

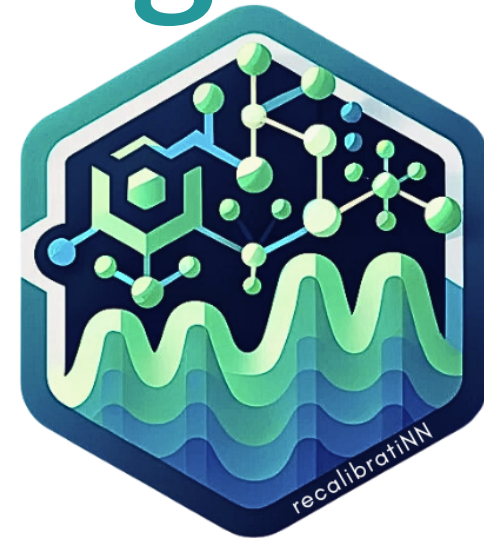
# Recalibration of Gaussian Neural Networks regression models:

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  $(x_i, y_i)$ ,  $i \in (1, \dots, n)$  generated by an heteroscedastic non-linear model:

$$x_i \sim \text{Uniform}(1, 10)$$

$$y_i | x_i \sim \text{Normal}(\mu = f_1(x_i), \sigma = f_2(x_i))$$

$$f_1(x) = 5x^2 + 10 ; f_2(x) = 30x$$

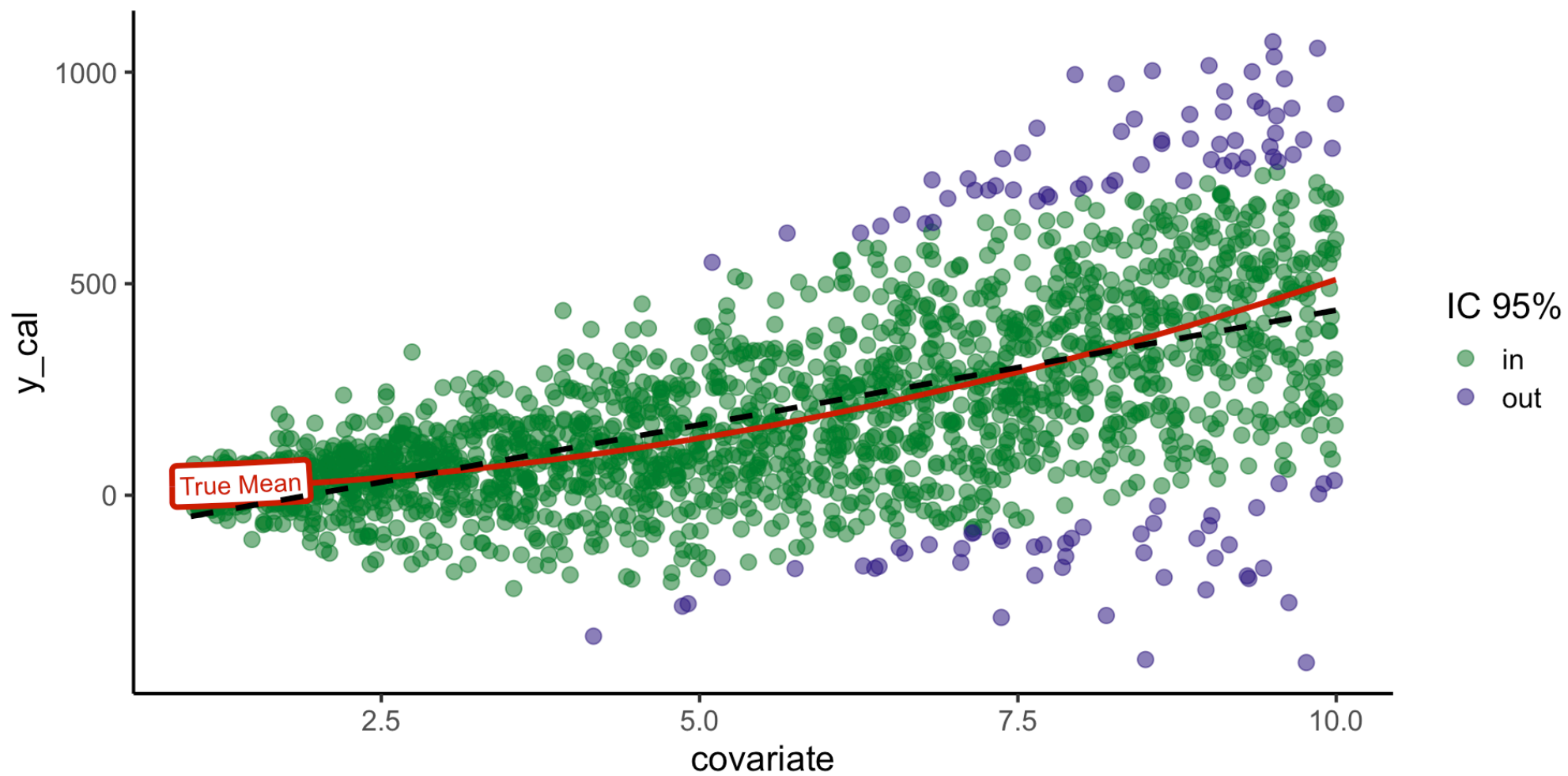
And the fitted model,

$$\hat{y}_i = \beta_0 + \beta_1 x_i + \epsilon_i, \epsilon_i \text{ iid } \sim N(0, \sigma)$$

# Observing miscalibration

A simple linear regression, just to warm up.

- Global Coverage: 94.45%.



