Web Appendix

Detecting Routines: Applications to Ridesharing CRM

Ryan Dew, Eva Ascarza, Oded Netzer, Nachum Sicherman

August 23, 2023

Contents

A. Web Appendix A: PyMC Implementation

Here, we share an efficient implementation of the model, using PyMC, built on the Aesara library, which we subsequently compiled to NumPyro and JAX to run on a GPU. We thank our partner company, Via, for their help on improving this code.

```
# Base imports
import numpy as np

# PyMC-related imports
import pymc as pm
import aesara
import aesara.tensor as at

# Other imports
from numpy import pi as pi

# reference: https://discourse.pymc.io/t/avoiding-looping-when-using-
#   gp-prior-on-latent-variables/9113/9
class FixedMatrixCovariance(pm.gp.cov.Covariance):
    def __init__(self, cov):
        # super().__init__(1, None)
        self.cov = at.as_tensor_variable(cov)
        self.input_dim = 1

    def full(self, X, Xs):
        # covariance matrix known, not explicitly function of X
        return self.cov

    def diag(self, X):
        return at.diag(self.cov)


def create_model(y, nz_mask, include_obs, n_week_fore, args):
    """Creates the usage model for routines.

    Args:
        y: n_cust x n_week_train array of usage counts (note: this code can be adapted to only use
            non-zero counts; this implementation includes zeros in y)
        nz_mask: an array specifying which elements of y are non-zero
        include_obs: an array that captures which columns in y happened after a customer's
            acquisition date + three weeks
        n_week_fore: the number of weeks ahead to forecast

    Returns:
        A PyMC model corresponding to our proposed routines model.
    """

    n_cust, n_week_train, n_dayhour = y.shape
    n_week_total = n_week_train + n_week_fore

    # inputs are always just the range of possible values, used for creating GPs later
    cust_inputs = np.arange(1, n_cust + 1)[:, None]
    week_inputs = np.arange(1, n_week_total + 1)[:, None]
    day_inputs = np.arange(1, 8)[:, None]
```

```
hour_inputs = np.arange(0, 24)[:, None]

routines_model = pm.Model()
with routines_model:
    # Create aesara objects from data:
    y_at = aesara.shared(y)
    nz_mask_at = aesara.shared(nz_mask)
    include_obs_at = aesara.shared(include_obs)

    # This will be used several times for hierarchical GPs:
    identity_matrix = at.eye(n_cust)
    identity_cov = FixedMatrixCovariance(identity_matrix)

    # GP model for random scaling term, alpha:

    alpha0_scale = pm.HalfNormal("alpha0_scale", sigma=10)
    alpha0 = pm.Normal(
        "alpha0", mu=0, sigma=alpha0_scale, shape=(n_cust)
    )

    ## Hyperparameters
    alpha_amp = pm.HalfNormal("alpha_amp")
    alpha_ls = pm.InverseGamma("alpha_ls", alpha=5, beta=5)

    ## Alpha cov
    cov_alpha = alpha_amp**2 * pm.gp.cov.Exponential(input_dim=1, ls=alpha_ls)

    ## Alpha prior
    alpha_offset_gp = pm.gp.LatentKron(cov_funcs=[identity_cov, cov_alpha])
    alpha_offset = alpha_offset_gp.prior(
        "alpha_offset", Xs=[cust_inputs, week_inputs]
    )
    alpha = pm.Deterministic(
        "alpha",
        alpha0.dimshuffle(0, "x") + alpha_offset.reshape((n_cust, n_week_total)),
    )

    # Common dayhour rate term, mu
    ## Day correlation term:
    mu_omega_chol, mu_omega_corr, mu_omega_scale = pm.LKJCholeskyCov(
        "mu_omega",
        n=7,
        eta=2.0,
        sd_dist=pm.HalfNormal.dist(shape=7),
        compute_corr=True,
    )
    cov_mu_day = FixedMatrixCovariance(mu_omega_corr)

    ## Periodic hour term:
    mu_amp = pm.HalfNormal("mu_amp")
    mu_ls = pm.TruncatedNormal("mu_ls", mu=0.5 * pi, sigma=0.25 * pi, lower=0)
    cov_mu_periodic = mu_amp**2 * pm.gp.cov.Periodic(
        input_dim=1,
        period=24,
        ls=mu_ls / 2, # note: /2 needed to recover original per kernel defn
    )

    ## Combine using Kronecker structure:
    mu_gp = pm.gp.LatentKron(cov_funcs=[cov_mu_day, cov_mu_periodic])
    mu = mu_gp.prior("mu", Xs=[day_inputs, hour_inputs])

    # GP model for routine scaling term, gamma
    ## Define gamma = gamma0 + gamma_offset
    gamma0_scale = pm.HalfNormal("gamma0_scale", sigma=10)
    gamma0 = pm.Normal(
        "gamma0", mu=0, sigma=gamma0_scale, shape=(n_cust)
    )

    ## Hyperparameters
    gamma_amp = pm.HalfNormal("gamma_amp")
    gamma_ls = pm.InverseGamma("gamma_ls", alpha=5, beta=11)

    ## Gamma Covariance
    cov_gamma = gamma_amp**2 * pm.gp.cov.Exponential(input_dim=1, ls=gamma_ls)

    ## Gamma prior
    gamma_offset_gp = pm.gp.LatentKron(cov_funcs=[identity_cov, cov_gamma])
    gamma_offset = gamma_offset_gp.prior(
        "gamma_offset", Xs=[cust_inputs, week_inputs]
    )
    gamma = pm.Deterministic(
        "gamma",
        gamma0.dimshuffle(0, "x") + gamma_offset.reshape((n_cust, n_week_total)),
    )

    # GP model for routine rate, eta

    ## Day correlation term:
    eta_omega_chol, eta_omega_corr, eta_omega_scale = pm.LKJCholeskyCov(
        "eta_omega",
        n=7,
        eta=2.0,
        sd_dist=pm.HalfNormal.dist(shape=7),
        compute_corr=True,
    )
    cov_eta_day = FixedMatrixCovariance(eta_omega_corr)

    ## Periodic hour term:
    eta_amp = pm.HalfNormal("eta_amp")
    eta_ls = pm.TruncatedNormal("eta_ls", mu=0.5 * pi, sigma=0.25 * pi, lower=0)
    cov_eta_periodic = eta_amp**2 * pm.gp.cov.Periodic(
        input_dim=1,
        period=24,
        ls=eta_ls / 2, # note: /2 needed to recover original per kernel defn
    )

    ## Combine using Kronecker structure:
    eta_gp = pm.gp.LatentKron(
        cov_funcs=[identity_cov, cov_eta_day, cov_eta_periodic]
    )
    eta_unshaped = eta_gp.prior(
        "eta_unshaped", Xs=[cust_inputs, day_inputs, hour_inputs]
    )
    eta = pm.Deterministic("eta", eta_unshaped.reshape((n_cust, n_dayhour)))

    # Compute likelihood
    intensity = at.exp(
        alpha.dimshuffle(0, 1, "x")[:, :n_week_train, :] + mu.dimshuffle("x", "x", 0)
        ) + at.exp(
            gamma.dimshuffle(0, 1, "x")[:, :n_week_train, :] + eta.dimshuffle(0, "x", 1)
        )
        lp1 = at.sum(y_at[nz_mask_at] * at.log(intensity[nz_mask_at]))
        lp2 = at.sum(intensity[include_obs_at])
        pm.Potential("lp", lp1 - lp2)

    return routines_model
```

B. Web Appendix B: Additional Details of Simulations

We conducted a number of simulations to evaluate the performance of the model under different data conditions. In the first set of simulations, we examined parameter recovery and scalability under different data settings, assuming the data was generated by our model. In the second set of simulations, we added a churn process to the data generating process, to examine how the model performed in the presence of customer churn. In both studies, the data (without churn) were generated by the following process:

- $\mu(j)$ was drawn from a GP with a constant mean of 0, and the day-hour kernel with lengthscale 5, amplitude 2, and LKJ scaling parameter 2.
- $\eta_i(j)$ was drawn (hierarchically) from a GP with the same parameters as $\mu(j)$
- $\gamma_i(j)$ was drawn (hierarchically) from a GP with a common mean of -8, and an exponential kernel with amplitude 1.5, and lengthscale 5, implying somewhat less smoothness than that found in our empirical setting.
- $\alpha_i(j)$ was drawn (hierarchically) from a GP with the same parameters as $\gamma_i(j)$, but with a lengthscale of 3.

The key features of this simulation are: (1) the means of -8 for $\gamma_i(j)$ and $\alpha_i(j)$ imply a rate of usage similar to that observed in our true data; and (2) the amplitude values imply a rather high amount of variation across people, such that some customers will be very random and not routine, while others will be routine but not random (and some will be in between).

Simulation Study 1: Scalability and Parameter Recovery

To assess both parameter recovery and scalability, we varied the number of customers from 100 to 5,000 (assuming 20 weeks of data), and we varied the number of weeks from 10 to 320 (assuming 200 customers). In each case, we then estimated the model, and measured parameter recovery and runtime.

Parameter Recovery Across all the simulations, we found that the quality of parameter recovery was nearly identical, regardless of the number of customers, and the number of weeks. Thus, in this section, we focus on the results from the simulation with the least data: 100 customers, 20

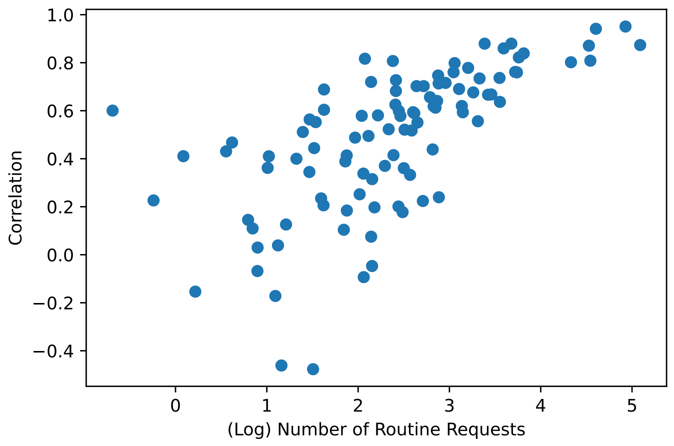


Figure W-1: Correlation Between True and Estimated $\eta_i(j)$
We plot the correlation between the true and estimate values of $\eta_i(j)$, as a function of the logged number of (true) routine requests the person made.

weeks. As reported in the main body of the manuscript, parameter recovery for the focal model parameters is excellent: for population-level parameters like $\mu(j)$, the correlation between true and estimated values is 0.96. For the individual-level parameters, that correlation is slightly lower but still quite high, with a correlation of 0.74 for $\eta_i(j)$, and 0.80 for $\alpha_i(w)$.

