Step 1: One Batter (No Hierarchy)

Quesion: we have one batter who saw 100 pitches and swung 45 times. We want to estimate his true swing probability.

$$y \sim \text{Binomial}(n, \theta)$$

$$\theta \sim \text{Beta}(2, 2)$$

- y is the number of swings(45) out of n = 100 pitches.
- The prior Beta(2,2) means we start with a mild belief that swing rate is $2/2+2$ around 0.5 but not extreme.
- After seeing data, the posterior becomes more confident near 0.45.

Hint

We place a Bayesian hierarchical layer on the swing probability so that players with few observed pitches are informed by the overall league trend, while players with abundant data can express their unique swing tendencies. This balances stability and individuality, yielding more reliable player-level swing probabilities.

In MLB (and most sports datasets), data are hierarchical and unbalanced:

Some batters face thousands of pitches. Others — rookies, pinch hitters, injured players — may have only a few dozen.

If you estimate each batter's swing rate independently (e.g. raw mean or separate regression), the low-sample players will have extremely noisy estimates. If you instead pool all players together (one global model), you'll ignore meaningful individual differences.

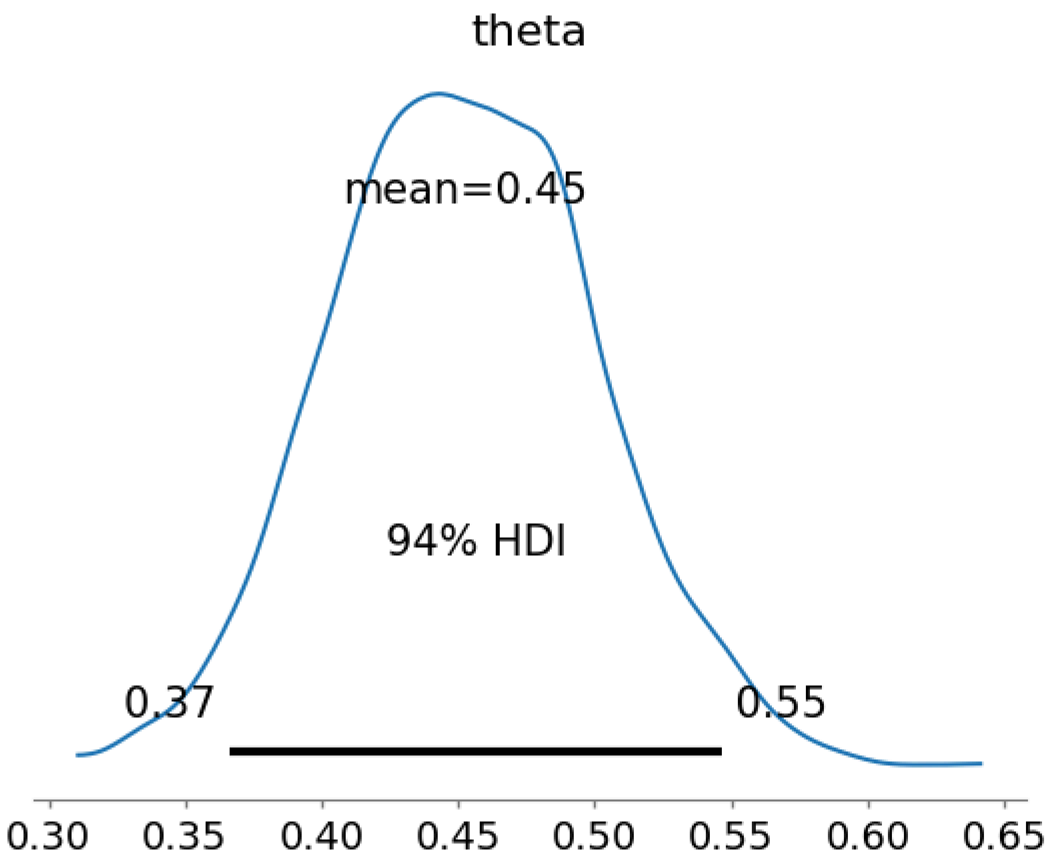
In [89]:
```
import pymc as pm
import arviz as az

n = 100
y = 45

with pm.Model() as model_simple:
    theta = pm.Beta("theta", alpha=2, beta=2)
    y_obs = pm.Binomial("y_obs", n=n, p=theta, observed=y)
    trace = pm.sample(1000, tune=1000, random_seed=42)

az.plot_posterior(trace)
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [theta]
```

```
Output()
```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 dr
aws total) took 1 seconds.
```

