Variance All the Way Down: Exploring the Impact of RNA-Seq Pipeline Choices on Differential Expression Variance

Art Tay and Hunter Schuler

Data

We first pull raw RNA-Seq reads for the ovarian cancer study GSE282674 from the NCBI-NIH sequence read archive (SRA). This dataset contains 9 samples with 3 in the treatment group. The reads are also single-ended. The data was extracted using the default `prefetch` settings in the `sra-toolkit`. We then generated `fastq` files using `fasterq-dump`. We initially varied the parameters, but found no impact on the resulting `fastq` files. We then applied `fastp` to filter out low quality reads and trim excess bases. After that, the reads were aligned and the genes counted. At this stage we save the resulting count matrices under the various aligners and `fastp` parameters, so that they can be compared to the official count matrix posted on the archive. After this, we used `edgeR` to normalize the counts and compute p-values and effect sizes. Baseline p-values and effect sizes were computing using the official count matrix and no count normalization. See Table 1 for pipeline parameters.

Table 1: Basic RNA-Seq Differential Analysis End-to-End Pipeline

Pipeline Steps	Software	Options	Choices
1. Pull SRA data from the NIH.	prefetch	NA	NA
2. Compute quality scores.	fasterq-dump	NA	NA
3. Filter low quality reads.	fastp	–qualified_quality_phred X –length_required X	20-30 30-50
4. Trim excess bases.	fastp	–trim_poly_g –trim_ploy_x	1 or 0 1 or 0
5. Align and count genes.	Various	Default	Salmon, Kallisto STAR, HISAT2
6. Count normalization.	edgeR	calcNormFactors(method='X')	TMM, TMMwsp, RLE upperquartile ALDEx2, none
7. Differential expression analysis.	edgeR	Default	NA

We are interested in the effect of pipeline choices on the variation of 3 quantities: counts, p-values, and effect sizes. To quantify variation we compute the average sum of squared differences between pipeline outputs and the NIH baseline. Assume that there are G genes. Let A_{gX} denote the resulting outcome of pipeline X, and let B_g denote the corresponding result based on the official count matrix. Now we can define the form of our target variable:

$$Y^2 = \frac{1}{G}\sum_{g=1}^{G}(A_{gX} - B_g)^2 \tag{1}$$

Note that the true variable of interest is Y, since it will be on the same scale as the underlying variable.

Preprocessing

```
count_recipe <- count_sd_df |>
    select(-c(runtime_sec, gene_overlap_percent)) |>
    recipe(count_sd ~ .) |>
    step_dummy(all_factor_predictors()) |>
    step_interact(terms = ~ (all_predictors())^2) |>
    # Removes interactions between different levels of the same dummy.
    step_zv(all_predictors()) |>
    step_intercept() |>
    prep()

count_sd_df_clean <- count_recipe |> bake(new_data = NULL)

DE_recipe <- DE_sd_df |>
    select(-c(runtime_sec, gene_overlap_percent)) |>
    recipe(p_value_sd + effect_size_sd ~ .) |>
    step_dummy(all_factor_predictors()) |>
    step_interact(terms = ~ (all_predictors())^2) |>
    # Removes interactions between different levels of the same dummy.
    step_zv(all_predictors()) |>
    step_intercept() |>
    prep()

DE_sd_df_clean <- DE_recipe |> bake(new_data = NULL)
p_value_sd_df_clean <- DE_sd_df_clean |> select(-effect_size_sd)
effect_size_sd_df_clean <- DE_sd_df_clean |> select(-p_value_sd)
```

Models

We propose a Bayesian hierarchical model building from the fact that $Y \geq 0$.

$$\begin{aligned}
Y_i &\sim f_{Y_i}(y) \quad s.t. \quad Y_i \geq 0 \\
\mathbb{E}\, Y_i &= \exp(X_i\beta) \quad \text{since} \quad \mathbb{E}\, Y_i \geq 0 \\
\mathbb{V}\, Y_i &= a \cdot \mu_i^b \\
\beta &\sim \mathcal{N}(0, \sigma_\beta^2) \\
a &\sim \text{Gamma}(c, d) \\
b &\sim \mathcal{N}(0, \sigma_b^2)
\end{aligned} \tag{2}$$

This model has a couple of key advantages: 1. Assumption are explicit, but variable.

