Churn, Bayes-by, Churn

Michael Betancourt

2026-05-01

More case studies are available on my website.

A repository containing all of the files used to generate this chapter is available on GitHub.

Table of contents

1 Setup **2**

2 Data Exploration **3**

3 Model 1: Constant Stimulus Model **11**
3.1 Observational Model 11
3.2 Prior Model 13
3.3 Posterior Quantification 15
3.4 Posterior Retrodictive Checks 17

4 Model 2: Constant Stimulus/Hazard Model with Censoring **20**
4.1 Censored Observational Model 21
4.2 Posterior Quantification 21
4.3 Posterior Retrodictive Checks 23

5 Model 3: Power Stimulus/Hazard Model with Censoring **24**
5.1 Observational Model 24
5.2 Prior Model 25
5.3 Posterior Quantification 29
5.4 Posterior Retrodictive Checks 31

6 Model 4: Power Stimulus/Hazard Model with Censoring and Proportional Stimuli/Hazards **33**
6.1 Observational Model 33
6.2 Transforming the Covariates 35

6.3 Demographic Baseline 39
6.4 Prior Model 40
6.5 Posterior Quantification 44
6.6 Posterior Retrodictive Checks 45
6.7 Posterior Inferences 48

7 Customer Lifetime Value By Segment **49**
7.1 Posterior Quantification 52
7.2 Posterior Inferences 53

8 Conclusion **57**

License **58**

Original Computing Environment **58**

How do customers bring value to a company? In many cases value is driven not so much by the total number of customers at any given time, but rather for how long those customers will stay engaged. This constant ebb and flow of active customers is known as **churn**.

To properly forecast income for a company in this case, need to infer when customers will leave. In this case study we'll model the lifetime of bank customers using survival modeling techniques, and then use those inferences to predict the most profitable customer segments.

1 Setup

Let's begin by setting up our local `python` environment.

```
import matplotlib
import matplotlib.pyplot as plot
plot.rcParams['figure.figsize'] = [8, 4]
plot.rcParams['figure.dpi'] = 100
plot.rcParams['font.family'] = "Serif"

import math
import numpy
import json
```

This includes loading up `pystan`.

```
# Needed to run pystan through a jupyter kernel
import nest_asyncio
nest_asyncio.apply()

import stan
```

Next, we have a suite Markov chain Monte Carlo diagnostics and estimation tools; this code and supporting documentation are both available on GitHub.

```
import mcmc_analysis_tools_pystan3 as util
```

Finally, we have a suite of probabilistic visualization functions based on Markov chain Monte Carlo estimation. Again the code and supporting documentation are available on GitHub.

```
import mcmc_visualization_tools as putil
```

2 Data Exploration

To better understand customer churn, customer acquisition and loss were recorded across a sixty month observation period. Only those customers who joined after the observation period were recorded, but many customers were still active when the observation period ended.

```
with open("data/historical_data.json", "r") as infile:
  data = json.load(infile)
```

All times were recorded in months relative to the start of the observation window. In this units, the observation period started at time $T_0 = 0$months and ended at time $T_1 = 60$months.

```
print(f'Observation period: [{data['T0']}, {data['T1']}]')
```

```
Observation period: [0, 60]
```

In total, 2243 customers had churned by the end of the observation period, with 257 additional customers still active.

```
print(f'{data['N_churn']} churned customers')
```

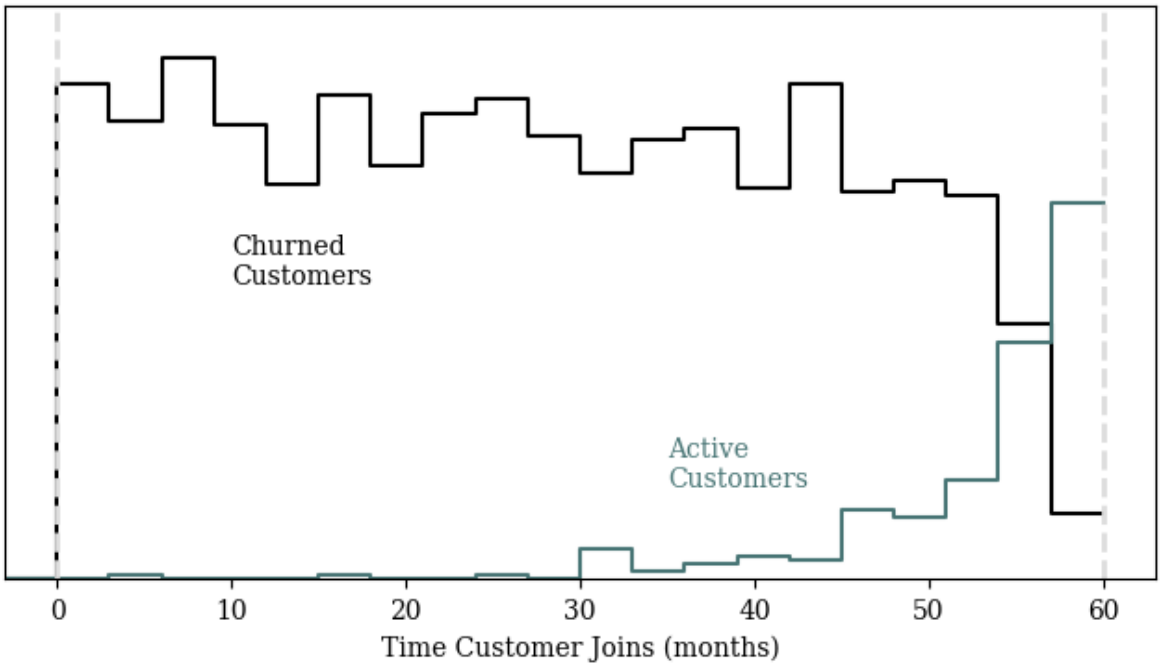
```
2243 churned customers
```

```
print(f'{data['N_active']} active customers')
```

```
257 active customers
```

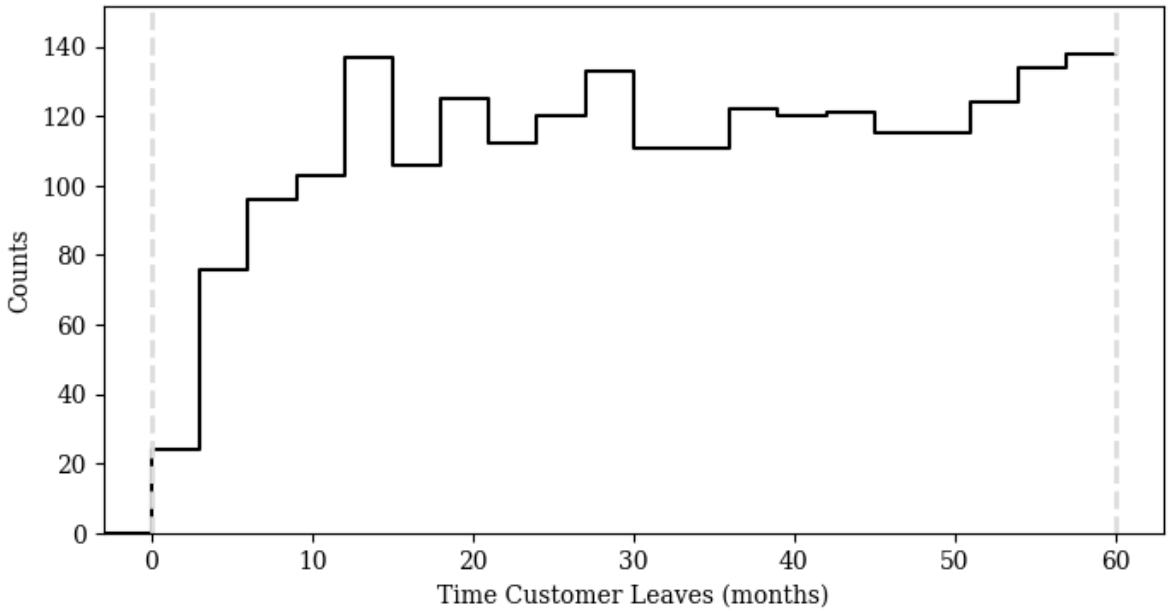
The hard cutoff of the observation period results in some interesting selection effects in the observations. As we reach the end of the observation period, churned customers are less likely to have joined because they had less time to leave before the cutoff. On the other hand, active customers were more likely to join as we approached the cutoff, where they had less time to leave.

```
putil.plot_line_hists(plot.gca(),
                      data['t_join_churn'],
                      data['t_join_active'],
                      data['T0'] - 3, data['T1'] + 3, 3,
                      xlabel="Time Customer Joins (months)")

plot.gca().axvline(x=data['T0'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")
plot.gca().axvline(x=data['T1'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")

plot.gca().text(10, 80, "Churned\nCustomers",
                color='black')

plot.gca().text(35, 25, "Active\nCustomers",
                color=putil.mid_teal)

plot.show()
```

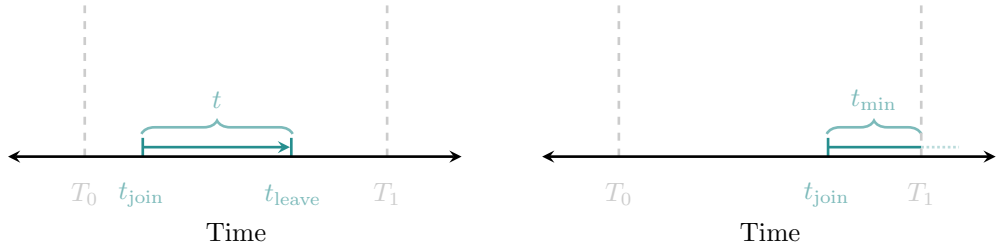


Because data collection ended at T_1, we observed leave times for only the customers who had churned before the observational period ended.

```
putil.plot_line_hist(plot.gca(), data['t_leave_churn'],
                     data['T0'] - 3, data['T1'] + 3, 3,
                     xlabel="Time Customer Leaves (months)")

plot.gca().axvline(x=data['T0'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")
plot.gca().axvline(x=data['T1'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")

plot.show()
```



The join and leave times for a particular customer are not as interesting as their difference, which defines the observed customer *lifetime* or *tenure* (Figure 1).

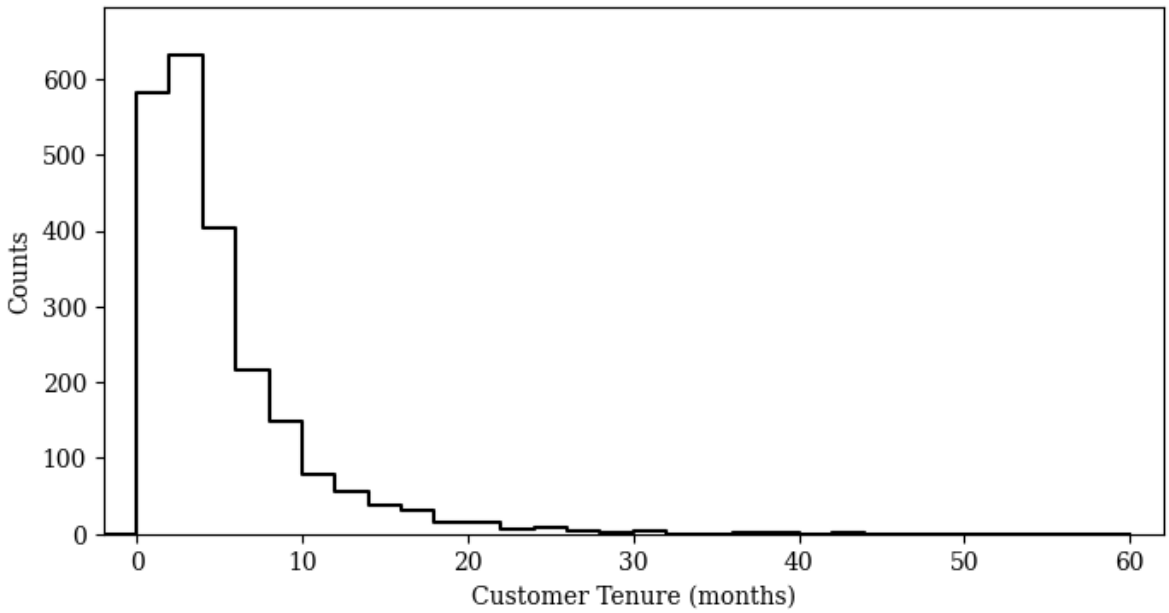


(a) (b)

Figure 1: Our data contains two types of customers (a) Churned customers joined and then left before the end of the observation period $[T_0, T_1]$. Customer lifetime is given by the time between joining and leaving. (b) Active customers joined during the observation period, but had not yet left when the observation period ended. We don't know what the overall tenure for these customers was, but we can bound it below by $t_{\text{min}} = T_1 - t_{\text{join}}$.

While most customers churned after twenty months or so, some more loyal customers persisted for quite a bit longer.

```
tenure_churn = [ tl - tj for tj, tl
                 in zip(data['t_join_churn'],
                        data['t_leave_churn']) ]

putil.plot_line_hist(plot.gca(), tenure_churn,
                     data['T0'] - 2, data['T1'] + 2, 2,
                     xlabel="Customer Tenure (months)")
plot.show()
```



The data also includes demographic information for each of the observed customers, regardless of whether or not they had left by the end of the observation period (**Table 1**).

A little over half of all churned customers participated in online banking.

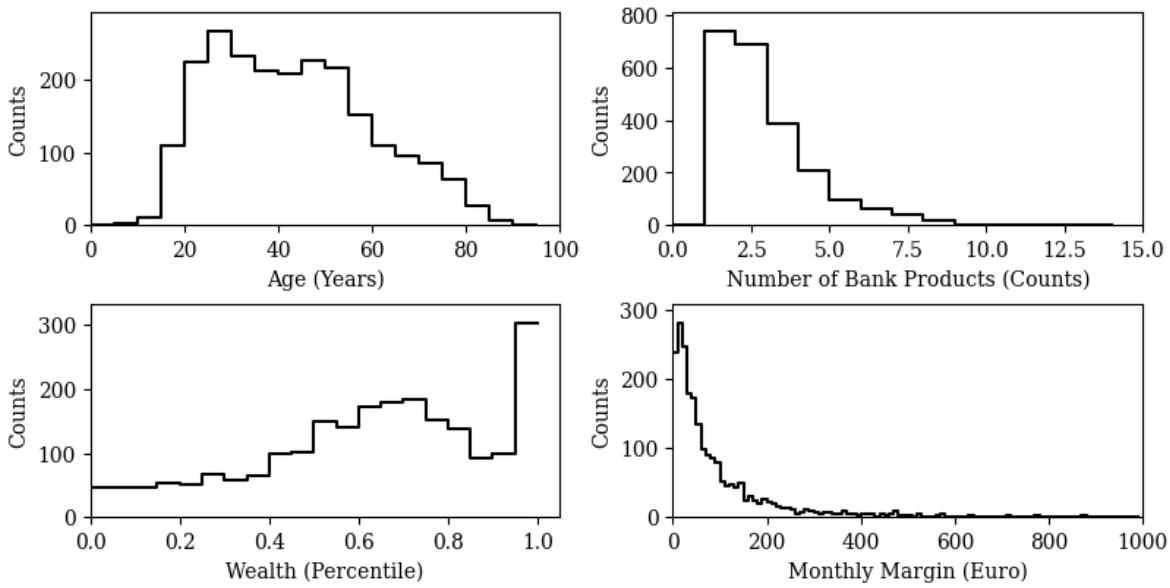
```
N_online_banking = sum([ 1 for o in
                         data['online_banking_churn']
                         if o == 1 ])

print(f'{N_online_banking} out of {data['N_churn']} '
      f'({100 * N_online_banking / data['N_churn']:.2f}%) '
       'churned customers used online banking.')
```

```
1548 out of 2243 (69.01%) churned customers used online banking.
```

The other demographic variables span a pretty wide range of values, but there is some interesting structure. For example, we can see a secondary peak of high wealth, high margin customers.