While these correlation is quite high, especially for an individual-level parameter, they are imperfect. Much of this imperfection stems from a feature of the model: if most of a customer's transactions were generated from a routine, then estimating the parameters of that person's random process will be difficult to capture, and vice versa. In general, there are many (very negative) values of the parameters that can generate zero transactions. Hence, beyond evaluating the correlation between simulated and estimated values, we explore two additional aspects of parameter recovery: first, for people who actually had routine requests, is the estimated routine correct? And second, is the estimated number of routine requests recovered correctly? Figures W-1 and W-2 show both of these comparisons, respectively. Specifically, in Figure W-1, we show that the correlation between true and estimated values of $\eta_i(j)$ approaches 1 as the (log) number of routine requests grows, becoming very accurate even for a modest number of routine requests (e.g., for more than 20 total routine requests, or, on the log scale, for values above $\log 20 \approx 3$). In Figure W-2, we see that the total number of routine requests over the calibration period is very accurately recovered. Together, these results give strong evidence that our model is statistically identified, even for small amounts of data.

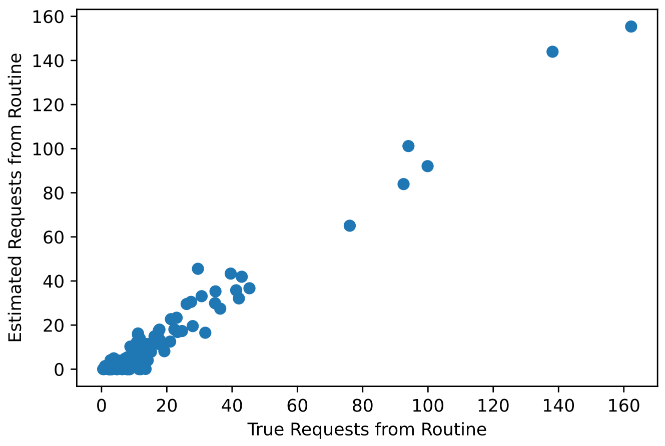


Figure W-2: Correlation Between True and Estimated Total Routineness ($\sum_w E_{iw}^{\text{Routine}}$)
We plot the true versus estimated number of requests coming, in total, from a customer's routine.

Scalability Figure W-3 shows the runtime (in seconds) of the model as we increase the number of customers and the number of time periods. We see that the scaling of the model is superlinear across both dimensions, though the number of customers is the biggest bottleneck. The superlinear scaling is interesting: it suggests that running minibatches of data is much more efficient than running all the data at once. For example, going from 100 customers to 200 customers increases the computation time by approximately 10 minutes, whereas going from 100 customers to 1000 customers increases the computation time by approximately 3.5 hours (~200 minutes).

As described above, the model can recover the true data generating process accurately, even with very few customers. Practically, this suggests that, if the goal is to compute routineness and estimate each customer's routine pattern, leveraging all customers together in one hierarchical model is not essential. Rather, the analyst can split the data into batches that include the full history of different groups of customers, and obtain the desired results in an effective manner. For example, estimating the routines of 1,000,000 customers over 20 weeks of data following this estimation strategy, assuming 10 compute nodes in parallel, would take approximately 14 hours, which we argue is quite feasible for most companies with established data science tools.

Alternatively, relatively recent advances in Bayesian computation, including stochastic gradient HMC (Dang, Quiroz, Kohn, Minh-Ngoc, and Villani 2019) or, in the approximate case, stochastic variational inference (Hoffman, Blei, Wang, and Paisley 2013), could also likely remedy this bottleneck, given those methods can compute computationally demanding parts of the inference algorithm, like the gradient of the log posterior, using only mini-batches of data (in this case,

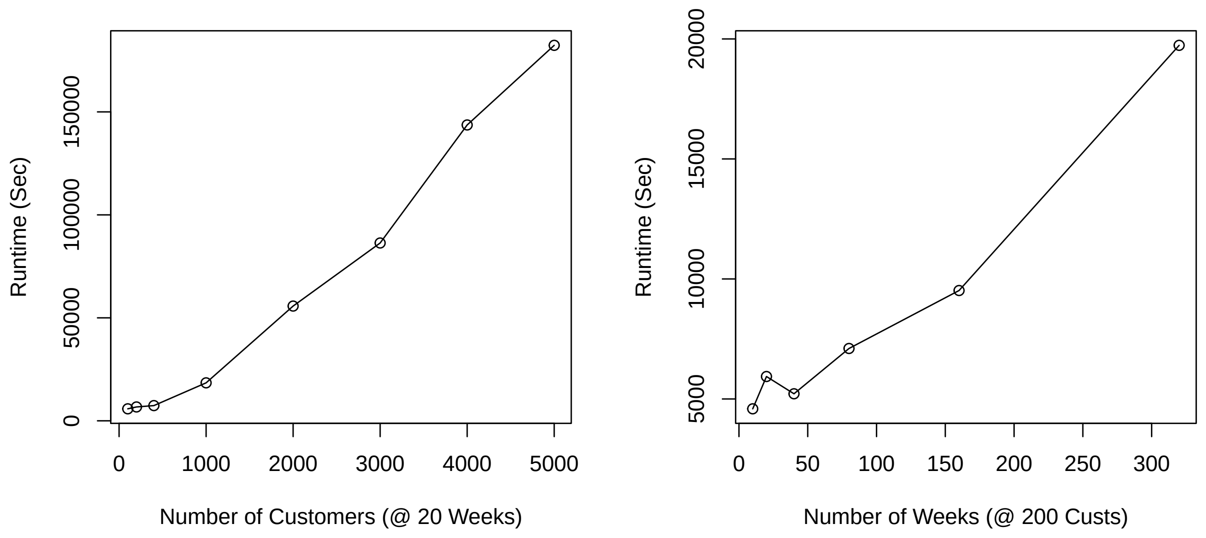


Figure W-3: Scalability Results

Runtime in seconds for estimating the routines model on simulated data.

customers). The degree to which those approaches might reduce the total compute time would depend on how much of the total compute time is dominated by calculating the gradient.

Simulation Study 2: Churn and Forecasting

In our second study we simulated data for 200 customers, for 142 weeks in total. This number of weeks was set by taking the number of calibration weeks in our true data, 38, and assuming we want to forecast 2 years into the future, an additional 104 weeks. Unlike in the previous study, where the data generating process is the same as the model, in this case, we added a churn process that is *not* part of the model, to understand how different rates of churn may affect the reliability of the model and its ability to forecast future spending. We ran three simulations, varying the propensity at which customers churn from the service. Specifically, we assumed that, each week, each customer has a certain probability of churning, which we set to be either 0 (i.e., the same as in the previous study), 0.004 (a moderate rate), or 0.02 (a high rate). If a customer churns, it means their number of transactions is zero from that point forward. We estimate the model using 38 weeks of calibration, and test forecasting over 104 weeks.

Parameter Recovery First, although it was not the focus of this study, we did examine parameter recovery just as in Study 1. As described in the main body of the paper, if customers churn, this

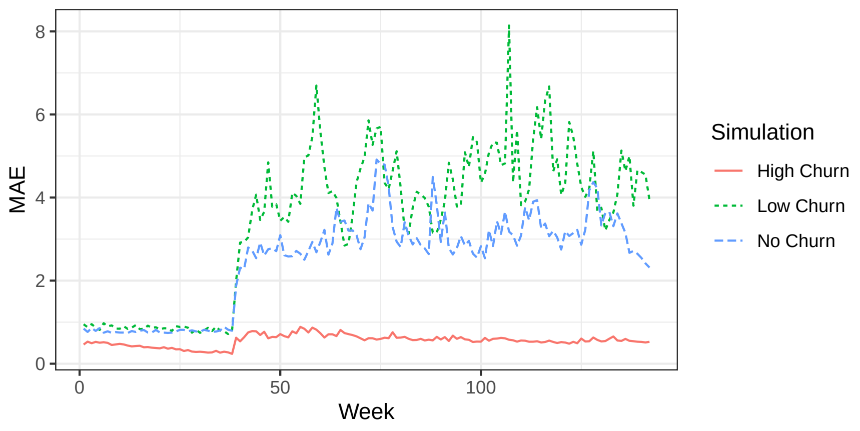


Figure W-4: MAE of Three Simulations with Varying Degrees of Churn
The mean absolute error over the entire 142 weeks (38 calibration, 104 holdout), for the three simulations.

can be accommodated in our framework by setting $\alpha_i(w)$ or $\gamma_i(w)$ to low values. Hence, we do not expect these parameters to be "recovered," given our addition of a churn process. However, importantly, the day-hour rate parameters should be relatively unaffected. Indeed, we find this to be true: across all three simulations, the correlation between true and estimated parameters was around 0.95 $\mu(j)$, and (on average) 0.75 for $\eta_i(j)$.

Fit and Forecasting Next, we look at how the rate of churn affects the ability of the model to explain and predict the data. In-sample, across all three simulations, the model performs well. Again, the model can address churn by setting the weekly scaling terms to be negative. However, out-of-sample performance depends on the overall propensity to churn. In the case of no churn, the out-of-sample performance is decent, and moreover, does not deteriorate over time. This is as predicted, given the model assumes stationary transaction processes; the forecasted GPs for the weekly scaling terms revert to their means over time, predicting future spending will occur at the historical average rate. Interestingly, when churn is high, the model also performs well: in this case, many people churn early, leading the weekly scaling terms to be very low throughout the calibration period, resulting in essentially a forecast of zero usage. In the middle case, with a modest rate of churn, the model's out-of-sample performance somewhat deteriorates: in this case, as people churn out-of-sample, the model's forecast of stationary usage rates cannot match reality, leading to larger errors in forecasting usage rates. We illustrate the week-over-week MAE of the three models together in Figure W-4.

C. Web Appendix C: Model with Covariates

The model presented in the main manuscript is general as it can be applied to a wide range of contexts and only requires transactional data, which are easily available to essentially any analyst. In some cases, the analyst might also have access to information such as firm interventions or other shocks in the service, and might be interested in exploring to what degree these changes relate to, and possibly drive, routine and non-routine usage. As a general model of usage timing, our framework is fairly flexible, and can be extended to incorporate covariates, in a number of ways.

Model Specifications

Let us denote the available covariates as x_{it}. These covariates could be at the weekly level (x_{iw}), or at the more granular week-day-hour level (i.e., x_{iwj}), or just at the customer level (i.e., x_i). For example, x_{it} could be an indicator capturing a promotion run by the company during week w, or a lagged variable capturing the quality of service the customer received during their last trip, or has received, on average, until that point.

First, recall the rate of our overall usage request process:

$$\lambda_{it} = \exp(\gamma_{iw} + \eta_{ij}) + \exp(\alpha_{iw} + \mu_j). \tag{W-1}$$

We can incorporate covariates in both terms, linearly or nonlinearly. For now, we focus on the linear case, and on incorporating covariates into the routine term. Intuitively, this specification corresponds to "covariate-driven routines": covariates may affect the degree to which customers adopt routines. To simplify the subsequent discussion, we also assume that the covariates are available at the weekly level, x_{iw}, although the same modeling approach can be taken for any type of covariate. With this assumption, the covariate-driven routines model is given by:

$$\lambda_{it} = \exp(\gamma_{iw} + \beta_R' x_{iw} + \eta_{ij}) + \exp(\alpha_{iw} + \mu_j). \tag{W-2}$$

In this covariates-driven routine model, the degree to which the day-hour rate η_{ij} governs behavior during week w is determined by both the weekly scaling parameter, γ_{iw}, and by the value of the covariates. In this sense, γ_{iw} captures the "residual" routine variation, week-over-week, modulo the covariate effect $\beta_R' x_{iw}$. While this specification is intuitive, it also changes how we think about smoothness and γ_{iw}: in this specification, γ_{iw} may be highly non-smooth, given it can only be interpreted relative to the covariate effect.

