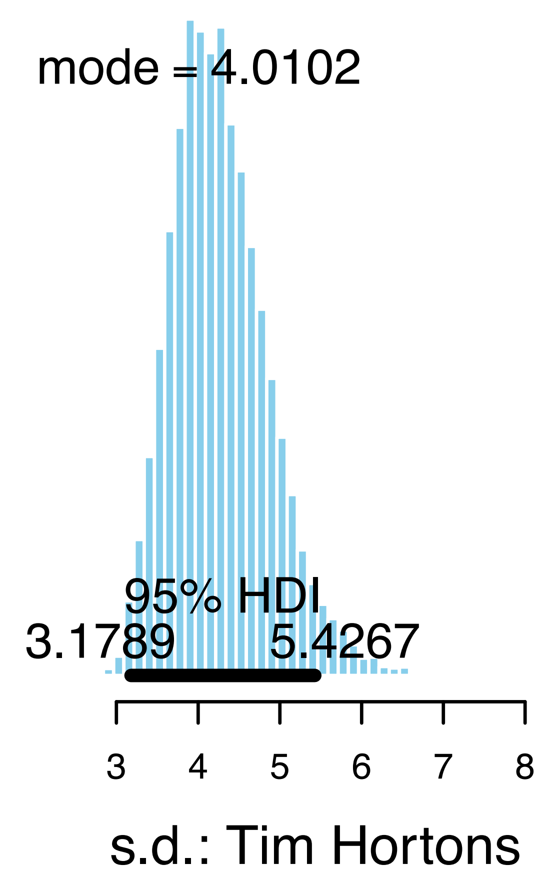
Evaluate Results

Plot estimates

```
histInfo = plotPost(b1[, 1], xlab = "Wait time: Second Cup")
histInfo = plotPost(b1[, 2], xlab = "Wait time: Starbucks")
histInfo = plotPost(b1[, 3], xlab = "Wait time: Tim Hortons")
```



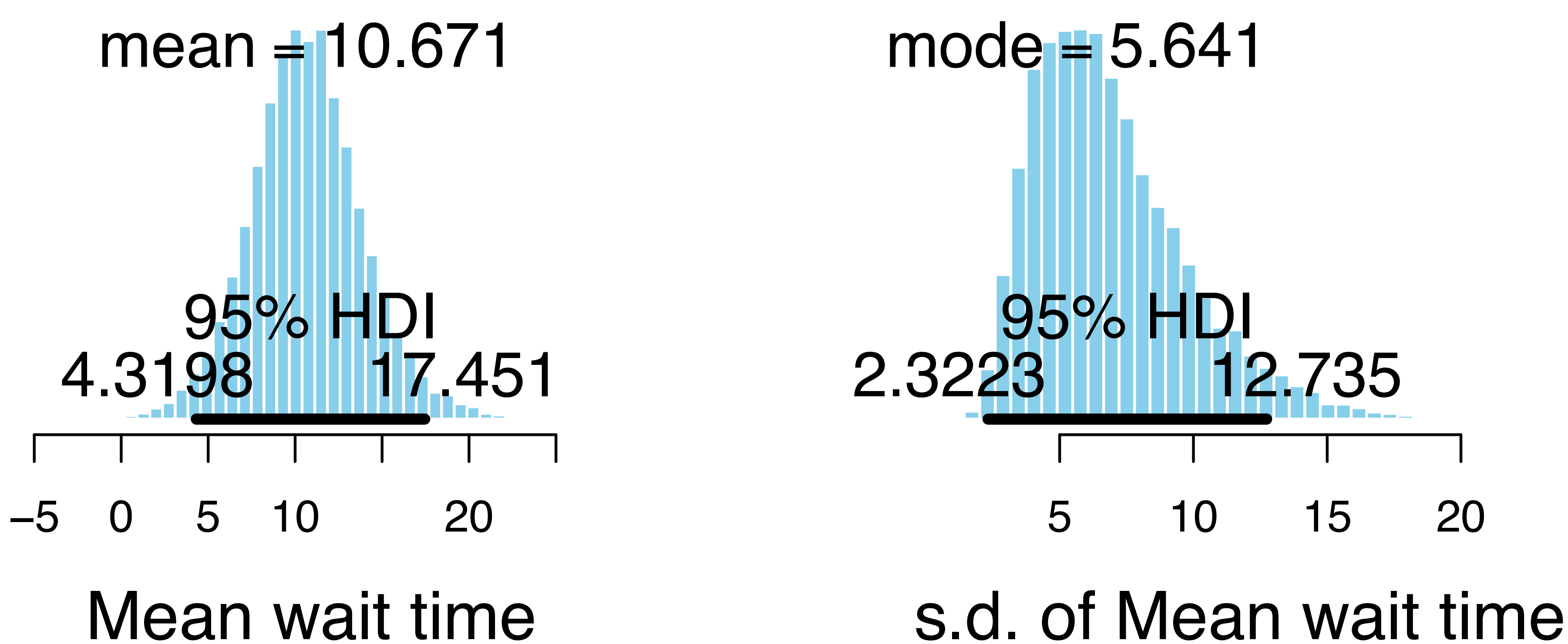




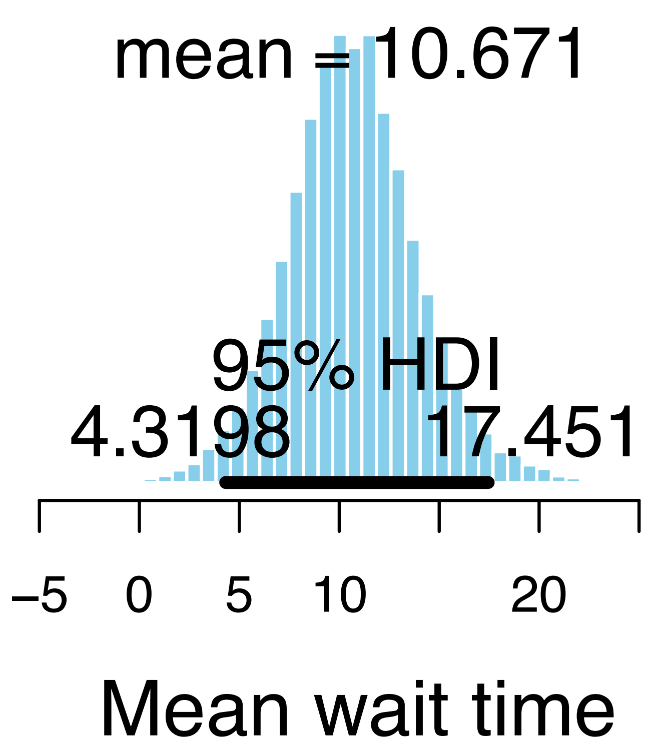
Evaluate Results

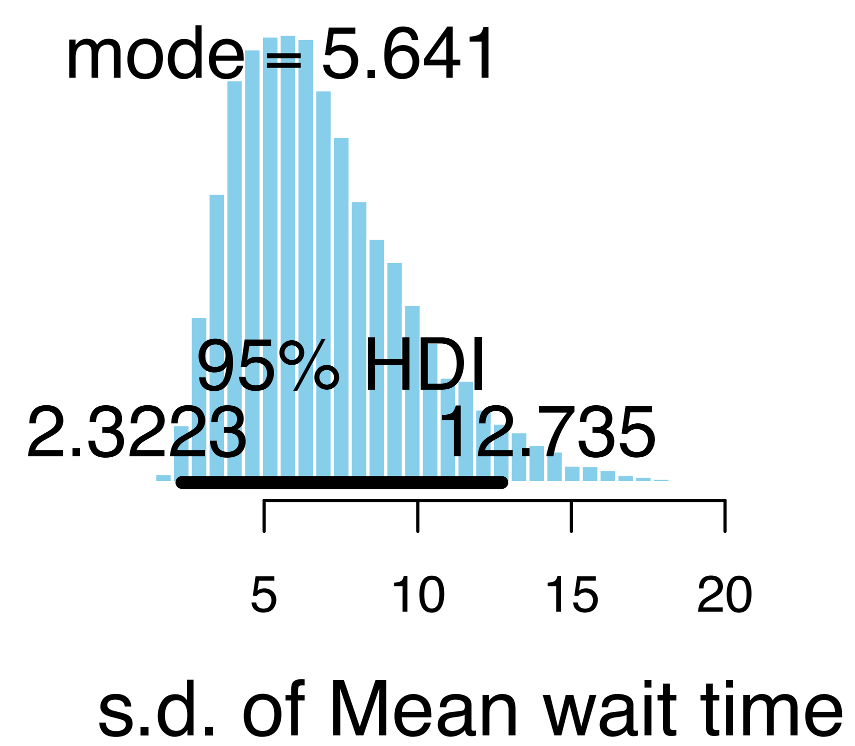
Plot estimates

```
par(mfrow = c(1, 2))
histInfo = plotPost(shopMean, xlab = "Mean wait time")
histInfo = plotPost(shopMeanSD, xlab = "s.d. of Mean wait time", showMode = TRUE)
```



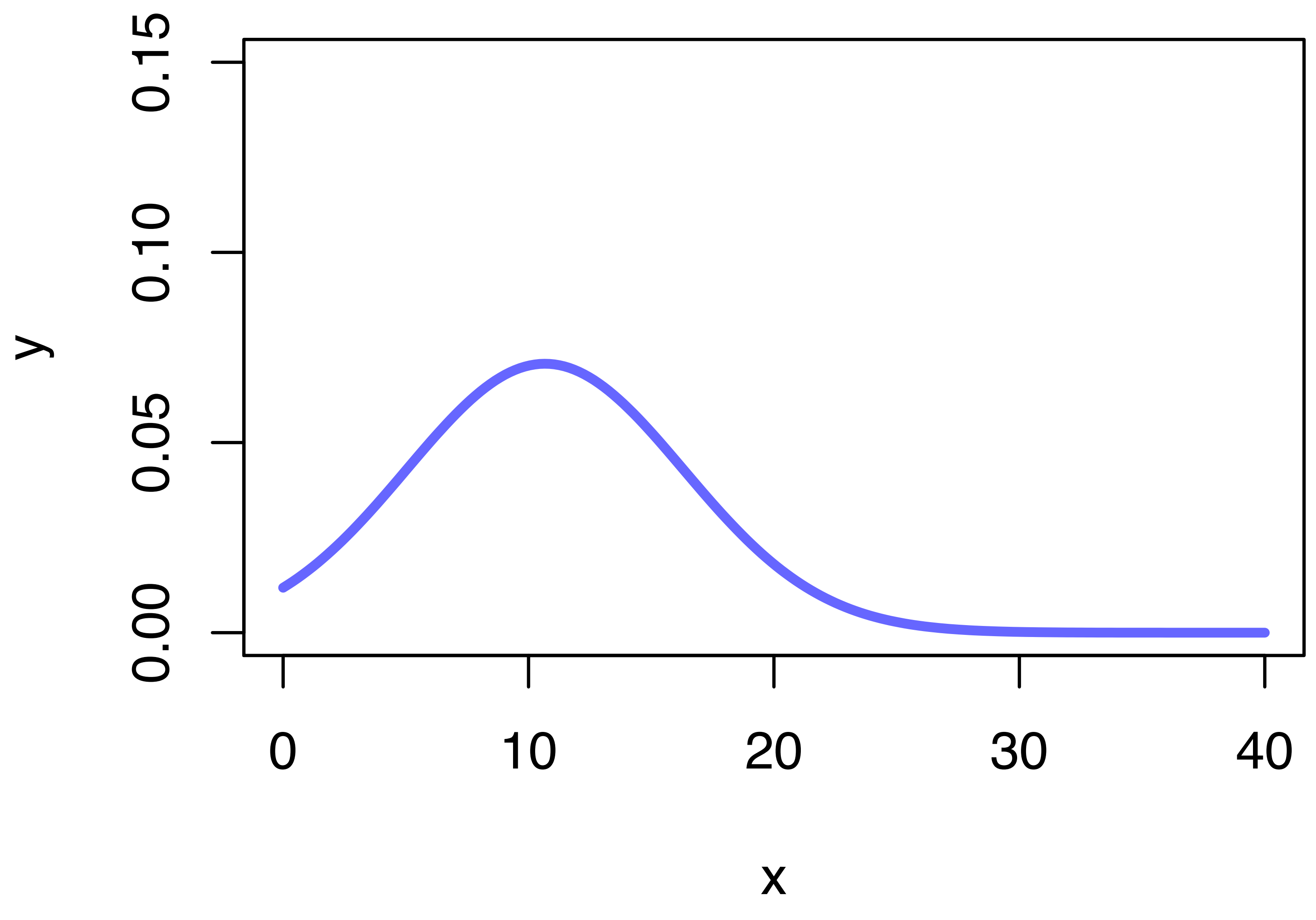
- What does a normal distribution with a mean of `10.671`, and a s.d. of `5.641` look like?