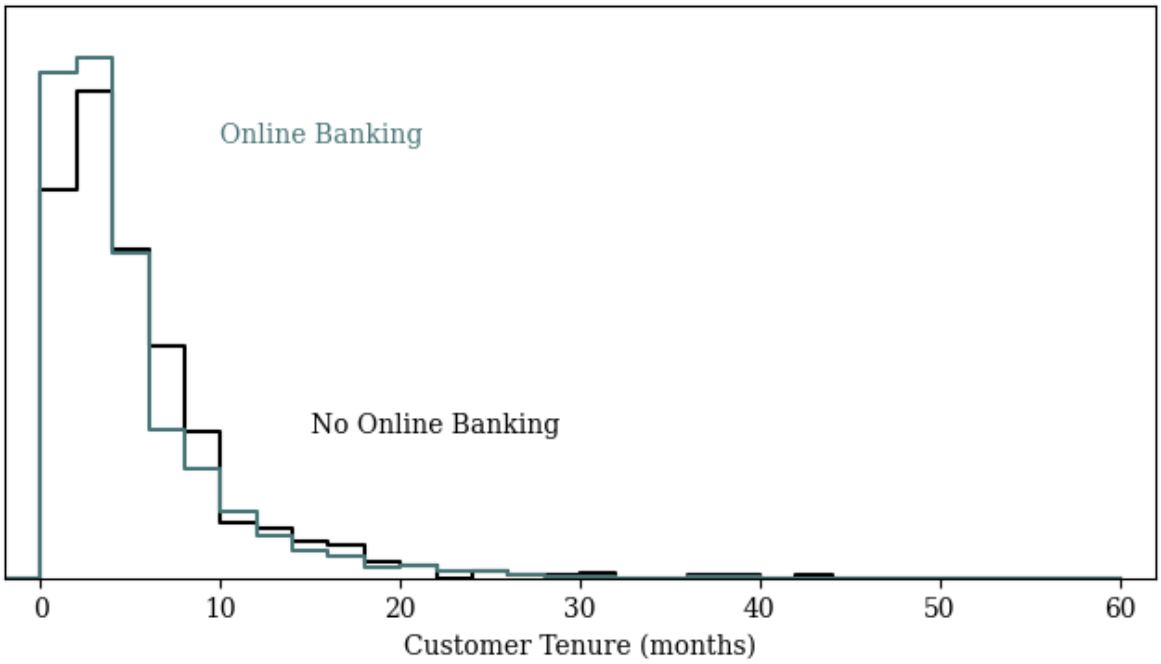
```
bin_min = [ 0, 0, 0, 0 ]
bin_max = [ 100, 15, 1.05, 1000 ]
bin_delta = [ 5, 1, 0.05, 10 ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_line_hist(axarr[i1, i2],
                       [ x[i] for x in data['X_churn'] ],
                       bin_min[i], bin_max[i], bin_delta[i],
                       xlabel=data['x_names'][i])

plot.show()
```

```
1 values (0.04%) fell below the histogram binning.
6 values (0.27%) fell above the histogram binning.
```

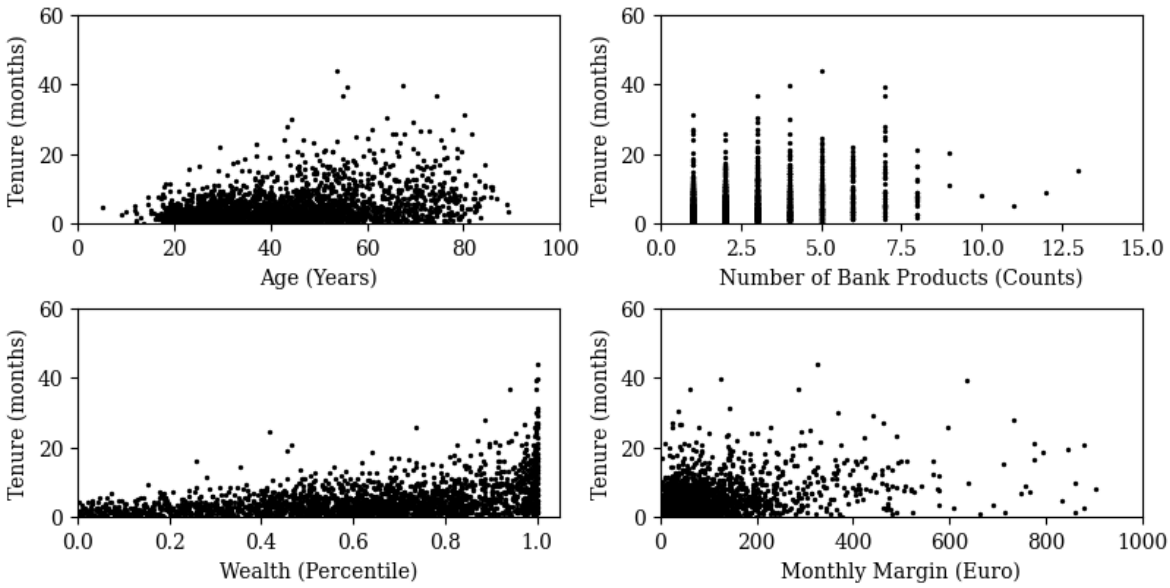


Given this demographic information, a natural question to consider is whether or not any behavior of interest, such as customer lifetimes, varies across the demographics. The empirical covariation doesn't show any obvious pattern.

```
putil.plot_line_hists(plot.gca(),
                      [ t for t, o
                        in zip(tenure_churn,
                               data['online_banking_churn'])
                        if o == 0 ],
                      [ t for t, o
                        in zip(tenure_churn,
                               data['online_banking_churn'])
                        if o == 1 ],
                      data['T0'] - 2, data['T1'] + 2, 2,
                      prob=True,
                      xlabel="Customer Tenure (months)")

plot.gca().text(15, 0.04, "No Online Banking",
                color='black')

plot.gca().text(10, 0.12, "Online Banking",
                color=putil.mid_teal)

plot.show()
```




```
f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  axarr[i1, i2].scatter([ x[i] for x in data['X_churn'] ],
                        tenure_churn,
                        color="black", s=2)
  axarr[i1, i2].set_xlim([bin_min[i], bin_max[i]])
  axarr[i1, i2].set_xlabel(data['x_names'][i])
  axarr[i1, i2].set_ylim([data['T0'], data['T1']])
  axarr[i1, i2].set_ylabel("Tenure (months)")

plot.show()
```



That said, raw scatter plots are not the most sensitive way to quantify demographic heterogeneity!

3 Model 1: Constant Stimulus Model

Having familiarized ourselves with the data, let's now attempt to infer the configuration of a customer lifetime model with those observations.

3.1 Observational Model

Churn is unavoidable in practice; it's only a matter of time before a customer eventually leaves. Modeling how long it takes for an inevitable event to occur is the purview of survival modeling.

In survival modeling, we don't directly model event times. Instead, we model the temporal **stimulus**, classically referred to as the **hazard**, that eventually triggers an event to occur. When considering customer lifetimes, the stimulus might be interpreted as modeling the overall satisfaction of a customer relative to their other banking options.

Given a stimulus function $\lambda(t)$, we can derive a cumulative stimulus function,

$$\Lambda(t) = \int_0^t \mathrm{d}t' \, \lambda(t'),$$

which quantifies how much stimulus has built-up by a certain time. The cumulative stimulus function then defines a complementary cumulative distribution function, or survival function,

$$S(t) = \exp\left(-\Lambda(t)\right).$$

If stimulus accumulates quickly, then the probability that an event has not yet occurred by a given time rapidly decays.

Finally, given a survival function we can derive a time-to-event model,

$$p(t) = \lambda(t)\, S(t).$$

Events are most likely to occur when the instantaneous stimulus is large, but the survival function has not yet decayed too much.

Let's start simple and assume that the stimulus for customer churn is fixed in time,

$$\lambda(t) = \gamma.$$

In this case, the cumulative stimulus function grows linearly,

$$\Lambda(t) = \int_0^t \mathrm{d}t'\, \lambda(t') = \gamma\, t,$$

while the survival function decays exponentially,

$$S(t) = \exp\left(-\Lambda(t)\right) = \exp(-\gamma\, t).$$

Conveniently, the resulting time-to-event model reduces to an exponential probability density function,

$$\begin{aligned} p(t) &= \lambda(t)\, S(t) \\ &= \gamma\, \exp(-\gamma\, t) \\ &= \text{exponential}(t \mid \gamma). \end{aligned}$$

This familiar form makes implementing the model in practice particularly straightforward.

Note that the time-to-event model doesn't model customer leave times but rather customer tenures,

$$t = t_{\text{leave}} - t_{\text{join}}.$$

In terms of our raw data, the time-to-event model is given by

$$p(t_{\text{leave}} - t_{\text{join}}) = \text{exponential}(t_{\text{leave}} - t_{\text{join}} \mid \gamma).$$

3.2 Prior Model

To perform a Bayesian analysis, we need to complement this observational model with a prior model. The prior model quantifies any available domain expertise that we want to incorporate into the analysis.

For this analysis, let's say that our domain expertise was already used to design the original observation period. Customer tenure longer than 60 months is not impossible, but it should be relatively rare.

There are various ways to quantify "rare", but here let's say that we want to tune our prior model so that

$$\pi(t < 60 \text{ months }) \approx 0.99,$$

or, equivalently,

$$\pi(t > 60 \text{ months }) \approx 0.01.$$

That said, our domain expertise elicitation is rarely precise enough to justify being all that precious about this particular tail probability. For example, a prior model that gives the tail probabilities

$$\pi(t > 60 \text{ months }) \approx 0.05$$

or even

$$\pi(t > 60 \text{ months }) \approx 0.10$$

will likely result in equivalent inferences in practice. Our priority here is just suppressing the most unrealistic behaviors.

Conveniently, the tail probability in a survival model is given directly by the survival function. Consequently, our domain expertise directly constrains the reasonable behaviors of the survival function,

$$\begin{aligned} \pi(t > 60 \text{ months }) &= 0.01 \\ S(60 \text{ months }) &= 0.01 \\ \exp(-\gamma\, 60 \text{ months }) &= 0.01. \end{aligned}$$

This, in, turn, constrains the reasonable values of γ,

$$\begin{aligned} \exp(-\gamma\, 60 \text{ months }) &= 0.01 \\ -\gamma\, 60 \text{ months } &= \log(0.01) \\ \gamma &= -\frac{\log(0.01)}{60 \text{ months}} \\ \gamma &\approx \frac{0.1}{\text{months}}. \end{aligned}$$

As γ increased above this value, the model predicts tenures that concentrate more and more strongly towards zero. On the other hand, when γ is below this value, the model predicts more diffuse tenures. Overall, we'll need to balance these two extremes behaviors.

Let's give a bit of wiggle room and suppress model configurations with

$$1 \lessapprox \gamma,$$

or, equivalently,

$$0 \lessapprox \eta = \frac{1}{\gamma} \lessapprox 1.$$

We can enforce this soft constraint this with a half-normal prior model that allocates 99% of its probability between 0 and 1,

$$p(\eta) = \text{half-normal}(\eta \mid 0, 1/2.57).$$

To verify that this prior model behaves as expected, we can perform a prior predictive check. Here we'll compare the prior predictions for tenure against our elicited domain expertise.

```
t_upper = 60
```

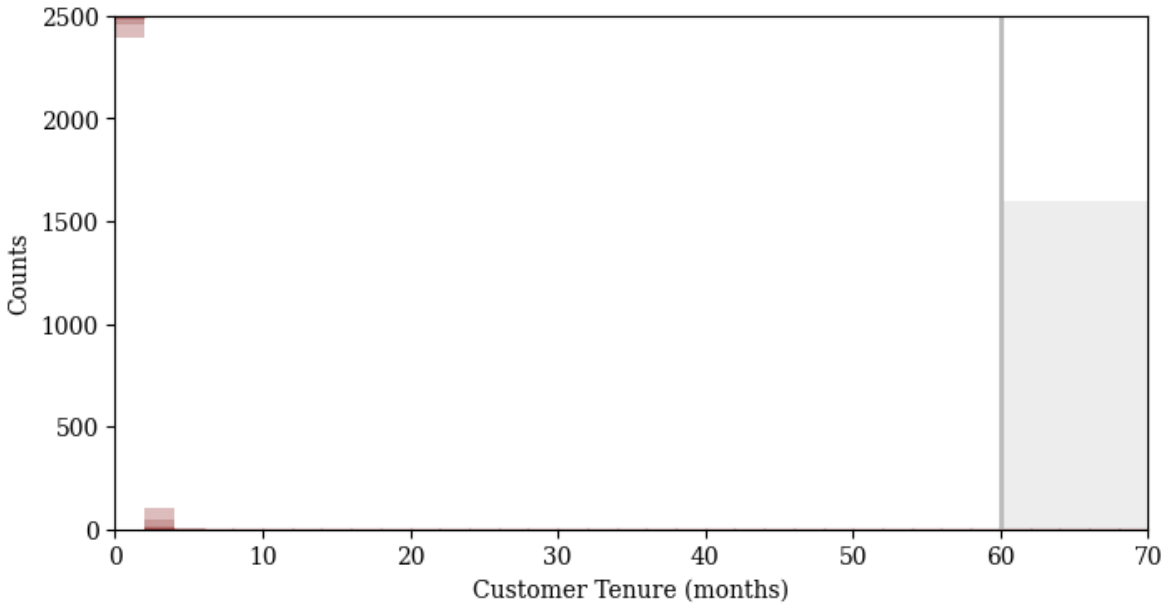
```
with open('stan_programs/model1_prior.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data={'N': 2500, 't_upper': t_upper})
fit = model.sample(num_samples=1024, refresh=0)

samples = util.extract_expectand_vals(fit)
```

```
Building...
```

Qualitatively, the prior predictions from our model look great. Shorter tenures are more likely, but very long tenures are not entirely impossible.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 'tenure_pred',
  0, t_upper + 12, 2,
  xlabel='Customer Tenure (months)'
)
plot.gca().set_xlim([0, t_upper + 10])
```

```
rect = plot.Rectangle((t_upper, 0), 10, 1600,
                      facecolor="#BBBBBB", alpha=0.25)
plot.gca().add_patch(rect)
plot.axvline(x=t_upper, linewidth=2, color="#BBBBBB")

plot.show()
```



We can also provide a more quantitative check by computing the prior predictive tail probability.

```
est = util.ensemble_mcmc_est(samples['upper_tail_p_hat'])

print(f'pi( t > 60 ) = {est[0]:.3f} +/- {2 * est[1]:.3f}')
```

```
pi( t > 60 ) = 0.000 +/- 0.000
```

This isn't too far from our initial 1% target!

3.3 Posterior Quantification

Together, the observational model and the prior define a full Bayesian model. We can summarize the structure of this model using a directed graph (Figure 2), but to work with it in practice we'll need to implement it as a probabilistic program.

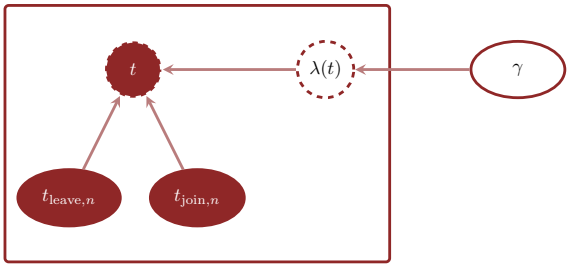


Figure 2: Our initial survival model concerns how long a customer stays after joining, $t_{\text{leave}} - t_{\text{join}}$.

Given the full Bayesian model, we can construct a posterior distribution that quantifies how consistent each model configuration is with the observed data.

```
numeric_keys = ['T0', 'T1', 'I', 'N_churn', 'X_churn',
                'online_banking_churn', 't_join_churn',
                't_leave_churn', 'N_active', 'X_active',
                'online_banking_active', 't_join_active']
stan_data = {k: data[k] for k in numeric_keys}
```

In practice, we'll use `Stan`'s Hamiltonian Monte Carlo sampler to quantify the shape of the posterior distribution.

```
with open('stan_programs/model1.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=stan_data)
fit = model.sample(num_samples=1024, refresh=0)
```

```
Building...
```

As powerful as Hamiltonian Monte Carlo is, the accuracy of this posterior quantification is not guaranteed. To ensure faithful posterior inferences, we have to first review the available computational diagnostics. In this case, there fortunately don't appear to be any signs of computational problems.

```
diagnostics = util.extract_hmc_diagnostics(fit)
util.check_all_hmc_diagnostics(diagnostics)

samples = util.extract_expectand_vals(fit)
base_samples = util.filter_expectands(samples, ['eta'])
util.check_all_expectand_diagnostics(base_samples)
```

```
All Hamiltonian Monte Carlo diagnostics are consistent with accurate
Markov chain Monte Carlo.