Caveats and Implementation

We now discuss important caveats to using the covariates model as well as details of its implementation, which might be helpful to analysts hoping to use this model in practice. First, as mentioned, the covariates extension of our basic model can be estimated with covariates that vary along any dimension. An alternative formulation could specify that covariates drive the degree to which random needs arise, rather than routines. Given our focus in this paper on understanding routines, we found the covariate-driven routines specification more natural.

Another caveat is that estimating the effects of covariates requires more data per-person: to achieve satisfactory convergence on real data, we had to restrict the data to customers who made at least five requests during the calibration period. While is not particularly onerous, it is more limiting than the no-covariates model presented in the main body of the paper, which typically does not have convergence issues, even in the presence of infrequent users.

Finally, forecasting with the covariates model is non-trivial. All of the estimated parameters are estimated *modulo* to the covariates. Given many covariates of interest will be dynamic, the analyst is thus forced to make forecasts for the value of the covariates, too. Practical strategies for doing so may include using simple lagged or average values, but these assumptions may be far from reality.

Application: Past Service Quality and Routineness

We estimated the covariate-driven routines model on the same data described in the main body of the paper. For the covariates, we used the lagged, standardized values of several *proposal-level* variables, in particular, the **cost per mile**, **expected wait time**, **trip time per mile**, and **walking distance** of the previous proposal (or, the average of all proposals in the previous session). As noted previously, to achieve satisfactory convergence, we limited the data to customers who had at least five sessions during the calibration period, resulting in 1,841 total customers. Given the use of lagged variables, we also only included trips for which we could observe the previous value of the covariates of interest. The benefit of using proposal-level variables is that they are recorded for (nearly) every session.[1]

[1]Every session, barring occasional technical glitches.

Table W-1: Estimated Covariate Effects

Posterior summary of the estimated covariate effects. The mean and SD refer to the estimated posterior distribution. HDI refers to highest (posterior) density intervals.

Variable	Mean	SD	HDI 2.5%	HDI 97.5%
Cost Per Mile	0.010	0.013	-0.014	0.033
Expected Wait Time	-0.049	0.012	-0.075	-0.028
Trip Time Per Mile	-0.004	0.014	-0.032	0.020
Walking Distance	-0.007	0.010	-0.025	0.012

First, the population-level parameters were estimated to be quite similar to the main model. This finding, by and large, makes sense: covariates should not alter, for instance, that behavior on weekdays tends to be more similar to behavior on other weekdays, and likewise for weekends. Nor does it seem to alter the general day-hour patterns and types of routines recovered by the model.

Recall that the new parameter of interest in this model is β_R, the effect of the covariates on the routineness term. We give the estimated values of β_R in Table W-1, specifically reporting posterior means, standard deviations, and 95% highest (posterior) density intervals (HDI). We see that, while most of the covariates have posterior means in sensible directions, the only covariate whose HDI excludes zero is expected wait time. The negative effect here suggests that higher past wait times are a key predictor of *lower* routineness in the future. This finding is reasonable: the company notes that long wait times are among the biggest factors driving customer satisfaction. Wait time is such a significant variable for Via that they often use wait time as a focal unit of analysis, by, for example, computing “wait time elasticities (of demand)” and thinking of the effects of different potential interventions, like discounts, in terms of “effective wait time.”

D. Web Appendix D: Additional Case Studies from the Quasi-Simulation

In this section, we present additional analyses for the 32 synthetic case studies in our quasi-simulation. We present them in three parts: first, we present a set of illustrative cases that highlight the model's ability to meaningfully decompose transactions. Then, we analyze the links between routine, clumpy, and regular behavior. Finally, we analyze data drawn from the Pareto-GGG (Platzer and Reutterer 2016) under our model.

Illustrative Cases Table W-2 describes the set of illustrative cases, in terms of how each simulation was generated, and Figure W-5 shows the model-based decomposition for each case. The figures are interpreted analogously as the "Decomposition" figures in the main manuscript. We corroborate that, in general, the correct insights are well recovered by the model. The only exception occurs in cases where usage is purely random. There, the model may attribute some of that random usage to a routine (e.g., cases 1 and 2). Also of note are cases 6 and 7, where the true data generating process was a mix of two routines: one routine in the first half, and a new routine in the second half. We see that even though the model has no mechanism to learn multiple routines it still classifies these customers as fully routine. However, the routine it learns is a mixture of the two, which is a limitation of our model.

Note that, for all of the simulations in Table W-2, the data generating process was deterministic. These same patterns can be recovered, even in the presence of noise. To illustrate, we included two additional cases, which mimic cases 3 and 4 from Table W-2 (i.e., high and low frequency routines), but where at each routine time, the customer uses the service probabilistically: with probability 0.5, they do not make a request; with probability 0.4, they make one request; with probability 0.095, they make two requests; and with probability 0.005, they make three requests. We plot the full results for these customers in Figures W-6 and W-7. As we can see, the model accurately parses both routines, although there is some fluctuation in the level of routineness, following the stochastic request process. In both cases, the model also accurately recovers the number of request times (5 and 2). For the expected requests, these are now fractional, again following the stochastic request process. In short: even in stochastic settings, the model has no issue meaningfully decomposing behavior.

Comparisons to Clumpiness and Regularity In addition to the cases shown previously, we also generated customers with clumpy and regular behavior. We now zero in on several examples

Table W-2: Illustrative cases.
Descriptions of the simulated customers.

Case	Label	Simulation procedure
1	Random (High)	Customer makes a request at 5 day-hours each week, randomly sampled each week from the empirical distribution
2	Random (Low)	Customer makes a request at 2 day-hours each week, randomly sampled each week from the empirical distribution
3	Routine (High)	Randomly sample 5 day-hours; customer makes a request at these times, every week
4	Routine (Low)	Randomly sample 2 day-hours; customer makes a request at these times, every week
5	Commuter	Customer rides every weekday at 8 AM, and 5 PM
6	Two Routines (High)	For the first 19 weeks, the customer follows a routine, generated as in Case 1; then the customer abruptly shifts to a new routine for the remaining 19 weeks, redrawing the times at which she requests rides
7	Two Routines (Low)	For the first 19 weeks, the customer follows a routine, generated as in Case 2; then the customer abruptly shifts to a new routine for the remaining 19 weeks, redrawing the times at which she requests rides
8	Random then Routine (High)	For the first 19 weeks, the customer follows the Random (High) procedure; for the last 19 weeks, the customer follows the Routine (High) procedure
9	Random then Routine (Low)	For the first 19 weeks, the customer follows the Random (Low) procedure; for the last 19 weeks, the customer follows the Routine (Low) procedure
10	Routine then Random (High)	For the first 19 weeks, the customer follows the Routine (High) procedure; for the last 19 weeks, the customer follows the Random (High) procedure
11	Routine then Random (Low)	For the first 19 weeks, the customer follows the Routine (High) procedure; for the last 19 weeks, the customer follows the Random (High) procedure
12	Random then Dead (High)	For the first 19 weeks, the customer follows the Random (High) procedure, then stops making requests
13	Random then Dead (Low)	For the first 19 weeks, the customer follows the Random (Low) procedure, then stops making requests
14	Routine then Dead (High)	For the first 19 weeks, the customer follows the Routine (High) procedure, then stops making requests
15	Routine then Dead (Low)	For the first 19 weeks, the customer follows the Routine (Low) procedure, then stops making requests

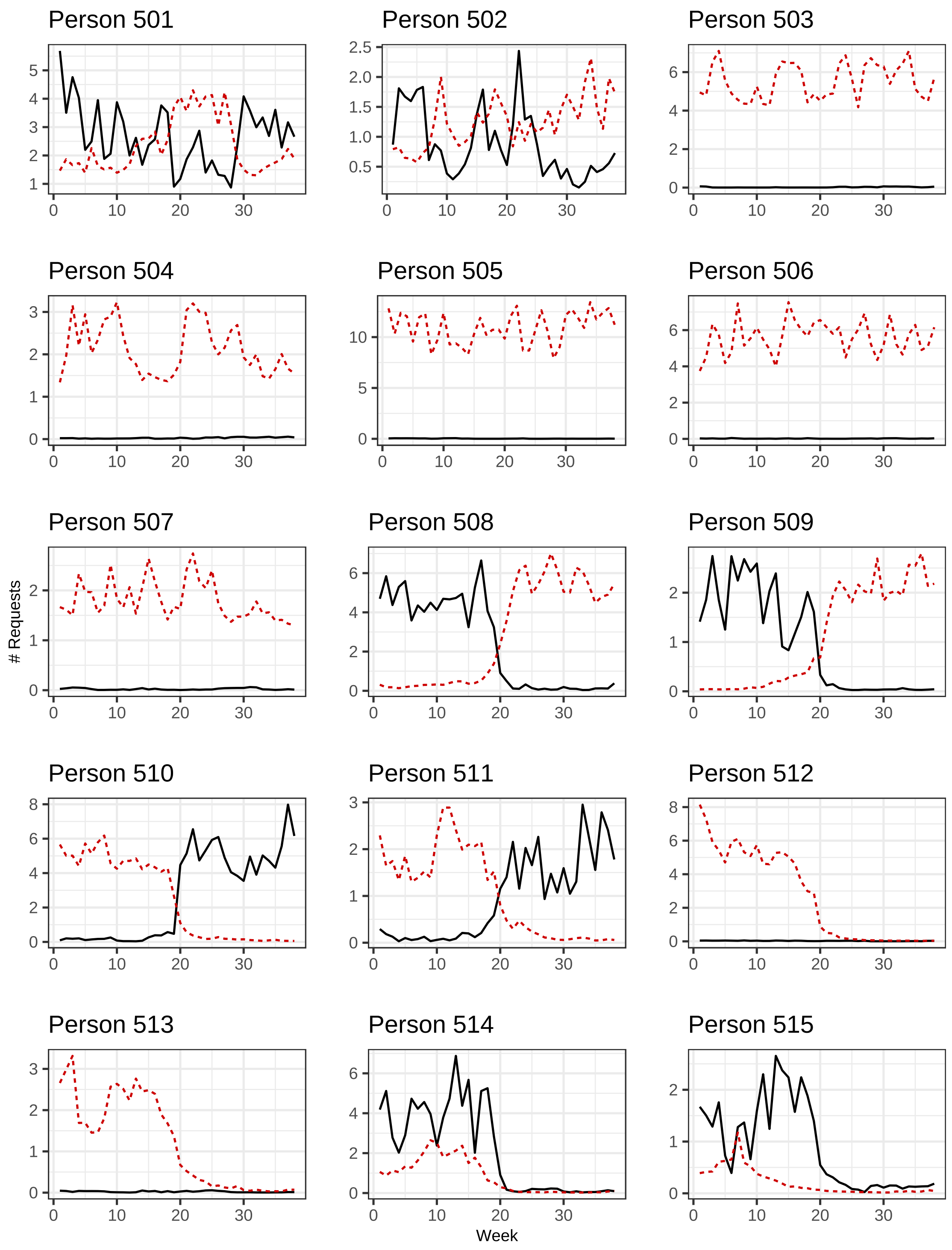


Figure W-5: Full simulation results.
The model-based decomposition for all 15 simulated cases, as described in the main body o f the paper and in Table W-2. The red dashed line is routine usage, while the black solid line is random usage. We see that, by and large, the model can correctly parse the correct data-generating pattern.