```
Out[89]: <Axes: title={'center': 'theta'}>
```



Interpretation

- The posterior mean is near 0.45.
- The Beta prior smooths extreme cases (e.g., if n were small).
- You can read the 94% credible interval to understand uncertainty in swing probability.

Step 2: Multiple Batters (Bayesian Hierarchical)

Question: We now have three batters, each with a different number of observed pitches:

Batter	n (pitches)	y (swings)
A	50	20
B	200	110
C	30	5

We want to estimate each batter’s true swing probability, allowing them to “share information.” That’s exactly what a Bayesian hierarchical model does.

$$y_i \sim \text{Binomial}(n_i, \theta_i)$$

$$\text{logit}(\theta_i) = \mu + \alpha_i$$

$$\alpha_i \sim \text{Normal}(0, \sigma)$$

$$\mu \sim \text{Normal}(0, 1), \quad \sigma \sim \text{HalfNormal}(1)$$

- Each batter i has their own swing rate θ_i.
- Instead of estimating each separately, we let them come from a common distribution centered at μ.
- Batters with few observations get pulled toward the group mean → this is partial pooling.

In [90]:
```python
import numpy as np
import pymc as pm
import arviz as az

n = np.array([50, 200, 30])
y = np.array([20, 110, 5])
n_batters = len(n)

with pm.Model() as model_hier:
    mu = pm.Normal("mu", 0, 1)
    sigma = pm.HalfNormal("sigma", 1)
    alpha = pm.Normal("alpha", 0, sigma, shape=n_batters)

    theta = pm.Deterministic("theta", pm.math.sigmoid(mu + alpha))
    y_obs = pm.Binomial("y_obs", n=n, p=theta, observed=y)

    trace = pm.sample(2000, tune=1000, random_seed=42)

az.summary(trace, var_names=["mu", "sigma", "theta"])
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, alpha]
Output()
```

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 dr
aws total) took 1 seconds.
```

Out[90]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
mu	-0.409	0.497	-1.426	0.466	0.013	0.010	1578.0	2213.0
sigma	0.931	0.428	0.271	1.772	0.008	0.006	2417.0	2470.0
theta[0]	0.400	0.066	0.280	0.529	0.001	0.001	10247.0	5397.0
theta[1]	0.544	0.035	0.479	0.610	0.000	0.000	8148.0	6590.0
theta[2]	0.220	0.078	0.083	0.363	0.001	0.001	4321.0	4464.0

Interpretation

- mu: overall league-wide swing tendency (in logit space).
- sigma: variation across batters.
- theta[i]: posterior swing probability for each batter i.
- Batter B (lots of data) → posterior near 110 / 200 = 0.55.
- Batter C (few pitches) → pulled toward the group mean instead of 0.17.

That's the key benefit of the hierarchical model.

Code Explanation

I want to describe more on the bayesian model context code in PyMC - everything indented inside this block defines the structure of our probabilistic model. I am very confused the first time I watch all those line of codes.

Try to think like, there are all random variables and their relationships within this model.

- mu = pm.Normal("mu", 0, 1), defines the global mean parameter(shared by all batters)
- sigma = pm.HalfNormal("sigma", 1), defines the spread of batter effects.
- alpha = pm.Normal("alpha", 0, sigma, shape=n_batters), defines a random effect for each batter.
- theta = pm.Deterministic("theta", pm.math.sigmoid(mu + alpha)), defines log-odds convert probability.

Step 3: Add Game Context — The Two-Strike Effect

Question: Given the situation batters swing more than with two strikes. We want to know each batter's baseline swing tendency, and the shared league-wide "two-strike bump"

$$y_{ij} \sim \text{Bernoulli}(p_{ij})$$

$$\text{logit}(p_{ij}) = \mu + \alpha_i + \beta \cdot TS_{ij}$$

$$\alpha_i \sim \text{Normal}(0, \sigma)$$

$$\mu \sim \text{Normal}(0,1), \quad \sigma \sim \text{HalfNormal}(1), \quad \beta \sim \text{Normal}(0,1)$$

- $y_{ij} = 1$ if batter i swings at pitch j
- $TS_{ij} = 1$ if the count has two strikes, else 0
- α_i captures batter-specific baseline swing tendency
- β captures the average league-wide increase in swing probability when there are two strikes

In [91]:
```
import numpy as np
import pymc as pm
import arviz as az

np.random.seed(42)

# Simulated data
n_batters = 3
pitches_per_batter = 100
batter_idx = np.repeat(np.arange(n_batters), pitches_per_batter)
two_strike = np.random.binomial(1, 0.3, size=n_batters * pitches_per_batter)

# True parameters (for simulation)
mu_true = -0.5          # average swing log-odds
sigma_true = 0.8        # batter variation
beta_true = 1.2         # two-strike effect
alpha_true = np.random.normal(0, sigma_true, n_batters)

# Simulate swing outcomes
logit_p = mu_true + alpha_true[batter_idx] + beta_true * two_strike
p = 1 / (1 + np.exp(-logit_p))
y = np.random.binomial(1, p)
```

In [92]:
```
coords = {"batter": np.arange(n_batters)}

with pm.Model(coords = coords) as model_context:
    mu = pm.Normal("mu", 0, 1)
    sigma = pm.HalfNormal("sigma",1)
    alpha = pm.Normal("alpha", 0, sigma, dims = "batter")
    beta = pm.Normal("beta", 0, 1)

    logit_p = mu_true + alpha[batter_idx] + beta * two_strike
    p = pm.math.sigmoid(logit_p)

    y_obs = pm.Bernoulli("y_obs", p, observed=y)
    trace = pm.sample(2000, tune=1000, random_seed=42)

az.summary(trace, var_names=["mu", "sigma", "beta", "alpha"])
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, alpha, beta]
```

```
Output()
```

