Recalibration of Gaussian Neural Networks regression models:

the *recalibratiNN* package

Carolina Musso
Instituto de Pesquisa e Estatística do DF, Brazil

2024-07-11

A proper introduction

- Disclaimer: Me, the package and everything else.
  - Academic: Biological invasions, Fire Ecology, Ecotoxicology ...
  - Public Sector: Epidemiology, Sampling design and inference...
  - Free time: Bachelor degree in Statistics, Computational statistics, Bayesian methods, Neural Networks and Recalibration.
- R!
- Basically, I really wanted to develop a package.

Introduction: Neural Networks nowadays

- It should be able to quantify its uncertainty.
- NN can be constructed to produce probabilistic results:
  - Optimized by the log-likelihood.
  - Like any model, it can be miscalibrated.
    - A 95% CI should contain 95% of the true output.
    - $\mathbb{P}(Y \leq \hat{F}_Y^{-1}(p)) = p, \forall\, p \in [0, 1]$

> **Note**
>
> If optimized by MSE, I will be assuming a normal distribution.

Observing miscalibration

Consider a synthetic data set $(x_i, y_i), i \in (1, \ldots, n)$ generated by an heteroscedastic non-linear model:

$$x_i \sim Uniform(1, 10)$$

$$y_i|x_i \sim Normal(\mu = f_1(x_i), \sigma = f_2(x_i))$$
$$f_1(x) = 5x^2 + 10 \; ; \; f_2(x) = 30x$$

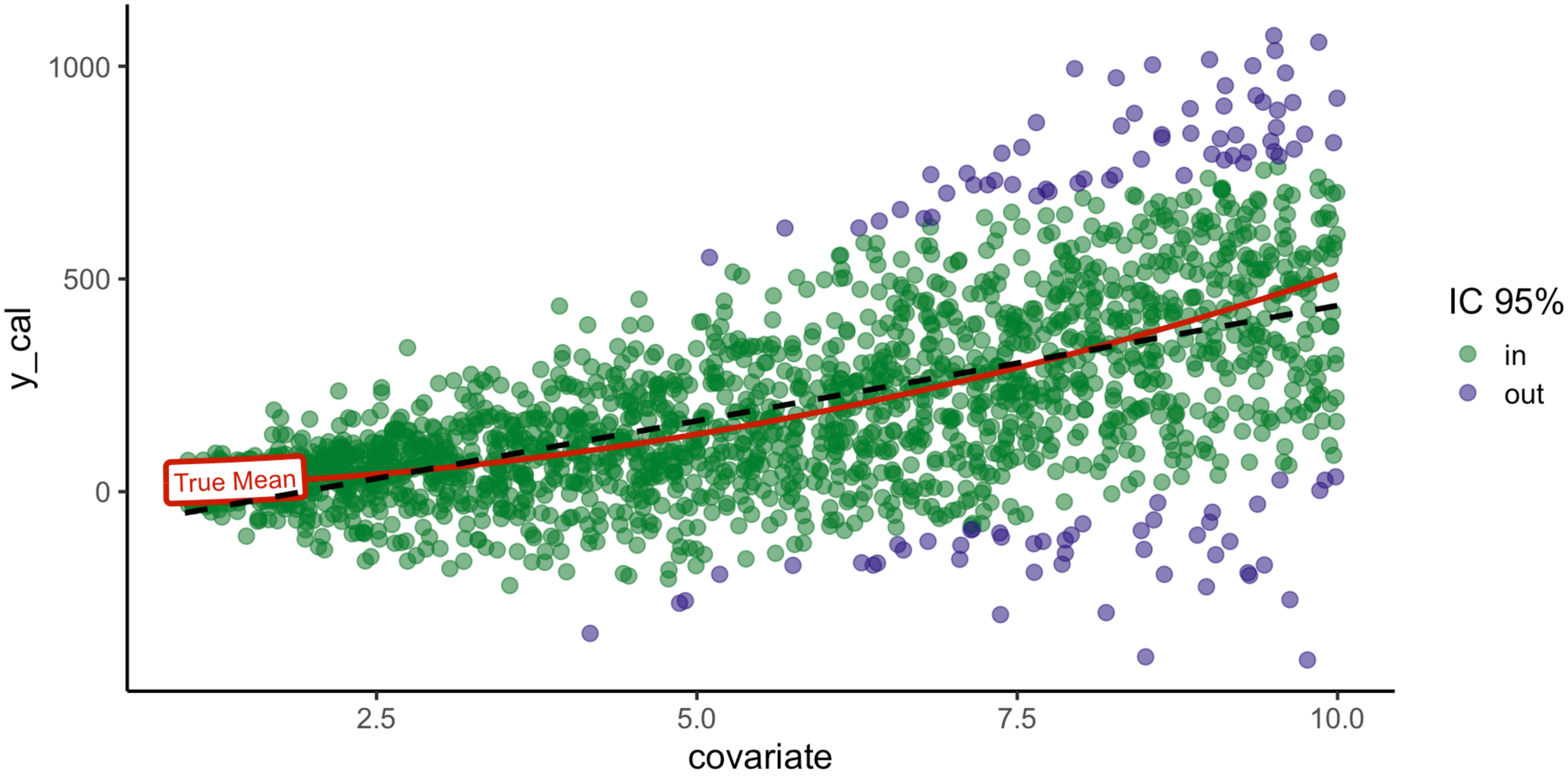
And the fitted model,

$$\hat{y_i} = \beta_0 + \beta_1 x_i + \epsilon_i, \; \epsilon_i \; iid \sim N(0, \sigma)$$

Observing miscalibration

A simple linear regression, just to warm up.

- Global Coverage: 94.45%.