1. Natural parameter shrinkage via the prior on β. Handles multicollinearity and high dimensionality of X.
2. Does not assume constant variance. Specifically, we are applying the variance-power law from the Tweedie family of distributions [cite].

$$\text{Var}\, Y \propto (\mathbb{E}\, Y)^p \tag{3}$$

$a > 0$ and represents a common variance scale ie if $b = 0$ we recover the classical log-Normal regression model. $b \in \mathbb{R}$ where $b > 0$ indicates over-dispersion and $b < 0$ indicates under-dispersion.

3. Clear interpretation and interval estimation.
4. Clear model diagnostic checks.

Pre-specified Model Settings:

1. $f_{Y_i}(y)$ is log-Normal, Gamma, or Weibull.
2. $\sigma_\beta^2 = 100 \Rightarrow$ flat prior.
3. $c = d = 0.01 \Rightarrow$ flat prior.
4. $\sigma_b^2 = 1$. High shrinkage because we do not expect crazy over dispersion.

Pre-specified Model Validation:

1. Convergence - $\hat{R} \leq 1.1$
2. Fit - Posterior Predictive Distribution Quantiles (10%)
3. General Stan warnings.

Decision Criteria:

1. Assuming valid fit a convergence, we will base the "significance" of a predictor based on a 95% credible interval.
2. Agreement across likelihood is seen as stronger evidence.

