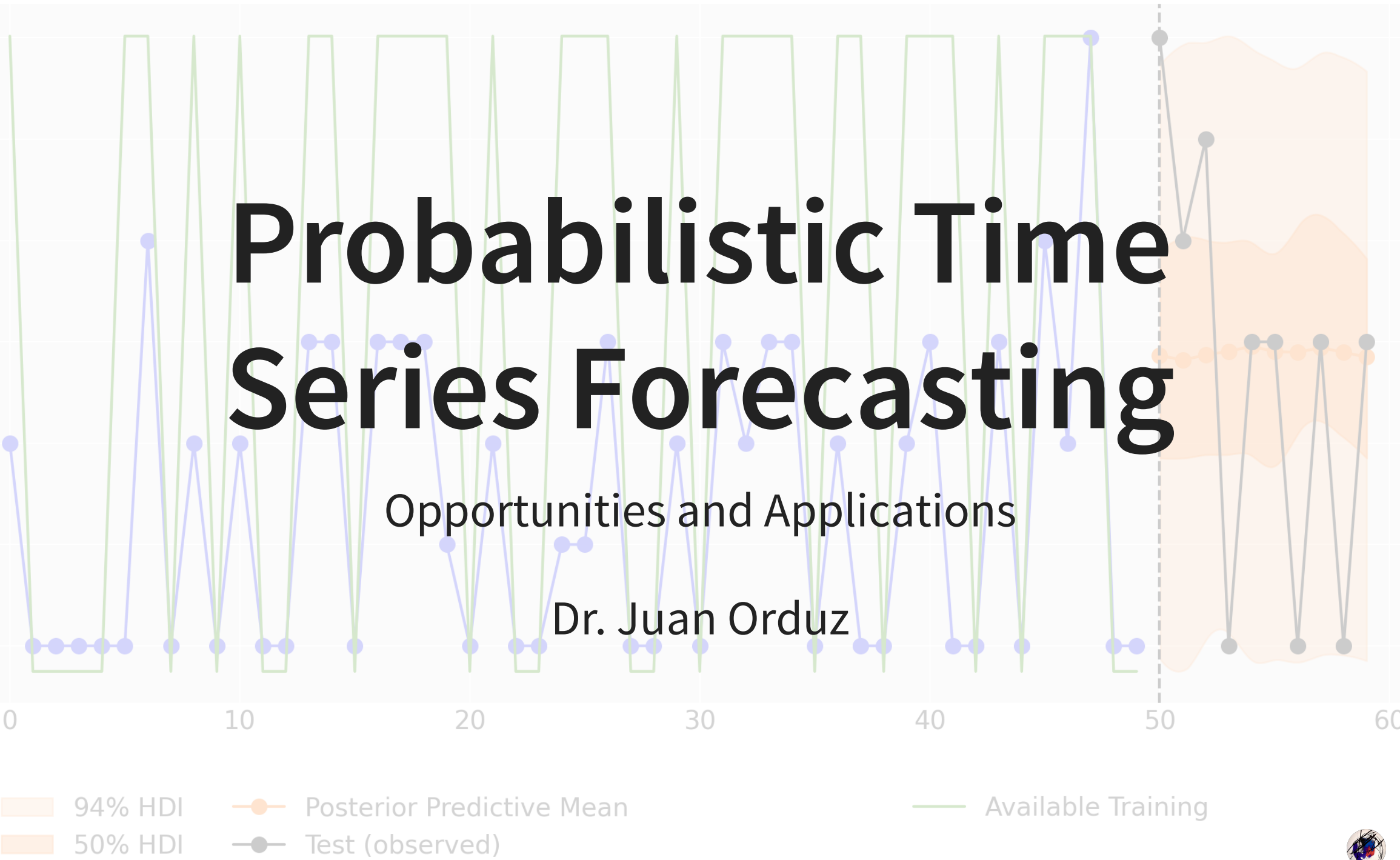


Probabilistic Time Series Forecasting

Opportunities and Applications

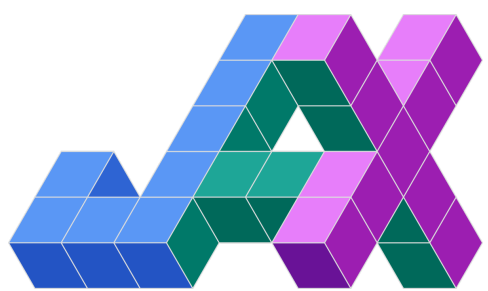
Dr. Juan Orduz

Outline

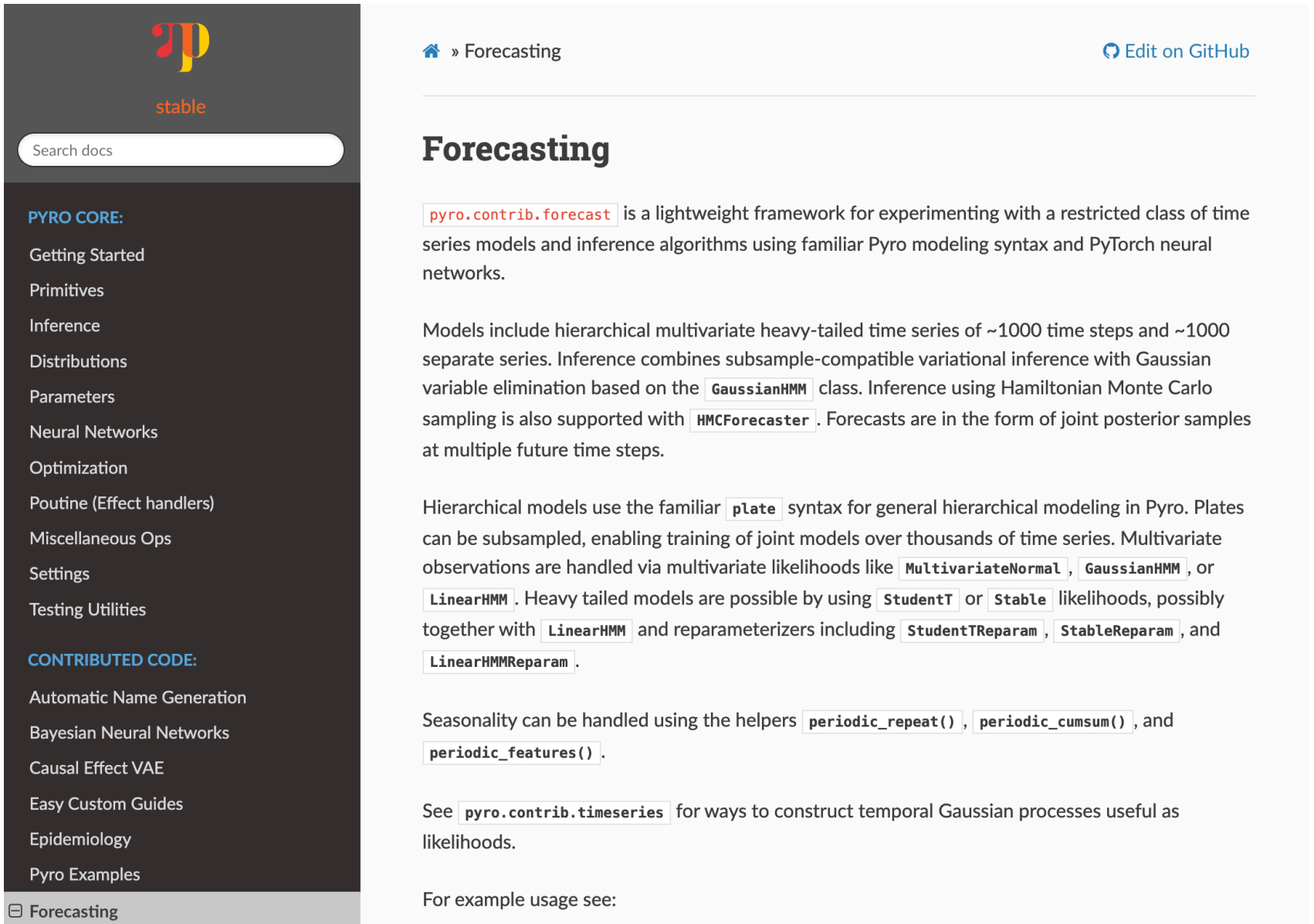
Why Probabilistic Forecasting?

- **Interpretability**
  - Trust on the results
- **Uncertainty quantification**
  - Risk Assessment
  - Decision Making
- **Customization**
  - Feature engineering
  - Special constraints
  - Calibrate with domain knowledge
- **Scale**
  - Good results on production environments
  - Take advantage of GPU

NumPyro is a lightweight probabilistic programming library that provides a NumPy backend for Pyro. It relies on JAX for automatic differentiation and JIT compilation to GPU / CPU.



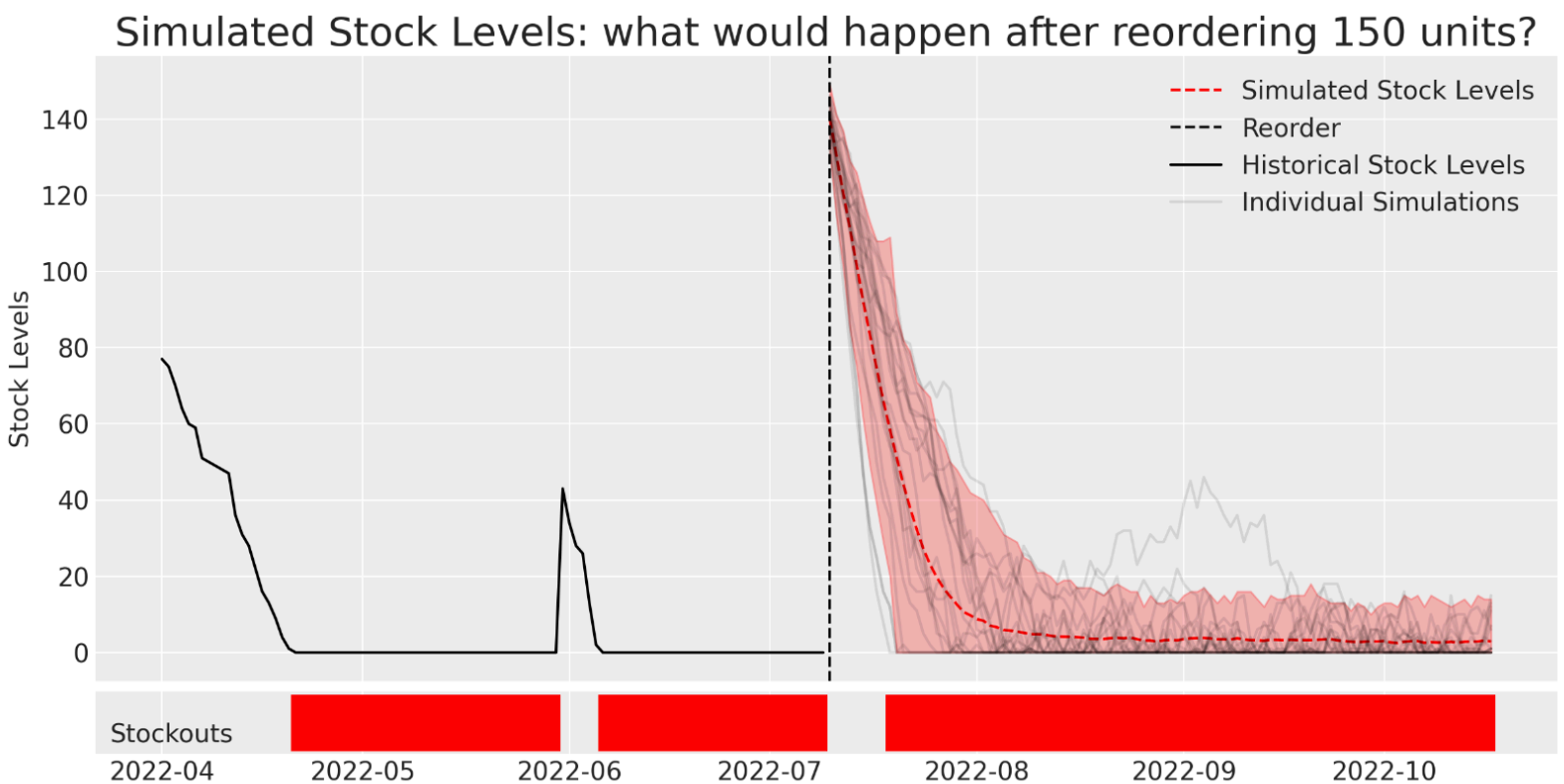
Pyro Forecasting Module 🔥



Kyle Caron

Modeling Anything With First Principles: Demand under extreme stockouts

TIME SERIES | DEMAND MODELING | CAUSAL INFERENCE | SUPPLY CHAIN | DISCRETE CHOICE | SURVIVAL ANALYSIS



NumPyro - SGT Example Model 🫠

Model

The model we are going to use is called **Seasonal, Global Trend**, which when tested on 3003 time series of the M-3 competition, has been known to outperform other models originally participating in the competition:

$$
\begin{aligned}
\text{exp-val}_t &= \text{level}_{t-1} + \text{coef-trend} \times \text{level}_{t-1}^{\text{pow-trend}} + \text{s}_t \times \text{level}_{t-1}^{\text{pow-season}}, \\
\sigma_t &= \sigma \times \text{exp-val}_t^{\text{powx}} + \text{offset}, \\
y_t &\sim \text{StudentT}(\nu, \text{exp-val}_t, \sigma_t)
\end{aligned}
$$

, where `level` and `s` follows the following recursion rules:

$$
\begin{aligned}
\text{level-p} &= \begin{cases} y_t - \text{s}_t \times \text{level}_{t-1}^{\text{pow-season}} & \text{if } t \leq \text{seasonality}, \\ \text{Average}\left[y(t-\text{seasonality}+1), \ldots, y(t)\right] & \text{otherwise}, \end{cases} \\
\text{level}_t &= \text{level-sm} \times \text{level-p} + (1 - \text{level-sm}) \times \text{level}_{t-1}, \\
\text{s}_{t+\text{seasonality}} &= \text{s-sm} \times \frac{y_t - \text{level}_t}{\text{level}_{t-1}^{\text{pow-trend}}} + (1 - \text{s-sm}) \times \text{s}_t.
\end{aligned}
$$

😅 ...

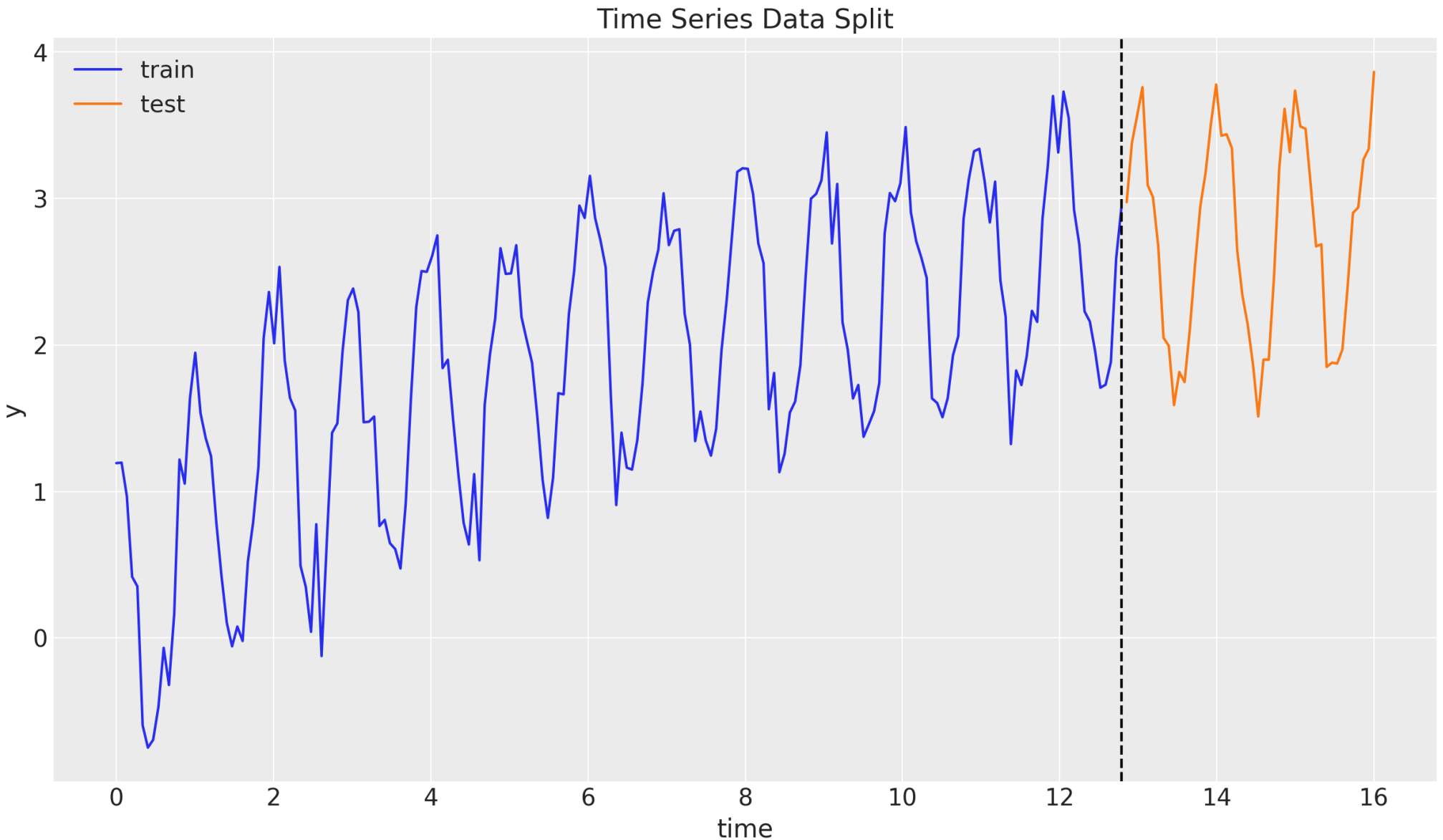
Scan ⭐

An efficient implementation of `for` loops

```python
def scan(f, init, xs):
  """Pure Python implementation of scan.

  Parameters
  ----------
  f : A  a Python function to be scanned.
  init : An initial loop carry value
  xs : The value over which to scan along the leading axis.
  """
  carry = init
  ys = []
  for x in xs:
    carry, y = f(carry, x)
    ys.append(y)
  return carry, np.stack(ys)
```

We need recursive relationships! $y_t \longmapsto y_{t+1}$

Example: Exponential Smoothing



Example: Exponential Smoothing

$$\hat{y}_{t+h|t} = l_t$$
$$l_t = \alpha y_t + (1-\alpha) l_{t-1}$$

- y_t is the observed value at time t.
- $\hat{y}_{t+h|t}$ is the forecast of the value at time $t+h$ given the information up to time t.
- l_t is the level at time t.
- α is the smoothing parameter. It is a value between 0 and 1.