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 dr
aws total) took 1 seconds.
There was 1 divergence after tuning. Increase `target_accept` or reparameter
ize.
```

Out[92]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
mu	-0.004	1.008	-1.865	1.882	0.010	0.011	9653.0	6026.0
sigma	1.087	0.395	0.470	1.828	0.004	0.005	7933.0	6128.0
beta	0.720	0.271	0.223	1.238	0.003	0.002	6292.0	6572.0
alpha[0]	0.156	0.216	-0.257	0.549	0.002	0.002	8189.0	6609.0
alpha[1]	-0.250	0.222	-0.646	0.191	0.002	0.002	8116.0	7232.0
alpha[2]	1.658	0.265	1.183	2.168	0.003	0.003	6848.0	5842.0

Step4: Add Pitch-Level Features(distance,balls,strikes,two-strike)

- Keep a batter random intercept(partial pooling)
- Add fixed effects for distance from the zone edge, balls, strikes, and a two-strike indicator.
- subset pitches with dist > 1, and define outside_i = dist_i - 1, defines how outside the zone the pitch is

Data restriction and feature:
$\text{Use only pitches with } \text{dist}_i > 1, \quad \text{and define } \text{outside}_i = \text{dist}_i - 1.$

Likelihood and linear predictor: $y_i \sim \text{Bernoulli}(p_i), \qquad \text{logit}(p_i) = \eta_i$

$$\eta_i = \mu + a_{\text{batter}[i]} + \beta_{\text{out}}\ \text{outside}_i + \beta_{\text{balls}}\ \text{balls}_i + \beta_{\text{strikes}}\ \text{strikes}_i + \beta_{\text{TS}}\ TS_i$$

Hierarchical prior and hyperpriors:
$a_b \sim \mathcal{N}(0, \sigma_a), \qquad \mu \sim \mathcal{N}(0, 1.5), \quad \sigma_a \sim \text{HalfNormal}(1)$

Coefficients: $\beta_{\text{out}}, \beta_{\text{balls}}, \beta_{\text{strikes}}, \beta_{\text{TS}} \sim \mathcal{N}(0, 1)$

- y_i = 1 if the batter swung at pitch i.
- TS_i if it's a two-strike count(else 0).
- $alpha_{batter[i]}$ is the batter random intercept.

In [93]:

```
# model_df
# Prepare real data (swing, dist, balls, strikes, batter_id) ---
# !pip install pybaseball pymc arviz numpy pandas  # (uncomment if needed)
```

```python
import numpy as np
import pandas as pd
import pymc as pm
import arviz as az

# ----------
# 0) Data: pull real Statcast pitches via pybaseball
# ----------
from pybaseball import statcast  # docs show statcast(start_dt, end_dt) → Da

def get_statcast_pitch_data(start_dt: str, end_dt: str, team: str | None = N
    """
    Pull pitch-by-pitch Statcast and engineer features to match your synthet
      - swing (0/1)
      - dist (distance from zone center, approximated)
      - balls, strikes (counts)
      - batter, pitcher (encoded ints)
      - park (encode home_team as proxy for park)
    Optionally filter to games involving a given team (e.g., 'LAD').
    """
    raw = statcast(start_dt=start_dt, end_dt=end_dt)

    if team:
        raw = raw[(raw["home_team"] == team) | (raw["away_team"] == team)]

    # Keep only needed columns (present in Statcast export)
    keep_cols = [
        "batter", "pitcher", "home_team", "away_team",
        "balls", "strikes", "plate_x", "plate_z",
        "type", "description", "game_date"
    ]
    df = raw.loc[:, [c for c in keep_cols if c in raw.columns]].copy()

    # Robustness: drop rows without location
    df = df.dropna(subset=["plate_x", "plate_z"])

    # Swing indicator:
    #  - type 'X' (ball in play) and type 'S' (any strike) generally imply a
    #  - use description to refine (foul, swinging_strike, foul_tip, hit_int
    swing_descriptions = {
        "swinging_strike", "swinging_strike_blocked",
        "foul", "foul_tip", "foul_bunt", "foul_pitchout",
        "hit_into_play", "hit_into_play_no_out", "hit_into_play_score"
    }
    df["swing"] = np.where(
        (df["type"].isin(["X"])) |
        ((df["type"] == "S") & (df["description"].isin(swing_descriptions)))
        (df["description"].isin(swing_descriptions)),
        1, 0
    )

    # Approx distance from center of zone (same scaling you used)
    df["dist"] = np.sqrt((df["plate_x"] / 0.7) ** 2 + (df["plate_z"] / 1.0)

    # Counts → ints
    df["balls"] = df["balls"].fillna(0).astype(int)
    df["strikes"] = df["strikes"].fillna(0).astype(int)

    # Encode park (proxy using home team categorical code for the game)
    # df["park"] = pd.factorize(df["home_team"])[0]

    # Re-encode batter/pitcher to dense 0..K-1 (PyMC coords-friendly)
    batter_codes, batter_uniques = pd.factorize(df["batter"])
    pitcher_codes, pitcher_uniques = pd.factorize(df["pitcher"])
    df["batter_id"] = batter_codes
    df["pitcher_id"] = pitcher_codes

    model_df = df[["swing", "dist", "balls", "strikes", "batter_id"]].copy()
    return model_df, df, batter_uniques, pitcher_uniques

# Example: one month of Dodgers games in July 2025
model_df, raw_df, batter_uniques, pitcher_uniques = get_statcast_pitch_data(
    start_dt="2025-06-01", end_dt="2025-07-01", team="LAD"
)
print(model_df.head(), model_df.shape)
```