Results

```
library(cmdstanr)
# cmdstanr::install_cmdstan() # One time.

options(mc.cores = parallel::detectCores())

n_chains <- 4
n_iter <- 5000
burn_in <- 1000
```

```
lik <- 1 #1=log-Normal, 2=Gamma, 3=Weibull
sigma_beta <- 100
c <- 0.01; d <- 0.01
sigma_b <- 1

data <- count_sd_df_clean
X <- data |> select(-count_sd)
Y <- data |> pull(count_sd)
N <- dim(X)[1]
P <- dim(X)[2]

stan_data_list <- list(
    N = N, P = P, X = X, Y = Y,
    lik = lik, sigma_beta = sigma_beta, c = c, d = d, sigma_b = sigma_b
)
```

```r
model_1 <- cmdstan_model("./model.stan")

fit_1 <- model_1$sample(
    data = stan_data_list,
    chains = n_chains, iter_sampling = n_iter, iter_warmup = burn_in,
    seed = 04272025
)
```

```
Running MCMC with 4 chains, at most 64 in parallel...

Chain 1 Iteration:    1 / 6000 [  0%]  (Warmup)
Chain 2 Iteration:    1 / 6000 [  0%]  (Warmup)
Chain 3 Iteration:    1 / 6000 [  0%]  (Warmup)
Chain 4 Iteration:    1 / 6000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 6000 [  1%]  (Warmup)
Chain 1 Iteration:  100 / 6000 [  1%]  (Warmup)
Chain 4 Iteration:  100 / 6000 [  1%]  (Warmup)
Chain 3 Iteration:  100 / 6000 [  1%]  (Warmup)
Chain 2 Iteration:  200 / 6000 [  3%]  (Warmup)
Chain 3 Iteration:  200 / 6000 [  3%]  (Warmup)
Chain 4 Iteration:  200 / 6000 [  3%]  (Warmup)
Chain 1 Iteration:  200 / 6000 [  3%]  (Warmup)
Chain 2 Iteration:  300 / 6000 [  5%]  (Warmup)
Chain 1 Iteration:  300 / 6000 [  5%]  (Warmup)
Chain 3 Iteration:  300 / 6000 [  5%]  (Warmup)
Chain 4 Iteration:  300 / 6000 [  5%]  (Warmup)
Chain 2 Iteration:  400 / 6000 [  6%]  (Warmup)
Chain 3 Iteration:  400 / 6000 [  6%]  (Warmup)
Chain 4 Iteration:  400 / 6000 [  6%]  (Warmup)
Chain 1 Iteration:  400 / 6000 [  6%]  (Warmup)
Chain 2 Iteration:  500 / 6000 [  8%]  (Warmup)
Chain 3 Iteration:  500 / 6000 [  8%]  (Warmup)
Chain 4 Iteration:  500 / 6000 [  8%]  (Warmup)
Chain 1 Iteration:  500 / 6000 [  8%]  (Warmup)
Chain 2 Iteration:  600 / 6000 [ 10%]  (Warmup)
Chain 3 Iteration:  600 / 6000 [ 10%]  (Warmup)
Chain 4 Iteration:  600 / 6000 [ 10%]  (Warmup)
Chain 1 Iteration:  600 / 6000 [ 10%]  (Warmup)
Chain 2 Iteration:  700 / 6000 [ 11%]  (Warmup)
Chain 3 Iteration:  700 / 6000 [ 11%]  (Warmup)
Chain 4 Iteration:  700 / 6000 [ 11%]  (Warmup)
Chain 1 Iteration:  700 / 6000 [ 11%]  (Warmup)
Chain 2 Iteration:  800 / 6000 [ 13%]  (Warmup)
Chain 3 Iteration:  800 / 6000 [ 13%]  (Warmup)
Chain 1 Iteration:  800 / 6000 [ 13%]  (Warmup)
Chain 4 Iteration:  800 / 6000 [ 13%]  (Warmup)
Chain 2 Iteration:  900 / 6000 [ 15%]  (Warmup)
Chain 3 Iteration:  900 / 6000 [ 15%]  (Warmup)
Chain 4 Iteration:  900 / 6000 [ 15%]  (Warmup)
Chain 1 Iteration:  900 / 6000 [ 15%]  (Warmup)
Chain 2 Iteration: 1000 / 6000 [ 16%]  (Warmup)
Chain 2 Iteration: 1001 / 6000 [ 16%]  (Sampling)
```

```
Chain 3 Iteration: 1000 / 6000 [ 16%]  (Warmup)
Chain 3 Iteration: 1001 / 6000 [ 16%]  (Sampling)
Chain 4 Iteration: 1000 / 6000 [ 16%]  (Warmup)
Chain 4 Iteration: 1001 / 6000 [ 16%]  (Sampling)
Chain 1 Iteration: 1000 / 6000 [ 16%]  (Warmup)
Chain 1 Iteration: 1001 / 6000 [ 16%]  (Sampling)
Chain 2 Iteration: 1100 / 6000 [ 18%]  (Sampling)
Chain 3 Iteration: 1100 / 6000 [ 18%]  (Sampling)
Chain 4 Iteration: 1100 / 6000 [ 18%]  (Sampling)
Chain 1 Iteration: 1100 / 6000 [ 18%]  (Sampling)
Chain 2 Iteration: 1200 / 6000 [ 20%]  (Sampling)