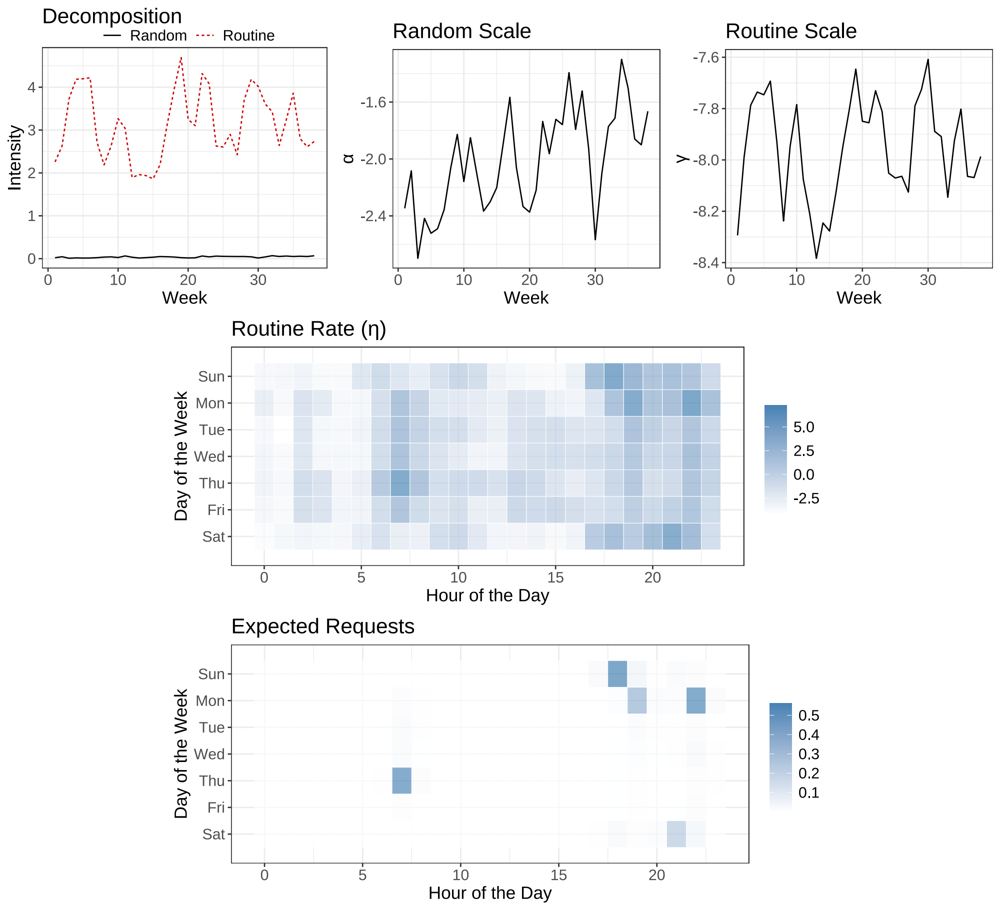


Figure W-6: Noisy version of Case 1

The model-based decomposition and associated parameters for a noisy version of Case 1, i.e., a high frequency routine.

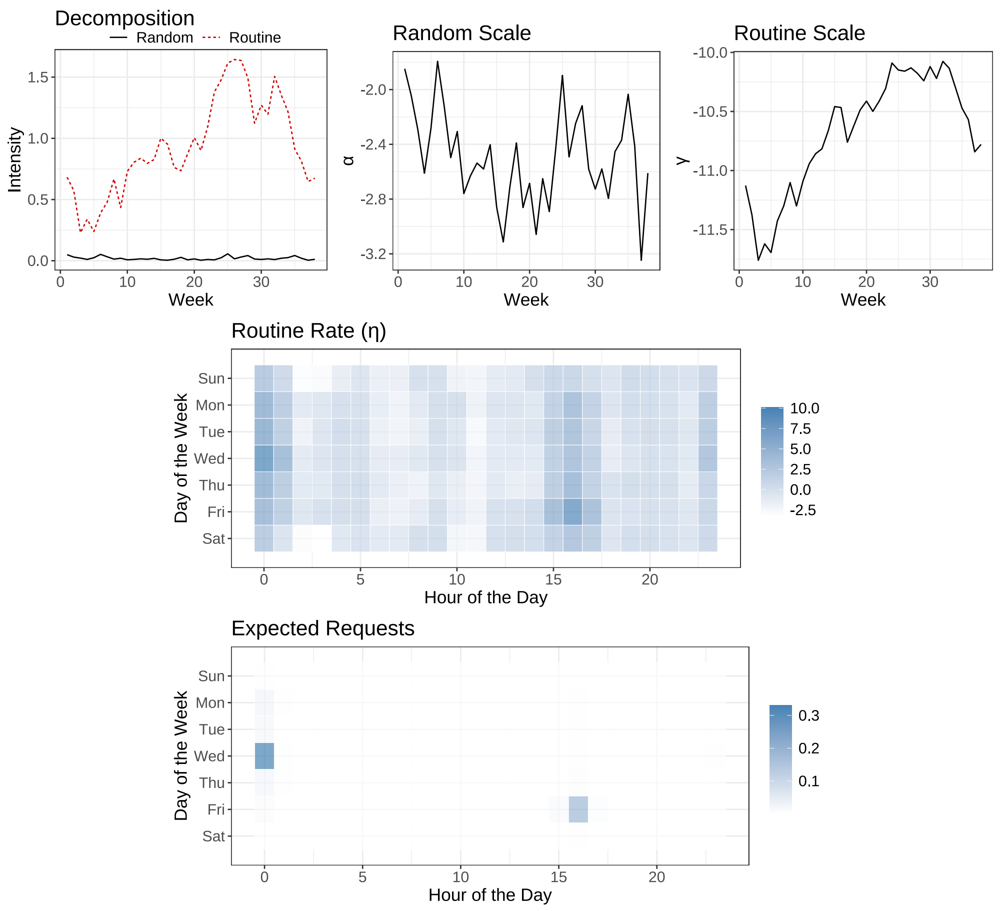


Figure W-7: Noisy version of Case 2

The model-based decomposition and associated parameters for a noisy version of Case 2, i.e., a low frequency routine.

of those cases, to illustrate the connections between routines, clumpiness, and regularity. In particular, we first focus on the synthetic cases that inspired Figure **??** (in the main body). As described in the paper, routines can also generate clumpy or regular behavior. In fact, they can also exhibit regular clumps. To illustrate these patterns, we generated several additional customers, featuring clumpy and/or regular behavior. The first is a customer who does not use the service during the week, but always makes requests on the weekend, at 10am, 5pm, and 11pm — this user is high on clumpiness. Then, we generated another user who always transacts at 8am, every day — very regular. To validate that these cases are clumpy and regular, for clumpiness, the first customer has a clumpiness score of 0.17 versus almost zero for the second. In terms of regularity, the first customer always has identical intertransaction times, while the second does not. Figures W-8 and W-9 show our model's estimation of these customers' behavior, respectively. The model identifies that both of these cases have high routineness: the decomposition attributes all of their behavior to a routine. In short, this illustrates that routines can generate both clumpy and regular behavior.[2]

Perhaps more interestingly, we can also generate regularities that are not, exactly routine. Imagine, for instance, a customer who makes a clump of transactions every 40 hours. Now, this is admittedly rather strange transaction behavior, but it is interesting to see how our model decomposes such a pattern. The 168 day-hours in a week are not evenly divisible by 40, so this behavior does not lead to a consistent week-over-week behavior. However, it is, in some sense, equivalent to a probabilistic routine, at all hours that are divisible by 40. We see that is exactly what the model recovers, presented in Figure W-10. Similarly, we can also generate clumpy behavior where the timing of the clumpy behavior is totally random. In this case, the model correctly parses that the behavior is not routine at all, and attributes the majority of usage to the non-routine process.

[2]We have additional case studies that are variations on this theme, including stochastic timing, and lower transaction rates, which have results consistent with these case studies, and which all can be shared upon request.

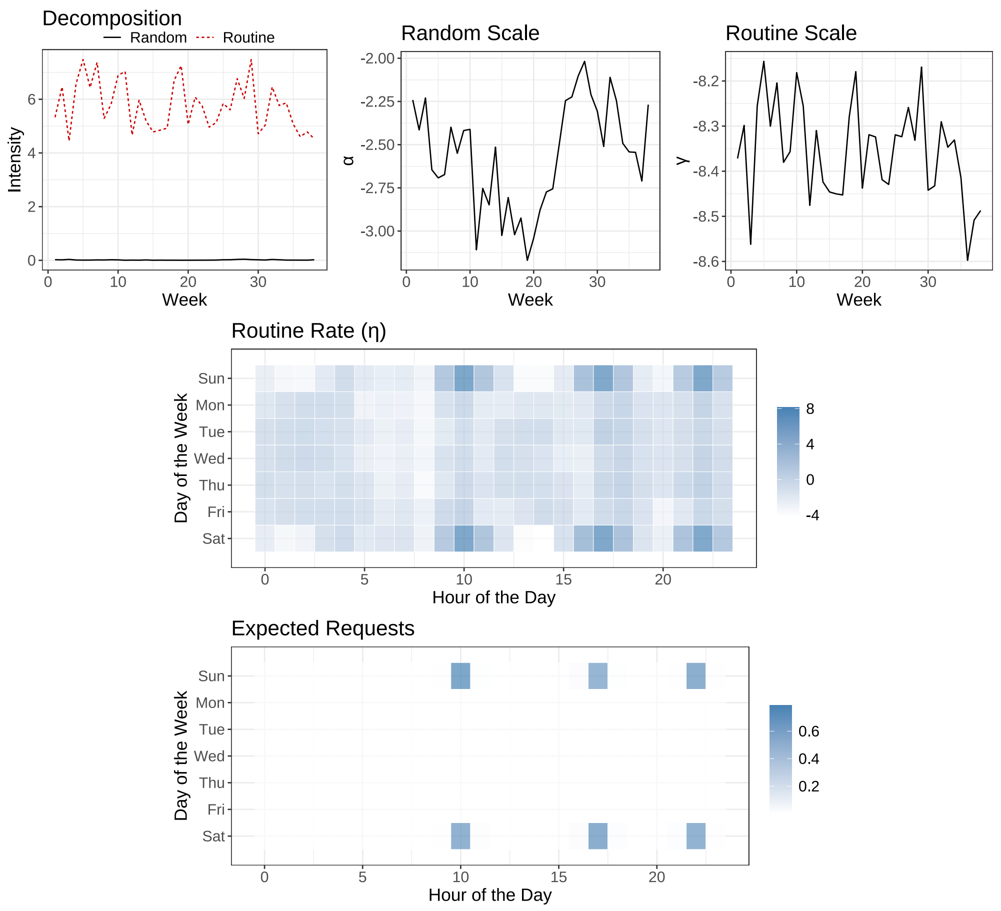


Figure W-8: Routine and Clumpy

The model-based decomposition and associated parameters for a case where behavior is routine and clumpy.