All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

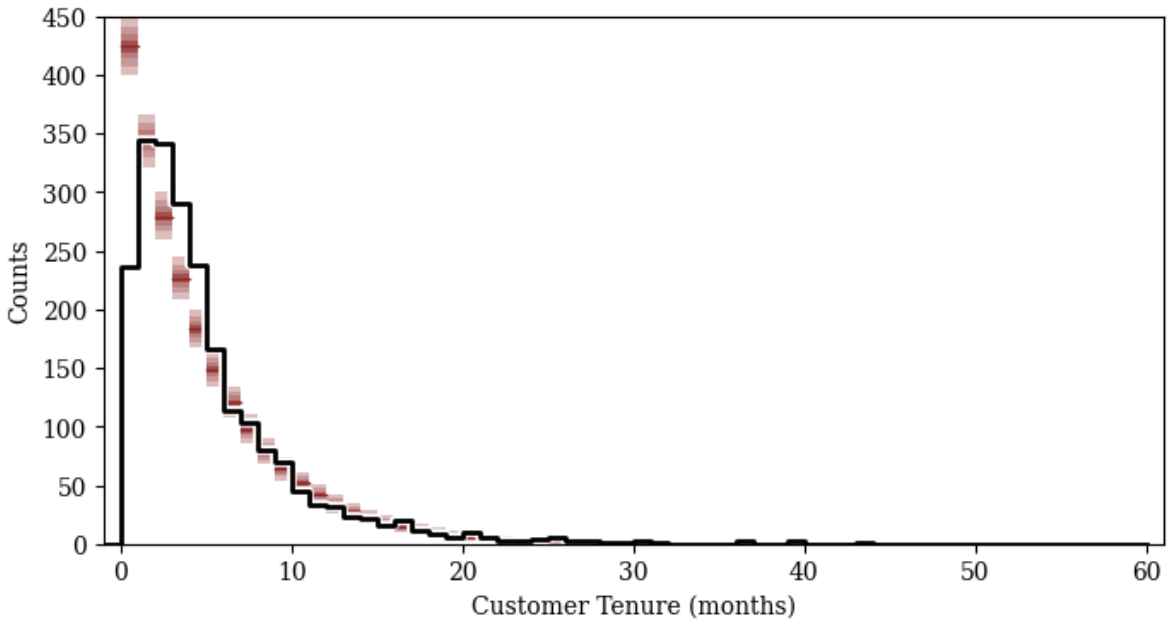
3.4 Posterior Retrodictive Checks

Before reviewing any posterior inferences, we first have to judge the posterior retrodictive performance of our model. This requires investigating how well the posterior predictive distribution recovers various features of the observed data. Inferences drawn from a model with poor retrodictive performance poorly describe the actual system being analyzed.

Here the posterior predictive distribution of tenures skews towards smaller tenures more strongly than what we see in the observed data.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 'tenure_churn_pred',
  data['T0'] - 1, data['T1'] + 1, 1,
  baseline_values=tenure_churn,
  xlabel='Customer Tenure (months)'
)
plot.show()
```

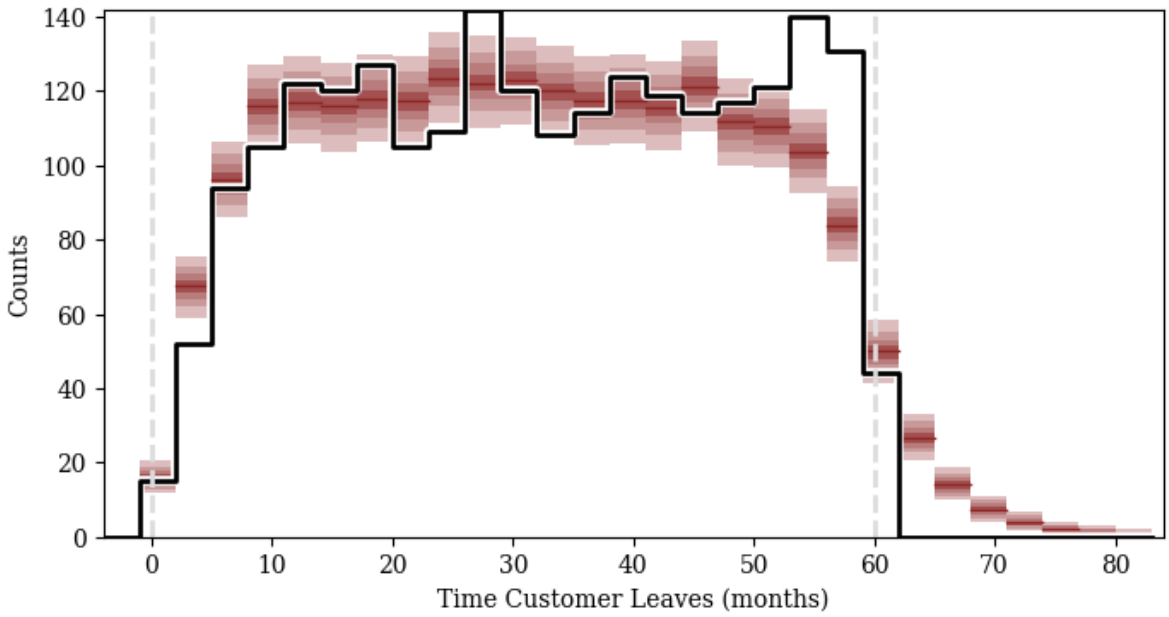
```
25 predictive values (0.00%) fell above the histogram binning.
```



On their own, the posterior predictive leave times not only decay too quickly, but also extend into impossible values! This suggests that we might not be properly accounting for the constraints of the finite observation period.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 't_leave_churn_pred',
  data['T0'] - 3, data['T1'] + 23, 3,
  baseline_values=data['t_leave_churn'],
  xlabel="Time Customer Leaves (months)"
)
plot.gca().axvline(x=data['T0'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")
plot.gca().axvline(x=data['T1'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")

plot.show()
```

```
2414 predictive values (0.03%) fell above the histogram binning.
```

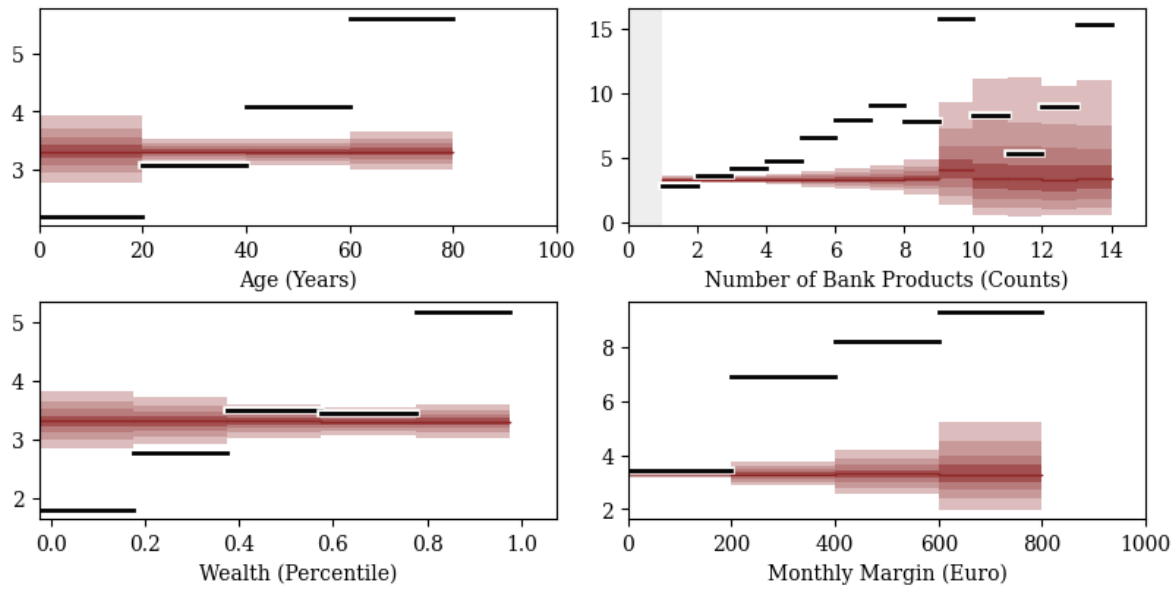


Although we haven't included demographic heterogeneity in our model, we can still see if the posterior predictions behave similarly to the observed data across the demographics. Here we'll compare median tenure in bins of each demographic variable; for a more detailed discussion of this summary see this section.

```
bin_min = [ 0, 0, 0, 0 ]
bin_max = [ 100, 15, 1.05, 1000 ]
bin_delta = [ 20, 1, 0.20, 200 ]

pred_names = [ f'tenure_churn_pred[{n + 1}]'
               for n in range(data['N_churn']) ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_conditional_median_quantiles(
    axarr[i1, i2], samples, pred_names,
    [ x[i] for x in data['X_churn'] ],
    bin_min[i], bin_max[i], bin_delta[i],
    tenure_churn, residual=False,
    xlabel=data['x_names'][i], ylabel=""
```

```
  )

plot.show()
```

```
1 conditioning values (0.04%) fell below the histogram binning.
6 conditioning values (0.27%) fell above the histogram binning.
```



There is clear retrodictive disagreement here, as well.

4 Model 2: Constant Stimulus/Hazard Model with Censoring

All of this poor retrodictive performance indicates that our initial model is nowhere near adequate enough to fit the observed data. Fortunately, the precise nature of the retrodictive tensions motivates a variety of explicit improvements to our model.

Let's start by addressing the impossible posterior predictions for the customer leave times. Predicted leave times for churned customers have to be *before* the end of the observational window, otherwise those customers would still be active.

At the same time, we are not completely ignorant of the leave times of the active customers. They must be after the close of the observational period.

Conveniently, both of these behaviors are straightforward to accommodate into our survival model.

4.1 Censored Observational Model

When making predictions for churned customers, t_{leave} has to be smaller than the right observational boundary T_1,

$$t_{\text{leave}} < T_1.$$

This then constrains the possible tenures; if the customer joined at t_{join} then

$$\begin{aligned} t_{\text{leave}} &< T_1 \\ t_{\text{leave}} - t_{\text{join}} &< T_1 - t_{\text{join}} \\ t &< T_1 - t_{\text{join}}. \end{aligned}$$

In other words, we have to *truncate* the time-to-event model when making predictions.

To accommodate active customers, we can treat their unobserved leave times as auxiliary parameters that we then marginalize. For a single active customer, the leave time model is given by

$$p(t_{\text{leave}} \mid t_{\text{join}}, \theta) = p(t_{\text{leave}} - t_{\text{join}} \mid \theta),$$

where $p(t \mid \theta)$ is the time-to-event model. To marginalize the unobserved t_{leave}, we integrate over the possible values,

$$\begin{aligned} &\int_{T_1}^{\infty} \mathrm{d}t_{\text{leave}}\, p(t_{\text{leave}} \mid t_{\text{join}}, \theta) \\ &\qquad = \int_{T_1}^{\infty} \mathrm{d}t_{\text{leave}}\, p(t_{\text{leave}} - t_{\text{join}} \mid \theta) \\ &\qquad = \int_{T_1 - t_{\text{join}}}^{\infty} \mathrm{d}t\, p(t \mid \theta) \\ &\qquad = \pi[t > T_1 - t_{\text{join}}] \\ &\qquad = S(T_1 - t_{\text{join}}). \end{aligned}$$

In other words, each active customer doesn't contribute a point-wise density $p(t)$ to the observational model, but rather an entire interval probability $\pi[t > T_1 - t_{\text{join}}]$.

4.2 Posterior Quantification

With a bit more care, our full Bayesian model now considers both the churned and active customers (Figure 3).

Conveniently, censored observations and predictions can both be implemented with the log survival function.

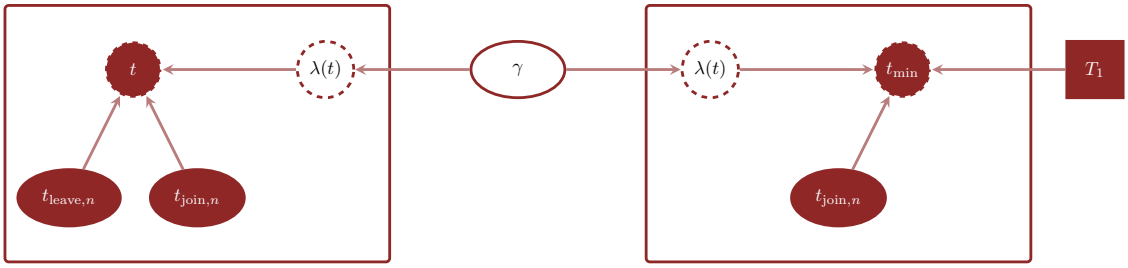


Figure 3: Although we don't observe the precise leave times of the active customers, we can lower bound them by how far the join times are from the end of the observation period. These censored observations also inform the parameters of our survival model.

```
with open('stan_programs/model2.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=stan_data)
fit = model.sample(num_samples=1024, refresh=0)
```

```
Building...
```

Despite the added model complexity, no computational problems have arisen when we quantify the resulting posterior distribution. Hamiltonian Monte Carlo is well-suited for exploring complex posterior distributions!

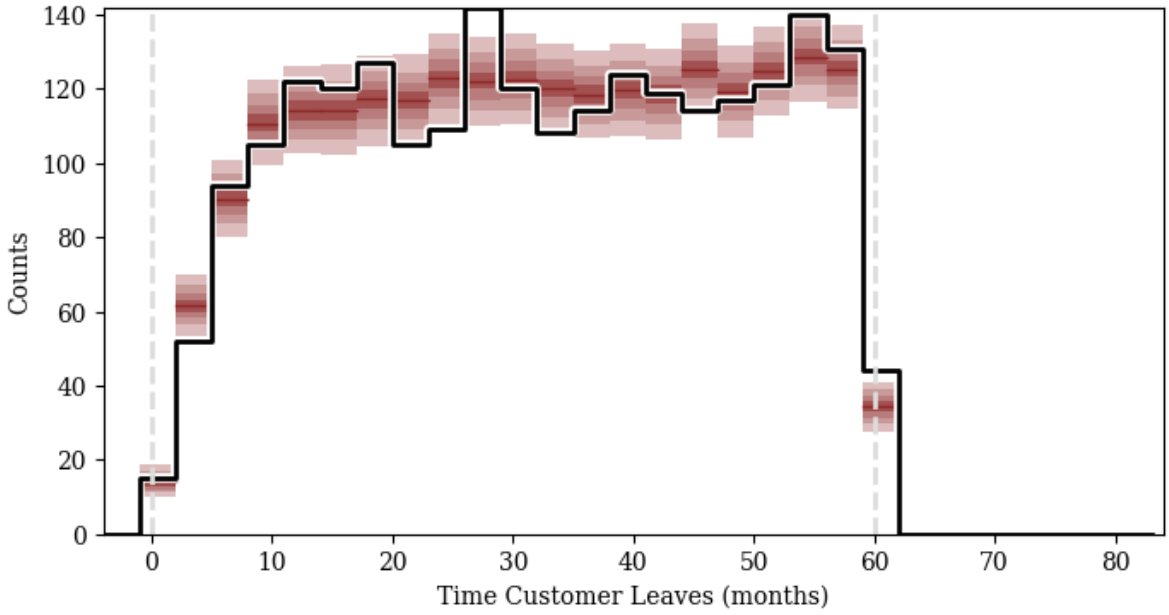
```
diagnostics = util.extract_hmc_diagnostics(fit)
util.check_all_hmc_diagnostics(diagnostics)

samples = util.extract_expectand_vals(fit)
base_samples = util.filter_expectands(samples, ['eta'])
util.check_all_expectand_diagnostics(base_samples)
```

```
All Hamiltonian Monte Carlo diagnostics are consistent with accurate
Markov chain Monte Carlo.

All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

4.3 Posterior Retrodictive Checks

Did these changes improve the adequacy of our model at all? Well, the posterior predictive leave times now agree much better with the observed behavior.

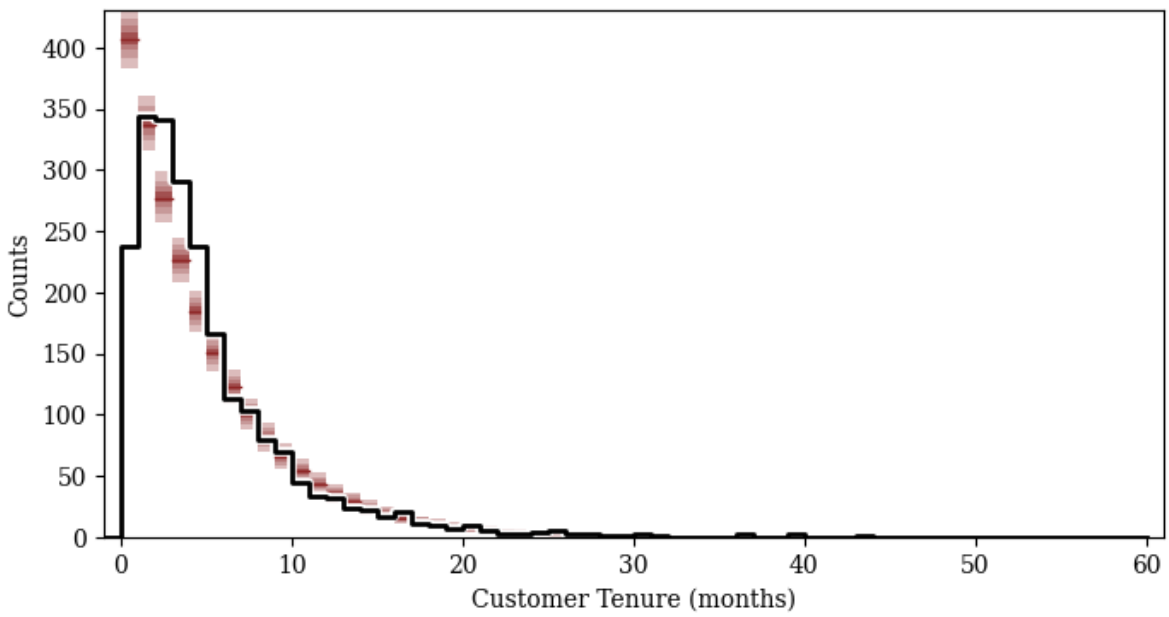
```
putil.plot_hist_quantiles(
  plot.gca(), samples, 't_leave_churn_pred',
  data['T0'] - 3, data['T1'] + 23, 3,
  baseline_values=data['t_leave_churn'],
  xlabel="Time Customer Leaves (months)"
)
plot.gca().axvline(x=data['T0'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")
plot.gca().axvline(x=data['T1'], linewidth=2,
                   linestyle='dashed', color="#DDDDDD")

plot.show()
```



Unfortunately, there is still tension in the customer tenure. Specifically, the observed behavior is peaked away from zero while the posterior predictive behavior peaks at zero.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 'tenure_churn_pred',
  data['T0'] - 1, data['T1'] + 1, 1,
```

```
  baseline_values=tenure_churn,
  xlabel='Customer Tenure (months)'
)
plot.show()
```



Reflecting on our model, the exponential time-to-event model *always* peaks at zero. In other words, there is no configuration of our current survival model that can match this observed behavior.

