

# aggreCAT: an R package for mathematically aggregating expert judgements

**Elliot Gould** 


School of Agriculture, Food and Ecosystem Sciences, The University of Melbourne  
School of Historical and Philosophical Studies, The University of Melbourne  
elliott.gould@unimelb.edu.au

**Charles T. Gray**

Evidence Synthesis Lab, Newcastle University

**Aaron Willcox** 

School of Historical and Philosophical Studies, The University of Melbourne

**Rose E. O'Dea** 

School of Historical and Philosophical Studies, The University of Melbourne

**Rebecca Groenewegen** 

School of Agriculture, Food and Ecosystem Sciences, The University of Melbourne

**David P. Wilkinson** 

School of Agriculture, Food and Ecosystem Sciences, The University of Melbourne

**Abstract.** Structured elicitation protocols, such as the IDEA protocol, are used to elicit probabilistic judgements from multiple domain experts about uncertain events across fields including ecology, biosecurity risk assessment, and metascience. Individual expert judgements must subsequently be mathematically aggregated into a single group forecast. While the simplest case involves combining a set of point-estimates from multiple individuals, this process is further complicated when judgements include uncertainty bounds, or when elicitation is conducted across multiple rounds. This paper presents aggreCAT, an open-source R package that provides 29 aggregation methods for combining individual expert judgements into a single probabilistic estimate, accommodating designs ranging from single-round point estimates to multi-round three-point elicitation. The package follows tidy data principles, enabling straightforward integration with existing R workflows for application at scale. Methods range from unweighted arithmetic combinations to performance-weighted schemes and Bayesian models, with weights derived from uncertainty intervals, shifts in judgements between elicitation rounds, and breadth of expert reasoning. We provide worked examples illustrating the mechanics of representative aggregation methods, a general workflow for batch aggregation across multiple forecasts and methods, and built-in functions for evaluating and visualising forecast performance against known outcomes. aggreCAT fills a substantive gap in open software for mathematically aggregating expert judgement, and is intended to support researchers and decision analysts in rapidly and rigorously synthesising outputs from structured elicitation exercises.

**Keywords.** mathematical aggregation; expert judgement; replicability; R

## 1 Introduction

Expert judgement is frequently used to inform forecasting about uncertain future events across a range of fields and applications including ecological management and decision making (Legge et al. 2022), biosecurity risk analyses (Wittmann et al. 2015), replicability forecasts (Mody et al. 2026), and horizon scanning exercises (Sutherland et al. 2017). Judgements from groups of experts tend to perform better

---

than a single expert (Goossens et al. 2008), and it is best-practice to elicit judgements from diverse groups so that group members can bring “different perspectives, cross-examine each others’ reasoning, and share information”, however judgements or forecasts must then be distilled into a single forecast, ideally accompanied by estimates of uncertainty around those estimates (Hanea et al. 2021). Judgements from multiple experts may be combined into a single score using either behavioural approaches that force experts into forming consensus, or by using mathematical approaches (Goossens et al. 2008).

Aggregated elicited judgements form a critical component of decision-making across multiple contexts, and while there are a variety of methods for mathematically aggregating expert judgements into single point-predictions, there has been a dearth of accessible tools for implementing any aggregation more complex than linear averages. Many existing software tools are closed-source, are irreproducible, or written in programming languages rarely used by ecologists or metascientists. Methods implemented in R, one of the most widely used programming languages in the life sciences (Gao et al. 2025), are limited in scope, or have been archived from CRAN. The R package **SHELF** implements only a single method (weighted linear pool) for aggregating expert judgements (Oakley 2026), but integrates with **expertsurv** to incorporate expert judgement into survival analysis (Cooney and White 2023), while **opera** provides a suite of methods for aggregating time-series predictions, but does not implement methods for aggregating point-predictions with uncertainty bounds (Gaillard and Goude 2026).