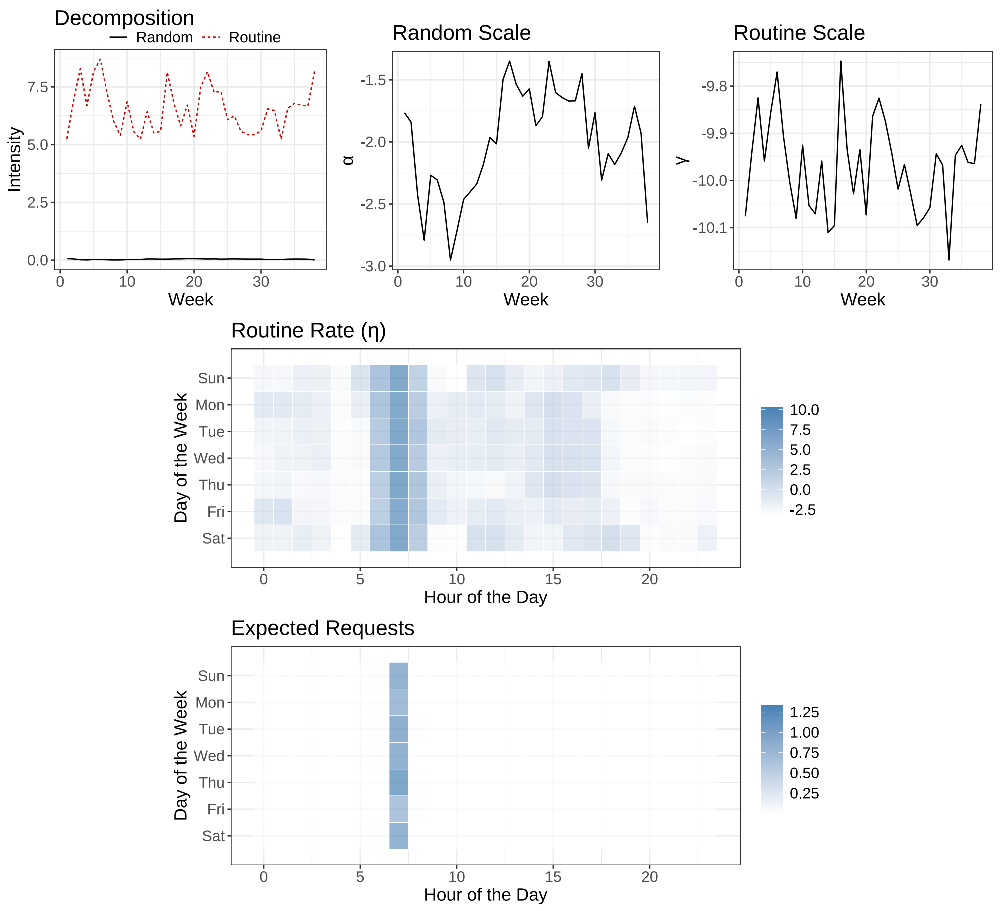


Figure W-9: Routine and Regular

The model-based decomposition and associated parameters for a case where behavior is routine and regular.

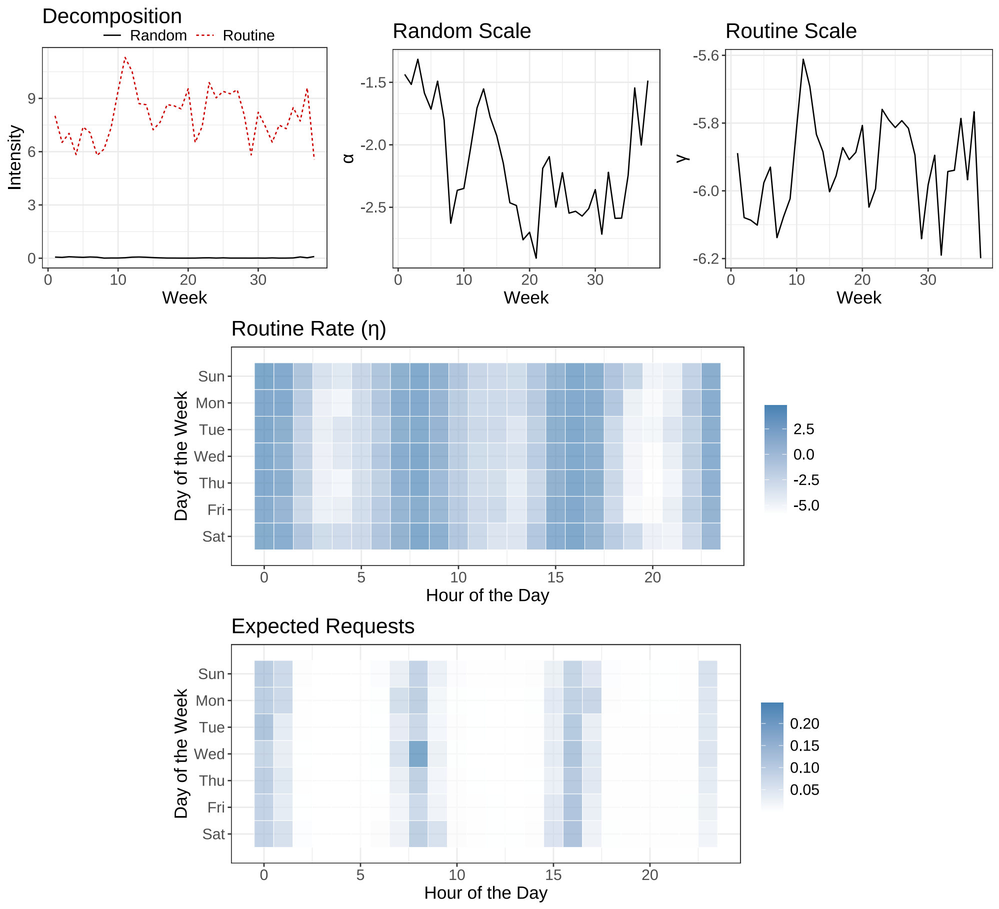


Figure W-10: Clumpy with a 40 hour cycle.
The model-based decomposition and associated parameters for a case where a clump of transactions occurs roughly every 40 hours.

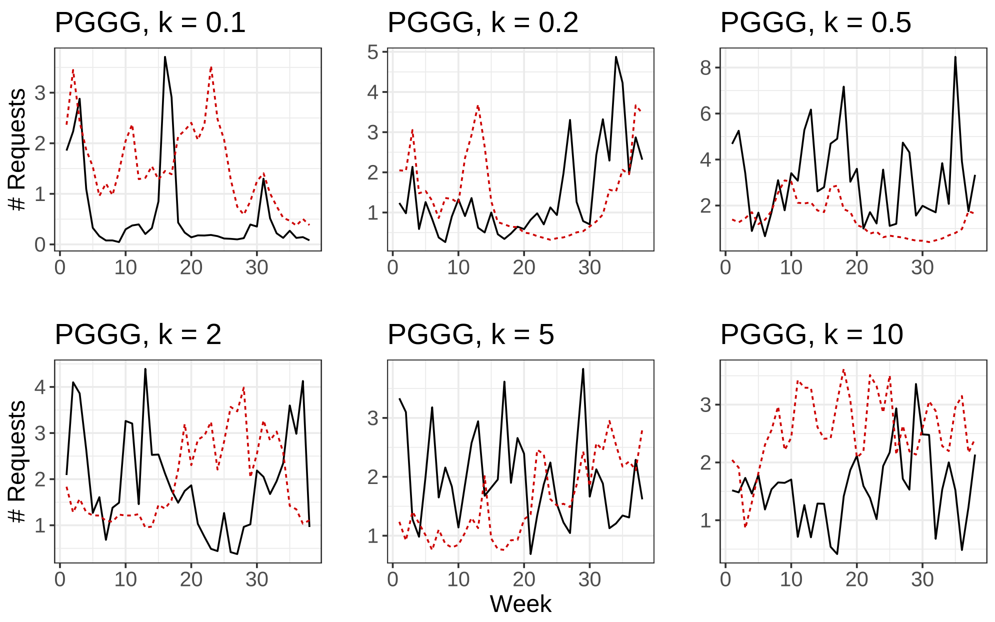


Figure W-11: Six Pareto-GGG Case Studies
The model-based decompositions for six customers generated from the Pareto-GGG model (with no churn). Similar to prior plots, the red dashed line is the routineness, while the black solid line is the non-routine ("random") process.

Pareto-GGG Finally, we generated a set of customers from the Pareto-GGG model with no churn. In essence, this is equivalent to gamma-distributed interarrival times, with individual-specific rates k_i, following the specification given in Platzer and Reutterer (2016). Specifically, we generated six synthetic customers, varying $k_i = 0.1, 0.2, 0.5, 2, 5, 10$, ranging from very clumpy to very regular. Intuitively, we would hypothesize that, for most values of k, the model would classify these consumers as non-routine. Only when k grows very large, and the interarrival times become extremely regular, should we see some routineness emerge. Figure W-11 shows the results of the decomposition, applied to our six synthetic cases. The results are consistent with our hypothesis: as k rises, more usage is attributed to a routine, especially in the case of $k = 10$ which corresponds to very regular transaction patterns, akin to our "every day at 8am" consumer from before. For values lower than $k = 10$, we find that much of the usage is attributed to the random process. Occasionally, we see a bit of routine usage: this can be attributed mostly to stochasticity producing similar weekly transaction times.

E. Web Appendix E: Details of Clustering Routines

To cluster routines into routine types, we used the `latrend` package in the `R` statistical programming language (Den Teuling 2023), which is specifically designed to cluster longitudinal data. In particular, we used the parametric longitudinal k-means method (`lcMethodLMKM` in `latrend`), which has been found to provide superior performance compared to traditional longitudinal K-means methods for trajectory clustering (Den Teuling, Pauws, and van den Heuvel 2023). Regarding the selection of number of clusters, we estimated the model with $K = 1, \ldots, 12$ clusters and examined the WMAE (weighted mean absolute error) as measure of fit (Figure W-12).