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

# PIT - Values

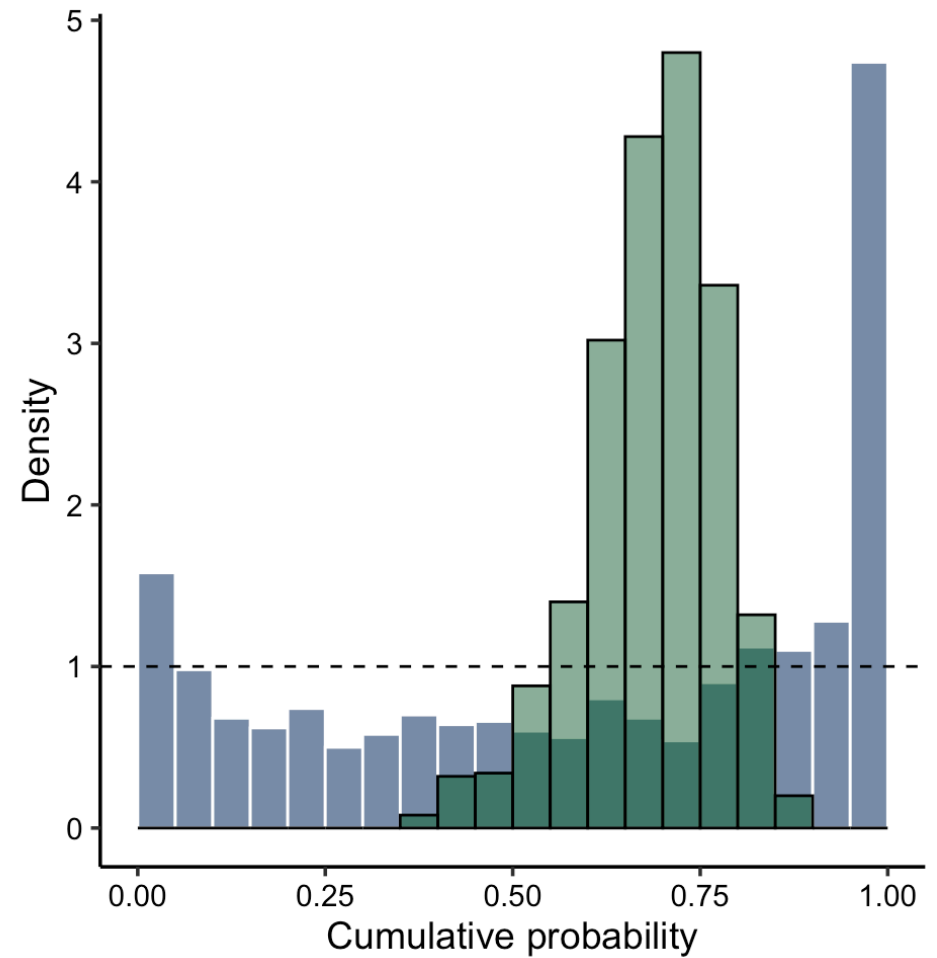
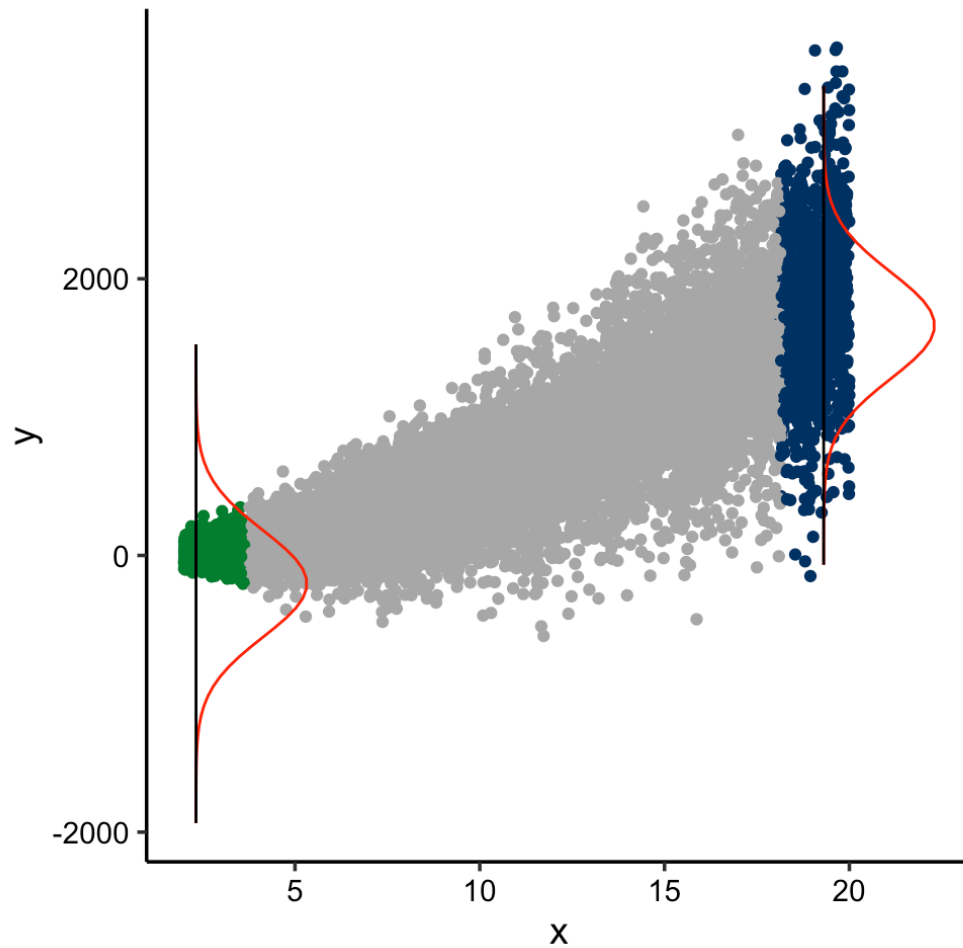
- Histogram of Probability Integral Transform (PIT) values.
- Let  $F_Y(y)$  be the CDF of a continuous random variable  $Y$ , then:

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

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

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

# Visualizing PIT-values





# Recalibration

## Available Packages

- R: `probably`
- Python: `ml_insights`
- Only global, focused on classification problems, and only applicable in 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:**

1:

- 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,  $\hat{F}(\cdot | \mathbf{X})$ .
- 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)**

2: **for**  $i \leftarrow 1$  **to**  $n$  **do**

3:   Set  $p_{\text{rec}}^{(i)} = \hat{F}_i(y_{\text{rec}}^{(i)} | \mathbf{x}_{\text{rec}}^{(i)})$ .