Let's try to address this inadequacy before we consider the retrodictive performance across the demographic variables.

5 Model 3: Power Stimulus/Hazard Model with Censoring

Fortunately, our initial survival model isn't too difficult to generalize.

5.1 Observational Model

Instead of a constant stimulus, let's allow the stimulus to potentially increase with a power of time,

$$\lambda(t) = \alpha \frac{1}{\sigma} \left(\frac{t}{\sigma} \right)^{\alpha - 1}$$

This results in the cumulative stimulus function

$$\Lambda(t) = \int_0^t \mathrm{d}t' \, \lambda(t') = \left(\frac{t}{\sigma}\right)^{\alpha},$$

and the survival function

$$S(t) = \exp\left(-\Lambda(t)\right) = \exp\left(-\left(\frac{t}{\sigma}\right)^{\alpha}\right).$$

Interestingly, the generalized time-to-event model also reduces to a familiar form, this time the Weibull family of probability density functions,

$$\begin{aligned} p(t) &= \lambda(t)\, S(t) \\ &= \alpha \frac{1}{\sigma} \left(\frac{t}{\sigma}\right)^{\alpha-1} \exp\left(-\left(\frac{t}{\sigma}\right)^{\alpha}\right) \\ &= \text{Weibull}(t \mid \alpha, \sigma). \end{aligned}$$

Note that if $\alpha = 1$ then this model reduces to our initial exponential model with $\lambda = 1/\sigma$. Because we've *expanded* our model, we're allowing our inferences to consider more complex behaviors, but not forcing it to.

5.2 Prior Model

The single threshold that we considered before was enough to constrain the reasonable configurations of γ in our initial model. It is not, however, sufficient to constrain both α and σ in our generalized model.

Here we'll elicit addition domain expertise to inform *two* thresholds: one suppressing unreasonably large tenures, and one suppressing unreasonably small tenures. Let's say that customers leaving before only a single month should be rare.

This gives us two tail probability constraints,

$$\begin{aligned} \pi(t < 1 \text{ month }) &\approx 0.01 \\ \pi(t > 60 \text{ months }) &\approx 0.01, \end{aligned}$$

which gives us two equations for the reasonable values of α and σ.

First we have

$$\begin{aligned} S(1 \text{ month }) &= 0.99 \\ \exp\left(-\left(\frac{1 \text{ month}}{\sigma}\right)^{\alpha}\right) &= 0.99 \\ \left(\frac{1 \text{ month}}{\sigma}\right)^{\alpha} &= -\log(0.99) \\ -\alpha \log\left(\frac{\sigma}{1 \text{ month}}\right) &= \log(-\log(0.99)) \\ \alpha \log\left(\frac{\sigma}{1 \text{ month}}\right) &= -\log(-\log(0.99)). \end{aligned}$$

Second we have

$$\begin{aligned} S(60 \text{ months }) &= 0.01 \\ \exp\left(-\left(\frac{60 \text{ months}}{\sigma}\right)^{\alpha}\right) &= 0.01 \\ \left(\frac{60 \text{ months}}{\sigma}\right)^{\alpha} &= -\log(0.01) \\ -\alpha \log\left(\frac{\sigma}{60 \text{ months}}\right) &= \log(-\log(0.01)) \\ \alpha \log\left(\frac{\sigma}{60 \text{ months}}\right) &= -\log(-\log(0.01)). \end{aligned}$$

After first defining

$$\kappa = \frac{\log(-\log(0.99))}{\log(-\log(0.01))},$$

we can divide the two equations to give an equation entirely in terms of σ,

$$\begin{aligned} \frac{\log\left(\frac{\sigma}{1 \text{ month}}\right)}{\log\left(\frac{\sigma}{1 \text{ month}}\right) - \log 60} &= \kappa \\ \log\left(\frac{\sigma}{1 \text{ month}}\right) &= \kappa \log\left(\frac{\sigma}{1 \text{ month}}\right) - \kappa \log 60 \\ (1-\kappa) \log\left(\frac{\sigma}{1 \text{ month}}\right) &= -\kappa \log 60 \\ \log\left(\frac{\sigma}{1 \text{ month}}\right) &= -\frac{\kappa}{1-\kappa} \log 60 \\ \sigma &= 60^{-\frac{\kappa}{1-\kappa}} \text{ month }. \end{aligned}$$

Model configurations with σ larger than this value will be too diffuse, allocating excessive probability below and above our two thresholds.

Next, we can solve the first equation for σ,

$$\log(\sigma) = -\frac{\log(-\log(0.99))}{\alpha},$$

and then substitute it into the second equation to give an equation entirely in terms of α,

$$\begin{aligned}
\alpha \log\left(\frac{\sigma}{60 \text{ months}}\right) &= -\log(-\log(0.01)) \\
\alpha \log\left(\frac{\sigma}{1 \text{ month}}\right) - \alpha \log 60 &= -\log(-\log(0.01)) \\
\alpha \left(-\frac{\log(-\log(0.99))}{\alpha}\right) - \alpha \log 60 &= -\log(-\log(0.01)) \\
-\log(-\log(0.99)) - \alpha \log 60 &= -\log(-\log(0.01)) \\
-\alpha \log 60 &= -\log(-\log(0.01)) + \log(-\log(0.99)) \\
\alpha &= \frac{\log(-\log(0.01)) - \log(-\log(0.99))}{\log 60}.
\end{aligned}$$

Model configurations with α *below* this value will concentrate at tenures that are too long.

Adding a little bit of a buffer, these calculations suggest the parameter constraints

$$0 \lessapprox \sigma \lessapprox 75,$$

and

$$1 \lessapprox \alpha.$$

The latter is equivalent to

$$0 \lessapprox \omega = \frac{1}{\alpha} \lessapprox 1.$$

Once again, half-normal prior models neatly ensure the desired regularization,

$$\begin{aligned}
p(\sigma) &= \text{half-normal}(\sigma \mid 0, 75/2.57) \\
p(\omega) &= \text{half-normal}(\omega \mid 0, 1/2.57)
\end{aligned}$$

We can verify that this prior model behaves as expected by implementing another prior predictive check.

```
t_lower = 1
t_upper = 60
```

```
with open('stan_programs/model3_prior.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data={'N': 2500,
                         't_lower': t_lower,
                         't_upper': t_upper})
```

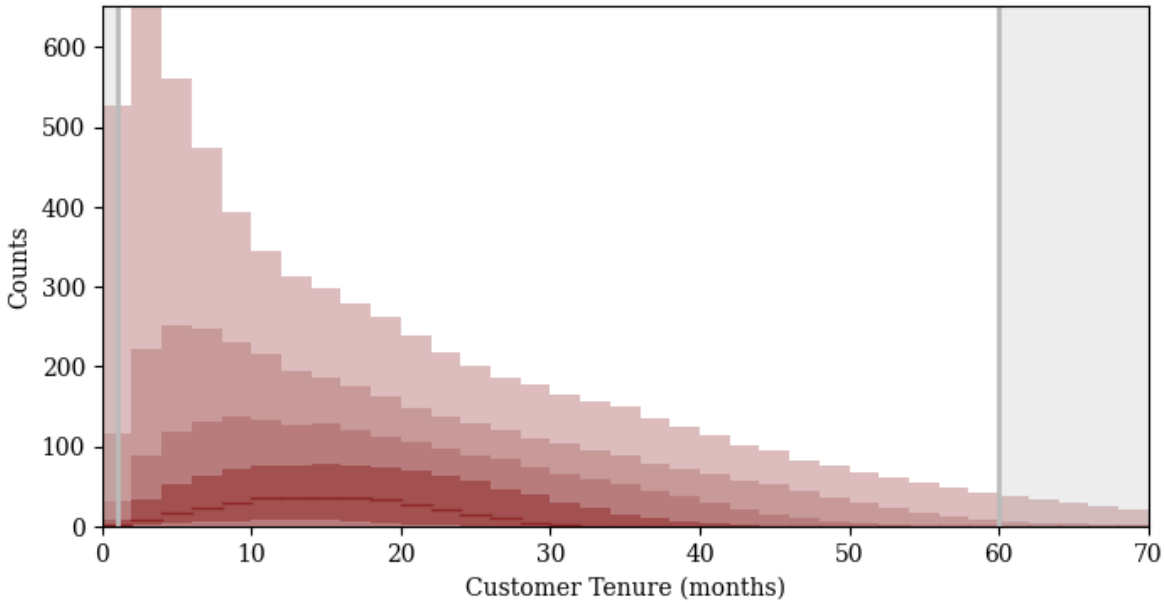
```
fit = model.sample(num_samples=1024, refresh=0)

samples = util.extract_expectand_vals(fit)
```

```
Building...
```

Qualitatively, the prior predictive behavior is similar to that in our previous model.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 'tenure_pred',
  0, t_upper + 12, 2,
  xlabel='Customer Tenure (months)'
)
plot.gca().set_xlim([0, t_upper + 10])

rect_lower = plot.Rectangle((0, 0), t_lower, 1600,
                      facecolor="#BBBBBB", alpha=0.25)
plot.gca().add_patch(rect_lower)
plot.axvline(x=t_lower, linewidth=2, color="#BBBBBB")

rect_upper = plot.Rectangle((t_upper, 0), 10, 1600,
                      facecolor="#BBBBBB", alpha=0.25)
plot.gca().add_patch(rect_upper)
plot.axvline(x=t_upper, linewidth=2, color="#BBBBBB")

plot.show()
```

```
226894 predictive values (2.22%) fell above the histogram binning.
```



Quantitatively, both prior predictive tail probabilities are reasonable. Again, our domain expertise elicitation is rarely precise enough to justify being too precious about the tail probabilities.

```
est = util.ensemble_mcmc_est(samples['lower_tail_p_hat'])

print(f'pi( t < 1 ) = {est[0]:.3f} +/- {2 * est[1]:.3f}')
```

```
pi( t < 1 ) = 0.040 +/- 0.009
```

```
est = util.ensemble_mcmc_est(samples['upper_tail_p_hat'])

print(f'pi( t > 60 ) = {est[0]:.3f} +/- {2 * est[1]:.3f}')
```

```
pi( t > 60 ) = 0.043 +/- 0.005
```

5.3 Posterior Quantification

The graphical structure of this new model is the same as before, we just replace the initial stimulus function with a more general one (Figure 4).

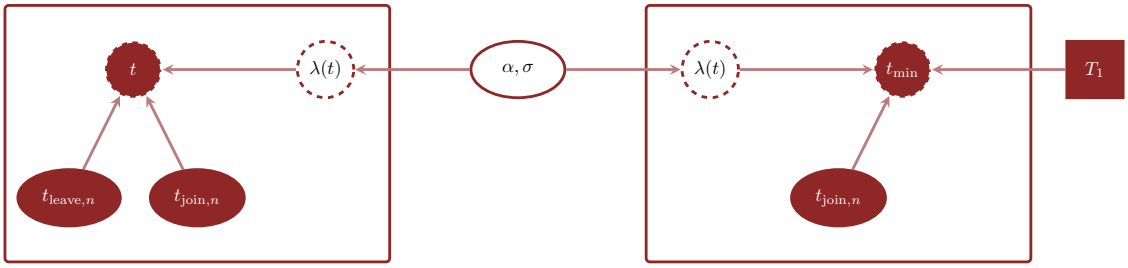


Figure 4: Moving from a constant to a power stimulus model changes the derived time-to-event model, but not the overall structure of the data generating process.

```
with open('stan_programs/model3.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=stan_data)
fit = model.sample(num_samples=1024, refresh=0)
```

```
Building...
```

Conveniently, the posterior computation remains well-behaved.

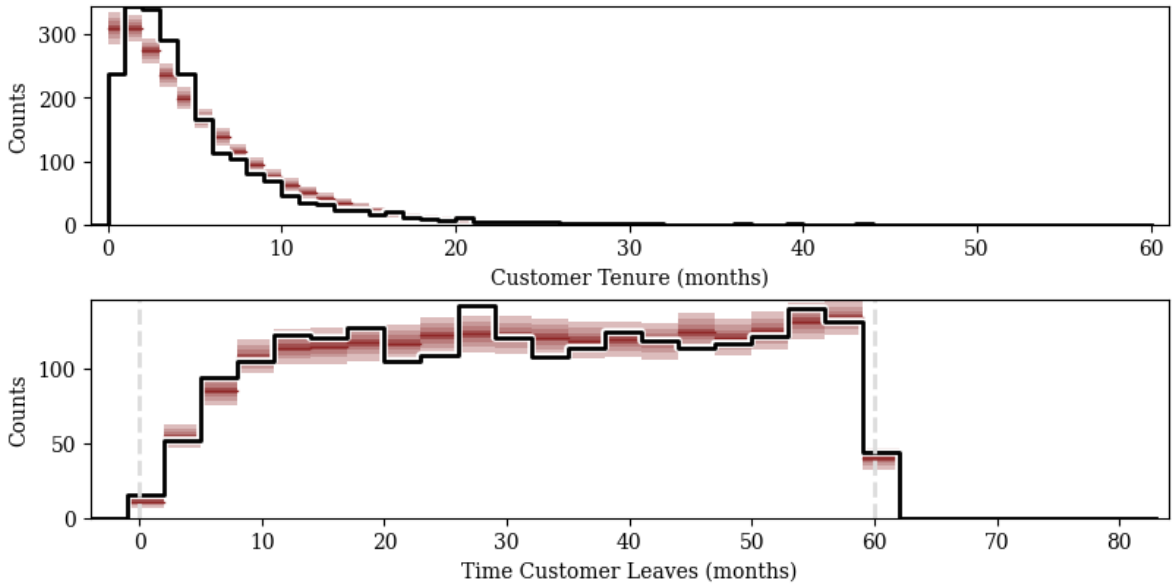
```
diagnostics = util.extract_hmc_diagnostics(fit)
util.check_all_hmc_diagnostics(diagnostics)

samples = util.extract_expectand_vals(fit)
base_samples = util.filter_expectands(samples,
                                      ['omega', 'sigma'])
util.check_all_expectand_diagnostics(base_samples)
```

```
All Hamiltonian Monte Carlo diagnostics are consistent with accurate
Markov chain Monte Carlo.

