STAT 656: HW3

Rohan Dekate

Gibbs sampling for normal hierarchical models

```
library(coda)
```

2. Write down the form of the conditional distributions above. Here, the material in Lecture 3, 7 and 13 will be useful.

The full joint probability distribution of the hierarchical model is the product of the likelihood and all priors.

$$p(\mu, \tau^2, \{\theta_j\}, \{\sigma_j^2\}, Y) = \left(\prod_{j=1}^{J}\prod_{i=1}^{n_j} p(y_{ij}|\theta_j, \sigma_j^2)\right) \times \left(\prod_{j=1}^{J} p(\theta_j|\mu, \tau^2)\right) \times p(\mu|\tau^2)p(\tau^2)\left(\prod_{j=1}^{J} p(\sigma_j^2)\right)$$

Likelihood: $p(y_{ij}|\theta_j, \sigma_j^2) = \mathcal{N}(y_{ij}|\theta_j, \sigma_j^2)$ for $i = 1, \cdots, n_j$ and $j = 1, \cdots, J$. We can summarize this using the sample mean $\bar{y}_j = \frac{1}{n_j}\sum_i y_{ij}$ which has the distribution:

$$p(\bar{y_{ij}}|\theta_j, \sigma_j^2) = \mathcal{N}(\bar{y_j}|\theta_j, \sigma_j^2/n_j)$$

Group-level Prior:

$$p(\theta_j|\mu, \tau^2) = \mathcal{N}(\theta_j|\mu, \tau^2)$$

Hyperpriors: $p(\mu|\tau^2) \propto 1$ which implies a flat prior on μ. We will assume $p(\tau^2) \propto 1/\tau = (\tau^2)^{-1/2}$ which is the same functional form as that of the unknown variances $p(\sigma_j^2) \propto 1/\sigma_j = (\sigma_j^2)^{-1/2}$.

For both the models (Unknown but identical variances and Unknown and independent variances) the conditional distribution for the hyperpriors will be the same.

1. (*For both models*) Sample from $\mu, \tau^2|\theta_1, \cdots, \theta_j, \sigma_1^2, \cdots, \sigma_j^2, Y$. To do this we sample sequentially from $p(\mu|\tau^2, \{\theta_j\})$ and $p(\tau^2|\mu, \{\theta_j\})$
    1. Full conditional for μ:

       $p(\mu|\tau^2, \theta_1, \cdots, \theta_J) \propto \prod_{j=1}^{J} p(\theta_j|\mu, \tau^2)$.

       Given the flat prior $p(\mu|\tau^2) \propto 1$, we have:

       $p(\mu|\tau^2, \{\theta_j\}) \propto \prod_{j=1}^{J} \mathcal{N}(\theta_j|\mu, \tau^2) \propto \exp\left(-\frac{1}{2\tau^2}\sum_{j=1}^{J}(\theta_j - \mu)^2\right)$

       This is the kernel for a normal distribution. By completing the square for μ, the conditional distribution is:

$$\mu|\tau^2, \theta_1, \cdots, \theta_J \sim \mathcal{N}(\bar{\theta}, \frac{\tau^2}{J})$$

where $\bar{\theta} = \frac{1}{J}\sum_{j=1}^{J}\theta_j$.

2. Full conditional for τ^2:

$$p(\tau^2|\mu,\theta_1,\cdots,\theta_J) \propto p(\tau^2)\prod_{j=1}^{J} p(\theta_j|\mu,\tau^2) \propto (\tau^2)^{-1/2} \times \prod_{j=1}^{J}\frac{1}{\sqrt{2\pi\tau^2}}\exp\left(-\frac{(\theta_j-\mu)^2}{2\tau^2}\right)$$

$$\propto (\tau^2)^{-1/2} \times (\tau^2)^{-J/2}\exp\left(-\frac{1}{2\tau^2}\sum_{j=1}^{J}(\theta_j-\mu)^2\right)$$

$$\propto (\tau^2)^{-\frac{J+1}{2}}\exp\left(-\frac{1}{2\tau^2}\sum_{j=1}^{J}(\theta_j-\mu)^2\right)$$

This is the kernel of an Inverse-Gamma distribution.

$$\text{Inv-Gamma}(\alpha,\beta) \propto (x)^{-(\alpha+1)}e^{\beta/x}$$

We have $\alpha+1 = \frac{J+1}{2} \implies \alpha = \frac{J-1}{2}$ and $\beta = \frac{1}{2}\sum(\theta_j-\mu)^2$.

$$\tau^2|\mu,\theta_1,\cdots,\theta_J \sim \text{Inv-Gamma}\left(\frac{J-1}{2}, \frac{1}{2}\sum_{j=1}^{J}(\theta_j-\mu)^2\right).$$

2. Model 1: Unknown but identical variances: $\sigma_1^2,\cdots,\sigma_j^2 = \sigma^2$.

Full Conditional for θ_j

$$p(\theta_j|\text{rest}) \propto p(\theta_j|\mu,\tau^2) \times (\bar{y_j}|\theta_j,\sigma^2) \propto \exp\left(-\frac{(\theta_j-\mu)^2}{2\tau^2}\right) \times \exp\left(-\frac{(\bar{y_j}-\theta_j)^2}{2\sigma^2/n_j}\right)$$

Thus, we have a normal prior and a normal likelihood. The posterior will also be normal.

Posterior Precision:

$$\frac{1}{V_j} = \frac{1}{\tau^2} + \frac{1}{\sigma^2/n_j} = \frac{1}{\tau^2} + \frac{n_j}{\sigma^2}.$$

Posterior Mean:

$$B_j = V_j\left(\frac{\mu}{\tau^2} + \frac{\bar{y_j}}{\sigma^2/n_j}\right) = V_j\left(\frac{\mu}{\tau^2} + \frac{n_j\bar{y_j}}{\sigma^2}\right)$$

$$\theta_j|\mu,\tau^2,\sigma^2,Y \sim \mathcal{N}(B_j,V_j)$$

Full Conditional for σ^2

$$p(\sigma^2|\text{rest}) \propto p(\sigma^2) \times \prod_{j=1}^{J}\prod_{i=1}^{n_j} p(y_{ij}|\theta_j,\sigma^2)$$

$$\propto (\sigma^2)^{-1/2} \times \prod_{j=1}^{J} \prod_{i=1}^{n_j} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_{ij}-\theta_j)^2}{2\sigma^2}\right)$$

$$\propto (\sigma^2)^{-1/2} \times (\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

$$\propto (\sigma^2)^{-\frac{N+1}{2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

This is the kernel of Inverse-Gamma distribution.

$\alpha + 1 = \frac{N+1}{2} \implies \alpha = \frac{N-1}{2}$ and $\beta = \frac{1}{2} \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2$

Therefore,

$$\sigma^2 | \theta_j, Y \sim \text{Inv-Gamma}\left(\frac{N-1}{2}, \frac{1}{2} \sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

3. Model 2: Unknown and independent variances: $\sigma_1^2, \cdots, \sigma_j^2$ are distinct.

   Full conditional for θ_j

$$p(\theta_j|\text{rest}) \propto p(\theta_j|\mu,\tau^2) \times (\bar{y_j}|\theta_j, \sigma_j^2)\theta_j|\mu,\tau^2,\sigma_j^2, Y \sim \mathcal{N}(B_j, V_j) V_j = \left(\frac{1}{\tau^2} + \frac{n_j}{\sigma_j^2}\right)^{-1} B_j = V_j \left(\frac{\mu}{\tau^2} + \frac{n_j \bar{y_j}}{\sigma_j^2}\right)$$

   Full conditional for σ_j^2

$$p(\sigma_j^2|\text{rest}) \propto p(\sigma_j^2) \times \prod_{i=1}^{n_j} p(y_{ij}|\theta_j, \sigma_j^2)$$

$$\propto (\sigma_j^2)^{-1/2} \times (\sigma_j^2)^{-n_j/2} \exp\left(-\frac{1}{2\sigma_j^2} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

$$\propto (\sigma_j^2)^{-\frac{n_j+1}{2}} \exp\left(-\frac{1}{2\sigma_j^2} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

   This is the kernel of Inverse-Gamma distribution with parameters specific to group j.

   $\alpha_j + 1 = \frac{n_j+1}{2} \implies \alpha_j = \frac{n_j-1}{2}$ and $\beta = \frac{1}{2} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2$

   Therefore,

$$\sigma_j^2 | \theta_j, Y \sim \text{Inv-Gamma}\left(\frac{n_j-1}{2}, \frac{1}{2} \sum_{i=1}^{n_j} (y_{ij}-\theta_j)^2\right)$$

3. Write down R code to implement the Gibbs samplers. Here code from the lectures will be very useful.

```
NBA_data = read.csv("NBA_data.csv")
NBA_data = NBA_data[,-1]  # Drop column 1 (just indices)
head(NBA_data)
```

```
##   number_games_played    Y_bar sample_sd number_remaining_games
## 1                  10 108.0000  5.868939                     72
## 2                  10 107.2000  8.841820                     72
## 3                   8 106.8750  6.957781                     74
## 4                  10 104.8000 11.545080                     72
## 5                  10 103.8000 13.870830                     72
## 6                   9 103.4444 14.292580                     73
##   average_remaining_games        team
## 1               110.50556     Phoenix
## 2               100.68556        Utah
## 3               105.18514      Denver
## 4               106.78167 GoldenState
## 5                94.70028     Boston
## 6               104.45041 Washington
```

```
n_j <- NBA_data$number_games_played
y_bar <- NBA_data$Y_bar
s_j_sq <- NBA_data$sample_sd^2
J <- length(y_bar)
N <- sum(n_j)
team_names <- NBA_data$team

# Data for prediction
n_rem   <- NBA_data$number_remaining_games
y_rem_obs <- NBA_data$average_remaining_games
```

```
gibbs_sampler_model1 <- function(y_bar, n_j, s_j_sq, J, N, n_iter, burn_in) {
  # Store posterior samples after burn-in
  n_draws <- n_iter - burn_in
  posterior_draws <- list(
    mu = matrix(NA, nrow = n_draws, ncol = 1),
    tau_sq = matrix(NA, nrow = n_draws, ncol = 1),
    sigma_sq = matrix(NA, nrow = n_draws, ncol = 1),
    theta = matrix(NA, nrow = n_draws, ncol = J)
  )

  # Initialize parameters
  mu_curr <- mean(y_bar)
  tau_sq_curr <- var(y_bar)
  sigma_sq_curr <- mean(s_j_sq)
  theta_curr <- y_bar

  # Sampling loop
  for (s in 1:n_iter) {
    # Update mu
    post_var_mu <- tau_sq_curr / J
    post_mean_mu <- mean(theta_curr)
    mu_curr <- rnorm(1, mean = post_mean_mu, sd = sqrt(post_var_mu))
```

```
    # Update tau
    post_shape_tau <- (J - 1) / 2
    post_scale_tau  <- 0.5 * sum((theta_curr - mu_curr)^2)
    tau_sq_curr <- 1 / rgamma(1, shape = post_shape_tau, rate = post_scale_tau)

    # Update theta
    post_var_theta <- 1 / (1 / tau_sq_curr + n_j / sigma_sq_curr)
    post_mean_theta <- post_var_theta * (mu_curr / tau_sq_curr + (n_j * y_bar) / sigma_sq_curr)
    theta_curr <- rnorm(J, mean = post_mean_theta, sd = sqrt(post_var_theta))

    # Update sigma
    sum_of_squares <- sum((n_j - 1) * s_j_sq + n_j * (y_bar - theta_curr)^2)
    post_shape_sigma <- (N - 1) / 2
    post_scale_sigma  <- 0.5 * sum_of_squares
    sigma_sq_curr <- 1 / rgamma(1, shape = post_shape_sigma, rate = post_scale_sigma)

    # Store samples after burn-in
    if (s > burn_in) {
      draw_index <- s - burn_in
      posterior_draws$mu[draw_index, ]       <- mu_curr
      posterior_draws$tau_sq[draw_index, ]   <- tau_sq_curr
      posterior_draws$sigma_sq[draw_index, ] <- sigma_sq_curr
      posterior_draws$theta[draw_index, ]    <- theta_curr
    }
    if (s %% 1000 == 0) cat("  Iteration", s, "/", n_iter, "\n")
  }
   return(posterior_draws)
}
```

```
gibbs_sampler_model2 <- function(y_bar, n_j, s_j_sq, J, n_iter, burn_in) {
  # Store posterior samples after burn-in
  n_draws <- n_iter - burn_in
  posterior_draws <- list(
    mu = matrix(NA, nrow = n_draws, ncol = 1),
    tau_sq = matrix(NA, nrow = n_draws, ncol = 1),
    sigma_sq = matrix(NA, nrow = n_draws, ncol = J), # Distinct sigma
    theta = matrix(NA, nrow = n_draws, ncol = J)
  )

  # Initialize parameters
  mu_curr <- mean(y_bar)
  tau_sq_curr <- var(y_bar)
  sigma_sq_curr <- s_j_sq
  theta_curr <- y_bar

  # Sampling loop
  for (s in 1:n_iter) {
    # Update mu
    post_var_mu <- tau_sq_curr / J
    post_mean_mu <- mean(theta_curr)
    mu_curr <- rnorm(1, mean = post_mean_mu, sd = sqrt(post_var_mu))

    # Update tau
```

```
    post_shape_tau <- (J - 1) / 2
    post_scale_tau  <- 0.5 * sum((theta_curr - mu_curr)^2)
    tau_sq_curr <- 1 / rgamma(1, shape = post_shape_tau, rate = post_scale_tau)

    # Update theta
    post_var_theta <- 1 / (1 / tau_sq_curr + n_j / sigma_sq_curr)
    post_mean_theta <- post_var_theta * (mu_curr / tau_sq_curr + (n_j * y_bar) / sigma_sq_curr)
    theta_curr <- rnorm(J, mean = post_mean_theta, sd = sqrt(post_var_theta))

    # Update sigma
    sum_of_squares_j <- (n_j - 1) * s_j_sq + n_j * (y_bar - theta_curr)^2
    post_shape_sigma <- (n_j - 1) / 2
    post_scale_sigma  <- 0.5 * sum_of_squares_j
    sigma_sq_curr <- 1 / rgamma(J, shape = post_shape_sigma, rate = post_scale_sigma)

    # Store samples after burn-in
    if (s > burn_in) {
      draw_index <- s - burn_in
      posterior_draws$mu[draw_index, ]       <- mu_curr
      posterior_draws$tau_sq[draw_index, ]   <- tau_sq_curr
      posterior_draws$sigma_sq[draw_index, ] <- sigma_sq_curr
      posterior_draws$theta[draw_index, ]    <- theta_curr
    }
    if (s %% 1000 == 0) cat("  Iteration", s, "/", n_iter, "\n")
  }
   return(posterior_draws)
}
```

4. Run the two Gibbs samplers on the NBA dataset and summarize the results. Specifically, for μ, τ^2 and a few θ_j and σ_j^2 , plot the MCMC traceplots, and diagnose mixing. Also plot the corresponding posterior distributions.

```
set.seed(656)
N_ITER <- 10000
BURN_IN <- 2000
```

```
results_model1 <- gibbs_sampler_model1(y_bar, n_j, s_j_sq, J, N,
                                       n_iter = N_ITER, burn_in = BURN_IN)
```

```
##   Iteration 1000 / 10000
##   Iteration 2000 / 10000
##   Iteration 3000 / 10000
##   Iteration 4000 / 10000
##   Iteration 5000 / 10000
##   Iteration 6000 / 10000
##   Iteration 7000 / 10000
##   Iteration 8000 / 10000
##   Iteration 9000 / 10000
##   Iteration 10000 / 10000
```

```
results_model2 <- gibbs_sampler_model2(y_bar, n_j, s_j_sq, J,
                                       n_iter = N_ITER, burn_in = BURN_IN)
```

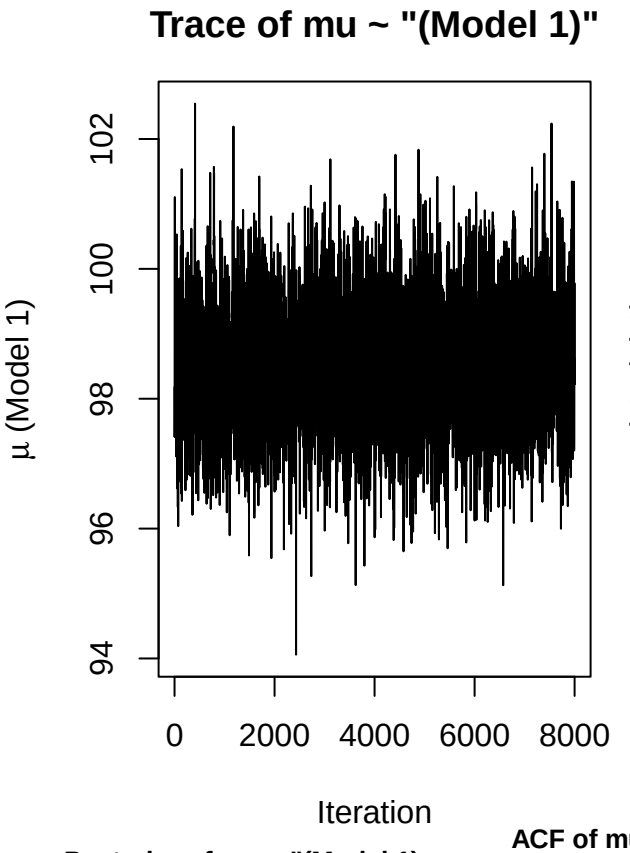
```
##   Iteration 1000 / 10000
```

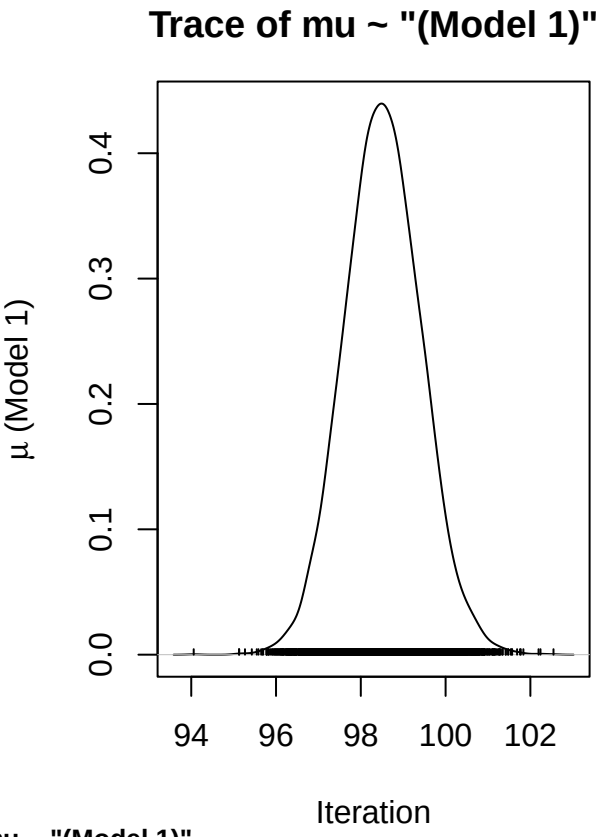
```
##   Iteration 2000 / 10000
##   Iteration 3000 / 10000
##   Iteration 4000 / 10000
##   Iteration 5000 / 10000
##   Iteration 6000 / 10000
##   Iteration 7000 / 10000
##   Iteration 8000 / 10000
##   Iteration 9000 / 10000
##   Iteration 10000 / 10000
```