```
This is a large query, it may take a moment to complete
```

```
  0%|                                                      | 0/31 [00:00<?,
?it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
  3%|█                                            | 1/31 [00:02<01:06,  2.2
1s/it]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 10%|████                                         | 3/31 [00:02<00:19,  1.4
7it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 13%|██████                                       | 4/31 [00:02<00:14,  1.8
2it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 16%|███████                                      | 5/31 [00:02<00:10,  2.4
3it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 19%|█████████                                    | 6/31 [00:03<00:08,  3.0
0it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 23%|██████████                                   | 7/31 [00:03<00:08,  2.9
4it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
```

```
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 26%|███████████                                  | 8/31 [00:04<00:11,  2.0
5it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
```

```
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
```

```
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 42%|██████████████████                          | 13/31 [00:04<00:03,  5.0
3it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 45%|████████████████████                         | 14/31 [00:05<00:05,  2.8
4it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 55%|████████████████████████                     | 17/31 [00:06<00:04,  3.3
1it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 58%|██████████████████████████                   | 18/31 [00:06<00:03,  3.6
3it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
100%|█████████████████████████████████████████████| 31/31 [00:07<00:00,  4.0
4it/s]
      swing      dist  balls  strikes  batter_id
2068      0  2.486424      0        2          0
2146      1  2.415738      0        1          0
2214      0  2.535801      0        0          0
2260      1  3.979268      3        2          1
2355      1  1.283328      3        2          1 (8337, 5)
```

In [94]:
```python
model_df.head(10)
```

Out[94]:

	swing	dist	balls	strikes	batter_id
2068	0	2.486424	0	2	0
2146	1	2.415738	0	1	0
2214	0	2.535801	0	0	0
2260	1	3.979268	3	2	1
2355	1	1.283328	3	2	1
2428	0	4.04061	2	2	1
2513	0	3.041155	1	2	1
2592	0	3.335189	0	2	1
2682	1	3.652264	0	1	1
2771	0	2.530913	0	0	1

In [95]:
```python
import numpy as np, pandas as pd
import pymc as pm, arviz as az


# --- Prepare data (assumes df has: swing, dist, balls, strikes, batter_id)
```

```
df = model_df.copy()  # or your DataFrame name
df = df.loc[df["dist"] > 1].copy()
df["outside"] = df["dist"] - 1.0
df["TS"] = (df["strikes"] == 2).astype(int)


df["swing"]   = pd.to_numeric(df["swing"], errors="coerce")
df["balls"]   = pd.to_numeric(df["balls"], errors="coerce")
df["strikes"] = pd.to_numeric(df["strikes"], errors="coerce")
df["dist"]    = pd.to_numeric(df["dist"], errors="coerce")

df = df.dropna(subset=["swing", "balls", "strikes", "dist"])

# Factorize batter ids -> integer index 0..B-1
b_idx, b_unique = pd.factorize(df["batter_id"])
B = len(b_unique)

y = df["swing"].astype(int).values
outside = df["outside"].astype(int).values
balls = df["balls"].astype(int).values
strikes = df["strikes"].astype(int).values
TS = df["TS"].values

coords = {"batter": np.arange(B), "pitch": np.arange(len(y))}

with pm.Model(coords=coords) as mdl:
    # Hyperpriors
    mu = pm.Normal("mu", 0.0, 1.5)
    sigma_a = pm.HalfNormal("sigma_a", 1.0)

    # Batter random intercept
    a_batter = pm.Normal("a_batter", 0.0, sigma_a, dims="batter")

    # Fixed effects
    beta_out     = pm.Normal("beta_out", 0.0, 1.0)
    beta_balls   = pm.Normal("beta_balls", 0.0, 1.0)
    beta_strikes = pm.Normal("beta_strikes", 0.0, 1.0)
    beta_TS      = pm.Normal("beta_TS", 0.0, 1.0)

    # Linear predictor
    eta = (mu
           + a_batter[b_idx]
           + beta_out * outside
           + beta_balls * balls
           + beta_strikes * strikes
           + beta_TS * TS)

    p = pm.math.sigmoid(eta)

    # Likelihood
    y_obs = pm.Bernoulli("y_obs", p, observed=y)

    idata = pm.sample(2000, tune=1500, chains=4, target_accept=0.9, random_s

# --- Inspect results ---
```

```
az.summary(idata, var_names=["mu","sigma_a","beta_out","beta_balls","beta_st
# az.plot_posterior(idata, var_names=["beta_out","beta_balls","beta_strikes"
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma_a, a_batter, beta_out, beta_balls, beta_strikes, beta_TS]
Output()
```