**aggreCAT** is an open, reliable, and modular R package for the mathematical aggregation of expert elicitation data with a straightforward user interface. The **aggreCAT** package was developed by the by the repliCATS (Collaborative Assessment for Trustworthy Science) project (Fraser et al. 2023) as part of the DARPA SCORE (Systematizing Confidence in Open Research and Evidence) program (Alipourfard et al. 2021), which aimed to generate quantitative ‘confidence scores’ – estimates of the likely replicability of research claims from the social and behavioural sciences – for use as a proxy measure of credibility in the absence of direct replication effort. **aggreCAT** arose from the need for a software solution to reliably conduct mathematical aggregation at scale. The repliCATS project used **aggreCAT** to aggregate expert elicited judgements into confidence scores for > 4000 claims (Mody et al. 2026). While originally a bespoke implementation for repliCATS, we have since developed **aggreCAT** into a more general R package that can handle most types of elicitation data requiring mathematical aggregation, allowing for wider applicability.

**aggreCAT** provides a suite of 29 aggregation methods that explore different approaches to mathematical aggregation from straightforward arithmetic calculations to Bayesian statistical models. In addition, we provide extra functionality such as plotting functions and performance evaluation against known outcomes. While the repliCATS project uses the IDEA protocol to structure the elicitation process (Hemming et al. 2017), we have generalised **aggreCAT** to work with a variety of elicitation protocols. **aggreCAT** fills a large void in open software for aggregating elicited judgements, providing enormous benefit to researchers and decision makers across any field of research.

In this paper we give an overview of the **aggreCAT** package so that researchers may apply any of the the aggregation functions described in (Hanea et al. 2021), as well as additional methods provided by **aggreCAT**, to their own expert elicitation datasets where mathematical aggregation is required.

## 2 Mathematically aggregating expert judgements

The aggregation methods in **aggreCAT** were developed in the context of the IDEA protocol (Box 1), a structured elicitation procedure that draws on the ‘wisdom of crowds’ to elicit probabilistic judgements from groups of experts about uncertain events.

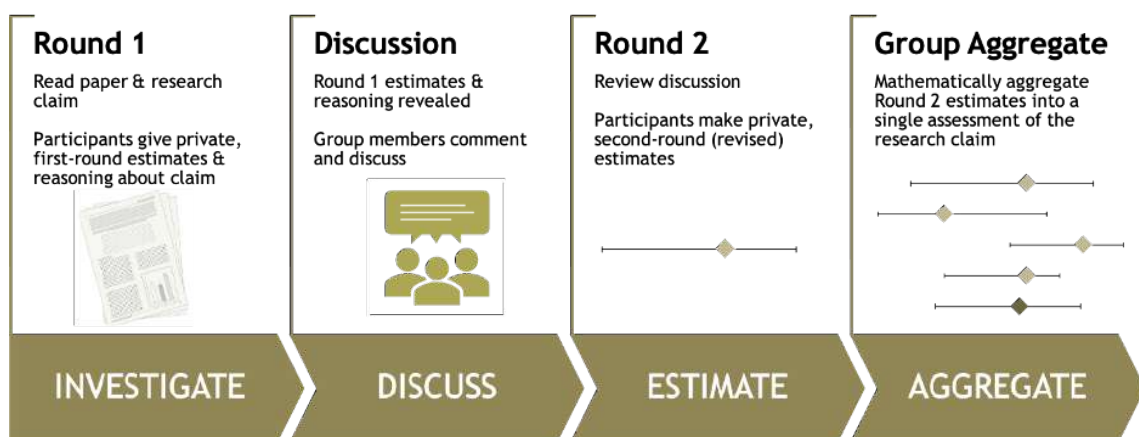
This elicitation structure directly determines which aggregation methods in **aggreCAT** are applicable to a given dataset: most methods require only the post-discussion (*Round 2*) single-point judgements, some additionally use the pre-discussion (*Round 1*) judgements, and a small number require supplementary ratings or externally collected data. These requirements are summarised for each method in Table 5.

Although the aggregation methods were developed in the context of the IDEA protocol, all methods requiring only a single round of elicitation can be applied to any structured elicitation procedure, including those that include multiple rounds of elicitation, or that elicit three-point estimates without enforcing behavioural consensus.

### Box 1: The repliCATS IDEA Protocol

The repliCATS project used the IDEA protocol in the context of forecasting the replicability of Social and Behavioural Science (SBS) claims (Fraser et al. 2023). Participants were asked to estimate the probability (expressed as percentages from 0–100%) that a direct replication would produce a statistically significant effect in the same direction as the original claim, providing a best estimate plus lower and upper bounds (Hemming et al. 2017, fig. 1). Participants also provided brief justifications and supplementary ratings (e.g., comprehensibility, plausibility, and involvement in the original study).