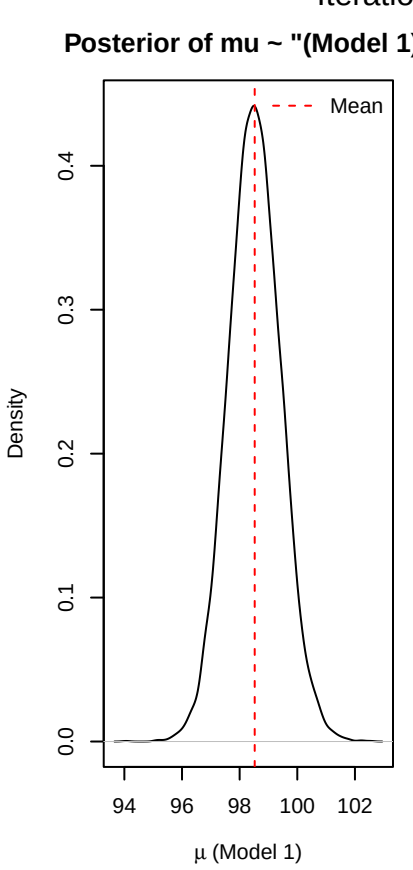
```r
plot_diagnostics <- function(draws, param_name) {

  par(mfrow = c(1, 3), mar = c(4, 4, 3, 1))

  # Trace Plot
  plot(as.mcmc(draws), type = 'l',
       main = paste("Trace of", param_name),
       ylab = param_name,
       xlab = "Iteration")

  # Density Plot
  plot(density(draws),
       main = paste("Posterior of", param_name),
       xlab = param_name)
  # Add a line for the posterior mean
  abline(v = mean(draws), col = "red", lty = 2)
  legend("topright", "Mean", col="red", lty=2, bty="n")

  # Autocorrelation (ACF) Plot
  acf(draws,
      main = paste("ACF of", param_name),
      lag.max = 30) # Only show up to lag 30

  # Reset plotting window
  par(mfrow = c(1, 1))
}
```

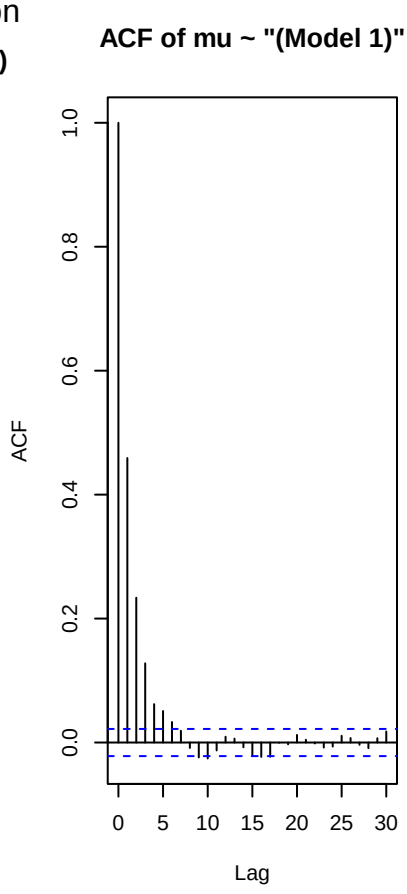
Model 1 Posterior Analysis

```r
plot_diagnostics(results_model1$mu, expression(mu ~ "(Model 1)"))
```


Trace of mu ~ "(Model 1)"
μ (Model 1)
94
96
98
100
102
0
2000
4000
6000
8000
Iteration


Trace of mu ~ "(Model 1)"
μ (Model 1)
0.0
0.1
0.2
0.3
0.4
94
96
98
100
102
Iteration

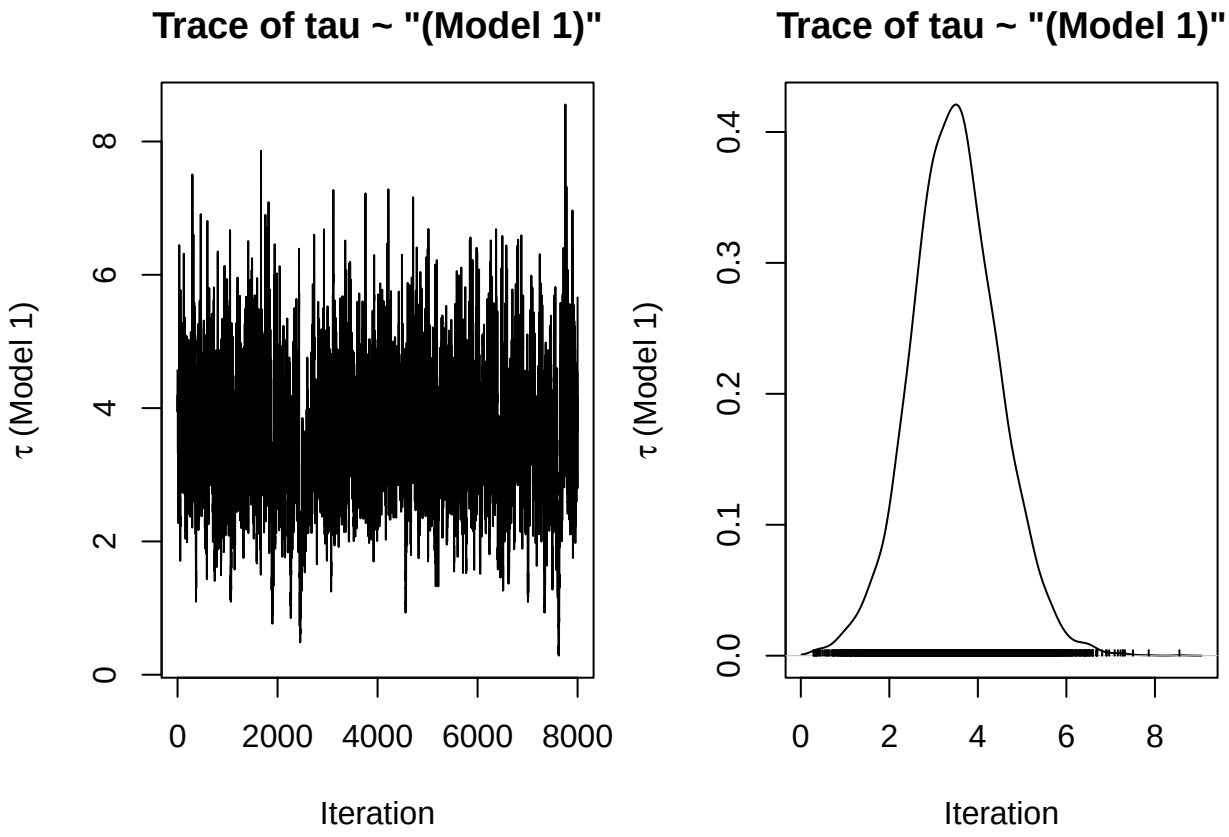

Posterior of mu ~ "(Model 1)
Mean
Density
0.0
0.1
0.2
0.3
0.4
94
96
98
100
102
μ (Model 1)

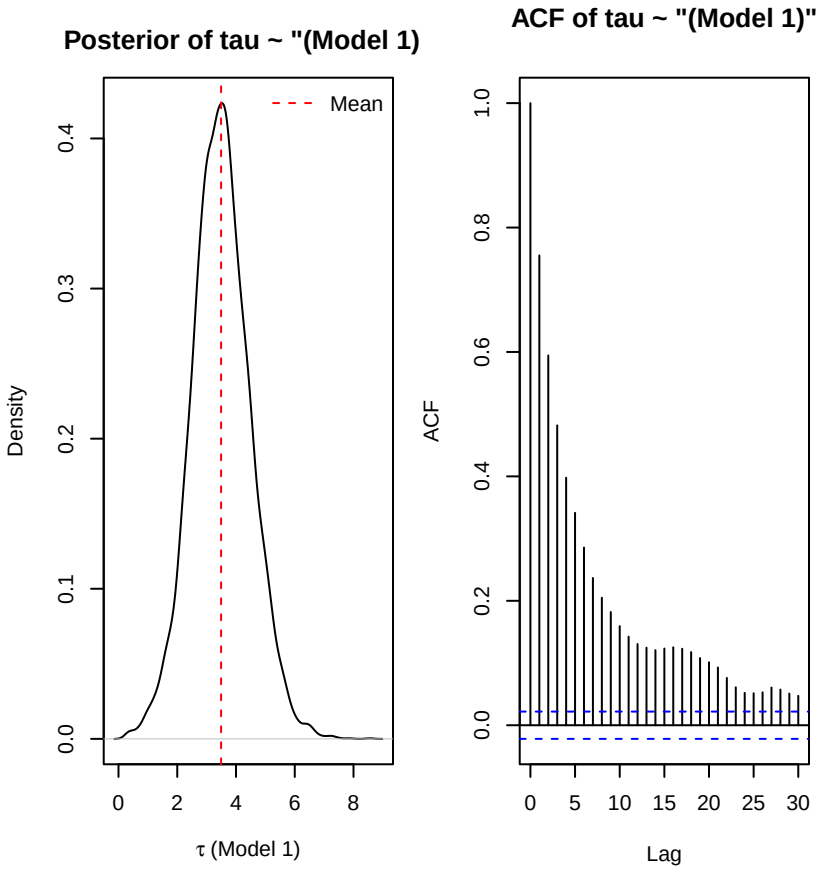

ACF of mu ~ "(Model 1)"
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

```r
effectiveSize(results_model1$mu)
```

```
##     var1
## 2631.608
```

```r
plot_diagnostics(sqrt(results_model1$tau_sq), expression(tau ~ "(Model 1)"))
```



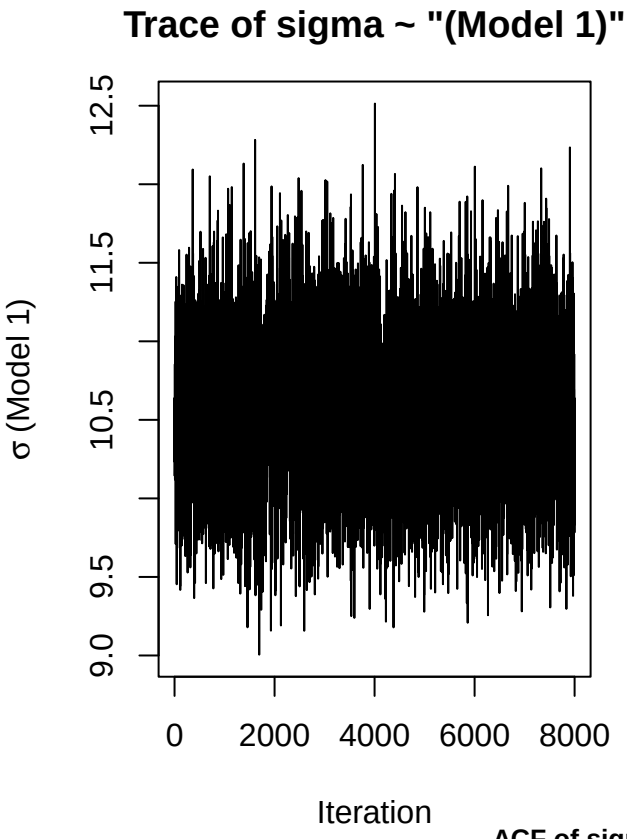


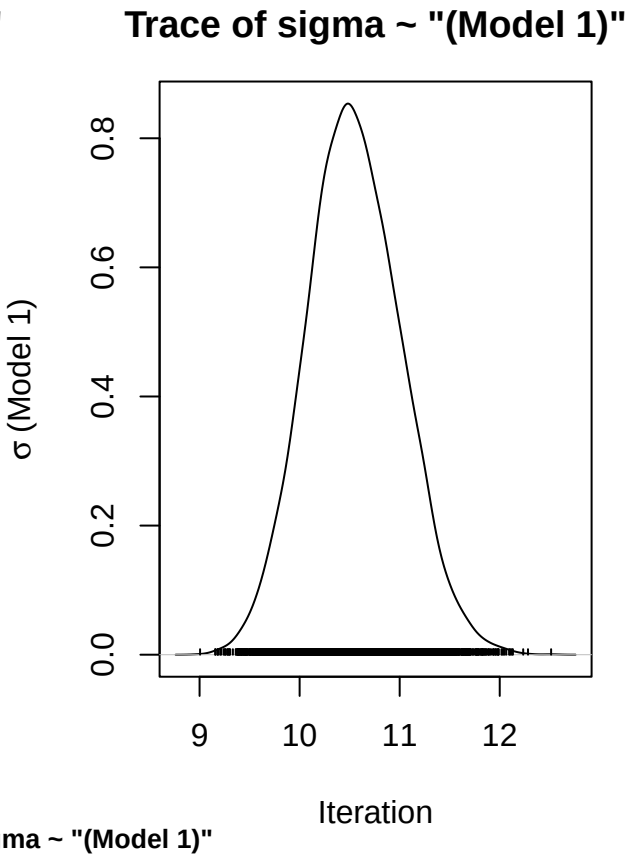

```
effectiveSize(results_model1$tau_sq)
```

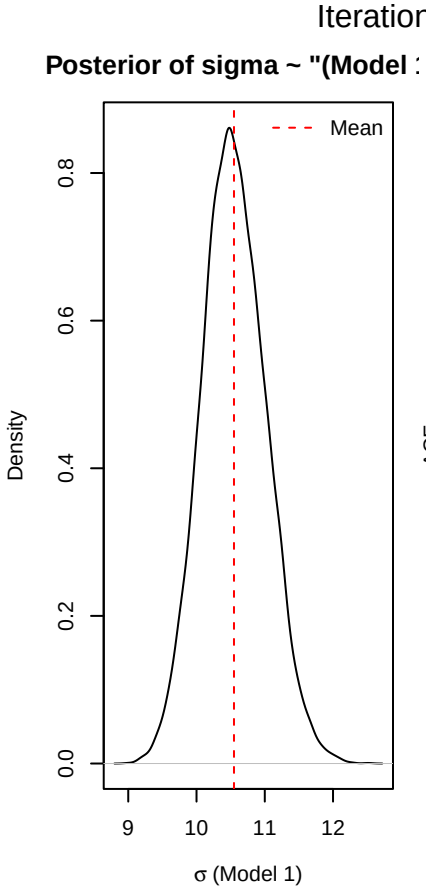
```
##     var1
## 1095.456
```

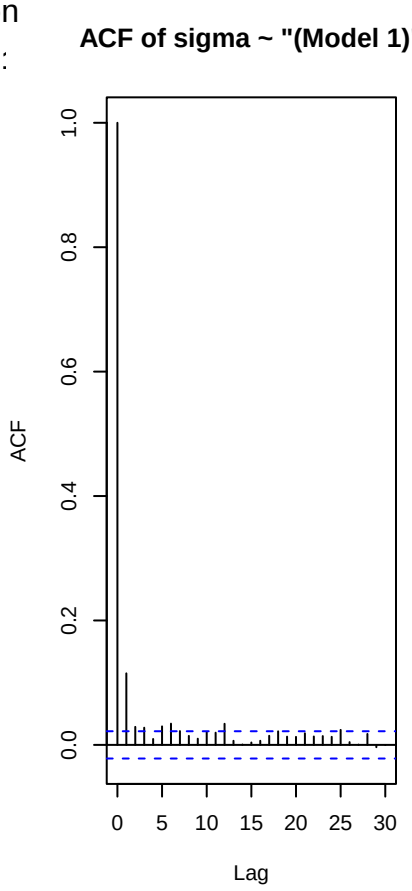
```
# Select a high-scoring and low-scoring team for detailed analysis
team_high_idx <- which.min(y_bar)
team_low_idx  <- which.max(y_bar)
```

```
plot_diagnostics(sqrt(results_model1$sigma_sq), expression(sigma ~ "(Model 1)"))
```


Trace of sigma ~ "(Model 1)"
σ (Model 1)
12.5
11.5
10.5
9.5
9.0
0
2000
4000
6000
8000
Iteration