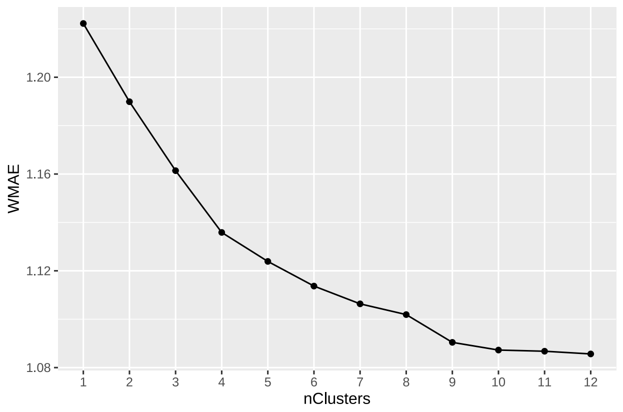
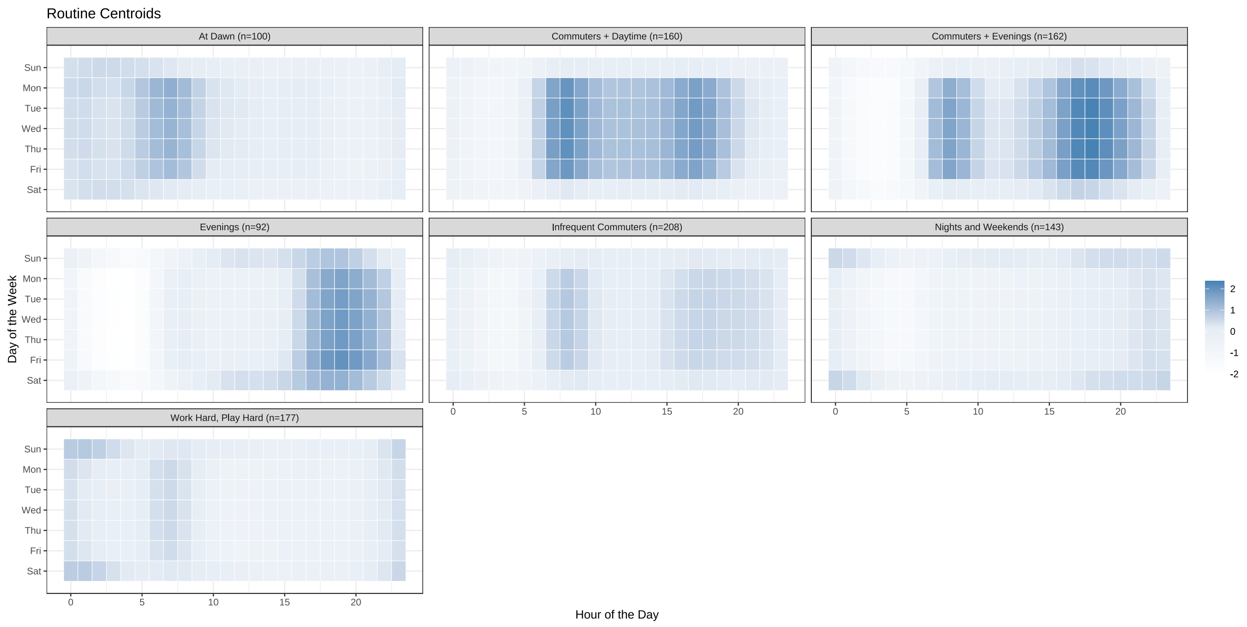
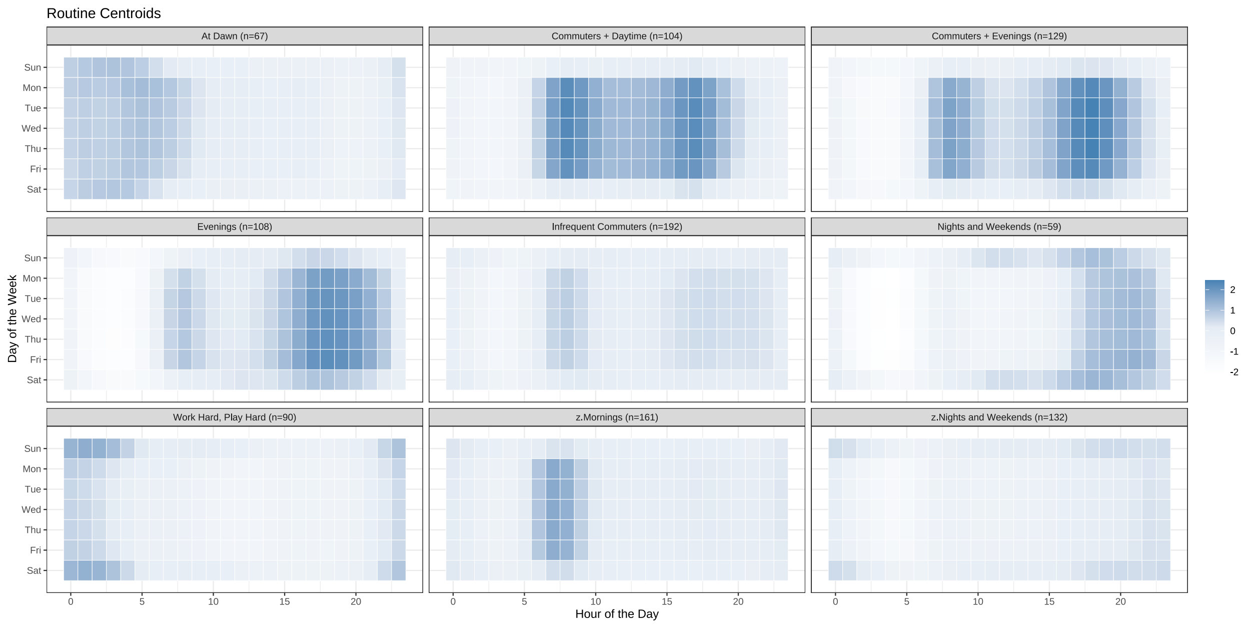


Figure W-12: Clustering Analyses: Selecting the Number of Clusters
WMAE (weighted mean absolute error) of each cluster solution, with clusters varying from $N = 1, \ldots, 12$.

Based on WMAE, there were two solutions that seemed reasonable, which corresponded to the 7-cluster and the 9-cluster solutions. We compared the individual clusters across them (Figures W-13a and W-13b) and corroborated that most clusters (Commuters + Daytime, Evenings, Infrequent Commuters, Commuters + Evenings, At Dawn) remained largely unchanged. Compared with the 7-cluster solution, the 9-cluster solution seemed to split two clusters (Nights and Weekends, Work Hard, Play Hard), providing more nuances to the day-hour patterns of those two clusters. Given the consistency across both solutions, we chose the solution with 7 clusters for simplicity.



(a) 7-Cluster Solution



(b) 9-Cluster Solution

Figure W-13: Comparing 7-cluster and 9-cluster Solutions
For easy comparison, clusters 1 through 7 are in the same order across the figures. The last two clusters in Figure W-13b are the new clusters that emerged from the 9-cluster solution.

F. Web Appendix F: Details of the LSTM Benchmark

As a benchmark, we trained long-short term memory deep learning models (Goodfellow, Bengio, and Courville 2016; Sarkar and De Bruyn 2021) with the goal of predicting in which day-hours a customer would request a ride. We evaluated two LSTM modeling frameworks: one in which we trained one small LSTM per user, and one in which we trained a single large LSTM to predict across all users. With appropriate tuning of the architectures, we found the small individual-level models performed better, and thus, we focus on that framework below, and include its statistics in the paper. In all cases, the loss function was a binary cross-entropy loss, focused on predicting if there would be a ride request during each of the 168 day-hours in a week. As training data, we used the same 38-week window as the other models. The specific architecture fed a sequence of 3-weeks of data through an LSTM layer with 32 hidden units and tanh activation functions, the output of which was flattened and fed through a dense layer, again with 32 hidden units. The output is a sequence of probabilities over the subsequent week, predicting the likelihood of seeing a ride during each day-hour, for that customer.

To forecast with this architecture, we need to use the same sliding window (three weeks in, one week out). Thus, the forecasting task is slightly different than in the other models, which forecast 10 weeks ahead, given 38 weeks of data. In the LSTM, while the parameters are learned using the full 38 weeks of training data, to actually make a prediction, we use the prior 3 weeks of data to predict what the subsequent week will look like. Using this mechanism to forecast more than one week ahead thus requires us to assume the predictions are true, and feed them back in as part of the subsequent input sequence. In turn, this requires assuming a threshold for the forecasted probabilities, above which we will assume a ride takes place. The threshold we chose is a probability of 0.2. We chose this relatively low threshold for two reasons: first, on the training data, we observed that probability predictions were rarely above the more standard cut-off of 0.5. Moreover, as shown in the paper, the choice of 0.2 gave good performance on our focal task, which again was ranking likely day-hours for rides. Note, though, that the combination of this forecasting mechanism and this threshold choice leads to a high number of forecasted requests: uncertainty is propagated through the sliding window, as prior predictions are treated as truth, and there is a relatively low threshold for assuming a ride occurs. It may be possible to further optimize this choice of threshold to achieve good performance on *both* tasks, or to change the loss in a way that naturally captures both volume and timing. We put a reasonable amount of effort into carefully building this architecture to do a good job at the day-hour ranking task, to ensure we are comparing our model to a properly developed benchmark. We view further optimizations as beyond the scope of this work.

G. Web Appendix G: Churn and Stationarity in Ridesharing Behavior

As discussed in both the main body of the paper and in Appendix B, while the proposed model for routines can capture changing rates of usage over time through the parameters $\alpha_i(w)$ and $\gamma_i(w)$, it does not explicitly capture latent attrition (i.e., churn). In standard customer base analysis models, like the Pareto-GGG benchmark, the possibility of churn is captured by a latent attrition specification that specifically allows for a customer to move from an active state to a churned or "dead" state, wherein purchases stops forever. In contrast, our model can "zero out" purchasing by setting $\alpha_i(w)$ and $\gamma_i(w)$ to be low values, thus suggesting zero purchasing (i.e., the customers become "inactive"). We model these trajectories using Gaussian processes, which notably, eventually revert to their means when forecasting far from the range of the data. In our model, the means of these functions are individual-level constants reflecting the average level of non-routine and routine behavior (α_{0i} and γ_{0i} respectively). The rate at which the trajectories revert to their means is determined by the GP kernel's lengthscale parameter (ρ_α and ρ_γ respectively). This mean reversion feature suggests that our model will work best when individual-level purchasing (or, in our application, requesting) is relatively stationary; that is, if, in the long run, rates of usage roughly mirror past rates of usage.

In our simulations in Appendix B, we showed forecasting performance of our model is impoverished when there is a moderate level of churn. In this section, we investigate to what degree churn rates, and the lack of an explicit attrition process, may be a problem in our empirical application. To do so, we consider forecasting future number of requests over our 10-week holdout period across different subsets of customers.[3] Specifically, we consider four different subsets of customers:

1. **High vs. low variability:** We look at variance in customer-level in-sample purchasing rates, and perform a median split, dividing customers into low variability and high variability groups.

2. **Zero vs. non-zero holdout purchasing:** Within our 10-week validation window, we look at whether or not customers made any purchases.

[3]Note that our relatively short 10-week holdout period may limit our ability to find a substantial impact of churn: in contrast to typical customer base analysis applications, 10 weeks is a relatively short holdout period. Moreover, the lengthscale of the GP will limit the amount of mean reversion that is seen over short forecasting windows.

3. **Maybe churned, 5-week windows:** We look at whether customers made any requests in the 5 weeks before the end of the training window, and compare that to whether they made any requests in the 5 weeks before that. Intuitively, if a customer made no requests in the most recent 5 weeks, but did request rides in the 5 weeks prior to that, this suggests that they may have churned. Thus, we divide customers in this way, with "maybe churned" customers being those that did not make requests in the most recent 5 weeks, but did make requests before then.