4: **end for**

**Recalibration**

5: **for**  $j \leftarrow 1$  **to**  $m$  **do**

6:   Compute  $\mathbf{h}_{\text{new}}^{(j)} = g(\mathbf{x}_{\text{new}}^{(j)})$ , where  $g$  denotes the network's mapping to the  $l$ -th layer.

7:   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.

8:   **for**  $i \in I_j$  **do**

9:     Set  $\tilde{y}_i^{(j)} = \hat{F}_j^{-1}(p_{\text{rec}}^{(i)} | \mathbf{x}_{\text{new}}^{(j)})$  and assign it a weight  $w_i^{(j)} \propto K_u(\|\mathbf{h}_{\text{rec}}^{(i)} - \mathbf{h}_{\text{new}}^{(j)}\|)$ .

10:   **end for**

11: **end for**

**Output:**

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

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

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

---

# The Package

- On GitHub
- and on CRAN

# recalibratiNN package

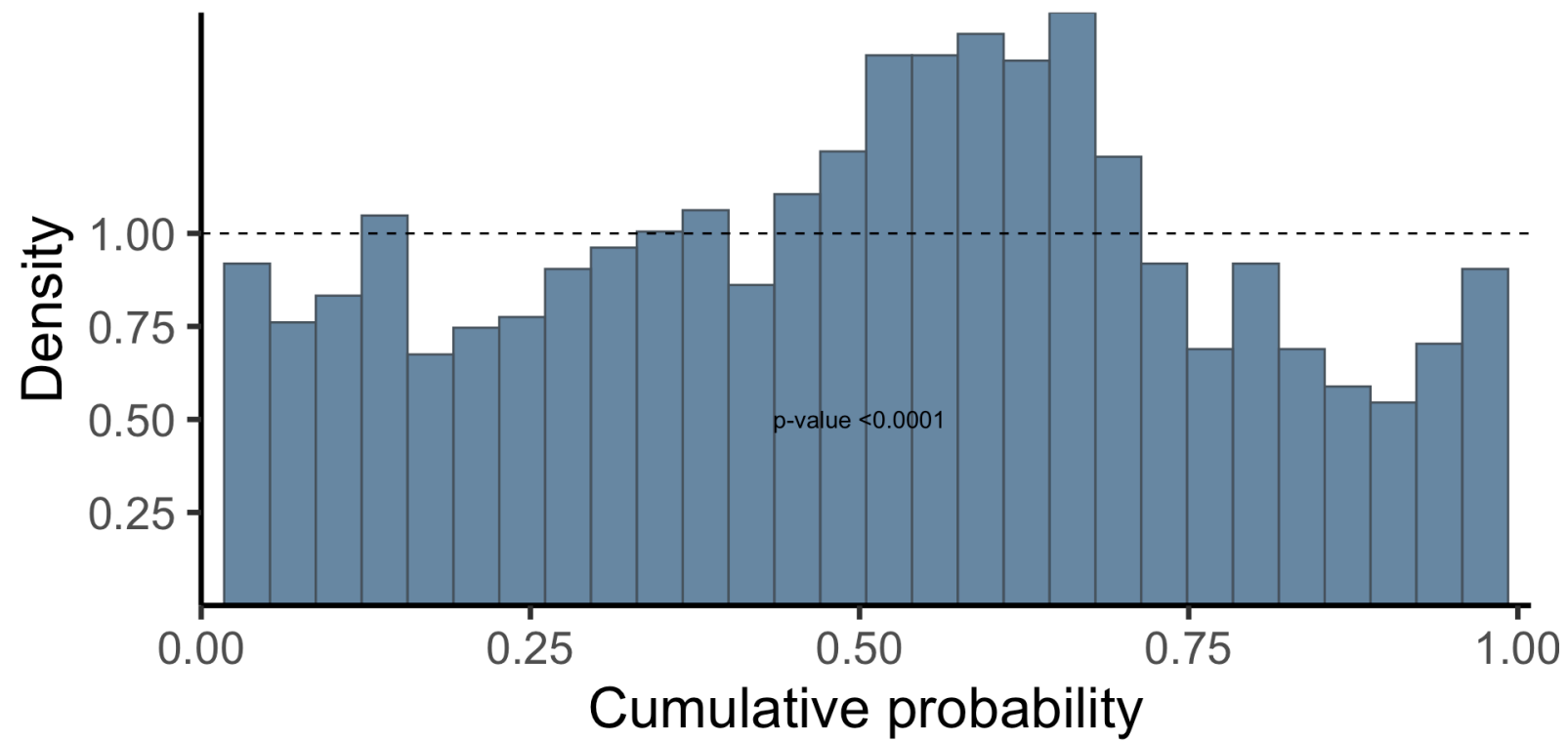
- 7 functions & 10 dependencies

Function	Description	Arguments
PIT_global	Calculates PIT values for the entire dataset	yca, 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