Example: Exponential Smoothing

$$\hat{y}_{t+h|t} = l_t$$

$$l_t = \alpha y_t + (1 - \alpha) l_{t-1}$$

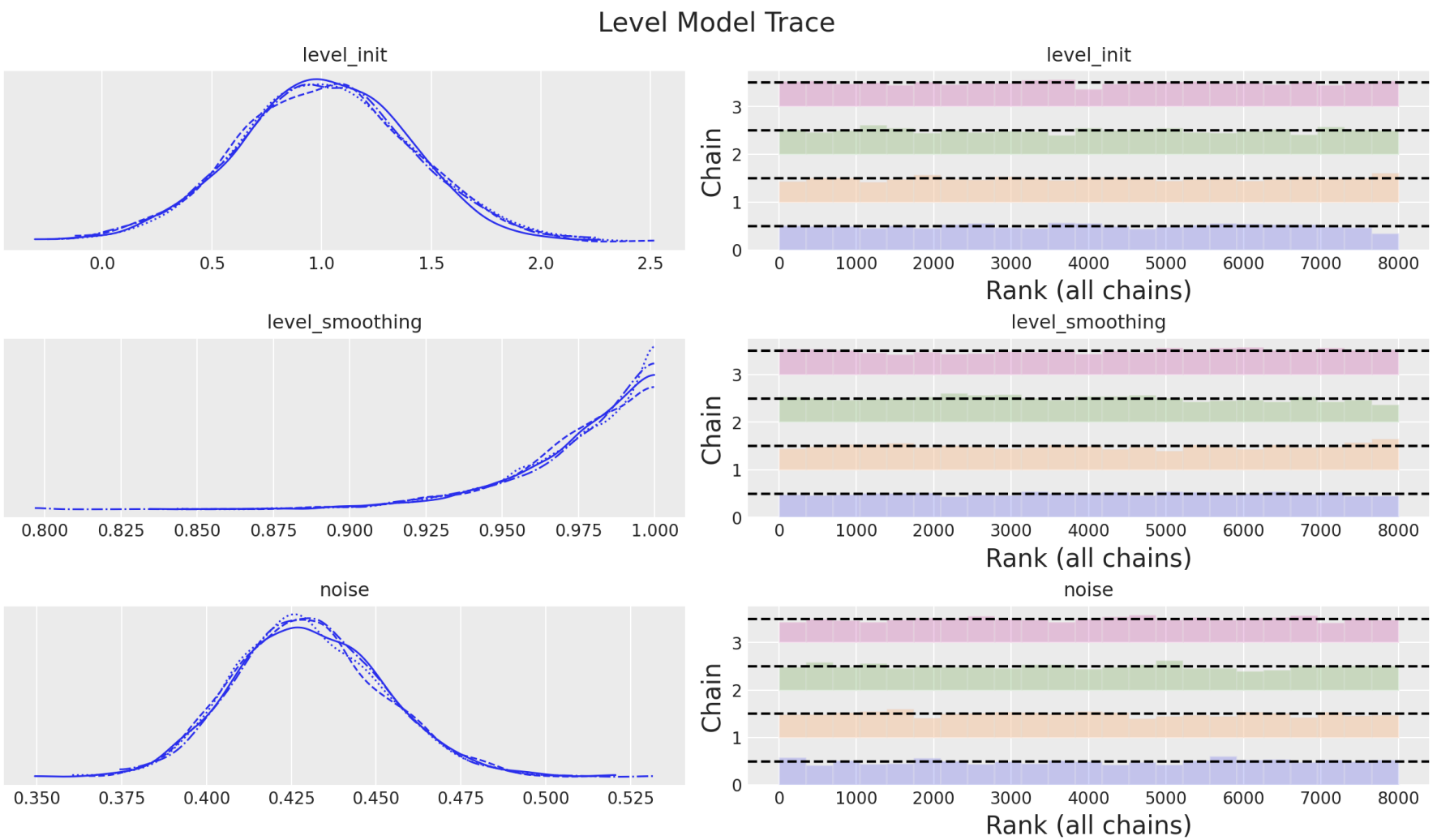
```
    def transition_fn(carry, t):
        previous_level = carry

        level = jnp.where(
            t < t_max,
            level_smoothing * y[t] + (1 - level_smoothing) * previous_level
            previous_level,
        )

        mu = previous_level
        pred = numpyro.sample("pred", dist.Normal(loc=mu, scale=noise))

        return level, pred
```

Example: Exponential Smoothing

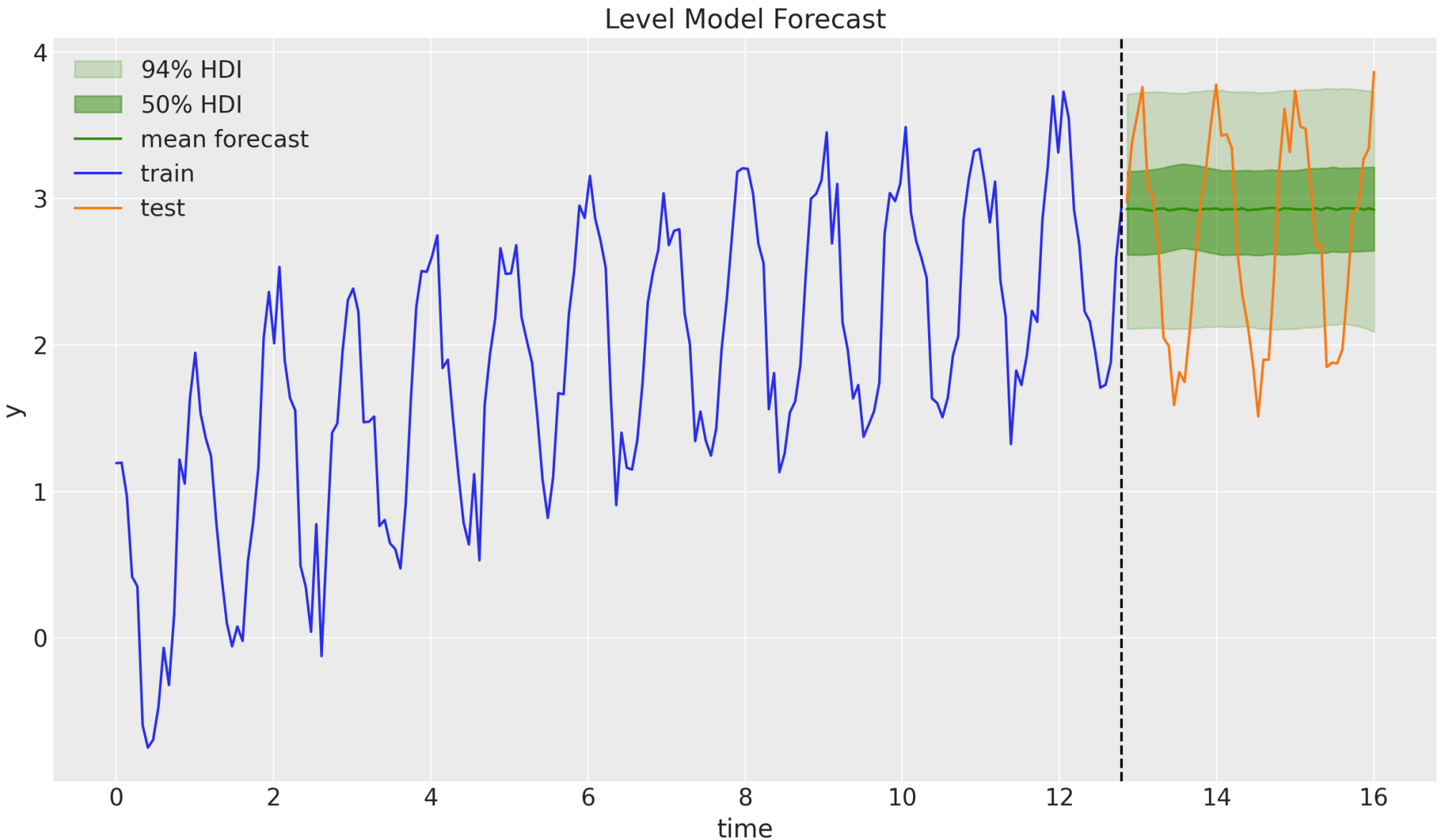
```python
def level_model(y: ArrayLike, future: int = 0) -> None:
    # Get time series length
    t_max = y.shape[0]

    # --- Priors ---
    ## Level
    level_smoothing = numpyro.sample(
        "level_smoothing", dist.Beta(concentration1=1, concentration0=1)
    )
    level_init = numpyro.sample("level_init", dist.Normal(loc=0, scale=1))

    ## Noise
    noise = numpyro.sample("noise", dist.HalfNormal(scale=1))

    # --- Transition Function ---
    def transition_fn(carry, t):
        . . .

    # --- Run Scan ---
    with numpyro.handlers.condition(data={"pred": y}):
        _, preds = scan(
```