{fig-align='center' width=960 110%}

PIT - Values

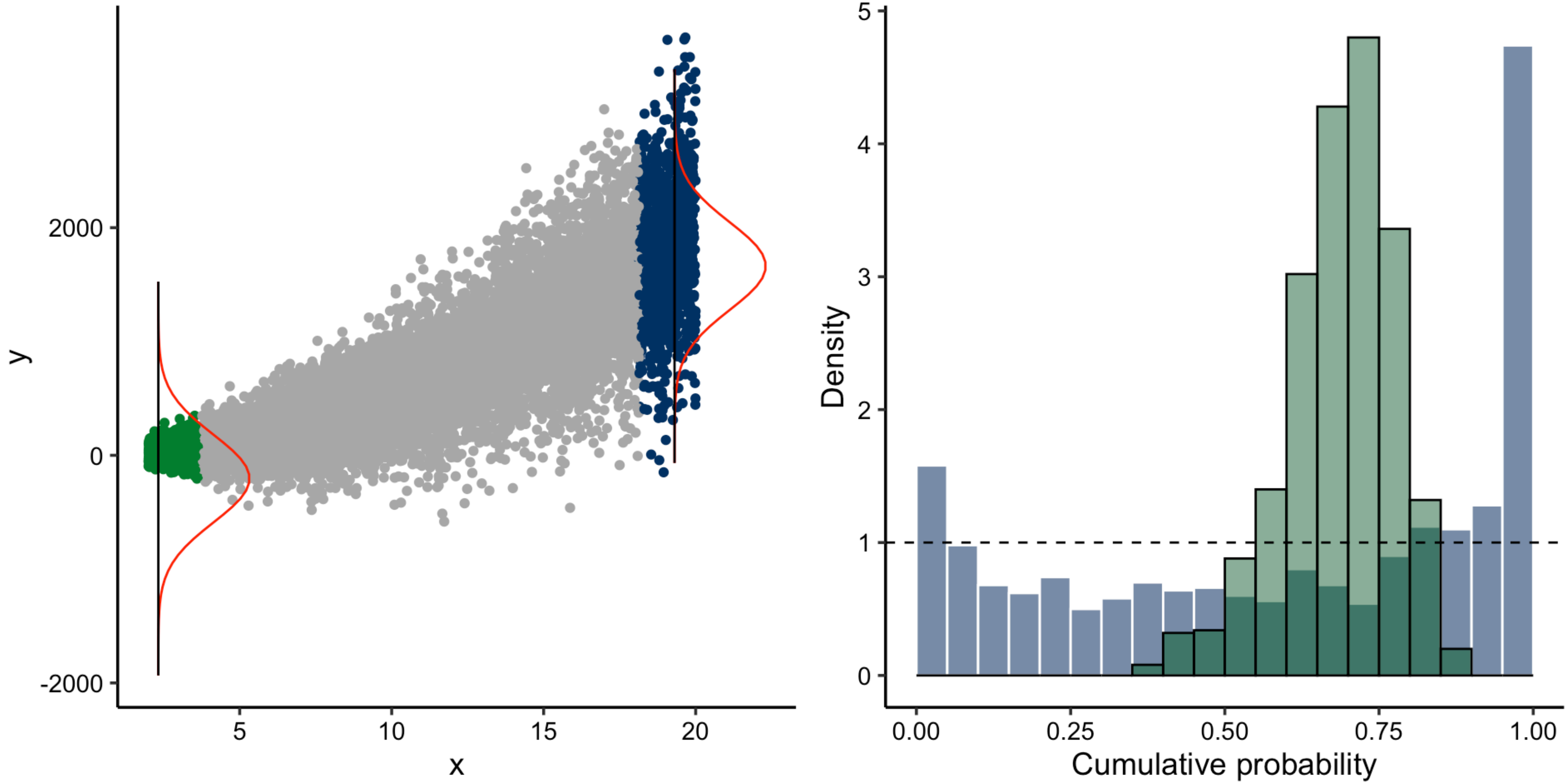
- Histogram of Probability Integral Transform (PIT) values.
- Let $F_Y(y)$ be the CDF of a continuous random variable Y, then:

$$U = F_Y(Y) \sim Uniform(0, 1)$$

- In particular, if $Y \sim Normal(\mu, \sigma)$:

$$Y = F_Y^{-1}(U) \sim Normal(\mu, \sigma)$$

Visualizing PIT-values



Recalibration

Available Packages

- R: probably
- Python: ml_insights
- Only global, focused on classification problems, and only applicable in t
  covariate space.

Method:

- Torres et al (2024): Calibration across various representations of the covariate space: useful for Artificial Neural Networks (ANNs).

Algorithm

Algorithm 1 Torres *et al.* (2023) method implemented in the package

Input:

- Recalibration set, $\{y_{\text{rec}}^{(i)}, \mathbf{x}_{\text{rec}}^{(i)}\}_{i=1}^{n}$, and new set, $\{\mathbf{x}_{\text{new}}^{(j)}\}_{j=1}^{m}$.
- A neural network and its associated predictive distribution, $\hat{F}(\cdot \mid \mathbf{X})$.
- A positive integer l defining the network's layer where the samples are to be compared.
- Neural network's outputs of the l-th layer on the recalibration set, $\{\mathbf{h}_{\text{rec}}^{(i)}\}_{i=1}^{n}$.
- A smoothing kernel $K_u(d)$ with scale parameter $u > 0$, which may be defined indirectly from a positive integer k that represents the number of observations to be used for recalibration.

Cumulative probabilities (PIT-values)

for $i \leftarrow 1$ **to** n **do**
  Set $p_{\text{rec}}^{(i)} = \hat{F}_i(y_{\text{rec}}^{(i)} | \mathbf{x}_{\text{rec}}^{(i)})$.
end for

Recalibration

for $j \leftarrow 1$ **to** m **do**
  Compute $\mathbf{h}_{\text{new}}^{(j)} = g(\mathbf{x}_{\text{new}}^{(j)})$, where g denotes the network's mapping to the l-th layer.
  Apply the approximate KNN search method to identify the set of indices, I_j, corresponding to the observations in $\{y_{\text{rec}}^{(i)}, \mathbf{x}_{\text{rec}}^{(i)}\}_{i=1}^{n}$ for which $\|\mathbf{h}_{\text{rec}}^{(i)} - \mathbf{h}_{\text{new}}^{(j)}\|$ are within the k-smallest values.
  for $i \in I_j$ **do**
    Set $\tilde{y}_i^{(j)} = \hat{F}_j^{-1}(p_{\text{rec}}^{(i)} | \mathbf{x}_{\text{new}}^{(j)})$ and assign it a weight $w_i^{(j)} \propto K_u(\|\mathbf{h}_{\text{rec}}^{(i)} - \mathbf{h}_{\text{new}}^{(j)}\|)$.
  end for
end for

Output:

A set of k weighted samples $\{(\tilde{y}_i^{(j)}, w_i^{(j)})\}$ from the recalibrated predictive distribution

$$\tilde{F}_j(\cdot \mid \mathbf{x}_{\text{new}}^{(j)}),$$

for $j = 1, \ldots, m$.