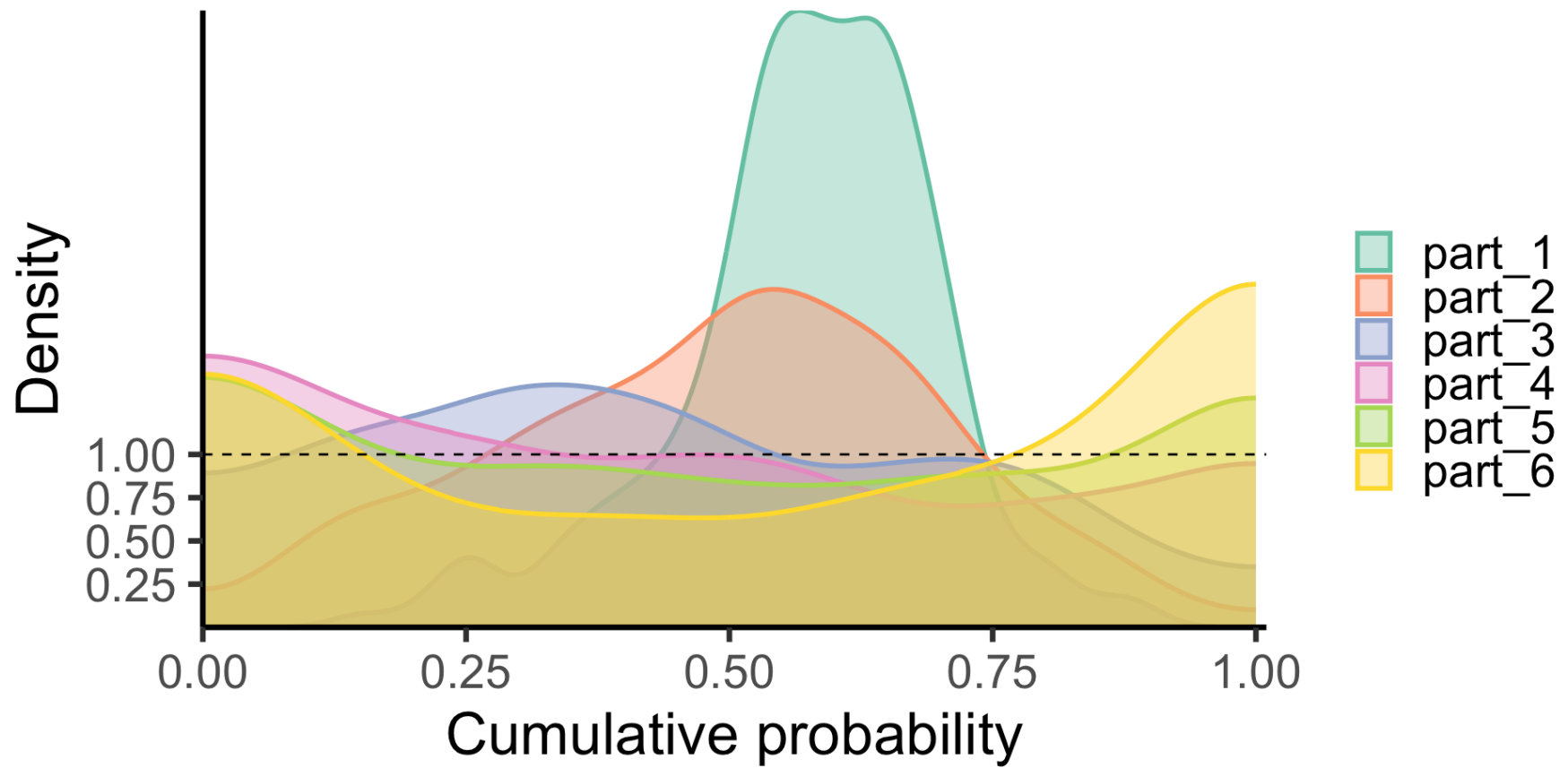
```
1 pit <- PIT_global(ycal = y_cal, # true values from calib. set.  
2                   yhat = y_hat_cal, # predictions for calb. set.  
3                   mse   = MSE_cal) # MSE from calibration set.  
4  
5 gg_PIT_global(pit,  
6               type = "histogram",  
7               fill = "steelblue4",  
8               alpha = 0.8,  
9               print_p = TRUE  
10              )
```



# Local Calibration

```
1 pit_local <- PIT_local(xcal = x_cal,  
2                       ycal = y_cal,  
3                       yhat = y_hat_cal,  
4                       mse = MSE_cal,  
5                       clusters = 6,  
6                       p_neighbours = 0.2,  
7                       PIT = PIT_global)  
8  
9 gg_PIT_local(pit_local)
```





# Neural Networks

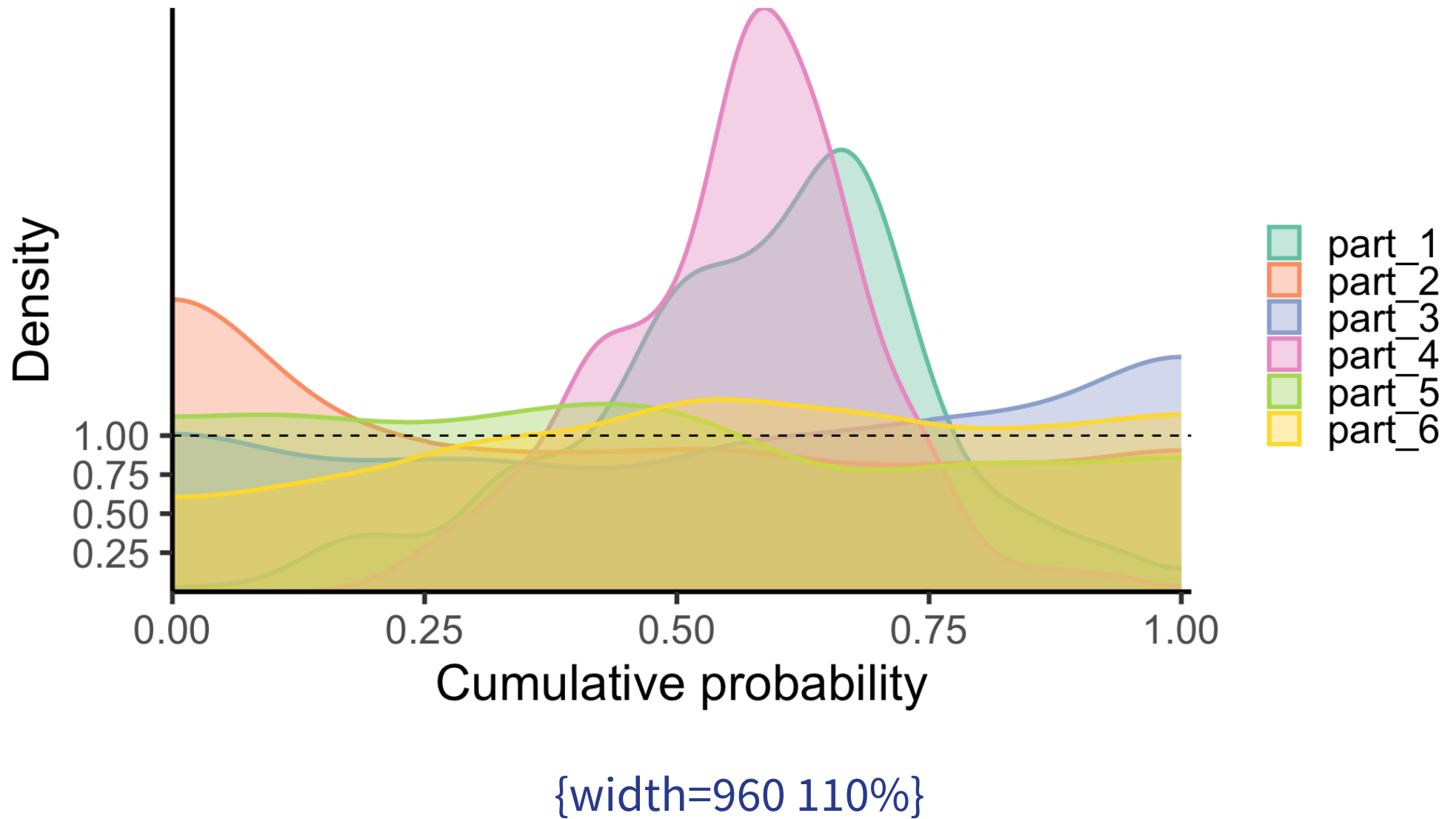
# Data

```
1 set.seed(42)    # The Answer to the Ultimate Question of Life, The Universe, and
2
3 n <- 10000
4
5 x <- cbind(x1 = runif(n, -3, 3),
6            x2 = runif(n, -5, 5))
7
8 mu_fun <- function(x) {
9   abs(x[,1]^3 - 50*sin(x[,2]) + 30)}
10
11 mu <- mu_fun(x)
12 y <- rnorm(n,
13           mean = mu,
14           sd=20*(abs(x[,2]/(x[,1]+ 10))))
15
```