4. **Maybe churned, 10-week windows:** This split is the same as the previous one, but using 10-week windows instead.

Then, we evaluate the mean absolute error in forecasting the number of requests across the two conditions in each split. We compare the performance of our model to the Pareto-GGG.

The full results of the analysis are shown in Table W-3. Across all splits, we find very small performance differences across the two models. Starting with high vs. low variability, the MAE of our model is 4.54 for low variability customers, versus 10.91 for high variability customers. In comparison, for the Pareto-GGG, the MAEs are 4.01 and 11.02, respectively. Turning to zero holdout requests, the MAE of our model is 2.25 for people who made zero holdout requests, versus 1.98 for the Pareto-GGG, suggesting our model is able to identify people who have become inactive. For "maybe churners" computed with the 5-week window, the MAE of our model is 3.69, versus 3.45 for the Pareto-GGG, and for "maybe churners" computed with the 10-week window, the MAE of our model is 2.69, versus 2.46 for the Pareto-GGG. Across these splits, we can see that the Pareto-GGG often improves on the proposed model, but the improvements are marginal. These results suggest that the purchasing processes in our data are relatively stationary, such that the lack of an explicit latent attrition process is not a substantial limitation for our model's forecasting performance.

Splitting Criteria	Our Model	Pareto-GGG
Split 1: Low vs. High Variance in Requests		
Low Variance	4.54	4.01
High Variance	10.91	11.02
Split 2: Made a Request During Holdout		
No	2.25	1.98
Yes	9.60	9.40
Split 3: Maybe Churned (5 Weeks)		
No	8.38	8.17
Yes	3.69	3.45
Split 4: Maybe Churned (10 Weeks)		
No	8.36	8.18
Yes	2.96	2.46

Table W-3: MAE Across Splits and Models
A comparison of model MAE across the four different splitting criterion, and the two models, showing a minimal difference in forecasting performance over our 10-week holdout period.

H. Web Appendix H: Calculating CP and MAP

In our discussion of our model's ability to predict ride times, we employed the conditional precision (CP) and mean average precision (MAP) statistics. We now describe how to compute those statistics. The basis of both of CP and MAP is the *top-k precision*, denoted $p(k)$, which is computed as follows: Let T denote the set of day-hours that a customer requested rides in a given week, and let R_k denote a ranking of the k most likely day-hours for that customer to request a ride, as predicted by the model. Here, we are ignoring i and w subscripts for notational simplicity: in our application, this ranking would be computed on a per-customer, per-week basis. With this notation, top-k precision is given by $p(k) := |R_k \in T|/k$, which, simply put, captures the fraction of the day-hours in R_k in which the customer actually requested a ride. As a running example, let us consider a person who took four rides, i.e., $y = |T| = 4$. Suppose those rides happened at day-hours $T = \{11, 22, 33, 44\}$. Suppose then that the model's top-6 predicted ride times were $R_6 = \{11, 22, 32, 33, 45, 44\}$. Then $p(1) = 1$ (since the top-ranked ride actually happened), $p(2) = 1$ (since both of the top-2 rides happened), but $p(3) = 2/3$ (since the rank-3 ride did not happen), and so on.[4] Having defined $p(k)$, we can now define our two metrics of interest:

- **Mean Average Precision (MAP):** To define MAP, we first define the average precision (AP) of the day-hour rankings for a given customer (again omitting the i and w subscripts):

$$AP = \frac{1}{y} \sum_{k=1}^{168} p(k)\, \mathbb{I}(\text{Request} @ \text{k}). \tag{W-3}$$

  Here, $\mathbb{I}(\text{Request} @ \text{k}) = 1$ if the user made a request at the day-hour ranked k and 0 otherwise. The MAP is then the mean AP across all customers. To illustrate the intuition behind MAP, let us return to our example with true ride times $T = \{11, 22, 33, 44\}$, and $R_6 = \{11, 22, 32, 33, 45, 44\}$. The average precision for this user would be $\frac{1}{4}(1 + 1 + 0 + \frac{3}{4} + 0 + \frac{4}{6} + 0 + \dots) = 0.854$. The AP is always between 0 and 1, with higher values indicating that the model is producing better rankings of the day-hours for that customer. The AP will be 1 if all of the user's rides happened during the highest ranked hours.

- **Conditional Precision (CP):** The conditional precision is similar to the classic precision metric. Suppose that we know a given user rode y times in a given week; the conditional preci-

[4] Note that this metric does not separate adjacent mispredictions (e.g., if the true day-hour of a ride were 33, a prediction of day-hour 34 or 14 would both count as mispredictions). However, note that our model is likely to rank similarly adjacent day-hours due to the correlation between days and hours induced by the day-hour kernel.

sion captures which of those y day-hours the model predicts correctly. Mathematically, CP is equal to $p(y)$.

Note that, while these metrics all share the word “precision” in their names, they are connected to both precision and recall, in the classic senses of those terms. MAP, in particular, can be viewed as measuring the area under the precision-recall curve. As an example, imagine a customer who made two requests, one of which was expected by the model and ranked first, and another which was unexpected and ranked last. The MAP of this scenario is $\frac{1}{2}(1 + \frac{1}{168})$, which takes into account both the successful ranking of the first request, and the very unsuccessful ranking of the second request.

I. Web Appendix I: Additional Summary Statistics

Table W-4: Additional summary statistics.
Summary statistics for the ride-related covariates. The variables above the horizontal line are variables about the proposal itself; those below the line are about the actual trip that was taken.

Statistic	N	Mean	St. Dev.	Min	Max
Requests Before	38,305	0.9	3.0	0.0	103.0
Ride Distance	38,305	3.3	2.5	0.1	27.9
Airport Ride	38,305	0.03	0.2	0.0	1.0
Solo Trip	38,305	0.9	0.3	0.0	1.0
Had ViaExpress Proposal	38,305	0.4	0.5	0.0	1.0
Had Shared Taxi Proposal	38,305	0.04	0.2	0.0	1.0
Sedan	38,304	0.3	0.4	0.0	1.0
Van	38,304	0.3	0.5	0.0	1.0
Ride Cost (Cents)	38,167	879.1	878.1	0.0	11,936.0
Driver ETA	38,305	7.7	4.0	0.1	70.1
ETA Destination	38,305	31.1	13.5	0.1	168.1
Speed	38,305	0.1	0.7	0.01	131.1
Pickup Walking Dist.	38,305	109.2	83.5	0.0	600.0
# Passengers Request.	38,305	1.2	0.5	1	6
Pickup Delay	16,159	1.1	2.7	−12.2	48.3
Dropoff Delay	16,159	2.6	8.9	−41.4	630.0
Dropoff Walking Dist.	19,374	88.8	70.2	0.0	577.0
# On-board (Pickup)	16,159	0.8	1.0	0.0	5.0
# On-board (Dropoff)	16,159	0.7	1.0	0.0	5.0
Max On-board	16,159	2.5	1.3	1.0	7.0

J. Web Appendix J: Additional Behaviors by Routine Type

In Figure W-14, we show how six different behaviors vary across our routine types. The specific behaviors are:

- Airport Rides – whether a request is to the airport or not
- Pickup Walking Dist – how long customers have to walk to get their shared ride
- Proposed ETA – how long the customer is expected to wait until the driver arrives, as part of the ride proposal
- Proposed Speed – calculated by looking at the requested trip length, and the proposed trip time
- Ride Cost (cents) – how much the customer will pay for the trip
- Solo Trips – how often the customer requests a trip just for 1 passenger

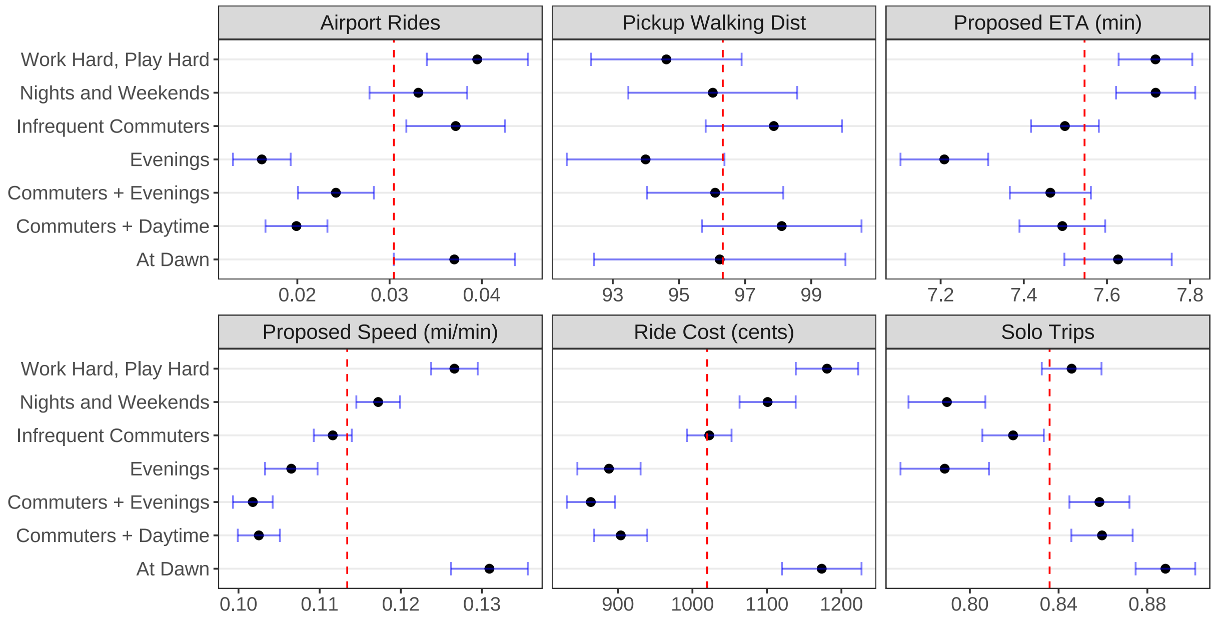


Figure W-14: Request characteristics, by routine type
The average value of 6 proposal-level variables, by routine type. Error bars represent standard errors, and the vertical line represents the overall mean.

K. Web Appendix K: Which Customers Have Routines

In this appendix, we consider which types of customers, in terms of observable characteristics, tend to develop routines. Understanding what makes for a routine customer can further validate our routineness metric, and suggest ways in which the focal firm might consider acquiring routine customers, or cultivating routines. To explore this phenomenon, we regress each user's week 38 routineness on a number of variables describing that user's activity and trip types averaged over the training period.[5] The results are shown in Table W-5.

Table W-5: Predictors of routineness.
Regression of week 38 routineness (DV) on average trip characteristics (IVs), where the average is taken over the whole training window. (These results are robust if we use the average routineness as the DV instead of its week 38 value.)