The IDEA protocol proceeds through four phases: *Investigate*, *Discuss*, *Estimate*, and *Aggregate* (Figure 1). Elicitation is conducted over two rounds: experts first provide independent judgements (*Round 1*, ‘Investigate’), then review anonymised peer estimates and ‘Discuss’ reasons for disagreement, and finally submit revised judgements (*Round 2*, ‘Estimate’). In the final *Aggregate* phase, these individual probabilistic judgements are mathematically combined into a single group forecast.



**Figure 1.** The IDEA protocol as deployed by the repliCATS project (reproduced with permission from Wintle et al. 2021).

## 2.1 Notation and problem formulation

First, we describe some preliminary mathematical notation used to represent each aggregation method. The total number of forecasts assessed,  $C$ , is indexed by  $c = 1, \dots, C$ . Note that in the example dataset provided with **aggreCAT**, each forecast corresponds to a unique research claim, indexed by `paper_id` (see Section 2.2.2). Every forecast  $c$  is assessed by  $N$  experts indexed by  $i = 1, \dots, N$ .

For each forecast,  $c$ , an individual,  $i$  assesses the probability of a some event occurring (e.g., a replication study finding a significant result in the same direction as the original claim) by providing up to either a single-point best estimate  $B_{i,c}$ , or three-point estimates. For three-point estimates, each expert provides three probabilities: a lower bound  $L_{i,c}$ , an upper bound  $U_{i,c}$ , and a best estimate  $B_{i,c}$ , satisfying the inequalities:  $0 \leq L_{i,c} \leq B_{i,c} \leq U_{i,c} \leq 1$ . These probabilities are then aggregated using one of the methods provided in **aggreCAT** to obtain a group or aggregate probability, denoted by  $\hat{p}_c$ . The aggregated probability calculated using a specific method is given by  $\hat{p}_c(\text{MethodID})$ . Each aggregation is assigned a unique *MethodID* which is the abbreviation of the mathematical operation used in calculating the weights.

Some aggregators employ weighting schemes that are informed by proxies for good forecasting performance whereby experts' estimates are weighted differently by aspects, such as; engagement, openness to changing their mind in light of new facts, extremity or informativeness of estimates, and by statistical knowledge as measured in a quiz.

Equation 1 defines standardised notation for describing weighted linear combinations of individual judgements where normalised weights are denoted by  $\tilde{w}_{method}$ :

$$\hat{p}_c(MethodID) = \frac{1}{N} \sum_{i=1}^N \tilde{w}_{method_{i,c}} B_{i,c} \quad (1)$$

The suite of aggregators include methods that weight judgements at the judgement-level, and methods that weight judgements at the participant-level. Judgement-level weights are calculated per participant per forecast, and therefore vary across forecasts for the same individual. Participant-level weights are calculated per participant, and therefore are identical across forecasts for the same participant. We provide the option for user-supplied weights at both levels. Some aggregation methods in **aggreCAT** rescale judgement-level weights by some participant-level measure across all forecasts, in which case a secondary index  $d = 1, \dots, D$  is used to denote the different forecasts that an individual has provided judgements for, where  $D$  is the total number of forecasts that an individual has provided judgements for.

For ease of debugging and applications across different aggregators, we make use of `weight_*` functions to calculate weights for methods that require weighting. These functions are called internally within the wrapper functions when needed, see Table 5 for details.

The performance of each method is evaluated by comparing the aggregated confidence scores  $\hat{p}_c$  to the known outcomes  $j$  using scoring rules, see Section 4.2. For each forecast  $c$ , with known outcome,  $j$ ,  $j = 1$  if the event occurred and  $j = 0$  if the event did not occur. Future work will include methods for evaluating performance when known outcomes are non-binary, e.g. RMSE for counts.

## 2.2 Aggregation wrapper functions

**aggreCAT** provides 29 aggregation methods in total, and these are grouped by type into eight wrapper functions. These functions, denoted by the `*WAgg` suffix (weighted aggregation), are: `AverageWAgg()` (calculations using measures of the average), `LinearWAgg()` (linear-weighted aggregation), `IntervalWAgg()` (weights calculated from uncertainty bounds), `ShiftingWAgg()` (weights calculated from estimate shifts between rounds), `DistributionWAgg()` (averaged probability distributions constructed from three-point elicitation), `ExtremisationWAgg()` (applies extremisation techniques to averaged estimates), `ReasoningWAgg()` (weights calculated from breadth of reasoning), and `BayesianWAgg()` (Bayesian aggregation methods). The specific aggregation *method* is within each wrapper function and is applied according to the type argument.