Trace of sigma ~ "(Model 1)"
σ (Model 1)
0.8
0.6
0.4
0.2
0.0
9
10
11
12
Iteration

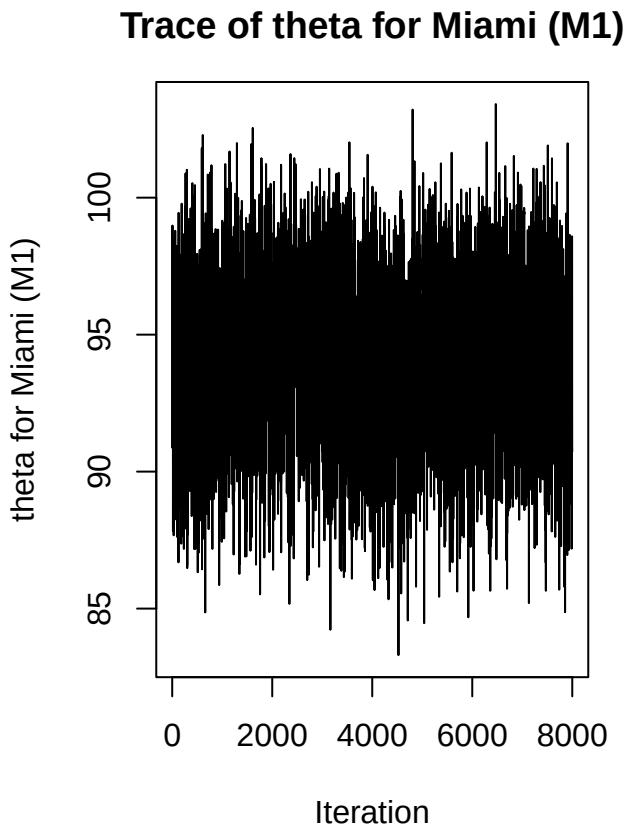

Posterior of sigma ~ "(Model
Mean
Density
0.8
0.6
0.4
0.2
0.0
9
10
11
12
σ (Model 1)

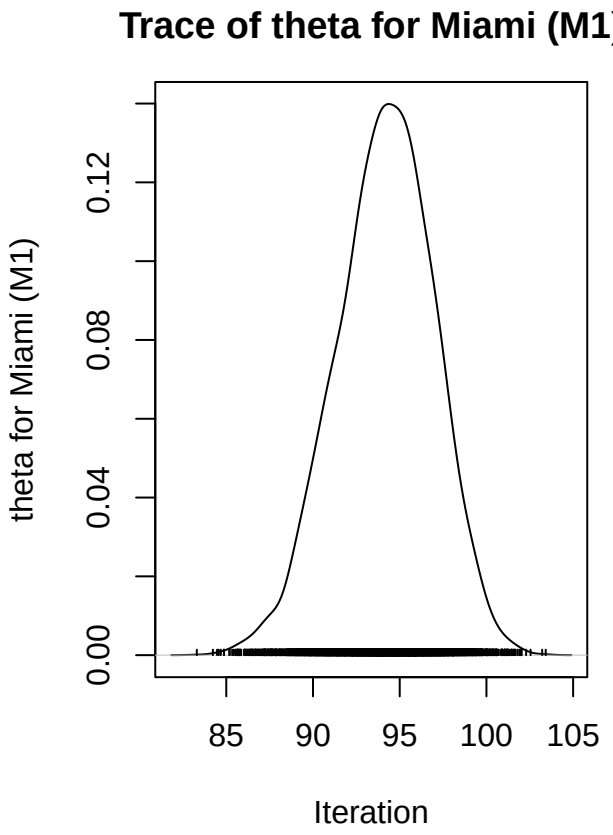

ACF of sigma ~ "(Model 1)"
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Lag

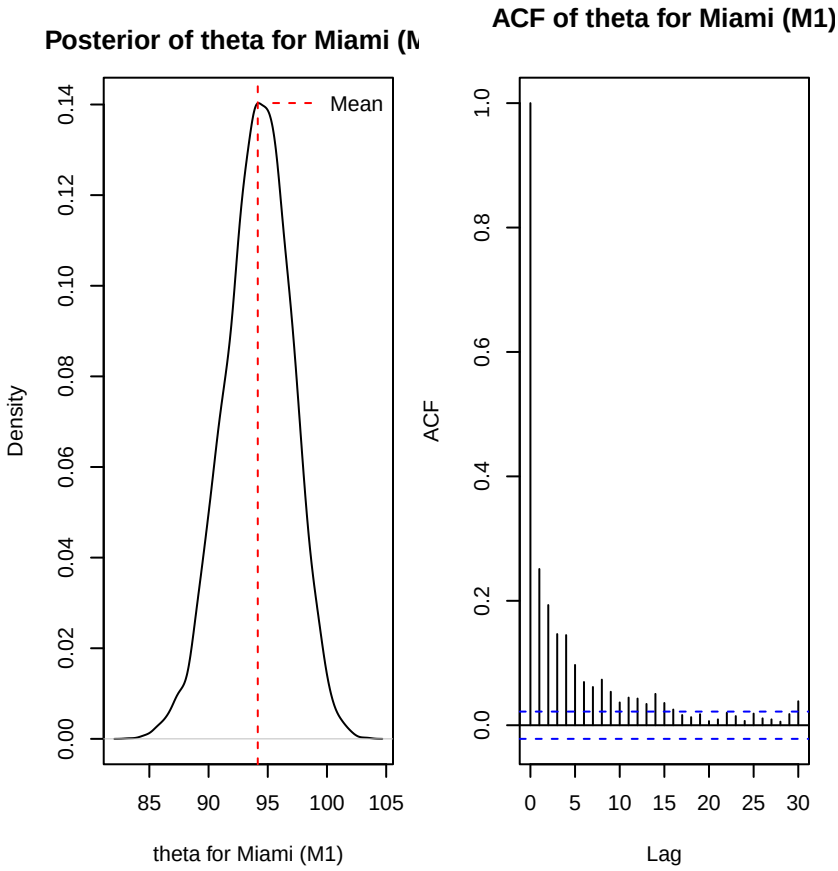
```r
effectiveSize(results_model1$sigma_sq)
```

```
##    var1
## 4393.3
```

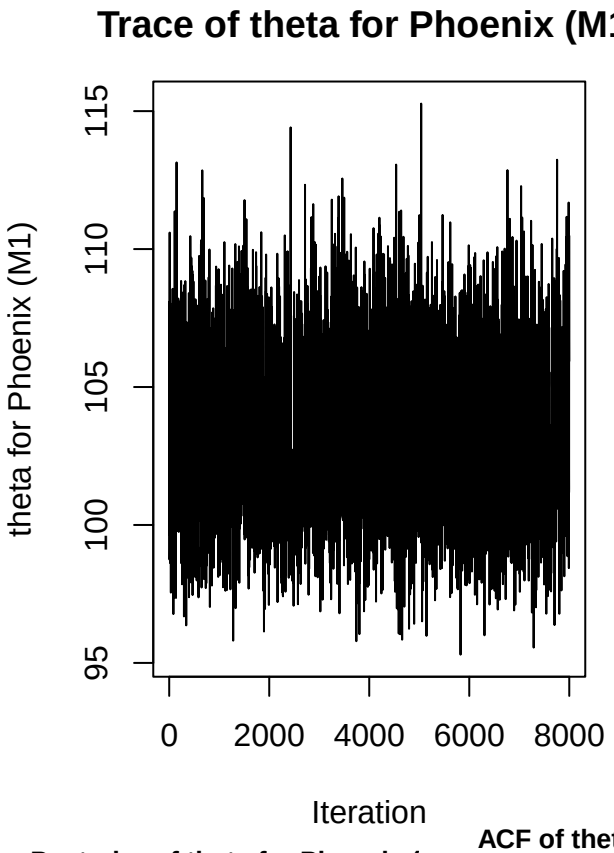
```r
plot_diagnostics(results_model1$theta[, team_high_idx], paste(expression(theta),"for", team_names[team_
```

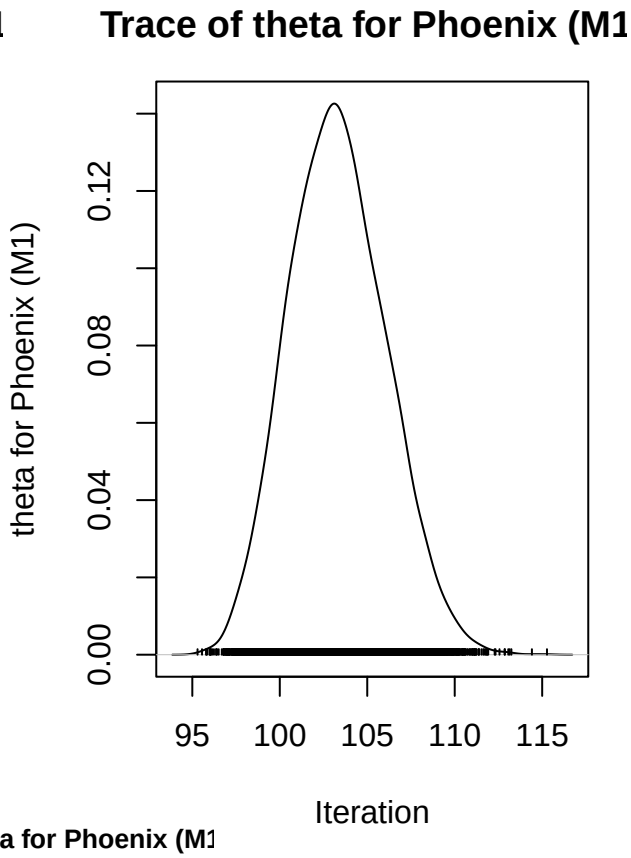


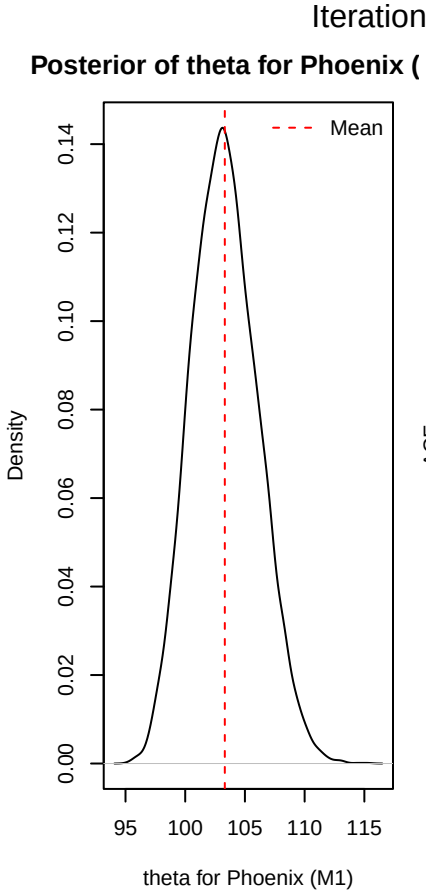


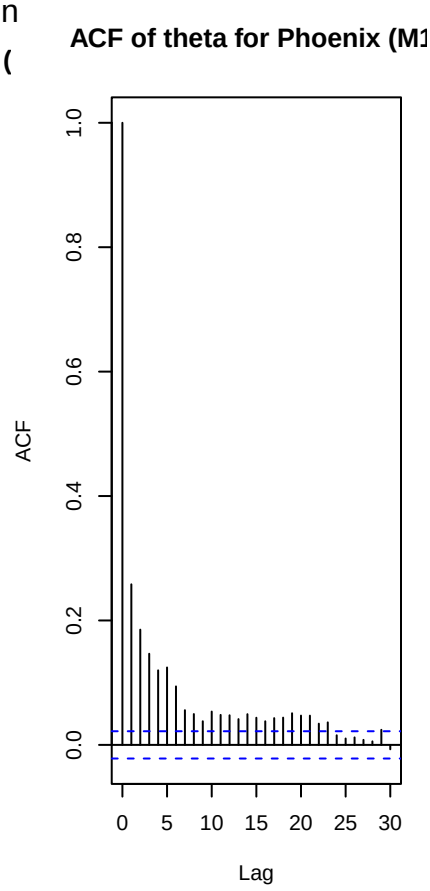



```
plot_diagnostics(results_model1$theta[, team_low_idx], paste(expression(theta),"for", team_names[team_l
```


Trace of theta for Phoenix (M1
theta for Phoenix (M1)
95
100
105
110
115
0
2000
4000
6000
8000
Iteration

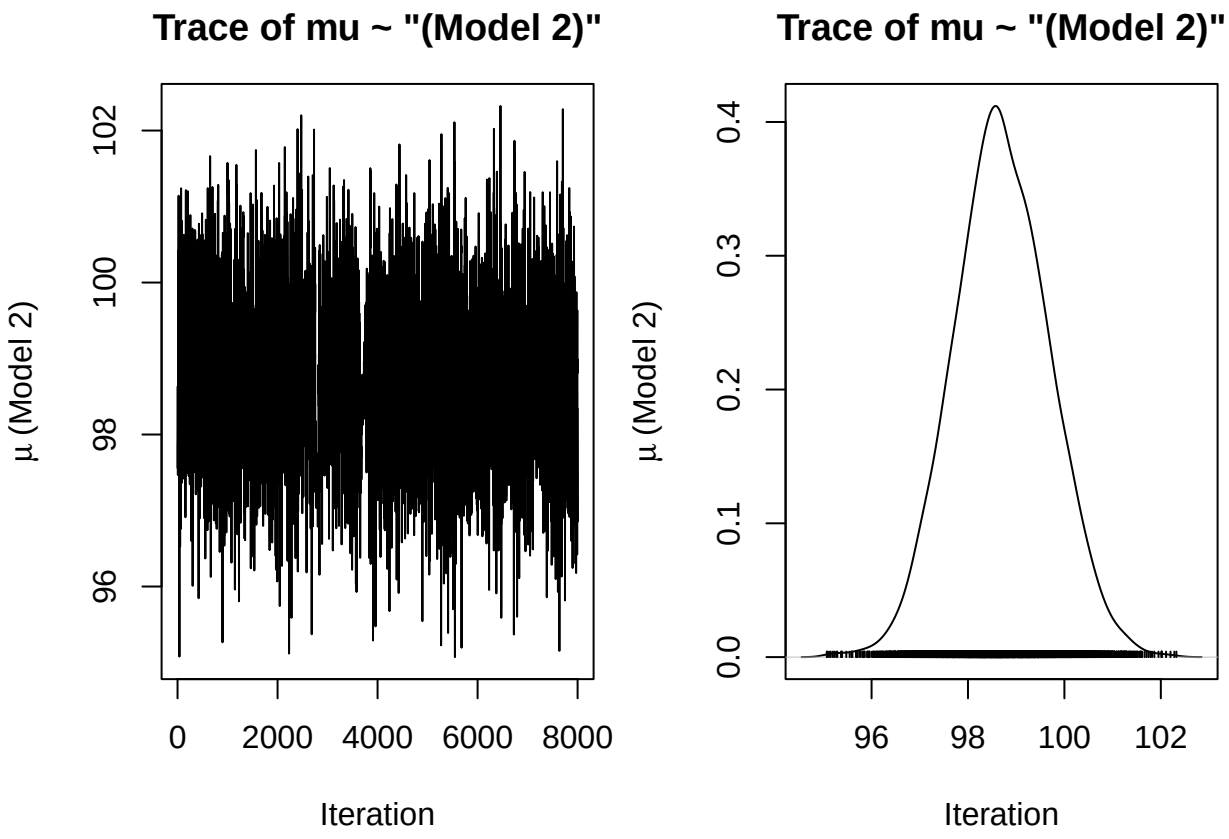

Trace of theta for Phoenix (M1
theta for Phoenix (M1)
0.00
0.04
0.08
0.12
95
100
105
110
115
Iteration

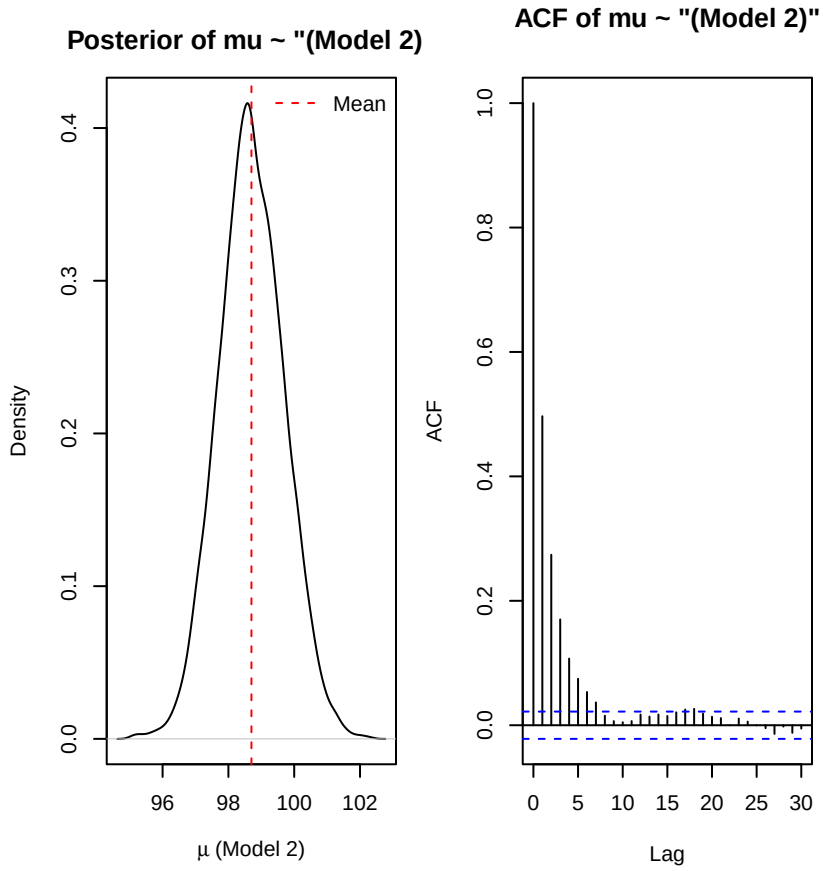

Posterior of theta for Phoenix (
Mean
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
95
100
105
110
115
theta for Phoenix (M1)