The Package

- On GitHub
- and on CRAN

recalibratiNN package

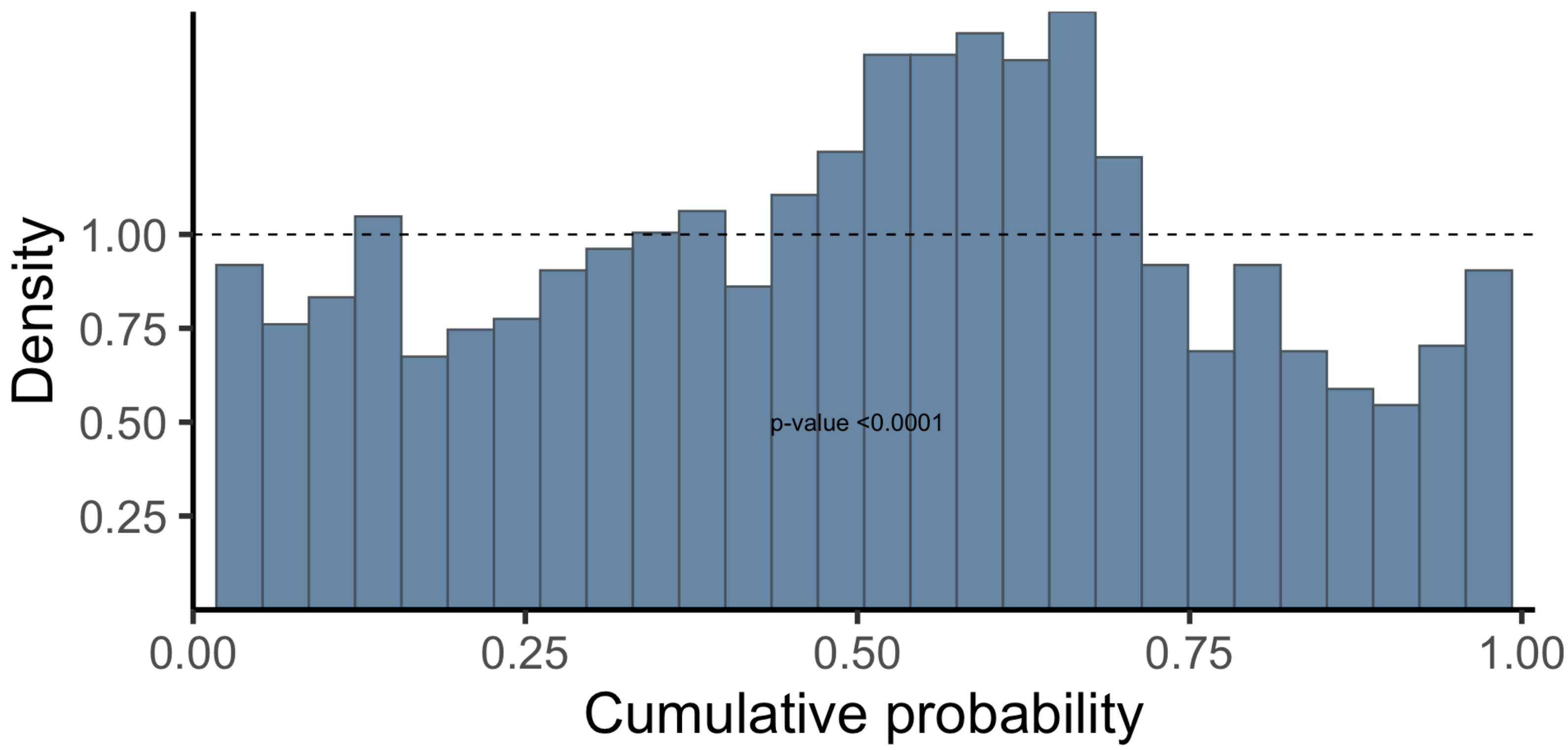
- 7 functions & 10 dependencies

Function	Description	Arguments
PIT_global	Calculates PIT values for the entire dataset	ycal, yhat, mse
PIT_local	Calculates PIT values for each cluster	xcal, ycal, yhat, mse, clusters, p_neighbours, PIT
gg_PIT_global	Plots PIT values histogram	pit, type, fill, alpha, print_p
gg_PIT_local	Plots PIT values densities for kmeans clusters	pit_local, alpha, linewidth, pal, facet
recalibrate	Recalibrates the model	yhat_new, space_new, space_cal, pit_values, mse, type p_neighbours, epsilon