All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

5.4 Posterior Retrodictive Checks

Overall, the retrodictive agreement for both the tenure and leave times has improved quite a bit. That said, the agreement still leaves something to be desired.

```
f, axarr = plot.subplots(2, 1, layout="constrained")

putil.plot_hist_quantiles(
  axarr[0], samples, 'tenure_churn_pred',
  data['T0'] - 1, data['T1'] + 1, 1,
  baseline_values=tenure_churn,
  xlabel='Customer Tenure (months)'
)


putil.plot_hist_quantiles(
  axarr[1], samples, 't_leave_churn_pred',
  data['T0'] - 3, data['T1'] + 23, 3,
  baseline_values=data['t_leave_churn'],
  xlabel="Time Customer Leaves (months)"
)
axarr[1].axvline(x=data['T0'], linewidth=2,
                 linestyle='dashed', color="#DDDDDD")
axarr[1].axvline(x=data['T1'], linewidth=2,
                 linestyle='dashed', color="#DDDDDD")

plot.show()
```

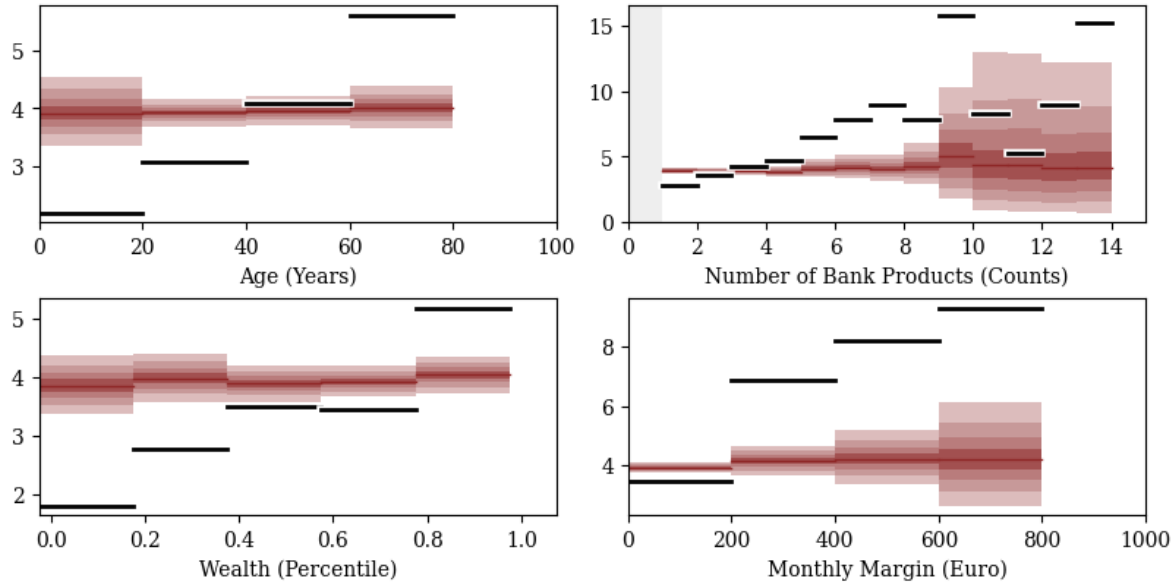


This residual tension isn't necessarily surprising. Remember that we have yet to consider demographic heterogeneity! Indeed, when we examine the binned medians, we see strong disagreement between the posterior predictions and the observed data.

```
bin_min = [ 0, 0, 0, 0 ]
bin_max = [ 100, 15, 1.05, 1000 ]
bin_delta = [ 20, 1, 0.20, 200 ]

pred_names = [ f'tenure_churn_pred[{n + 1}]'
               for n in range(data['N_churn']) ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_conditional_median_quantiles(
    axarr[i1, i2], samples, pred_names,
    [ x[i] for x in data['X_churn'] ],
    bin_min[i], bin_max[i], bin_delta[i],
    tenure_churn, residual=False,
    xlabel=data['x_names'][i], ylabel=""
  )
```

```
plot.show()
```

```
1 conditioning values (0.04%) fell below the histogram binning.
6 conditioning values (0.27%) fell above the histogram binning.
```



6 Model 4: Power Stimulus/Hazard Model with Censoring and Proportional Stimuli/Hazards

In order to improve our model, we have to address a question that is easy to take for granted but is often diabolically subtle. Exactly what behaviors could vary with the available demographic variables?

6.1 Observational Model

Instead of considering which features of the time-to-event model can vary with external variables, the *proportional stimulus model* allows the external variables to scale the underlying stimulus function. This dependence then propagates to the derived time-to-event model.

The precise coupling between the stimulus function behavior and the demographic variables is not always static; it can generally vary with the behavior of the demographic variables. In the language of regression modeling, the conditional relationship between event time and

the demographic variables can be confounded with the marginal behavior of the external variables.

Here we will assume that any confounding is negligible, so that we can safely ignore the latter. That said, this is a strong assumption that has to be justified from domain expertise in a real analysis.

If we denote the continuous demographic variables for the nth customer as

$$(x_{n1}, \ldots, x_{nI})$$

and the categorical demographic variables as

$$(l_n^1, \ldots, l_n^K),$$

then the proportional stimulus model can be written as

$$\lambda(t) = \lambda_0(t) \, \exp(f(x_{n1}, \ldots, x_{nI}, \nu_{l_n^1}^1, \ldots, \nu_{l_n^K}^K))$$

To ensure that the survival model is well-behaved, the outputs of the function f must be positive.

The baseline stimulus function $\lambda_0(t)$ models the behavior for the implicit baseline configuration satisfying

$$f(x_{n1}, \ldots, x_{nI}, \nu_{l_n^1}^1, \ldots, \nu_{l_n^K}^K,) = 1.$$

If the output of f is smaller than one, then it suppresses the driving stimulation, leading to later event times. When the output is larger than one, the stimulus is amplified and event times accelerate. The precise form of f determines exactly how customer tenure varies as the demographic variables deviate from the implicit baseline.

There is no end of possible function models that we might consider. Proportional stimulus models, however, almost always assume an exponential-linear functional form,

$$\lambda(t) = \lambda_0(t) \, \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i} \right) + \sum_{k=1}^{K} \delta_{l_n^k}^k \right),$$

where

$$\delta_{l_n^k}^k = \nu_{l_n^k}^k - \nu_{l_0^k}^k.$$

We can always interpret this exponential-linear model as a local approximation to a nonlinear functional model around a baseline configuration of the demographic variables. Conveniently, the configuration for this functional model is explicitly given by $x_{ni} = x_{0i}$ and $l_n^k = l_0^k$.

Given this functional model, the complete proportional power stimulus function is given by

$$\lambda(t) = \alpha \, \frac{1}{\sigma} \left(\frac{t}{\sigma} \right)^{\alpha - 1} \, \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i} \right) + \sum_{k=1}^{K} \delta_{l_n^k}^k \right)$$

with the resulting cumulative stimulus function

$$\Lambda(t) = \left(\frac{t}{\sigma}\right)^{\alpha} \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i}\right) + \sum_{k=1}^{K} \delta_{l_n^k}^k\right),$$

and survival function

$$S(t) = \exp\left(-\left(\frac{t}{\sigma}\right)^{\alpha} \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i}\right) + \sum_{k=1}^{K} \delta_{l_n^k}^k\right)\right).$$

The time-to-event model then becomes

$$\begin{aligned} p(t) = \; & \alpha \frac{1}{\sigma} \left(\frac{t}{\sigma}\right)^{\alpha-1} \\ & \cdot \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i}\right) + \sum_{k=1}^{K} \delta_{l_n^k}^k\right) \\ & \cdot \exp\left(-\left(\frac{t}{\sigma}\right)^{\alpha} \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i}\right) + \sum_{k=1}^{K} \delta_{l_n^k}^k\right)\right) \\ = \; & \alpha \frac{1}{\tau} \left(\frac{t}{\tau}\right)^{\alpha-1} \exp\left(-\left(\frac{t}{\tau}\right)^{\alpha}\right) \\ = \; & \text{weibull}(t \mid \alpha, \tau) \end{aligned}$$

where

$$\tau = \sigma \exp\left(\sum_{i=1}^{I} \beta_i \left(x_{ni} - x_{0i}\right) + \sum_{k=1}^{K} \delta_{l_n^k}^k\right)$$

Coincidentally, only the second parameter of the time-to-event model varies with the demographic variables in this case. More generally, multiple if not all of the time-to-event model parameters can vary with the demographic variables.

6.2 Transforming the Covariates

One weakness of the exponential-linear model is that is can poorly approximate functional behavior when the input variables concentrate near a boundary. Unfortunately, this is exactly the case here for the observed wealth percentiles and monthly margins. That said, we can often improve the approximation power of the exponential-linear by first unconstraining these variables.

Allowing each continuous demographic variable to be transformed is equivalent to the slightly more sophisticated functional model

$$\begin{aligned} & f(x_{n1}, \ldots, x_{nI}, \nu^1_{l^1_n}, \ldots, \nu^K_{l^K_n}) \\ & \quad = \exp\left(\sum_{i=1}^{I} \beta_i \left(g_i(x_{ni}) - g_i(x_{0i}) \right) + \sum_{k=1}^{K} \delta^k_{l^k_n} \right) \end{aligned}$$

for appropriate transformation functions g_i. This, in turn, is equivalent to using modified demographic variables,

$$\begin{aligned} & f(x_{n1}, \ldots, x_{nI}, \nu^1_{l^1_n}, \ldots, \nu^K_{l^K_n}) \\ & \quad = \exp\left(\sum_{i=1}^{I} \beta_i \left(g_i(x_{ni}) - g_i(x_{0i}) \right) + \sum_{k=1}^{K} \delta^k_{l^k_n} \right) \\ & \quad = \exp\left(\sum_{i=1}^{I} \beta_i \left(z_{ni} - z0i \right) + \sum_{k=1}^{K} \delta^k_{l^k_n} \right) \end{aligned}$$

The interval constraints of wealth percentiles can be transformed away with the logit function, while the positivity constraint of the monthly margins can be transformed away with the natural logarithm. We'll leave the observed age and product counts unmodified, as they concentrate relatively far away from their lower zero boundaries.

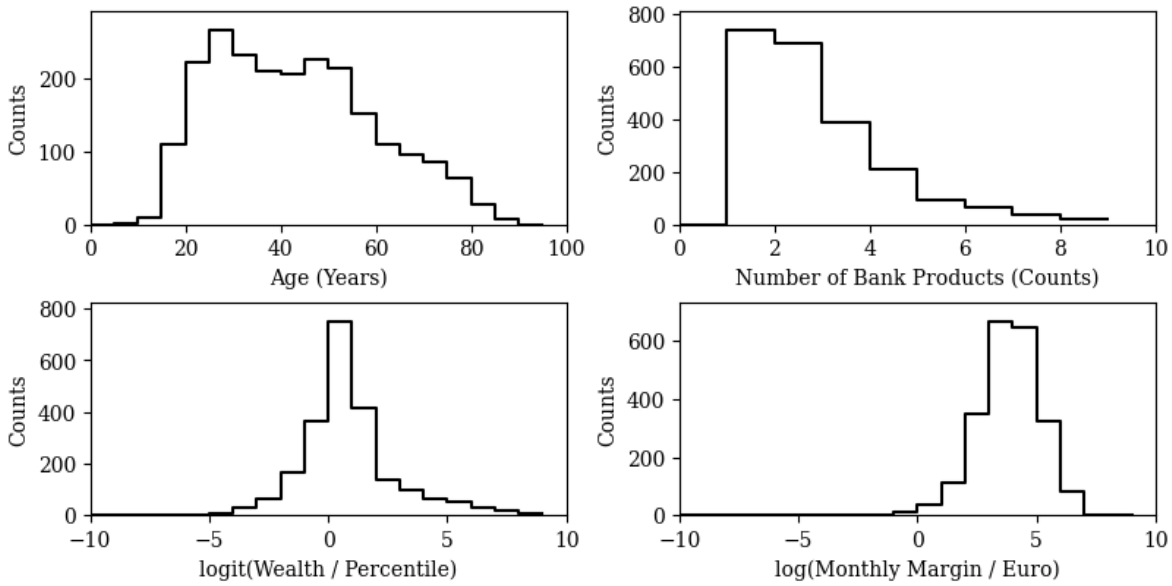
```
def logit(x):
  return math.log(x / (1 - x))

def transform_covariates(x):
  return [ x[0], x[1], logit(x[2]), math.log(x[3]) ]

z_names = [ data['x_names'][0],
            data['x_names'][1],
            f'logit(Wealth / Percentile)',
            f'log(Monthly Margin / Euro)'    ]
```

```
Z_churn = [ transform_covariates(x)
            for x in data['X_churn'] ]

Z_active = [ transform_covariates(x)
             for x in data['X_active'] ]
```

As expected, the transformed demographic variables no longer bunch up at their lower boundary.

```
bin_min = [ 0, 0, -10, -10 ]
bin_max = [ 100, 10, 10, 10 ]
bin_delta = [ 5, 1, 1, 1 ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_line_hist(axarr[i1, i2],
                       [ z[i] for z in Z_churn ],
                       bin_min[i], bin_max[i], bin_delta[i],
                       xlabel=z_names[i])
plot.show()
```

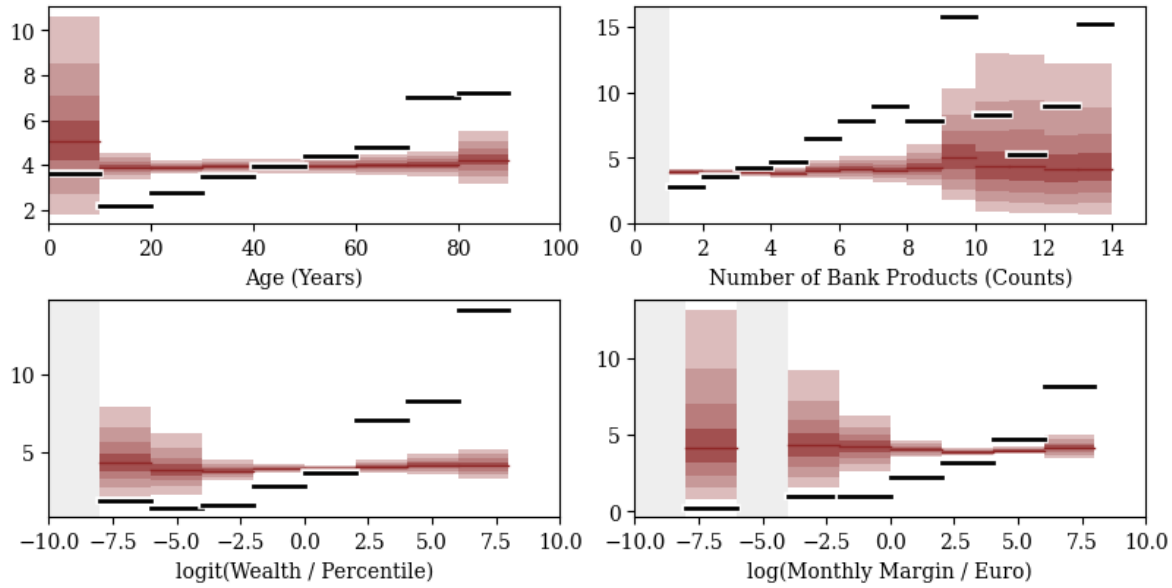
```
1 values (0.04%) fell below the histogram binning.
3 values (0.13%) fell above the histogram binning.
18 values (0.80%) fell above the histogram binning.
```



Our model recovers customer tenure binned in these transformed demographic variables just as poorly as it did in bins of the unprocessed demographic variables.

```
bin_min = [ 0, 0, -10, -10 ]
bin_max = [ 100, 15, 10, 10 ]
bin_delta = [ 10, 1, 2, 2 ]

pred_names = [ f'tenure_churn_pred[{n + 1}]'
               for n in range(data['N_churn']) ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_conditional_median_quantiles(
    axarr[i1, i2], samples, pred_names,
    [ z[i] for z in Z_churn ],
    bin_min[i], bin_max[i], bin_delta[i],
    tenure_churn, residual=False,
    xlabel=z_names[i], ylabel=""
  )

plot.show()
```

```
1 conditioning values (0.04%) fell below the histogram binning.
18 conditioning values (0.80%) fell above the histogram binning.
```



6.3 Demographic Baseline

All that's left is to specify a baseline configuration of the demographic variables. This baseline effectively defines a default customer, relative to whom we can interpret the demographic heterogeneity. Note that this default customer can be completely hypothetical, and does not have to correspond to any particular observed customer.

We'll take the baseline online banking configuration to be false. Because online banking features only two categories, true and false, we can implement this contribution by adding a single parameter δ_{online} for when a customer participates in online banking.

This leaves the continuous demographic variables.

```
x0 = [
 40,  # Age (years)
 2,   # Number of bank products (counts)
 0.5, # Wealth (Percentile)
 50   # Monthly Margin (euros)
]
```

To use this baseline configuration in our model, however, we'll need to apply the appropriate transformations.

```
z0 = transform_covariates(x0)

stan_data['z0'] = z0
stan_data['Z_churn'] = Z_churn
stan_data['Z_active'] = Z_active
```

6.4 Prior Model

One benefit of the proportional stimulus model is that we can motivate an informative prior model for the new parameters by reasoning about thresholds in proportional changes.

Consider, for example, a categorical demographic variable. If the value of *only* this variable changes from its baseline to δ_l^k, then the stimulus function is proportionally scaled by $\exp(\delta_l^k)$. Consequently, constraining the reasonable proportional variations constrains reasonable values for δ_l^k,

$$\begin{aligned} \frac{1}{\delta} &\lessapprox \exp(\delta_l^k) \lessapprox \delta \\ -\log \delta &\lessapprox \delta_l^k \lessapprox +\log \delta \end{aligned}$$

For online banking participation, we'll take $\delta = 5$ so that

$$|\delta_{\text{online}}| < \log 5.$$

Note that 500% is an absolutely huge proportional change; this prior model is *extremely* weakly informative relative to what domain expertise would be available in most applications!