ACF of theta for Phoenix (M1
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

Model 2 Posterior Analysis

```
plot_diagnostics(results_model2$mu, expression(mu ~ "(Model 2)"))
```

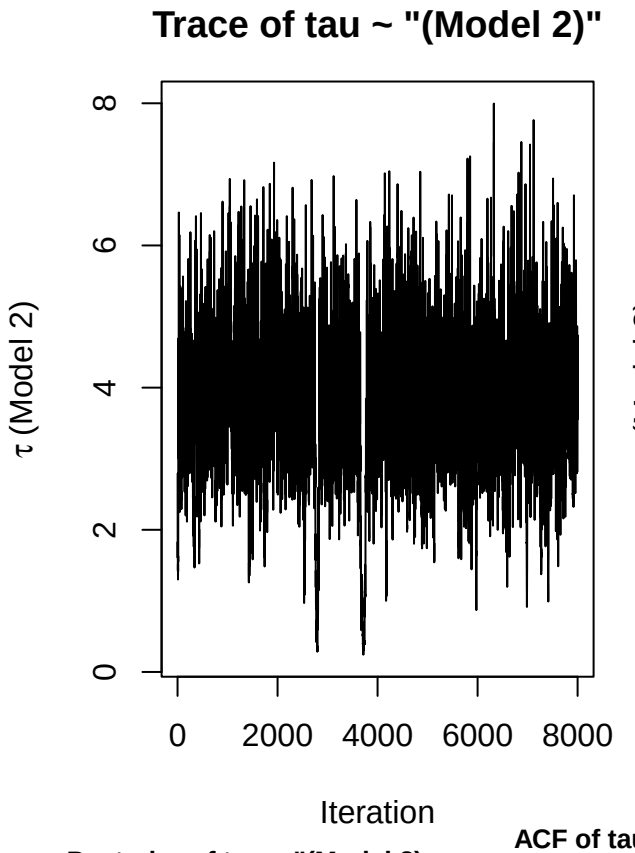


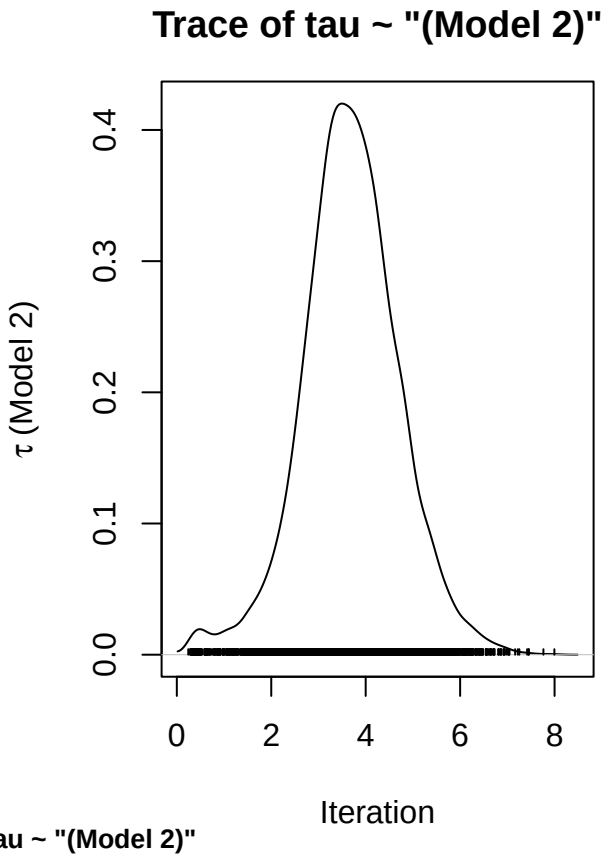


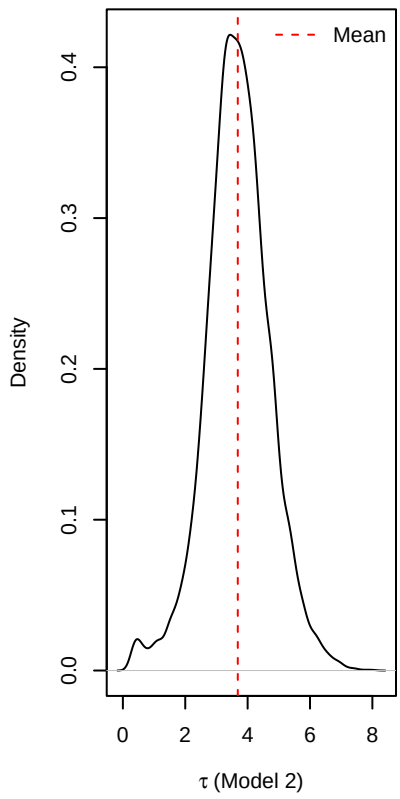

```
effectiveSize(results_model2$mu)
```

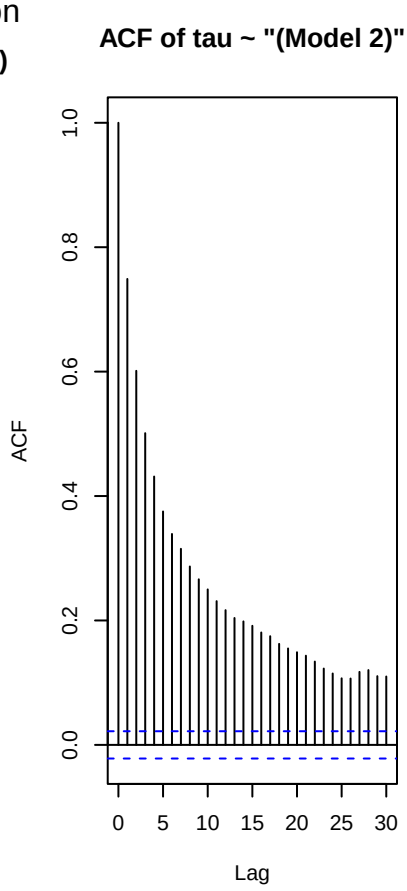
```
##     var1
## 2365.731
```

```
plot_diagnostics(sqrt(results_model2$tau_sq), expression(tau ~ "(Model 2)"))
```


Trace of tau ~ "(Model 2)"
τ (Model 2)
0
2
4
6
8
0
2000
4000
6000
8000
Iteration


Trace of tau ~ "(Model 2)"
τ (Model 2)
0.0
0.1
0.2
0.3
0.4
0
2
4
6
8
Iteration

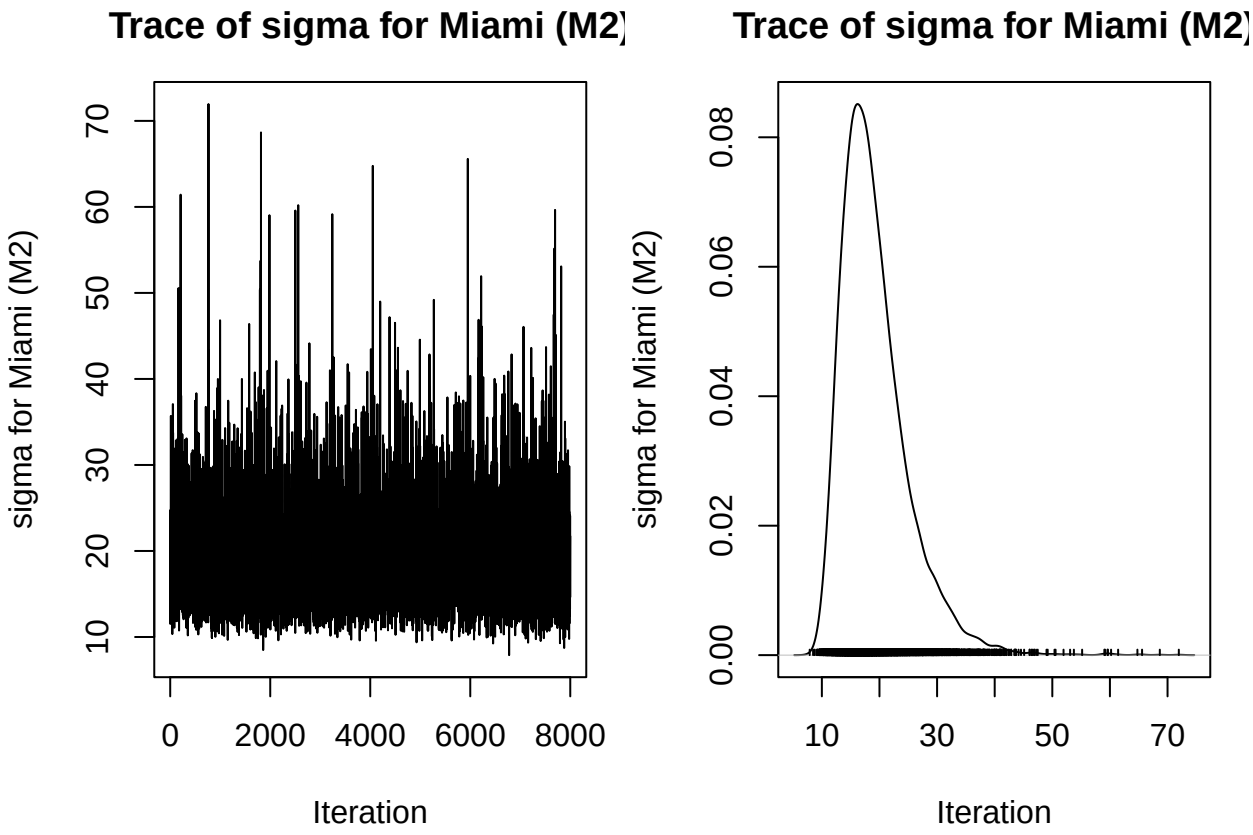

Posterior of tau ~ "(Model 2)
Mean
Density
0.0
0.1
0.2
0.3
0.4
0
2
4
6
8
τ (Model 2)

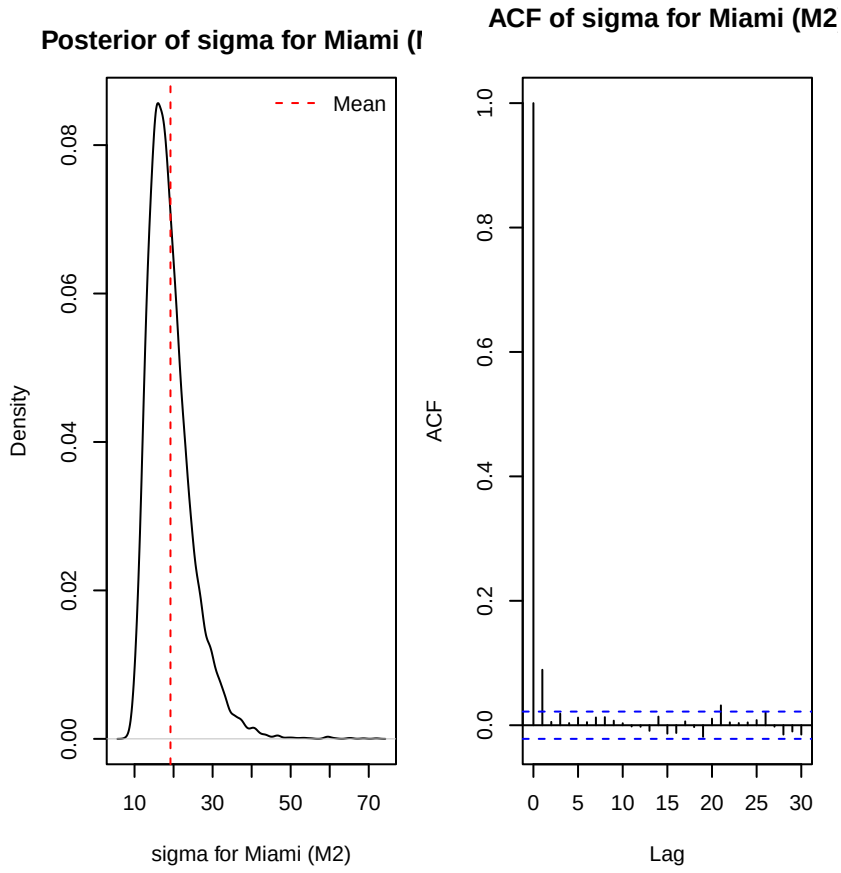

ACF of tau ~ "(Model 2)"
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

```
effectiveSize(results_model2$tau_sq)
```

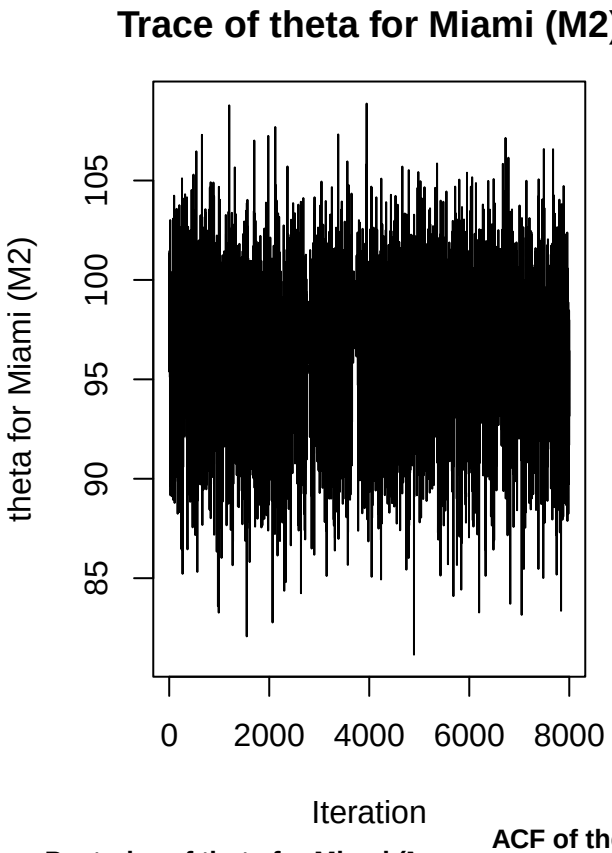
```
##     var1
## 975.5416
```

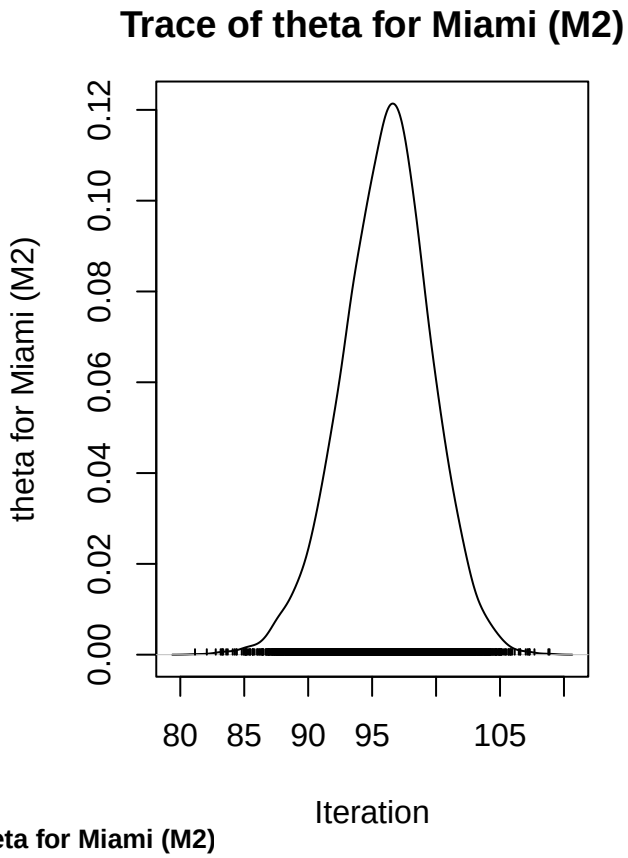
```
plot_diagnostics(sqrt(results_model2$sigma_sq[, team_high_idx]), paste(expression(sigma),"for", team_nar
```

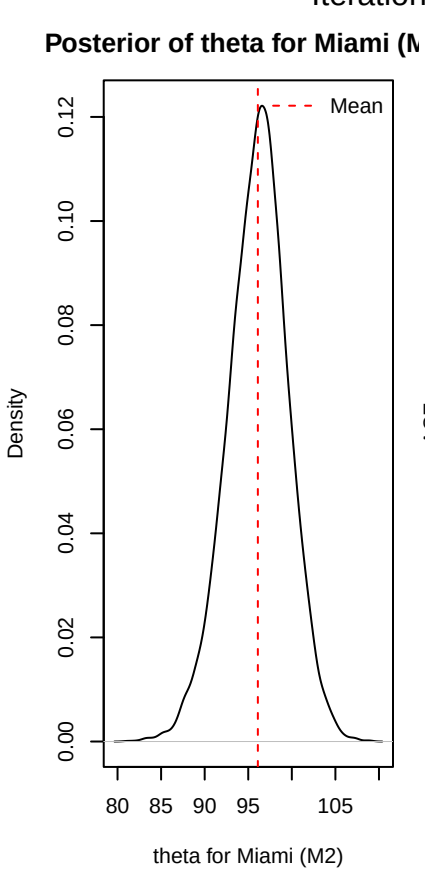


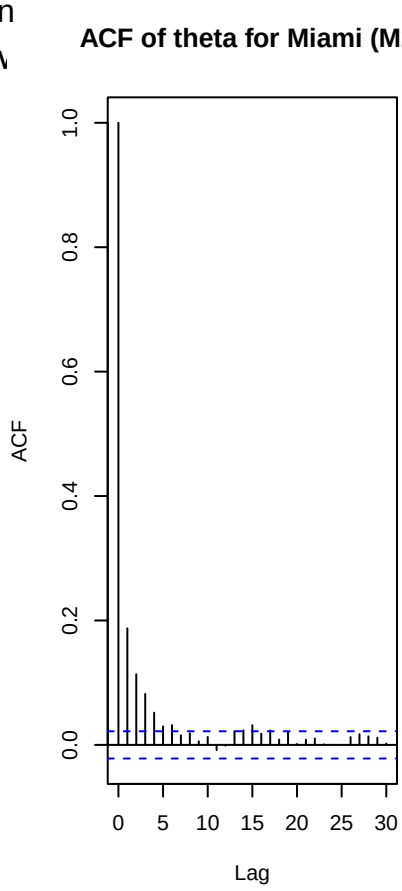



```
plot_diagnostics(results_model2$theta[, team_high_idx], paste(expression(theta),"for", team_names[team_
```