Visualizing miscalibration

Global Calibration

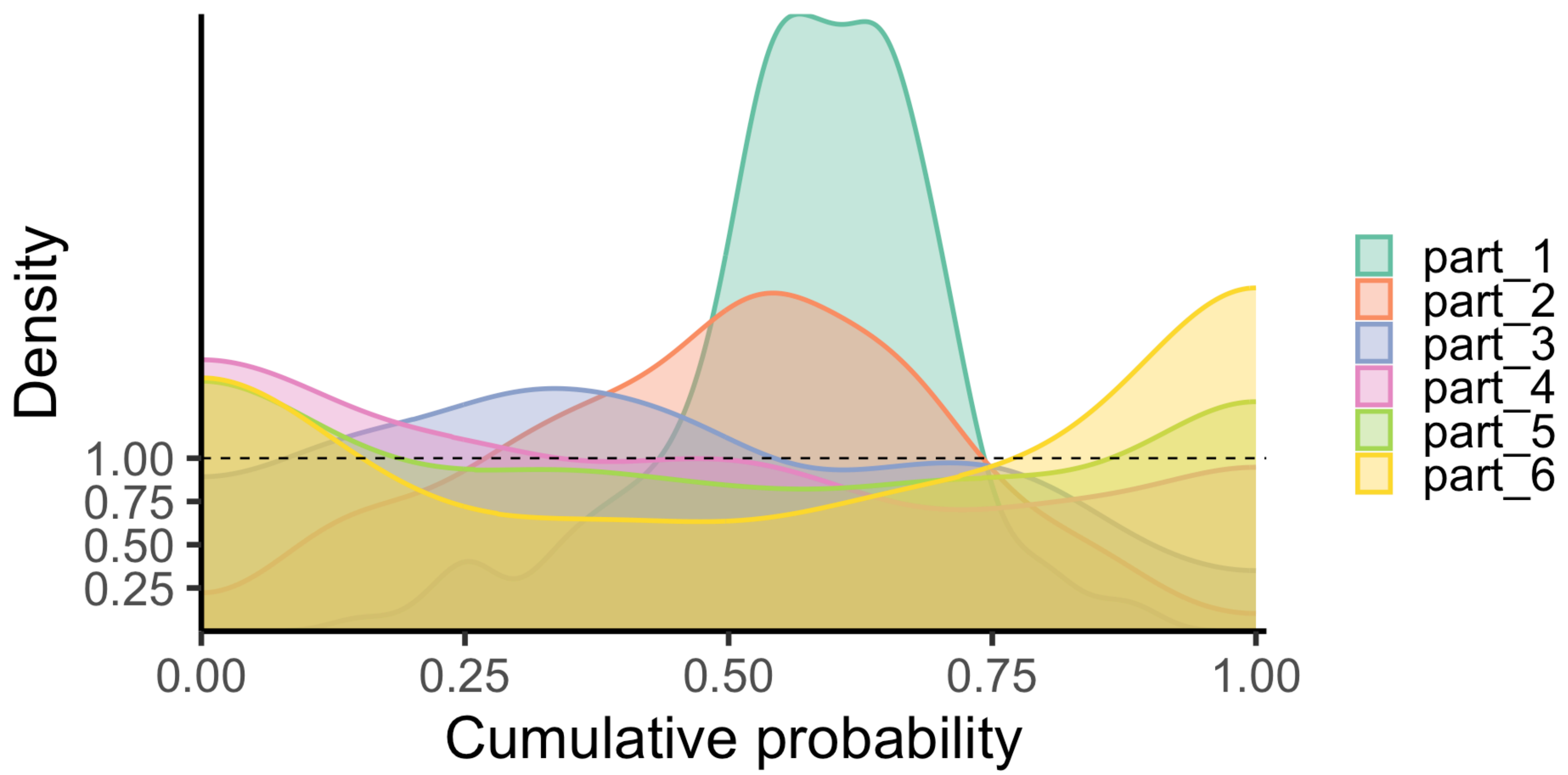
```r
pit <- PIT_global(ycal = y_cal, # true values from calib. set.
                  yhat = y_hat_cal, # predictions for calb. set.
                  mse  = MSE_cal) # MSE from calibration set.

gg_PIT_global(pit,
               type = "histogram",
              fill = "steelblue4",
              alpha = 0.8,
              print_p = TRUE
            )
```



Density
1.00
0.75
0.50
0.25
p-value <0.0001
0.00
0.25
0.50
0.75
1.00
Cumulative probability

Local Calibration

```r
pit_local <- PIT_local(xcal = x_cal,
                       ycal = y_cal,
                       yhat = y_hat_cal,
                       mse = MSE_cal,
                       clusters = 6,
                       p_neighbours = 0.2,
                       PIT = PIT_global)

gg_PIT_local(pit_local)
```



Density
1.00
0.75
0.50
0.25
0.00
0.25
0.50
0.75
1.00
Cumulative probability
part_1
part_2
part_3
part_4
part_5
part_6

Neural Networks

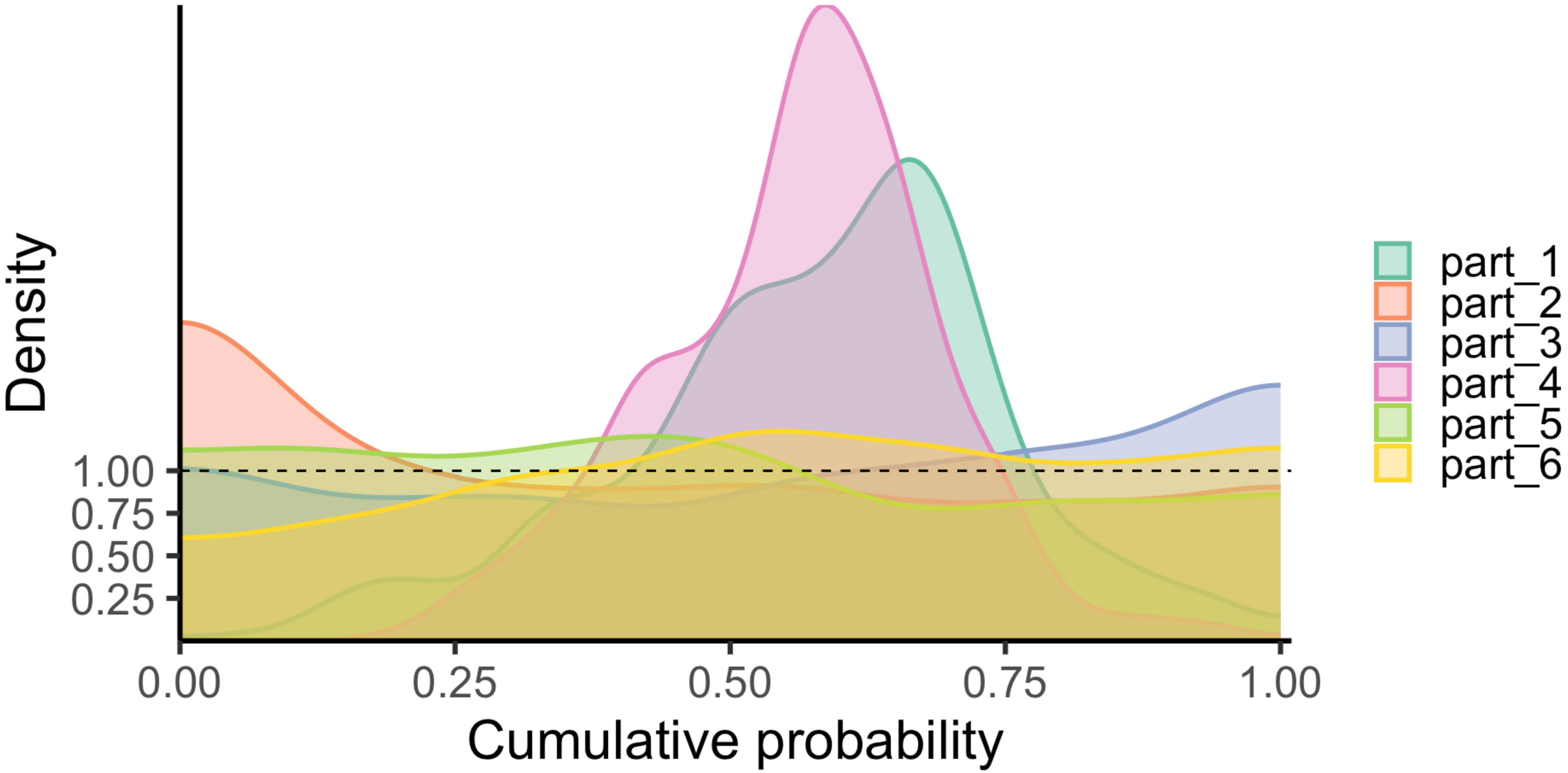
Data

```r
set.seed(42)   # The Answer to the Ultimate Question of Life, The Universe, and

n <- 10000

x <- cbind(x1 = runif(n, -3, 3),
           x2 = runif(n, -5, 5))

mu_fun <- function(x) {
  abs(x[,1]^3 - 50*sin(x[,2]) + 30)}

mu <- mu_fun(x)
y <- rnorm(n,
           mean = mu,
           sd=20*(abs(x[,2]/(x[,1]+ 10))))