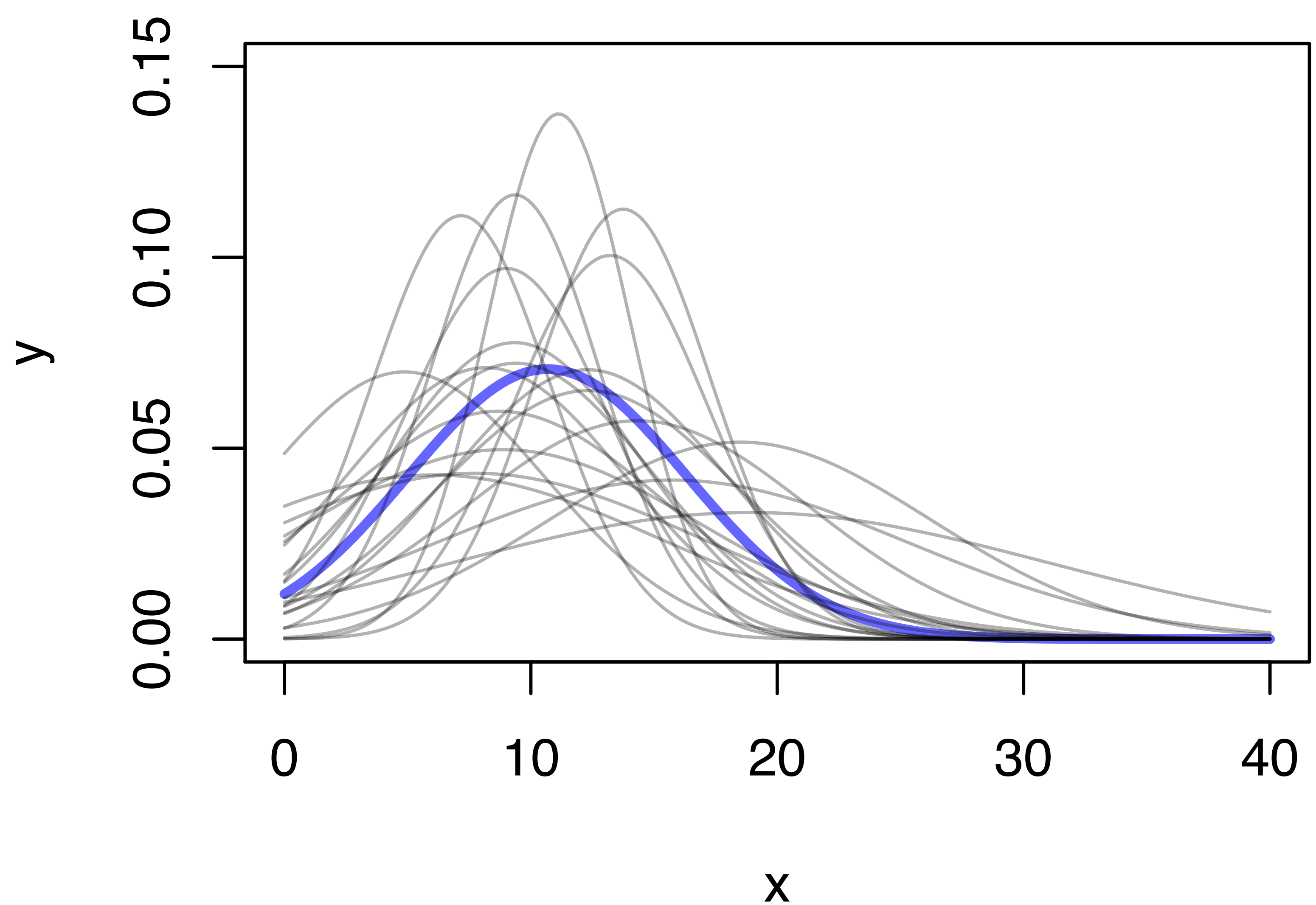
- What does a normal distribution with a mean of `10.671`, and a s.d. of `5.641` look like?

```
x = seq(from = 0, to = 40, length = 200)
y = dnorm(x, mean = 10.671, sd = 5.641)

plot(y ~ x, type = "l", ylim = c(0, 0.15), lwd = 3, col = rgb(0, 0, 1, 0.6))
```



- What about the variation!?

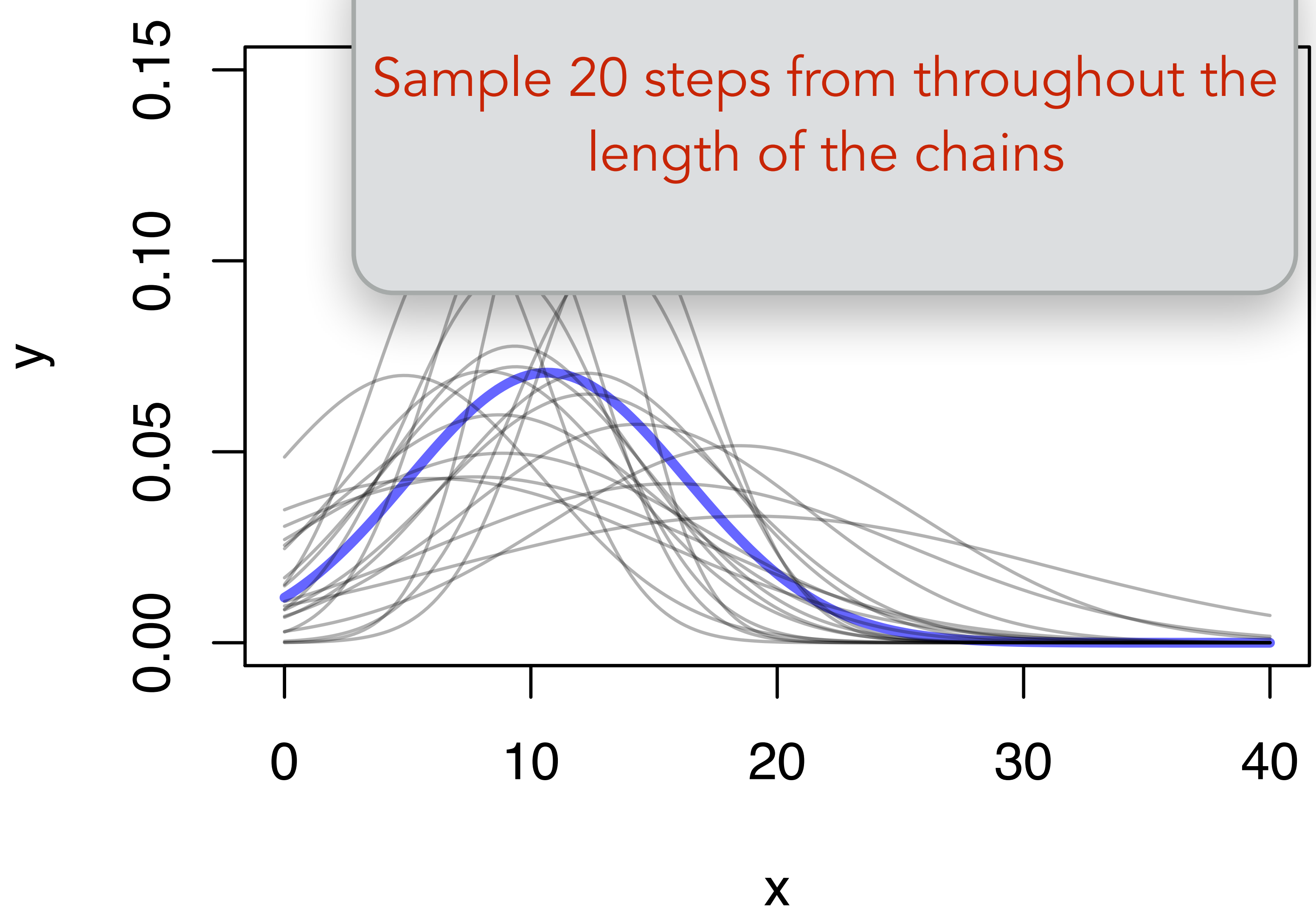
```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```



- What about the variation!?

```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```

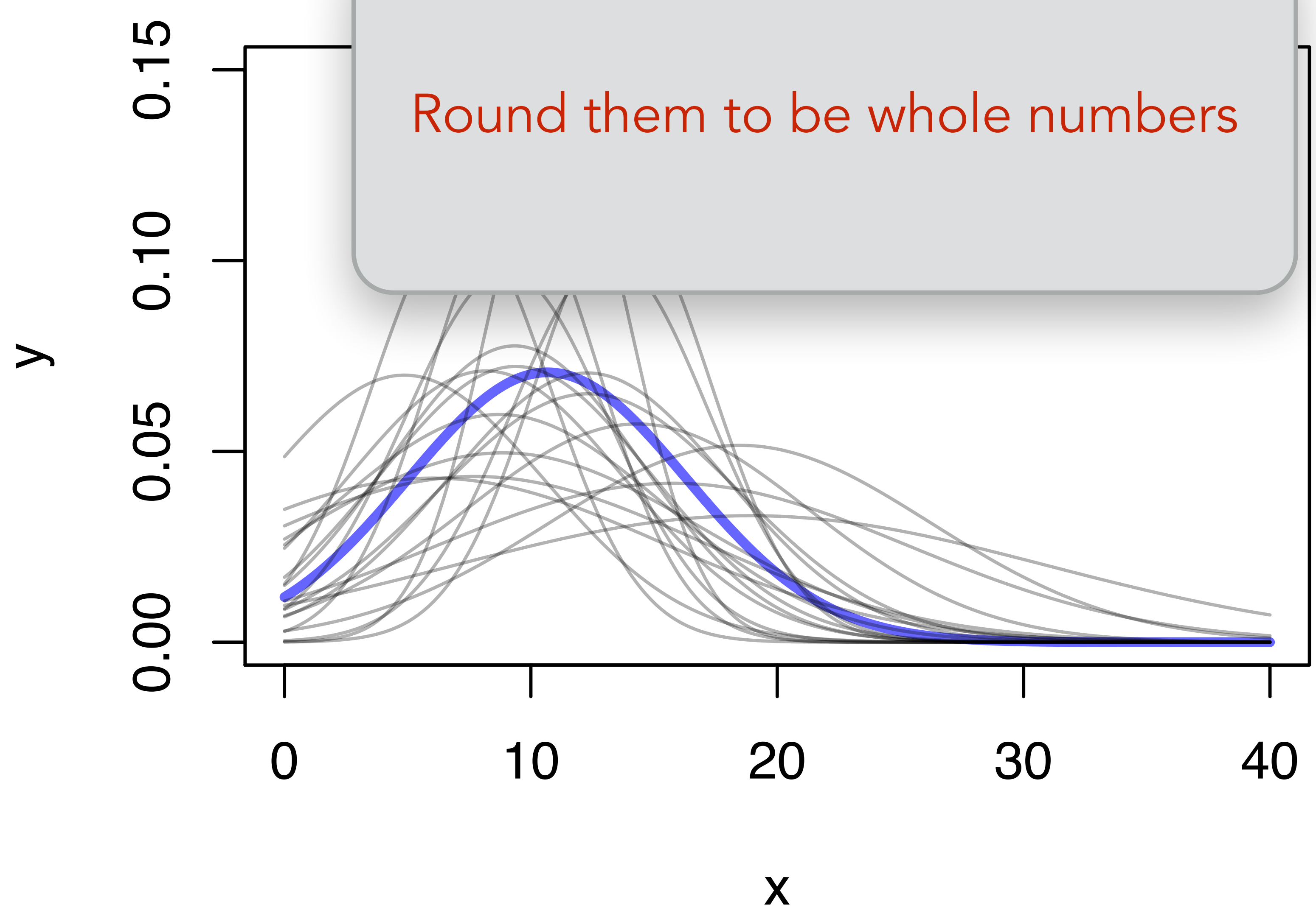
Sample 20 steps from throughout the length of the chains