# Keras

```
1 model_nn <- keras_model_sequential()  
2  
3 model_nn |>  
4   layer_dense(input_shape=2,  
5               units=800,  
6               use_bias=T,  
7               activation = "relu",  
8               kernel_initializer="random_normal",  
9               bias_initializer = "zeros") %>%  
10  layer_dropout(rate = 0.1) %>%  
11  layer_dense(units=800,  
12             use_bias=T,  
13             activation = "relu",  
14             kernel_initializer="random_normal",  
15             bias_initializer = "zeros") |>
```

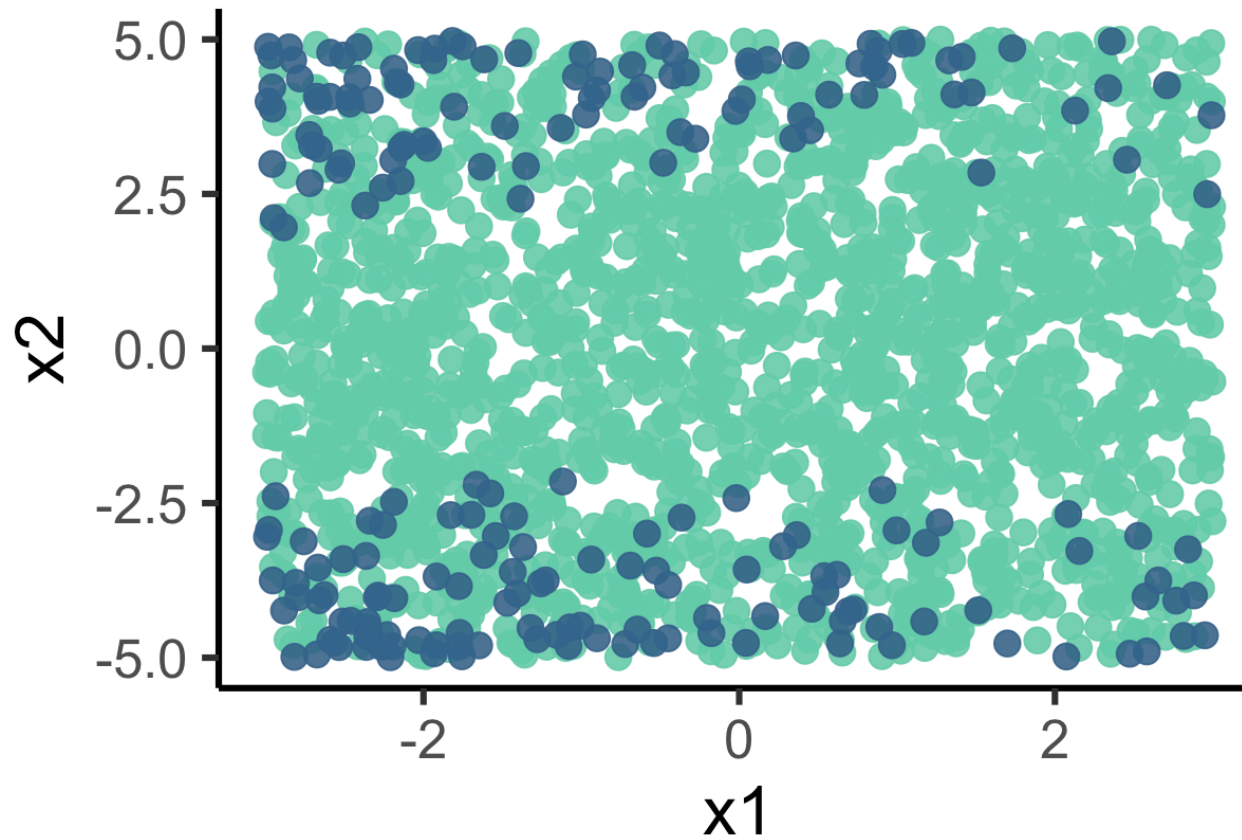
# Observing miscalibration





# Coverage

Original coverage: 89.5 %



Confidence Interval

- in
- out

{width=960 110%}





# Recalibration

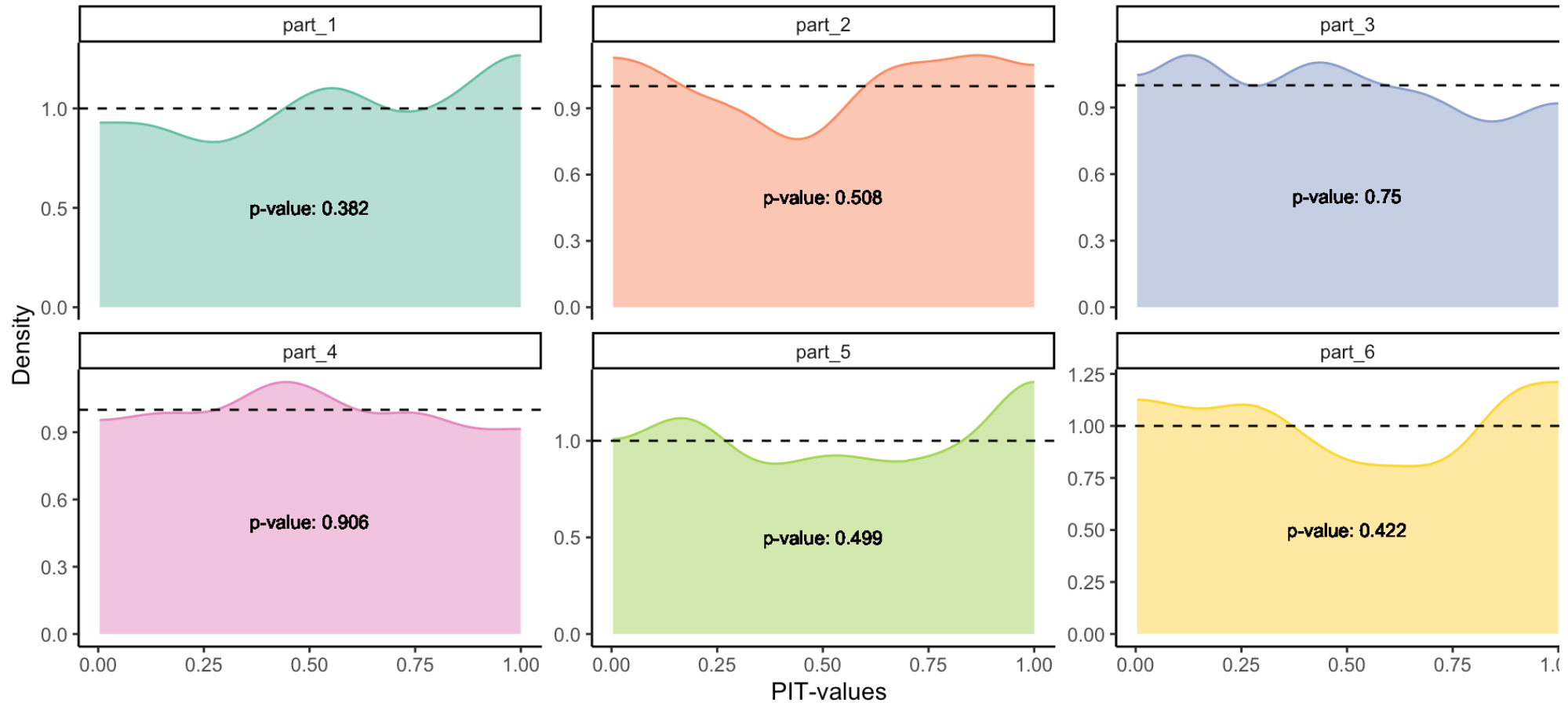
```
1 recalibrated <-  
2   recalibrate(  
3     pit_values = pit,      # global pit values calculated earlier.  
4     mse = MSE_cal,        # MSE from calibration set  
5     yhat_new = y_hat_test, # predictions of test set  
6     space_cal = x_cal,    # covariates of calibration set  
7     space_new = x_test,   # covariates of test set  
8  
9  
10  
11     type = "local",      # type of calibration  
12     p_neighbours = 0.08) # proportion of calibration to use as nearest neighbors  
13  
14 y_hat_rec <- recalibrated$y_samples_calibrated_wt
```

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

# Shall we see?

After Local Calibration

It looks so much better!!