### 2.2.1 'Tidy' aggregation and required inputs

The design philosophy of **aggreCAT** is principled on 'tidy' data (Wickham 2014). Each aggregation method expects a `data.frame` or `tibble` of judgements as its input, and returns a `tibble` containing the variables `method`, `paper_id`, `cs` and `n_experts` (see Section 3.2 for illustration of outputs); where `method` is a character vector corresponding to the aggregation method name specified in the type argument. Each aggregation is applied as a summary function (Wickham and Grolemund 2017), and therefore returns a single row with a single confidence score `cs` for each claim or `paper_id`. The number of expert judgements summarised in the aggregated confidence score is returned in `n_experts`. Because of the tidy nature of the aggregation outputs, multiple aggregation methods can be applied to the same data with the results of all aggregation methods row-bound together in a single `tibble` (see the example `repliCATS` workflow in Section 4).

---

### 2.2.2 Package data

The **aggreCAT** package includes data collected by the repliCATS project during a pilot study at a two-day workshop in the Netherlands in July 2019, at which 25 participants assessed the replicability of 25 unique social and behavioural science research claims with known outcomes using the IDEA protocol (Wintle et al. 2023).

The dataset is distributed over eight data objects, each containing different types of data collected during the repliCATS project. The core data object `data_ratings` is a tidy `data.frame` of probabilistic three-point judgements (lower bound, best estimate, upper bound - as percentages) elicited from participants for each claim, across two rounds of elicitation. Additional data objects capture participants' written justifications (`data_justifications`), peer comments and ratings (`data_comments`), and three supplementary datasets collected externally to the IDEA protocol: a statistical knowledge quiz (`data_supp_quiz`), qualitatively coded reasoning categories (`data_supp_reasons`), and model-derived Bayesian priors (`data_supp_priors`).

Known replication outcomes sourced from previous large-scale replication projects (Klein et al. 2014, 2018; Ebersole et al. 2016; Camerer et al. 2018; Open Science Collaboration 2015) are provided in `data_outcomes`. And confidence scores calculated from 22 aggregation methods on the repliCATS dataset are provided in `data_confidence_scores`.

Full descriptions of each data object, are provided in the **aggreCAT** package documentation (`?data_ratings`) and in the 'Package Datasets' vignette available on the **aggreCAT** pkgdown website. For further details about the dataset, see Wintle et al. (2023).

Different aggregators require different minimum data inputs, including the type of judgements elicited (single- or three-point estimates), the number of elicitation rounds (one or two rounds), whether additional data is used to construct weights. For a full summary of each aggregation method, its `*WAgg` wrapper function and data requirements, see Table 5.

## 3 Demonstrating mathematical aggregation with aggreCAT

Here, we demonstrate the application of several aggregation methods in **aggreCAT** to a subset of the repliCATS dataset. We first describe the general workflow for applying aggregation methods in **aggreCAT** (Box 2), then demonstrate the application of several different aggregators to a single claim with judgements from five participants, highlighting the different data requirements and mechanics of each method, including; the type of judgements elicited (single- or three-point estimates), number of elicitation rounds (one or two rounds), whether judgements are weighted at the judgement- or participant- level, the type of mathematical operation used to combine judgements, and whether weights are calculated internally or require user-supplied data.

Here we demonstrate the application of aggregation methods for each group of methods using a set of 'focal claims' selected from the pilot study dataset supplied with the **aggreCAT** package.

### Box 2: Aggregation Workflow

All **aggreCAT** aggregation wrappers share five core arguments: `expert_judgements` (a tidy `data.frame` of expert judgements), `type` (the aggregation method applied to `expert_judgements`), `optional_name` (to override the default method label), `percent_toggle` (convert percentage inputs to probabilities), and `round_2_filter` (filter to round 2 estimates).

Some methods require additional data. For example, `ReasoningWAgg()` requires `reasons` (qualitatively coded reasoning categories), and `BayesianWAgg()` requires `priors` (claim-level prior information).