Example: Exponential Smoothing

Posterior Distribution Parameters

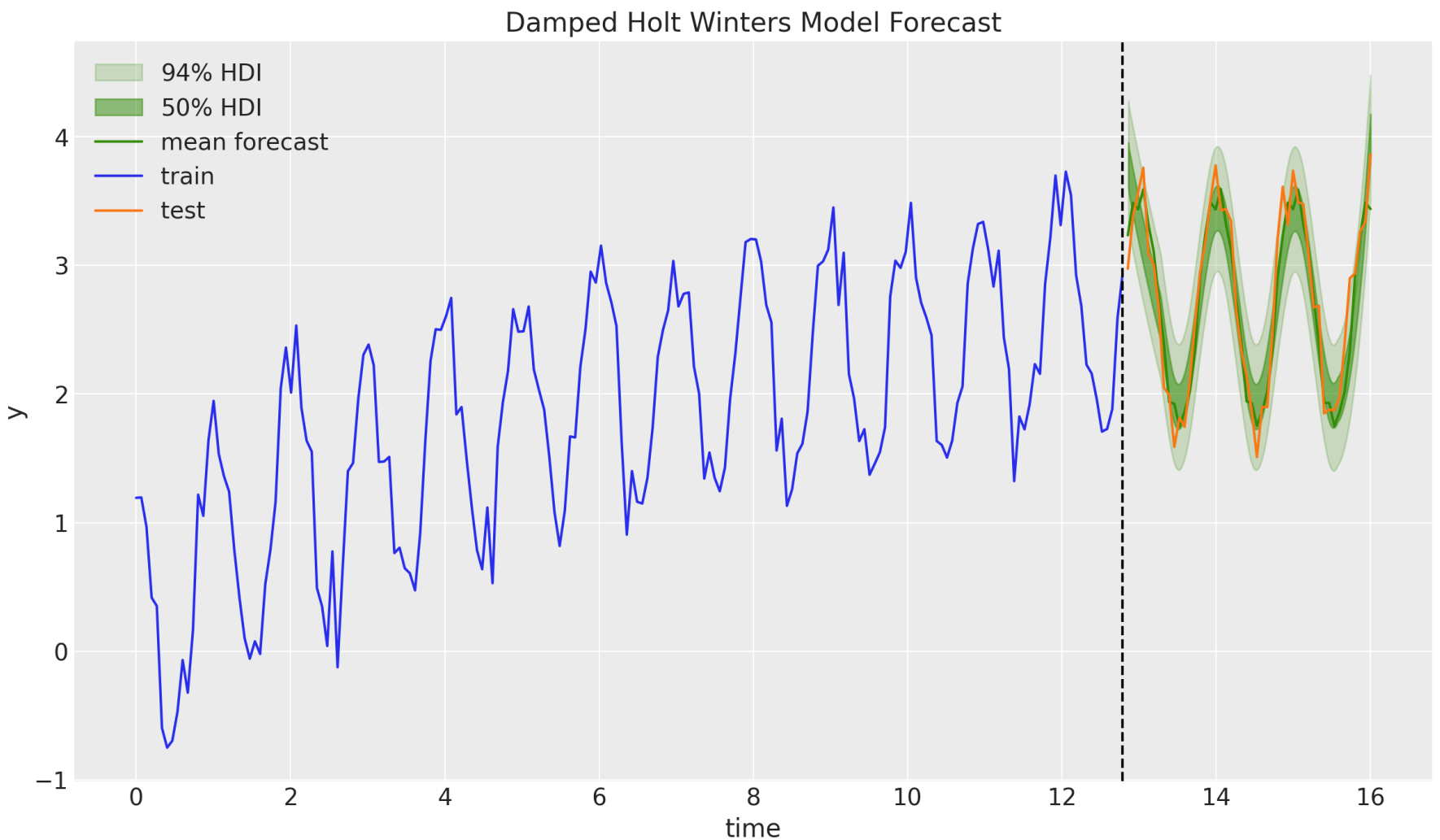


Example: Exponential Smoothing



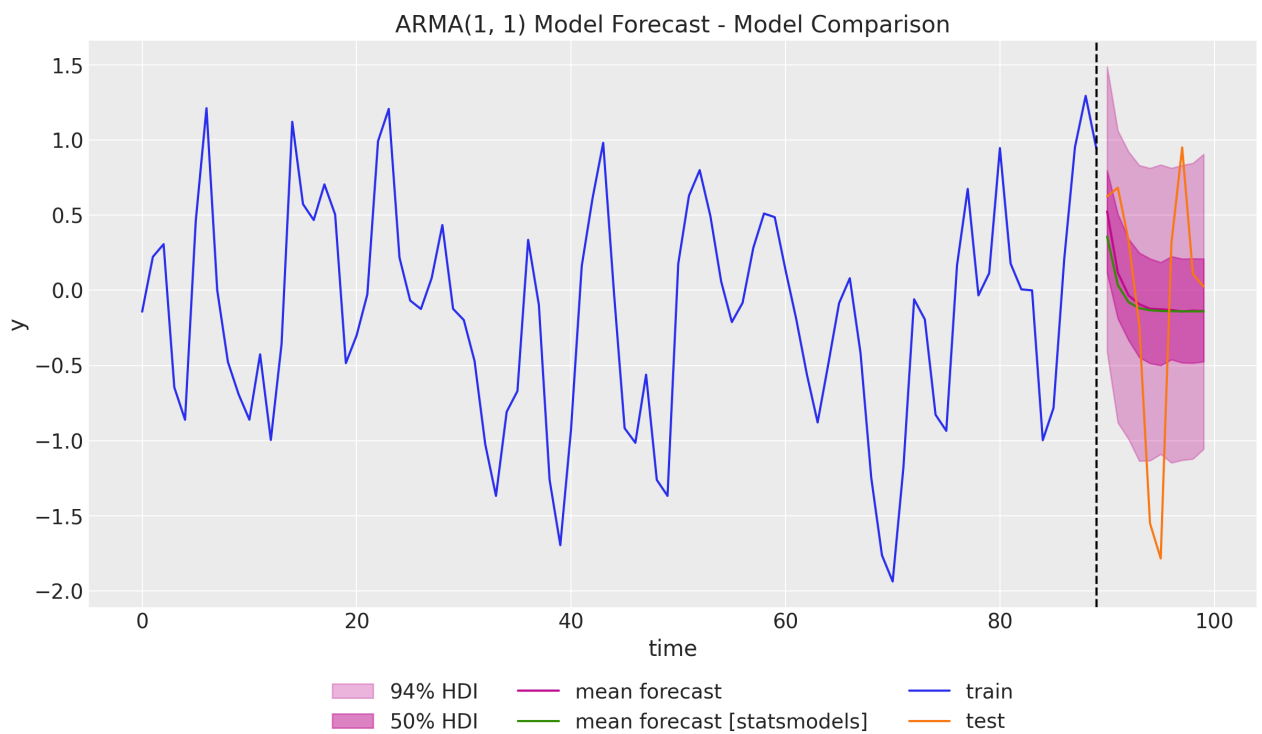
Example: Exponential Smoothing

Trend + Seasonal + Damped



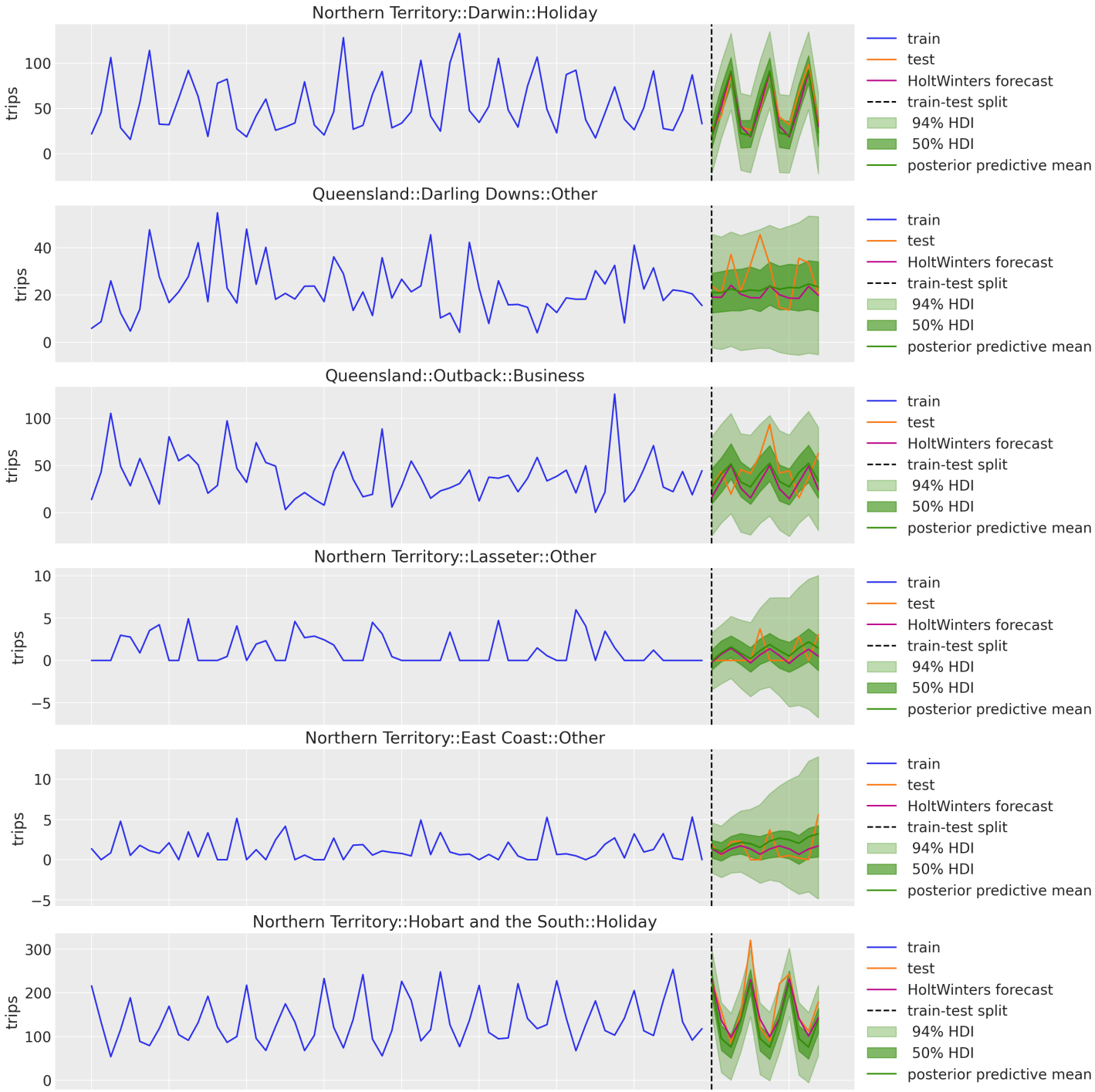
Example: ARMA(1, 1) Model

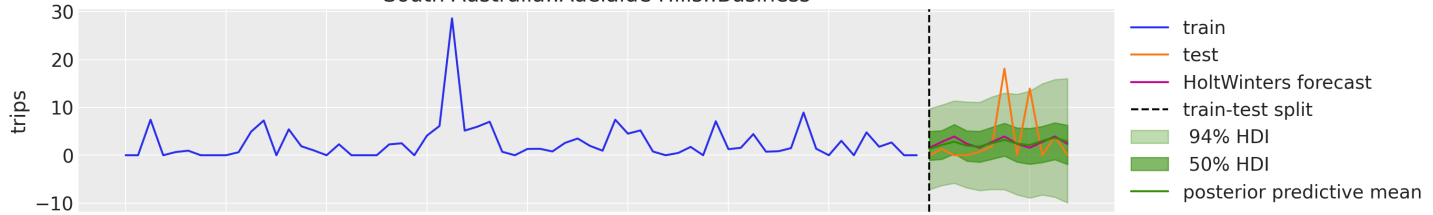
```
    def transition_fn(carry, t):
        y_prev, error_prev = carry
        ar_part = phi * y_prev
        ma_part = theta * error_prev
        pred = mu + ar_part + ma_part
        error = y[t] - pred
        return (y[t], error), error
```



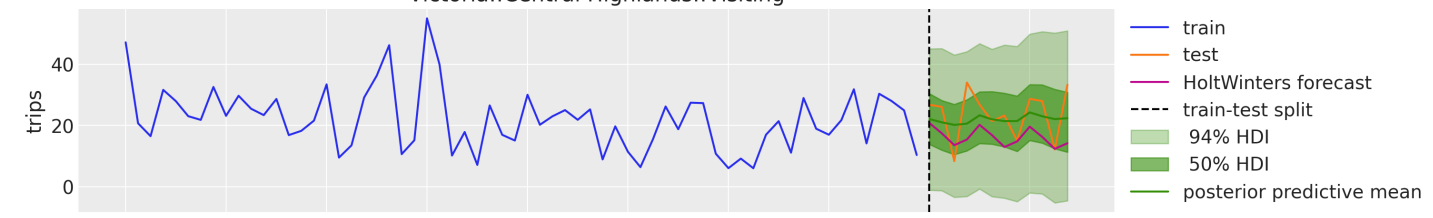
Hierarchical Exponential Smoothing

NUTS Forecast

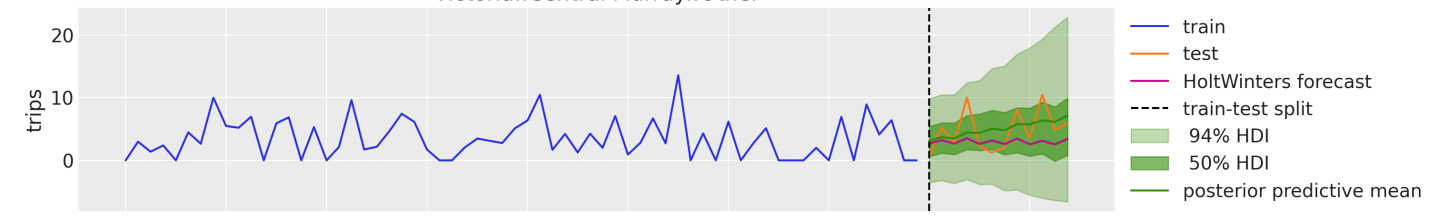




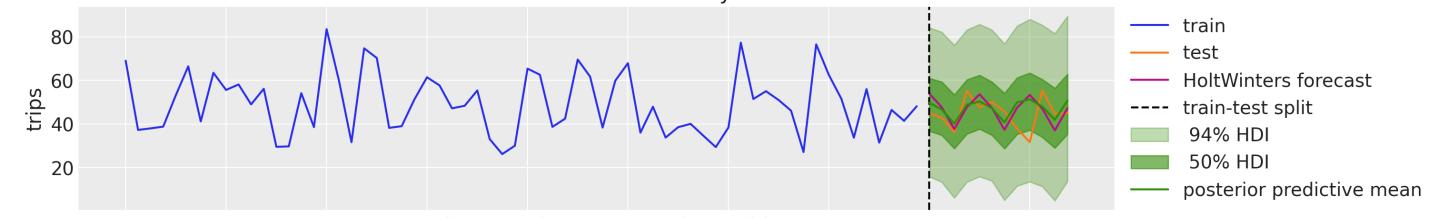
Victoria::Central Highlands::Visiting



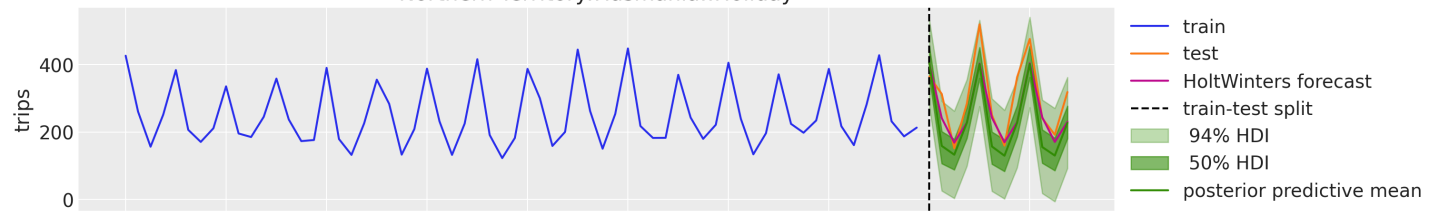
Victoria::Central Murray::Other



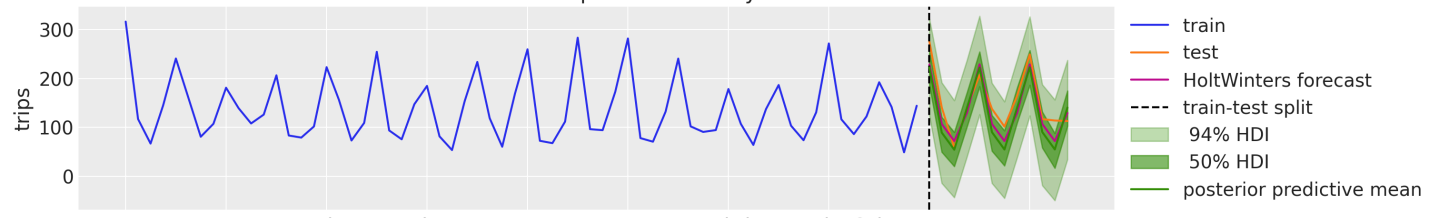
Victoria::Ballarat::Holiday



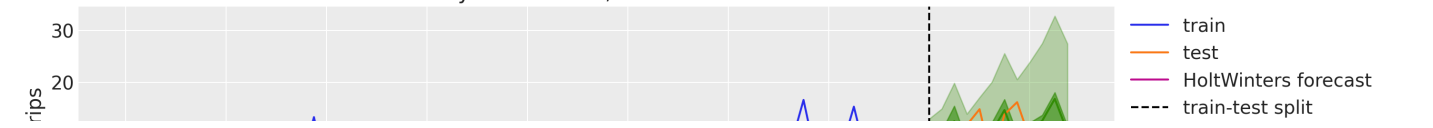
Northern Territory::Tasmania::Holiday



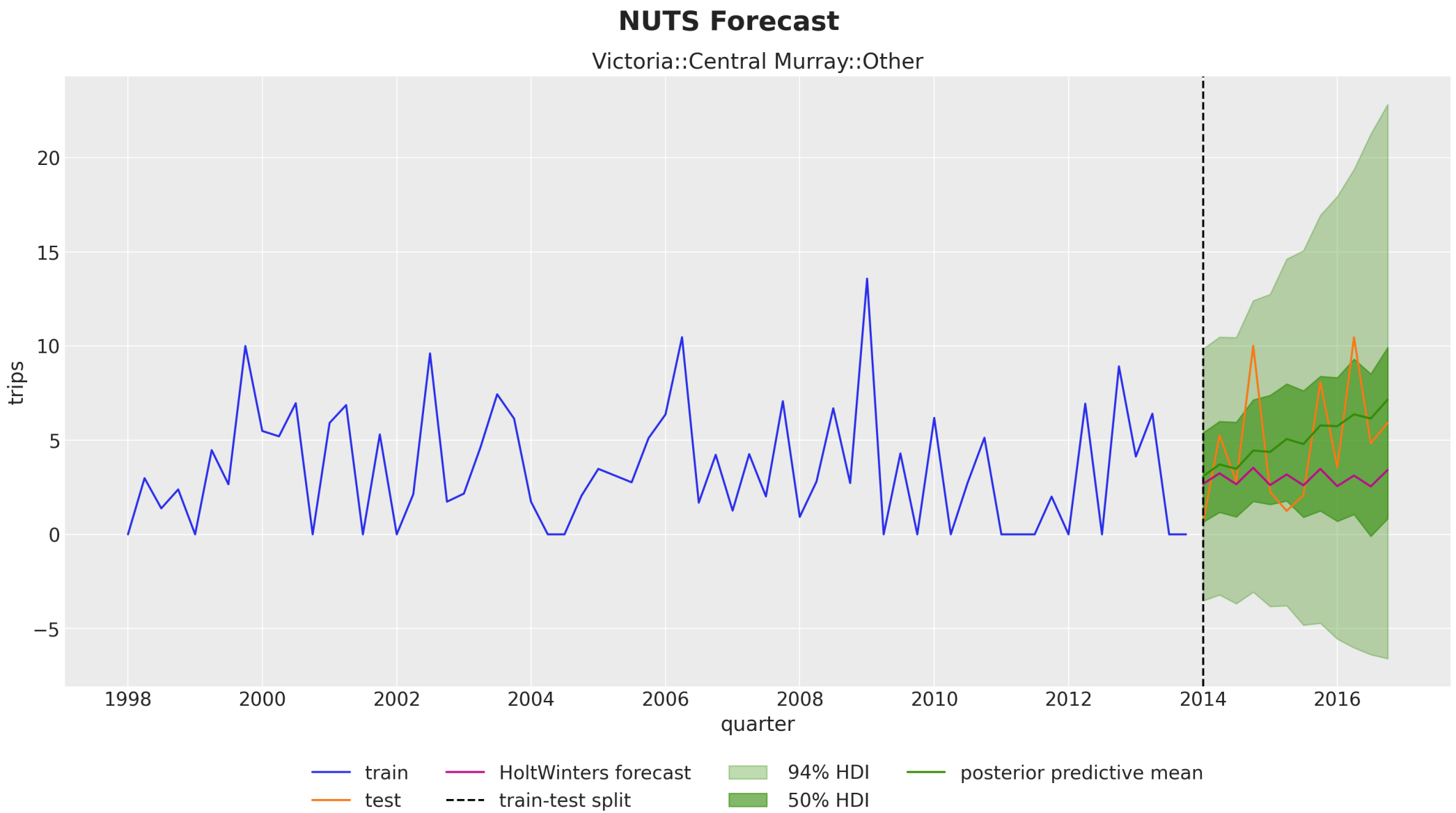
Victoria::Phillip Island::Holiday



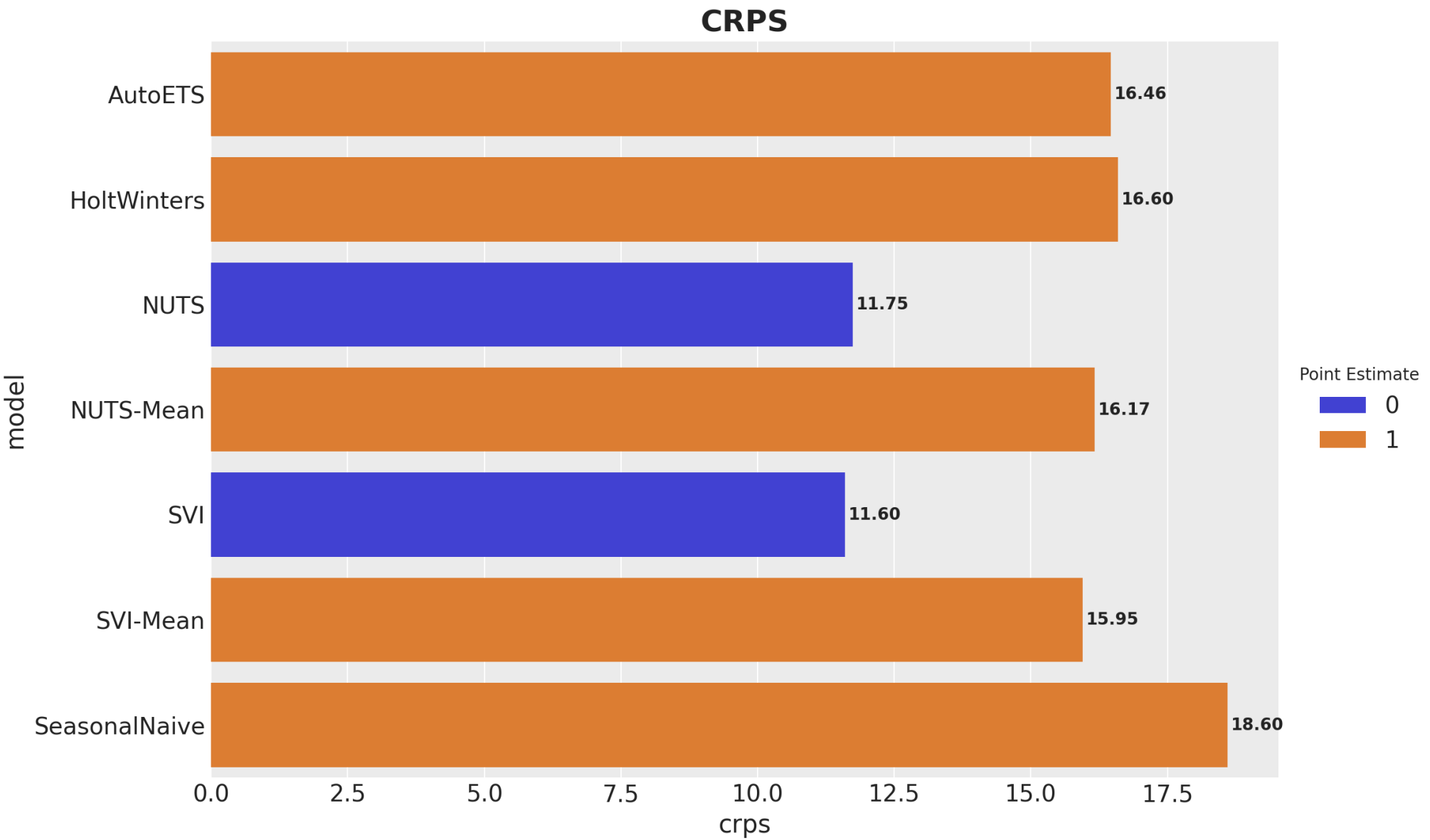
Northern Territory::Launceston, Tamar and the North::Other