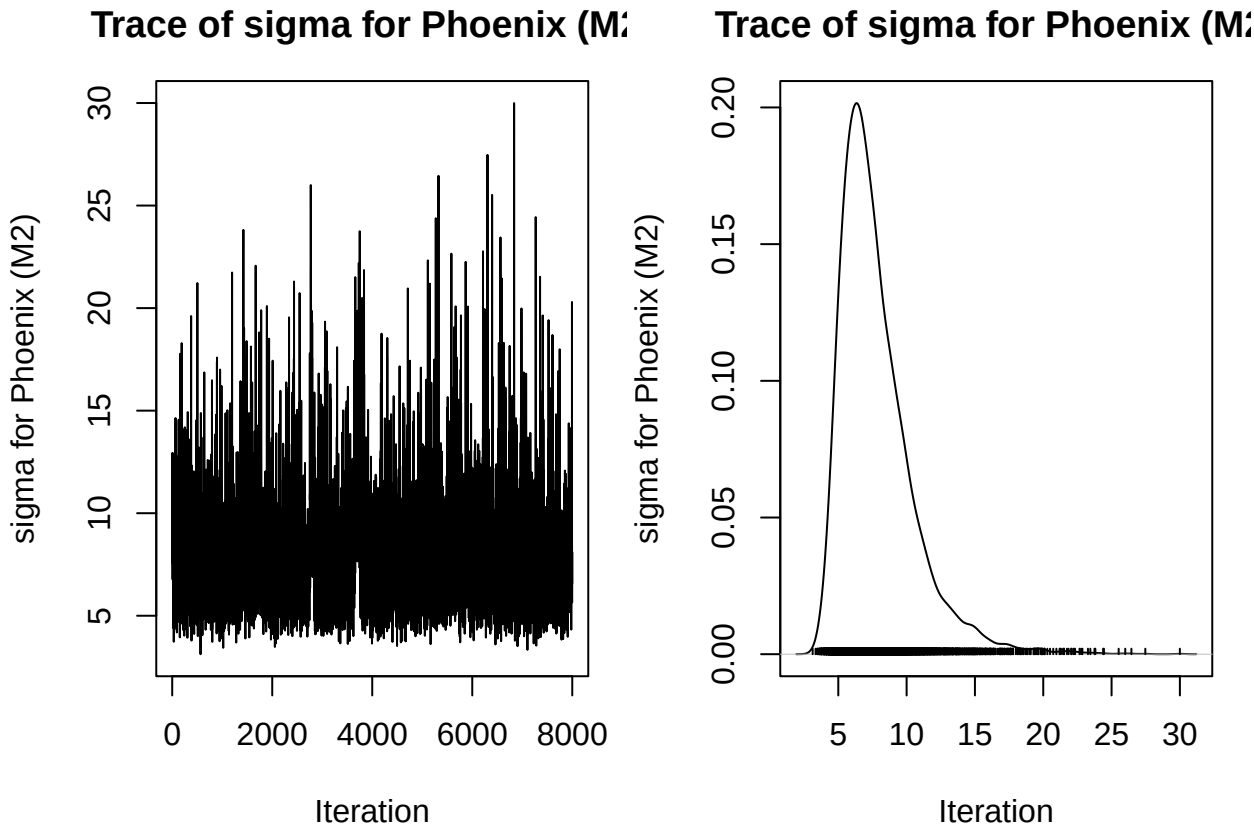

Trace of theta for Miami (M2)
theta for Miami (M2)
85
90
95
100
105
0
2000
4000
6000
8000
Iteration

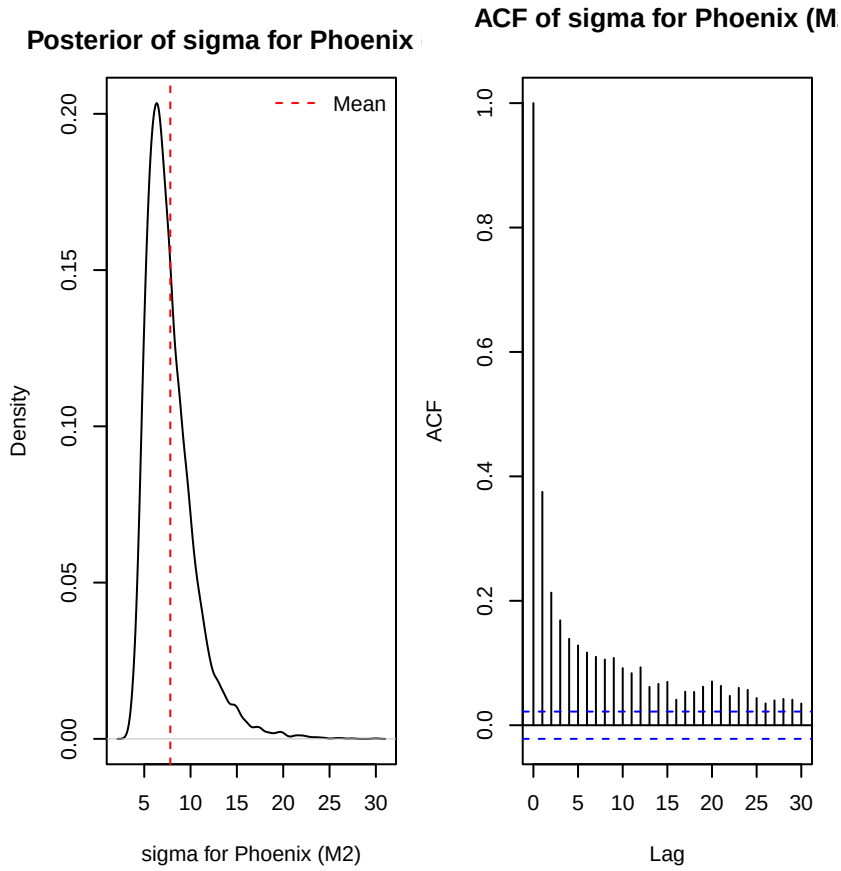

Trace of theta for Miami (M2)
theta for Miami (M2)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
80
85
90
95
105
Iteration


Posterior of theta for Miami (M
Mean
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
80
85
90
95
105
theta for Miami (M2)

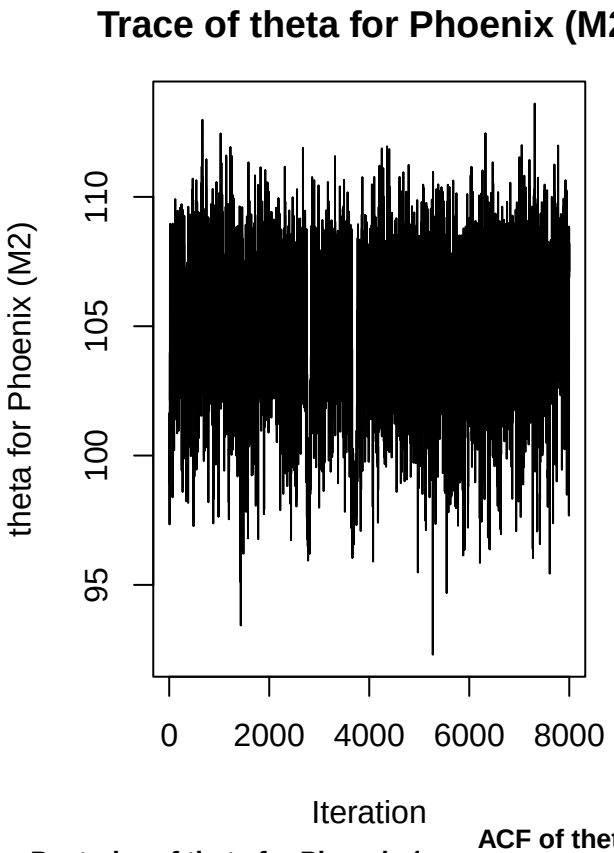

ACF of theta for Miami (M2)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

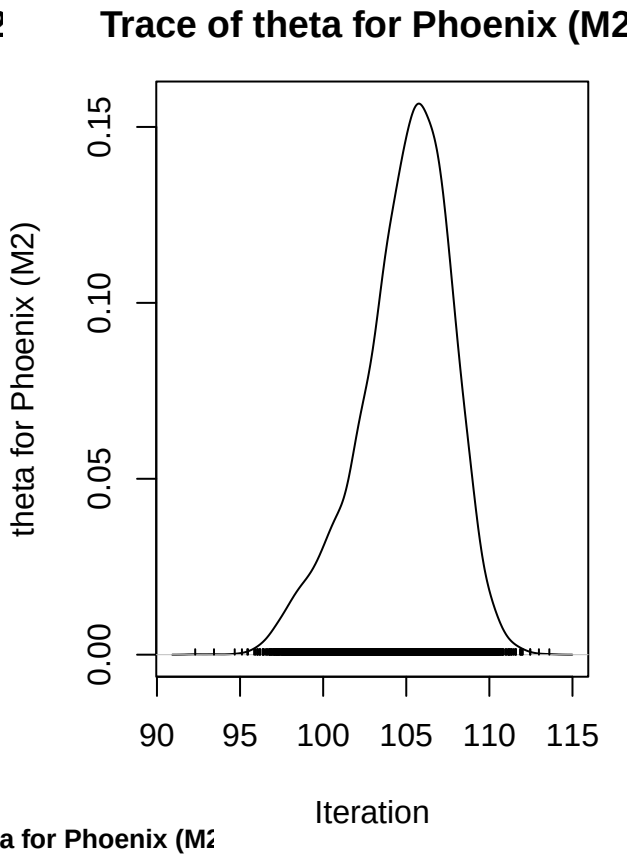
```
plot_diagnostics(sqrt(results_model2$sigma_sq[, team_low_idx]), paste(expression(sigma),"for", team_nam
```

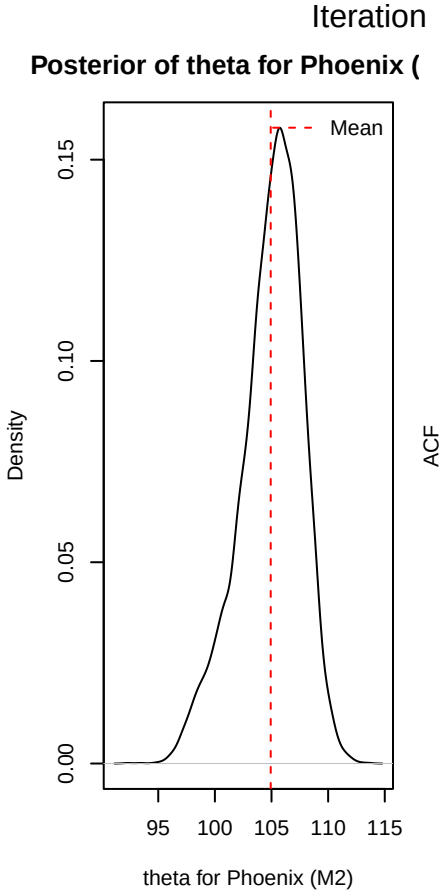


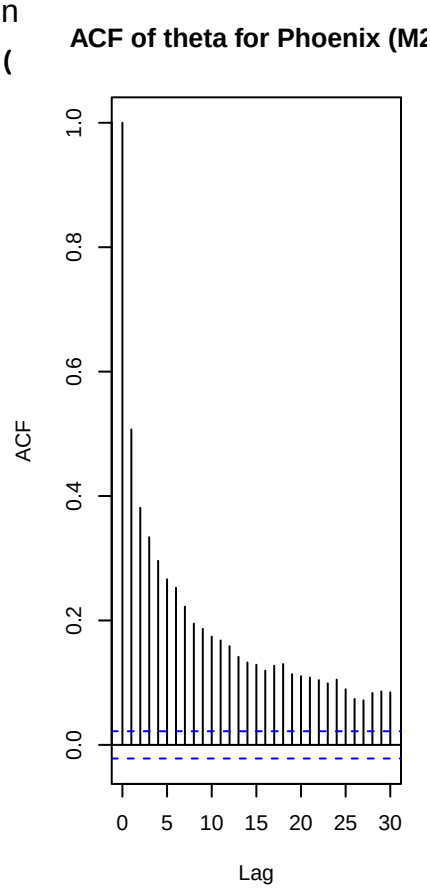



```
plot_diagnostics(results_model2$theta[, team_low_idx], paste(expression(theta),"for", team_names[team_l
```


Trace of theta for Phoenix (M2
theta for Phoenix (M2)
95
100
105
110
0
2000
4000
6000
8000
Iteration


Trace of theta for Phoenix (M2
theta for Phoenix (M2)
0.00
0.05
0.10
0.15
90
95
100
105
110
115
Iteration


Posterior of theta for Phoenix (
Mean
Density
0.00
0.05
0.10
0.15
95
100
105
110
115
theta for Phoenix (M2)


ACF of theta for Phoenix (M2
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

5. Comment on how the posterior distributions differ from each other. Use each model to make predictions on the NBA games from the rest of the season (following code from Lecture 7), and comment on which model (or the model with known variances from Lecture 7) performs best.

The trace plots for all major parameters in both Model 1 and Model 2 show good mixing. The chains appear stationary. The Gibbs samplers have converged to the stationary posterior distribution and the draws are reliable.

- μ: The posterior distributions for μ between Model 1 and Model 2 appear nearly identical. This is expected, as the overall league average is a high-level parameter that is informed by all teams.
- τ: The posterior for τ is also very similar between the two models.
- σ: This is where the two models differ.
  - Model 1 produces a single posterior distribution for a shared σ. The mean of this posterior is around 10.5, representing a single, pooled estimate of game-to-game variability for all teams.
  - Model 2 produces separate posterior distributions for each team's σ_j. Model 2 estimates a much higher σ_j for Phoenix and a lower one for Miami, capturing the individual team characteristics.
- θ: The posterior distributions for individual thetas are subtly different as it is dependent on σ for both the models. Because Model 2 allows for different σ_j, it can adjust the amount of "shrinkage" toward the overall mean (μ) for each team.

```
predict_season <- function(post_draws, n_remaining) {
    n_draws <- nrow(post_draws$mu)
    J <- ncol(post_draws$theta)

    # Store predictions for each posterior draw
    y_pred_draws <- matrix(NA, nrow = n_draws, ncol = J)

    for (s in 1:n_draws) {
        theta_s <- post_draws$theta[s, ]

        # sigma_sq can be a single value (Model 1) or a vector (Model 2)
        sigma_sq_s <- post_draws$sigma_sq[s, ]
        if (length(sigma_sq_s) == 1) {
            sigma_sq_s <- rep(sigma_sq_s, J)
        }

        pred_var <- sigma_sq_s / n_remaining
        y_pred_draws[s, ] <- rnorm(J, mean = theta_s, sd = sqrt(pred_var))
    }

    # Return the mean prediction across all posterior draws
    return(colMeans(y_pred_draws))
}
```

```
# Generate predictions for all models
pred_model1 <- predict_season(results_model1, n_rem)
pred_model2 <- predict_season(results_model2, n_rem)
```

```
# Calculate Root Mean Squared Error (RMSE) for each
rmse_model1 <- sqrt(mean((pred_model1 - y_rem_obs)^2))
rmse_model2 <- sqrt(mean((pred_model2 - y_rem_obs)^2))
cat("RMSE for Model 1 (Identical Unknown Var):", round(rmse_model1, 4), "\n")
```

