MATH - 574
Bayesian Computational Statistics - Project Report

HIERARCHICAL BAYESIAN MODELING FOR CUSTOMER PURCHASE BEHAVIOR

Name	A#
Sailaxman Kotha	A20561652

May 9, 2025

Contents

Find code here

1. Introduction

Instacart, a grocery ordering and delivery app aims to make it easy to fill your refrigerator and pantry with your personal favorites and staples when you need them. After selecting products through the Instacart app, personal shoppers review your order and do the in-store shopping and delivery for you.

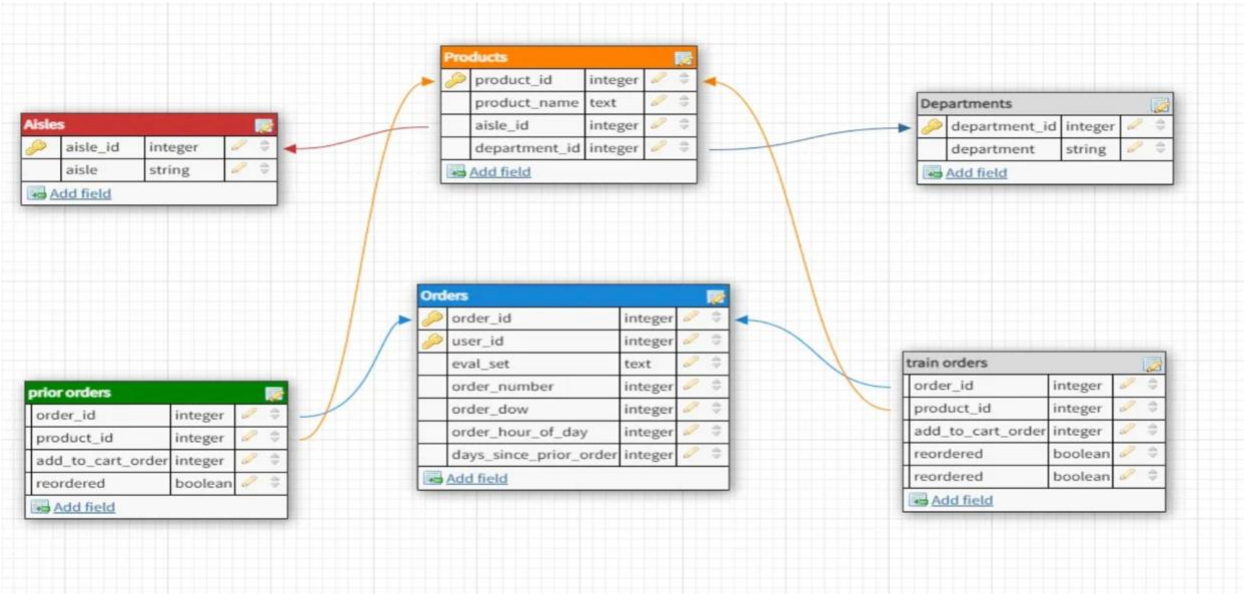
Motivation: Online grocery shopping data are notoriously sparse and skewed. Most customers place only a handful of orders and only buy a tiny fraction of the available catalog, while occasional power users or promotional events produce very large basket sizes. At the same time, business decisions like stocking popular items, staffing delivery slots, and personalizing recommendations demand not only accurate predictions of what and how much users will buy next, but also a clear sense of the uncertainty around those predictions. Traditional machine learning classifiers and regressors may achieve good average accuracy on dense segments, but they fail to adapt to ‘cold-start’ users or niche products and offer no principled way to quantify risk when planning inventory or marketing actions. To address these challenges, we adopt a hierarchical Bayesian logistic model for reorder probability, allowing user and product intercepts to ‘shrink’ toward the overall mean when data are limited and a Poisson Bayesian count regression framework for cart-size forecasting, which yields full predictive distributions instead of single-point estimates. This combination delivers robust, well-calibrated predictions across the long-tail of ecommerce behavior and equips Instacart with the credible intervals and tail-risk estimates.

2. Data Description and Problem Analysis

Instacart dataset contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, between 4 and 100 of their orders, which the sequence of products purchased in each order. The dataset also provides the week and hour of day the order was replaced, and a relative measure of time between orders.

The dataset comprises six tables.

- orders: Each row represents grocery orders namely prior, test and train.
- order_products_prior: This table includes prior orders. It indicates whether a product in an order is a reorder or not.
- order_products_train: Includes training orders.
- products: Includes all products and their related information.
- aisles: Includes all aisles and their related information.
- departments: Includes all departments and their related information.



Relational diagram of the dataset

~ For the Bayesian hierarchical logistic regression, the target variable is reorder which is to predict whether the user u_n reorders p_n on that order.

$$y_n \in \{0, 1\}, \quad y_n = 1 \text{ if user } u_n \text{ reorders product } p_n \text{ on that order}$$

For reorder, x_n = (order_number, order_dow, order_hour_of_day) plus user_idx and product_idx for hierarchical intercepts.

~ For the Bayesian poisson regression, the target variable is cart_size which is to predict the number of items in the final cart or the order.

$$y_i = \#\{\text{items in order } i\} \in \{0, 1, 2, \dots\}.$$

For cart size, the table aggregated to x_i = (order_number, order_dow, order_hour_of_day, y_{user})

Because order_number, order_dow, order_hour_of_day and avg_cart_size span different units, we standardized each column k so that over the training set.

$$\text{mean}\big(x_{1:N,k}\big) = 0, \quad \text{sd}\big(x_{1:N,k}\big) = 1.$$

Distributional assumptions for modeling

1. Reorder model:

$$y_n \sim \text{Bernoulli}\big(\sigma(\eta_n)\big), \quad \eta_n = \alpha + \gamma_{u_n} + \delta_{p_n} + \beta^\top \mathbf{x}_n.$$

2. Cart-size model:

$$y_i \sim \text{Poisson}\big(\lambda_i\big), \quad \log \lambda_i = \beta_0 + \beta^\top \mathbf{x}_i.$$

3. Model Description

Bayesian Hierarchical Logistic Regression

Bayesian Hierarchical Logistic Regression is a powerful statistical modeling technique particularly suited for predicting binary outcomes, such as whether a user will reorder a specific product. In our case, the data naturally exhibit a hierarchical structure, where individual reorder decisions are nested within users, and possibly also within product categories. Each user has unique purchasing habits, and products vary in their likelihood of being reordered, making it essential to model both global patterns and individual-level variability. This model addresses such complexity by incorporating fixed effects (e.g., average reorder probability based on product features or user behavior) and random effects (e.g., user-specific intercepts that capture individual reordering tendencies). The Bayesian framework further enhances the model by allowing the integration of prior knowledge—such as expected product reorder rates or behavioral assumptions about users—and produces posterior distributions for all parameters, providing credible intervals and full uncertainty quantification. This is especially valuable in our application, where capturing subtle behavioral patterns and accounting for uncertainty can

significantly improve reorder prediction accuracy and inform personalized recommendation strategies.

To predict whether a given user will reorder a particular product on a given order. Mathematically, for each data point n (an order-product pair), we observe

$$y_n = \begin{cases} 1 & \text{if the product was reordered} \\ 0 & \text{otherwise,} \end{cases}$$

Global Intercept → $\alpha \sim N(0, 1)$

User-level random intercepts → $u_{\text{raw},j} \sim N(0,1)$ for each user $j = 1, \ldots, U.$

Scale by $\sigma_u \sim \text{HalfNormal}(1)$ to obtain $\gamma_{u,j} = \sigma_u \, u_{\text{raw},j}.$

Allows each user to have their own baseline reorder propensity, but 'shrinks' very sparse users toward the population mean.

Product-level random intercepts → $p_{raw,k}$ ~ N(0,1) and $\delta_{p,k} = \sigma_p \, p_{raw,k}$ with σ_p ~ HalfNormal(1)

Fixed slopes → β_{num}, β_{dow}, β_{hour} ~ N(0,1) for predictors (order_number, order_dow, order_hour)

Linear predictor and likelihood →

$$\eta_n = \alpha + \gamma_{u_n} + \delta_{p_n} + \beta_{\text{num}} \, x_{n,1} + \beta_{\text{dow}} \, x_{n,2} + \beta_{\text{hour}} \, x_{n,3}$$

Pr(y_n = 1) = $\sigma(\eta_n)$ = $\frac{1}{1 + e^{-\eta_n}}$ and assume y_n ~ Bernoulli($\sigma(\eta_n)$).

If a user u_n is typically likely to reorder (large γ_{u_n}, and the product p_n is popular (large δ_{p_n}), then η_n will be high, leading to a high reorder probability.

Log-Loss

For a given target y_n and probabilistic prediction $\hat{y}_n$, the log loss is defined by

$$-\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{p}_n + (1 - y_n) \log(1 - \hat{p}_n) \right]$$

Where $\hat{p}_n$ is the posterior-mean probability.

y is the label. Since this is logistic regression, every value of y must either be 0 or 1.

$\hat{y}_n$ is the predicted value, given the set of features in x.

Log-loss penalizes extreme misconfidence most heavily.