Hierarchical Exponential Smoothing

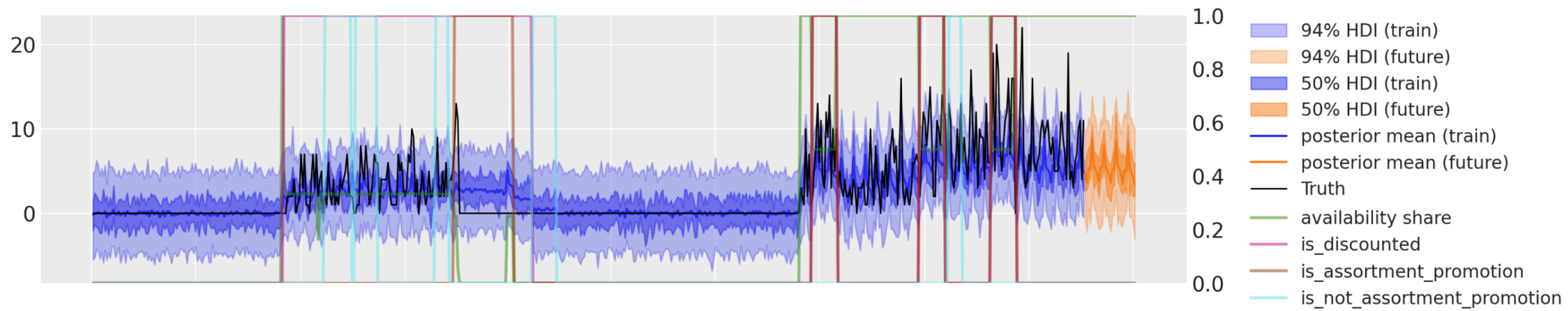


Hierarchical Exponential Smoothing



Baseline Model

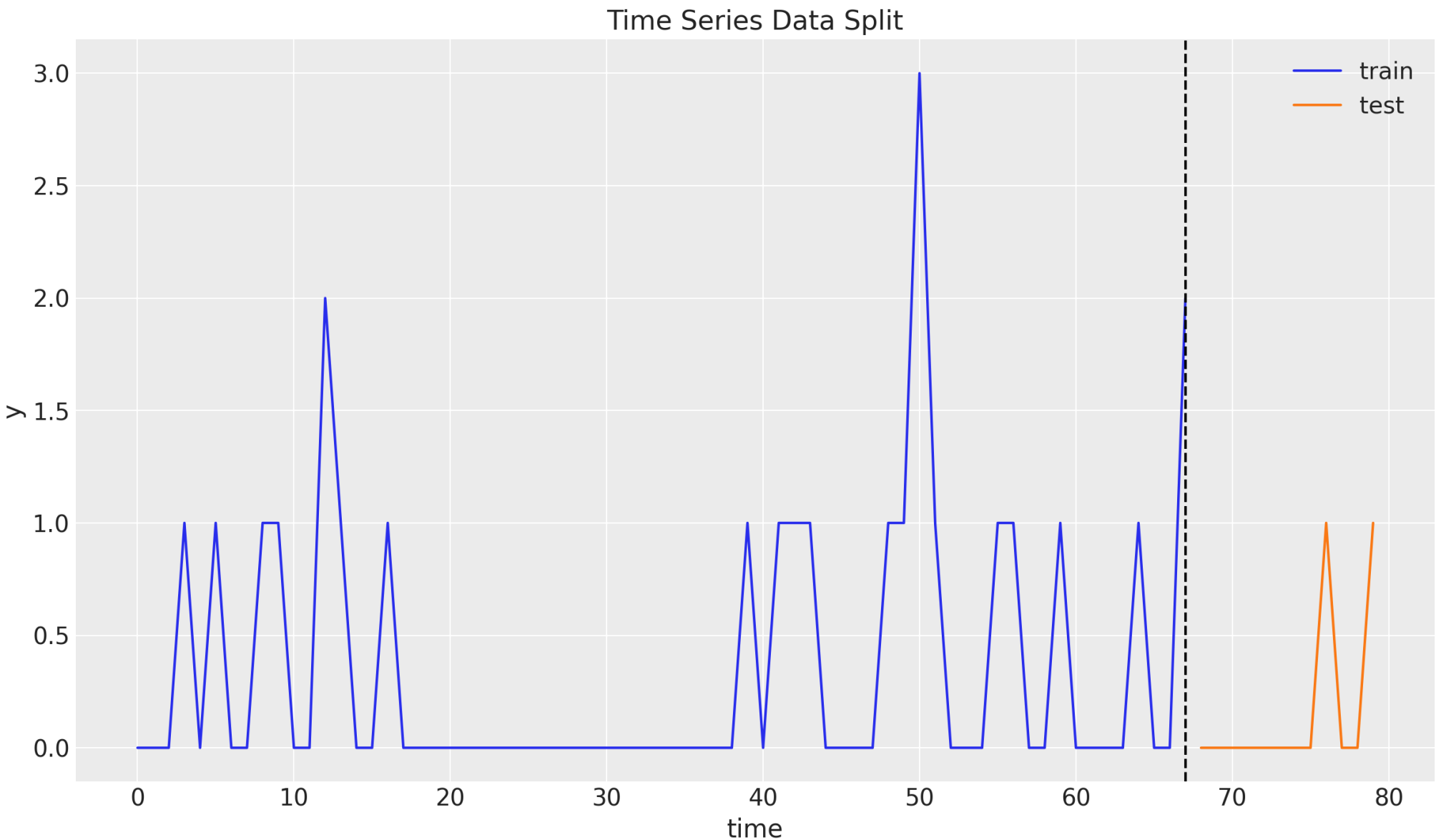
Local Level Model + Seasonality + Covariates



- Local level model to capture the trend component.
- Seasonality using a Fourier modes.
- Add covariates to account for promotions and discounts.
- Global factor to account the availability of the product.

Scalability: 40K time-series can be fitted in less than 10 minutes in a GPU.

Intermittent Time Series



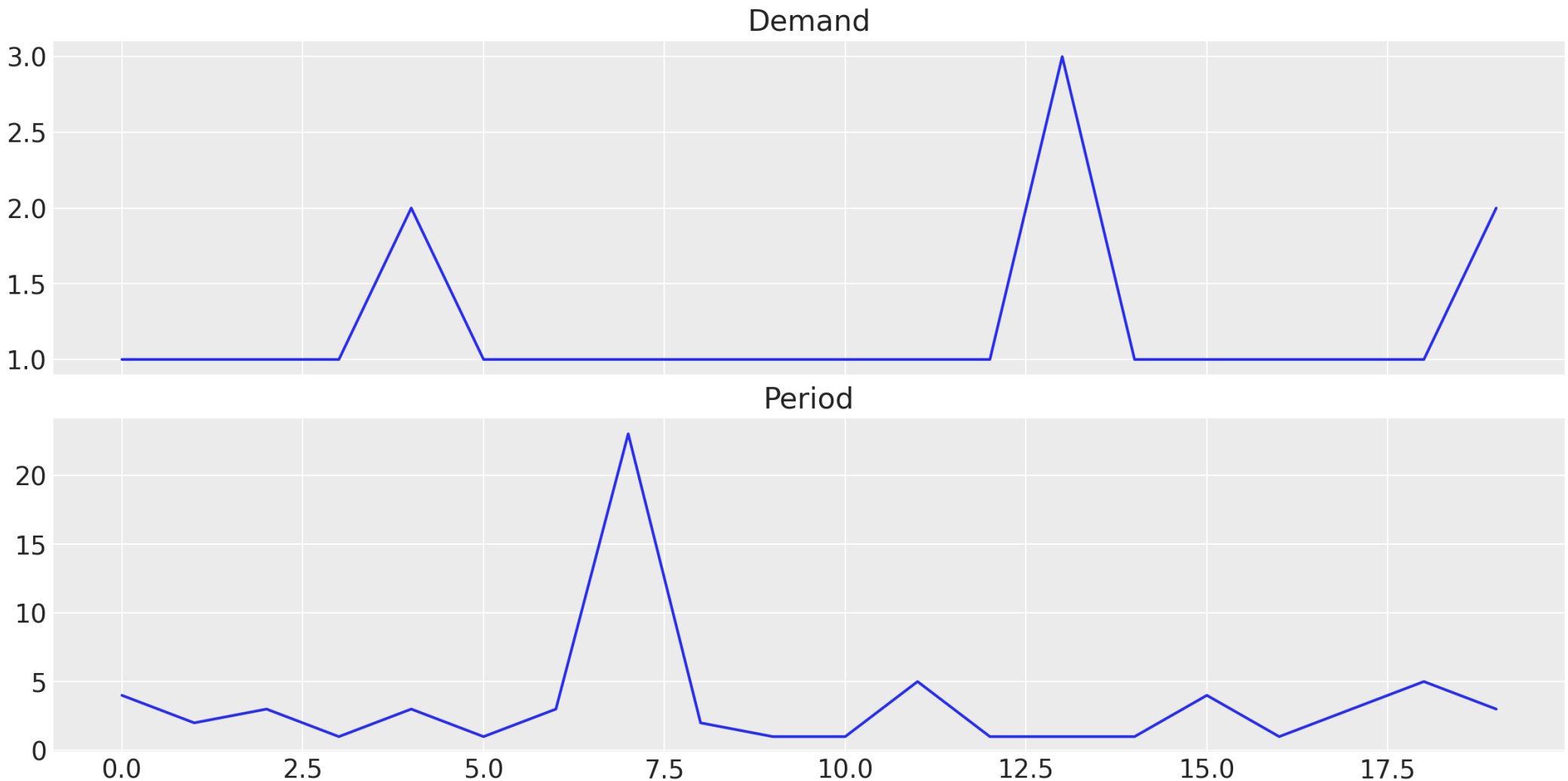
Croston's Method

The method is based on the idea of **separating the demand size z_t and the demand interval p_t**, and then **forecasting them separately** using simple exponential smoothing.

- z_t: keep the non-zero values of y_t.
- p_t: keep the time between non-zero values of y_t.

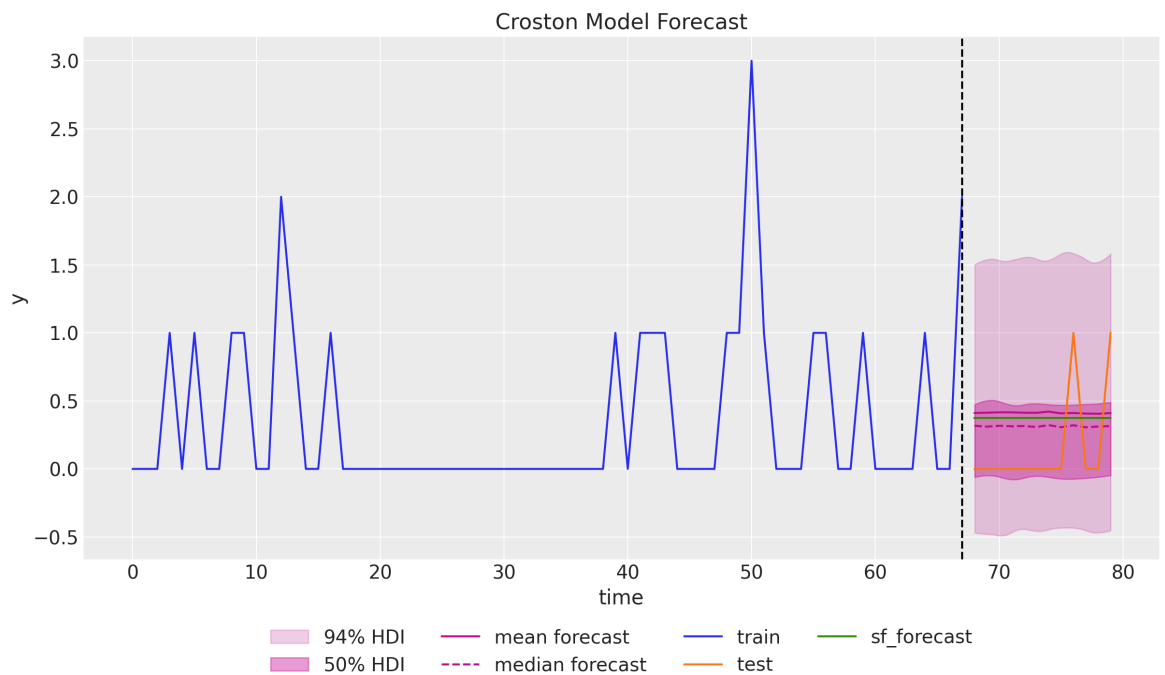
$$\hat{y}_{t+h} = \frac{\hat{z}_{t+h}}{\hat{p}_{t+h}}$$

Croston's Method



Croston's Method

```python
def croston_model(z: ArrayLike, p_inv: ArrayLike, future: int = 0) -> None:
    z_forecast = scope(level_model, "demand")(z, future)
    p_inv_forecast = scope(level_model, "period_inv")(p_inv, future)

    if future > 0:
        numpyro.deterministic("z_forecast", z_forecast)
        numpyro.deterministic("p_inv_forecast", p_inv_forecast)
        numpyro.deterministic("forecast", z_forecast * p_inv_forecast)
```



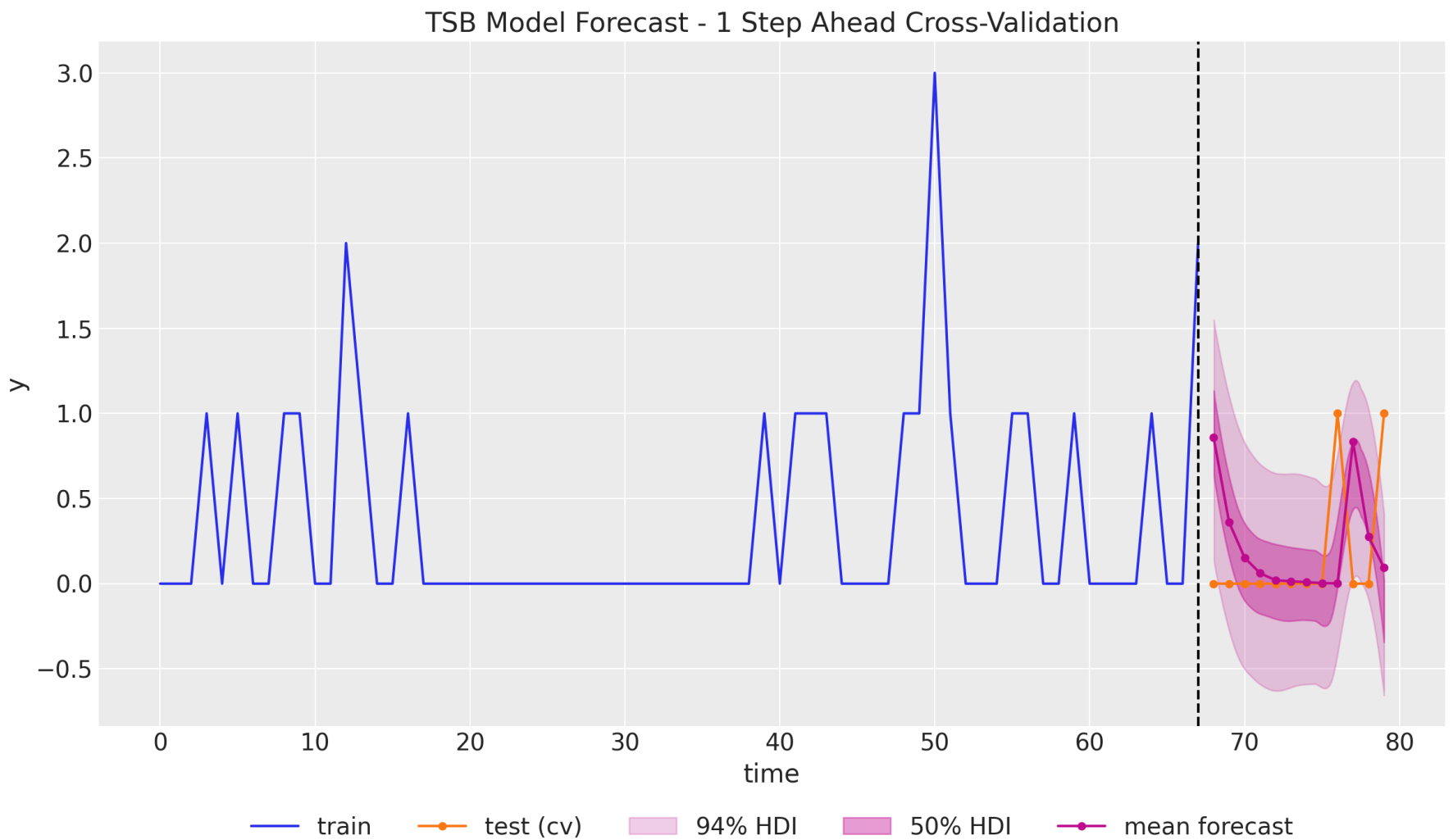
TSB Method

- The TSB method is similar to the Croston's method: constructs two different time series out of the original one and then forecast each of them separately, so that the final forecast is generated by combining the forecasts of the two time series.