Chain 3 Iteration: 1200 / 6000 [ 20%]  (Sampling)
Chain 4 Iteration: 1200 / 6000 [ 20%]  (Sampling)
Chain 1 Iteration: 1200 / 6000 [ 20%]  (Sampling)
Chain 2 Iteration: 1300 / 6000 [ 21%]  (Sampling)
Chain 3 Iteration: 1300 / 6000 [ 21%]  (Sampling)
Chain 4 Iteration: 1300 / 6000 [ 21%]  (Sampling)
Chain 1 Iteration: 1300 / 6000 [ 21%]  (Sampling)
Chain 2 Iteration: 1400 / 6000 [ 23%]  (Sampling)
Chain 3 Iteration: 1400 / 6000 [ 23%]  (Sampling)
Chain 4 Iteration: 1400 / 6000 [ 23%]  (Sampling)
Chain 1 Iteration: 1400 / 6000 [ 23%]  (Sampling)
Chain 2 Iteration: 1500 / 6000 [ 25%]  (Sampling)
Chain 3 Iteration: 1500 / 6000 [ 25%]  (Sampling)
Chain 1 Iteration: 1500 / 6000 [ 25%]  (Sampling)
Chain 4 Iteration: 1500 / 6000 [ 25%]  (Sampling)
Chain 2 Iteration: 1600 / 6000 [ 26%]  (Sampling)
Chain 3 Iteration: 1600 / 6000 [ 26%]  (Sampling)
Chain 1 Iteration: 1600 / 6000 [ 26%]  (Sampling)
Chain 4 Iteration: 1600 / 6000 [ 26%]  (Sampling)
Chain 2 Iteration: 1700 / 6000 [ 28%]  (Sampling)
Chain 3 Iteration: 1700 / 6000 [ 28%]  (Sampling)
Chain 4 Iteration: 1700 / 6000 [ 28%]  (Sampling)
Chain 1 Iteration: 1700 / 6000 [ 28%]  (Sampling)
Chain 2 Iteration: 1800 / 6000 [ 30%]  (Sampling)
Chain 3 Iteration: 1800 / 6000 [ 30%]  (Sampling)
Chain 4 Iteration: 1800 / 6000 [ 30%]  (Sampling)
Chain 1 Iteration: 1800 / 6000 [ 30%]  (Sampling)
Chain 2 Iteration: 1900 / 6000 [ 31%]  (Sampling)
Chain 3 Iteration: 1900 / 6000 [ 31%]  (Sampling)
Chain 4 Iteration: 1900 / 6000 [ 31%]  (Sampling)
Chain 1 Iteration: 1900 / 6000 [ 31%]  (Sampling)
Chain 2 Iteration: 2000 / 6000 [ 33%]  (Sampling)
Chain 3 Iteration: 2000 / 6000 [ 33%]  (Sampling)
Chain 1 Iteration: 2000 / 6000 [ 33%]  (Sampling)
Chain 4 Iteration: 2000 / 6000 [ 33%]  (Sampling)
Chain 2 Iteration: 2100 / 6000 [ 35%]  (Sampling)
Chain 1 Iteration: 2100 / 6000 [ 35%]  (Sampling)
Chain 3 Iteration: 2100 / 6000 [ 35%]  (Sampling)
Chain 4 Iteration: 2100 / 6000 [ 35%]  (Sampling)
Chain 2 Iteration: 2200 / 6000 [ 36%]  (Sampling)
Chain 3 Iteration: 2200 / 6000 [ 36%]  (Sampling)
Chain 1 Iteration: 2200 / 6000 [ 36%]  (Sampling)
Chain 4 Iteration: 2200 / 6000 [ 36%]  (Sampling)
```

```
Chain 2 Iteration: 2300 / 6000 [ 38%]  (Sampling)
Chain 3 Iteration: 2300 / 6000 [ 38%]  (Sampling)
Chain 1 Iteration: 2300 / 6000 [ 38%]  (Sampling)
Chain 4 Iteration: 2300 / 6000 [ 38%]  (Sampling)
Chain 2 Iteration: 2400 / 6000 [ 40%]  (Sampling)
Chain 1 Iteration: 2400 / 6000 [ 40%]  (Sampling)
Chain 3 Iteration: 2400 / 6000 [ 40%]  (Sampling)
Chain 4 Iteration: 2400 / 6000 [ 40%]  (Sampling)
Chain 2 Iteration: 2500 / 6000 [ 41%]  (Sampling)
Chain 3 Iteration: 2500 / 6000 [ 41%]  (Sampling)
Chain 1 Iteration: 2500 / 6000 [ 41%]  (Sampling)
Chain 4 Iteration: 2500 / 6000 [ 41%]  (Sampling)
Chain 2 Iteration: 2600 / 6000 [ 43%]  (Sampling)
Chain 3 Iteration: 2600 / 6000 [ 43%]  (Sampling)
Chain 1 Iteration: 2600 / 6000 [ 43%]  (Sampling)
Chain 4 Iteration: 2600 / 6000 [ 43%]  (Sampling)
Chain 2 Iteration: 2700 / 6000 [ 45%]  (Sampling)
Chain 1 Iteration: 2700 / 6000 [ 45%]  (Sampling)
Chain 3 Iteration: 2700 / 6000 [ 45%]  (Sampling)
Chain 4 Iteration: 2700 / 6000 [ 45%]  (Sampling)
Chain 1 Iteration: 2800 / 6000 [ 46%]  (Sampling)
Chain 2 Iteration: 2800 / 6000 [ 46%]  (Sampling)
Chain 3 Iteration: 2800 / 6000 [ 46%]  (Sampling)
Chain 4 Iteration: 2800 / 6000 [ 46%]  (Sampling)