```
Sampling 4 chains for 1_500 tune and 2_000 draw iterations (6_000 + 8_000 dr
aws total) took 36 seconds.
```

Out[95]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess
mu	-0.254	0.056	-0.360	-0.148	0.001	0.001	4878.0	57
sigma_a	0.202	0.035	0.133	0.266	0.001	0.001	825.0	13
beta_out	-0.386	0.027	-0.439	-0.336	0.000	0.000	7384.0	56
beta_balls	0.211	0.026	0.162	0.261	0.000	0.000	8135.0	58
beta_strikes	0.674	0.057	0.569	0.784	0.001	0.001	6067.0	60
...	...	...	...	...	...	...	...	
a_batter[120]	-0.088	0.201	-0.486	0.274	0.002	0.003	9914.0	50
a_batter[121]	-0.039	0.185	-0.373	0.328	0.002	0.002	10368.0	53
a_batter[122]	0.036	0.187	-0.309	0.401	0.002	0.002	12185.0	58
a_batter[123]	-0.185	0.194	-0.544	0.186	0.003	0.003	6199.0	46
a_batter[124]	-0.001	0.193	-0.368	0.358	0.002	0.003	10984.0	56

131 rows × 9 columns

Step 5 Home Run Rate

Furthermore, given that the batter already decided to swing, what is the probability of a successful outcome (in your case, a home run) for all batters in the dataset, considering game context(balls, strikes, dist), pitcher effect(pitcher_id)

In [96]:

```
# Subset swing = 1
# model_df
# Prepare real data (swing, dist, balls, strikes, batter_id) ---
# !pip install pybaseball pymc arviz numpy pandas  # (uncomment if needed)

import numpy as np
import pandas as pd
import pymc as pm
import arviz as az

# ---------
```

```python
# 0) Data: pull real Statcast pitches via pybaseball
# ----------
from pybaseball import statcast  # docs show statcast(start_dt, end_dt) → Da

def get_statcast_pitch_data(start_dt: str, end_dt: str, team: str | None = N
    """
    Pull pitch-by-pitch Statcast and engineer features to match your synthet
      - swing (1)
      - dist (distance from zone center, approximated)
      - balls, strikes (counts)
      - batter, pitcher (encoded ints)
    Optionally filter to games involving a given team (e.g., 'LAD').
    """
    raw = statcast(start_dt=start_dt, end_dt=end_dt)

    if team:
        raw = raw[(raw["home_team"] == team) | (raw["away_team"] == team)]

    # Keep only needed columns (present in Statcast export)
    keep_cols = [
        "batter", "pitcher", "home_team", "away_team",
        "balls", "strikes", "plate_x", "plate_z",
        "type", "description", "game_date", "events"
    ]
    df = raw.loc[:, [c for c in keep_cols if c in raw.columns]].copy()

    # Robustness: drop rows without location
    df = df.dropna(subset=["plate_x", "plate_z"])

    # Swing indicator:
    #   - type 'X' (ball in play) and type 'S' (any strike) generally imply a
    #   - use description to refine (foul, swinging_strike, foul_tip, hit_int
    swing_descriptions = {
        "swinging_strike", "swinging_strike_blocked",
        "foul", "foul_tip", "foul_bunt", "foul_pitchout",
        "hit_into_play", "hit_into_play_no_out", "hit_into_play_score"
    }
    df["swing"] = np.where(
        (df["type"].isin(["X"])) |
        ((df["type"] == "S") & (df["description"].isin(swing_descriptions)))
        (df["description"].isin(swing_descriptions)),
        1, 0
    )
    # Filter Swing
    df = df[df["swing"] == 1].copy()

    # Approx distance from center of zone (same scaling you used)
    df["dist"] = np.sqrt((df["plate_x"] / 0.7) ** 2 + (df["plate_z"] / 1.0)

    # Counts → ints
    df["balls"] = df["balls"].fillna(0).astype(int)
    df["strikes"] = df["strikes"].fillna(0).astype(int)

    # Encode park (proxy using home team categorical code for the game)
    # df["park"] = pd.factorize(df["home_team"])[0]
    # Re-encode batter/pitcher to dense 0..K-1 (PyMC coords-friendly)
    batter_codes, batter_uniques = pd.factorize(df["batter"])
    pitcher_codes, pitcher_uniques = pd.factorize(df["pitcher"])
    df["batter_id"] = batter_codes
    df["pitcher_id"] = pitcher_codes

    df["is_home_run"] = (df["events"] == "home_run").astype(int)

    model_df = df[["is_home_run","batter_id", "pitcher_id", "dist", "balls",
    return model_df, df, batter_uniques, pitcher_uniques

# Example: one month of Dodgers games in July 2025
model_df, raw_df, batter_uniques, pitcher_uniques = get_statcast_pitch_data(
    start_dt="2025-06-01", end_dt="2025-07-01", team="LAD"
)
print(model_df.head(), model_df.shape)
```

```
This is a large query, it may take a moment to complete
```

```
  0%|                                                      | 0/31 [00:00<?, 
?it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors` 
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
  3%|█▍                                            | 1/31 [00:02<01:09,  2.3
1s/it]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors` 
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
  6%|██▉                                           | 2/31 [00:02<00:34,  1.1
8s/it]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors` 
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 13%|█████▉                                        | 4/31 [00:02<00:13,  2.0
1it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors` 
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 16%|███████▍                                      | 5/31 [00:03<00:10,  2.5
2it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
and will raise in a future version. Use to_datetime without passing `errors` 
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 19%|████████▉                                     | 6/31 [00:03<00:09,  2.7
2it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated 
```

```
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 26%|███████████                                  | 8/31 [00:03<00:05,  3.9
9it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 29%|█████████████                                | 9/31 [00:04<00:06,  3.1
5it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
```

```
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
```

```
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 32%|██████████                                | 10/31 [00:04<00:06,  3.0
6it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 35%|███████████                               | 11/31 [00:05<00:08,  2.4
6it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 39%|████████████                              | 12/31 [00:05<00:07,  2.5
1it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
```