- What about the variation!?

```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```

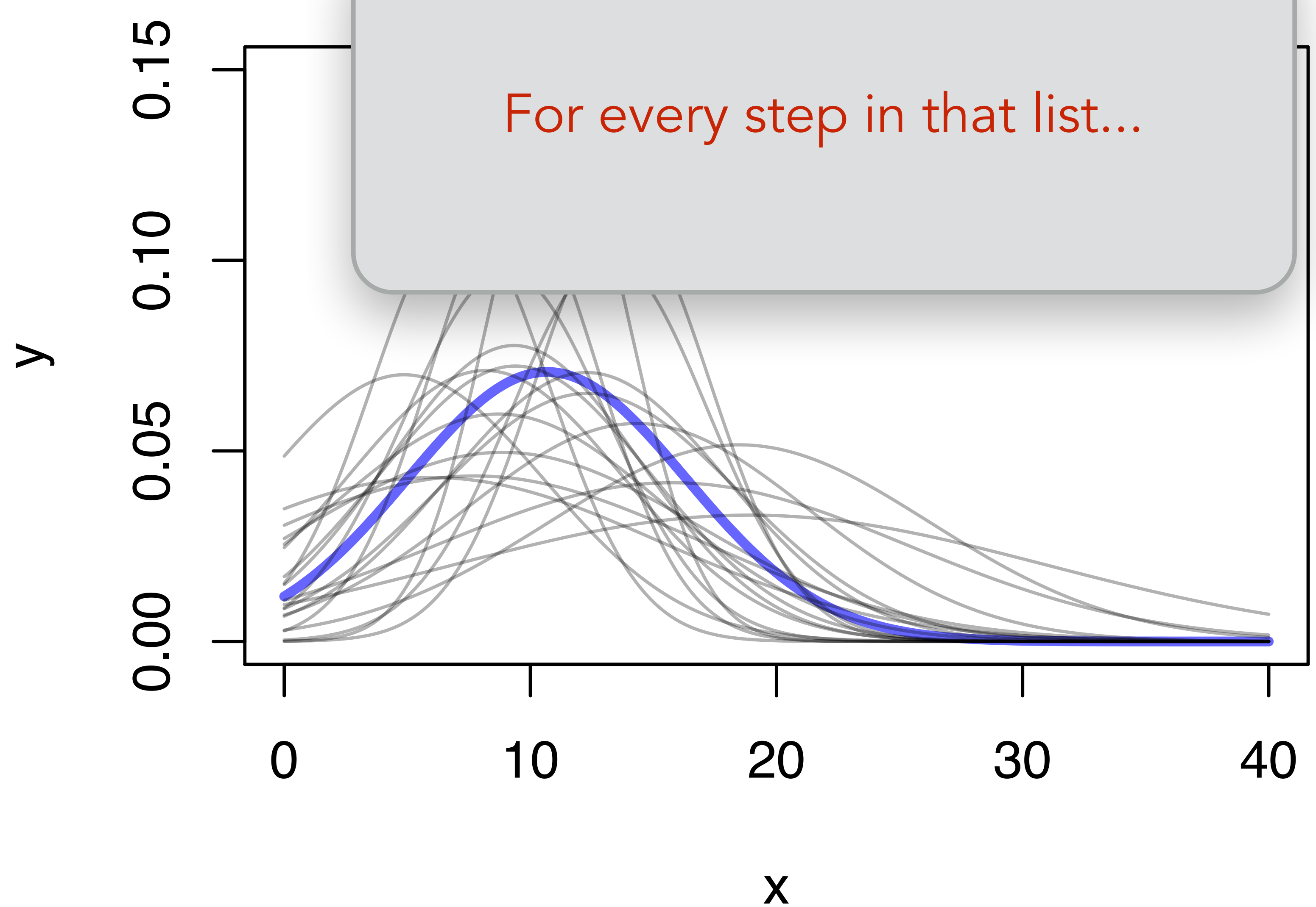
Round them to be whole numbers



- What about the variation!?

```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```

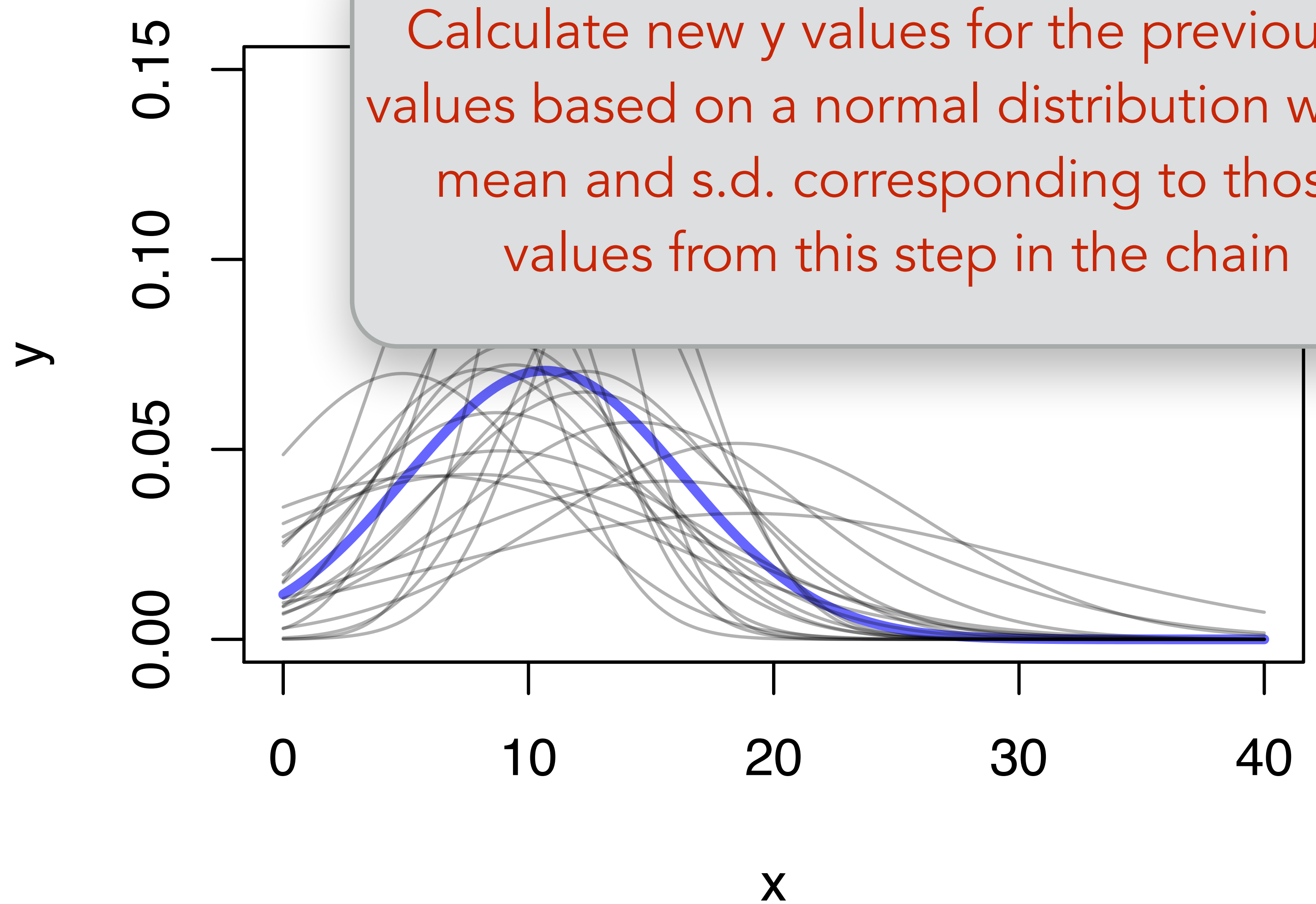
For every step in that list...