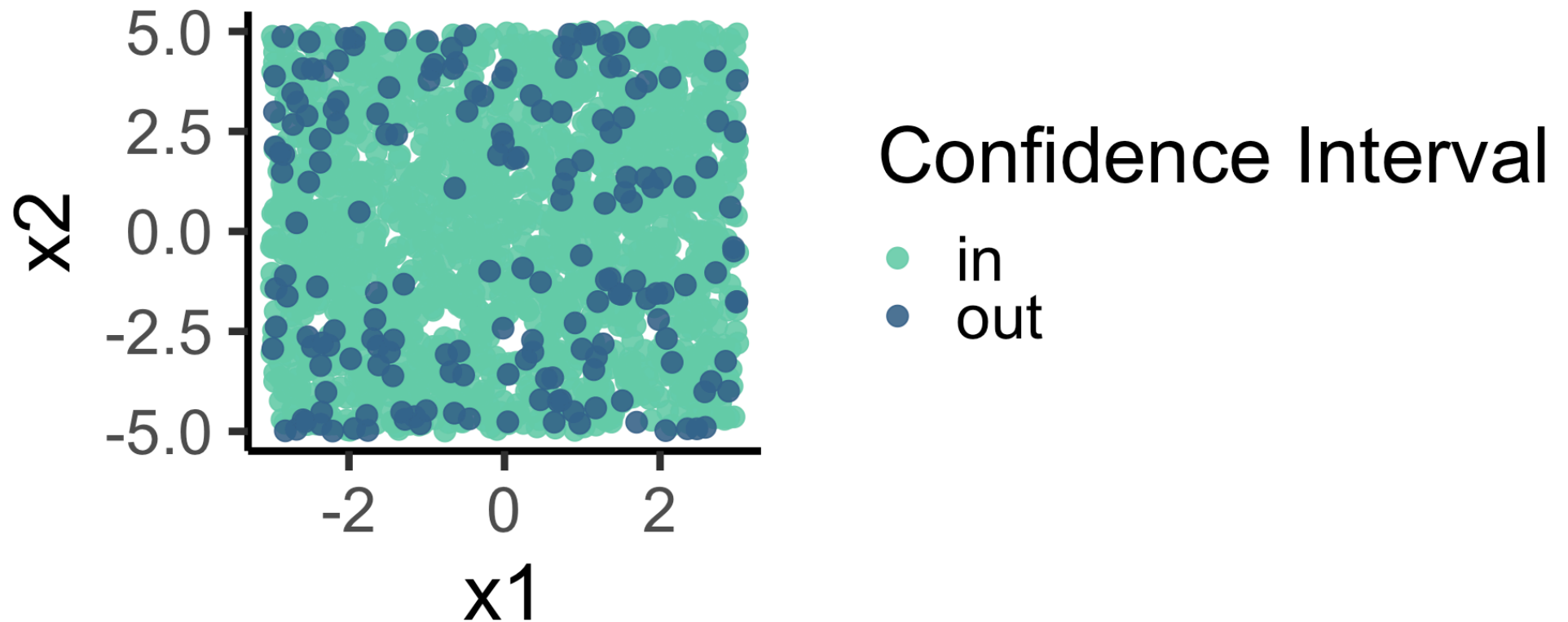


{width=960 110}



# Coverage

Recalibrated coverage: 90.8 %