```

Keras

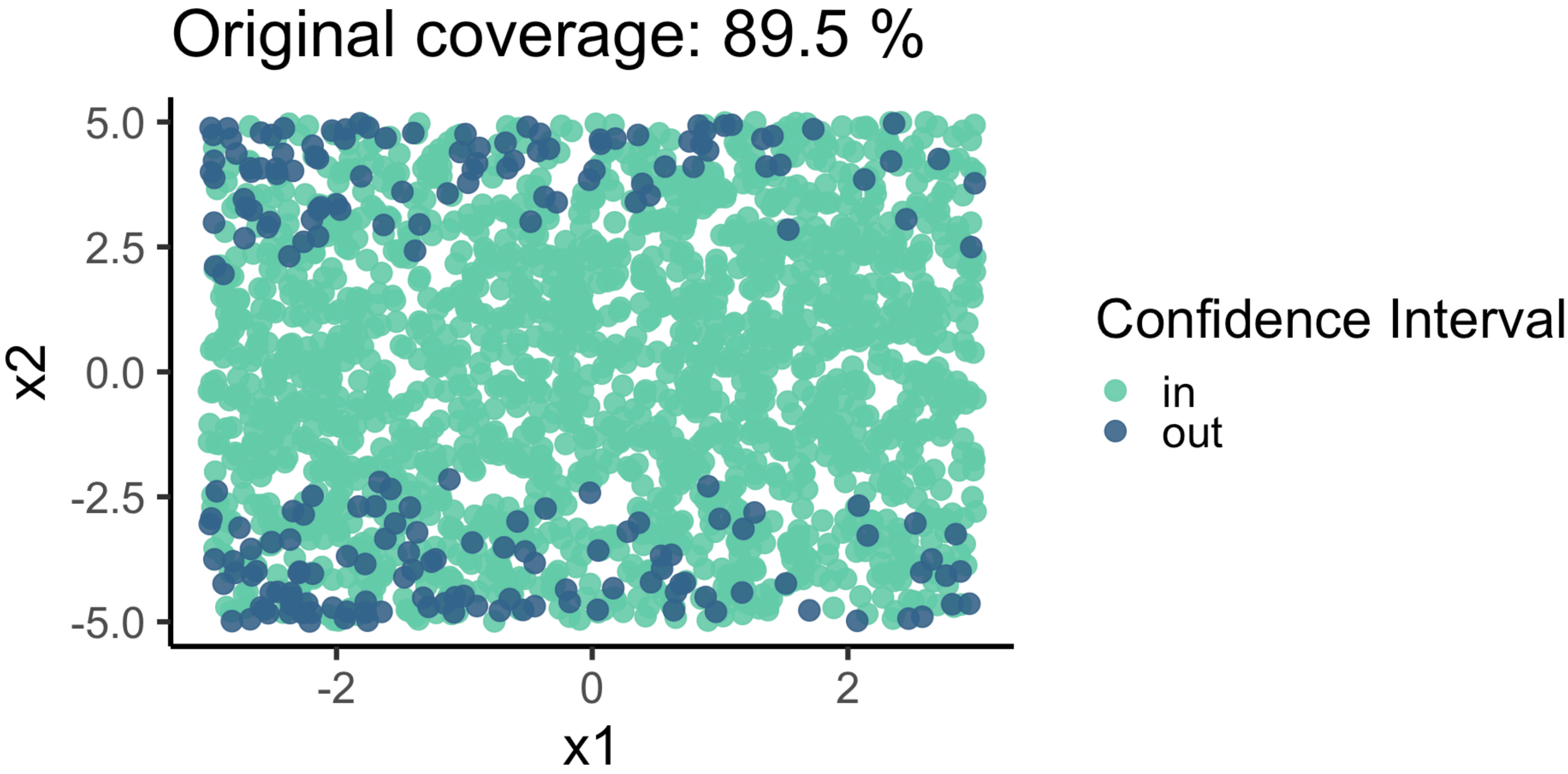
```r
model_nn <- keras_model_sequential()

model_nn |>
  layer_dense(input_shape=2,
              units=800,
              use_bias=T,
              activation = "relu",
              kernel_initializer="random_normal",
              bias_initializer = "zeros") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units=800,
              use_bias=T,
              activation = "relu",
              kernel_initializer="random_normal",
              bias_initializer = "zeros") |>
```