- What about the variation!?

```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```

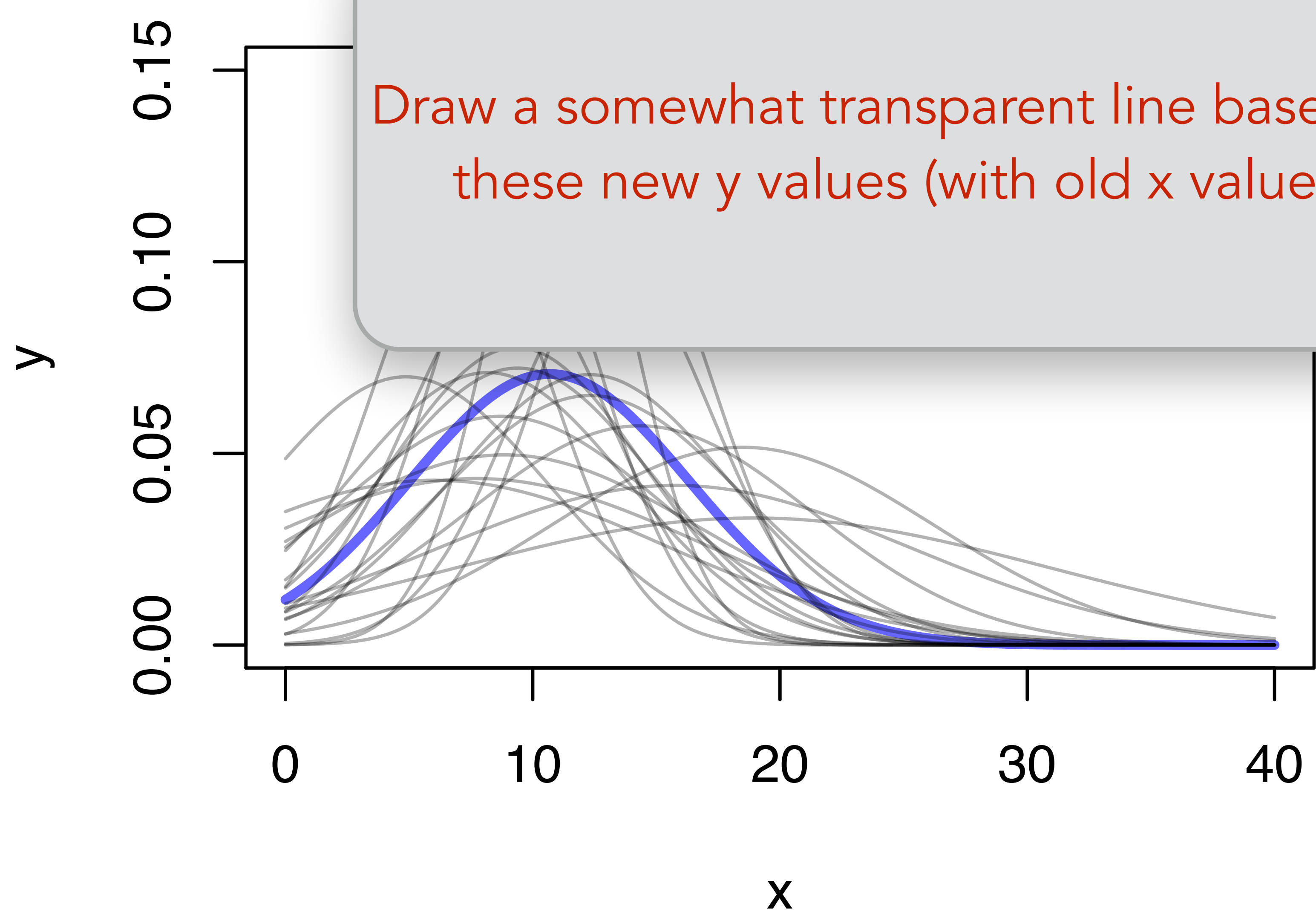
Calculate new y values for the previous x values based on a normal distribution with a mean and s.d. corresponding to those values from this step in the chain

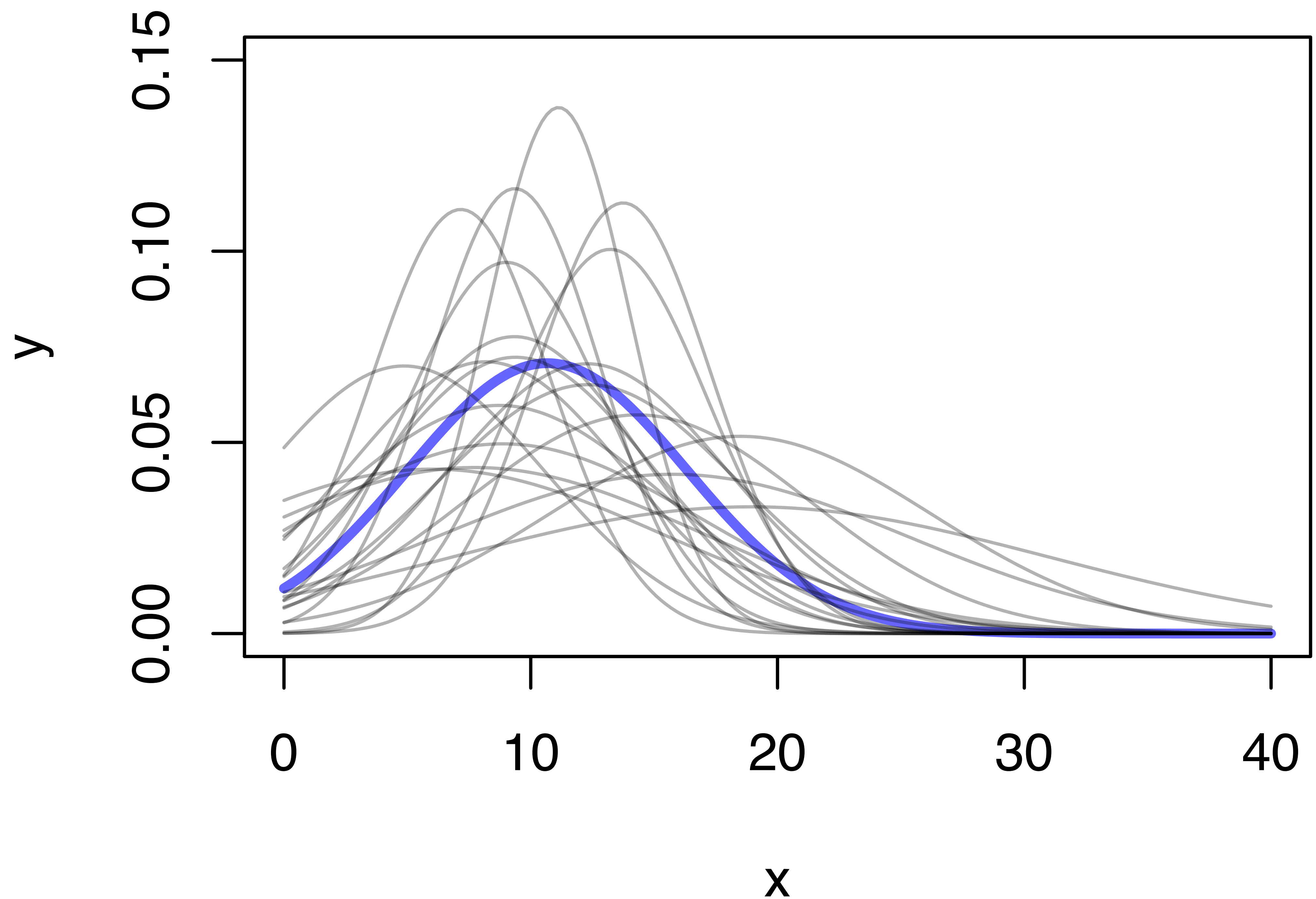


- What about the variation!?

```
samps1 = seq(from = 1, to = length(shopMean), length = 20)
samps2 = round(samps1)
for (i in samps2) {
  y = dnorm(x, mean = shopMean[i], sd = shopMeanSD[i])
  lines(x, y, col = rgb(0, 0, 0, 0.3))
}
```

Draw a somewhat transparent line based on these new y values (with old x values)




0.00
0.05
0.10
0.15
y
0
10
20
30
40
x

Posterior Predictive Check

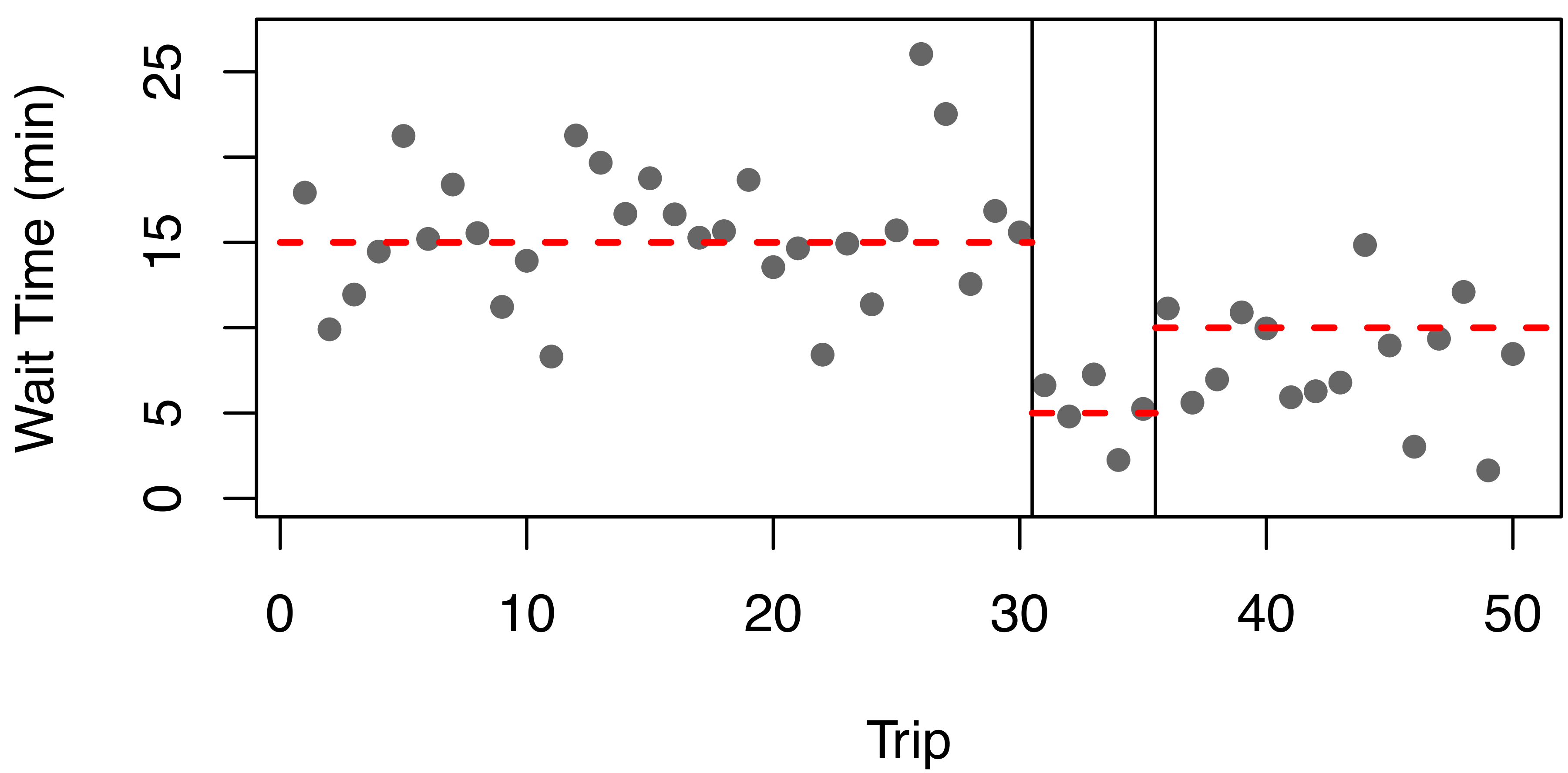
- Calculate mean, low, and high expected wait times for each visit

```
#--- Mean expected value for each visit to coffee shop ---#
ypredMean = apply(ypred, 2, mean)

#--- Upper and lower expected 95% HDI for each visit ---#
ypredLow  = apply(ypred, 2, quantile, probs = 0.025)
ypredHigh = apply(ypred, 2, quantile, probs = 0.975)
```

Posterior Predictive Check

- Plot original values

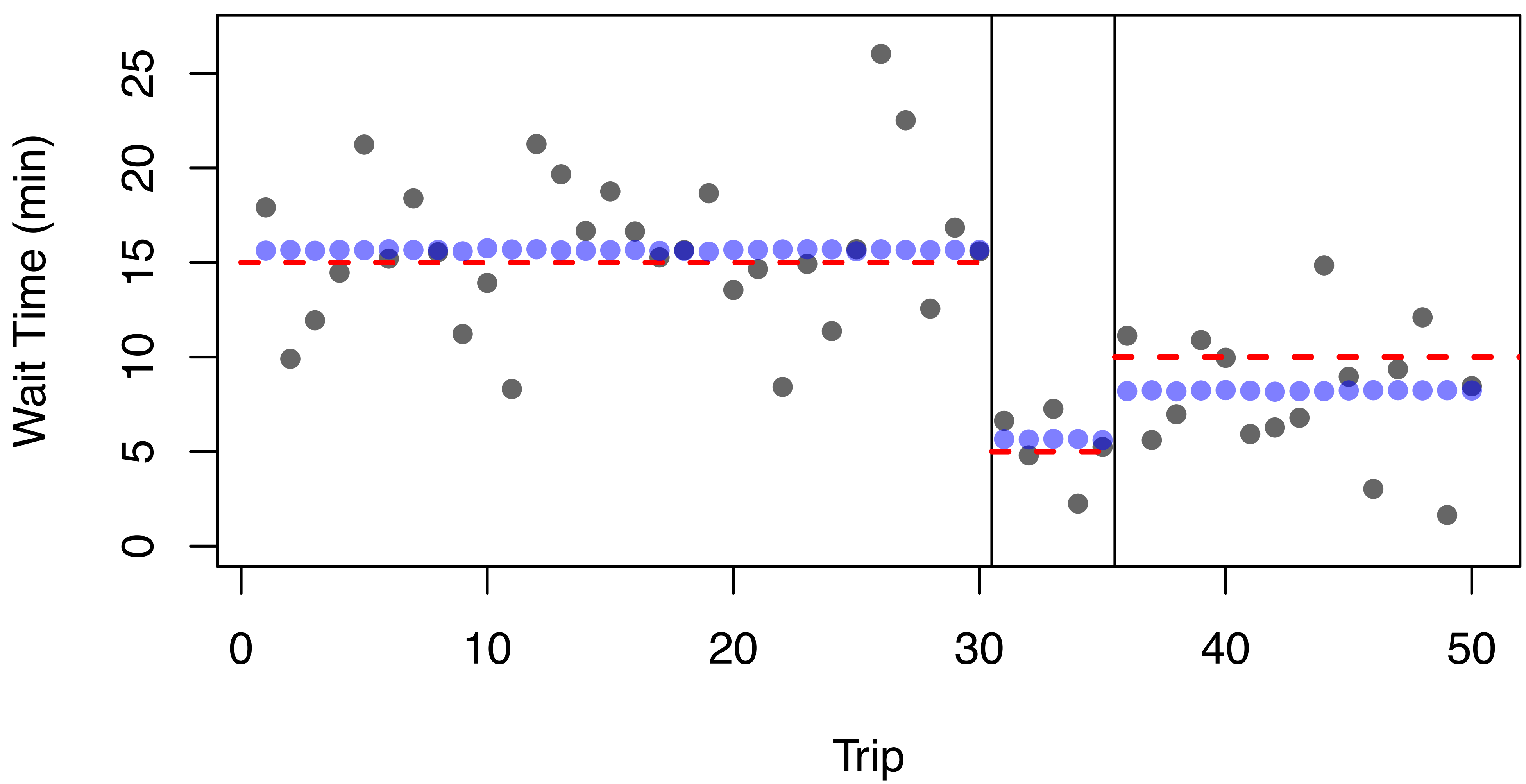
```
par(mfrow = c(1, 1))
plot(coffee$trip, coffee$time, ylim = c(0, 27), xlab = "Trip", ylab = "Wait Time
    (min)", pch = 16, col = rgb(0, 0, 0, 0.6))
abline(v = 30.5)
abline(v = 35.5)
segments(x0 = 0, y0 = 15, x1 = 30.5, y1 = 15, lwd = 2, lty = 2, col = "red")
segments(x0 = 30.5, y0 = 5, x1 = 35.5, y1 = 5, lwd = 2, lty = 2, col = "red")
segments(x0 = 35.5, y0 = 10, x1 = 52, y1 = 10, lwd = 2, lty = 2, col = "red")
```



Posterior Predictive Check

- Add predicted mean values