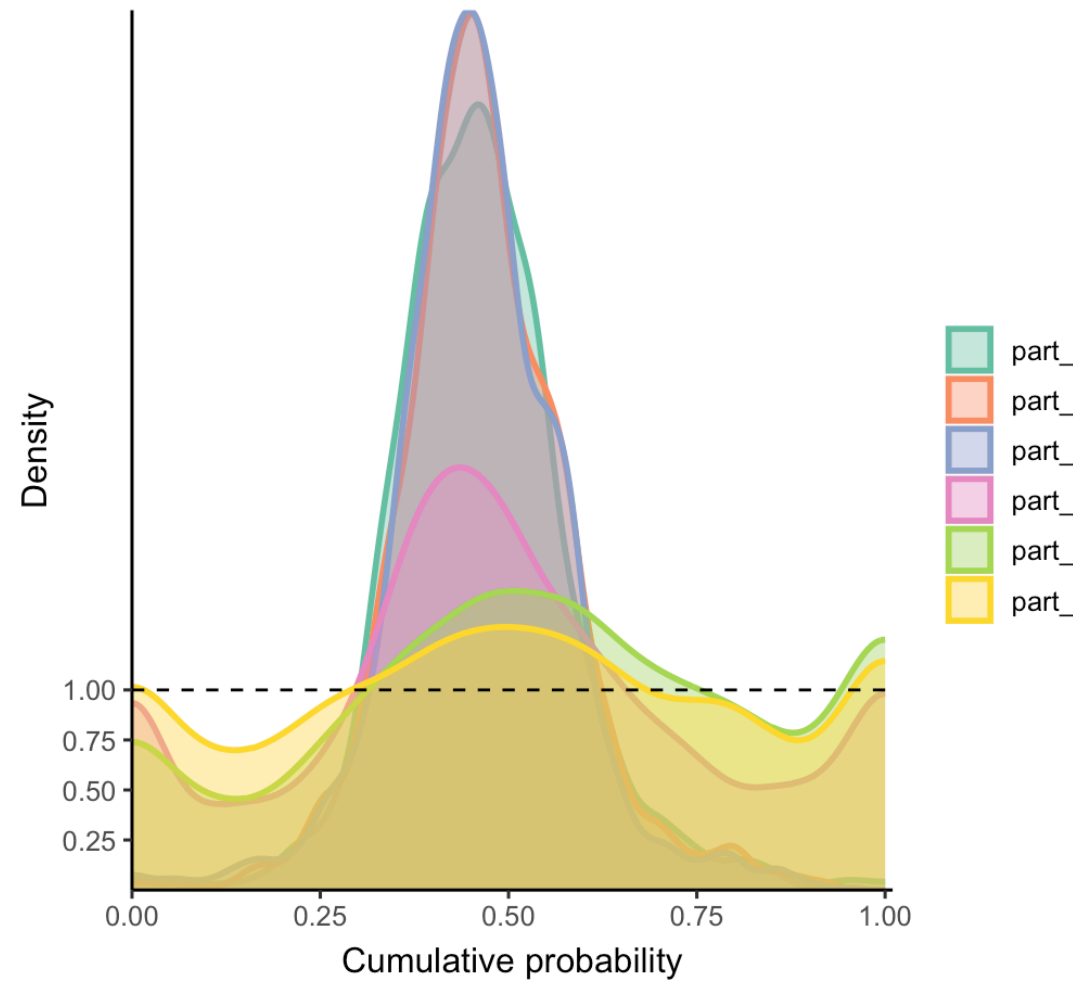
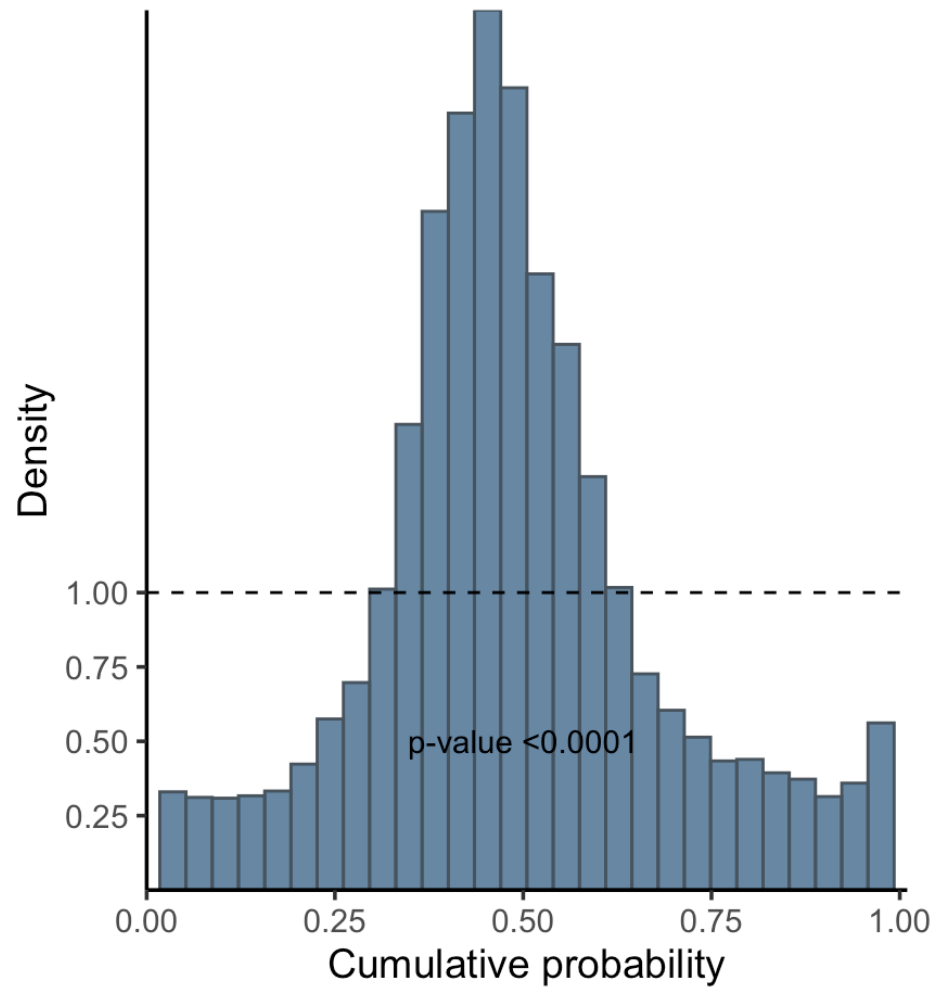