Chain 2 Iteration: 2900 / 6000 [ 48%]  (Sampling)
Chain 3 Iteration: 2900 / 6000 [ 48%]  (Sampling)
Chain 1 Iteration: 2900 / 6000 [ 48%]  (Sampling)
Chain 4 Iteration: 2900 / 6000 [ 48%]  (Sampling)
Chain 3 Iteration: 3000 / 6000 [ 50%]  (Sampling)
Chain 1 Iteration: 3000 / 6000 [ 50%]  (Sampling)
Chain 2 Iteration: 3000 / 6000 [ 50%]  (Sampling)
Chain 4 Iteration: 3000 / 6000 [ 50%]  (Sampling)
Chain 1 Iteration: 3100 / 6000 [ 51%]  (Sampling)
Chain 2 Iteration: 3100 / 6000 [ 51%]  (Sampling)
Chain 3 Iteration: 3100 / 6000 [ 51%]  (Sampling)
Chain 4 Iteration: 3100 / 6000 [ 51%]  (Sampling)
Chain 1 Iteration: 3200 / 6000 [ 53%]  (Sampling)
Chain 2 Iteration: 3200 / 6000 [ 53%]  (Sampling)
Chain 3 Iteration: 3200 / 6000 [ 53%]  (Sampling)
Chain 4 Iteration: 3200 / 6000 [ 53%]  (Sampling)
Chain 3 Iteration: 3300 / 6000 [ 55%]  (Sampling)
Chain 1 Iteration: 3300 / 6000 [ 55%]  (Sampling)
Chain 2 Iteration: 3300 / 6000 [ 55%]  (Sampling)
Chain 4 Iteration: 3300 / 6000 [ 55%]  (Sampling)
Chain 3 Iteration: 3400 / 6000 [ 56%]  (Sampling)
Chain 1 Iteration: 3400 / 6000 [ 56%]  (Sampling)
Chain 2 Iteration: 3400 / 6000 [ 56%]  (Sampling)
Chain 4 Iteration: 3400 / 6000 [ 56%]  (Sampling)
Chain 1 Iteration: 3500 / 6000 [ 58%]  (Sampling)
Chain 3 Iteration: 3500 / 6000 [ 58%]  (Sampling)
Chain 2 Iteration: 3500 / 6000 [ 58%]  (Sampling)
Chain 4 Iteration: 3500 / 6000 [ 58%]  (Sampling)
Chain 1 Iteration: 3600 / 6000 [ 60%]  (Sampling)
Chain 3 Iteration: 3600 / 6000 [ 60%]  (Sampling)
```

```
Chain 2 Iteration: 3600 / 6000 [ 60%]  (Sampling)
Chain 4 Iteration: 3600 / 6000 [ 60%]  (Sampling)
Chain 1 Iteration: 3700 / 6000 [ 61%]  (Sampling)
Chain 3 Iteration: 3700 / 6000 [ 61%]  (Sampling)
Chain 2 Iteration: 3700 / 6000 [ 61%]  (Sampling)
Chain 4 Iteration: 3700 / 6000 [ 61%]  (Sampling)
Chain 1 Iteration: 3800 / 6000 [ 63%]  (Sampling)
Chain 3 Iteration: 3800 / 6000 [ 63%]  (Sampling)
Chain 2 Iteration: 3800 / 6000 [ 63%]  (Sampling)
Chain 4 Iteration: 3800 / 6000 [ 63%]  (Sampling)
Chain 1 Iteration: 3900 / 6000 [ 65%]  (Sampling)
Chain 3 Iteration: 3900 / 6000 [ 65%]  (Sampling)
Chain 2 Iteration: 3900 / 6000 [ 65%]  (Sampling)
Chain 4 Iteration: 3900 / 6000 [ 65%]  (Sampling)
Chain 3 Iteration: 4000 / 6000 [ 66%]  (Sampling)
Chain 1 Iteration: 4000 / 6000 [ 66%]  (Sampling)
Chain 2 Iteration: 4000 / 6000 [ 66%]  (Sampling)
Chain 4 Iteration: 4000 / 6000 [ 66%]  (Sampling)
Chain 3 Iteration: 4100 / 6000 [ 68%]  (Sampling)
Chain 1 Iteration: 4100 / 6000 [ 68%]  (Sampling)
Chain 2 Iteration: 4100 / 6000 [ 68%]  (Sampling)
Chain 4 Iteration: 4100 / 6000 [ 68%]  (Sampling)
Chain 3 Iteration: 4200 / 6000 [ 70%]  (Sampling)
Chain 1 Iteration: 4200 / 6000 [ 70%]  (Sampling)
Chain 2 Iteration: 4200 / 6000 [ 70%]  (Sampling)
Chain 4 Iteration: 4200 / 6000 [ 70%]  (Sampling)
Chain 1 Iteration: 4300 / 6000 [ 71%]  (Sampling)
Chain 3 Iteration: 4300 / 6000 [ 71%]  (Sampling)
Chain 2 Iteration: 4300 / 6000 [ 71%]  (Sampling)
Chain 4 Iteration: 4300 / 6000 [ 71%]  (Sampling)
Chain 1 Iteration: 4400 / 6000 [ 73%]  (Sampling)
Chain 3 Iteration: 4400 / 6000 [ 73%]  (Sampling)
Chain 2 Iteration: 4400 / 6000 [ 73%]  (Sampling)
Chain 4 Iteration: 4400 / 6000 [ 73%]  (Sampling)
Chain 1 Iteration: 4500 / 6000 [ 75%]  (Sampling)
Chain 3 Iteration: 4500 / 6000 [ 75%]  (Sampling)