```
par(new = TRUE)
plot(coffee$trip, ypredMean, ylim = c(0, 27), axes = FALSE, xlab = "", ylab = "",
    main = "", pch = 16, col = rgb(0, 0, 1, 0.5))
```

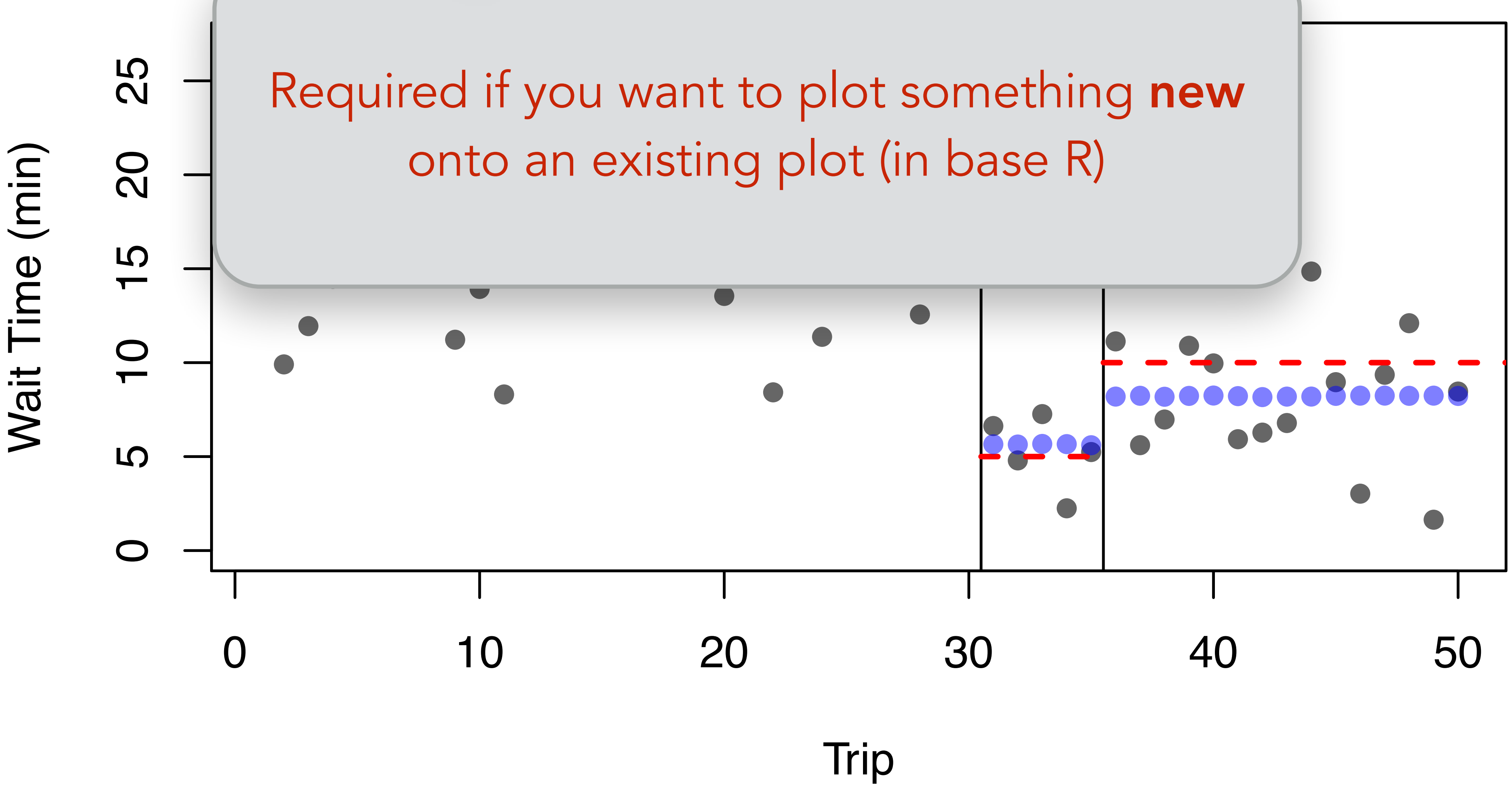


Posterior Predictive Check

- Add predicted mean values

```
par(new = TRUE)
plot(coffee$trip, ypredMean, ylim = c(0, 27), axes = FALSE, xlab = "", ylab = "",
    main = "", pch = 16, col = rgb(0, 0, 1, 0.5))
```

Required if you want to plot something **new** onto an existing plot (in base R)

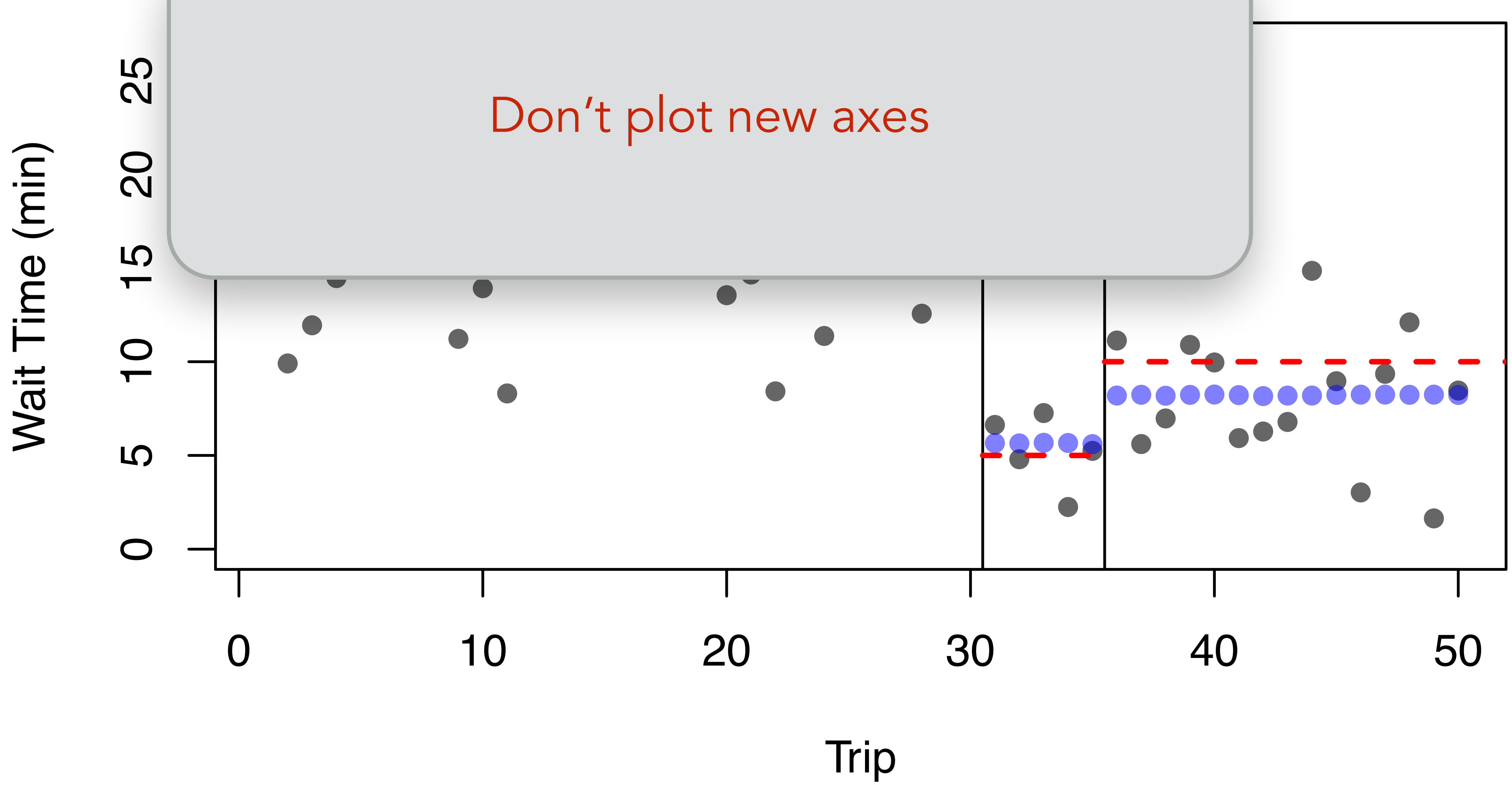


Posterior Predictive Check

- Add predicted mean values

```
par(new = TRUE)
plot(coffee$trip, ypredMean, ylim = c(0, 27), axes = FALSE, xlab = "", ylab = "",
    main = "", pch = 16, col = rgb(0, 0, 1, 0.5))
```

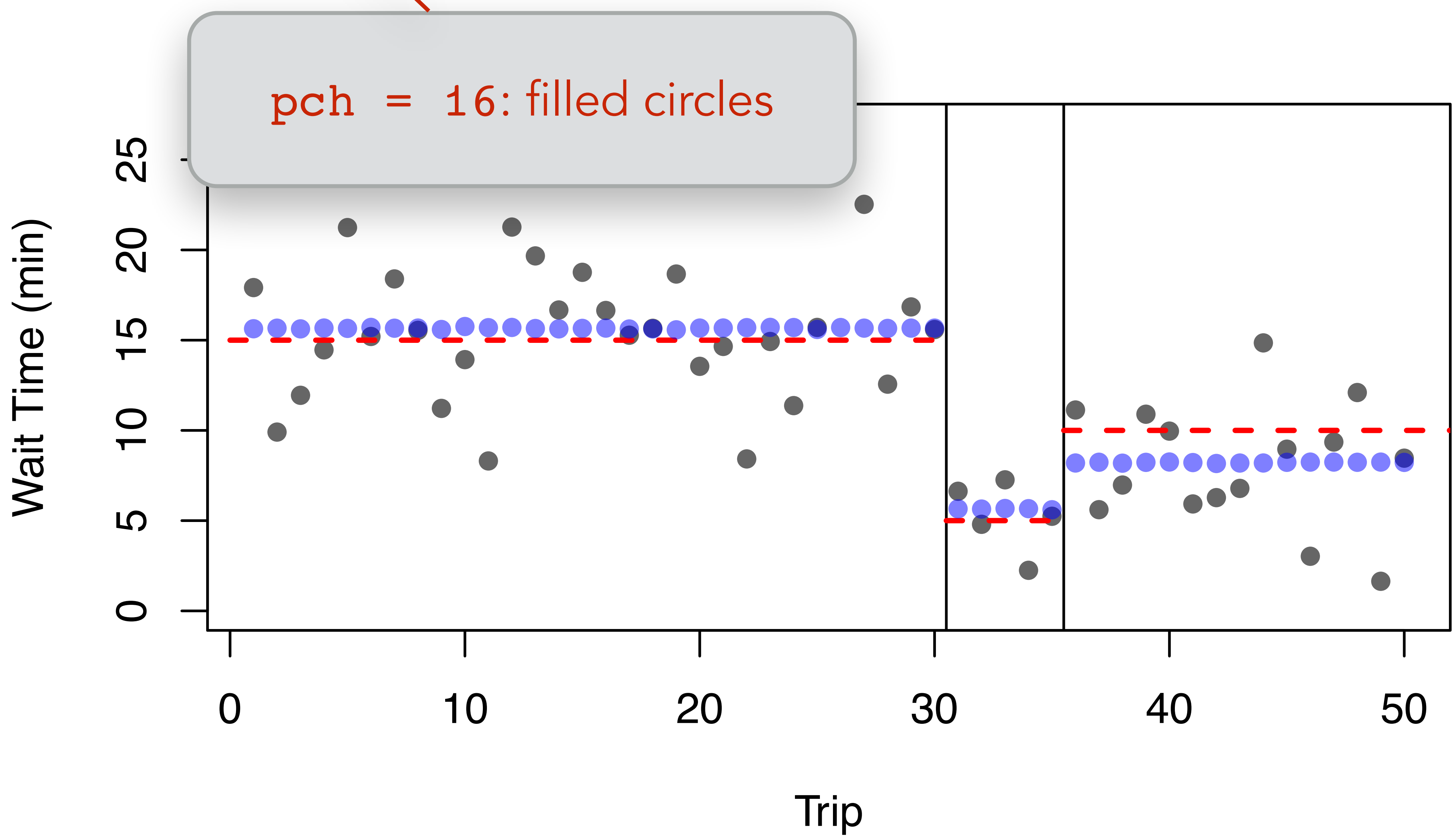
Don't plot new axes



Posterior Predictive Check

- Add predicted mean values