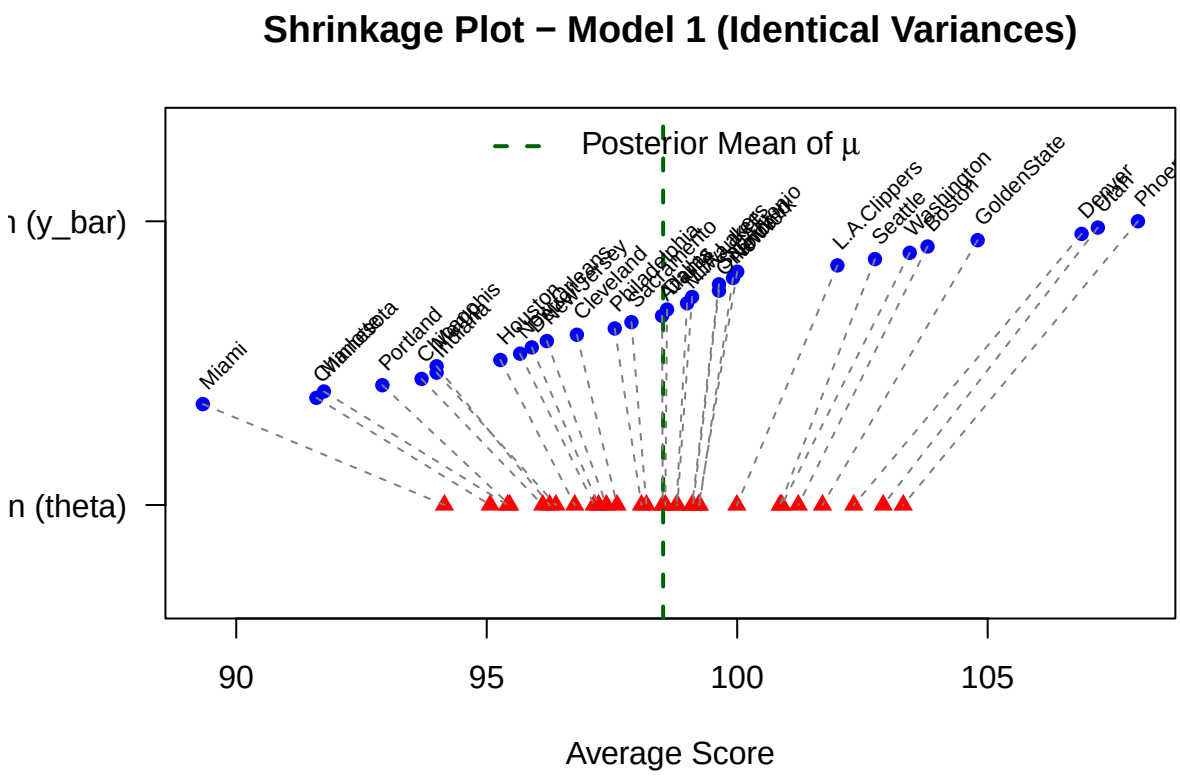
```
## RMSE for Model 1 (Identical Unknown Var): 3.4559
```

```
cat("RMSE for Model 2 (Independent Unknown Var):", round(rmse_model2, 4), "\n")
```

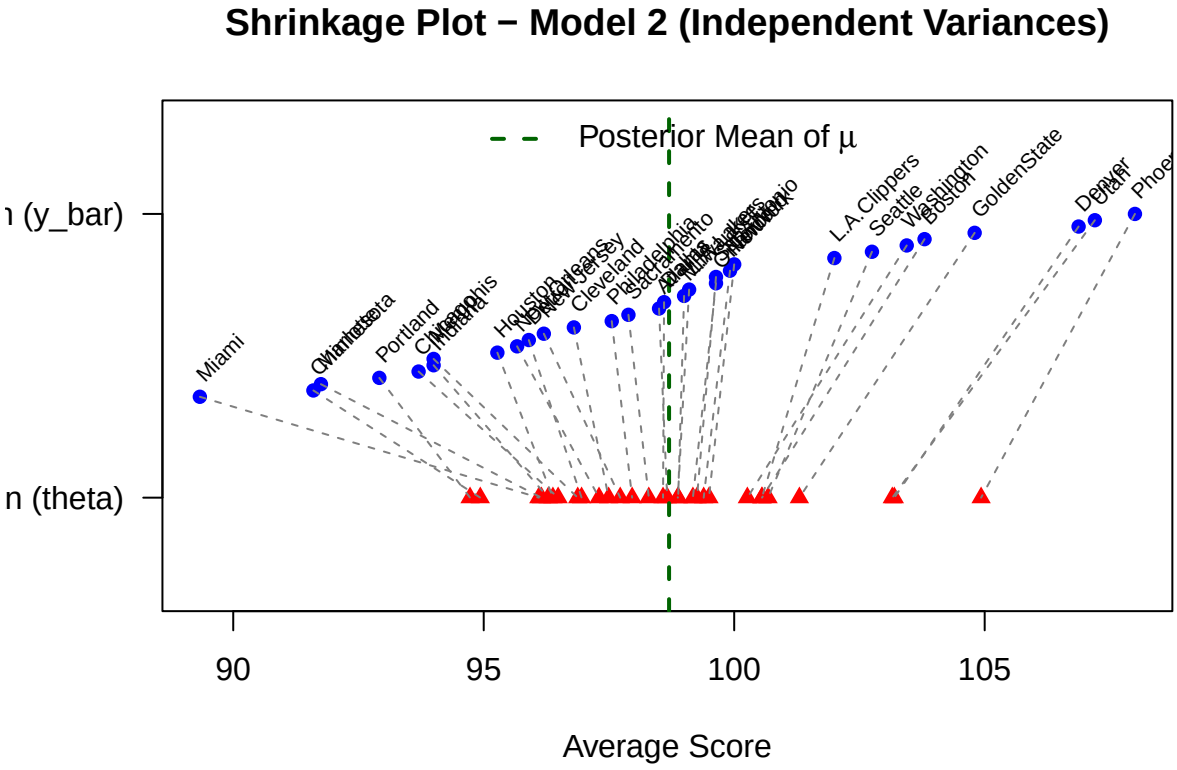
```
## RMSE for Model 2 (Independent Unknown Var): 3.3135
```

```
make_shrinkage_plot <- function(y_bar, post_theta, post_mu, team_names, model_title) {

  # Calculate posterior means for theta and mu
  theta_means <- colMeans(post_theta)
  mu_mean <- mean(post_mu)
  J <- length(y_bar)

  # Determine the x-axis limits to fit all data
  x_lims <- range(c(y_bar, theta_means, mu_mean))

  # Sort teams by sample mean to make the plot cleaner
  o <- order(y_bar)

  # Set up the plot
  plot(y_bar[o], (1:J) / (J/2) + 2, # Stagger labels for y_bar
       main = paste("Shrinkage Plot -", model_title),
       ylim = c(0, 5), # Y-range for points and labels
       xlim = x_lims,
       ylab = "",
       xlab = "Average Score",
       yaxt = "n",  # Turn off Y-axis
       pch = 16,    # Solid circle for y_bar
       col = "blue")

  # Add points for the posterior means
  points(theta_means[o], rep(1, J),
         pch = 17,    # Solid triangle for theta_mean
         col = "red")

  # Add Y-axis labels manually
  axis(side = 2, at = c(1, 4), labels = c("Posterior Mean (theta)", "Sample Mean (y_bar)"), las = 1)

  # Add a vertical line for the grand posterior mean (mu)
  abline(v = mu_mean, col = "darkgreen", lty = 2, lwd = 2)
  legend("top", expression(paste("Posterior Mean of ", mu)),
         col="darkgreen", lty=2, lwd=2, bty="n", horiz=TRUE)

  # Add segments connecting y_bar to theta_mean
  segments(y_bar[o], (1:J) / (J/2) + 2,
           theta_means[o], rep(1, J),
           lty = 2, col = "gray50")

  # Add team labels above the y_bar points
  text(y_bar[o], (1:J) / (J/2) + 2.2,
       team_names[o],
       cex = 0.7, srt = 45, adj = 0)
}
```

```
make_shrinkage_plot(y_bar,
                    results_model1$theta,
                    results_model1$mu,
                    team_names,
                    "Model 1 (Identical Variances)")
```




```
make_shrinkage_plot(y_bar,
                    results_model2$theta,
                    results_model2$mu,
                    team_names,
                    "Model 2 (Independent Variances)")
```



Model 2 performs marginally better than Model 1 based on its lower RMSE. However, it is better than Model 1 as it is more flexible and realistic. By allowing each team to have its own observation variance, it can make more tailored, and ultimately more accurate, predictions about future performance.

The Stroop Effect

```
library('rstan')
```

```
## Loading required package: StanHeaders
```

```
##
## rstan version 2.32.7 (Stan version 2.32.2)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

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

```r
library('ggplot2')
library('tidyverse')
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.4     v tibble    3.3.0
## v purrr     1.1.0     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::extract() masks rstan::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library('data.table')
```

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
library('bayesplot')
```

```
## This is bayesplot version 1.14.0
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
```

```r
library('posterior')
```

```
## This is posterior version 1.6.1
##
## Attaching package: 'posterior'
##
## The following object is masked from 'package:bayesplot':
##
##     rhat
##
## The following objects are masked from 'package:rstan':
##
##     ess_bulk, ess_tail
##
## The following objects are masked from 'package:stats':
```

```
##
##     mad, sd, var
##
## The following objects are masked from 'package:base':
##
##     %in%, match
```

```
library('janitor')
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
theme_set(theme_minimal(base_size = 12))
```

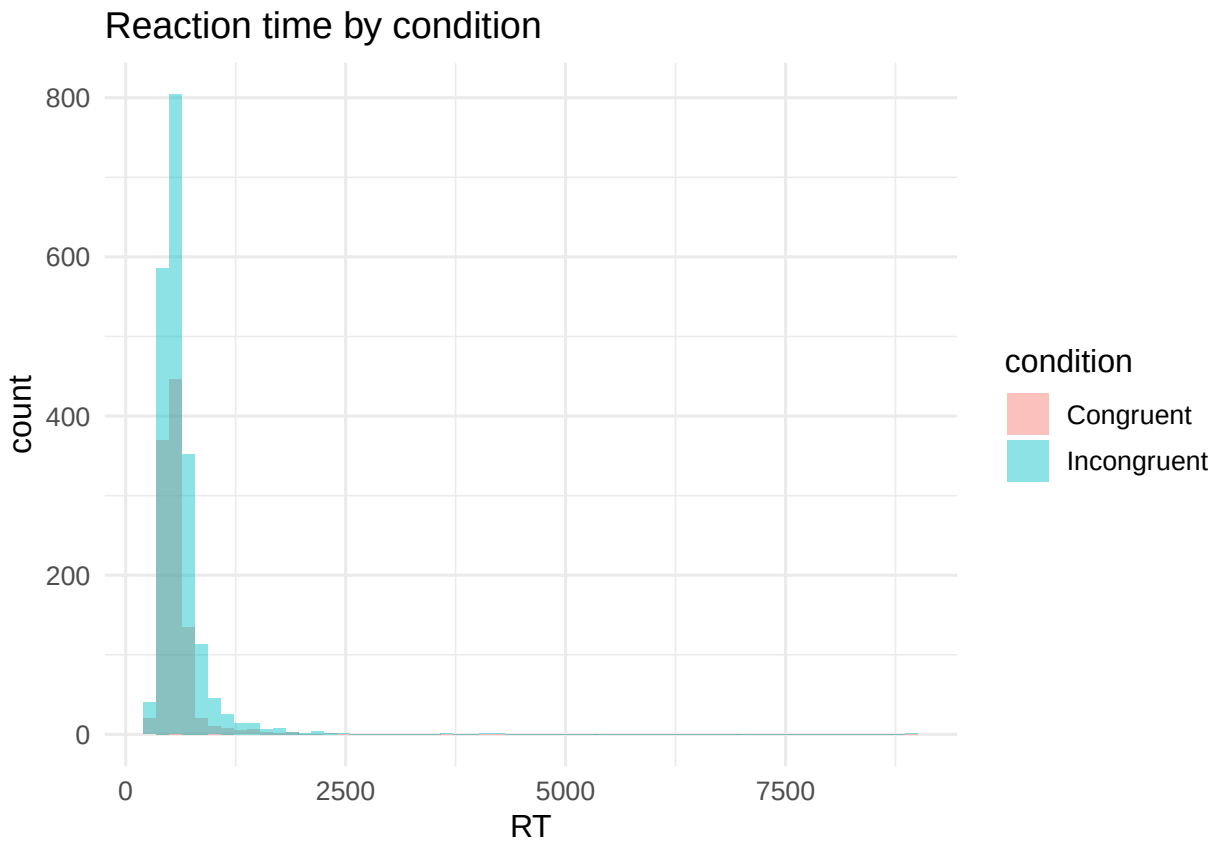
```
stroop_data = read.csv(file="stroop_dataset.csv", header=TRUE)
head(stroop_data)
```

```
##   X subj trial   condition  word color   RT
## 1 1    1     0   Congruent   red   red 1484
## 2 2    1     1 Incongruent green  blue 1316
## 3 3    1     2 Incongruent  blue green  628
## 4 4    1     3   Congruent green green  511
## 5 5    1     4   Congruent  blue  blue  509
## 6 6    1     5 Incongruent   red  blue  903
```

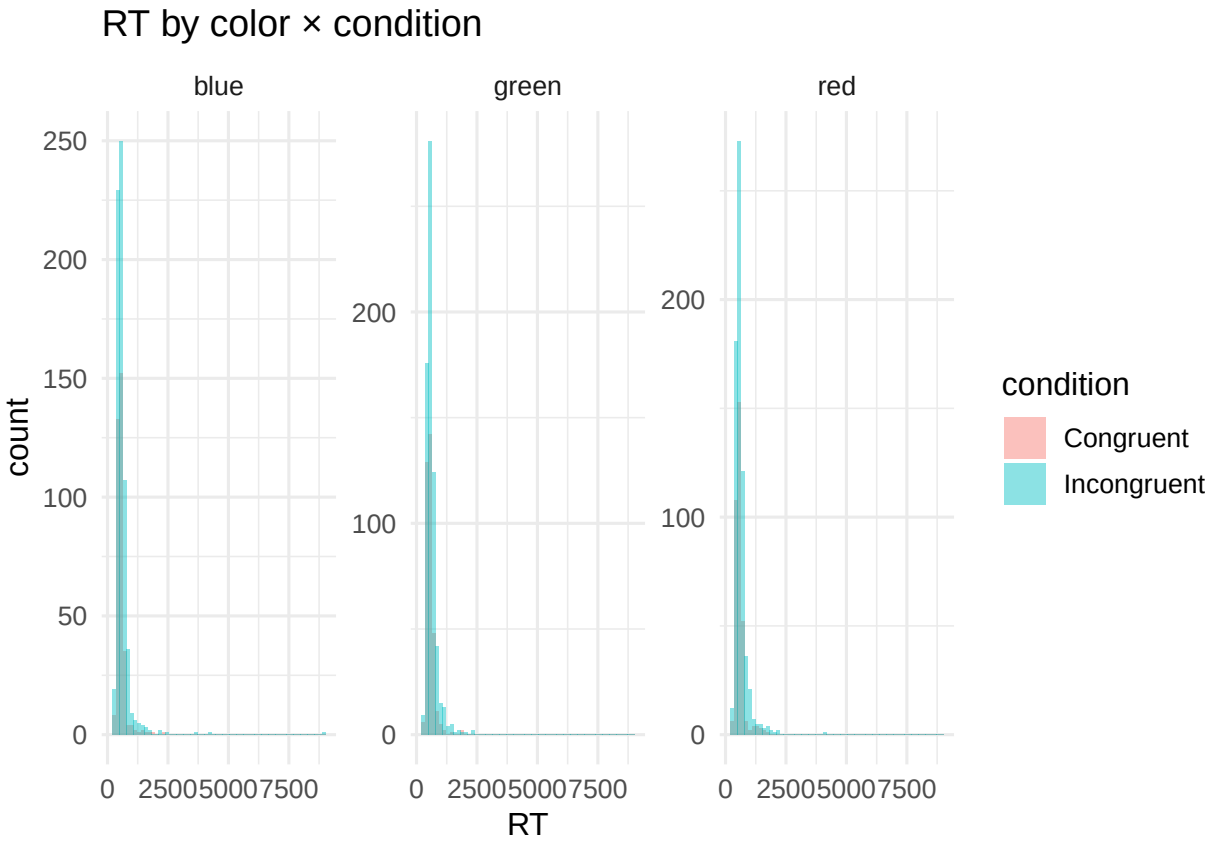
```
summary(stroop_data$RT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   276.0   469.0   550.0   600.0   648.8  8936.0
```

```
ggplot(stroop_data, aes(RT, fill = condition)) +
  geom_histogram(position = "identity", alpha = 0.45, bins = 60) +
  labs(title = "Reaction time by condition", x = "RT")
```




```
ggplot(stroop_data, aes(RT, fill = condition)) +
  geom_histogram(position = "identity", alpha = 0.45, bins = 60) +
  facet_wrap(~ color, scales = "free_y") +
  labs(title = "RT by color × condition", x = "RT")
```



RT's are positive and right-skewed. So we will model them as follows

$$y_i = \log(RT_i) \sim \mathcal{N}(\mu_i, \sigma)$$

We need to pool information across subjects and colors. For trial i, subject s_i, color c_i, and incongruency $I_i \in \{0, 1\}$.

$$\mu_i = \alpha + a_{s_i} + \gamma_{c_i} + (\beta + b_{s_i} + \delta_{c_i})I_i$$

- α: grand mean (log RT)
- a_s: subject intercept deviation
- γ_c: color intercept deviation
- β: grand Stroop effect
- b_s: subject deviation in Stroop effect
- δ_c: color deviation in Stroop effect
- σ: observation standard deviation

Prior (Weakly Informative)

- $\alpha \sim \mathcal{N}(\log(550), 0.5)$: baseline value of 550 with a small variance.
- $\beta \sim \mathcal{N}(0, 0.5)$: Stroop effect multiplicative factor.
- $\sigma \sim$ half-Normal$(0, 0.5)$
- $\tau \sim$ half-Normal$(0, 0.5)$