Bayesian Poisson Regression :

Bayesian Poisson Regression is a suitable and effective modeling approach for predicting count-based outcomes, such as the number of items a user adds to their basket during a transaction. In this context, the response variable is a non-negative integer that often depends on various user-level and session-level features, including prior shopping behavior, product preferences, time of day, and promotional activity. The Poisson regression model captures the relationship between these predictors and the expected count of items, assuming that the counts follow a Poisson distribution. By adopting a Bayesian framework, the model not only estimates the underlying rate (or intensity) of item addition but also incorporates prior beliefs about user behavior and allows for uncertainty quantification in all model parameters. This probabilistic perspective is particularly valuable in our application, where shopping patterns can vary significantly across users and contexts. The resulting posterior distributions provide credible intervals for predictions, enabling more reliable insights into basket size and informing downstream tasks such as inventory planning, dynamic pricing, and personalized marketing.

For each order i, predict the cart size
The model counts with a poisson and log-link,

$$\eta_i = \beta_0 + \sum_{j=1}^{4} \beta_j x_{ij},$$
$$\lambda_i = \exp(\eta_i),$$
$$y_i \sim \text{Poisson}(\lambda_i).$$

Where β_j are the coefficients of order_number, day_of_week, hour_of_day and avg_cart_size.

By Bayes' rule, the joint distribution over all β is

$$p(\boldsymbol{\beta} \mid \{y_i, x_i\}) \propto \underbrace{\left[\prod_{j=0}^{4} p(\beta_j)\right]}_{\text{priors}} \times \underbrace{\prod_{i=1}^{N} p(y_i \mid \beta, x_i)}_{\text{likelihood}}$$

$$= \left[\prod_{j=0}^{4} \frac{1}{\sqrt{2\pi}} e^{-\beta_j^2/2}\right] \left[\prod_{i=1}^{N} \frac{e^{y_i \eta_i} e^{-e^{\eta_i}}}{y_i!}\right].$$

Posterior mean and expected cart for a new order, where S is samples drawn

$$\hat{\beta}_j = \tfrac{1}{S} \textstyle\sum_s \beta_j^{(s)}$$

$$\mathbb{E}[y \mid \mathbf{x}] = \mathbb{E}_{\boldsymbol{\beta}}\big[e^{\beta_0 + \sum \beta_j x_j}\big] \approx \frac{1}{S} \sum_{s=1}^{S} \exp\big(\beta_0^{(s)} + \sum_j \beta_j^{(s)}\, x_j\big)$$

Predictive Distribution

$$y^{\mathrm{rep},(s)} \;\sim\; \mathrm{Poisson}\Big(e^{\beta_0^{(s)} + \sum_j \beta_j^{(s)}\, x_j}\Big)$$

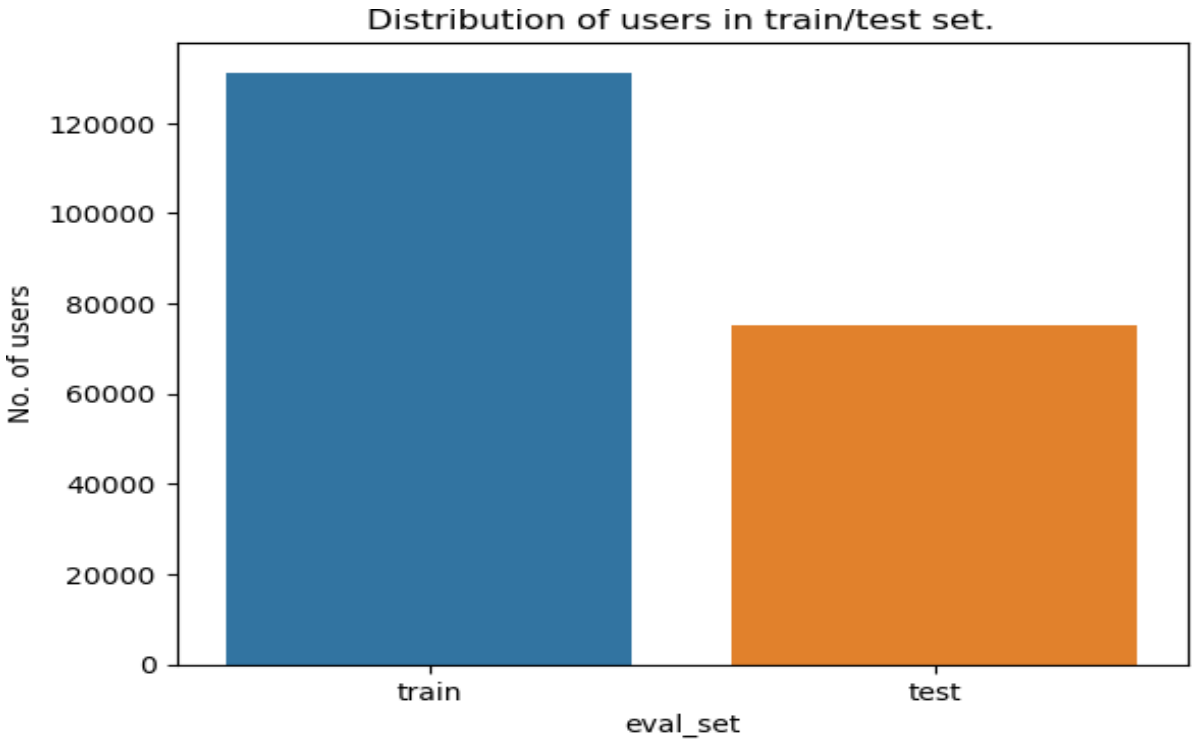
Root-Mean-Squared Error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \big(y_i - \hat{y}_i\big)^2}$$

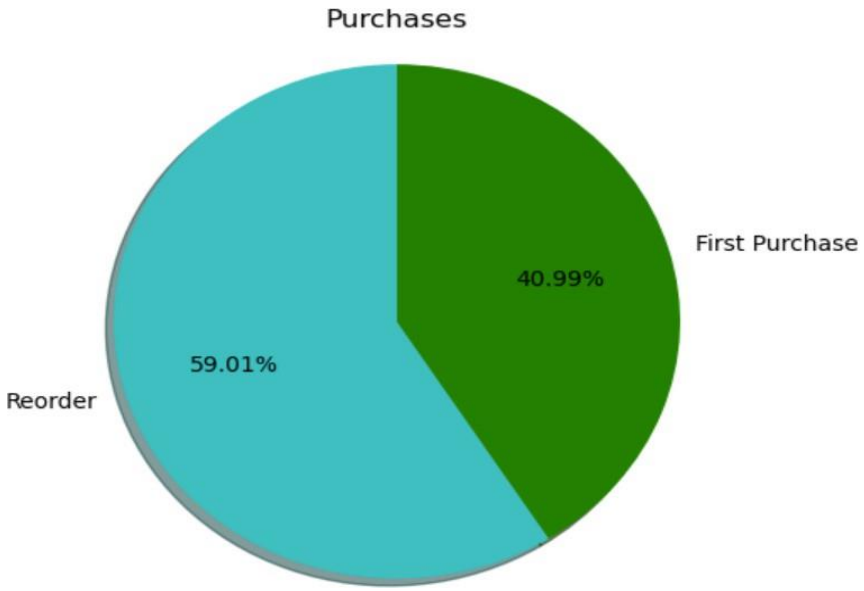
Mean Absolute Error (MAE)