```
par(new = TRUE)
plot(coffee$trip, ypredMean, ylim = c(0, 27), axes = FALSE, xlab = "", ylab = "",
    main = "", pch = 16, col = rgb(0, 0, 1, 0.5))
```

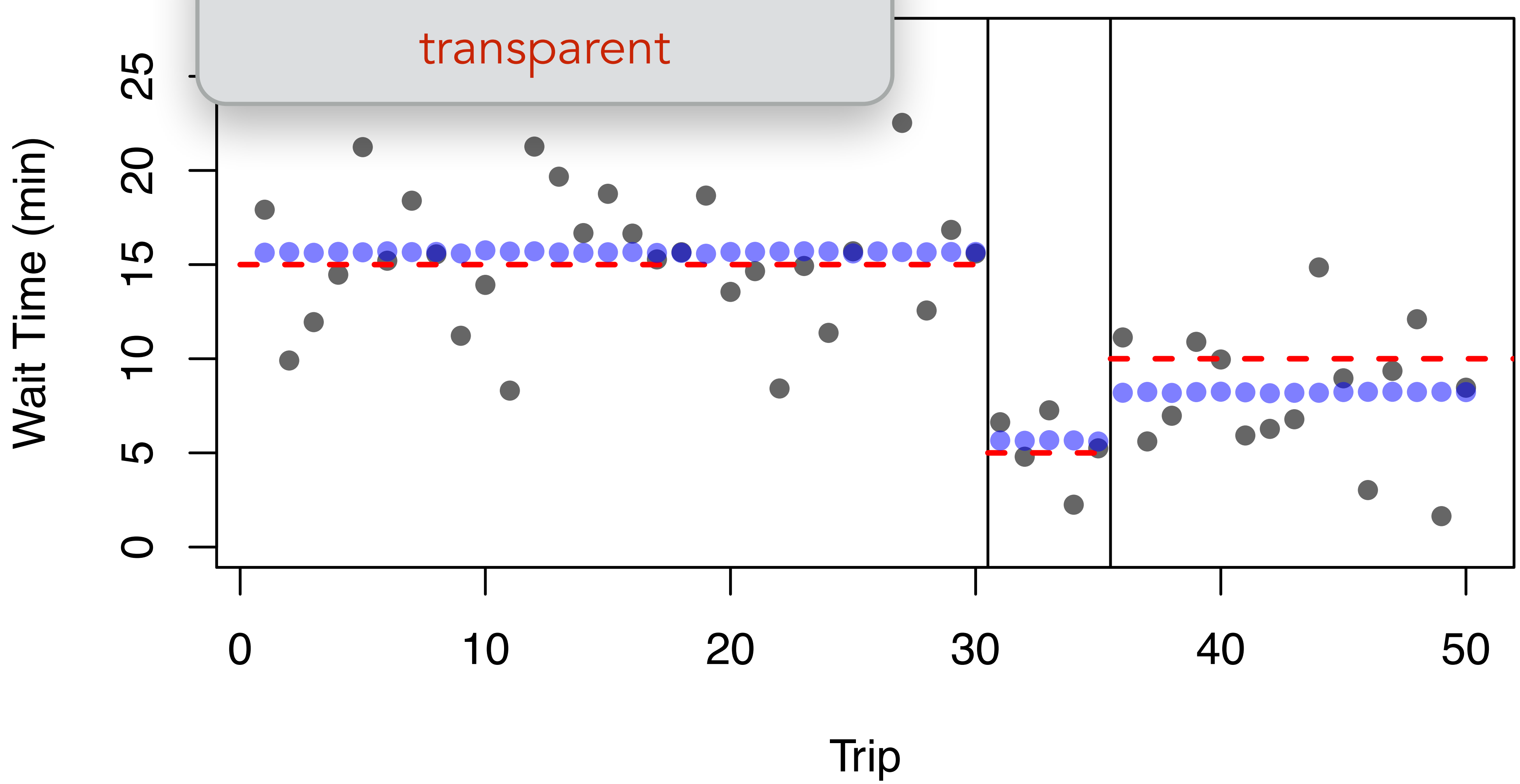


Posterior Predictive Check

- Add predicted mean values

```
par(new = TRUE)
plot(coffee$trip, ypredMean, ylim = c(0, 27), axes = FALSE, xlab = "", ylab = "",
    main = "", pch = 16, col = rgb(0, 0, 1, 0.5))
```

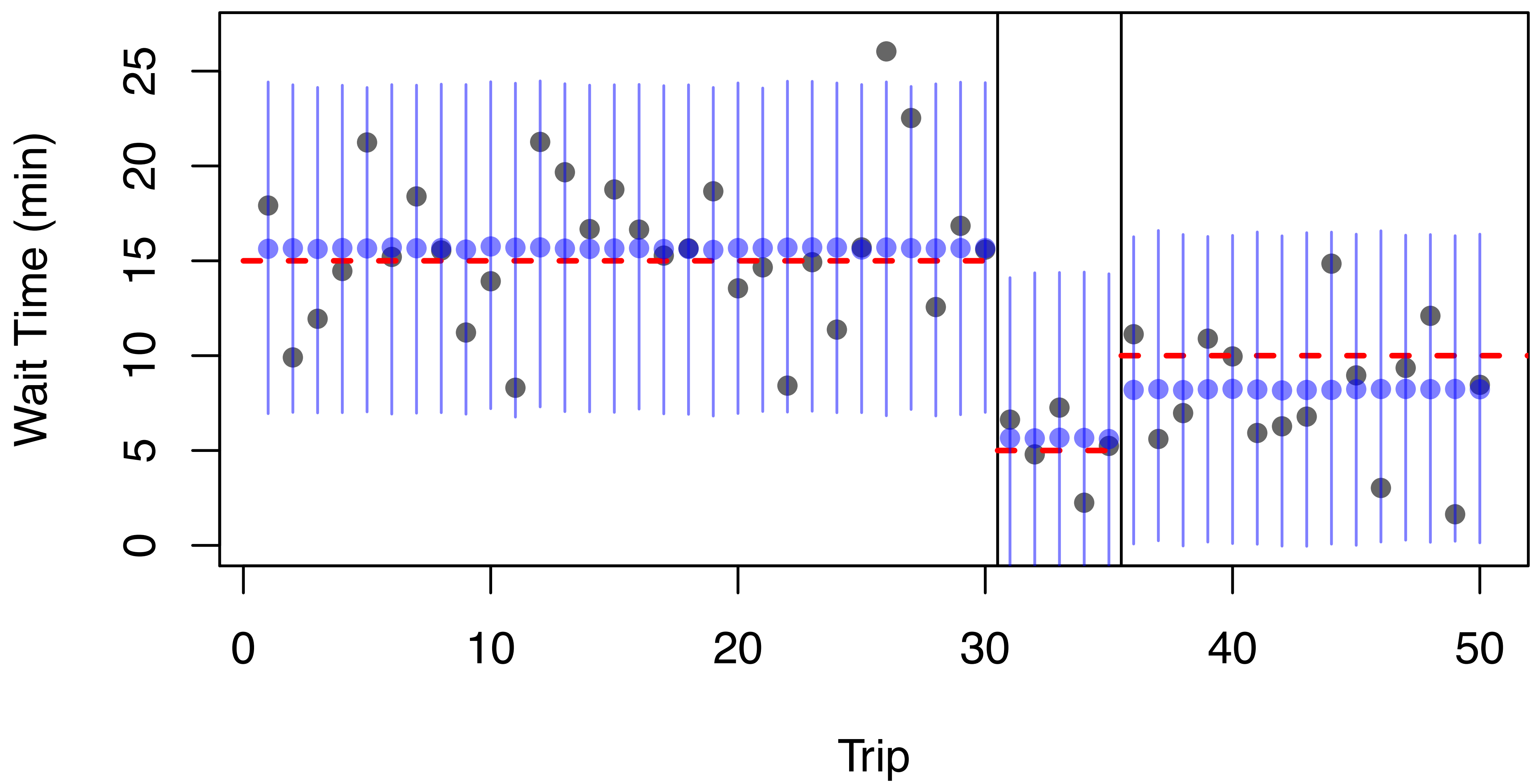
Make circles blue and transparent



Posterior Predictive Check

- Add HDI bars