- **The main difference between the two methods is that the TSB method uses the demand probability instead of the demand periods.**

TSB Method

1 - Step Ahead Time Slice Cross Validation

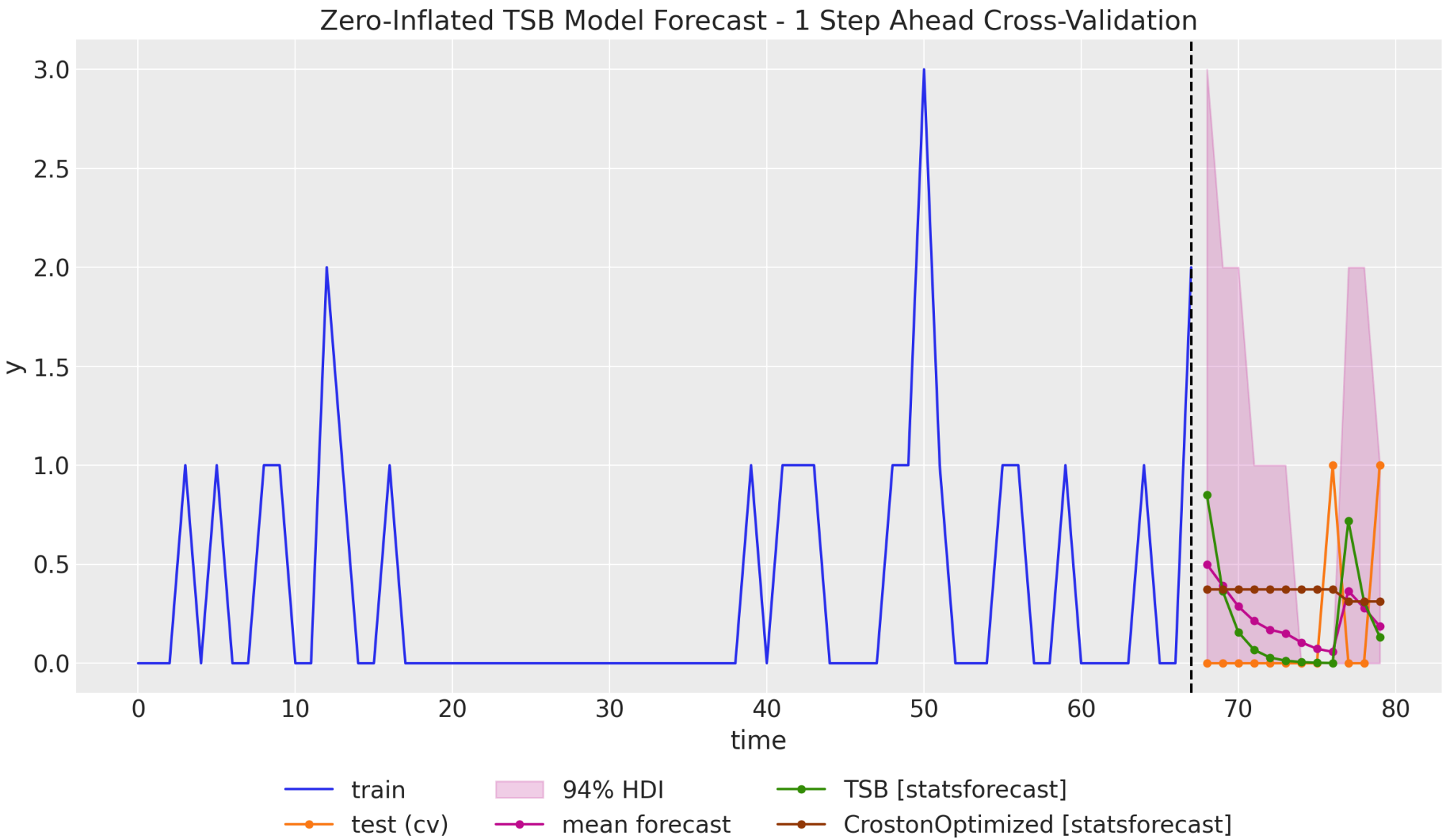


Zero-Inflated TSB Model

🧪 We can modify the TSB model to include zero-inflation by using a Zero-Inflated Negative Binomial Distribution.

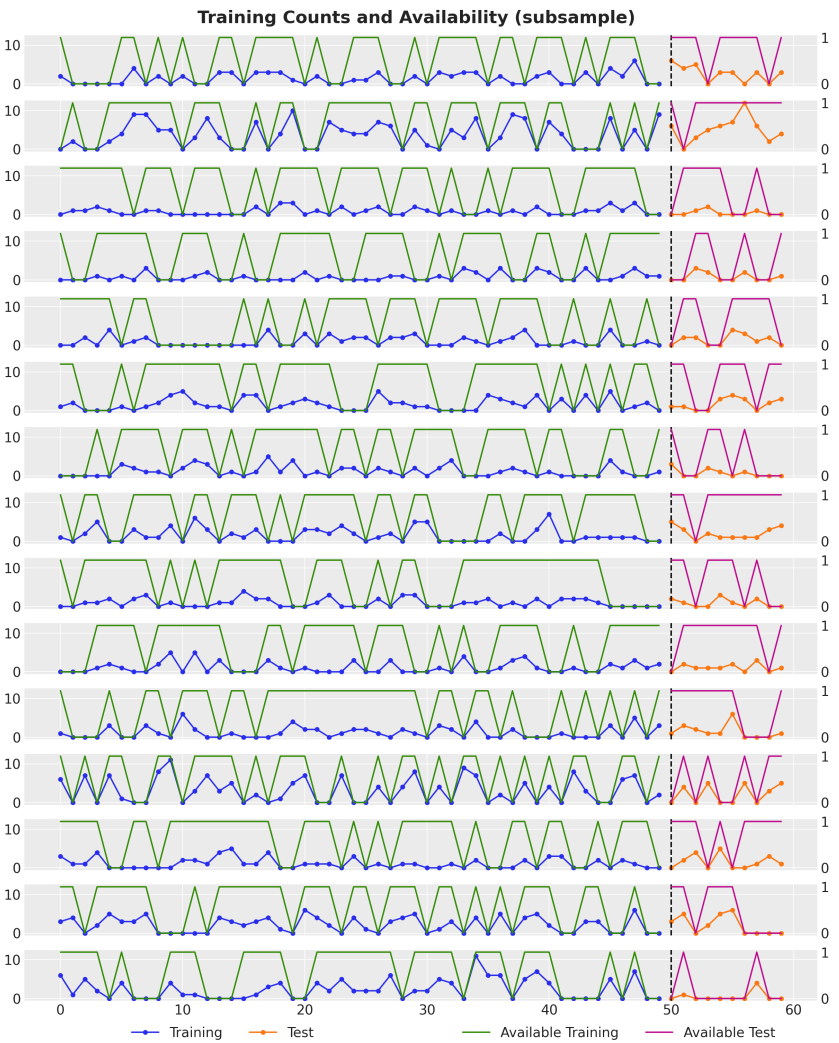
```
    def transition_fn(carry, t):
        z_prev, p_prev = carry

        z_next . . .
        p_next . . .

        mu = z_next
        gate = 1 - p_next
        pred = numpyro.sample(
            "pred",
            dist.ZeroInflatedNegativeBinomial2(
                mean=mu, concentration=concentration, gate=gate
            ),
        )
```

Time-Slice Cross Validation

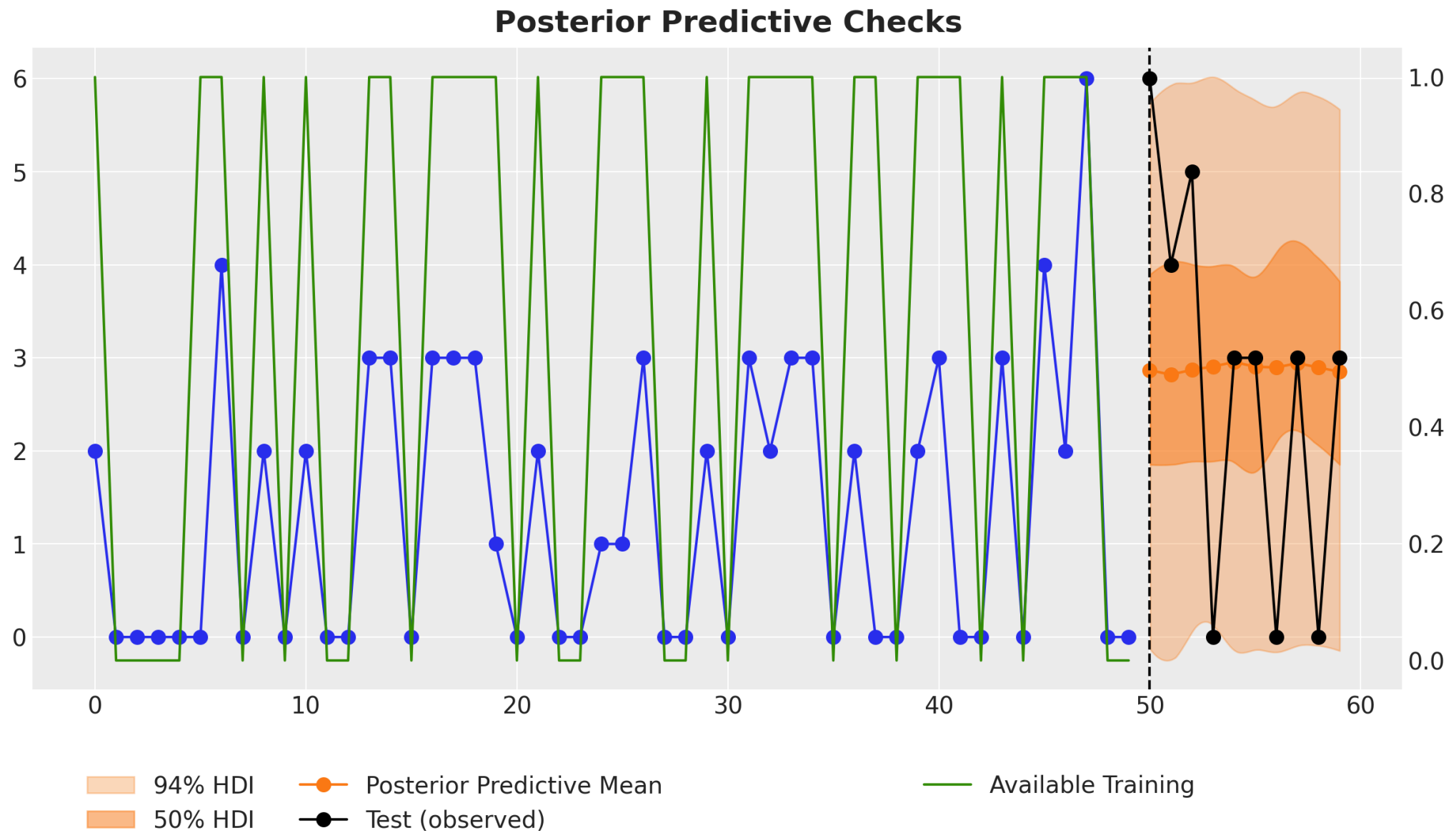


Why Zeros Happen?

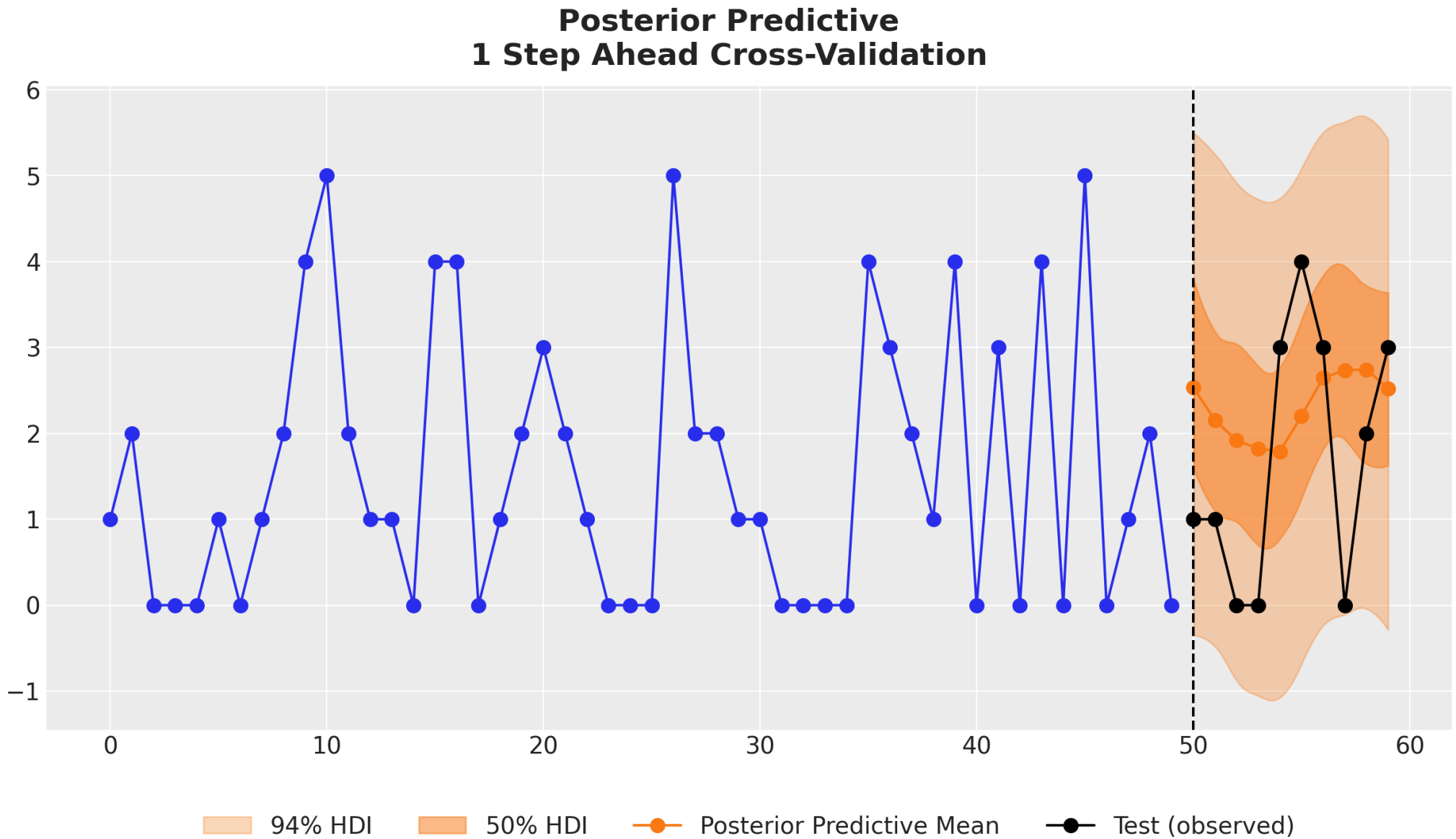
Simulation Study: Availability Constrains