To motivate prior models for coefficients of the remaining demographic variables, we can consider constraining the proportional variation when each component individually deviates from its baseline by a certain amount,

$$\begin{aligned} \frac{1}{\delta} &\lessapprox \exp(\beta_i \, \Delta x_i) \lessapprox \delta \\ -\log \delta &\lessapprox \beta_i \, \Delta x_i \lessapprox +\log \delta \\ -\frac{\log \delta}{\Delta x_i} &\lessapprox \beta_i \lessapprox +\frac{\log \delta}{\Delta x_i} \end{aligned}$$

For age we'll take $\delta = 5$ and $\Delta x_i = 10$ so that

$$|\beta_{\text{age}}| < \frac{\log 5}{10},$$

while for the number of banking products we'll take $\delta = 5$ and $\Delta x_i = 1$ so that

$$|\beta_{\text{products}}| < \frac{\log 5}{1}.$$

When working with the transformed demographic variables, however, we'll want to take those transformations into account. For example, log transforming a demographic variable is equivalent to the proportional variation

$$\begin{aligned}
\exp\left(\beta_i \left(\log x_i - \log x_{0i}\right)\right) &= \exp\left(\beta_i \, \log \frac{x_i}{x_{0i}}\right) \\
&= \exp\left(\log \left(\frac{x_i}{x_{0i}}\right)^{\beta_i}\right) \\
&= \left(\frac{x_i}{x_{0i}}\right)^{\beta_i}
\end{aligned}$$

Consequently, the corresponding prior model is informed by reasoning about proportional output variation given proportional input variation. If

$$x_i = \zeta \, x_{0i}$$

then

$$\begin{aligned}
\frac{1}{\delta} \lessapprox \left(\frac{x_i}{x_{0i}}\right)^{\beta_i} &\lessapprox \delta \\
\frac{1}{\delta} \lessapprox \left(\zeta\right)^{\beta_i} &\lessapprox \delta \\
-\log \delta \lessapprox \beta_i \, \log \zeta &\lessapprox +\log \delta \\
-\frac{\log \delta}{\log \zeta} \lessapprox \beta_i &\lessapprox +\frac{\log \delta}{\log \zeta}.
\end{aligned}$$

For monthly margin we'll take $\delta = 5$ and $\zeta = 2$ so that

$$|\beta_i| < \frac{\log 5}{\log 2}.$$

Accounting for logit transformed demographic variables is a bit more subtle. First we have

$$\begin{aligned}
\exp\left(\beta_i \left(\text{logit} x_i - \text{logit} x_{0i}\right)\right) &= \exp\left(\beta_i \, \left(\log \frac{x_i}{1 - x_i} - \log \frac{x_{0i}}{1 - x_{0i}}\right)\right) \\
&= \exp\left(\beta_i \, \log \left(\frac{x_i}{1 - x_i} \frac{1 - x_{0i}}{x_{0i}}\right)\right) \\
&= \exp\left(\log \left(\frac{x_i}{1 - x_i} \frac{1 - x_{0i}}{x_{0i}}\right)^{\beta_i}\right) \\
&= \left(\frac{x_i}{1 - x_i} \frac{1 - x_{0i}}{x_{0i}}\right)^{\beta_i}
\end{aligned}$$

In this case, we need to think about how the proportional output changes given proportional changes in the input *odds*,

$$\zeta = \frac{x_i}{1 - x_i} \frac{1 - x_{0i}}{x_{0i}}.$$

This gives

$$-\frac{\log \delta}{\log \zeta} \lessapprox \beta_i \lessapprox +\frac{\log \delta}{\log \zeta}.$$

For wealth percentile we'll take $\delta = 5$ and $\zeta = 2$ so that

$$|\beta_i| < \frac{\log 5}{\log 2}.$$

We don't want any unexpected interaction between the demographic variables to spoil our careful prior model development. Specifically, we should be diligent and perform a prior check.

```
simu_data={'t_lower': t_lower, 't_upper': t_upper,
           'I': data['I'], 'z0': z0,
           'N_churn': data['N_churn'], 'Z_churn': Z_churn,
           'online_banking_churn': data['online_banking_churn']}

with open('stan_programs/model4_prior.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=simu_data)
fit = model.sample(num_samples=1024, refresh=0)

samples = util.extract_expectand_vals(fit)
```

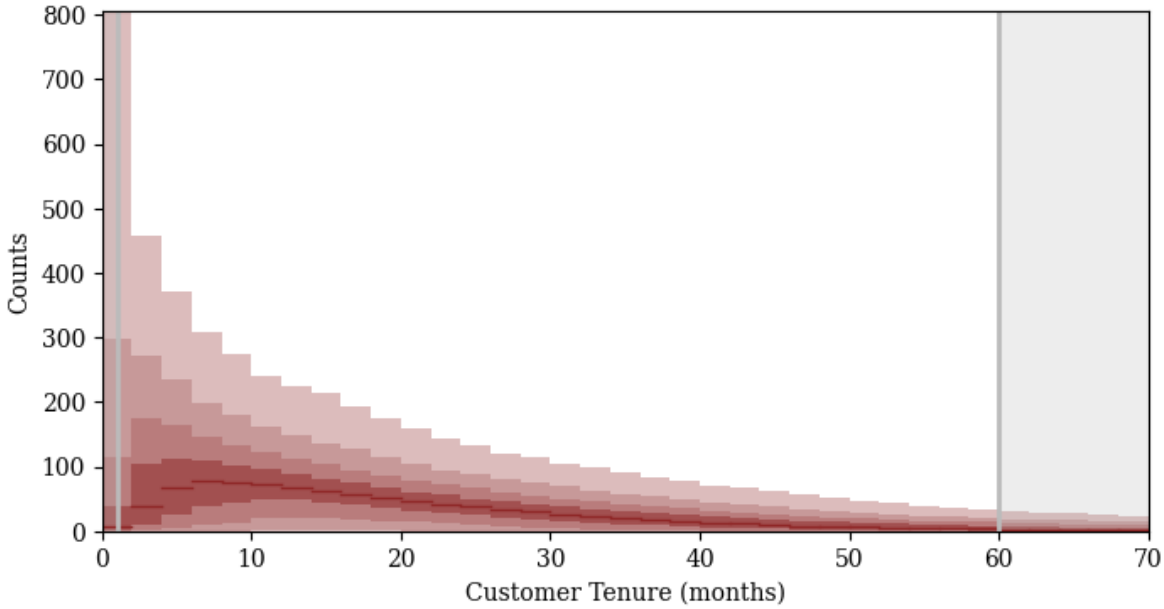
```
Building...
```

The shape of the prior predictive distribution in this new model is similar to that from the previous models, just a bit more diffuse.

```
putil.plot_hist_quantiles(
  plot.gca(), samples, 'tenure_pred',
  0, t_upper + 12, 2,
  xlabel='Customer Tenure (months)'
)
plot.gca().set_xlim([0, t_upper + 10])
```

```
rect_lower = plot.Rectangle((0, 0), t_lower, 1600,
                        facecolor="#BBBBBB", alpha=0.25)
plot.gca().add_patch(rect_lower)
plot.axvline(x=t_lower, linewidth=2, color="#BBBBBB")

rect_upper = plot.Rectangle((t_upper, 0), 10, 1600,
                        facecolor="#BBBBBB", alpha=0.25)
plot.gca().add_patch(rect_upper)
plot.axvline(x=t_upper, linewidth=2, color="#BBBBBB")

plot.show()
```

```
706640 predictive values (7.69%) fell above the histogram binning.
```



This excessive diffusion also manifests is higher tail probabilities.

```
est = util.ensemble_mcmc_est(samples['lower_tail_p_hat'])

print(f'pi( t < 1 ) = {est[0]:.3f} +/- {2 * est[1]:.3f}')
```

```
pi( t < 1 ) = 0.057 +/- 0.009
```

```
est = util.ensemble_mcmc_est(samples['upper_tail_p_hat'])

print(f'pi( t > 60 ) = {est[0]:.3f} +/- {2 * est[1]:.3f}')
```

```
pi( t > 60 ) = 0.104 +/- 0.005
```

Heavier tails, however, are no unexpected. Recall that we're using the same prior model for α and σ, and then incorporating *additional* variation in the tenure behavior with the proportional scaling.

That said, the prior predictive tail probabilities are still small enough that our prior model is in reasonable agreement with our roughly elicited tenure thresholds. Besides, we can always introduce a more informative prior model, for example narrowing the component prior models for α and σ, if needed.

6.5 Posterior Quantification

Incorporating a proportional stimulus into our previous model requires demographic variables for both the churned and active customers (Figure 5). To avoid overcrowding the graphical model, I'm showing only the demographic variables that are treated as continuous inputs here.

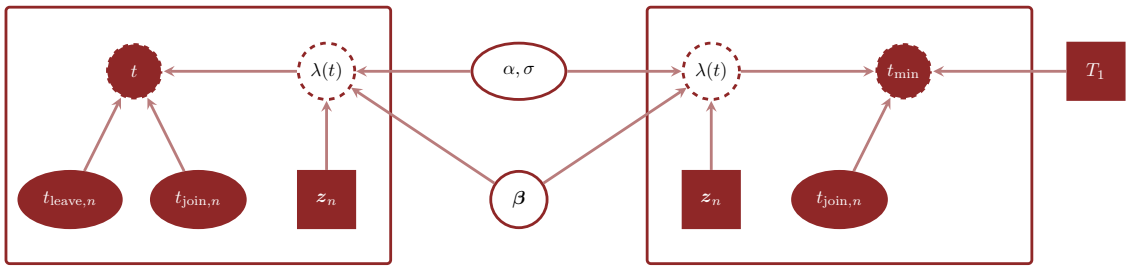


Figure 5: A proportional stimulus survival model allows the demographic variables to suppress or amplify the stimulus function, which then modifies the behavior of the time-to-event model.

Because the proportional scaling effectively modifies only the σ parameter of the time-to-event model, we just need to compute and then use τ in place of σ.

```
with open('stan_programs/model4.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=stan_data)
fit = model.sample(num_samples=1024, refresh=0)
```

```
Building...
```

Hamiltonian Monte Carlo remains undaunted by our increasingly-complex posterior distributions.

```
diagnostics = util.extract_hmc_diagnostics(fit)
util.check_all_hmc_diagnostics(diagnostics)

samples = util.extract_expectand_vals(fit)
base_samples = util.filter_expectands(samples,
                                      ['omega', 'sigma',
                                       'beta', 'delta_online'],
                                      check_arrays=True)
util.check_all_expectand_diagnostics(base_samples)
```

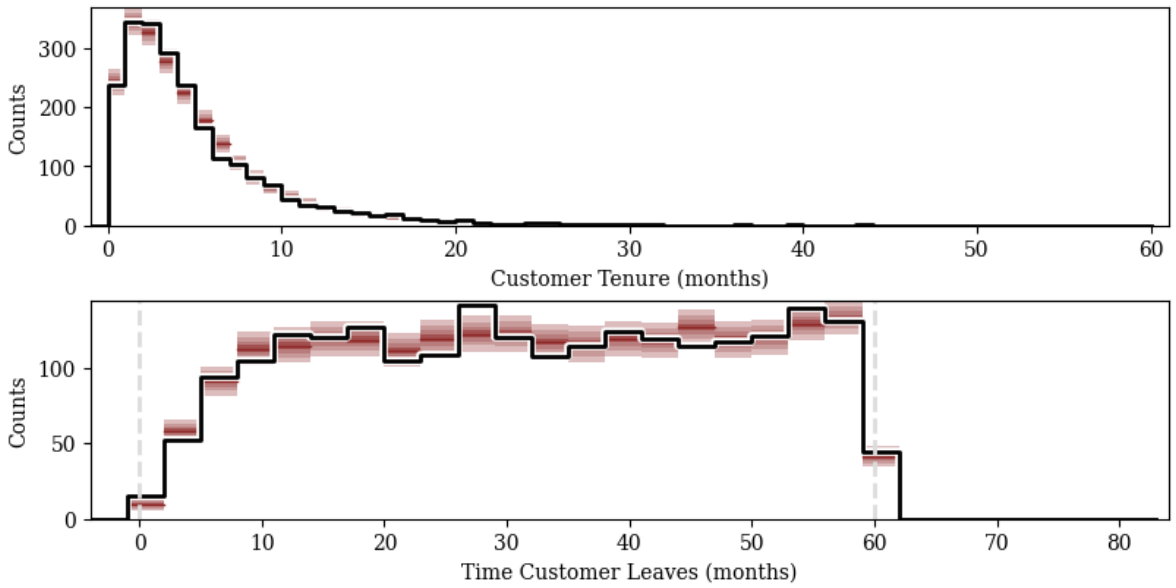
```
All Hamiltonian Monte Carlo diagnostics are consistent with accurate
Markov chain Monte Carlo.

All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

6.6 Posterior Retrodictive Checks

After all of this modeling work, the payoff is pretty rewarding. By incorporating demographic heterogeneity, our new model exhibits beautiful retrodictive agreement across all of the summaries that we have considered so far.

```
f, axarr = plot.subplots(2, 1, layout="constrained")

putil.plot_hist_quantiles(
  axarr[0], samples, 'tenure_churn_pred',
  data['T0'] - 1, data['T1'] + 1, 1,
```

```
  baseline_values=tenure_churn,
  xlabel='Customer Tenure (months)'
)


putil.plot_hist_quantiles(
  axarr[1], samples, 't_leave_churn_pred',
  data['T0'] - 3, data['T1'] + 23, 3,
  baseline_values=data['t_leave_churn'],
  xlabel="Time Customer Leaves (months)"
)
axarr[1].axvline(x=data['T0'], linewidth=2,
                 linestyle='dashed', color="#DDDDDD")
axarr[1].axvline(x=data['T1'], linewidth=2,
                 linestyle='dashed', color="#DDDDDD")

plot.show()
```

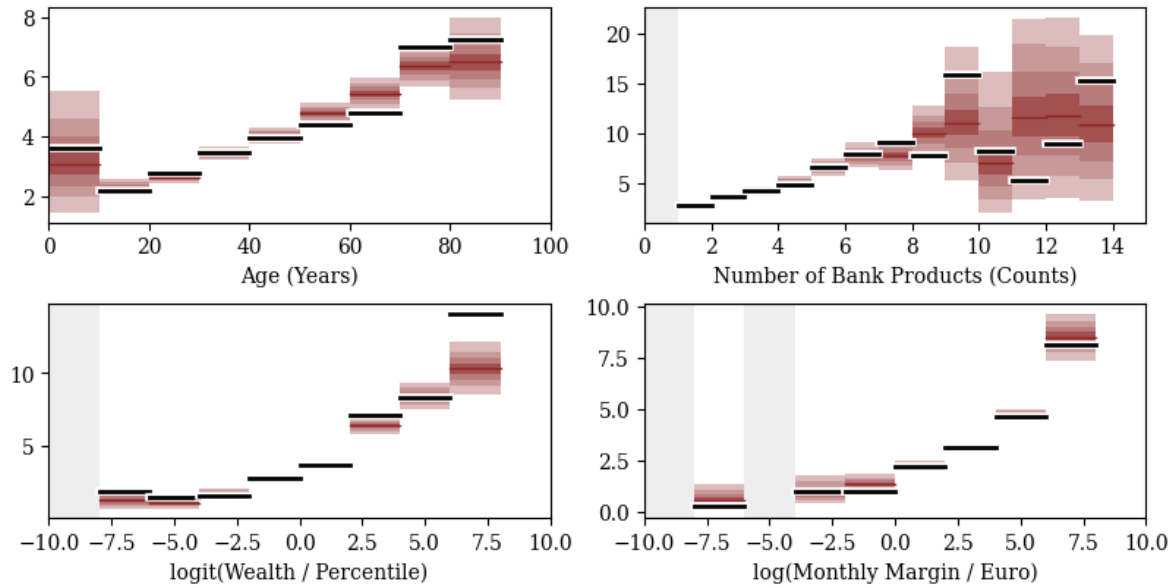


In particular, there's no indication that our demographic proportion model is inadequate, at least for the observed customer demographics.