- correlations $\sim LKJ(2)$: correlation priors allow intercept–slope correlations (e.g., slower subjects might show larger/smaller Stroop effects).

```
stan_code <- "
data {
  int<lower=1> N;
  vector[N] log_rt;
  int<lower=0,upper=1> incong[N];
  int<lower=1> S;
  int<lower=1> C;
  int<lower=1,upper=S> subj[N];
  int<lower=1,upper=C> color[N];
}
parameters {
  real alpha;     // grand intercept
  real beta;      // grand Stroop effect

  // Non-centered random effects: subject and color (intercept, slope)
  matrix[2, S] z_subj;
  matrix[2, C] z_col;

  vector<lower=0>[2] tau_subj;     // SDs for subj intercept & slope
  vector<lower=0>[2] tau_col;      // SDs for color intercept & slope
  cholesky_factor_corr[2] L_subj;  // corr (int,slope) for subjects
  cholesky_factor_corr[2] L_col;   // corr (int,slope) for colors

  real<lower=0> sigma;             // obs SD on log scale
}
transformed parameters {
  // Random effects in centered scale: eta = L * diag(tau) * z
  matrix[2, S] eta_subj = diag_pre_multiply(tau_subj, L_subj) * z_subj;
  matrix[2, C] eta_col  = diag_pre_multiply(tau_col,  L_col)  * z_col;

  row_vector[S] a_subj = (eta_subj[1, ]);
  row_vector[S] b_subj = (eta_subj[2, ]);
  row_vector[C] g_col  = (eta_col[1,  ]);
  row_vector[C] d_col  = (eta_col[2,  ]);
}
model {
  // Priors
  alpha ~ normal(log(550), 0.5);
  beta  ~ normal(0, 0.5);

  to_vector(z_subj) ~ normal(0,1);
  to_vector(z_col)  ~ normal(0,1);

  tau_subj ~ normal(0, 0.5);
  tau_col  ~ normal(0, 0.5);

  L_subj ~ lkj_corr_cholesky(2);
  L_col  ~ lkj_corr_cholesky(2);

  sigma ~ normal(0, 0.5);
```

```
  // Likelihood
  for (i in 1:N) {
    real mu = alpha + a_subj[subj[i]] + g_col[color[i]]
              + (beta + b_subj[subj[i]] + d_col[color[i]]) * incong[i];
    log_rt[i] ~ normal(mu, sigma);
  }
}
generated quantities {
  vector[N] log_rt_rep;
  vector[C] color_stroop;           // per-color Stroop effect (log scale)
  corr_matrix[2] R_subj;
  corr_matrix[2] R_col;

  R_subj = multiply_lower_tri_self_transpose(L_subj);
  R_col  = multiply_lower_tri_self_transpose(L_col);

  for (i in 1:N) {
    real mu = alpha + a_subj[subj[i]] + g_col[color[i]]
              + (beta + b_subj[subj[i]] + d_col[color[i]]) * incong[i];
    log_rt_rep[i] = normal_rng(mu, sigma);
  }
  for (c in 1:C) color_stroop[c] = beta + d_col[c];
}
"
writeLines(stan_code, "stroop_hier_rstan.stan")
```

```
# Compiling a model can take time. If you plan to use it across R sessions,
# you can save it as below
#saveRDS(stan_code, "stroop_hier_rstan.rds")
```

```
# Standardize & select
dt <- stroop_data |>
  mutate(condition = tolower(condition),
         color     = tolower(color),
         word      = tolower(word)) |>
  select(subj, trial, condition, word, color, RT) |>
  filter(is.finite(RT), RT > 0)

head(dt)
```

```
##   subj trial   condition  word color   RT
## 1    1     0   congruent   red   red 1484
## 2    1     1 incongruent green  blue 1316
## 3    1     2 incongruent  blue green  628
## 4    1     3   congruent green green  511
## 5    1     4   congruent  blue  blue  509
## 6    1     5 incongruent   red  blue  903
```

```
dt2 <- dt |>
  mutate(incong = if_else(condition == "incongruent", 1L, 0L),
         subj_i = as.integer(as.factor(subj)),
         color_i = as.integer(as.factor(color)),
         log_rt = log(RT))
head(dt2)
```

```
##   subj trial   condition  word color   RT incong subj_i color_i   log_rt
## 1    1     0   congruent   red   red 1484      0      1       3 7.302496
## 2    1     1 incongruent green  blue 1316      1      1       1 7.182352
## 3    1     2 incongruent  blue green  628      1      1       2 6.442540
## 4    1     3   congruent green green  511      0      1       2 6.236370
## 5    1     4   congruent  blue  blue  509      0      1       1 6.232448
## 6    1     5 incongruent   red  blue  903      1      1       1 6.805723
```

```
stan_data <- list(
  N      = nrow(dt2),
  log_rt = dt2$log_rt,
  incong = dt2$incong,
  S      = n_distinct(dt2$subj_i),
  C      = n_distinct(dt2$color_i),
  subj   = dt2$subj_i,
  color  = dt2$color_i
)
```

```
mod <- rstan::stan_model("stroop_hier_rstan.stan")
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 17.0.0 (clang-1700.3.19.1)'
## using SDK: 'MacOSX26.0.sdk'
## clang -arch arm64 -std=gnu2x -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Li
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
##   679 | #include <cmath>
##       |          ^~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
```

```
## Warning in rstan::stan_model("stroop_hier_rstan.stan"):
## data {
##   int<lower=1> N;
##   vector[N] log_rt;
##   int<lower=0,upper=1> incong[N];
##   int<lower=1> S;
##   int<lower=1> C;
##   int<lower=1,upper=S> subj[N];
##   int<lower=1,upper=C> color[N];
## }
## parameters {
##   real alpha;     // grand intercept
##   real beta;      // grand Stroop effect
##
##   // Non-centered random effects: subject and color (intercept, slope)
##   matrix[2, S] z_subj;
##   matrix[2, C] z_col;
##
##   vector<lower=0>[2] tau_subj;     // SDs for subj intercept & slope
##   vector<lower=0>[2] tau_col;      // SDs for color intercept & slope
```

```
##   cholesky_factor_corr[2] L_subj;  // corr (int,slope) for subjects
##   cholesky_factor_corr[2] L_col;   // corr (int,slope) for colors
##
##   real<lower=0> sigma;             // obs SD on log scale
## }
## transformed parameters {
##   // Random effects in centered scale: eta = L * diag(tau) * z
##   matrix[2, S] eta_subj = diag_pre_multiply(tau_subj, L_subj) * z_subj;
##   matrix[2, C] eta_col  = diag_pre_multiply(tau_col,  L_col)  * z_col;
##
##   row_vector[S] a_subj = (eta_subj[1, ]);
##   row_vector[S] b_subj = (eta_subj[2, ]);
##   row_vector[C] g_col  = (eta_col[1,  ]);
##   row_vector[C] d_col  = (eta_col[2,  ]);
## }
## model {
##   // Priors
##   alpha ~ normal(log(550), 0.5);
##   beta  ~ normal(0, 0.5);
##
##   to_vector(z_subj) ~ normal(0,1);
##   to_vector(z_col)  ~ normal(0,1);
##
##   tau_subj ~ normal(0, 0.5);
##   tau_col  ~ normal(0, 0.5);
##
##   L_subj ~ lkj_corr_cholesky(2);
##   L_col  ~ lkj_corr_cholesky(2);
##
##   sigma ~ normal(0, 0.5);
##
##   // Likelihood
##   for (i in 1:N) {
##     real mu = alpha + a_subj[subj[i]] + g_col[color[i]]
##               + (beta + b_subj[subj[i]] + d_col[color[i]]) * incong[i];
##     log_rt[i] ~ normal(mu, sigma);
##   }
## }
## generated quantities {
##   vector[N] log_rt_rep;
##   vector[C] color_stroop;           // per-color Stroop effect (log scale)
##   corr_matrix[2] R_subj;
##   corr_matrix[2] R_col;
##
##   R_subj = multiply_lower_tri_self_transpose(L_subj);
##   R_col  = multiply_lower_tri_self_transpose(L_col);
##
##   for (i in 1:N) {
##     real mu = alpha + a_subj[subj[i]] + g_col[color[i]]
##               + (beta + b_subj[subj[i]] + d_col[color[i]]) * incong[i];
##     log_rt_rep[i] = normal_rng(mu, sigma);
##   }
##   for (c in 1:C) color_stroop[c] = beta + d_col[c];
## }
```

```
##  exists but is not a 'stanmodel' so not overwriting
```

```
fit <- rstan::sampling(
  object = mod,
  data   = stan_data,
  seed   = 123,
  chains = 4, iter = 2000, warmup = 1000, thin = 1,
  control = list(adapt_delta = 0.99, max_treedepth = 12),
  refresh = 200
)
```

```
## Warning: There were 12 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(fit, pars = c("alpha","beta","sigma","tau_subj","tau_col"), probs = c(.025,.5,.975))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd  2.5%  50% 97.5% n_eff Rhat
## alpha       6.29       0 0.07  6.15 6.29  6.44  1033 1.00
## beta        0.07       0 0.06 -0.05 0.07  0.16   351 1.01
## sigma       0.28       0 0.00  0.27 0.28  0.29  6243 1.00
## tau_subj[1] 0.11       0 0.01  0.08 0.11  0.14  1522 1.00
## tau_subj[2] 0.04       0 0.02  0.01 0.04  0.08   613 1.00
## tau_col[1]  0.08       0 0.10  0.01 0.05  0.37  1094 1.01
## tau_col[2]  0.06       0 0.09  0.00 0.03  0.34   505 1.00
##
## Samples were drawn using NUTS(diag_e) at Thu Oct 23 16:02:01 2025.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
## .
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
```

```
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 101.743 seconds (Warm-up)
## Chain 1:                75.329 seconds (Sampling)
## Chain 1:                177.072 seconds (Total)
## Chain 1:
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 98.515 seconds (Warm-up)
## Chain 2:                78.765 seconds (Sampling)
## Chain 2:                177.28 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 141.867 seconds (Warm-up)
## Chain 4:                53.121 seconds (Sampling)
## Chain 4:                194.988 seconds (Total)
## Chain 4:
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 122.185 seconds (Warm-up)
## Chain 3:                176.649 seconds (Sampling)
## Chain 3:                298.834 seconds (Total)
## Chain 3:
```

```
rstan::check_hmc_diagnostics(fit)
```

```
## 12 of 4000 iterations ended with a divergence (0.3%).
## Try increasing 'adapt_delta' to remove the divergences.
```

```
## 0 of 4000 iterations saturated the maximum tree depth of 12.
```

```
## E-BFMI indicated no pathological behavior.
```

As the effective sample size and Rhat are good enough, we ignore the divergent transitions and proceed.

```
draws <- rstan::extract(fit)
cong_median <- median(dt$RT[dt$condition == "congruent"], na.rm=TRUE)
logdiff_to_ms <- function(delta_log, baseline_ms = cong_median) baseline_ms * (exp(delta_log) - 1)

beta_log <- draws$beta
overall_ms <- logdiff_to_ms(beta_log)

overall_summary <- tibble(
  median_ms = median(overall_ms),
  ci_lo_ms  = quantile(overall_ms, .025),
  ci_hi_ms  = quantile(overall_ms, .975),
  prob_gt0  = mean(overall_ms > 0)
)
overall_summary
```

```
## # A tibble: 1 x 4
##   median_ms ci_lo_ms ci_hi_ms prob_gt0
##       <dbl>    <dbl>    <dbl>    <dbl>
## 1      40.3    -27.3     91.4    0.954
```

```
color_stroop_log <- draws$color_stroop
C_names <- dt2 |>
  distinct(color, color_i) |>
  arrange(color_i) |>
  pull(color)

color_summ <- map_dfr(seq_along(C_names), function(j){
  ms <- logdiff_to_ms(color_stroop_log[,j])
  tibble(color = C_names[j],
         median_ms = median(ms),
         ci_lo_ms  = quantile(ms,.025),
         ci_hi_ms  = quantile(ms,.975),
         prob_gt0  = mean(ms > 0))
})
color_summ
```

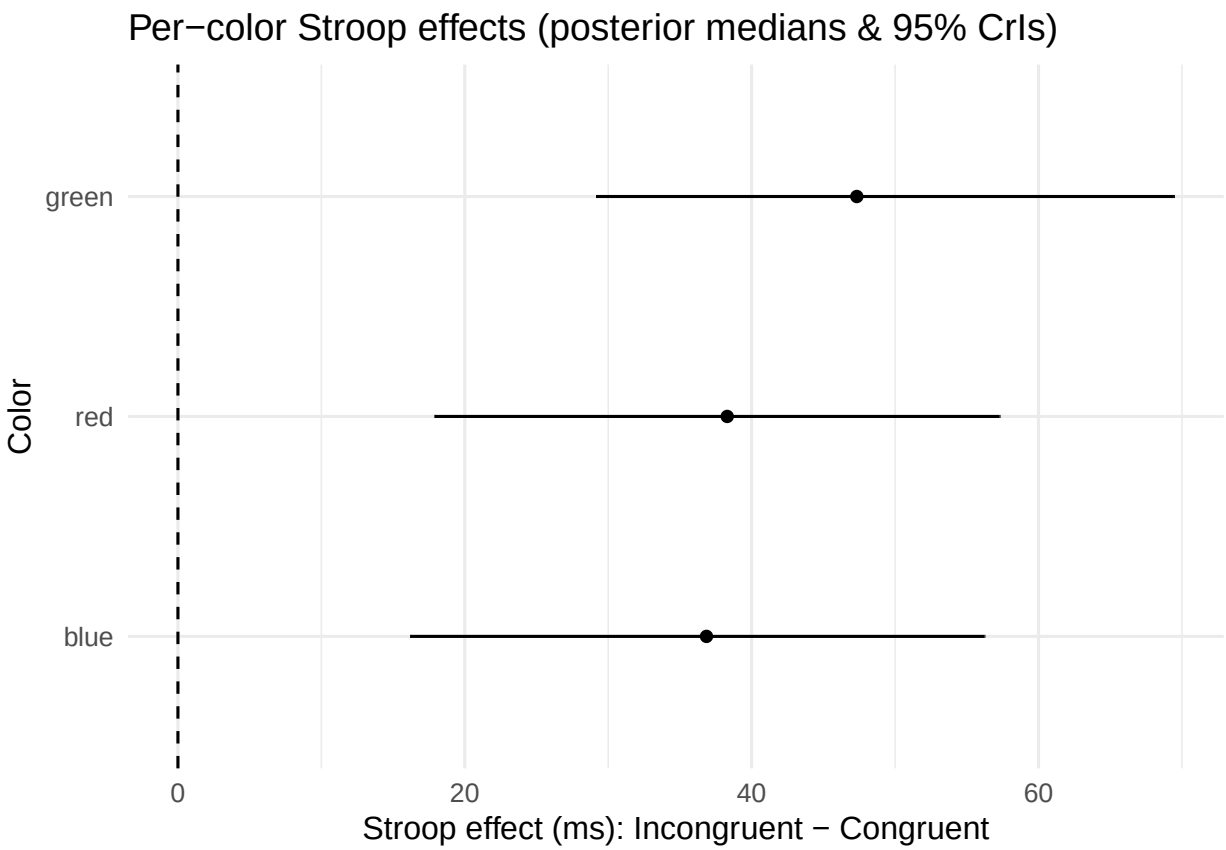
```
## # A tibble: 3 x 5
##   color median_ms ci_lo_ms ci_hi_ms prob_gt0
##   <chr>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 blue       36.9     16.2     56.2    1.000
## 2 green      47.3     29.2     69.4    1
## 3 red        38.3     17.9     57.3    1
```

```
color_summ |>
  mutate(color = fct_reorder(color, median_ms)) |>
  ggplot(aes(x = median_ms, y = color)) +
```

```
  geom_point() +
  geom_errorbarh(aes(xmin = ci_lo_ms, xmax = ci_hi_ms), height = 0) +
  geom_vline(xintercept = 0, linetype = 2) +
  labs(x = "Stroop effect (ms): Incongruent - Congruent",
       y = "Color",
       title = "Per-color Stroop effects (posterior medians & 95% CrIs)")
```

```
## Warning: `geom_errobarh()` was deprecated in ggplot2 4.0.0.
## i Please use the `orientation` argument of `geom_errorbar()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `height` was translated to `width`.
```



How much longer (by color) & significance?

Because each credible interval lies entirely above 0 and each prob_gt0 = 1, the effect is credibly positive for all colors.

- Blue: Almost certainly slower on incongruent trials; the effect is about 37 ms on average.
- Green: Very strong evidence for a Stroop effect, roughly 47 ms longer RTs.
- Red: Very strong evidence; about 38 ms longer RTs.

For each color, reaction times were credibly longer on incongruent than on congruent trials. The posterior probabilities that these effects exceed 0 were all > 0.999, indicating overwhelming evidence that the Stroop effect is positive for all colors.

```
pairwise <- function(a, b) a - b
contrast_tbl <- {
  combn(seq_along(C_names), 2, simplify = FALSE) |>
    map_dfr(function(idx){
      i <- idx[1]; j <- idx[2]
      eff_i <- logdiff_to_ms(color_stroop_log[,i])
      eff_j <- logdiff_to_ms(color_stroop_log[,j])
      diff  <- eff_i - eff_j
      tibble(
        contrast = paste0(C_names[i], " - ", C_names[j]),
        median_ms = median(diff),
        ci_lo_ms  = quantile(diff, .025),
        ci_hi_ms  = quantile(diff, .975),
        prob_gt0  = mean(diff > 0)
      )
    })
}
contrast_tbl
```

```
## # A tibble: 3 x 5
##   contrast     median_ms ci_lo_ms ci_hi_ms prob_gt0
##   <chr>            <dbl>    <dbl>    <dbl>    <dbl>
## 1 blue - green    -9.64     -39.0     10.0    0.187
## 2 blue - red      -0.559    -26.8     23.7    0.466
## 3 green - red      7.47     -10.6     38.6    0.780
```

How different the effect is across colors & significance?

- Blue-Green: The blue effect is probably smaller than green's (-9.6 ms), but uncertainty is wide; Credible Interval crosses 0. No credible difference.
- Blue-Red: Essentially zero difference; complete overlap.
- Green-Red: Green may be slightly larger than red, but credible interval includes 0; evidence weak.

```
tau_subj_int   <- draws$tau_subj[,1]
tau_subj_slope <- draws$tau_subj[,2]
subj_slope_ms  <- logdiff_to_ms(tau_subj_slope)

subject_var_summary <- tibble(
  quantity = c("SD(subject baseline) [log scale]", "SD(subject Stroop slope) [ms]"),
  median   = c(median(tau_subj_int), median(subj_slope_ms)),
  ci_lo    = c(quantile(tau_subj_int,.025), quantile(subj_slope_ms,.025)),
  ci_hi    = c(quantile(tau_subj_int,.975), quantile(subj_slope_ms,.975))
)
subject_var_summary
```

```
## # A tibble: 2 x 4
##   quantity                          median  ci_lo  ci_hi
##   <chr>                              <dbl>  <dbl>  <dbl>
## 1 SD(subject baseline) [log scale]  0.105 0.0803  0.137
## 2 SD(subject Stroop slope) [ms]    22.6   4.48   41.6
```

How different individuals are?