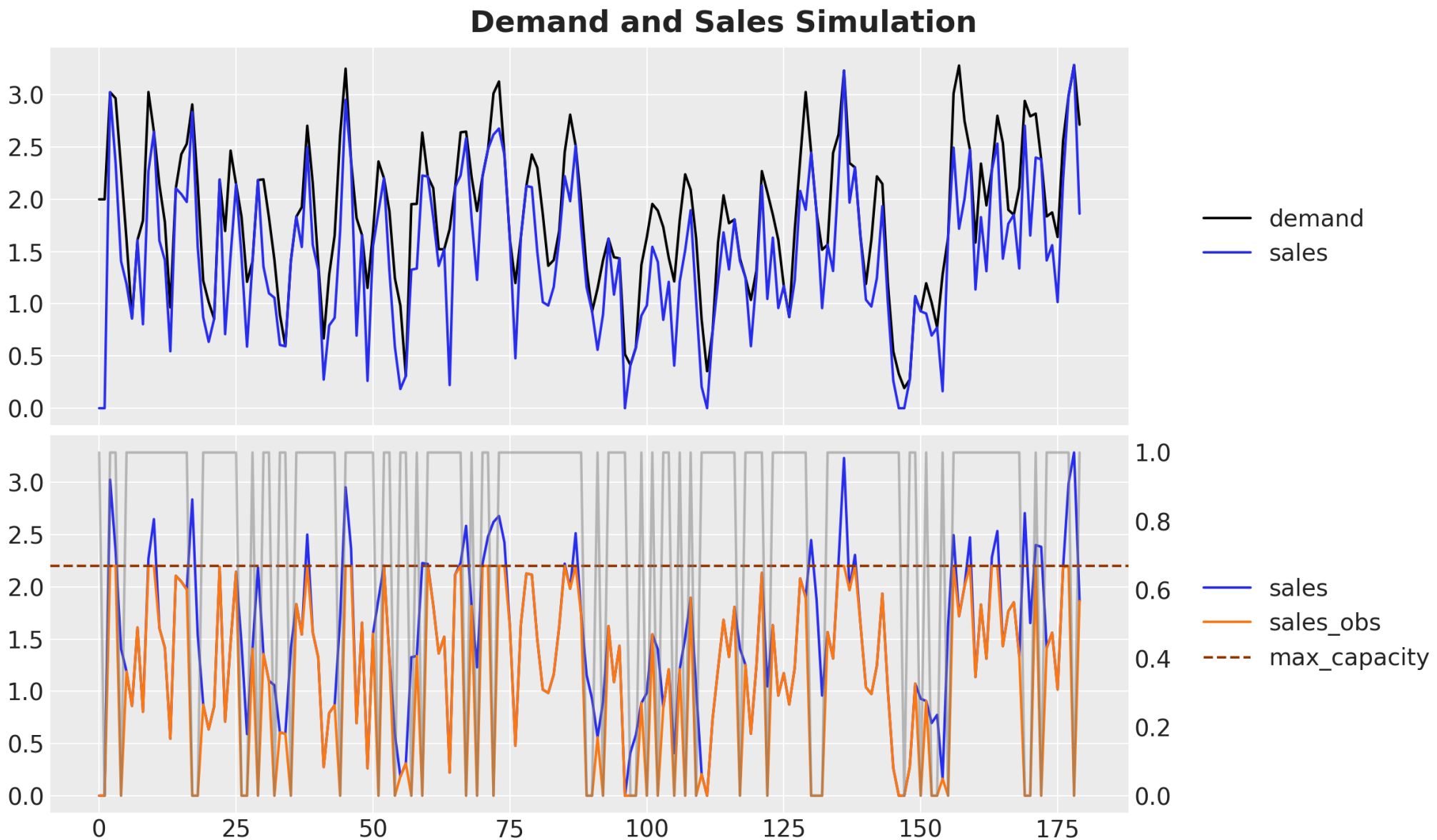
Hacking the TSB Model



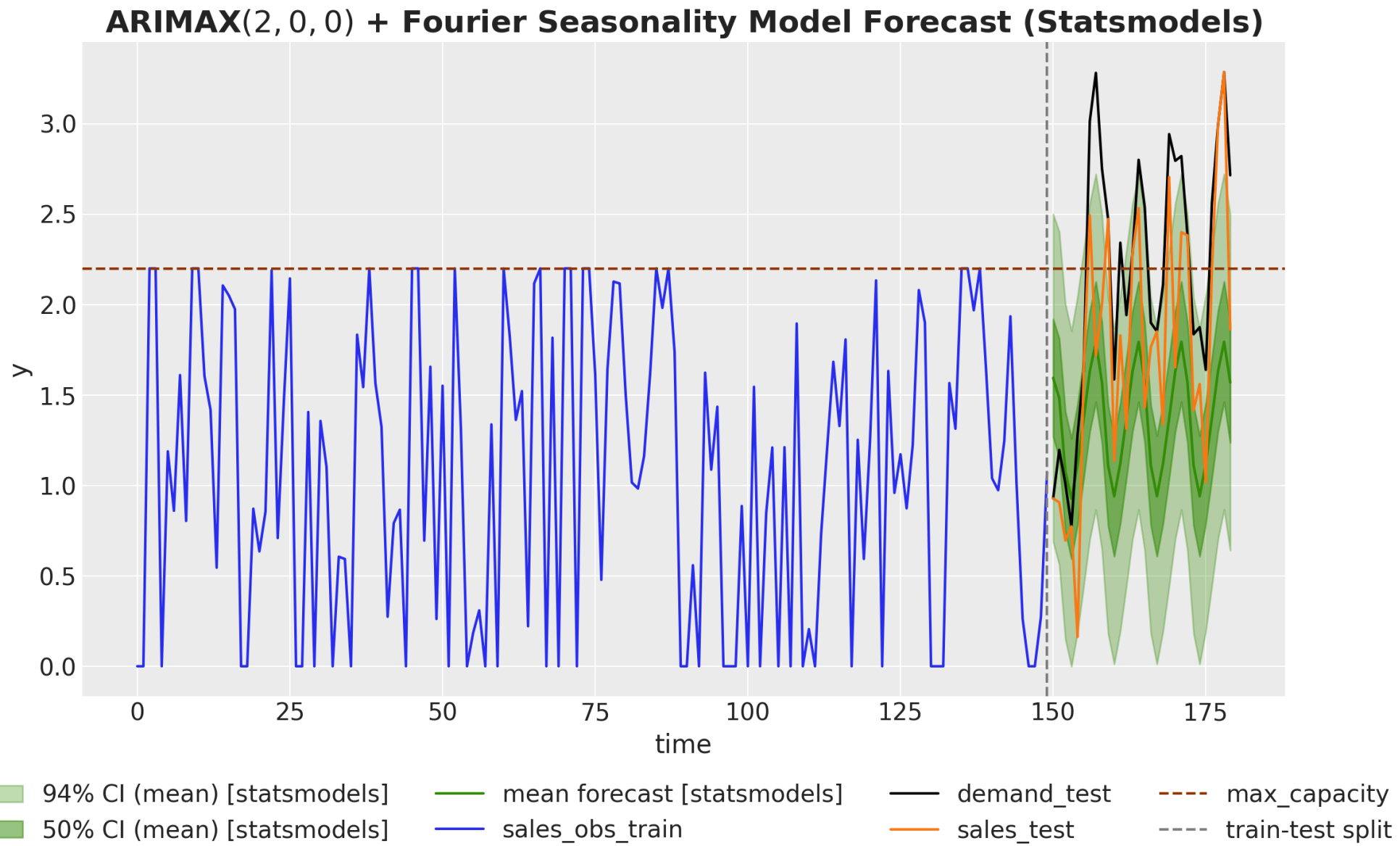
Hacking the TSB Model



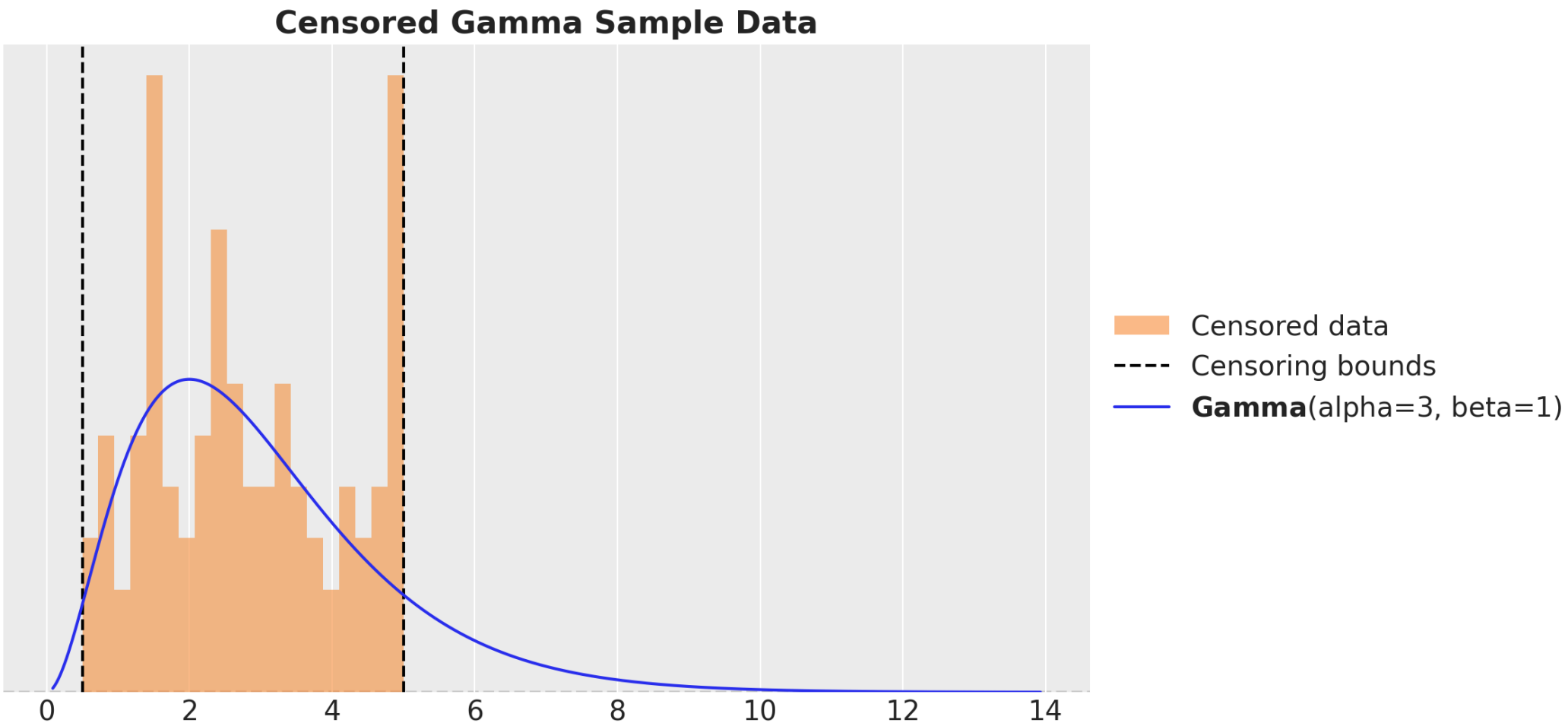
Forecasting Unseen Demand

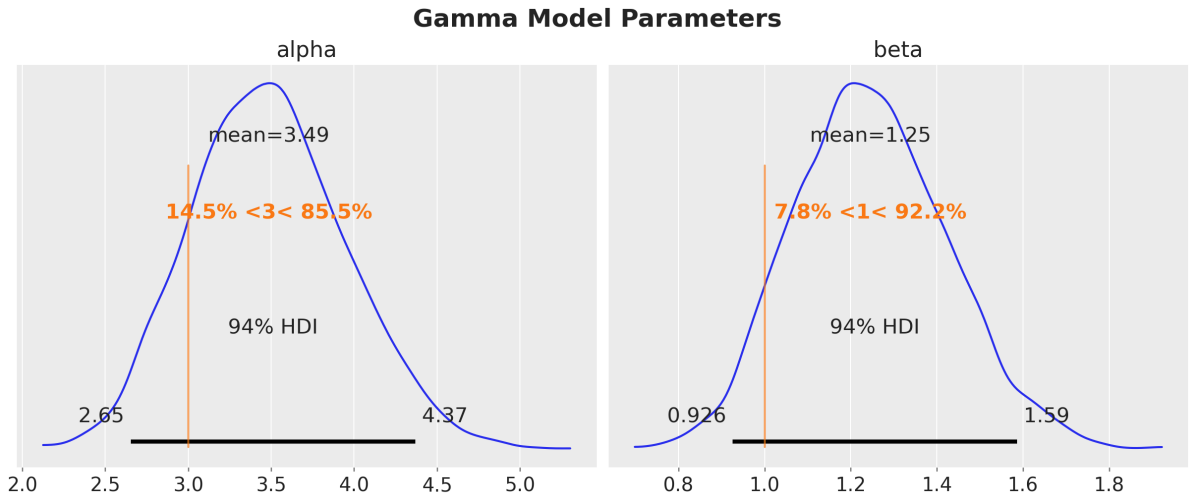


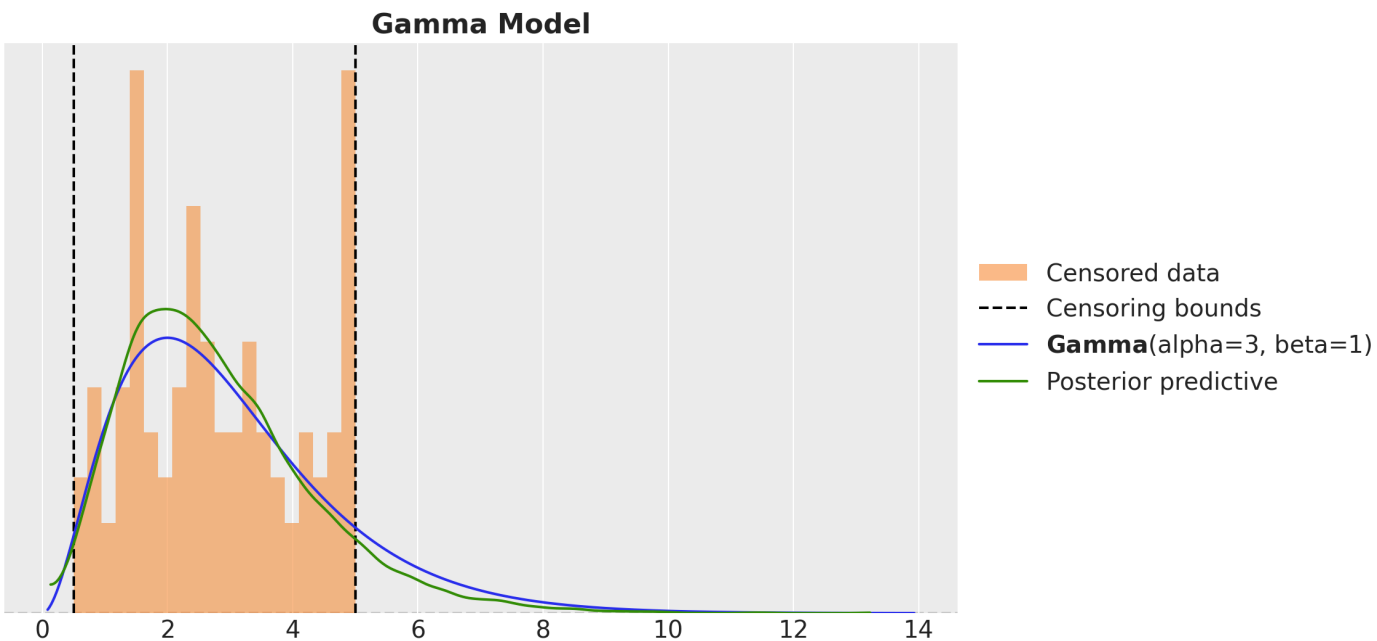
ARIMA Model

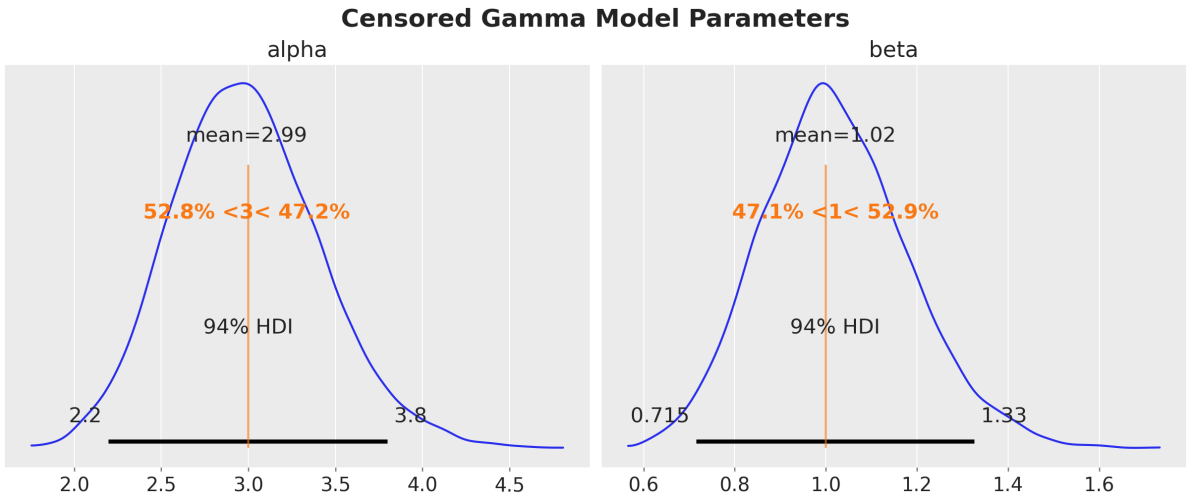

ARIMAX(2, 0, 0) + Fourier Seasonality Model Forecast (Statsmodels)
y
time
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
25
50
75
100
125
150
175
94% CI (mean) [statsmodels]
50% CI (mean) [statsmodels]
mean forecast [statsmodels]
sales_obs_train
demand_test
sales_test
max_capacity
train-test split