```
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 52%|███████████████████████                      | 16/31 [00:06<00:03,  4.4
4it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
 71%|████████████████████████████████             | 22/31 [00:06<00:01,  8.8
8it/s]/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/dat
ahelpers/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated
and will raise in a future version. Use to_datetime without passing `errors`
and catch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
/Users/brycelee/miniconda3/lib/python3.11/site-packages/pybaseball/datahelpe
rs/postprocessing.py:59: FutureWarning: errors='ignore' is deprecated and wi
ll raise in a future version. Use to_datetime without passing `errors` and c
atch exceptions explicitly instead
  data_copy[column] = data_copy[column].apply(pd.to_datetime, errors='ignor
e', format=date_format)
100%|██████████████████████████████████████████████| 31/31 [00:07<00:00,  4.1
4it/s]
      is_home_run  batter_id  pitcher_id      dist  balls  strikes
2146            0          0           0  2.415738      0        1
2260            0          1           0  3.979268      3        2
2355            0          1           0  1.283328      3        2
2682            0          1           0  3.652264      0        1
2839            0          2           0  3.626534      3        2 (3990, 6)
```

In [97]:
```
model_df
```

Out[97]:

	is_home_run	batter_id	pitcher_id	dist	balls	strikes
2146	0	0	0	2.415738	0	1
2260	0	1	0	3.979268	3	2
2355	0	1	0	1.283328	3	2
2682	0	1	0	3.652264	0	1
2839	0	2	0	3.626534	3	2
...	...	...	...	...	...	...
2982	0	121	4	3.9747	0	0
3143	0	122	4	3.243024	3	2
3840	0	115	4	1.561321	1	2
3890	0	115	4	3.398286	1	1
4125	0	115	4	1.974989	0	0

3990 rows × 6 columns

Write Equation

Likelihood and linear predictor: $y_i \sim \text{Bernoulli}(p_i), \qquad \text{logit}(p_i) = \eta_i$

$\eta_i = \mu + a_{\text{batter}[i]} + a_{\text{pitcher}[i]} + \beta_{\text{balls}}\ \text{balls}_i + \beta_{\text{strikes}}\ \text{strikes}_i + +\beta_{\text{dist}}\ \text{dist}_i$

$$a_{\text{batter}} \sim \mathcal{N}(0, \sigma_{\text{batter}}), \quad a_{\text{pitcher}} \sim \mathcal{N}(0, \sigma_{\text{pitcher}})$$

$$\mu \sim \mathcal{N}(\text{logit}(0.04),\ 1), \quad \sigma_{\text{batter}}, \sigma_{\text{pitcher}} \sim \text{HalfNormal}(1)$$

$$\beta_{\text{balls}},\ \beta_{\text{strikes}},\ \beta_{\text{dist}} \sim \mathcal{N}(0,1)$$

In [98]:
```
import numpy as np, pandas as pd
import pymc as pm, arviz as az


# --- Prepare data (assumes df has: swing, dist, balls, strikes, batter_id)
df = model_df.copy()  # or your DataFrame name

df["balls"]   = pd.to_numeric(df["balls"], errors="coerce")
df["strikes"] = pd.to_numeric(df["strikes"], errors="coerce")
df["dist"]    = pd.to_numeric(df["dist"], errors="coerce")
# standard distance
dist_mean, dist_std = dist.mean(), dist.std() if dist.std() > 0 else 1.0
dist_sc = (dist - dist_mean) / dist_std


df = df.dropna(subset=["balls", "strikes", "dist"])

# Factorize batter ids -> integer index 0..B-1
b_idx, b_unique = pd.factorize(df["batter_id"])
B = len(b_unique)

p_idx, p_unique = pd.factorize(df["pitcher_id"])
P = len(p_unique)

y = df["is_home_run"].astype(int).values
dist = df["dist"].astype(float).values
balls = df["balls"].astype(int).values
strikes = df["strikes"].astype(int).values

coords = {"batter": np.arange(B), "pitcher": np.arange(P)}

with pm.Model(coords=coords) as mdl:
    # Hyperpriors
    mu = pm.Normal("mu", pm.math.log(0.04/(1-0.04)), 1.0)
    sigma_batter = pm.HalfNormal("sigma_batter", 1.0)
    sigma_pitcher = pm.HalfNormal("sigma_pitcher", 1.0)

    # Batter random intercept
    a_batter = pm.Normal("a_batter", 0.0, sigma_batter, dims="batter")
    a_pitcher = pm.Normal("a_pitcher", 0.0, sigma_pitcher, dims="pitcher")


    # Fixed effects
    beta_balls   = pm.Normal("beta_balls", 0.0, 1.0)
    beta_strikes = pm.Normal("beta_strikes", 0.0, 1.0)
    beta_dist = pm.Normal("beta_dist", 0.0, 1.0)

    # Linear predictor
    eta = (mu
           + a_batter[b_idx]
           + a_pitcher[p_idx]
           + beta_balls * balls
           + beta_strikes * strikes
           + beta_dist * dist_sc
```

```
            )
    p = pm.math.sigmoid(eta)