$$\mathrm{MAE} = \frac{1}{N} \sum_{i=1}^{N} \big|y_i - \hat{y}_i\big|$$

4. Exploratory Data Analysis



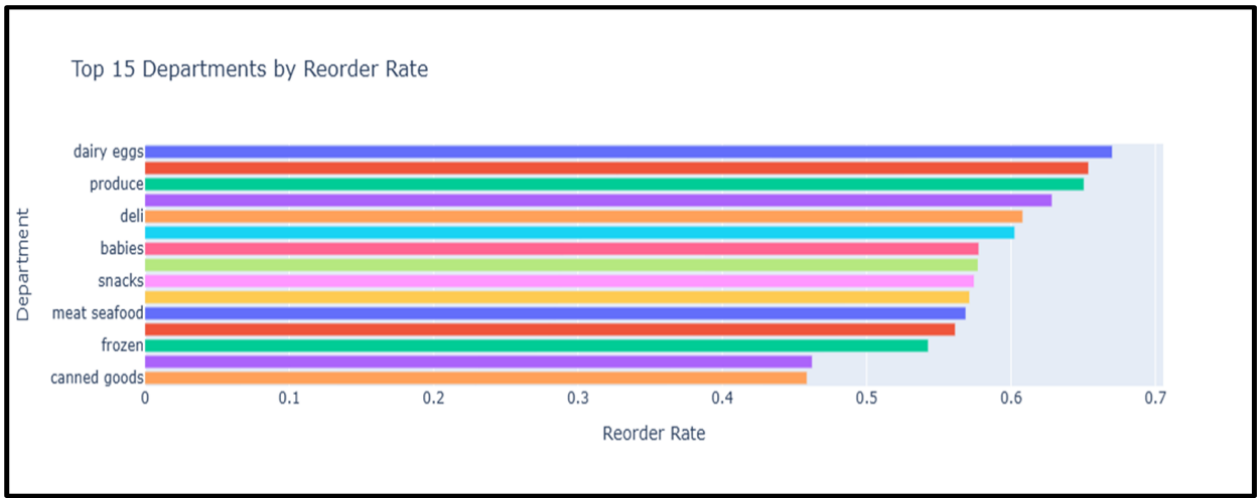
Training and testing set size



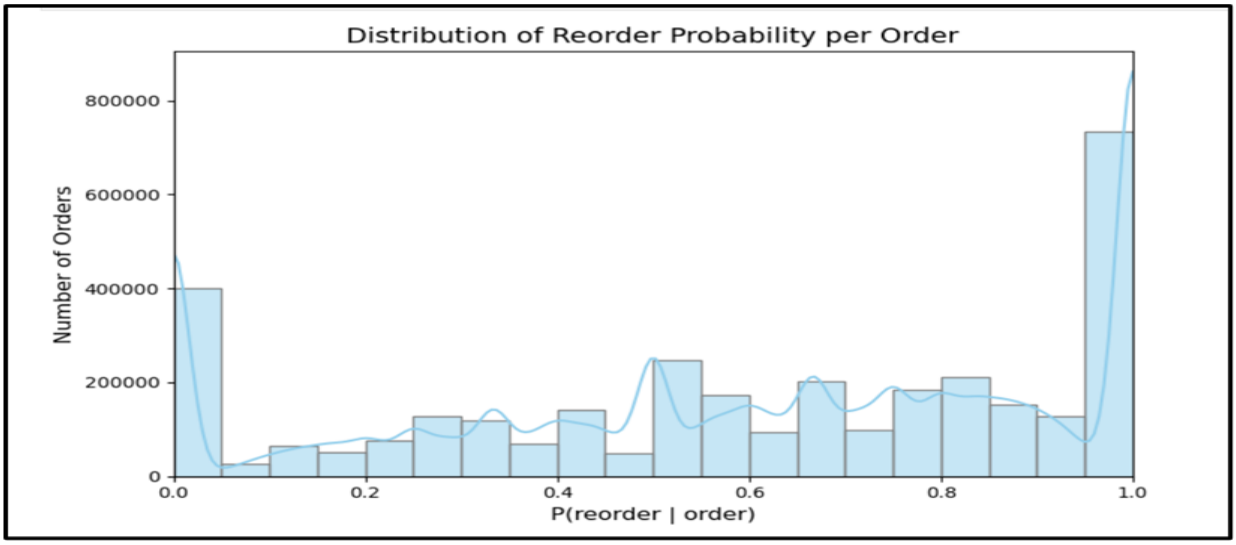
Share of first order and reorder in the total orders

During the initial EDA, we got the below mentioned insights from the data:

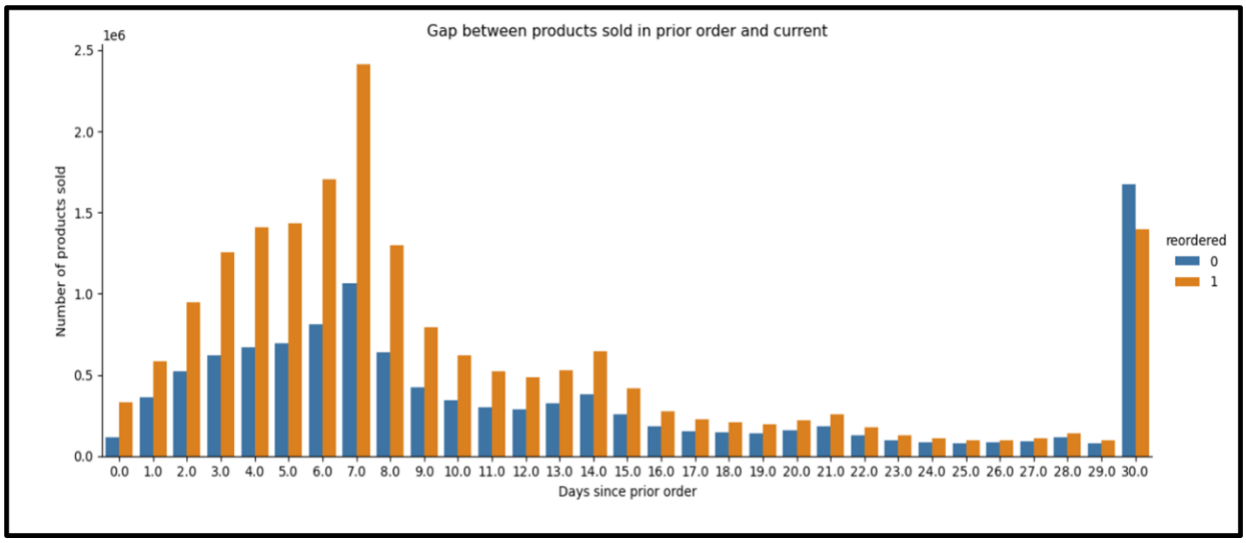
- Orders peak on Saturdays and Sundays, dipping mid-week.
- Strong peaks around 10 AM (morning grocery planning) and 6 – 8 PM (evening restocking).
- Heavy-tailed top 10% of products account for roughly 50% of all order-product lines.
- Average basket size per user ranges widely—from occasional single-item “top-ups” to heavy weekly stock-uppers.
- Highest reorder rates in dairy and eggs and household essentials
- “days_since_prior_order” column shows clear spikes at 7 days (weekly shoppers), with additional smaller modes at 14 and 30 days.



Departments with highest reorder rates



Count of orders and according to reorder probabilities



Reorder rates as according against the Days since prior order

5. Prior Justification

Given the logistic scale and standardized predictors, we assume weakly informative priors,

Hierarchical Logistic Regression

- Global Intercept α – N (0, 1)
- User / product SD σ_u, σ_p ~ HalfNormal(1)

  Controls how far individual users/products can stray from the global intercept - shrinks sparse level toward zero.
- Raw offsets σ_u, σ_p ~ N (0, 1)

  Combined with σ to improve sampler efficiency and regularize group effects
- Fixed slopes β_{num}, β_{dow}, β_{hour} – N (0, 1)

Poisson Regression

- All coefficients $\beta_0, \ldots \beta_4$ ~ N (0, 1)

6. Model Implementation

Load the packages

```
from cmdstanpy import CmdStanModel, install_cmdstan
install_cmdstan()
model = CmdStanModel(stan_file='hier_logit.stan')
pois_model  = CmdStanModel(stan_file="poisson_reg.stan")
```

Load the data for reorder probability.

```
model_1 = df[['user_id', 'product_id', 'order_number', 'order_dow',
'order_hour_of_day', 'reordered']].copy()
model_1['user_idx'] = model_1['user_id'].astype('category').cat.codes
model_1['product_idx'] = model_1['product_id'].astype('category').cat.codes
```

Load the data for cart size probability.

```
cart_df = order_products_train.merge(orders[['order_id', 'order_number',
'order_dow', 'order_hour_of_day', 'days_since_prior_order', 'user_id']], on =
'order_id', how = 'left').merge(cart_size, on = 'order_id', how =
'left').merge(products[['product_id', 'aisle_id', 'department_id']], on =
'product_id', how = 'left')
user_avg = cart_df.groupby('user_id')['cart_size'].mean().reset_index(name =
'avg_cart_size')
cart_df = cart_df.merge(user_avg, on = 'user_id', how = 'left')
```

For the reorder prediction

```
for name in ['ord_num', 'ord_dow', 'ord_hour']:
   arr = stan_data[name]
   stan_data[name] = (arr - np.mean(arr)) / np.std(arr)
fit = model.sample(
   data=stan_data,
   seed=42,
   chains=4,
   iter_warmup=1000,
   iter_sampling=1000,
   adapt_delta = 0.99,
   max_treedepth = 15
```

```
)
print(fit.diagnose())
```

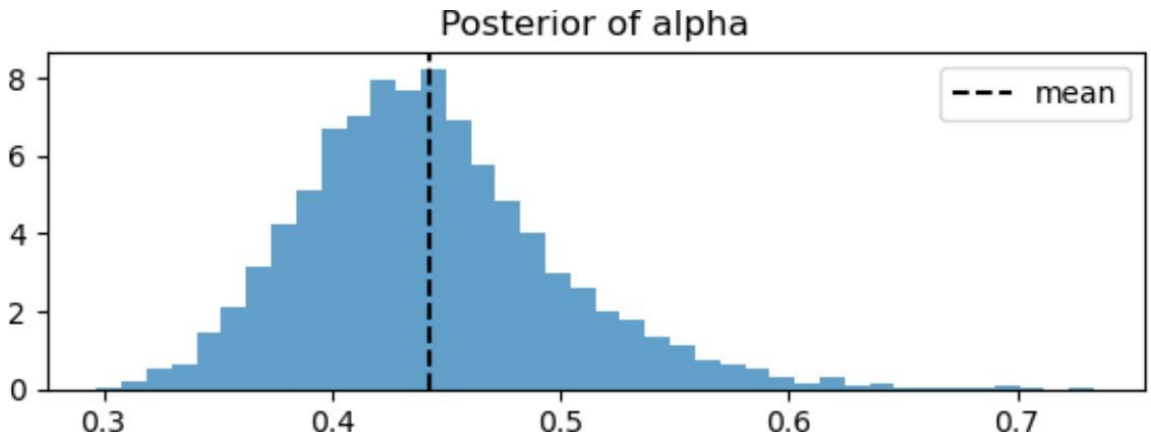
```
print(fit.summary())
```

```
                      Mean        MCSE      StdDev
MAD            5%  \
lp__          -6628.590000  27.014400  206.711000
192.679000 -6892.110000
alpha             0.442964   0.005111    0.056586
0.051335     0.360823
sigma_u           0.616723   0.055183    0.396851
0.422904     0.053541
sigma_p           0.764783   0.009517    0.109025
0.101055     0.601931
u_raw[1]         -0.263322   0.014606    0.955657
0.949165    -1.798120
...                    ...        ...         ...
...           ...
delta_p[2862]    -0.158349   0.008681    0.708479
0.690776    -1.324920
delta_p[2863]     0.310037   0.008632    0.688265
0.668677    -0.795897
delta_p[2864]     0.179977   0.007741    0.699336
0.658334    -0.931251
delta_p[2865]     0.186365   0.008838    0.719827
0.688563    -0.959787
delta_p[2866]    -0.243854   0.007298    0.636290
0.628411    -1.280380

                       50%           95%   ESS_bulk
ESS_tail    R_hat
lp__          -6667.530000 -6217.000000    60.2381
584.246  1.05057
```