```
for (i in 1:N) {
  segments(x0 = coffee$trip[i], y0 = ypredLow[i], x1 = coffee$trip[i], y1 =
    ypredHigh[i], col = rgb(0, 0, 1, 0.5))
}
```

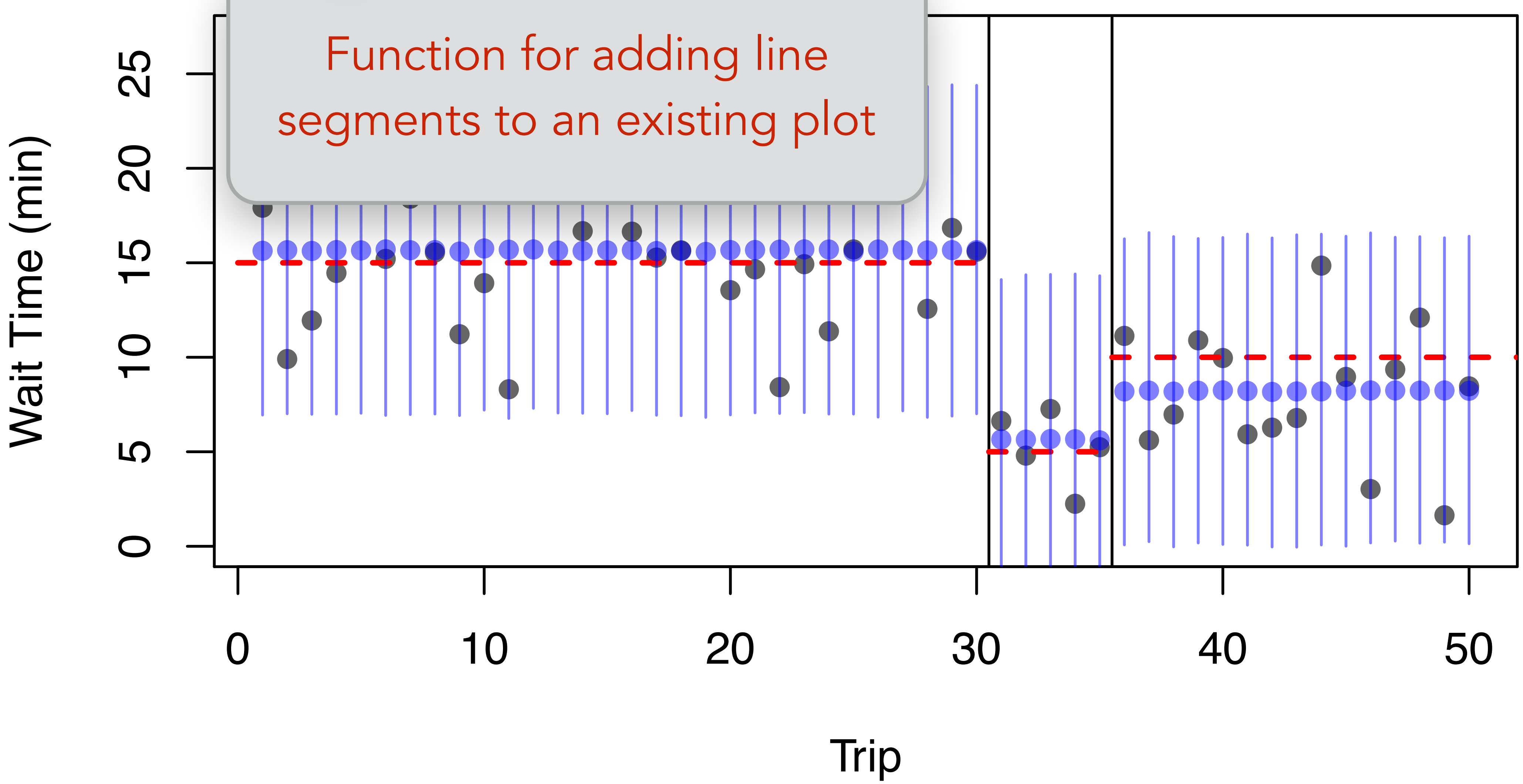


Posterior Predictive Check

- Add HDI bars

```
for (i in 1:N) {
  segments(x0 = coffee$trip[i], y0 = ypredLow[i], x1 = coffee$trip[i], y1 =
    ypredHigh[i], col = rgb(0, 0, 1, 0.5))
}
```

Function for adding line segments to an existing plot

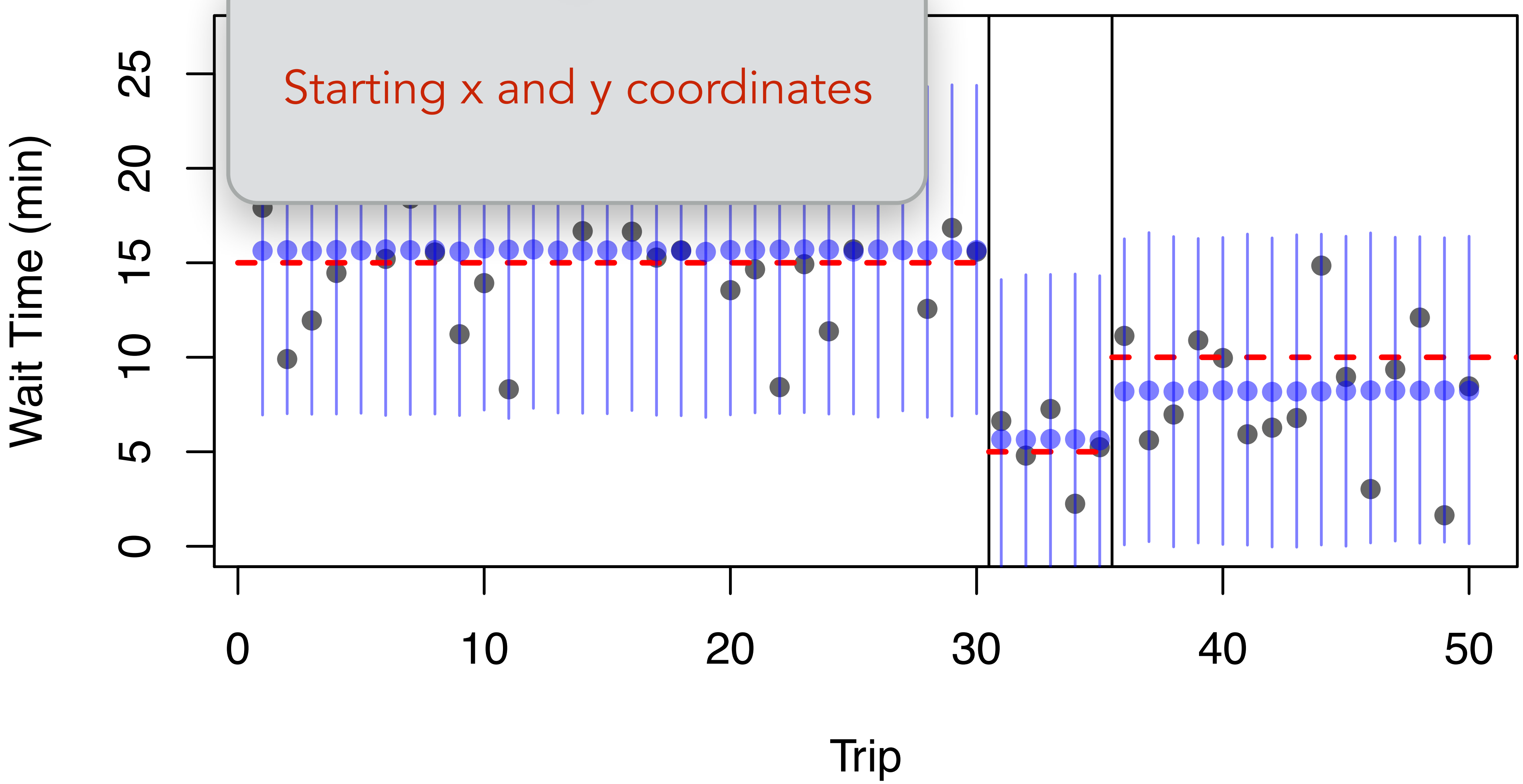


Posterior Predictive Check

- Add HDI bars

```
for (i in 1:N) {
  segments(x0 = coffee$trip[i], y0 = ypredLow[i], x1 = coffee$trip[i], y1 =
   ypredHigh[i], col = rgb(0, 0, 1, 0.5))
}
```

Starting x and y coordinates

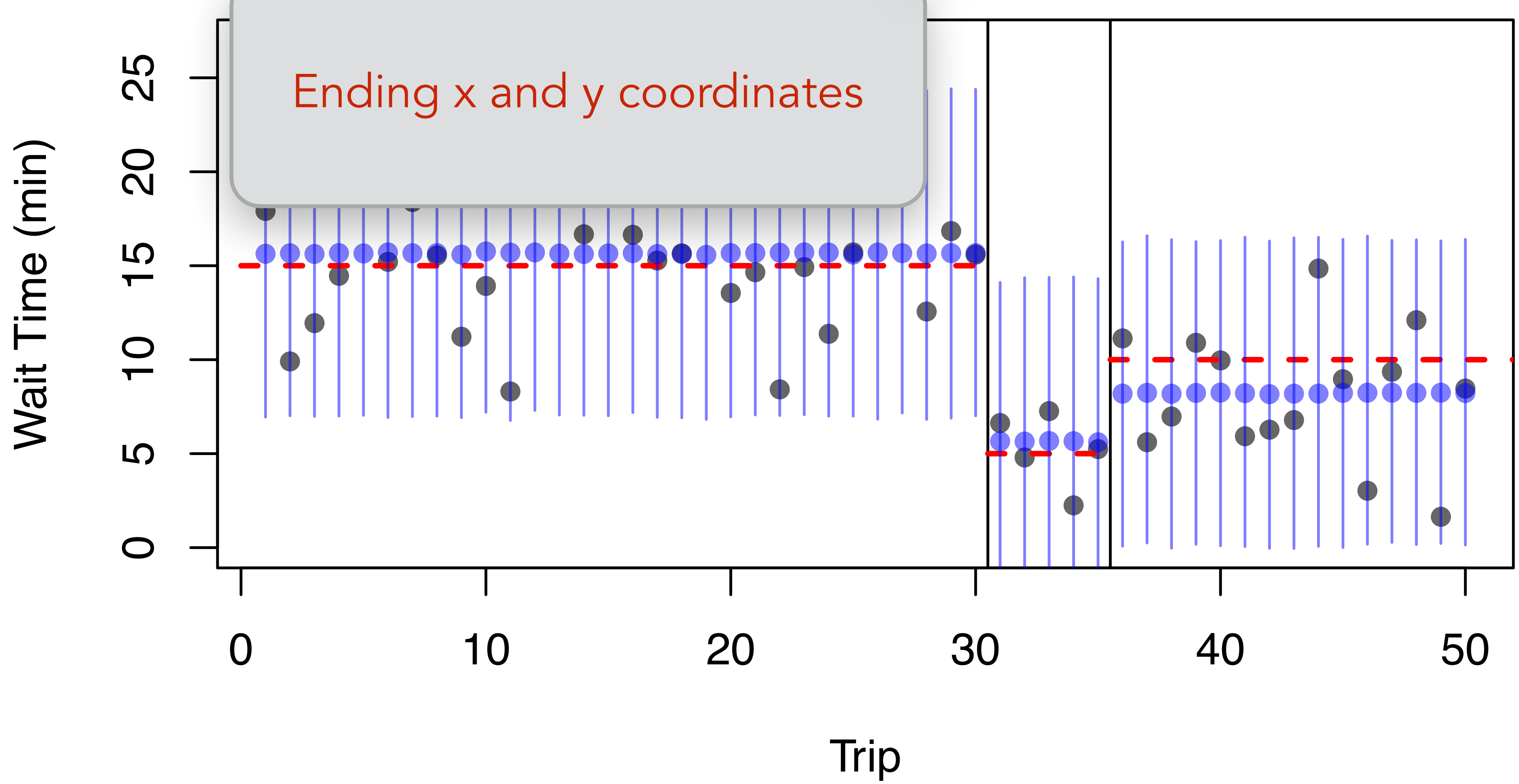


Posterior Predictive Check

- Add HDI bars