Chain 2 Iteration: 4500 / 6000 [ 75%]  (Sampling)
Chain 4 Iteration: 4500 / 6000 [ 75%]  (Sampling)
Chain 1 Iteration: 4600 / 6000 [ 76%]  (Sampling)
Chain 3 Iteration: 4600 / 6000 [ 76%]  (Sampling)
Chain 4 Iteration: 4600 / 6000 [ 76%]  (Sampling)
Chain 2 Iteration: 4600 / 6000 [ 76%]  (Sampling)
Chain 1 Iteration: 4700 / 6000 [ 78%]  (Sampling)
Chain 3 Iteration: 4700 / 6000 [ 78%]  (Sampling)
Chain 2 Iteration: 4700 / 6000 [ 78%]  (Sampling)
Chain 4 Iteration: 4700 / 6000 [ 78%]  (Sampling)
Chain 1 Iteration: 4800 / 6000 [ 80%]  (Sampling)
Chain 3 Iteration: 4800 / 6000 [ 80%]  (Sampling)
Chain 2 Iteration: 4800 / 6000 [ 80%]  (Sampling)
Chain 4 Iteration: 4800 / 6000 [ 80%]  (Sampling)
Chain 1 Iteration: 4900 / 6000 [ 81%]  (Sampling)
Chain 3 Iteration: 4900 / 6000 [ 81%]  (Sampling)
Chain 2 Iteration: 4900 / 6000 [ 81%]  (Sampling)
Chain 4 Iteration: 4900 / 6000 [ 81%]  (Sampling)
```

```
Chain 1 Iteration: 5000 / 6000 [ 83%]  (Sampling)
Chain 3 Iteration: 5000 / 6000 [ 83%]  (Sampling)
Chain 2 Iteration: 5000 / 6000 [ 83%]  (Sampling)
Chain 4 Iteration: 5000 / 6000 [ 83%]  (Sampling)
Chain 1 Iteration: 5100 / 6000 [ 85%]  (Sampling)
Chain 3 Iteration: 5100 / 6000 [ 85%]  (Sampling)
Chain 2 Iteration: 5100 / 6000 [ 85%]  (Sampling)
Chain 4 Iteration: 5100 / 6000 [ 85%]  (Sampling)
Chain 1 Iteration: 5200 / 6000 [ 86%]  (Sampling)
Chain 3 Iteration: 5200 / 6000 [ 86%]  (Sampling)
Chain 2 Iteration: 5200 / 6000 [ 86%]  (Sampling)
Chain 4 Iteration: 5200 / 6000 [ 86%]  (Sampling)
Chain 1 Iteration: 5300 / 6000 [ 88%]  (Sampling)
Chain 3 Iteration: 5300 / 6000 [ 88%]  (Sampling)
Chain 2 Iteration: 5300 / 6000 [ 88%]  (Sampling)
Chain 4 Iteration: 5300 / 6000 [ 88%]  (Sampling)
Chain 1 Iteration: 5400 / 6000 [ 90%]  (Sampling)
Chain 3 Iteration: 5400 / 6000 [ 90%]  (Sampling)
Chain 2 Iteration: 5400 / 6000 [ 90%]  (Sampling)
Chain 4 Iteration: 5400 / 6000 [ 90%]  (Sampling)
Chain 1 Iteration: 5500 / 6000 [ 91%]  (Sampling)
Chain 3 Iteration: 5500 / 6000 [ 91%]  (Sampling)
Chain 2 Iteration: 5500 / 6000 [ 91%]  (Sampling)
Chain 4 Iteration: 5500 / 6000 [ 91%]  (Sampling)
Chain 1 Iteration: 5600 / 6000 [ 93%]  (Sampling)
Chain 3 Iteration: 5600 / 6000 [ 93%]  (Sampling)
Chain 2 Iteration: 5600 / 6000 [ 93%]  (Sampling)
Chain 4 Iteration: 5600 / 6000 [ 93%]  (Sampling)
Chain 1 Iteration: 5700 / 6000 [ 95%]  (Sampling)
Chain 3 Iteration: 5700 / 6000 [ 95%]  (Sampling)
Chain 2 Iteration: 5700 / 6000 [ 95%]  (Sampling)
Chain 4 Iteration: 5700 / 6000 [ 95%]  (Sampling)
Chain 1 Iteration: 5800 / 6000 [ 96%]  (Sampling)
Chain 3 Iteration: 5800 / 6000 [ 96%]  (Sampling)
Chain 2 Iteration: 5800 / 6000 [ 96%]  (Sampling)
Chain 4 Iteration: 5800 / 6000 [ 96%]  (Sampling)
Chain 1 Iteration: 5900 / 6000 [ 98%]  (Sampling)
Chain 3 Iteration: 5900 / 6000 [ 98%]  (Sampling)
Chain 2 Iteration: 5900 / 6000 [ 98%]  (Sampling)
Chain 4 Iteration: 5900 / 6000 [ 98%]  (Sampling)
Chain 1 Iteration: 6000 / 6000 [100%]  (Sampling)
Chain 1 finished in 172.6 seconds.
Chain 3 Iteration: 6000 / 6000 [100%]  (Sampling)
Chain 3 finished in 172.7 seconds.
Chain 4 Iteration: 6000 / 6000 [100%]  (Sampling)
Chain 2 Iteration: 6000 / 6000 [100%]  (Sampling)
Chain 2 finished in 173.0 seconds.
Chain 4 finished in 172.9 seconds.

All 4 chains finished successfully.
Mean chain execution time: 172.8 seconds.
Total execution time: 173.2 seconds.
```