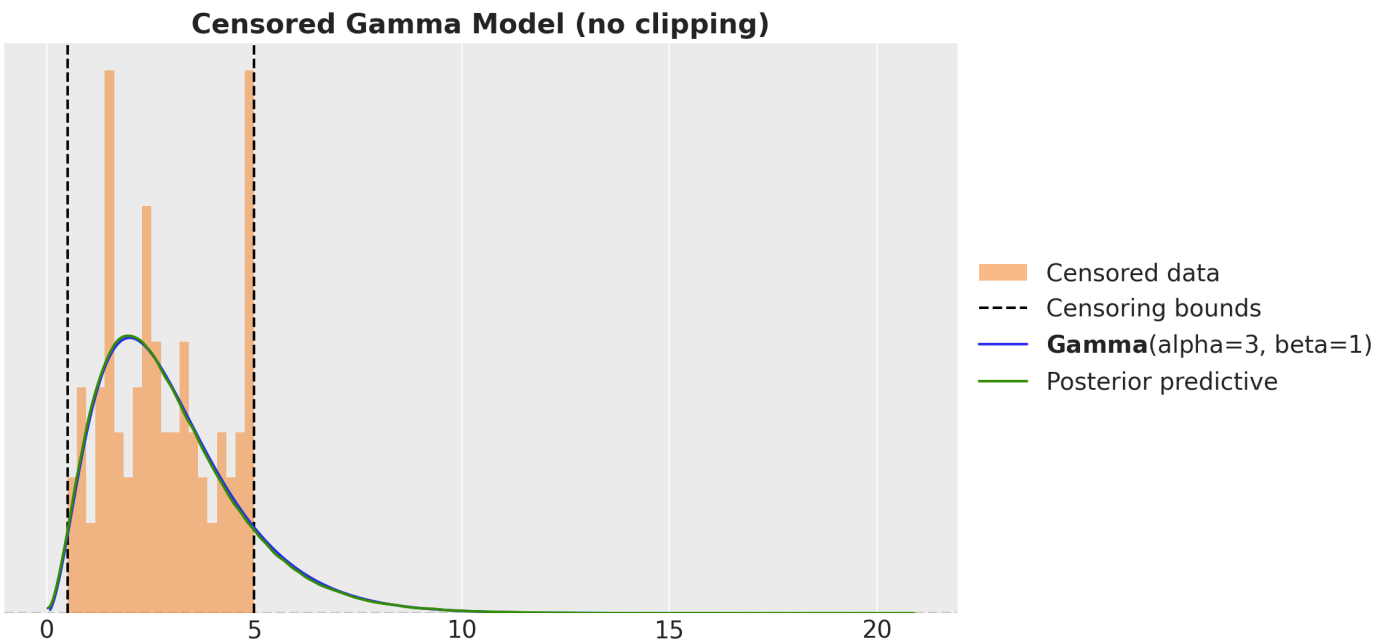
Censored Distributions




Gamma Model Parameters
alpha
mean=3.49
14.5% <3< 85.5%
94% HDI
2.65
4.37
2.0
2.5
3.0
3.5
4.0
4.5
5.0
beta
mean=1.25
7.8% <1< 92.2%
94% HDI
0.926
1.59
0.8
1.0
1.2
1.4
1.6
1.8


Gamma Model
0
2
4
6
8
10
12
14
Censored data
Censoring bounds
Gamma(alpha=3, beta=1)
Posterior predictive


Censored Gamma Model Parameters
alpha
mean=2.99
52.8% <3< 47.2%
94% HDI
2.2
3.8
2.0
2.5
3.0
3.5
4.0
4.5
beta
mean=1.02
47.1% <1< 52.9%
94% HDI
0.715
1.33
0.6
0.8
1.0
1.2
1.4
1.6


Censored Gamma Model (no clipping)
0
5
10
15
20
Censored data
Censoring bounds
Gamma(alpha=3, beta=1)
Posterior predictive

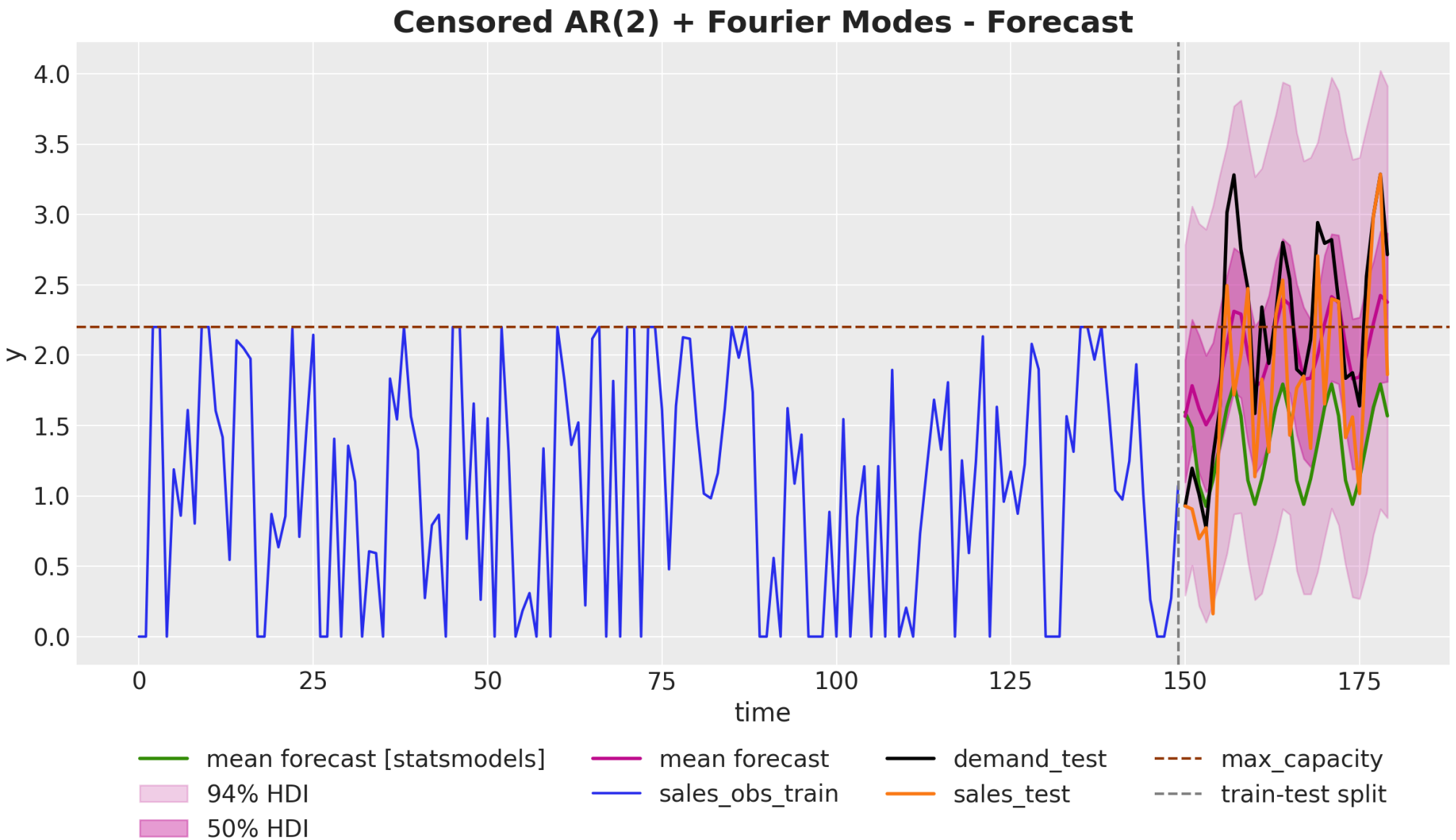
Censored Likelihood

```python
def censored_normal(loc, scale, y, censored):
    distribution = dist.Normal(loc=loc, scale=scale)
    ccdf = 1 - distribution.cdf(y)
    numpyro.sample(
        "censored_label",
        dist.Bernoulli(probs=ccdf).mask(censored == 1),
        obs=censored
    )
    return numpyro.sample("pred", distribution.mask(censored != 1))
```

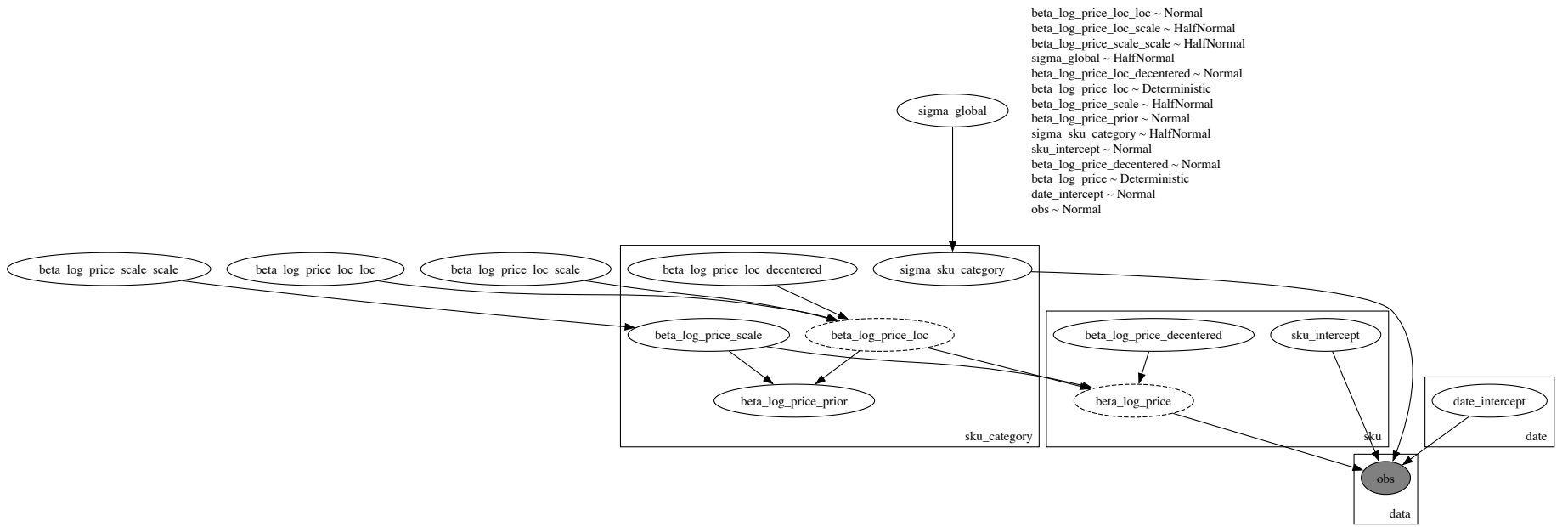
Change likelihood distribution in a time-series model:

```python
    ## Transition function for AR(2)
    def transition_fn(carry, t):
        y_prev_1, y_prev_2 = carry
        ar_part = phi_1 * y_prev_1 + phi_2 * y_prev_2
        pred_mean = mu + ar_part + seasonal[t]
        # Censored likelihood
        pred = censored_normal(pred_mean, sigma, y[t], censored[t])
        return (pred, y_prev_1), pred
```

Censored Time Series Forecast 💡



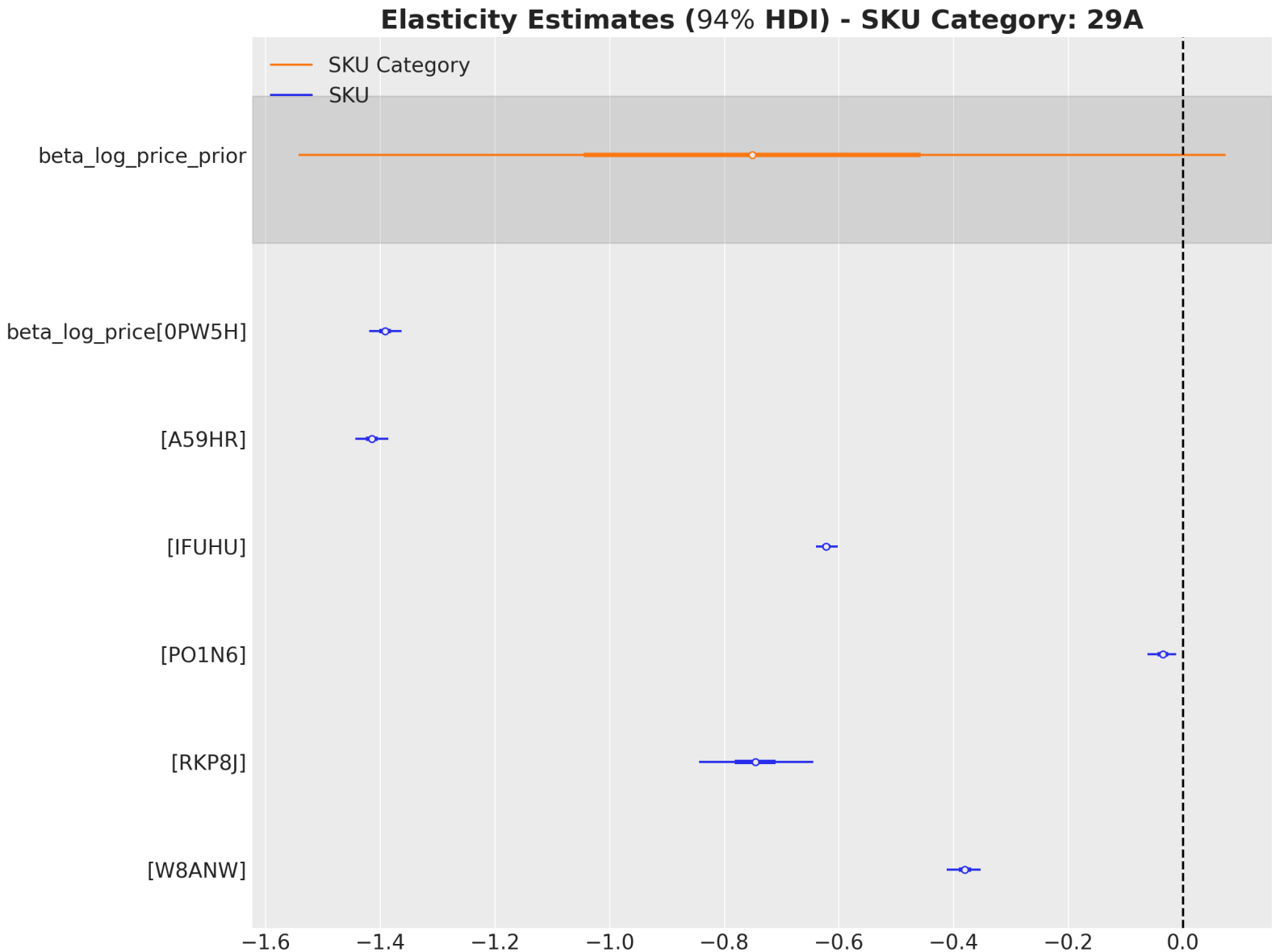
Hierarchical Pricing Elasticity Models



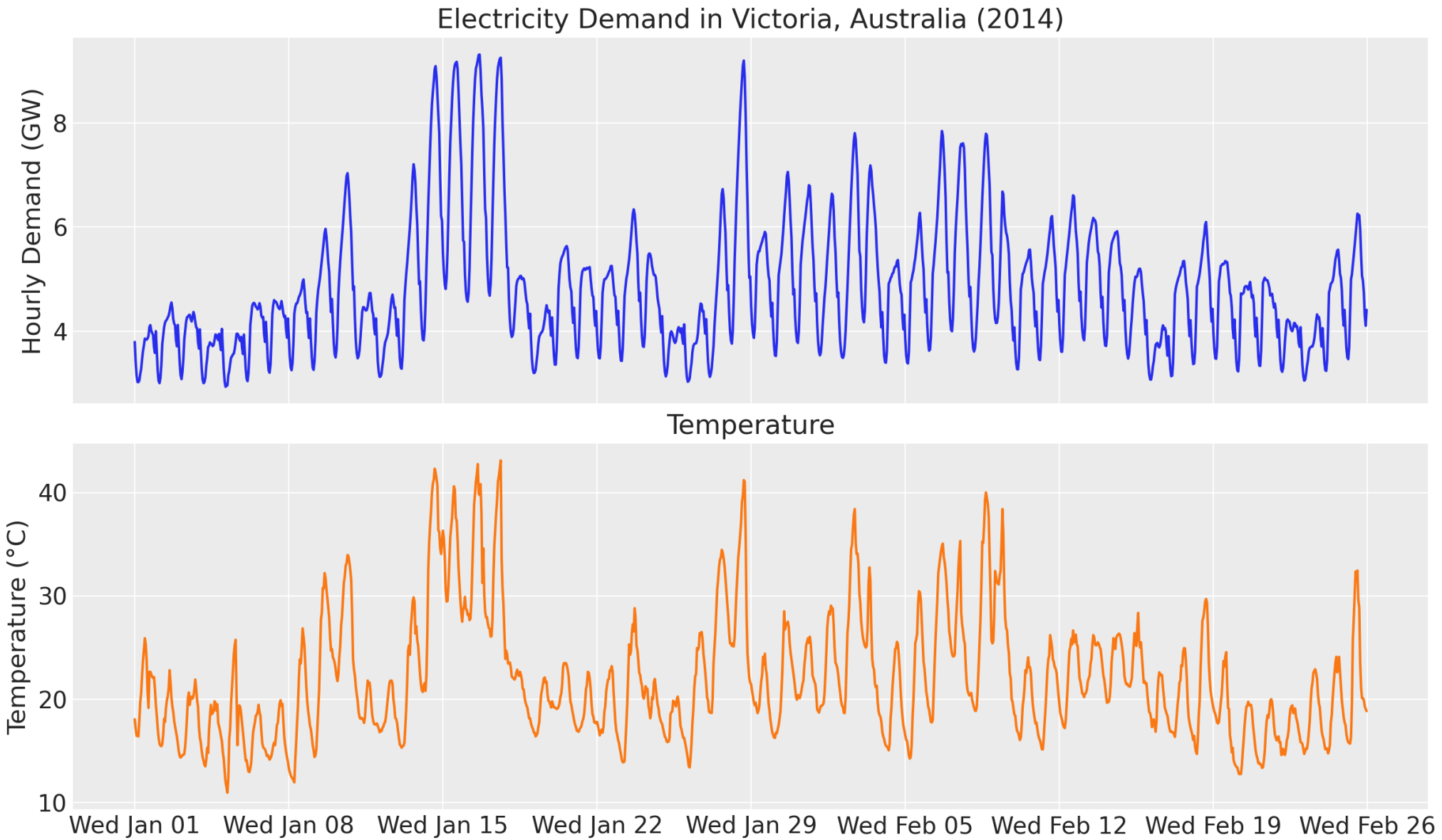
Idea 🤓

Use a hierarchical structure to regularize the demand elasticity parameters.