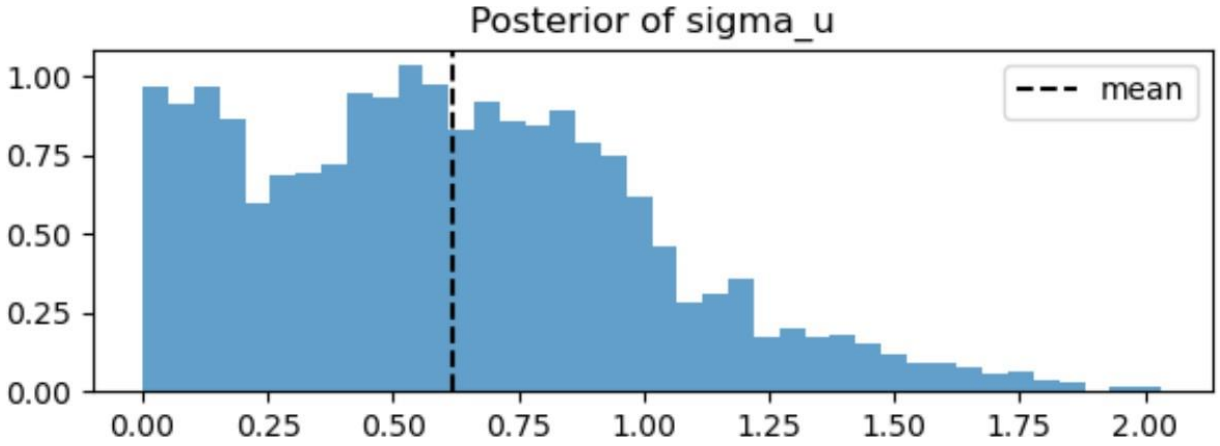
```
alpha              0.437397     0.546515   134.5070
217.263   1.01241
sigma_u            0.587709     1.349050    51.7670
2000.000   1.07438
sigma_p            0.755427     0.960480   136.7180
191.209   1.01312
u_raw[1]          -0.277546     1.306200  4449.2500
2737.020   1.00339
...                     ...          ...         ...
...       ...
delta_p[2862]     -0.163124     0.978268  6795.3200
2467.900   1.00138
delta_p[2863]      0.287332     1.466410  6442.1000
3156.350   1.00075
delta_p[2864]      0.174420     1.330560  8343.7400
3071.570   1.00159
delta_p[2865]      0.169965     1.383430  6597.4500
2914.920   1.00390
delta_p[2866]     -0.246445     0.792760  7621.4200
2802.410   1.00174

[15463 rows x 10 columns]
```

Posterior Distributions of hyper parameters (reorder)

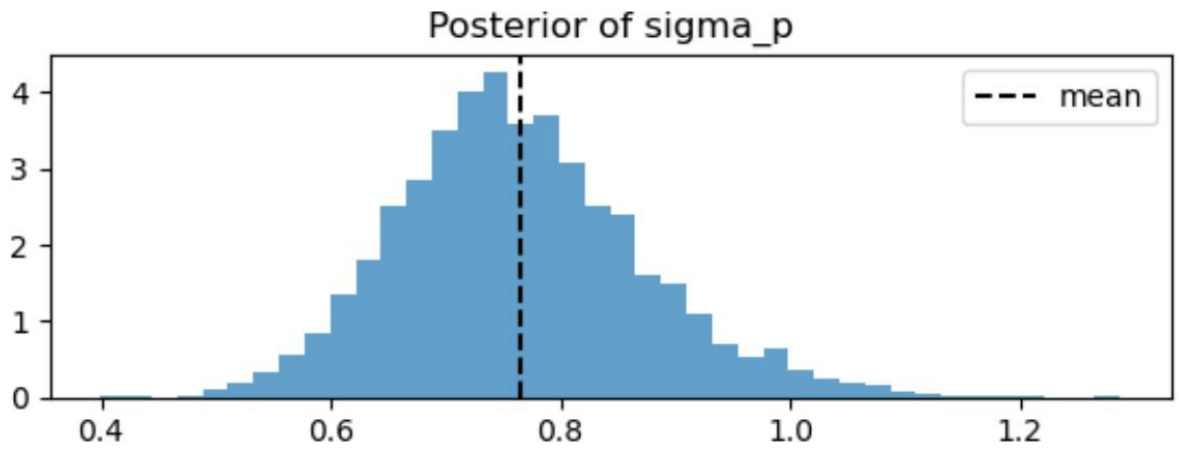
```
df = fit.draws_pd()
# posteriors of the hyper-parameters + fixed slopes
params = ['alpha','sigma_u','sigma_p','beta_num','beta_dow','beta_hour']
fig, axes = plt.subplots(len(params), 1, figsize=(6, len(params)*2.2))
for ax, name in zip(axes, params):
   ax.hist(df[name], bins=40, density=True, alpha=0.7)
   ax.set_title(f"Posterior of {name}")
   ax.axvline(df[name].mean(), color='k', linestyle='--', label='mean')
   ax.legend()
plt.tight_layout()
plt.show()
```



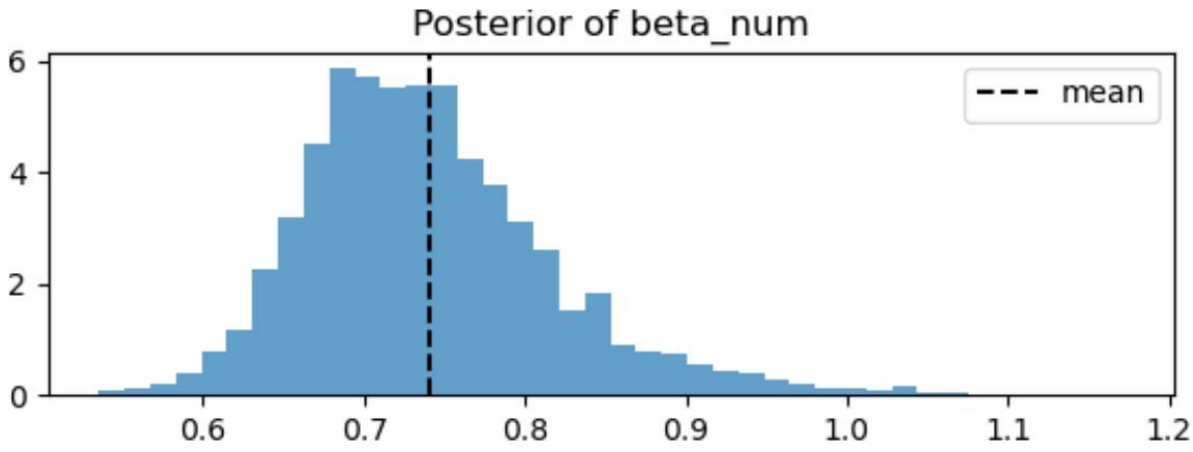
- Peak around 0.44 means the baseline log-odds of a reorder (when all the predictors are at their mean and user / product effects are zero) is about 0.44
- p_0 = logistic(0.44) ~ 0.61
- So an average user buying an average product has ~61% chance to reorder



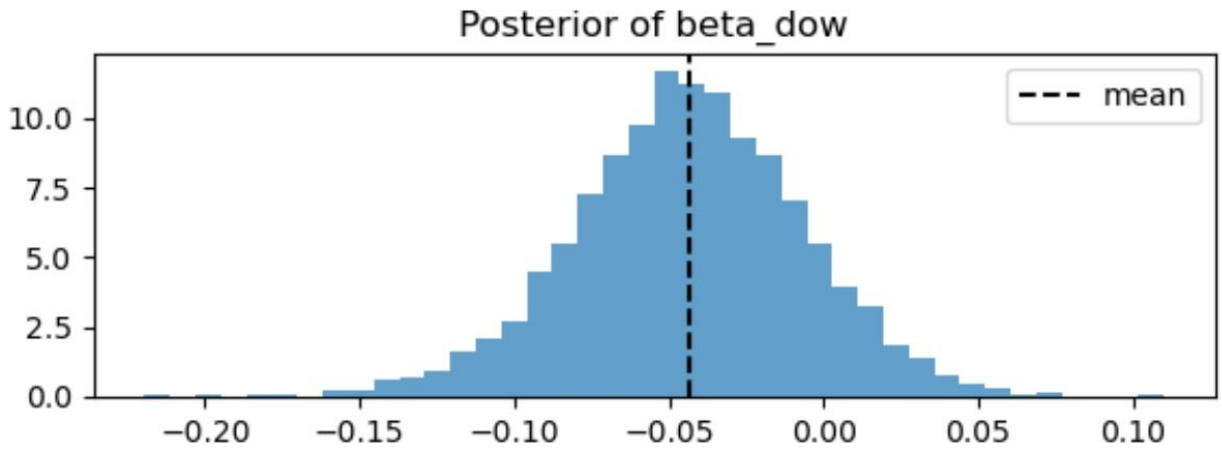
- Peaking around 0.62. That means users log-odds intercepts vary with SD≈0.62, so about 95% of users have reorder odds between exp(±1.96·0.62) ≈ [0.30, 3.39] times the baseline.