Each wrapper follows the same pipeline: pre-process judgements, compute weights when needed,

aggregate, then post-process results into tidy output. Pre-processing with `pre_process_judgements()` standardises inputs and filters to required rounds, elements, and questions. For example, methods requiring only round 2 estimates remove other rounds; methods requiring only best estimates retain only that element. Unless overridden by method-specific requirements or users, by default, single-round methods use round 2, single-point estimates use `three_point_best`, and three-point methods use `three_point_lower`, `three_point_upper` and `three_point_best`.

When weighting is required, un-normalised weights are computed either by dedicated helper functions or directly in the wrapper function for simpler methods (see Table 5). For instance, `ReasoningWAgg(type = "ReasonWAgg")` uses `weight_reason()`. Weights are then normalised within claim by dividing each expert's weight by the claim-level weight sum.

Finally, `post_process_results()` returns one row per claim with method name, confidence score, and number of aggregated experts.

### 3.1 Prepare focal claim data

Below we subset the dataset `data_ratings` to include a sample of four focal claims with judgements from five randomly-sampled participants. We select a single focal claim, `paper_id == "108"`, to demonstrate the application of different aggregation methods, and their unique data requirements (Table 1).

```
R> set.seed(1234)
R> focal_claims <- data_ratings |>
+   dplyr::filter(paper_id %in% c("24", "138", "186", "108"))
R> # select 5 users to highlight in focal claim demonstration
R> focal_users <- focal_claims |>
+   dplyr::distinct(user_name) |>
+   dplyr::slice_sample(n = 5)
R> # filter out non-focal users from focal claims
R> focal_claims <- focal_claims |>
+   dplyr::right_join(focal_users, by = "user_name") |>
+   dplyr::filter(stringr::str_detect(element, "three_point")) |>
+   dplyr::select(-question)
R> focal_claim_108 <- focal_claims |>
+   dplyr::filter(paper_id == "108")
R> focal_claims
```

# A tibble: 120 x 6

	round	paper_id	user_name	element	value	group
	<chr>	<chr>	<chr>	<chr>	<dbl>	<chr>
1	round_1	108	lvr6dwbqag	three_point_upper	90	UOM1
2	round_1	108	lvr6dwbqag	three_point_lower	40	UOM1
3	round_1	108	lvr6dwbqag	three_point_best	65	UOM1
4	round_1	108	f35wd3nhdz	three_point_upper	60	UOM3
5	round_1	108	f35wd3nhdz	three_point_lower	40	UOM3
6	round_1	108	f35wd3nhdz	three_point_best	51	UOM3
7	round_1	108	kf7fxabzu0	three_point_lower	70	UOM3
8	round_1	108	kf7fxabzu0	three_point_upper	90	UOM3
9	round_1	108	kf7fxabzu0	three_point_best	80	UOM3
10	round_1	108	g31jx5dffb	three_point_lower	70	UOM4

# i 110 more rows



**Table 1.** Example of expert judgement data for a single claim.

Claim ID	Participant	Lower Bound	Best Estimate	Upper Bound
108	g31jx5dffs	70	85	90
108	5nuvx01cj5	70	80	90
108	f35wd3nhdz	60	80	90
108	lvr6dwbqag	40	65	90
108	kf7fxabzu0	50	60	70

### 3.2 Unweighted linear combination of judgements

We first demonstrate the mechanics of mathematical aggregation and its implementation using the **aggreCAT** package with the simplest, unweighted aggregation wrapper, `AverageWAgg()`.