    # Likelihood
    y_obs = pm.Bernoulli("y_obs", p, observed=y)

    idata = pm.sample(2000, tune=1500, chains=4, target_accept=0.95, random_

# --- Inspect results ---
az.summary(idata, var_names=["mu","sigma_batter","sigma_pitcher", "a_batter"
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma_batter, sigma_pitcher, a_batter, a_pitcher, beta_balls, bet
a_strikes, beta_dist]
Output()
```

```
Sampling 4 chains for 1_500 tune and 2_000 draw iterations (6_000 + 8_000 dr
aws total) took 61 seconds.
There was 1 divergence after tuning. Increase `target_accept` or reparameter
ize.
The rhat statistic is larger than 1.01 for some parameters. This indicates p
roblems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.
A higher number is needed for reliable rhat and ess computation. See http
s://arxiv.org/abs/1903.08008 for details
```

Out[98]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk
mu	-4.072	0.247	-4.545	-3.623	0.017	0.004	202.0
sigma_batter	0.518	0.239	0.072	0.931	0.037	0.010	36.0
sigma_pitcher	0.297	0.186	0.024	0.630	0.039	0.013	21.0
a_batter[0]	-0.045	0.556	-1.130	1.061	0.005	0.024	11936.0
a_batter[1]	-0.059	0.543	-1.096	1.034	0.005	0.019	14334.0
...	...	...	...	...	...	...	...
a_pitcher[107]	-0.027	0.343	-0.715	0.671	0.004	0.032	9247.0
a_pitcher[108]	0.016	0.319	-0.598	0.690	0.003	0.028	9795.0
beta_balls	0.205	0.133	-0.045	0.451	0.001	0.002	8886.0
beta_strikes	-0.375	0.164	-0.684	-0.066	0.002	0.002	8785.0
beta_dist	-0.026	0.126	-0.265	0.206	0.001	0.002	9254.0

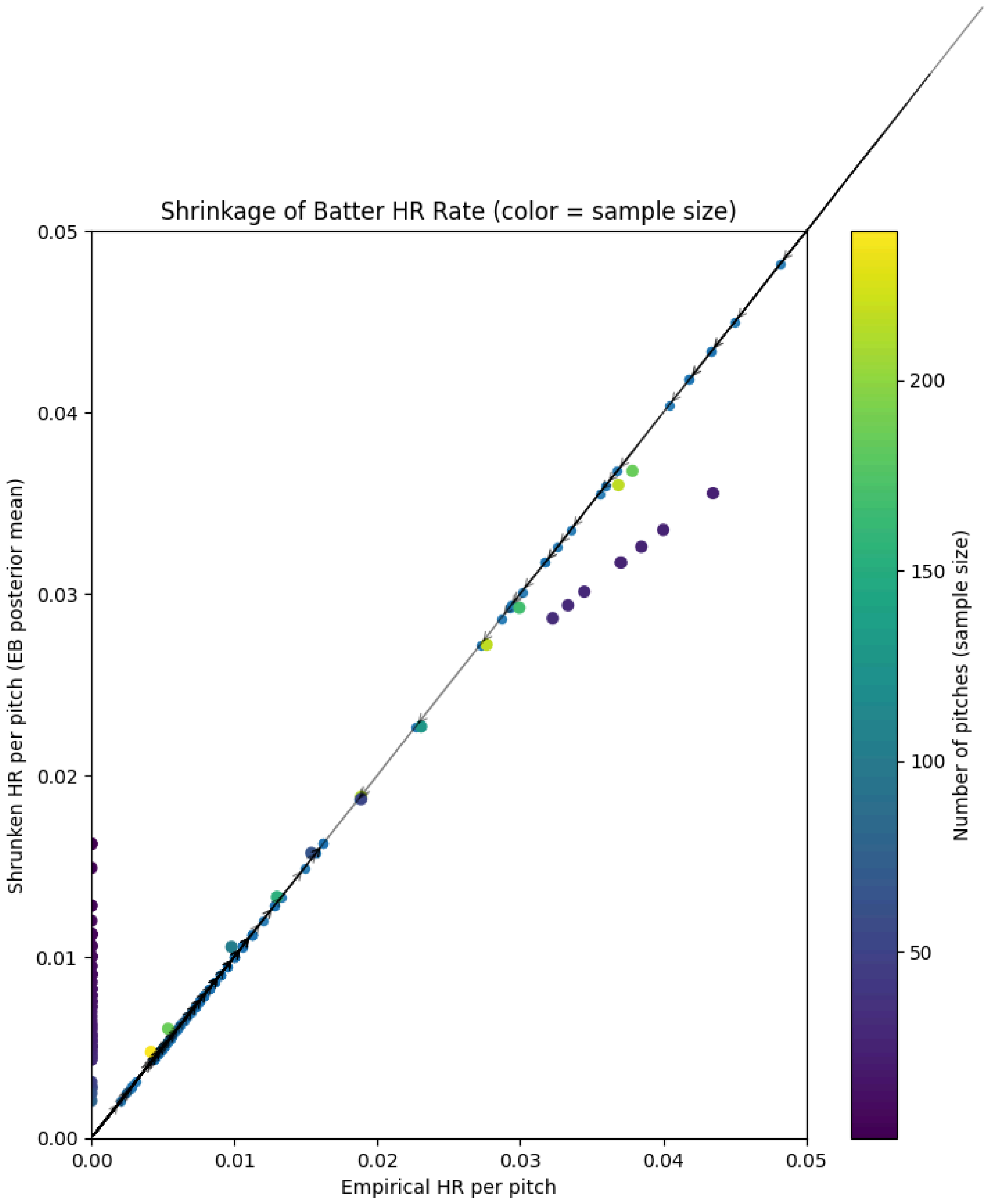
239 rows × 9 columns

In [99]:

```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import beta
```

```
# ------------------------------
# 0) Start from UNFILTERED data
# ------------------------------
raw_df = model_df.copy()   # <- use your full dataset here (no swing filter)

# Make sure these are numeric
raw_df["is_home_run"] = pd.to_numeric(raw_df.get("is_home_run"), errors="coe

# ------------------------------
# 1) Choose denominator (priority: PA -> AB -> pitches)
# ------------------------------
# Try to build a plate-appearance (PA) id:
# Many Statcast exports have (game_pk, at_bat_number) or (game_year, at_bat_
# We'll try the common pair (game_pk, at_bat_number); adapt if your columns
if {"game_pk","at_bat_number"}.issubset(raw_df.columns):
    pa = (raw_df
          .dropna(subset=["game_pk","at_bat_number","batter_id","is_home_rur
          .groupby(["game_pk","at_bat_number","batter_id"], as_index=False)
          .agg(hr_in_pa=("is_home_run","max")))  # any HR in the PA -> HR
    denom = (pa.groupby("batter_id", as_index=False)
               .agg(hr=("hr_in_pa","sum"), opps=("hr_in_pa","size")))
    denom_name = "HR per PA"
elif {"game_year","at_bat_number"}.issubset(raw_df.columns):
    pa = (raw_df
          .dropna(subset=["game_year","at_bat_number","batter_id","is_home_r
          .groupby(["game_year","at_bat_number","batter_id"], as_index=False
          .agg(hr_in_ab=("is_home_run","max")))
    denom = (pa.groupby("batter_id", as_index=False)
               .agg(hr=("hr_in_ab","sum"), opps=("hr_in_ab","size")))
    denom_name = "HR per AB"
else:
    # fallback: per pitch (least preferred, but works everywhere)
    tmp = raw_df.dropna(subset=["batter_id","is_home_run"])
    denom = (tmp.groupby("batter_id", as_index=False)
               .agg(hr=("is_home_run","sum"), opps=("is_home_run","size")))
    denom_name = "HR per pitch"

denom = denom.loc[denom["opps"] > 0].copy()
denom["emp_rate"] = denom["hr"] / denom["opps"]

# ------------------------------
# 2) Empirical Bayes Beta prior (method of moments)
#    Estimate alpha,beta from pooled mean/var across batters
# ------------------------------
# Pooled estimates (weighted by opps)
tot_hr   = denom["hr"].sum()
tot_opps = denom["opps"].sum()
m = tot_hr / tot_opps  # pooled mean

# Approximate variance of rates using within-batter binomial variance
# (small-sample robust enough for EB)
var_hat = (denom["emp_rate"] * (1 - denom["emp_rate"]) / denom["opps"]).mear

# Solve for alpha,beta; guard against tiny/negative var
var_hat = max(var_hat, 1e-8)
k = m * (1 - m) / var_hat - 1
```

```
alpha0 = max(m * k, 1e-6)
beta0  = max((1 - m) * k, 1e-6)

# ------------------------------
# 3) Posterior shrinkage per batter
# ------------------------------
denom["post_mean"] = (denom["hr"] + alpha0) / (denom["opps"] + alpha0 + beta
# 95% credible interval from Beta posterior
denom["post_lo"] = beta.ppf(0.025, denom["hr"] + alpha0, denom["opps"] - der
denom["post_hi"] = beta.ppf(0.975, denom["hr"] + alpha0, denom["opps"] - der

# ------------------------------
# 4) Shrinkage plots
# ------------------------------
# (A) Diagonal arrows: empirical -> shrunken
plt.figure(figsize=(7.5,7.5))
lims = [0, float(max(0.12, denom[["emp_rate","post_mean"]].to_numpy().max()*
plt.plot(lims, lims, lw=1, alpha=0.5, color="k")  # no-shrinkage line

for _, r in denom.iterrows():
    plt.annotate("", xy=(r["post_mean"], r["post_mean"]),
                 xytext=(r["emp_rate"], r["emp_rate"]),
                 arrowprops=dict(arrowstyle="->", lw=0.8, alpha=0.5))

plt.scatter(denom["post_mean"], denom["post_mean"], s=18, alpha=0.9)
plt.xlabel(f"Empirical {denom_name}")
plt.ylabel(f"Shrunken {denom_name} (EB posterior mean)")
plt.title(f"Shrinkage of Batter {denom_name}")
plt.xlim(lims); plt.ylim(lims)
plt.tight_layout()
plt.xlim(0, 0.05)
plt.ylim(0, 0.05)
plt.scatter(denom["emp_rate"], denom["post_mean"],
            c=denom["opps"], cmap="viridis", s=30)
plt.colorbar(label="Number of pitches (sample size)")
plt.title("Shrinkage of Batter HR Rate (color = sample size)")
plt.show()
```



- These are batters who had only a few pitches but hit one HR, giving an inflated raw rate.

The Bayesian model recognizes the small sample and shrinks that rate toward the league average (~2–3% per pitch).

```
In [ ]:
```