```
pred_names = [ f'tenure_churn_pred[{n + 1}]'
               for n in range(data['N_churn']) ]

f, axarr = plot.subplots(2, 2, layout="constrained")
```

```
for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_conditional_median_quantiles(
    axarr[i1, i2], samples, pred_names,
    [ z[i] for z in Z_churn ],
    bin_min[i], bin_max[i], bin_delta[i],
    tenure_churn, residual=False,
    xlabel=z_names[i], ylabel=""
  )

plot.show()
```

```
1 conditioning values (0.04%) fell below the histogram binning.
18 conditioning values (0.80%) fell above the histogram binning.
```

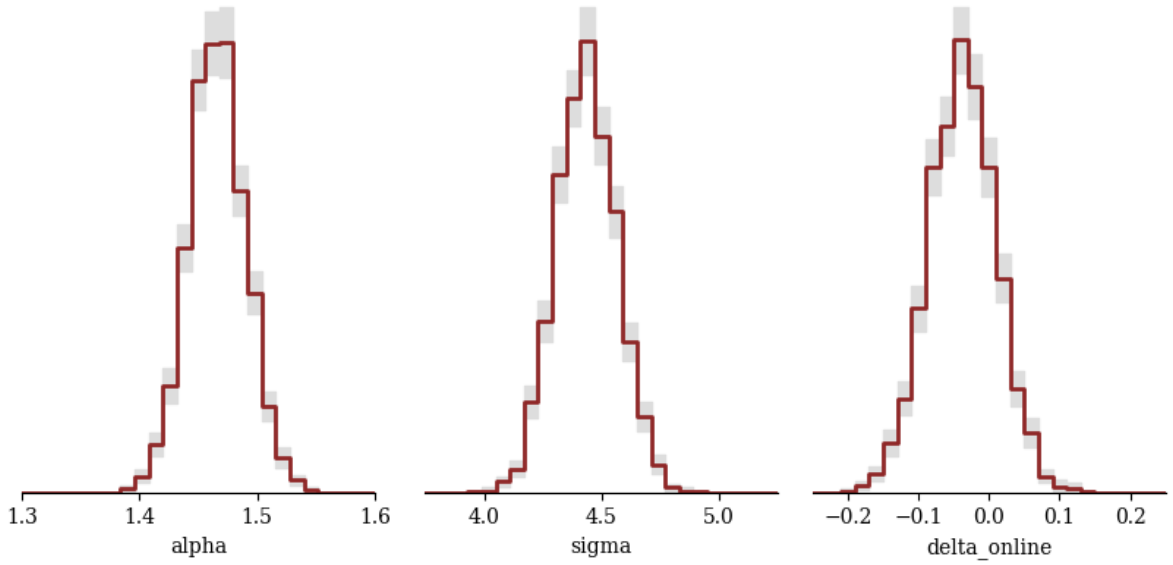


Failures of the exponential-linear proportion model would manifest as disagreement in these conditional median plots. Moreover, the shape of the residual difference between the posterior predictive and observed behaviors would directly inform the inadequacies of our initial functional model, and hence how to improve it.

6.7 Posterior Inferences

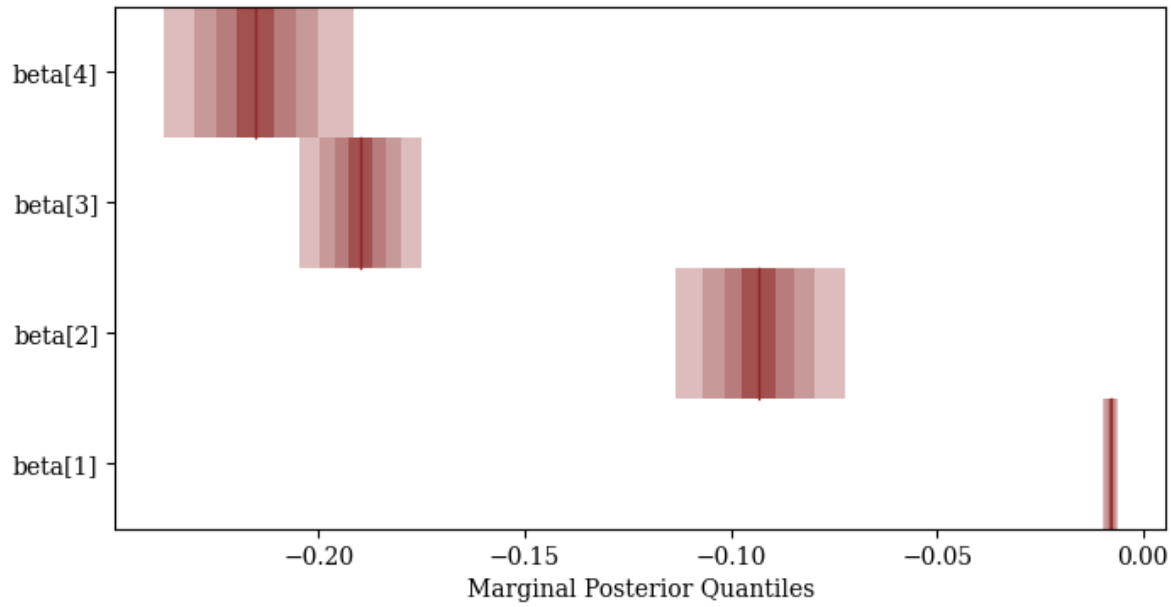
Now that none of our posterior retrodictive checks suggest any model inadequacies, we can interpret the derived posterior inferences as reasonable approximations to the true data generating behavior.

```
f, axarr = plot.subplots(1, 3, layout="constrained")

util.plot_expectand_pushforward(axarr[0], samples['alpha'],
                                25, flim=[1.3, 1.6],
                                display_name="alpha")

util.plot_expectand_pushforward(axarr[1], samples['sigma'],
                                25, flim=[3.75, 5.25],
                                display_name="sigma")

util.plot_expectand_pushforward(axarr[2], samples['delta_online'],
                                25, flim=[-0.25, 0.25],
                                display_name="delta_online")
plot.show()
```



We can see, for example, that the stimulus function, and the speed at which customers leave, deceases when any of the demographic variables move above their baseline values. Moreover, the decrease is most sensitive to increases in the logit wealth percentile.

```
names = [ f'beta[{i + 1}]'
          for i in range(data['I']) ]
putil.plot_disc_pushforward_quantiles_vert(
  plot.gca(), samples, names,
  yticklabels=names,
  xlabel="Marginal Posterior Quantiles"
)

plot.show()
```



7 Customer Lifetime Value By Segment

Now the real fun begins. Our posterior inferences can be used to not only learn about the customers observed in our study, but also make predictions for the behavior of new, unobserved customers.

For example, let's say that the bank's potential customer base has been stratified into segments, each of which is characterized by an ensemble of representative demographic configurations.

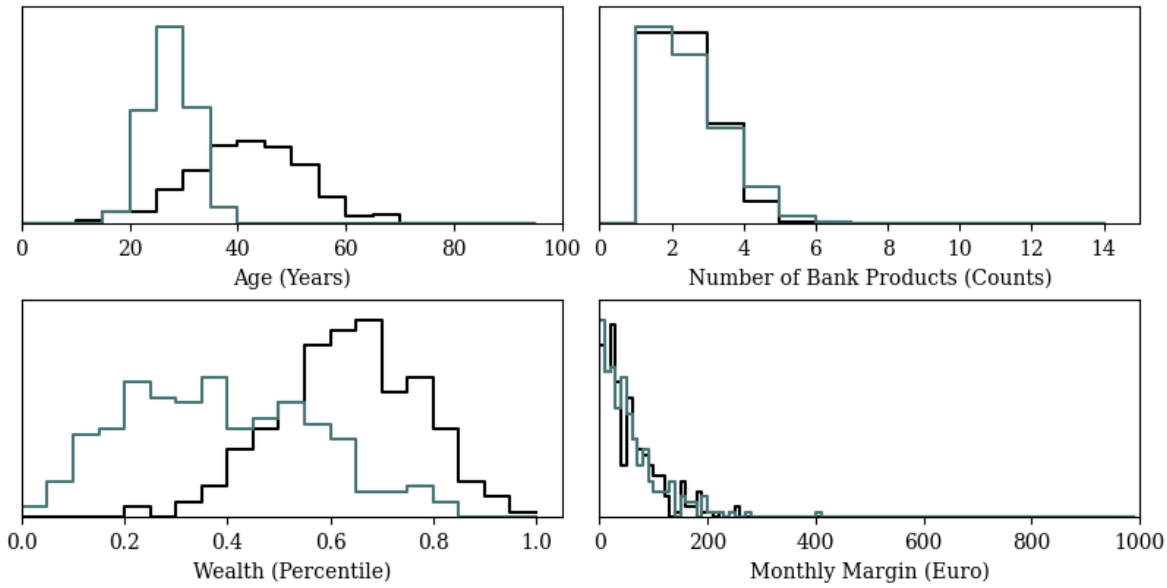
```
with open("data/customer_segmentation.json", "r") as infile:
  customer_segmentation = json.load(infile)
```

```
for i, name in enumerate(customer_segmentation['segment_names']):
  print(f'Segment {i + 1}: {name}')
```

```
Segment 1: Mass Retail
Segment 2: Private Banking
Segment 3: Wealth
Segment 4: Retired
Segment 5: Young Professional
Segment 6: Student
```

The customers in the private banking segment are older and more wealthy than the customers in the student segment.

```
bin_min = [ 0, 0, 0, 0 ]
bin_max = [ 100, 15, 1.05, 1000 ]
bin_delta = [ 5, 1, 0.05, 10 ]

f, axarr = plot.subplots(2, 2, layout="constrained")

for i in range(data['I']):
  i1 = i // 2
  i2 = i % 2

  putil.plot_line_hists(axarr[i1, i2],
                        [ x[i] for s, x in
                          zip(customer_segmentation['customer_segment_pred'],
                              customer_segmentation['X_pred'])
                          if s == 1 ],
                        [ x[i] for s, x in
                          zip(customer_segmentation['customer_segment_pred'],
                              customer_segmentation['X_pred'])
                          if s == 5 ],
                       bin_min[i], bin_max[i], bin_delta[i],
                       xlabel=data['x_names'][i])

plot.show()
```



When can use our model to predict how loyal the customers in each of these segments might be, and hence what their lifetime value to the bank might be. To be meticulous, we have to account for future discounting over the lifetime of a customer.

```
stan_data['S'] = customer_segmentation['S']

stan_data['segment_pred'] = \
  customer_segmentation['customer_segment_pred']
```

```
stan_data['value_per_month'] = [
  50,  # Mass Retail
  75, # Private Banking
  80, # Wealth
  45, # Retired
  60,  # Young Professional
  40,  # Student
]

stan_data['zero_rate'] = 0.10
```

If we also know how expensive customers in each of these segments are to acquire, then we can use these inferences to identify the most most profitable type of customers.

```
stan_data['acq_cost'] = [
  150,  # Mass Retail
  310, # Private Banking
  260, # Wealth
  175, # Retired
  175,  # Young Professional
  125,  # Student
]
```

7.1 Posterior Quantification

Because our inferences are the same, the basic structure of the full Bayesian model is the same. We just have to add the derived predictions (Figure 6).

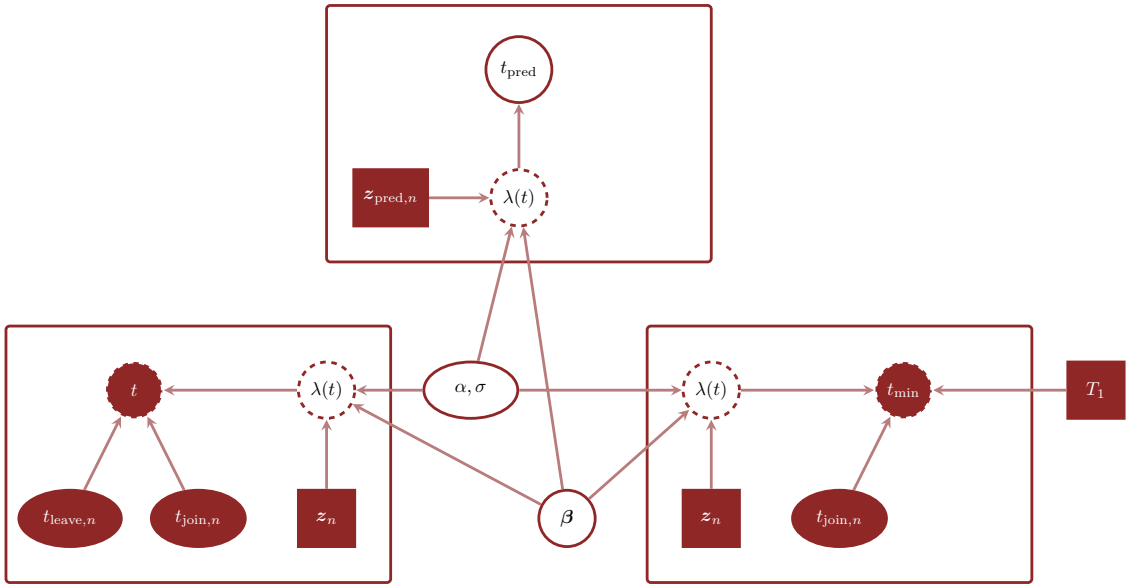


Figure 6: Once we've learned the consistent configurations of our model, we can make predictions for new customers.

In our Stan program, we need to pass in the segment demographics and then calculate predictions in the `generated quantities` block.

```
stan_data['N_pred'] = customer_segmentation['N_pred']
stan_data['Z_pred'] = [ transform_covariates(x)
                        for x in
```

```
                          customer_segmentation['X_pred'] ]
stan_data['online_banking_pred'] = \
  customer_segmentation['online_banking_pred']
```

```
with open('stan_programs/model5.stan', 'r') as file:
  stan_program = file.read()
model = stan.build(stan_program,
                   random_seed=8438338,
                   data=stan_data)
fit = model.sample(num_samples=1024, refresh=0)
```

```
Building...
```

Although we're quantifying the same posterior distribution, the modified `generated quantities` block does affect how pseudo-random numbers are consumed as Hamiltonian Monte Carlo runs. This, in turn, can affect the performance of the posterior quantification. Usually this isn't an issue, but to be safe we'll want to check our computational diagnostics again. Fortunately, there are no indications of problems here.

```
diagnostics = util.extract_hmc_diagnostics(fit)
util.check_all_hmc_diagnostics(diagnostics)

samples = util.extract_expectand_vals(fit)
base_samples = util.filter_expectands(samples,
                                      ['omega', 'sigma',
                                       'beta', 'delta_online'],
                                      check_arrays=True)
util.check_all_expectand_diagnostics(base_samples)
```

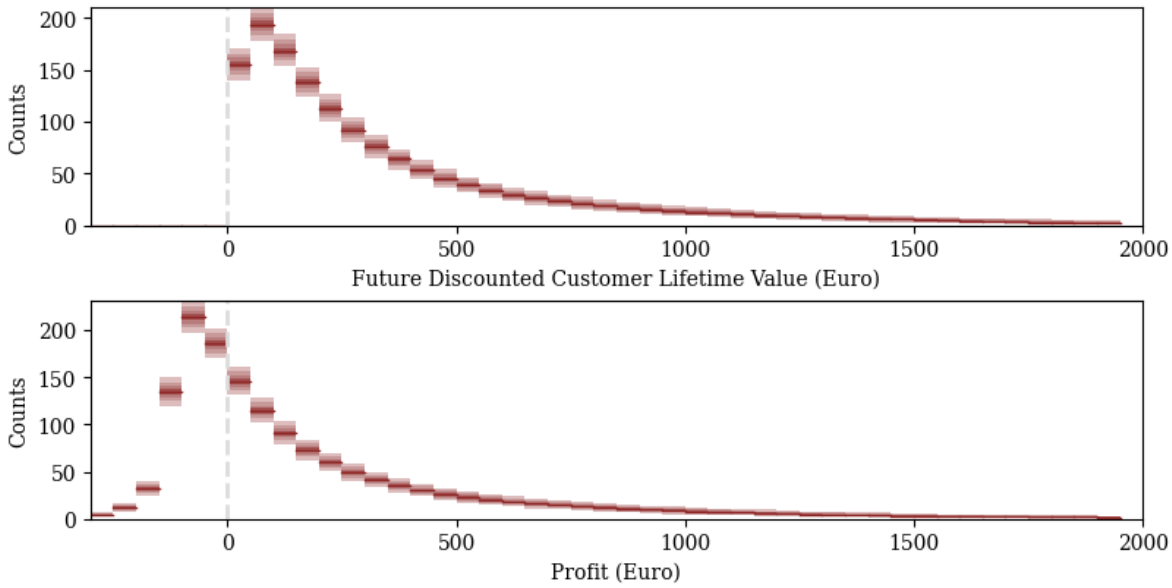
```
All Hamiltonian Monte Carlo diagnostics are consistent with accurate
Markov chain Monte Carlo.