Disclosures

AI Prompts:

Initial Prompt

Background:

1. There are 3 data frames each formatted like columns [outcome, intercept, predictors …].
2. The outcome Y is always positive.
3. The goal is the determine which predictors have non-zero marginal effects.

Requirements:

Use stan to fit hierarchical regression models.

1. For the likelihood, test a log-normal, a gamma, and a weibull. Note that we need to model the mean or a simple function of the mean. Ensure that the marginal effects can be put on the same scale as the outcome.
2. For the priors on the predictor marginal effects (betas) test Normal(0, 1), Normal(0, 10), and Normal(0, 100).
3. For each model report:

   Table 1:

   1. `Point estimate and 95% credible intervals for each predictor effect.`
   2. `The R hat convergence statistics.`
   3. `Effect sample size divided by the number of draws.`

   Order parameters first based on absolute distance from zero then by interval width ie the parameters with point estimates far from zero and/or short intervals should appear higher in the table.

   Table 2:
   In different rows, report the quartile values from:

   1. `Raw outcome data.`
   2. `Prior predictive distribution.`
   3. `Posterior predictive distribution.`

   Table 3: Same format as table 1, but for all non predictor parameters eg the hyper- priors

   Print any warnings given by Stan during the fit.

- The final output should be knit quarto pdf.
- Use tools like `r-targets`, `target markdown`, and `stan target`.
- Note that our HPC uses slurm.

Ask any clarifying questions before starting.

Follow-up Prompt

1. No grouping variable, but hyper-priors like this:

$$
\begin{aligned}
\mu_i &= X_i \beta \\
\sigma_i &= a \cdot \mu_i^b \\
\beta &\sim \mathcal{N}(0, 100) \\
a &\sim \text{Gamma}(c, d) \\
b &\sim \mathcal{N}(0, 10)
\end{aligned}
$$

We would like to allow for over dispersion.

2.

- The output document should have a distinct section for each dataframe. Within a given dataframes section report Table 1 - 3 per likelihood. Group columns under each prior scenario.
- Change table 2 to be 3 columns per prior scenario under this grouping scheme.
- Order the table based on the most disperse prior ie N(0, 100).
- The non-predictor parameters are going to be stuff like a, b, c, d in point 1.

3.

- I plan to use rstan.
- I prefer .sbatch script and slurm job arrays.
- Don't worry about specific MCMC parameters at this point. We need to confirm the models fit before a long run.

4.

- Draft _targets.R from scratch.
- I want a single report.

5.

- We have 3 csv files: df_1.csv, df_2.csv, and df_3.csv
- Data is already clean.

6.

- Yes. Put all result in a single qmd.
- I like to use kableExtra, but I can handle formatting. Focus on create the appropriate dataframe for each table.