Observing miscalibration



{width=960 110%}

Coverage



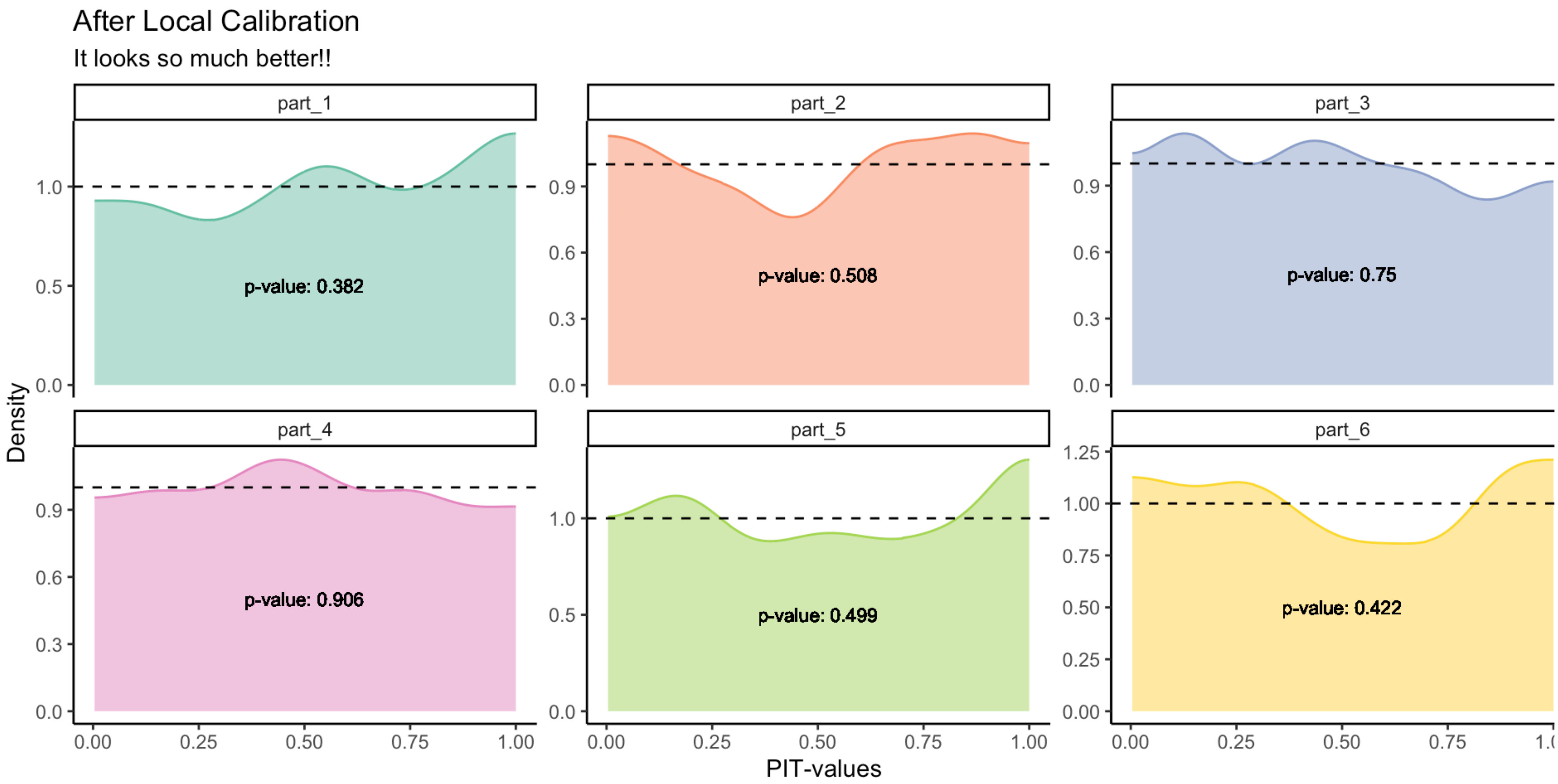
{width=960 110%}

Recalibration

```
recalibrated <-
  recalibrate(
     pit_values = pit,       # global pit values calculated earlier.
    mse = MSE_cal,          # MSE from calibration set
    yhat_new = y_hat_test, # predictions of test set
    space_cal = x_cal,      # covariates of calibration set
    space_new = x_test,     # covariates of test set



    type = "local",         # type of calibration
    p_neighbours = 0.08)    # proportion of calibration to use as nearest neighbo

y_hat_rec <- recalibrated$y_samples_calibrated_wt
```

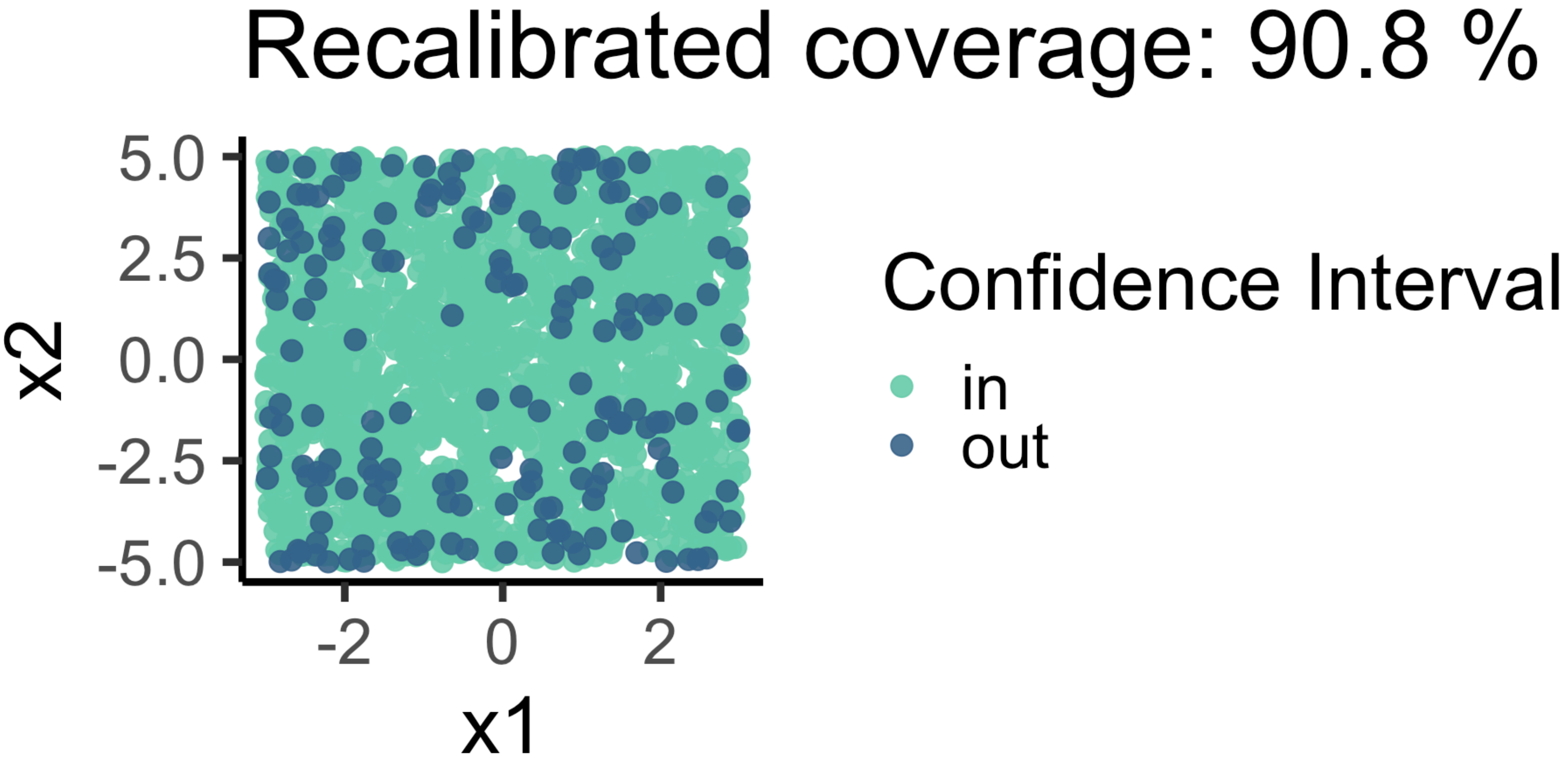
- That is it!
- These new values in `y_hat_rec` are, by definition, more calibrated tha the original ones.