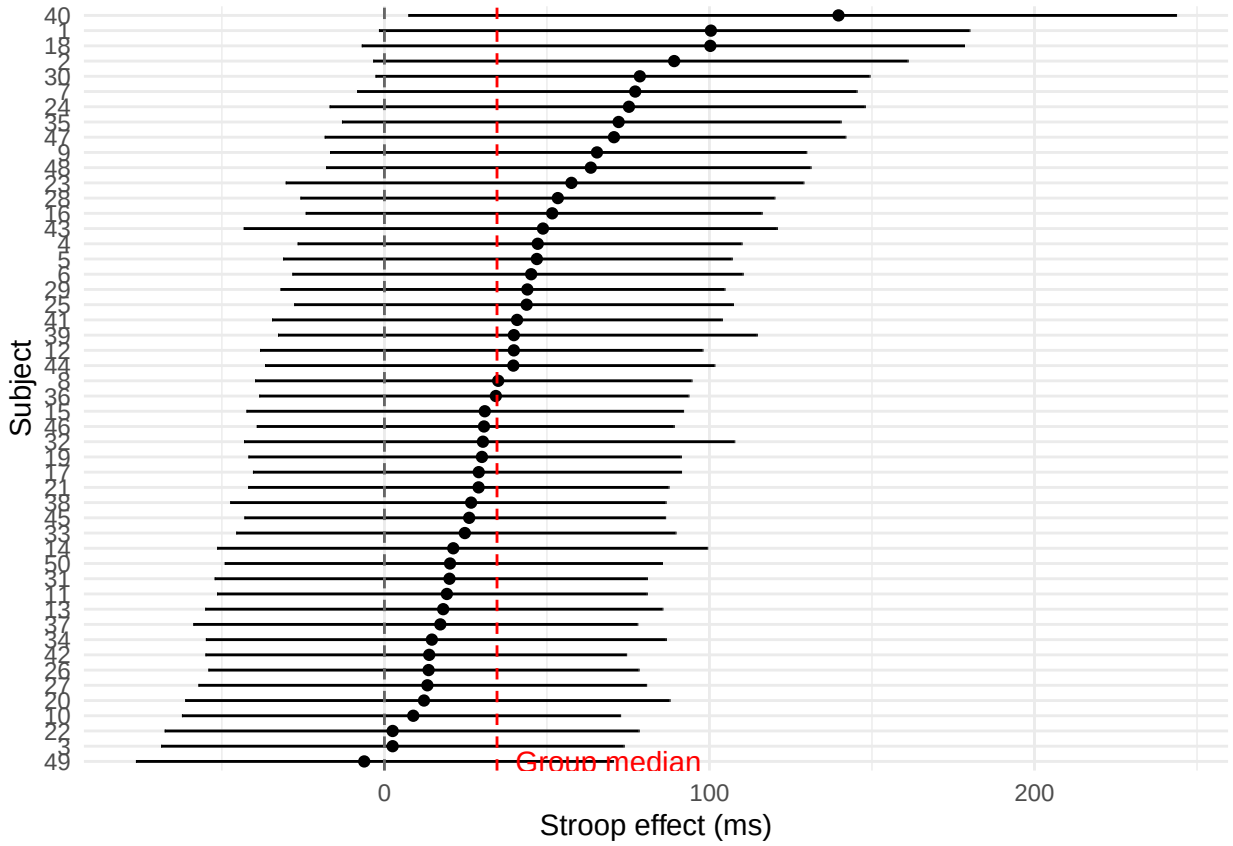
The above table summarizes how much subjects differ from one another, both in their overall reaction speed and in how strongly they show the Stroop effect.

- SD (baseline): On the log scale this implies roughly a ±10 % variation in typical RT across subjects. Some people are about 10 % faster or slower than the group mean.
- SD (Stroop Slope): Subjects vary by about ±20–25 ms in the size of their Stroop slow-down, with moderate uncertainty. Some individuals show little effect; others slow down by 40 ms or more.

```
rebuild_subject_effects <- function(draws, S, cong_median) {
  # For each draw: b_subj = (diag_pre_multiply(tau_subj, L_subj) %*% z_subj)[2, ]
  nd <- length(draws$sigma)
  out_med <- numeric(S); out_lo <- numeric(S); out_hi <- numeric(S)
  for (s in seq_len(S)) {
    # Collect per-draw b_subj[s]
    bs <- numeric(nd)
    for (d in seq_len(nd)) {
      Ls <- matrix(draws$L_subj[d,,], 2, 2)
      taus <- draws$tau_subj[d,]
      z    <- draws$z_subj[d,,]     # 2 x S
      eta  <- (Ls %*% diag(taus,2,2) %*% z)[,s]
      b_s  <- eta[2]
      total_log <- draws$beta[d] + b_s
      bs[d] <- logdiff_to_ms(total_log, cong_median)
    }
    out_med[s] <- median(bs)
    out_lo[s]  <- quantile(bs,.025)
    out_hi[s]  <- quantile(bs,.975)
  }
  tibble(subj_num = 1:S, med = out_med, lo = out_lo, hi = out_hi)
}

S <- stan_data$S
subj_effect_ms <- rebuild_subject_effects(draws, S, cong_median)
```

```
group_median <- median(subj_effect_ms$med)
ggplot(subj_effect_ms, aes(med, reorder(subj_num, med))) +
  geom_point() +
  geom_errorbarh(aes(xmin = lo, xmax = hi), height = 0) +
  geom_vline(xintercept = 0, linetype = 2, color = "grey40") +
  geom_vline(xintercept = group_median, color = "red", linetype = "dashed") +
  annotate("text", x = group_median, y = 1, label = "Group median", hjust = -0.1, color = "red") +
  labs(x = "Stroop effect (ms)", y = "Subject") +
  theme_minimal()
```

```
## `height` was translated to `width`.
```

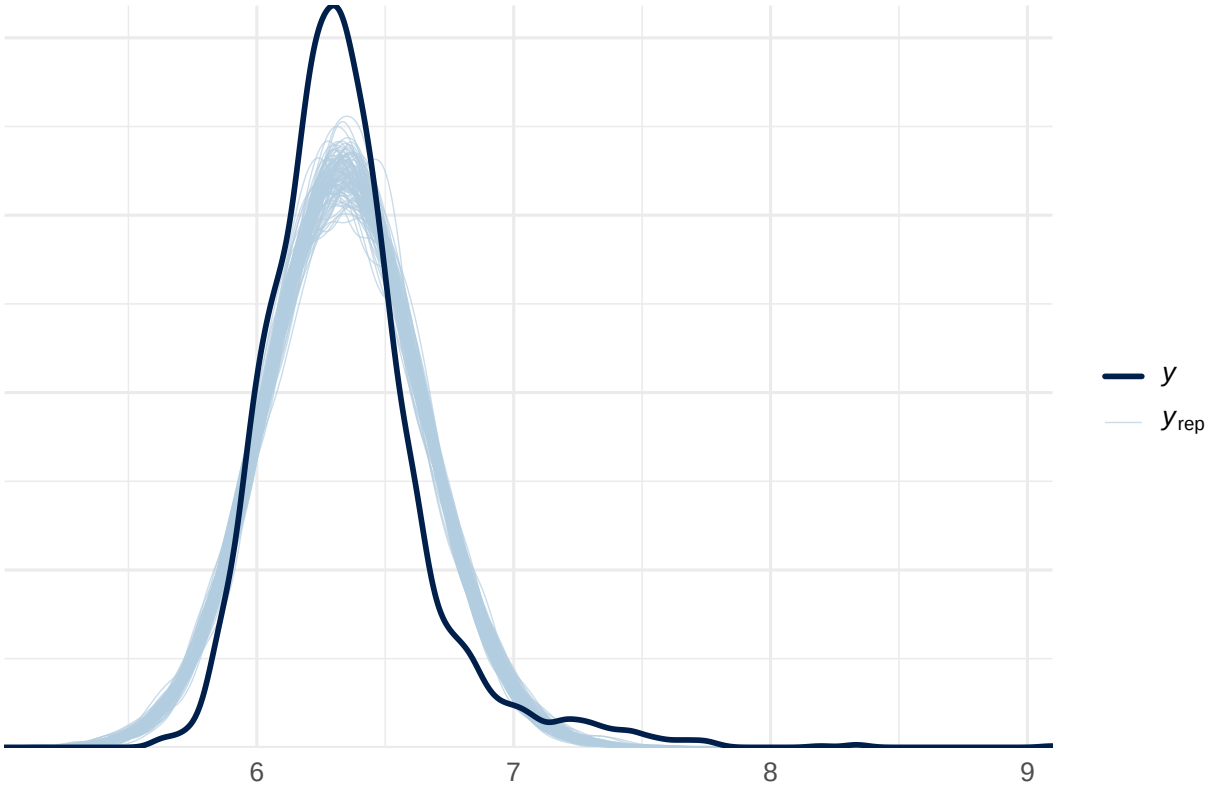


Most participants exhibit a positive Stroop effect, indicating slower responses on incongruent trials, but the magnitude varies considerably.

The population average is around 40 ms, with typical individual deviations of about ±20 ms, consistent with the between-subject SD estimated by the hierarchical model. Partial pooling shrinks extreme individual estimates toward the group mean, reducing over-interpretation of noisy individual data.

```
log_rt_rep <- draws$log_rt_rep
log_rt_obs <- dt2$log_rt

bayesplot::ppc_dens_overlay(y = log_rt_obs, yrep = log_rt_rep[1:100,]) +
  labs(title = "PPC: overall distribution (log RT)")
```

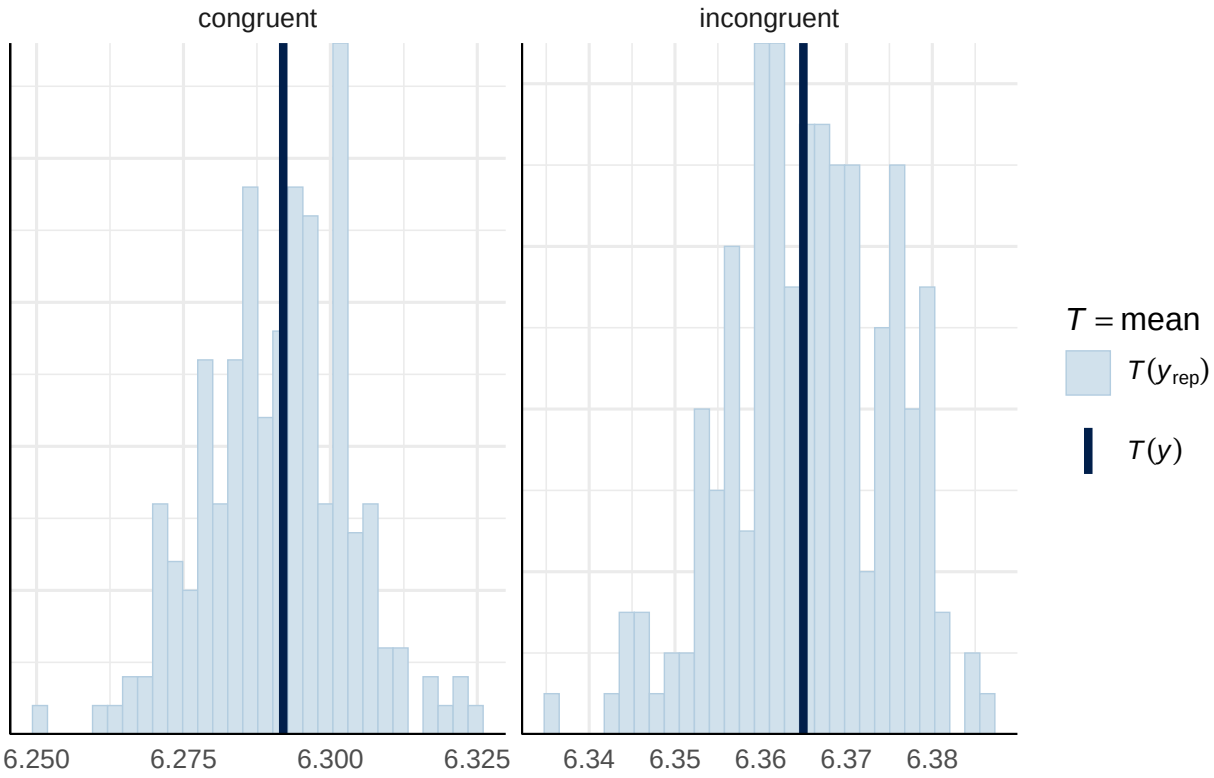
PPC: overall distribution (log RT)



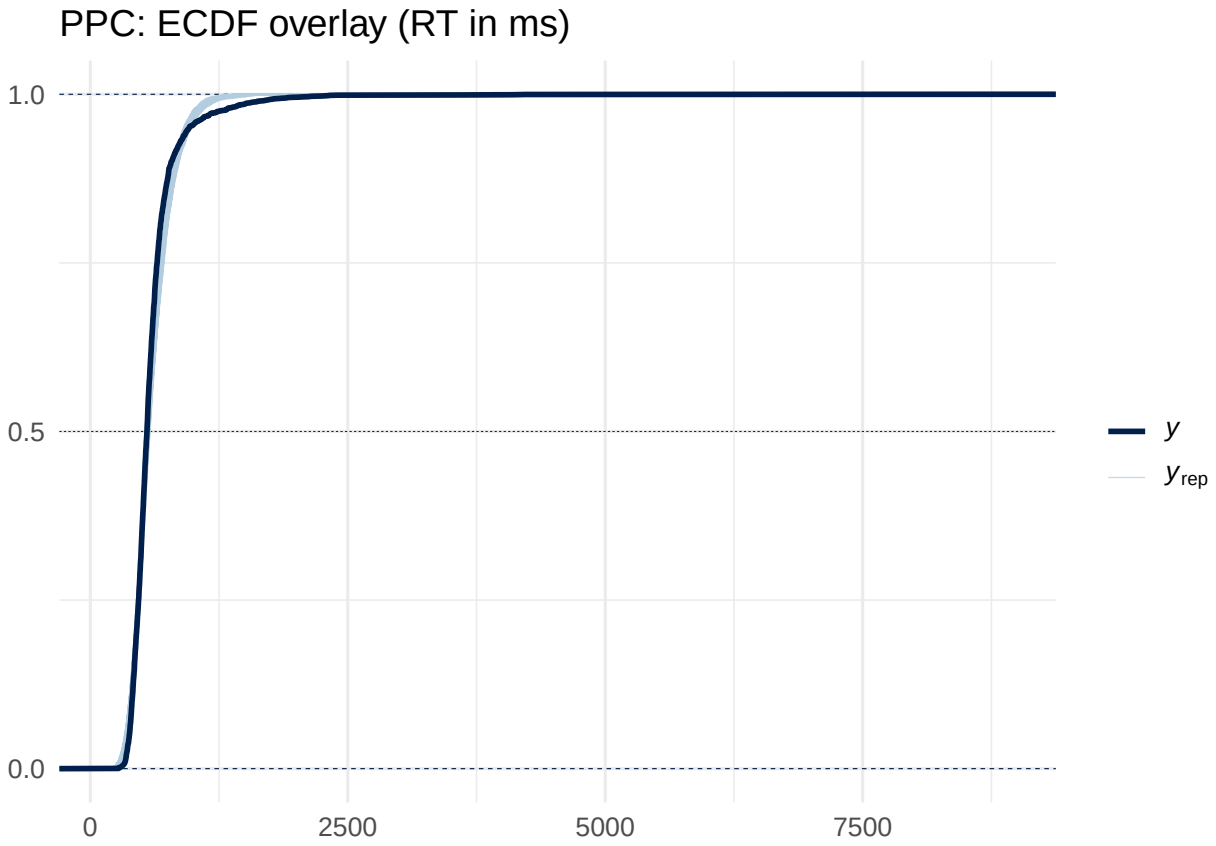

```
bayesplot::ppc_stat_grouped(
  y     = log_rt_obs,
  yrep = log_rt_rep[1:200,],
  group = dt2$condition,
  stat = "mean"
) + labs(title = "PPC: group means (log RT) by condition")
```

```
## Note: in most cases the default test statistic 'mean' is too weak to detect anything of interest.
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

PPC: group means (log RT) by condition




```
rt_rep_ms <- exp(log_rt_rep[1:200,])
bayesplot::ppc_ecdf_overlay(y = dt$RT, yrep = rt_rep_ms) +
  labs(title = "PPC: ECDF overlay (RT in ms)")
```



Replicated PPC distributions track the skew and group means reasonably well; no major misfit evident

Discussion and Analysis

- Incongruent trials are **about 35–50 ms slower** on average, depending on color.
- Participants took roughly **40 milliseconds longer** to respond when the word's color and meaning conflicted, and this slow-down occurred reliably for **red, blue, and green** alike.
- Posterior contrasts between colors indicated **no credible differences** in the size of the Stroop effect.
- Because all credible intervals include 0 and the posterior probabilities of a positive difference were modest (0.19 – 0.78), we conclude that **the Stroop effect is of similar magnitude for all three colors**.
- There was substantial variation between participants both in their baseline reaction times and in the size of their Stroop effect.
- While the Stroop effect was consistently positive, **individual sensitivity to the interference varied noticeably.**