Hierarchical Pricing Elasticity Models

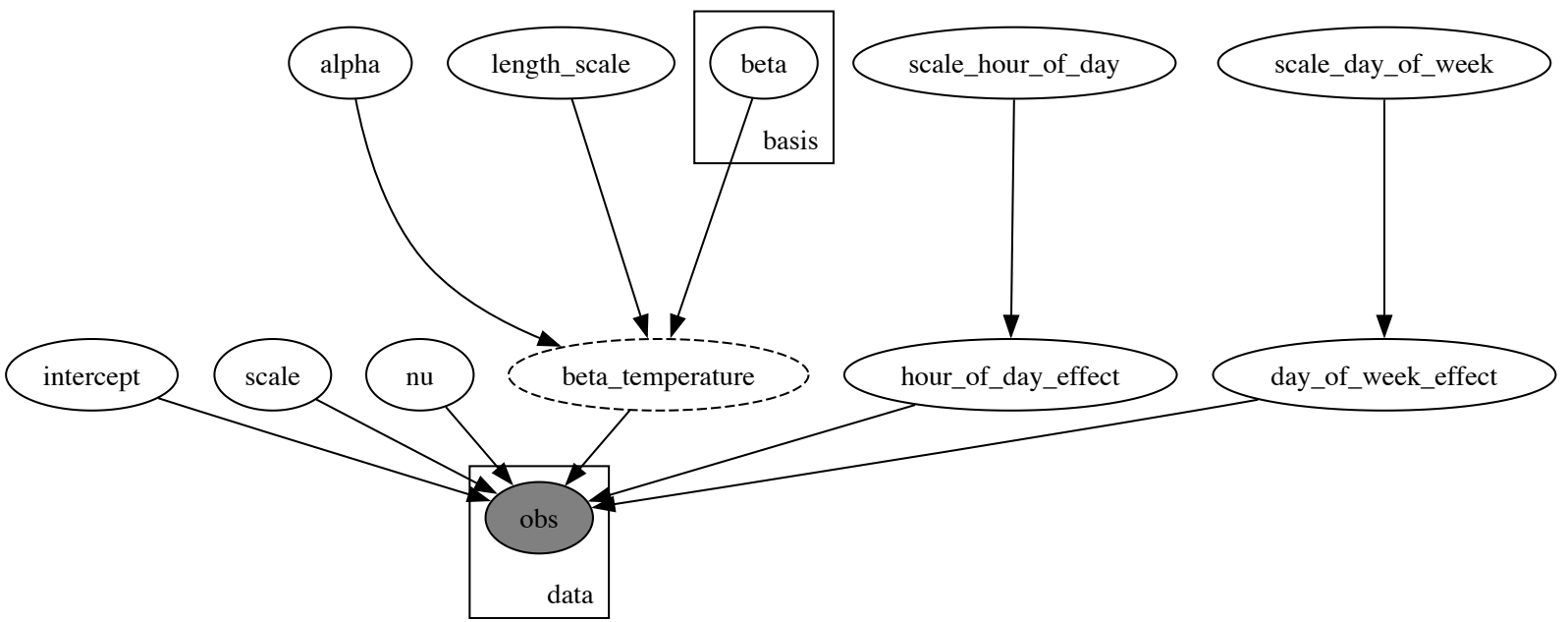


Dynamic Time-Series Model



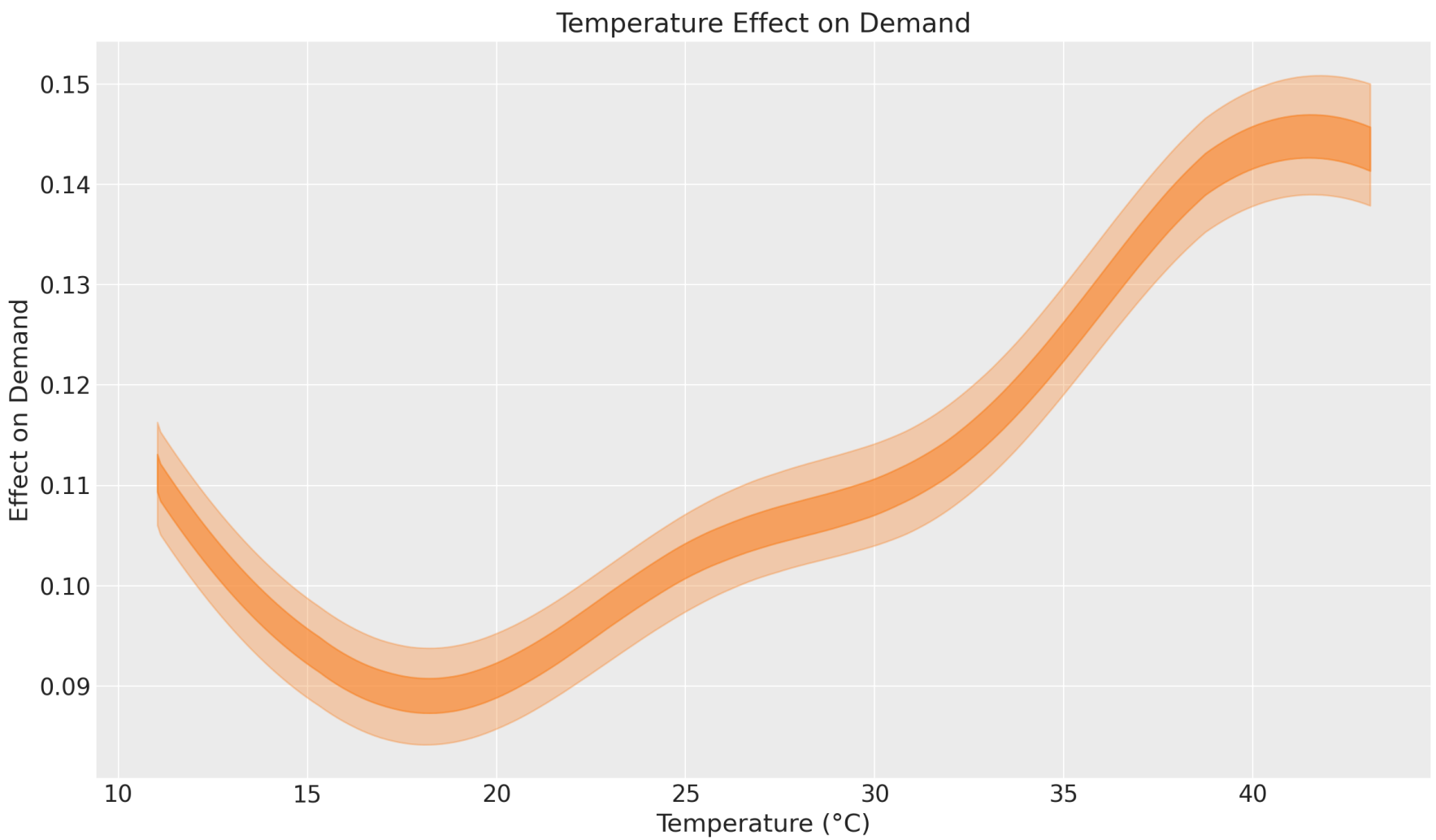
Dynamic Coefficients

Hilbert Space Gaussian Processes for Dynamic Coefficients



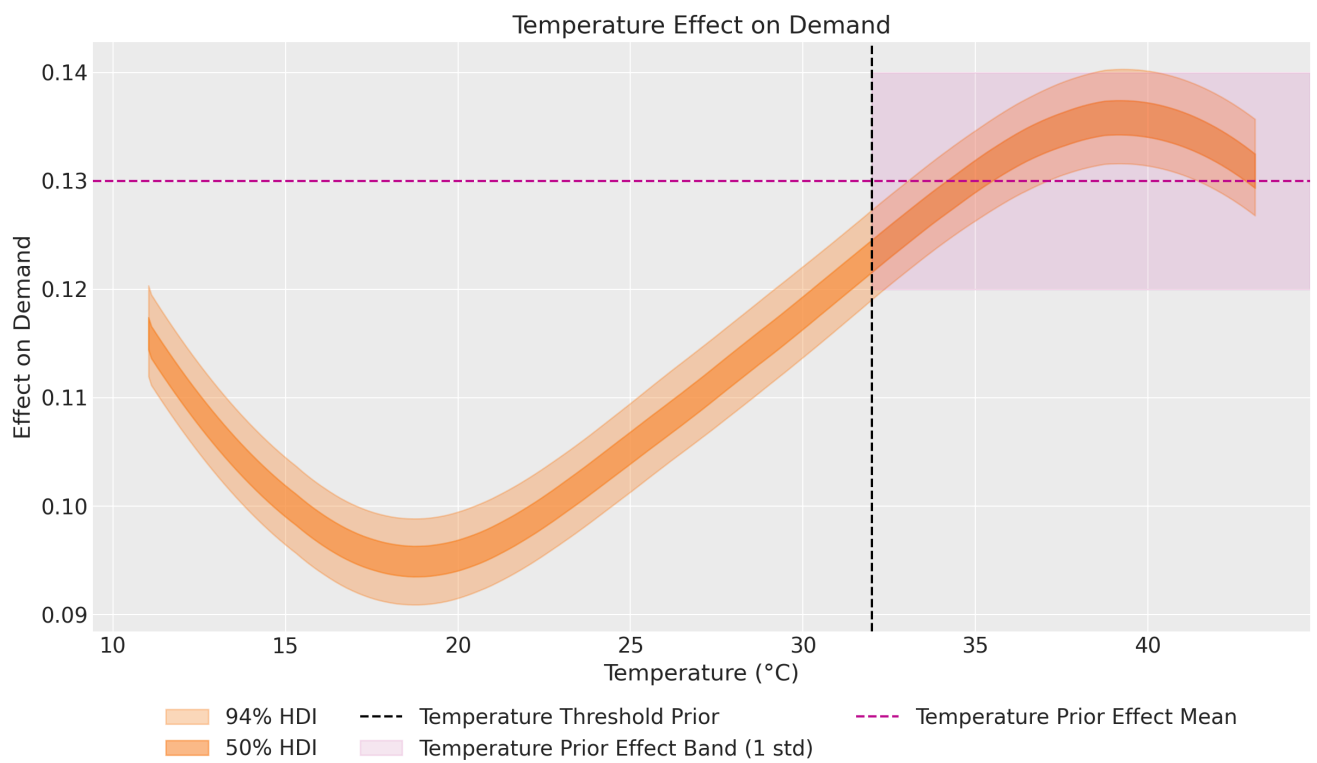
intercept ~ Normal
alpha ~ InverseGamma
length_scale ~ InverseGamma
scale ~ HalfNormal
nu ~ Gamma
beta ~ Normal
beta_temperature ~ Deterministic
scale_hour_of_day ~ HalfNormal
hour_of_day_effect ~ ZeroSumNormal
scale_day_of_week ~ HalfNormal
day_of_week_effect ~ ZeroSumNormal
obs ~ StudentT

Inferring Effect of Temperature on Electricity Demand



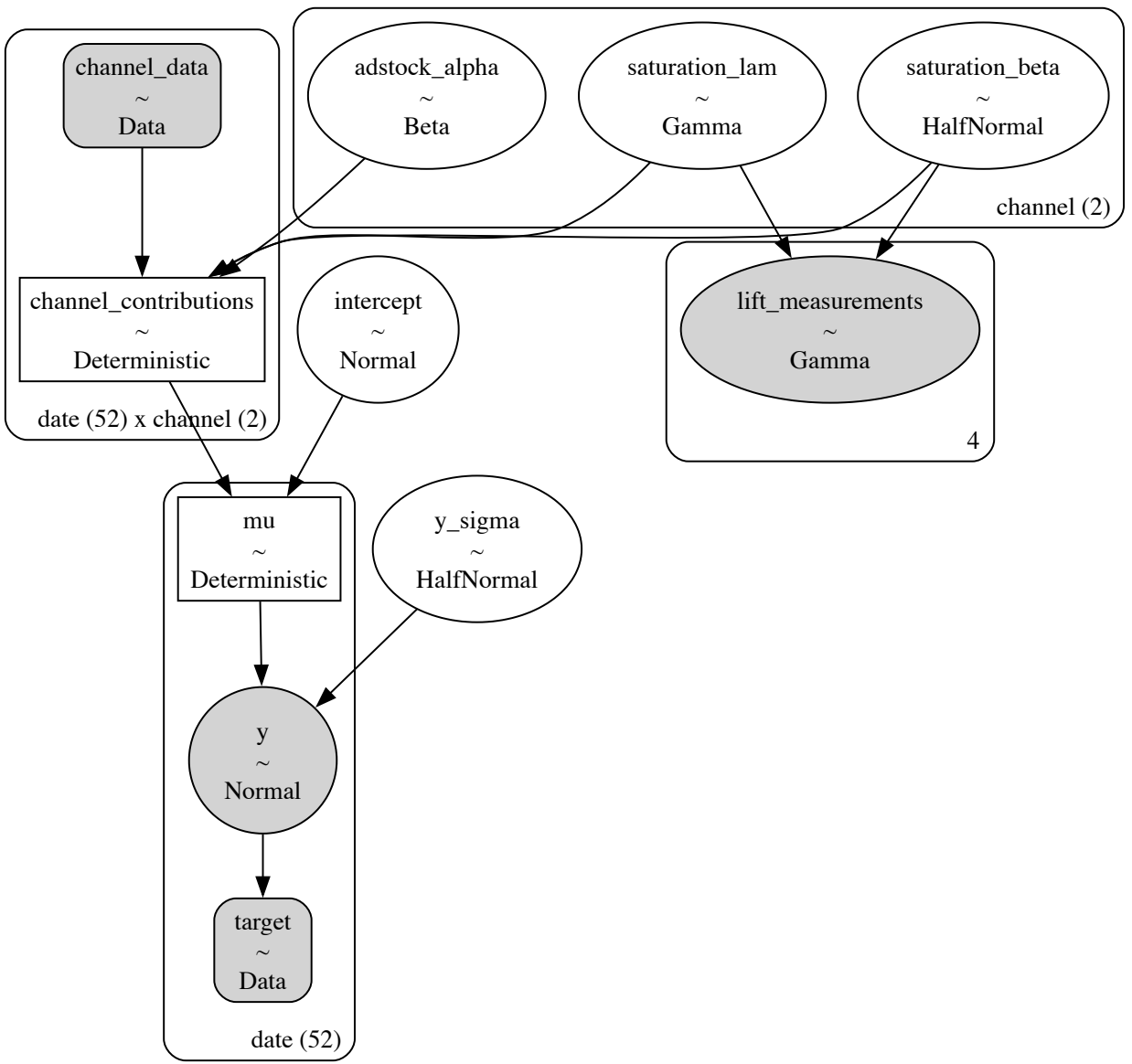
Calibrating a Demand Model 🧪

> Let us assume that we know from domain knowledge that the effect of temperature on demand over 32°C is somehow stable at around a value of 0.13.



MMM Calibration with Lift Tests

MMM Calibration with Lift Tests



References 📚

NumPyro Examples

- Introduction to Stochastic Variational Inference with NumPyro ⭐
- Bayesian Censoring Data Modeling
- Croston's Method for Intermittent Time Series Forecasting in NumPyro
- Demand Forecasting with Censored Likelihood
- Electricity Demand Forecast: Dynamic Time-Series Model with Prior Calibration
- From Pyro to NumPyro: Forecasting Hierarchical Models - Part I
- From Pyro to NumPyro: Forecasting Hierarchical Models - Part II
- Hacking the TSB Model for Intermediate Time Series to Accommodate for Availability Constraints
- Hierarchical Exponential Smoothing Model
- Hierarchical Pricing Elasticity Models
- Notes on Exponential Smoothing with NumPyro

References 

Packages:

- Nixtla
- Prophetverse
- Pyro - Forecasting
- Statsmodels - Time Series Models
- TimeSeers
- GluonTS
- Zalando: PyTorchTS

References 📚

Other Blogposts

- Finally! Bayesian Hierarchical Modelling at Scale
- Modeling Anything With First Principles: Demand under extreme stockouts
- PyMC-Marketing: Lift Test Calibration
- PyMC Labs: Unobserved Confounders, ROAS and Lift Tests in Media Mix Models
- PyMC Labs: Probabilistic Time Series Analysis - Opportunities and Applications

Books

- Forecasting: Principles and Practice, the Pythonic Way
- Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)

Papers

Thank you!

juanitorduz.github.io