All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

7.2 Posterior Inferences

By inferring customer lifetime value for each customer in each of the segments, we have a wealth of information that we can use for various forecasting tasks.

```
f, axarr = plot.subplots(2, 1, layout="constrained")

putil.plot_hist_quantiles(
  axarr[0], samples, 'clv_discount',
  -300, 2000, 50,
  xlabel='Future Discounted Customer Lifetime Value (Euro)'
)
axarr[0].axvline(x=0, linewidth=2, linestyle="dashed",
                 color='#DDDDDD')

putil.plot_hist_quantiles(
  axarr[1], samples, 'profit',
  -300, 2000, 50,
  xlabel='Profit (Euro)'
)
axarr[1].axvline(x=0, linewidth=2, linestyle="dashed",
                 color='#DDDDDD')

plot.show
```

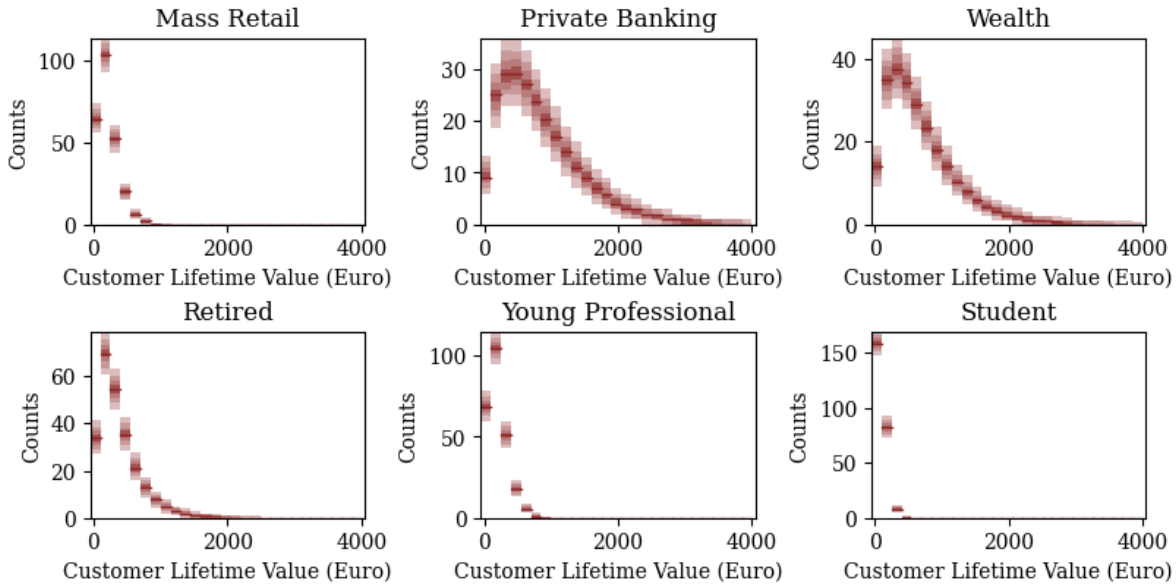
```
116785 predictive values (1.90%) fell above the histogram binning.
1345 predictive values (0.02%) fell below the histogram binning.
73020 predictive values (1.19%) fell above the histogram binning.
```



Namely, we can quantify the differences in customer lifetime value across the customer segments. For example, customers in the private banking, wealth, and retired segments offer much larger lifetime values than those in the mass retail, young professional, and student segments.

```
segments = customer_segmentation['customer_segment_pred']

f, axarr = plot.subplots(2, 3, layout="constrained")

for k in range(customer_segmentation['S']):
  k1 = k // 3
  k2 = k % 3

  filt_names = [ f'clv_discount[{n + 1}]'
                 for n in range(customer_segmentation['N_pred'])
                 if segments[n] == k + 1 ]
  filt_samples = util.filter_expectands(samples, filt_names)

  putil.plot_hist_quantiles(
    axarr[k1, k2], filt_samples, 'clv_discount',
    0, 4000, 150,
    xlabel='Customer Lifetime Value (Euro)',
    title=customer_segmentation['segment_names'][k]
  )

plot.show()
```

```
2927 predictive values (0.29%) fell above the histogram binning.
2497 predictive values (0.24%) fell above the histogram binning.
32 predictive values (0.00%) fell above the histogram binning.
```

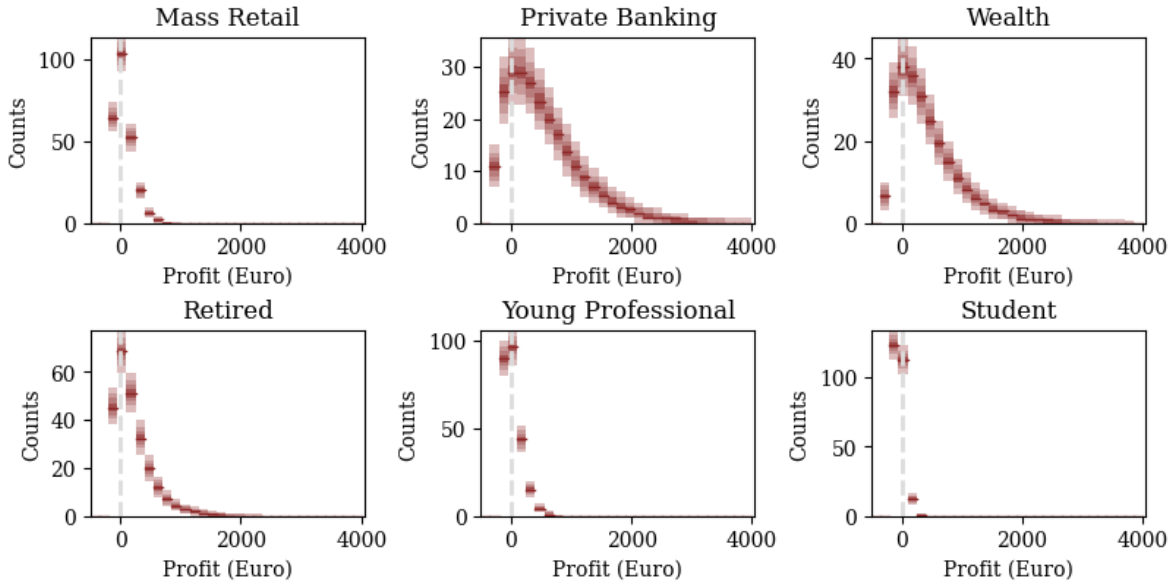


In order to compare potential profitability, however, we have to account for not only lifetime value but also acquisition costs.

```
segments = customer_segmentation['customer_segment_pred']

f, axarr = plot.subplots(2, 3, layout="constrained")

for k in range(customer_segmentation['S']):
  k1 = k // 3
  k2 = k % 3

  filt_names = [ f'profit[{n + 1}]'
                 for n in range(customer_segmentation['N_pred'])
                 if segments[n] == k + 1 ]
  filt_samples = util.filter_expectands(samples, filt_names)

  putil.plot_hist_quantiles(
    axarr[k1, k2], filt_samples, 'profit',
    -450, 4000, 150,
    xlabel='Profit (Euro)',
    title=customer_segmentation['segment_names'][k]
  )
  axarr[k1, k2].axvline(x=0, linewidth=2, linestyle="dashed",
                        color='#DDDDDD')
```

```
plot.show()
```

```
1829 predictive values (0.18%) fell above the histogram binning.
1845 predictive values (0.18%) fell above the histogram binning.
22 predictive values (0.00%) fell above the histogram binning.
```



Because we're quantifying *populations* of customer behaviors, we can examine not only the average profitability within a segment but also the profit volatility. Here we see that, compared to customers in the private banking, wealth, and retired segments, mass retail customers offer relatively small but stable profits. The opportunities in that segment are lower risk, but also much lower reward. On the other hand, student and young professional customers are relatively expensive to acquire in comparison to the meager profits they tend to provide.

These inferences could be used to directly optimize marketing strategies, inform reports about optimal marketing strategies, and more. Because we're already quantifying our inferences and predictions probabilistically, we can even incorporate uncertainty in the projected monthly values and acquisition costs. The possibilities are endless!

8 Conclusion

Accurately inferring customer churn is more subtle than it might first appear. If we don't properly account for active customers, for example, then we will tend to overestimate churn.

Properly accounting for both churned and active customers requires careful modeling assumptions that we must validate. Once we've invested the time to build an adequate model, however, we have access to sophisticated posterior inferences than can inform a diversity of important decisions.

License

Original Computing Environment

```
from watermark import watermark
print(watermark())
```

```
Last updated: 2026-06-04T12:24:27.075023-04:00

Python implementation: CPython
Python version       : 3.12.10
IPython version      : 9.6.0

Compiler    : Clang 13.0.0 (clang-1300.0.29.30)
OS          : Darwin
Release     : 24.6.0
Machine     : x86_64
Processor   : i386
CPU cores   : 16
Architecture: 64bit
```

```
print(watermark(packages="matplotlib, numpy, json, stan"))
```

```
matplotlib: 3.10.7
 numpy    : not installed
 json     : not installed
 stan     : not installed
```

Stan
Program 1 `model1\_prior.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real gamma) {
    return log(gamma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real gamma) {
    return gamma * t;
  }

  // Log survival function
  real log_survival(real t, real gamma) {
    return -Lambda(t, gamma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real gamma) {
    return l / gamma;
  }

  // Inverse survival function
  real inv_survival(real u, real gamma) {
    return Lambda_inv(-log(u), gamma);
  }
}

data {
  int<lower=0> N;
  real<lower=0> t_upper;
}

parameters {
  real<lower=0> eta; // Inverse scale parameter (Months)
}

model {
  // Prior Model
  // 0 <~ eta <~ 50
  target += normal_lpdf(eta | 0, 1 / 2.57);
}

generated quantities {
  array[N] real<lower=0> tenure_pred;
  real upper_tail_p_hat = 0;

  for (n in 1:N) {
    tenure_pred[n]
      = inv_survival(uniform_rng(0, 1), 1 / eta);
    if (tenure_pred[n] > t_upper)
```

Stan
Program 2 `model1.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real gamma) {
    return log(gamma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real gamma) {
    return gamma * t;
  }

  // Log survival function
  real log_survival(real t, real gamma) {
    return -Lambda(t, gamma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real gamma) {
    return l / gamma;
  }

  // Inverse survival function
  real inv_survival(real u, real gamma) {
    return Lambda_inv(-log(u), gamma);
  }
}

data {
  real           T0; // Months
  real<lower=T0> T1; // Months

  // Churned customers
  int<lower=0> N_churn;
  array[N_churn] real<lower=T0, upper=T1> t_join_churn;  // Months
  array[N_churn] real<lower=T0, upper=T1> t_leave_churn; // Months
}

parameters {
  real<lower=0> eta; // Inverse scale parameter (Months)
}

transformed parameters {
  real<lower=0> gamma = 1 / eta;
}

model {
  // Prior Model
  // 0 <~ eta <~ 40
  target += normal_lpdf(eta | 0, 1 / 2.57);
```

Stan
Program 3 `model2.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real gamma) {
    return log(gamma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real gamma) {
    return gamma * t;
  }

  // Log survival function
  real log_survival(real t, real gamma) {
    return -Lambda(t, gamma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real gamma) {
    return l / gamma;
  }

  // Inverse survival function
  real inv_survival(real u, real gamma) {
    return Lambda_inv(-log(u), gamma);
  }
}

data {
  real           T0; // Months
  real<lower=T0> T1; // Months

  // Churned customers
  int<lower=0> N_churn;
  array[N_churn] real<lower=T0, upper=T1> t_join_churn;  // Months
  array[N_churn] real<lower=T0, upper=T1> t_leave_churn; // Months

  // Active customers
  int<lower=0> N_active;
  array[N_active] real<lower=T0, upper=T1> t_join_active; // Months
}

parameters {
  real<lower=0> eta; // Inverse scale parameter (Months)
}

transformed parameters {
  real<lower=0> gamma = 1 / eta;
}

model {
```

Stan
Program 4 `model3\_prior.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real alpha, real sigma) {
    return log(alpha) - log(sigma) + (alpha - 1) * log(t / sigma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real alpha, real sigma) {
    return pow(t / sigma, alpha);
  }

  // Log survival function
  real log_survival(real t, real alpha, real sigma) {
    return -Lambda(t, alpha, sigma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real alpha, real sigma) {
    return sigma * pow(l, 1 / alpha);
  }

  // Inverse survival function
  real inv_survival(real u, real alpha, real sigma) {
    return Lambda_inv(-log(u), alpha, sigma);
  }
}

data {
  int<lower=0> N;
  real<lower=0> t_lower;
  real<lower=0> t_upper;
}

parameters {
  real<lower=0> omega; // Inverse shape parameter (Unitless)
  real<lower=0> sigma; // Scale parameter (Months)
}

model {
  // Prior Model
  // 1 <~ alpha
  // 0 <~ omega = 1 / alpha <~ 1 / 1
  target += normal_lpdf(omega | 0, 1 / 2.57);

  // 0 <~ sigma <~ 75
  target += normal_lpdf(sigma | 0, 75 / 2.57);
}

generated quantities {
  array[N] real<lower=0> tenure_pred;
```

Stan
Program 5 `model3.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real alpha, real sigma) {
    return log(alpha) - log(sigma) + (alpha - 1) * log(t / sigma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real alpha, real sigma) {
    return pow(t / sigma, alpha);
  }

  // Log survival function
  real log_survival(real t, real alpha, real sigma) {
    return -Lambda(t, alpha, sigma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real alpha, real sigma) {
    return sigma * pow(l, 1 / alpha);
  }

  // Inverse survival function
  real inv_survival(real u, real alpha, real sigma) {
    return Lambda_inv(-log(u), alpha, sigma);
  }
}

data {
  real           T0; // Months
  real<lower=T0> T1; // Months

  // Churned customers
  int<lower=0> N_churn;
  array[N_churn] real<lower=T0, upper=T1> t_join_churn;  // Months
  array[N_churn] real<lower=T0, upper=T1> t_leave_churn; // Months

  // Active customers
  int<lower=0> N_active;
  array[N_active] real<lower=T0, upper=T1> t_join_active; // Months
}

parameters {
  real<lower=0> omega; // Inverse shape parameter (Unitless)
  real<lower=0> sigma; // Scale parameter (Months)
}

transformed parameters {
  real<lower=0> alpha = 1 / omega;
}
```

Stan
Program 6 `model4\_prior.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real alpha, real sigma) {
    return log(alpha) - log(sigma) + (alpha - 1) * log(t / sigma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real alpha, real sigma) {
    return pow(t / sigma, alpha);
  }

  // Log survival function
  real log_survival(real t, real alpha, real sigma) {
    return -Lambda(t, alpha, sigma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real alpha, real sigma) {
    return sigma * pow(l, 1 / alpha);
  }

  // Inverse survival function
  real inv_survival(real u, real alpha, real sigma) {
    return Lambda_inv(-log(u), alpha, sigma);
  }
}

data {
  real<lower=0> t_lower;
  real<lower=0> t_upper;

  int I;            // Number of covariates
  row_vector[I] z0; // Covariate baselines

  // Churned customers
  int<lower=0> N_churn;
  matrix[N_churn, I] Z_churn;
  array[N_churn] int online_banking_churn;
}

parameters {
  real<lower=0> omega; // Inverse shape parameter (Unitless)
  real<lower=0> sigma; // Scale parameter (Months)

  vector[I] beta;    // Covariate slopes (1 / covariate units)
  real delta_online; // Online banking contribution
}

transformed parameters {
  real<lower=0> alpha = 1 / omega;
```

Stan
Program 7 `model4.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real alpha, real sigma) {
    return log(alpha) - log(sigma) + (alpha - 1) * log(t / sigma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real alpha, real sigma) {
    return pow(t / sigma, alpha);
  }

  // Log survival function
  real log_survival(real t, real alpha, real sigma) {
    return -Lambda(t, alpha, sigma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real alpha, real sigma) {
    return sigma * pow(l, 1 / alpha);
  }

  // Inverse survival function
  real inv_survival(real u, real alpha, real sigma) {
    return Lambda_inv(-log(u), alpha, sigma);
  }
}

data {
  real           T0; // Months
  real<lower=T0> T1; // Months

  int I;            // Number of covariates
  row_vector[I] z0; // Covariate baselines

  // Churned customers
  int<lower=0> N_churn;
  array[N_churn] real<lower=T0, upper=T1> t_join_churn;  // Months
  array[N_churn] real<lower=T0, upper=T1> t_leave_churn; // Months

  matrix[N_churn, I] Z_churn;
  array[N_churn] int online_banking_churn;

  // Active customers
  int<lower=0> N_active;
  array[N_active] real<lower=T0, upper=T1> t_join_active; // Months

  matrix[N_active, I] Z_active;
  array[N_active] int online_banking_active;
}
```

Stan
Program 8 `model5.stan`

```
functions {
  // Log stimulus function
  real log_lambda(real t, real alpha, real sigma) {
    return log(alpha) - log(sigma) + (alpha - 1) * log(t / sigma);
  }

  // Cumulative stimulus function
  real Lambda(real t, real alpha, real sigma) {
    return pow(t / sigma, alpha);
  }

  // Log survival function
  real log_survival(real t, real alpha, real sigma) {
    return -Lambda(t, alpha, sigma);
  }

  // Inverse cumulative stimulus function
  real Lambda_inv(real l, real alpha, real sigma) {
    return sigma * pow(l, 1 / alpha);
  }

  // Inverse survival function
  real inv_survival(real u, real alpha, real sigma) {
    return Lambda_inv(-log(u), alpha, sigma);
  }
}

data {
  real           T0; // Months
  real<lower=T0> T1; // Months

  int I;            // Number of covariates
  row_vector[I] z0; // Covariate baselines

  // Churned customers
  int<lower=0> N_churn;
  array[N_churn] real<lower=T0, upper=T1> t_join_churn;  // Months
  array[N_churn] real<lower=T0, upper=T1> t_leave_churn; // Months

  matrix[N_churn, I] Z_churn;
  array[N_churn] int online_banking_churn;

  // Active customers
  int<lower=0> N_active;
  array[N_active] real<lower=T0, upper=T1> t_join_active; // Months

  matrix[N_active, I] Z_active;
  array[N_active] int online_banking_active;

  // Prospective customers
```