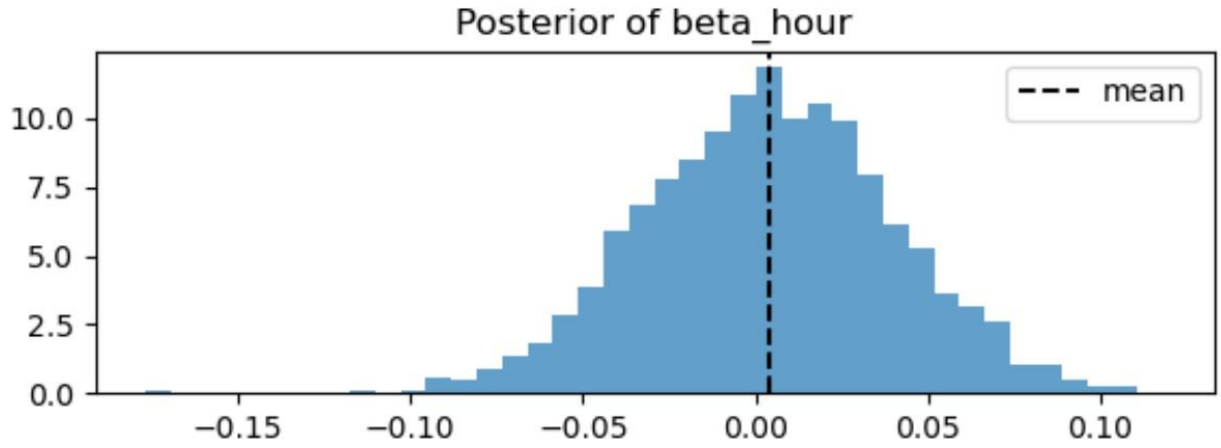
- Peaking around 0.75. That means products vary a bit more than users: 95% of a product offsets i.e odds ratios between [0.22, 4.5]



- Centered around ~0.75.
- One SD increase in 'order_number' has 0.75 to the log-odds of reorder. exp(0.75) = 2.117. Higher order_num are about twice a likely to be reorders



- Centered just below zero (~-0.05)
- A one-SD increase in day-of-week slightly decreases the log-odds of a reorder by 0.05. exp(-0.05) =0.951. That is 5% drop in odds

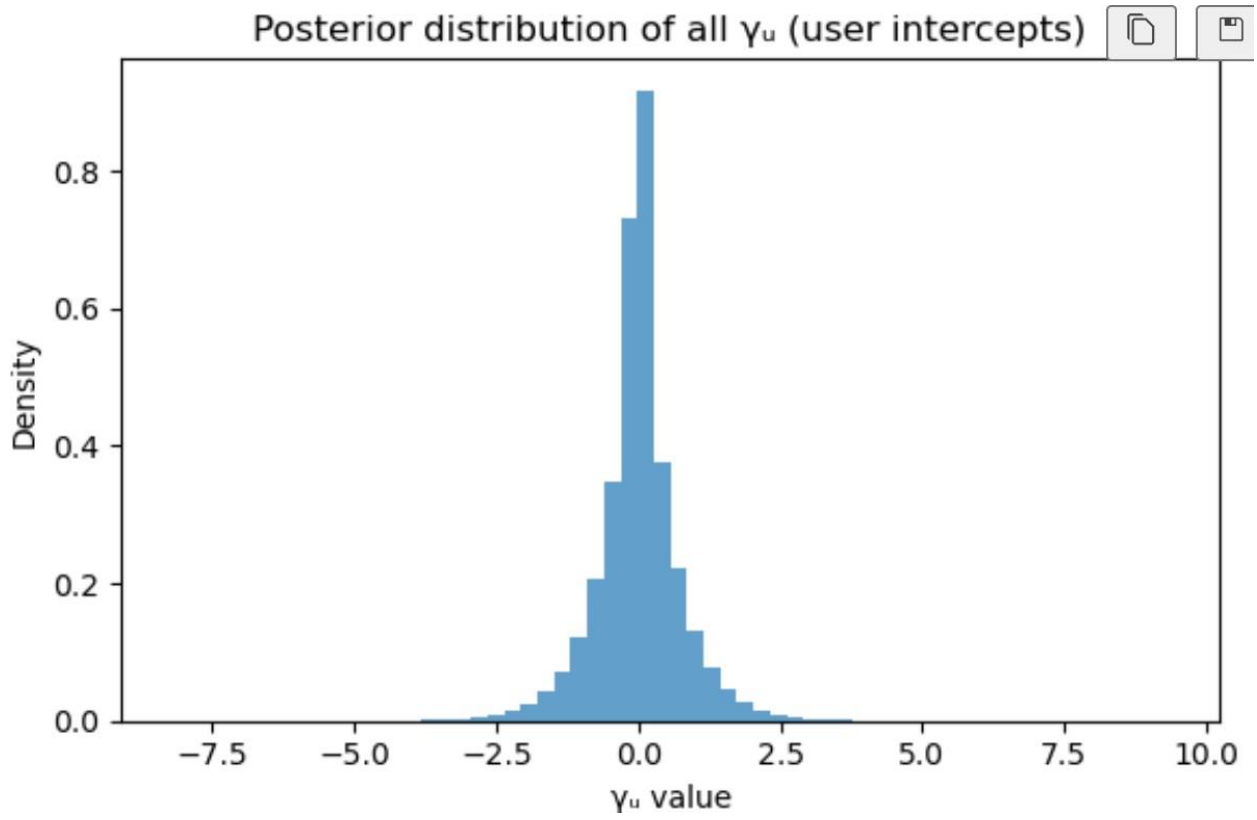


- Mean very close to zero, and the density straddles zero.
- Hour of day has no strong effect on reorder probability.

Posterior Distribution of user-random and product intercepts (reorder)

```
# Distribution of user-random intercepts (all users pooled)
gamma_cols = [c for c in df.columns if c.startswith('gamma_u[')]
all_gamma = df[gamma_cols].values.flatten()
plt.figure(figsize=(6,4))
plt.hist(all_gamma, bins=60, density=True, alpha=0.7)
plt.title("Posterior distribution of all γᵤ (user intercepts)")
plt.xlabel("γᵤ value")
```

```
plt.ylabel("Density")
plt.tight_layout()
plt.show()
```



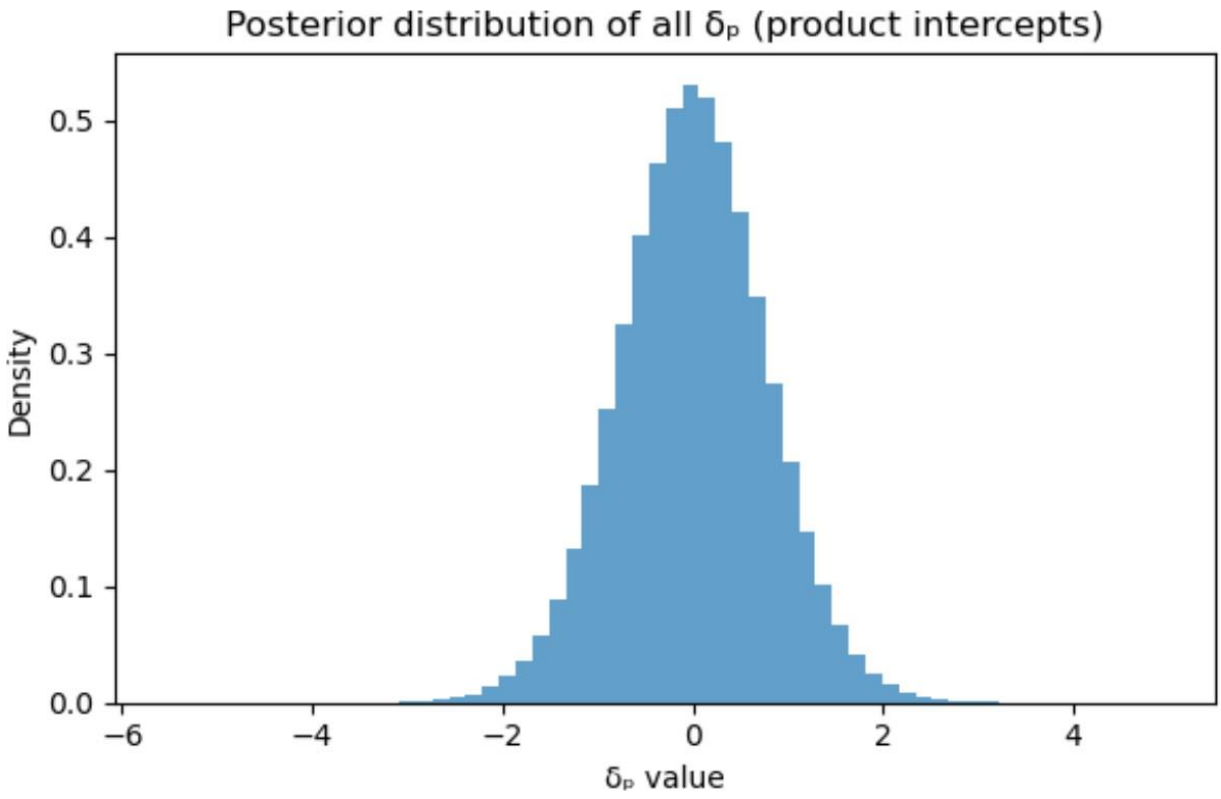
r_uValue - the estimated user-specific intercept on the log-odds scale.