{width=960 110}



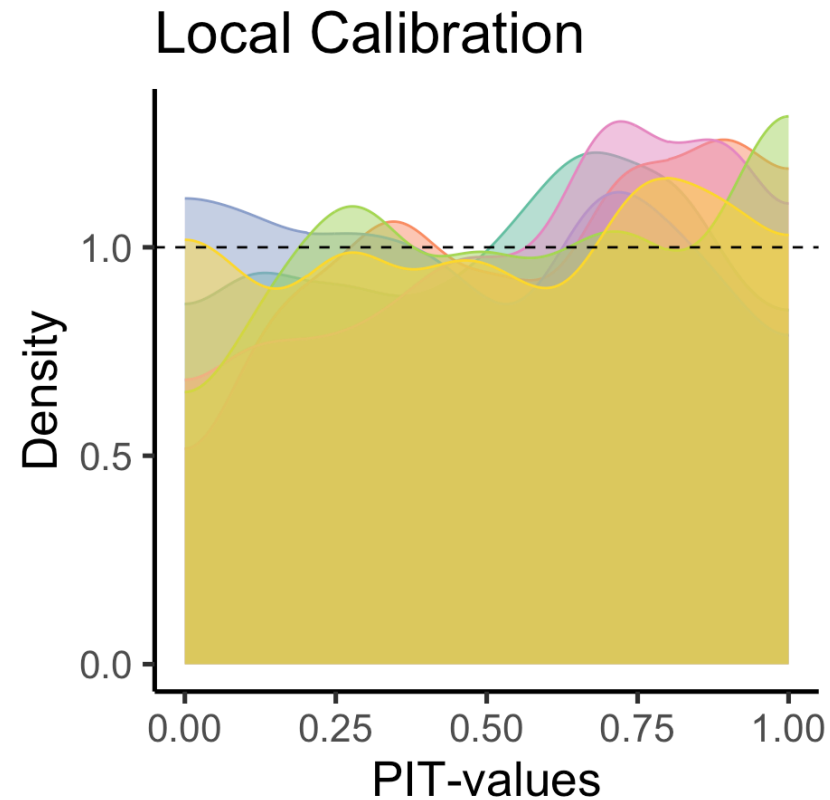
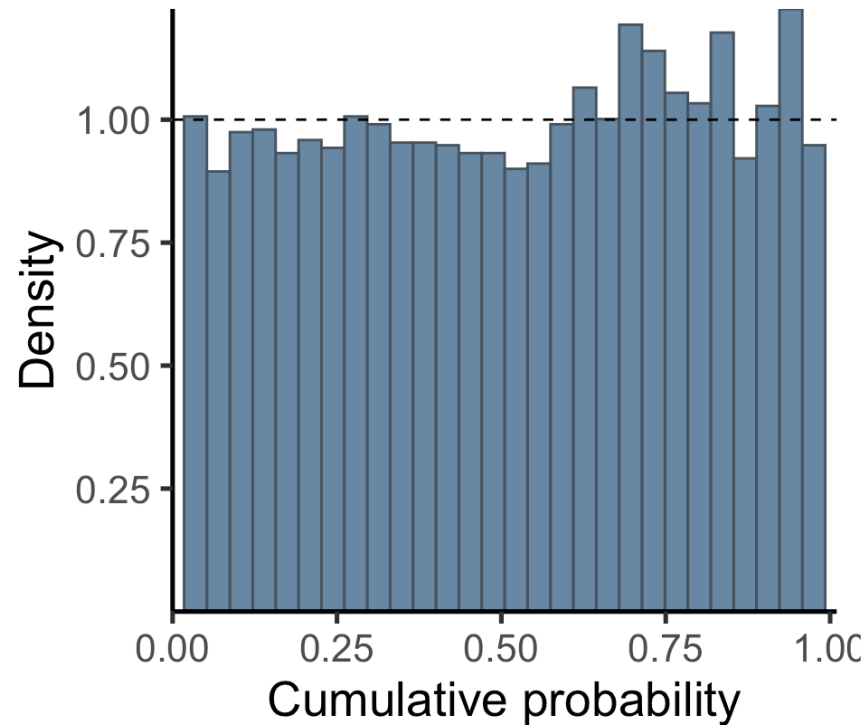
**Real data**

# 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