	Dependent variable:
	Routineness
Prob. Ride | Request	0.555^{***}
	(0.116)
First Week	-0.0003
	(0.001)
# Requests	0.401^{***}
	(0.008)
Price	-0.001^{***}
	(0.0001)
ETA Driver	0.047^{**}
	(0.019)
ETA Destination	-0.001
	(0.005)
# Passengers Req.	-0.131^{**}
	(0.061)
Walking Distance	-0.001^{*}
	(0.001)
Distance	0.116^{***}
	(0.025)
Airport Ride	-0.033
	(0.269)
Observations	2,000
R^2	0.605
Note:	$^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$

We find that high-routineness users have much in common: first, echoing our main CRM results, we see that routine customers have a higher probability of accepting a ride, given a request. We also find that customers who take longer trips, or request a ride with fewer other passengers

[5] The results are very similar when we consider the average routineness over all weeks as a DV. We prefer using week 38 routineness for consistency with the rest of the paper.

(i.e., are more likely to request a “solo trip,” as opposed to bringing friends along for the ride) are more likely to have higher routineness. Partly, this may be explained by the prevalence of commuting routines, which intuitively may be more likely to be solo trips, and may be from more remote areas of the city to more central ones. A similar self-selection story may explain the positive association of routineness and driver ETA: if routine customers are traveling at peak hours, it may take longer to find a driver. More interestingly, we find that routineness is associated with lower priced trips, and lower walking distance. While our analysis is not causal, these effects are suggestive: customers who are consistently confronted with high prices or high walking distances may stop using the platform, or never form routines.

L. Web Appendix L: “When” versus “What”: Incorporating Location Information

In the main body of the paper, we focused exclusively on temporal routines: that is, *when* someone interacts with the firm, not *what* they do in that interaction. In a retail setting, for instance, a customer may come in at exactly the same time each week, but may buy either the same items each time, or different items. In our focal context, customers may request rides at exactly the same times each week, but may travel to either the same location each time, or different locations. Consider, for example, two work commuters, both of whom work in the same location each day. In the morning, both users may always go between the same locations, home and work. In the evening, however, one of these commuters may always return home, while the other frequently goes out for drinks or dinner. In this sense, both customers have the same “when” routine but different “what” routines. In this section, we explore to what degree “what” routines — that is, location choice — are predictive of “when” routines, and what gains there may be in accounting for “what” routines, in addition to our previously defined routineness measure.

Metrics for Location Dispersion To understand the degree to which there are “what” routines in location choice, we first need a metric of how consistent location choices are. Mathematically, it is more natural to construct measures of how dispersed (that is, how *in*consistent) trip locations are.[6] To understand location dispersion, we look at both pick-up and drop-off locations. In our data, locations are saved as precise latitude/longitude coordinates (or, “lat/long”), measured to five decimals. To discretize our location data to correspond to New York City street blocks, we truncate the decimal to the nearest 300th.[7] With this discretization, we then define two measures of location dispersion:

1. **Shannon Entropy:** For this metric, we consider the empirical distribution of a user’s locations (both pick-up and drop-off). For instance, if a user made ten total trips, nine to location 1 and one to location 2, the empirical distribution would be (0.9, 0.1). We then compute the Shannon entropy of that distribution, defined as:

$$\text{Entropy} = -\sum_{\ell=1}^{L} p_k \log p_k. \tag{W-4}$$

[6]Though each of our measures could be easily converted to a consistency measure by inversion.
[7]Specifically, given a raw coordinate x, we compute a truncated coordinate, $x^* = \lfloor 300x + 0.5 \rfloor / 300$.

where p_ℓ is the empirical probability of the ℓth location, and L is the total number of locations. Intuitively, entropy captures how "predictable" a user's locations are. To illustrate, observe that $\text{Entropy}(0.9, 0.1) < \text{Entropy}(0.5, 0.5) < \text{Entropy}(0.1, 0.1, \ldots, 0.1)$.

2. **CRT Dispersion:** A feature of the entropy measure is that it does not take into account the total number of trips or locations for a given user. Hence, a user that takes 1,000 trips to the same 10 locations is just as entropic as a user who takes 10 trips to the same 10 locations. Hence, as an alternative to entropy, we consider a metric inspired by the Chinese Restaurant Table (CRT) process. The CRT process is a stochastic process, derived from the better known Chinese Restaurant Process, commonly used to model assignment to different groups (Zhou and Carin 2013). It captures a "rich get richer" process, whereby new observations are either assigned to existing groups, with probability proportional to the sizes of those groups, or they are assigned to a new group, at a rate proportional to a dispersion parameter. Such a process could be used to model the evolution of trip location choices: for a new trip, that trip may be to an existing location, or it may be to a new location. While estimating the full CRT model is complex and cumbersome, there exists an intuitive and easily computed estimator of the CRT dispersion parameter, θ (Durrett 2008, Chapter 1): given L unique locations in K total trips:
$$\theta \approx \frac{L}{\log K}. \tag{W-5}$$
In plain English, this metric is the number of unique locations divided by the log number of total trips. Contrasting this to entropy, if a user has just two unique locations in ten trips, $\theta = 2/\log(10) \approx 0.87$, regardless of whether those trips were split 5/5 or 9/1. But, unlike entropy, if a user visits 10 locations equally among 1000 trips, $\theta = 10/\log(1000) \approx 1.45$, whereas if the user visits 10 locations equally among 10 trips, $\theta = 10/\log(10) \approx 4.34$.

Location Results Having defined metrics of location dispersion, we now ask: first, to what degree is location dispersion related to temporal routineness? And second, is location choice also an important predictor of customer-level outcomes? To answer the first question, we plot the joint distribution of our routineness metric and the two metrics of location dispersion in Figure W-15.[8] We see that there is no obvious relationship between temporal routineness and location dispersion. This is supported by simple regression analyses: alone, entropy and CRT dispersion explain less

[8]For consistency, we focus on routineness as measured in the last week of our training data, but all the results are the same if we consider average routineness over the training data instead.

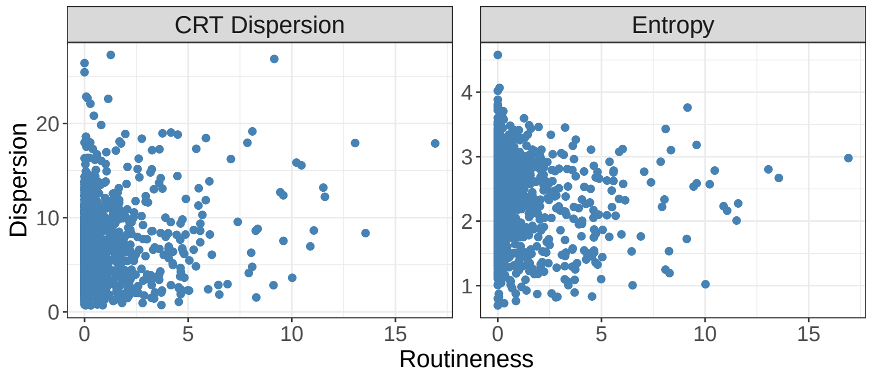


Figure W-15: Joint Distributions of Routineness and the Two Location Dispersion Metrics. *The joint distribution of routineness in week 38 (i.e., E_{i38}^{Routine}), and the two metrics of dispersion in location choice, entropy and CRT dispersion, showing no relationship between the two.*

than 5% of the variation in routineness.

When we regress routineness on both location metrics, together with other obvious individual-level controls (e.g., number of final week requests, the week the customer was acquired), we find CRT dispersion is a negative and significant predictor of routineness, while entropy is not significant. We report the full results from these regressions in Table W-6. Taken together, these results suggest that location dispersion is minimally predictive of temporal routineness, albeit in an intuitive direction: customers with less dispersed location choices seem to be slightly more routine.

Having established that routineness and location dispersion are distinct, we now ask: does location dispersion have an impact on customer behavior over and above temporal routineness? Specifically, we return to the focal analyses where we explored the role of routineness in explaining the number of future requests and the likelihood of a customer being active in the future, but now also include our location dispersion metrics. We find that, when the location dispersion metrics are included alongside routineness, neither entropy nor CRT dispersion is a significant predictor of either outcome, suggesting that, from a CRM perspective, "when" matters significantly more than "what." We display these results in Table W-7.

Table W-6: Explaining routineness with location dispersion.
Regression of week 38 routineness (DV) on the two measures of location dispersion, along with some standard customer-level controls (IVs).

	Dependent variable:		
	Routineness		
	(1)	(2)	(3)
Entropy	-0.220^{***}		-0.128^{*}
	(0.060)		(0.071)
CRT Disp.		0.081^{***}	-0.036^{***}
		(0.009)	(0.014)
Sessions			0.005^{***}
			(0.0004)
Prob. Ride | Request			0.115
			(0.124)
First Week			0.002^{**}
			(0.001)
# Requests ($w = 38$)			0.453^{***}
			(0.010)
Observations	2,000	2,000	2,000
R^2	0.007	0.036	0.684
Note:			$^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$ Intercept omitted for clarity.

Table W-7: Regressions of location dispersion and customer activity.
Regression of future activity (DVs) on individual-level summary statistics and the two measures of location dispersion.

	Dependent variable:	
	# Requests	Activity
	OLS	*logistic*
	(1)	(2)
Requests $(w = 38)$	1.812***	0.398***
	(0.236)	(0.104)
Recency	−0.227***	−0.141***
	(0.042)	(0.010)
Frequency	0.090***	−0.0002
	(0.009)	(0.003)
Routine $(w = 38)$	5.280***	1.069***
	(0.385)	(0.333)
Entropy	−1.848*	0.095
	(1.082)	(0.240)
CRT Disp.	0.108	−0.017
	(0.197)	(0.047)
Observations	2,000	2,000
R^2	0.575	
Note:	*p<0.1; **p<0.05; ***p<0.01	
	Intercept omitted for clarity.	

References

Dang, Khue-Dung, Matias Quiroz, Robert Kohn, Tran Minh-Ngoc, and Mattias Villani (2019), "Hamiltonian Monte Carlo with energy conserving subsampling," *Journal of Machine Learning Research*, 20, 1–31.

Den Teuling, Niek *latrend: A Framework for Clustering Longitudinal Data* (2023) `https://github.com/philips-software/latrend`, r package version 1.5.1.

Den Teuling, Niek G. P., Steffen C. Pauws, and Edwin R. van den Heuvel (2023), "A comparison of methods for clustering longitudinal data with slowly changing trends," *Communications in Statistics-Simulation and Computation*, 52 (3), 621–648.

Durrett, Richard (2008), *Probability Models for DNA Sequence Evolution*, Vol. 2. Springer.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016), *Deep Learning* MIT press.

Hoffman, Matthew D., David M. Blei, Chong Wang, and John Paisley (2013), "Stochastic variational inference," *Journal of Machine Learning Research*, 14, 1303–1347.

Platzer, Michael and Thomas Reutterer (2016), "Ticking Away the Moments: Timing Regularity Helps to Better Predict Customer Activity," *Marketing Science*, 35 (5), 779–799.

Sarkar, Mainak and Arnaud De Bruyn (2021), "LSTM Response Models for Direct Marketing Analytics: Replacing Feature Engineering with Deep Learning," *Journal of Interactive Marketing*, 53 (1), 80–95.

Zhou, Mingyuan and Lawrence Carin (2013), "Negative Binomial Process Count and Mixture Modeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37 (2), 307–320.