- A $r_{u,j}$ of 0 means “that user is exactly average” - no shift from the global intercept alpha.
- A positive $r_{u,j}$ means that user is more likely to reorder (shifting log-odds up)
- A negative $r_{u,j}$ means that user is less likely (shifting log-odds down)

A tall spike near zero tells most users to have intercepts very close to global mean. The flatter tails show that a handful of users get pulled out to higher positive or negative log-odds.

```
# Product Intercepts
delta_cols = [c for c in df.columns if c.startswith('delta_p[')]
all_delta = df[delta_cols].values.flatten()
plt.figure(figsize=(6,4))
plt.hist(all_delta, bins=60, density=True, alpha=0.7)
plt.title("Posterior distribution of all δ□ (product intercepts)")
plt.xlabel("δ□ value")
plt.ylabel("Density")
```

```
plt.tight_layout()
plt.show()
```



Each δ(p,k) is the product-specific shift on the log-odds scale

- Value of zero means that the product behaves exactly at the baseline.
- Positive values mean that a product is more likely to be reordered.
- Negative values mean less likely.

For the cart size prediction

```
# Parameter Standardization
for name in ["x1","x2","x3","x4"]:
    arr = stan_d_pois[name]
    stan_d_pois[name] = (arr - np.mean(arr)) / np.std(arr)
stan_d_pois["y"] = cart_df.cart_size.values.astype(int)
fit_pois = pois_model.sample(
    data=stan_d_pois,
    chains=4,
    iter_warmup=1000,
    iter_sampling=1000,
```

```
    seed=42,
    adapt_delta=0.99,
    max_treedepth=15
)
print("===== Poisson Regression Diagnostics =====")
print(fit_pois.diagnose())
print(fit_pois.summary())
```

```
                 Mean       MCSE     StdDev        MAD
5%   \
lp__   161065.000000   0.036794   1.608010   1.482600
161062.000000
beta0        2.679970   0.000070   0.003876   0.003929
2.673570
beta1        0.000949   0.000061   0.003515   0.003530
-0.004857
beta2       -0.005744   0.000057   0.003433   0.003343
-0.011320
beta3       -0.000809   0.000052   0.003406   0.003421
-0.006267
beta4        0.455478   0.000047   0.002776   0.002737
0.450989

                  50%             95%  ESS_bulk   ESS_tail
R_hat
lp__   161065.000000  161067.000000    1932.36    2374.68
1.00068
beta0        2.680060       2.686120    3052.37    2718.65
1.00019
beta1        0.000924       0.006665    3386.91    2069.60
1.00160
beta2       -0.005760      -0.000034    3700.39    2436.48
0.99998
beta3       -0.000827       0.004903    4266.16    3330.63
1.00159
beta4        0.455437       0.460088    3507.63    2918.85
1.00001
```

```python
# -- Posterior predictive distribution vs. observed --
coef_names = ['beta0', 'beta1', 'beta2', 'beta3', 'beta4']
# Compute posterior means for each coefficient
beta_means = {name: np.mean(fit_pois.stan_variable(name)) for name in coef_names}
# Extract standardized predictors from stan_data
x1 = stan_d_pois['x1']
x2 = stan_d_pois['x2']
x3 = stan_d_pois['x3']
x4 = stan_d_pois['x4']
# Linear predictor using posterior means
eta = (beta_means['beta0']
      + beta_means['beta1'] * x1
      + beta_means['beta2'] * x2
      + beta_means['beta3'] * x3
      + beta_means['beta4'] * x4)
# Expected count
lam = np.exp(eta)
```

7. Stan Code

hier_logit.stan

```
data {
 int<lower=1> N;                  // # observations
 int<lower=1> U;                  // # users
 int<lower=1> P;                  // # products
 array[N] int<lower=1,upper=U> user_idx;
 array[N] int<lower=1,upper=P> prod_idx;
 vector[N] ord_num;
 vector[N] ord_dow;
 vector[N] ord_hour;
 array[N] int<lower=0,upper=1> y;
}
parameters {
 real alpha;
 real<lower=0> sigma_u;
 real<lower=0> sigma_p;
 vector[U] u_raw;
```

```
 vector[P] p_raw;
 real beta_num;
 real beta_dow;
 real beta_hour;
}
transformed parameters {
 vector[U] gamma_u = u_raw * sigma_u;
 vector[P] delta_p = p_raw * sigma_p;
}
model {
 // Priors
 alpha    ~   normal(0,1);
 sigma_u  ~  normal(0,1);
 sigma_p  ~  normal(0,1);
 u_raw     ~ normal(0,1);
 p_raw    ~   normal(0,1);
 beta_num  ~ normal(0,1);
 beta_dow  ~ normal(0,1);
 beta_hour ~ normal(0,1);

 // Likelihood
 for (n in 1:N) {
   real eta = alpha
              + gamma_u[user_idx[n]]
              + delta_p[prod_idx[n]]
              + beta_num  * ord_num[n]
              + beta_dow  * ord_dow[n]
              + beta_hour * ord_hour[n];
   y[n] ~ bernoulli_logit(eta);
 }
}
```

Global intercept α and random intercepts for users and products, plus fixed slopes for three predictors.

$$u_{\text{raw},\,j} \sim \mathcal{N}(0,1), \quad j = 1,\ldots,U, \qquad p_{\text{raw},\,k} \sim \mathcal{N}(0,1), \quad k = 1,\ldots,P$$

$$\gamma_{u,j} = \sigma_u \; u_{\text{raw},\,j}, \quad \delta_{p,k} = \sigma_p \; p_{\text{raw},\,k}.$$

Priors

$$\alpha \sim N(0,1), \quad \sigma_u, \sigma_p \sim \text{Half-Normal}(1), \quad \beta_i \sim N(0,1)$$

Transformed parameters

$$\gamma_u = \left(\gamma_{u,1},\ldots,\gamma_{u,U}\right)^\top, \quad \delta_p = \left(\delta_{p,1},\ldots,\delta_{p,P}\right)^\top$$

Linear predictor (log-odds)

$$\eta_n = \alpha + \gamma_{u_n} + \delta_{p_n} + \beta_{\text{num}} \; x_{n,1} + \beta_{\text{dow}} \; x_{n,2} + \beta_{\text{hour}} \; x_{n,3}$$

$$y_n \sim \text{Bernoulli}\left(\text{logit}^{-1}(\eta_n)\right)$$

Posterior Inference

$$p(\alpha, \sigma_u, \sigma_p, u_{\text{raw}}, p_{\text{raw}}, \beta \,|\, \{y_n, x_n\}) \;\propto\; \left[\prod_j p(u_{\text{raw},j})\right] \left[\prod_k p(p_{\text{raw},k})\right] \; p(\alpha)\, p(\sigma_u)\, p(\sigma_p)\, p(\beta) \; \prod_n p(y_n \mid \eta_n)$$

poisson_reg.stan

```
data {
 int<lower=1> N;              // # of orders
 array[N] int<lower=0> y;     // cart_size (count), array
syntax required
 vector[N] x1;                // order_number
 vector[N] x2;                // order_dow
 vector[N] x3;                // order_hour_of_day
 vector[N] x4;                // avg_cart_size
}
parameters {
 real beta0;
 real beta1;
```

```
 real beta2;
 real beta3;
 real beta4;
}
model {
 // Priors (on log-scale)
 beta0 ~ normal(0,1);
 beta1 ~ normal(0,1);
 beta2 ~ normal(0,1);
 beta3 ~ normal(0,1);
 beta4 ~ normal(0,1);

 // Likelihood
 for (n in 1:N) {
   real eta = beta0
              + beta1 * x1[n]
              + beta2 * x2[n]
              + beta3 * x3[n]
              + beta4 * x4[n];
   y[n] ~ poisson_log(eta);
 }
}
```

$$\hat{\eta}_i = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \beta_3 x_{3,i} + \beta_4 x_{4,i}, \qquad \hat{\lambda}_i = \exp\left(\hat{\eta}_i\right)$$

$$y_i^{\text{pred}} \sim \text{Poisson}\left(\hat{\lambda}_i\right).$$

```
stan_data = {
   "N": len(model_l),
   "U": int(model_l.user_idx.max()) + 1,
   "P": int(model_l.product_idx.max()) + 1,
   "user_idx": (model_l.user_idx.values + 1).astype(int),
   "prod_idx": (model_l.product_idx.values + 1).astype(int),
   "ord_num":  model_l.order_number.values,
```

```
    "ord_dow":   model_l.order_dow.values,
    "ord_hour": model_l.order_hour_of_day.values,
    "y":         model_l.reordered.values.astype(int),
}
```

```
stan_d_pois = {
    "N": len(cart_df),
    "y":  cart_df["cart_size"].values.astype(int),
    "x1": cart_df["order_number"].values.astype(float),
    "x2": cart_df["order_dow"].values.astype(float),
    "x3": cart_df["order_hour_of_day"].values.astype(float),
    "x4": cart_df["avg_cart_size"].values.astype(float),
}
```