```
for (i in 1:N) {
  segments(x0 = coffee$trip[i], y0 = ypredLow[i], x1 = coffee$trip[i], y1 =
   ypredHigh[i], col = rgb(0, 0, 1, 0.5))
}
```

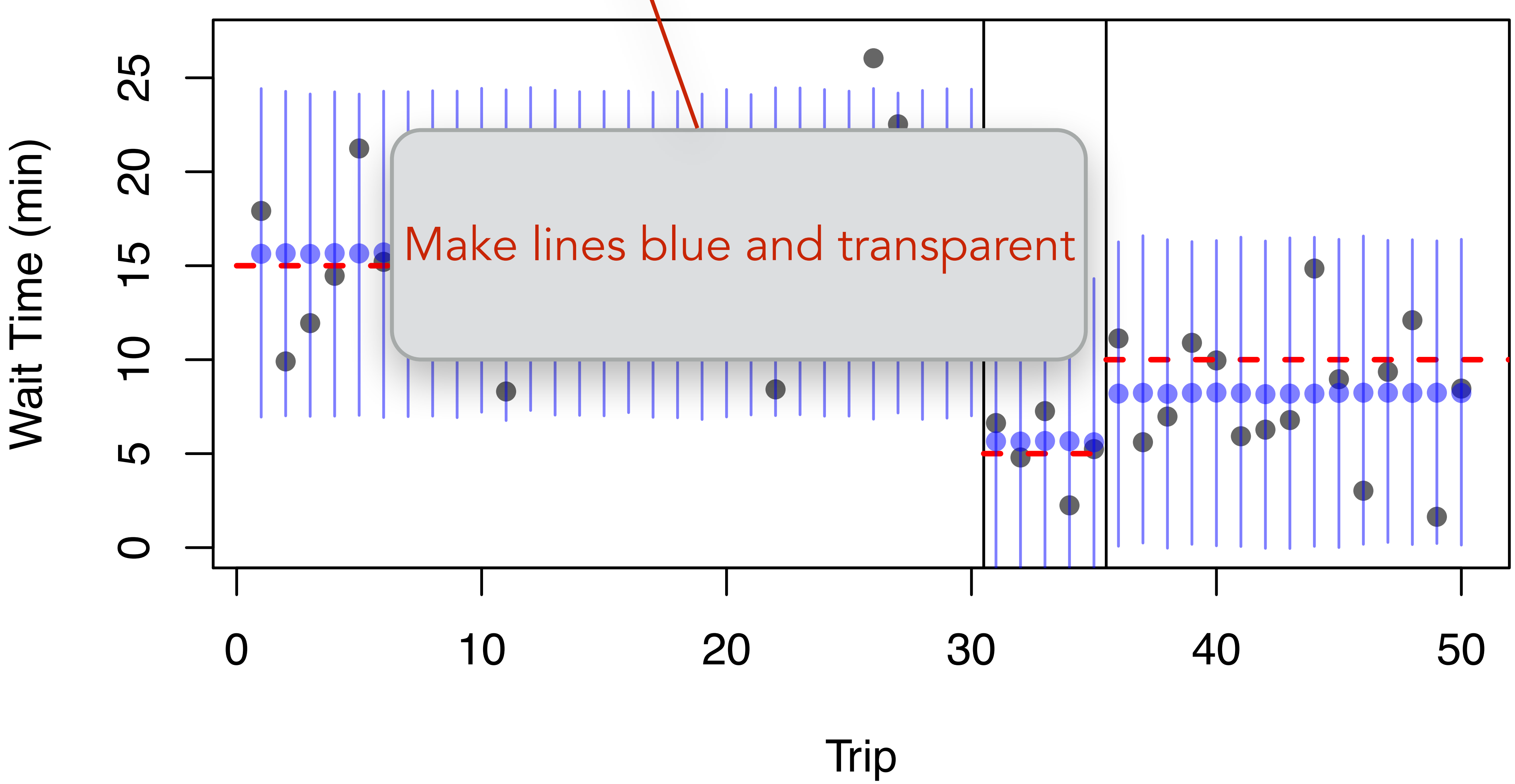
Ending x and y coordinates

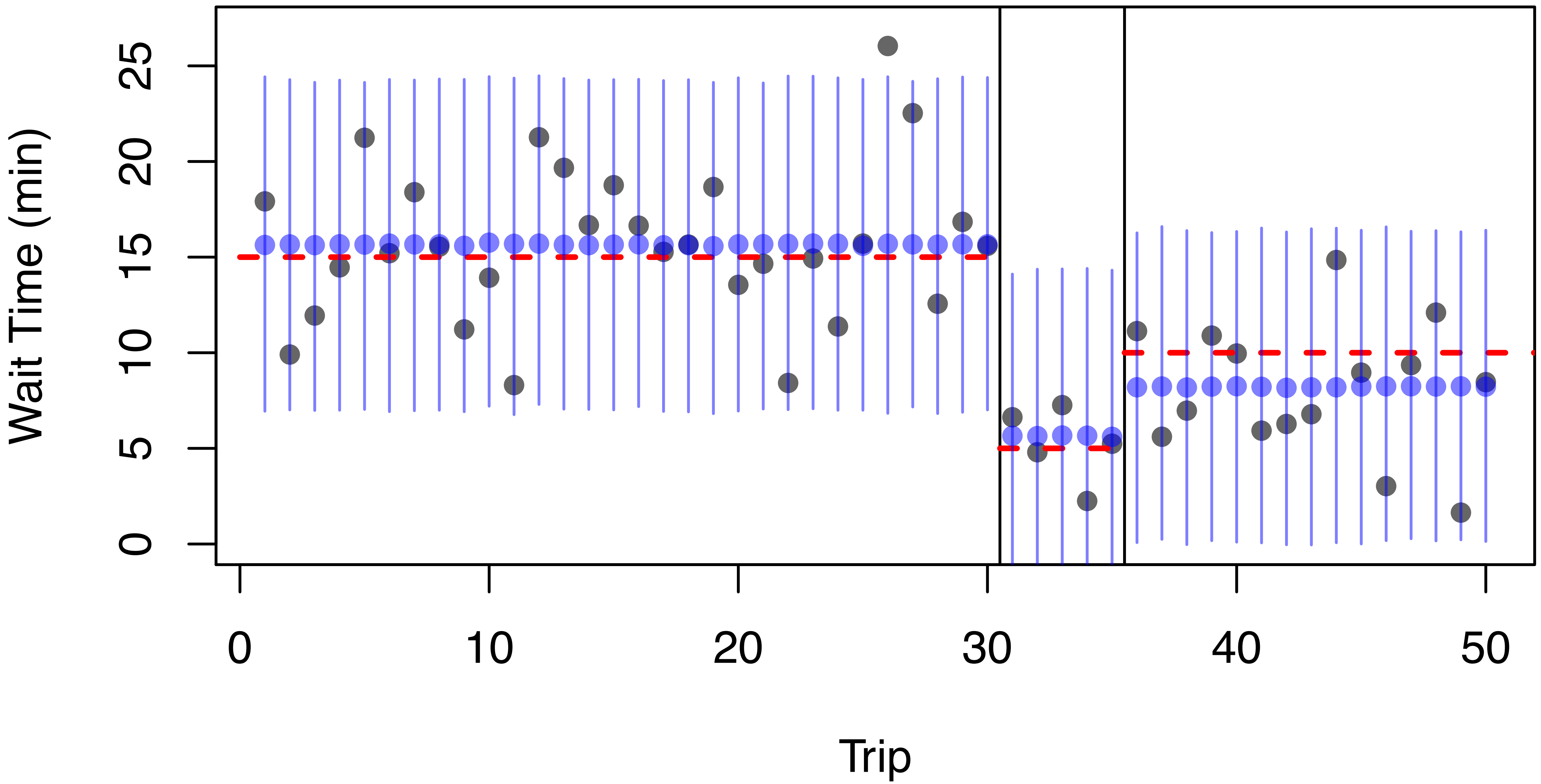


Posterior Predictive Check

- Add HDI bars

```
for (i in 1:N) {
  segments(x0 = coffee$trip[i], y0 = ypredLow[i], x1 = coffee$trip[i], y1 =
   ypredHigh[i], col = rgb(0, 0, 1, 0.5))
}
```





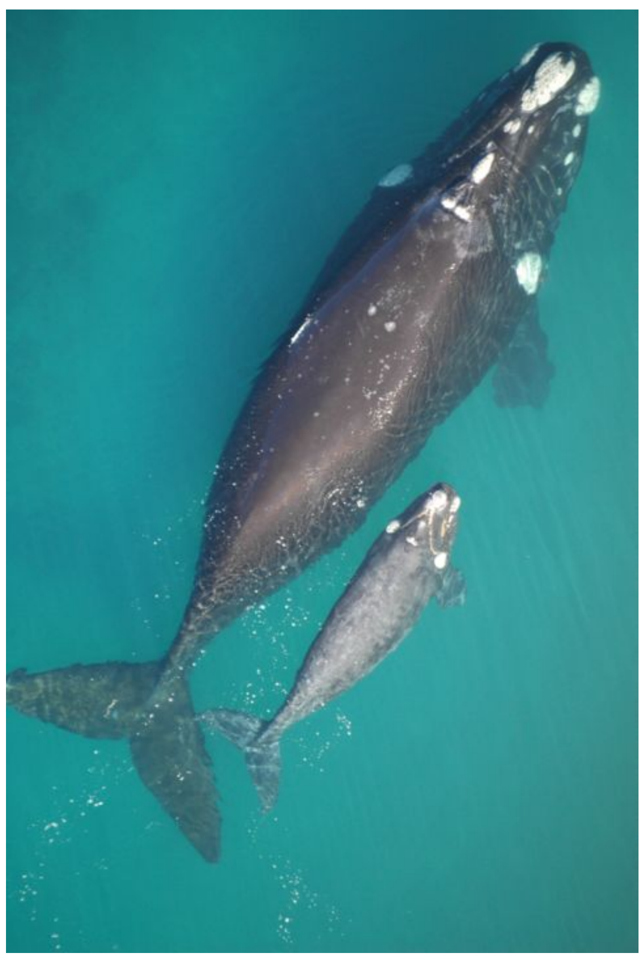
Wait Time (min)
0
5
10
15
20
25
0
10
20
30
40
50
Trip

Questions?

Another Application:
Repeated Measurements from Same Individuals (things)

Data

- Inter-birth intervals for North Atlantic right whales*
- Multi-locus heterozygosity
- Is there a relationship?



whales

Filter

	individual	ibi	mlh
1	1	3	0.82
2	1	3	0.82
3	1	4	0.82
4	1	3	0.82
5	2	6	0.54
6	2	6	0.54
7	3	5	0.62
8	4	8	0.40
9	4	9	0.40
10	4	6	0.40
11	4	8	0.40
12	5	3	0.81
13	6	4	0.74
14	7	5	0.59
15	8	5	0.60
16	9	6	0.43
17	10	5	0.60

*Not real data
Photograph used with permission from the New England Aquarium

Data

Some individuals present multiple times…

…others just once

whales

Filter

	individual	ibi	mlh
1	1	3	0.82
2	1	3	0.82
3	1	4	0.82
4	1	3	0.82
5	2	6	0.54
6	2	6	0.54
7	3	5	0.62
8	4	8	0.40
9	4	9	0.40
10	4	6	0.40
11	4	8	0.40
12	5	3	0.81
13	6	4	0.74
14	7	5	0.59
15	8	5	0.60
16	9	6	0.43
17	10	5	0.60

Options

1. Ignore that some individuals are repeated, and just analyze the data
   - Incorrect because each data point is not independent

whales

	individual	ibi	mlh
1	1	3	0.82
2	1	3	0.82
3	1	4	0.82
4	1	3	0.82
5	2	6	0.54
6	2	6	0.54
7	3	5	0.62
8	4	8	0.40
9	4	9	0.40
10	4	6	0.40
11	4	8	0.40
12	5	3	0.81
13	6	4	0.74
14	7	5	0.59
15	8	5	0.60
16	9	6	0.43
17	10	5	0.60

Options

2. Collapse multiple values to one average for each whale

    - Lose a lot of information, and data points

whales

Filter

	individual	ibi	mlh
1	1	3	0.82
2	1	3	0.82
3	1	4	0.82
4	1	3	0.82
5	2	6	0.54
6	2	6	0.54
7	3	5	0.62
8	4	8	0.40
9	4	9	0.40
10	4	6	0.40
11	4	8	0.40
12	5	3	0.81
13	6	4	0.74
14	7	5	0.59
15	8	5	0.60
16	9	6	0.43
17	10	5	0.60

Options

3. Hierarchical model!!!
   - Keep all data points, but include "individual" as a categorical parameter in the model
   - Make their effects hierarchical
   - Accounts for individual effects on the data

whales

Filter

	individual	ibi	mlh
1	1	3	0.82
2	1	3	0.82
3	1	4	0.82
4	1	3	0.82
5	2	6	0.54
6	2	6	0.54
7	3	5	0.62
8	4	8	0.40
9	4	9	0.40
10	4	6	0.40
11	4	8	0.40
12	5	3	0.81
13	6	4	0.74
14	7	5	0.59
15	8	5	0.60
16	9	6	0.43
17	10	5	0.60

Summary

- Should treat nominal data in a hierarchical way by **default**, unless you have a good reason for not doing so
- Hierarchical models are a great way to get the most of your data
  - Don't lose information by averaging across all groups
  - Don't violate assumptions by assuming all independent
- Hierarchical models never hurt (except your brain)
  - Will perform at least as well as a non-hierarchical counterpart
- No limitation to the number of levels
  - Can have groups of groups within groups, etc...
  - As long as you can keep it straight yourself

Questions?