Shall we see?



{width=960 110}

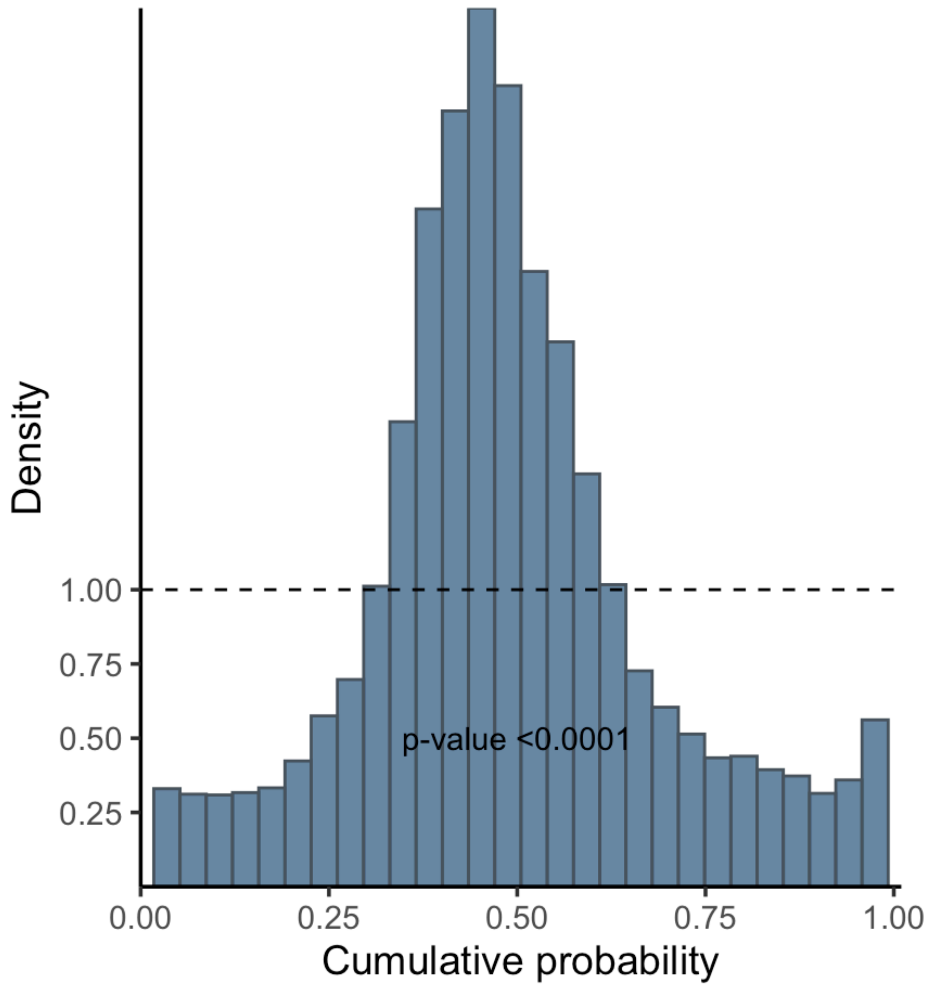
Coverage

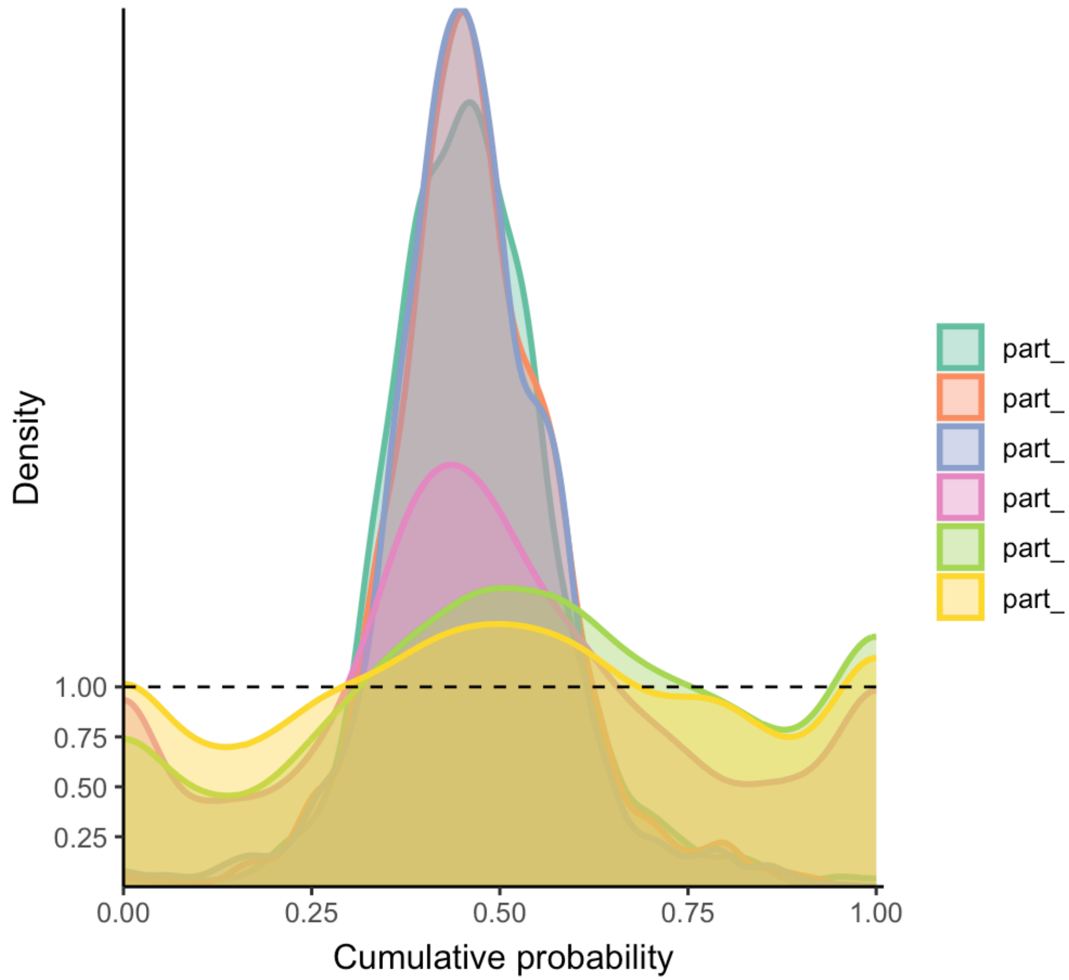


{width=960 110}

Real data

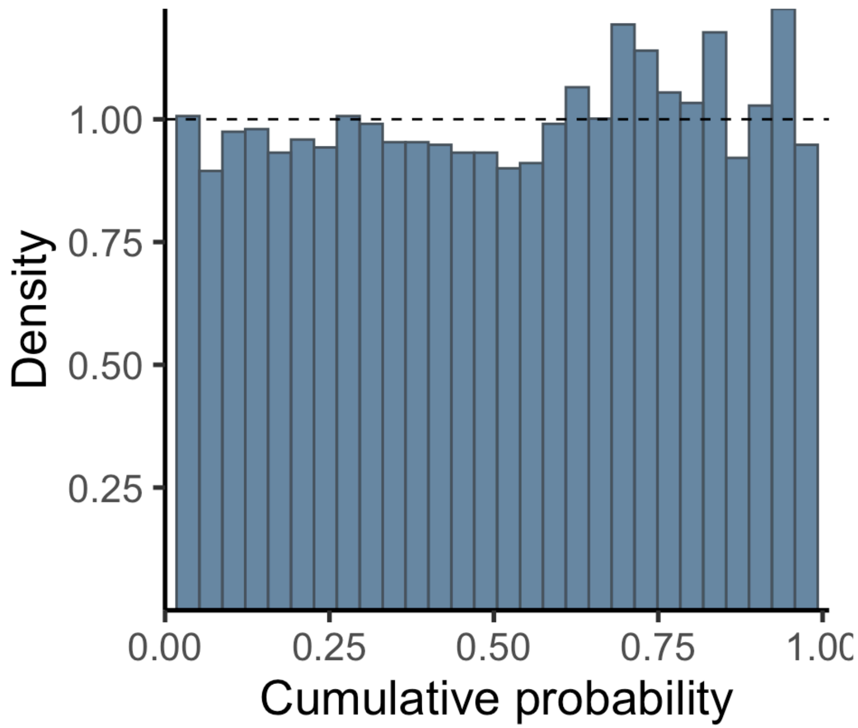
Diamonds dataset

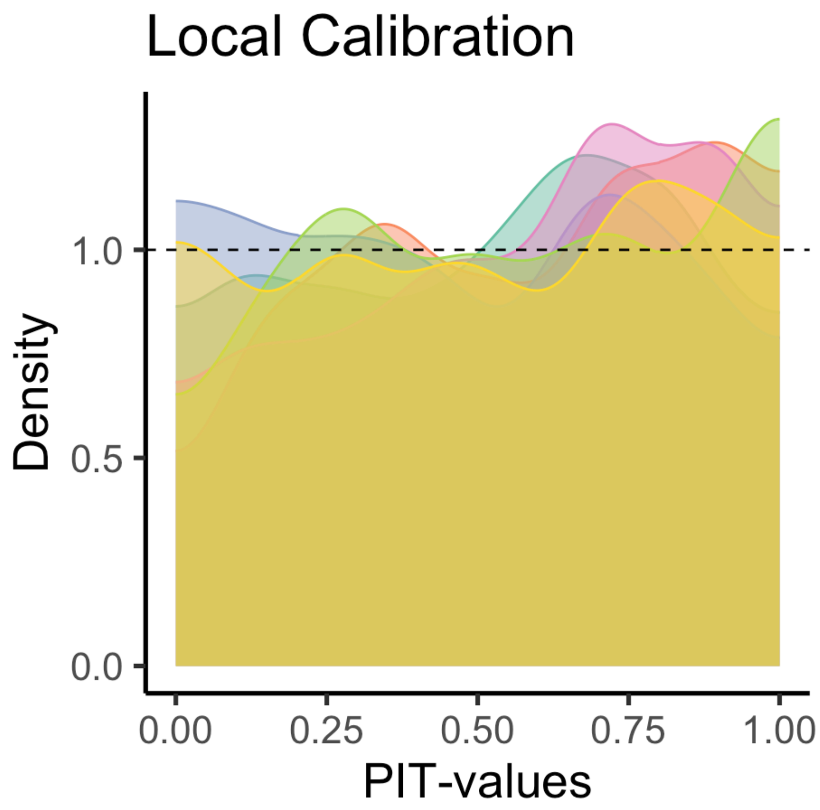




After Recalibration

Calibrated using a second hidden layer.





Conclusions and Future Work

- Effective Visualization of Miscalibration.
- Advantages related to other packages
  - Focused in regression models
  - Local recalibration
  - Recalibration at intermediate layers.

Future Developments:

- Integration with other packages, broader input types, cross-validation methods
- Handle models with arbitrary predictive distributions.

Thank You!

GitHub