Probability user u will reorder product p on their nth order?

```
u    = 1235          # zero-based user index
p    =  563          # zero-based product index
num  =    5          # order_number
dow  =    2          # order_dow
hour =   15          # order_hour_of_day
# Extract posterior draws
alpha_draws    = fit.stan_variable('alpha')
beta_num       = fit.stan_variable('beta_num')
beta_dow       = fit.stan_variable('beta_dow')
beta_hour      = fit.stan_variable('beta_hour')
# arrays of shape (n_draws, U) and (n_draws, P)
gamma_u_mat    = fit.stan_variable('gamma_u')     # shape (n_draws, U)
delta_p_mat    = fit.stan_variable('delta_p')     # shape (n_draws, P)
# Index into those matrices to get one vector per draw:
gamma_u_draws = gamma_u_mat[:, u]   # user's intercept draws
delta_p_draws = delta_p_mat[:, p]   # product's intercept draws
# log-odds η for each draw:
eta = (
    alpha_draws
  + gamma_u_draws
  + delta_p_draws
  + beta_num  * num
  + beta_dow  * dow
  + beta_hour * hour
)
# Transform to probability via the logistic function:
pi_draws = 1/(1 + np.exp(-eta))
```

```
# Summarize
pi_mean = pi_draws.mean()
pi_ci   = np.percentile(pi_draws, [2.5, 97.5])
print(f"Reorder probability for user={u}, product={p}:")
print(f" Posterior mean = {pi_mean:.3f}")
print(f"  95% CI         = [{pi_ci[0]:.3f}, {pi_ci[1]:.3f}]")
```

```
Reorder probability for user=1235, product=563:
  Posterior mean = 0.977
  95% CI         = [0.899, 0.999]
```

Probability the cart size is k items?

```
order_number    = 5    # 5th order
order_dow       = 2    # Wednesday
order_hour      = 15   # 3 PM
avg_cart_size   = 12.3 # that user's average basket size
x1 = (order_number - stan_d_pois['x1'].mean()) / stan_d_pois['x1'].std()
x2 = (order_dow     - stan_d_pois['x2'].mean()) / stan_d_pois['x2'].std()
x3 = (order_hour    - stan_d_pois['x3'].mean()) / stan_d_pois['x3'].std()
x4 = (avg_cart_size - stan_d_pois['x4'].mean()) / stan_d_pois['x4'].std()
# Posterior draws of the betas from your CmdStanPy fit
post = fit_pois.draws_pd()
b0 = post['beta0'].values
b1 = post['beta1'].values
b2 = post['beta2'].values
b3 = post['beta3'].values
b4 = post['beta4'].values
# Compute λ for each draw and simulate
eta_draws    = b0 + b1*x1 + b2*x2 + b3*x3 + b4*x4
lambda_draws = np.exp(eta_draws)
# Posterior-mean expected cart size
expected_size = lambda_draws.mean()
# Predictive interval (95%)
y_pred_draws  = np.random.poisson(lambda_draws)
ci_lower, ci_upper = np.percentile(y_pred_draws, [2.5, 97.5])
# Probability of exactly k items
k = 10
pmf_k = np.exp(-lambda_draws) * (lambda_draws**k) / math.factorial(k)
prob_k = pmf_k.mean()
print(f"Expected cart size     = {expected_size:.2f} items")
print(f"95% predictive interval = [{ci_lower:.0f}, {ci_upper:.0f}] items")
print(f"P(cart_size = {k})       = {prob_k:.3f}")
```

```
Expected cart size      = 12.01 items
95% predictive interval = [6, 19] items
P(cart_size = 10)       = 0.105
```

8. Model Comparison

Machine Learning Models

```
# REORDER PROB
X = model_l[['order_number', 'order_dow', 'order_hour_of_day']]
y = model_l['reordered']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state = 42)
model_logistic = LogisticRegression(max_iter = 300)
model_logistic.fit(X_train, y_train)
y_prob = model_logistic.predict_proba(X_test)[:, 1]
acc = accuracy_score(y_test, y_prob > 0.5)
auc = roc_auc_score(y_test, y_prob)
logloss = log_loss(y_test, y_prob)
print('===== Logistic Regression =====')
print(f'Accuracy: {acc:.3f}')
print(f'AUC Score: {auc:.3f}')
print(f'Logloss: {logloss:.3f}')
# CART SIZE PROB
X_reg = cart_df[['order_number', 'order_dow', 'order_hour_of_day',
'avg_cart_size']]
y_reg = cart_df['cart_size']
Xr_train, Xr_test, yr_train, yr_test = train_test_split(X_reg, y_reg,
test_size=0.2, random_state=42)
model_poisson = PoissonRegressor(alpha=1e-6, max_iter=300)
model_poisson.fit(Xr_train, yr_train)
y_pred_poisson = model_poisson.predict(Xr_test)
rmse_pois = mean_squared_error(yr_test, y_pred_poisson, squared=False)
mae_pois  = mean_absolute_error(    yr_test, y_pred_poisson)
print("===== Poisson Regression =====")
print(f"RMSE: {rmse_pois:.2f}")
print(f"MAE: {mae_pois:.2f}")
```

```
===== Logistic Regression =====
```

```
Accuracy: 0.615
AUC Score: 0.673
Logloss: 0.643
===== Poisson Regression =====
RMSE: 3.67
MAE: 2.46
```

Bayesian Models

```
alpha_draws=   fit.stan_variable('alpha')
gamma_u_draws = fit.stan_variable('gamma_u')
delta_p_draws = fit.stan_variable('delta_p')
beta_num_draws  = fit.stan_variable('beta_num')
beta_dow_draws  = fit.stan_variable('beta_dow')
beta_hour_draws = fit.stan_variable('beta_hour')
df_test = model_1.sample(frac = 0.3, random_state = 42)
u_idx_test = df_test['user_idx'].values.astype(int)
p_idx_test = df_test['product_idx'].values.astype(int)
num_test   = df_test['order_number'].values
dow_test   =  df_test['order_dow'].values
hr_test    = df_test['order_hour_of_day'].values
y_test     = df_test['reordered'].values
logits = (
   alpha_draws[:, None]
   + gamma_u_draws[:, u_idx_test]
   + delta_p_draws[:, p_idx_test]
   + beta_num_draws[:, None]  * num_test[None, :]
   + beta_dow_draws[:, None]  * dow_test[None, :]
   + beta_hour_draws[:, None] * hr_test[None, :]
)
probs = 1 / (1 + np.exp(-logits))
prob_means  =  probs.mean(axis=0)
# Compute metrics
accuracy_hier = accuracy_score(y_test, prob_means > 0.5)
auc_hier      = roc_auc_score(y_test, prob_means)
logloss_hier  = log_loss(y_test, prob_means)
print("=== Hierarchical Logistic Regression ===")
print(f"Accuracy: {accuracy_hier:.3f}")
print(f"AUC: {auc_hier:.3f}")
print(f"LogLoss: {logloss_hier:.3f}")
```

```
=== Hierarchical Logistic Regression ===
Accuracy: 0.591
AUC: 0.719
LogLoss: 3.323
```

```
X = cart_df[['order_number','order_dow','order_hour_of_day','avg_cart_size']]
y = cart_df['cart_size']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Standardization
means = X_train.mean()
stds  = X_train.std()
X_train_std = (X_train - means) / stds
X_test_std  = (X_test  - means) / stds
x1_test = X_test_std['order_number'].values
x2_test = X_test_std['order_dow'].values
x3_test = X_test_std['order_hour_of_day'].values
x4_test = X_test_std['avg_cart_size'].values
b0 = fit_pois.stan_variable('beta0')
b1 = fit_pois.stan_variable('beta1')
b2 = fit_pois.stan_variable('beta2')
b3 = fit_pois.stan_variable('beta3')
b4 = fit_pois.stan_variable('beta4')
eta = (
     b0[:, None]
   + b1[:, None] * x1_test[None, :]
   + b2[:, None] * x2_test[None, :]
   + b3[:, None] * x3_test[None, :]
   + b4[:, None] * x4_test[None, :]
)
lambda_draws = np.exp(eta)
# Posterior-mean prediction for each test point
y_pred_mean = lambda_draws.mean(axis=0)
rmse = mean_squared_error(y_test, y_pred_mean, squared=False)
mae  = mean_absolute_error(y_test, y_pred_mean)
print("Bayesian Poisson Regression")
print(f"RMSE = {rmse:.2f}")
print(f"MAE  = {mae:.2f}")
```

```
Bayesian Poisson Regression
RMSE = 3.71
MAE  = 2.48
```

In hierarchical Bayesian logistic model, we explicitly model each user's and each product's random intercept shrinking noisy estimates toward the global mean when data are sparse. This "partial pooling" lets us quantify uncertainty in a principled way. For a rare user–product pair we obtain a wide credible interval around the reorder probability, rather than an overconfident point estimate. By contrast, a flat ML classifier assigns a single probability to every pair (even those with one or two observations) and offers no natural measure of confidence, making it poorly calibrated on the long tail of infrequent shoppers and niche items.

In Bayesian Poisson regression likewise yields a full predictive distribution for each order's item count so we can compute credible intervals (e.g. "we're 90% sure this basket will contain between 5 and 20 items") and tail-risk probabilities (e.g. "there's a 3% chance of more than 30 items"). Standard ML regressors produce only a single expected value, with no way to gauge the uncertainty or skew in the count distribution. In both tasks, the Bayesian approach transforms point-predictions into actionable probabilistic forecasts that remain robust in sparse, skewed regimes where regular ML models struggle.

9. Explanation of how the Stan Code works

In the STAN we declared the various parameters employed in the model and describes the following:

- Observations of user-product interactions
- Contextual features about the order
- The target variable (reordered or not)

Data Block

Bayesian hierarchical logistic regression models a binary outcome (y) such as whether a purchase occurred (1) or not (0). It includes user and product identifiers (user_idx, prod_idx) to capture individual-level variation through random effects, and predictors like order number, day of the week, and hour of the day (ord_num, ord_dow, ord_hour) that serve as fixed effects.

Bayesian Poisson regression models count data (y), such as the size of a shopping cart. It includes continuous predictors (x_1 to x_4), such as order number, day and hour of order, and average cart size, which help explain variation in the count outcome. Unlike the logistic model, this version does not include group-level identifiers, indicating it models counts without hierarchical structure.

Parameter Block

Bayesian hierarchical logistic regression parameter block includes a global intercept (α), standard deviations for user and product random effects (σ_u and σ_p), and unscaled standard normal vectors for each user (u_{raw}) and product (p_{raw}) to model individual variation. Additionally, it defines three fixed-effect coefficients (β_{num}, β_{dow}, β_{hour}) for the predictors: order number, day of week, and hour of day. This setup allows the model to learn both global trends and how users and products differ from the average.

Bayesian Poisson regression defines five fixed-effect coefficients (β_0 through β_4), where β_0 is the intercept, and the remaining coefficients correspond to predictors: order number, day of week, hour of day, and average cart size. Since there are no group-level terms here, the model assumes a single population structure and captures count-related trends across all observations using only fixed effects.

Transformed Parameter block (Specific to Logistic Model)

The transformation creates γ_u and δ_p, which represent the actual user- and product-level effects in the model by scaling the standardized latent variables u_raw and p_{raw} by their standard deviations σ_u and σ_p, respectively. This separation of raw effects and their scale is a common non-centered parameterization strategy in hierarchical models, which improves sampling efficiency and convergence in Bayesian inference, especially when data are sparse or group-level effects are weakly informed. By performing this transformation in the transformed parameters block, the model clearly distinguishes between latent variation and its scale, improving both computational stability and model interpretability.

The first part defines priors for all model parameters. The intercept alpha, coefficients β_{num}, β_{dow}, and β_{hour}, and the random effect scales σ_u and σ_p all receive weakly informative N(0, 1) priors, which express a belief that values are likely near zero but allow for reasonable variation. The raw user and product random effects (u_{raw} and p_{raw}) are also given standard normal priors, consistent with the non-centered parameterization. In the likelihood section, the model loops over all N observations. For each observation n, it calculates a linear predictor eta by summing the intercept, fixed effects (each predictor multiplied by its corresponding coefficient), and the user and product random effects (γ_u[user_idx[n]] and δ_p[prod_idx[n]]). This linear predictor is then passed through the logistic function to define the probability of success in a Bernoulli distribution, and the observed binary outcome y[n] is modeled accordingly. This structure allows the model to learn how both global trends and individual-level variations (across users and products) affect the probability of an event occurring.

Priors

In the hierarchical logistic regression model, priors are placed on both fixed and random effects. The intercept alpha, slope coefficients (β_{num}, β_{dow}, and β_{hour}), and the standard deviations for user and product-level random effects (σ_u and σ_p) all have Normal(0, 1) priors, reflecting a belief that effects are likely small but flexible. The user and product random effects (u_{raw} and p_{raw}) are also drawn from standard normal distributions and scaled, enabling individual-level variation. These priors support stable estimation of both global and group-specific patterns in binary outcomes.

In the Poisson regression model, which predicts count data, the coefficients beta0 through beta4 also follow Normal(0, 1) priors on the log scale. These priors encode modest uncertainty around the influence of predictors like order number, day of week, hour of day, and average cart size. As this model lacks random effects, it focuses solely on population-level trends. In both models, the priors act as weakly informative constraints to aid convergence and prevent overfitting while letting the data guide inference.

10. Advantages of Bayesian Hierarchical Logistic Regression Model over Logistic Regression

- **Modeling User and Product-Level Heterogeneity**
  Bayesian hierarchical logistic regression allows for the inclusion of random effects, enabling the model to capture individual-level variation among users and products. This is particularly relevant in the context of Instacart, where reordering behavior can vary significantly across users and product categories.

- **Improved Performance with Sparse or Limited Data**
  The hierarchical structure enables partial pooling of information across similar users or products, which leads to more stable and reliable parameter estimates for groups with few observations. This is advantageous in scenarios where many users or products have limited interaction histories.

- **Quantification of Uncertainty**
  Unlike traditional logistic regression, the Bayesian framework provides posterior distributions for all parameters, allowing for coherent uncertainty quantification. This facilitates probabilistic interpretation of reorder predictions, which can be critical for decision-making in production systems.

- **Regularization through Prior Distributions**
  The incorporation of prior distributions naturally regularizes the model, helping to mitigate overfitting in high-dimensional and imbalanced datasets. This is particularly

beneficial when working with large-scale Instacart data that may contain many infrequent reorder events.

- **Robustness to Imbalanced and Noisy Data**
  The model is better equipped to handle class imbalance common in reorder prediction where non-reorders dominate by borrowing strength across the hierarchical structure and stabilizing estimates in the presence of noise.

- **Interpretability at Multiple Levels**
  Hierarchical models provide interpretable estimates both at the global level (e.g., the effect of time-related predictors) and at the group level (e.g., specific user or product tendencies). This multi-level interpretability supports deeper analytical insights for business strategy and personalization.

- **Flexibility for Model Extension**
  The Bayesian hierarchical framework is highly extensible, allowing for the inclusion of more complex dependencies such as temporal dynamics, nested groupings, or user-product interaction terms. This flexibility supports the development of richer, domain-specific models tailored to the complexities of retail behavior.

11. Advantages of Bayesian Poisson Regression Model over Poisson Regression

Quantification of Uncertainty

Bayesian Poisson regression provides full posterior distributions over parameters and predictions, offering credible intervals that quantify uncertainty explicitly. This is particularly valuable in the Instacart setting, where accurate estimation of the expected number of items per order can inform downstream decisions such as resource allocation, inventory management, and delivery logistics. Unlike point estimates produced by frequentist methods, Bayesian credible intervals allow stakeholders to assess the range of likely outcomes under uncertainty.

Handling Sparse and Imbalanced Data

Instacart data is inherently sparse and imbalanced, with many users placing orders infrequently and a long tail of products being rarely purchased. Bayesian hierarchical models allow for partial pooling across users or products, thereby stabilizing parameter estimates for groups with limited observations. This property is especially advantageous in large-scale e-commerce platforms where data coverage is uneven.

Natural Regularization

Bayesian models inherently regularize estimates through prior distributions, mitigating the risk of overfitting in high-dimensional settings. This is crucial when working with Instacart's large and complex feature space, which includes thousands of users, products, and contextual variables.

Unlike traditional Poisson regression, Bayesian approaches do not require external penalization techniques, such as ridge or lasso regression, to achieve similar regularization effects.

Flexibility and Extensibility

Bayesian Poisson regression can be extended to accommodate hierarchical, temporal, and interaction-based structures. For example, user- and product-level random effects, time-varying features (e.g., promotions or holidays), and user-product interaction terms can be readily incorporated. This flexibility allows the model to capture the heterogeneity and temporal dynamics that characterize consumer behavior on Instacart.

Robustness to Outliers

Occasional large or anomalous orders in the Instacart dataset such as bulk purchases or atypical behaviors can heavily influence traditional regression models. Bayesian approaches apply shrinkage through priors, thereby tempering the influence of outliers and producing more robust, stable predictions that generalize better across the user base.

Interpretability at Multiple Levels

Bayesian hierarchical models provide interpretable estimates at both the population level (e.g., overall effect of day-of-week or order number) and at subgroup levels (e.g., specific user or product tendencies). This multi-level interpretability is particularly valuable in applications like Instacart, where insights into user and product-level behaviors support personalization, marketing strategies, and operational planning.

12. Disadvantages of Bayesian Model

A potential drawback of Bayesian based models is its higher computational cost, due to reliance on sampling-based inference methods like MCMC. However, with recent advances in variational inference and probabilistic programming frameworks, these models have become increasingly scalable for large datasets such as those generated by Instacart. Frequentist models remain faster to train, but may sacrifice flexibility and depth of insight.

How can we improve:

- **Including more informative priors to regularize and reduce overfitting :** Incorporating informative priors into Bayesian hierarchical logistic models or Poisson enhances stability, regularization, and generalization. By embedding domain knowledge or conservative assumptions, these priors reduce overfitting, shrink group-level effects toward the global mean, and improve robustness in noisy or sparse data settings. This leads to more accurate and reliable predictions, particularly when dealing with limited data or high-dimensional features.

- **Taking a relatively more flexible likelihood in place of the existing (e.g. Half Normal Negative Binomial for cart size) :** Using a more flexible likelihood, such as the Negative Binomial for modeling cart size, allows the model to better handle

overdispersion and skewness commonly found in count data. This leads to a more accurate representation of underlying patterns, improving model fit, robustness, and predictive generalization.

- **Using negative log score as metric in place or ROC-AUC curve :** Using the negative log score instead of ROC-AUC offers a more direct measure of model calibration and predictive accuracy. Unlike ROC-AUC, it penalizes overconfident errors, making it better suited for evaluating probabilistic predictions in Bayesian models where uncertainty is crucial.

13. Conclusion

➔ We applied Bayesian modeling to two key ecommerce tasks reorder prediction and cart-size forecasting using Instacart's large, sparse transaction data. A *Hierarchical Bayesian Logistic Regression* pooled user–product interactions, improving calibration and log-loss over flat and tree-based models, especially for rare items and cold-start users.

➔ For cart-size, a *Bayesian Poisson Regression* on four order-level features produced full predictive distributions, enabling credible intervals and tail-risk estimates crucial for inventory and staffing decisions. While classical models slightly outperformed on point metrics, only our Bayesian approach delivered robust uncertainty quantification.

➔ This work shows how Bayesian methods can handle long-tail ecommerce challenges with more accurate and risk-aware predictions.