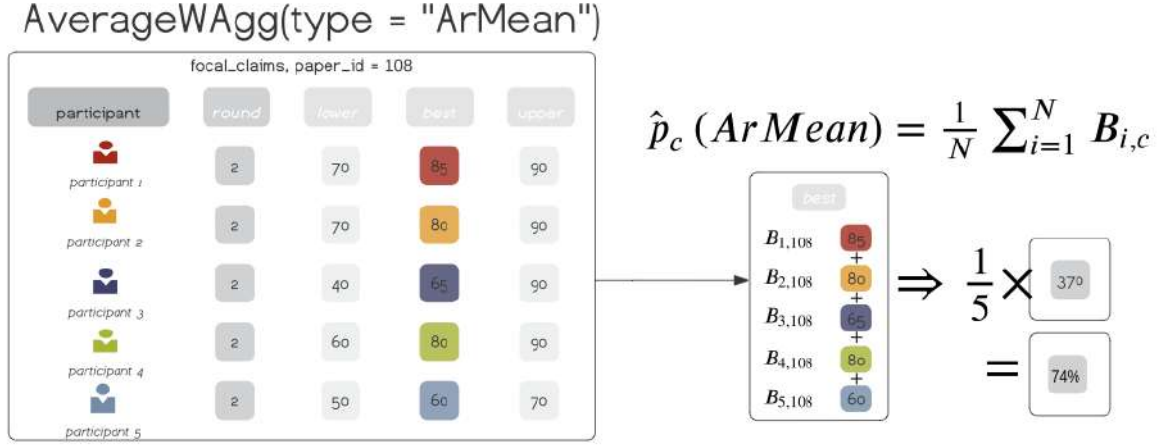
The `AverageWAgg()` wrapper function implements several methods that calculate different types of unweighted averaged best-estimates (see `?AverageWAgg`). The output is a tibble containing the method name, paper ID, confidence score, and number of experts whose judgements were aggregated. All other aggregation methods take this underlying computational blueprint, and expand on it according to the aggregation methods' requirements (See Box 2 for details).

Below we illustrate the mechanics of the arithmetic mean method 'ArMean', which takes the unweighted linear average of the best estimates,  $B_{i,c}$ , across all  $N$  experts for a given claim  $c$  to calculate the aggregated confidence score  $\hat{p}_c$  (Equation 2). The `type` argument is used across all aggregation wrapper functions to specify the method of aggregation. Below we apply `ArMean` to the focal claim 108 (Table 1), which has judgements from five participants. Note that the `AverageWAgg()` wrapper function implements several methods that calculate different types of unweighted averaged best-estimates (see `?AverageWAgg`), and in this case, we specify `type = "ArMean"` to apply the `ArMean` method.

$$\hat{p}_c (ArMean) = \frac{1}{N} \sum_{i=1}^N B_{i,c} \quad (2)$$

```
R> focal_claims |>
+   dplyr::filter(paper_id == "108") |>
+   AverageWAgg(type = "ArMean")
```

```
# A tibble: 1 x 4
  method paper_id    cs n_experts
  <chr>   <chr>    <dbl>    <int>
1 ArMean 108      74        5
```



**Figure 2.** ArMean with AverageWAgg() uses the Estimates (shown in colour) from each participant to compute the mean. We illustrate this using a single claim 108 for a subset of 5 out of 25 participants from the data\_ratings dataset. Note that the data representations in this figure are for explanatory purposes only, the data in the actual aggregation is tidy, with long form structure and format.

Other aggregation wrapper functions that do not weight judgements include: ExtremisationWAgg() DistributionWAgg() (see Table 5).

### 3.3 Weighted linear combination of Judgements

To improve the calibration, accuracy and informativeness of aggregated judgements, many aggregation methods in **aggreCAT** use weighting schemes to differentially weight individual judgements according to both measures and proxies of forecasting performance. In the absence of measures of experts' prior performance, proxies for forecasting performance, such as breadth and variability of qualitative reasons used by experts to justify their judgements (see Section 3.4.2), can be used to form weights (Hanea et al. 2021).

Weights may be calculated at the judgement-level or participant-level. Judgement-level weights are calculated per participant per claim, and therefore vary across claims for the same participant. Participant-level weights are calculated per participant, and therefore are identical across claims for the same participant.

#### 3.3.1 Judgement-level weights

IntervalWAgg uses judgement-level weights by aggregating linearly weighted best estimates with weights determined by the interval width of the forecast. The IntWAgg method implemented by the IntervalWAgg() function weights each participant's best estimate  $B_{i,c}$  by the width of their uncertainty intervals, i.e. the difference between an individual's upper  $U_{i,c}$  and lower bounds  $L_{i,c}$ . The intuition behind this weighting scheme is that participants who provide narrower uncertainty intervals are more confident in their estimate, and therefore likely to be more accurate for the target forecast. Hence, participants with narrower intervals are weighted more heavily in the aggregation than those with wider intervals.

For a given claim  $c$ , a vector of weights for all individuals is calculated from their upper and lower estimates using the weighting function, weight\_interval(), which calculates the interval width for each individual's estimate for the target forecast (Equation 3). The weights are then normalised across the claim (by dividing each weight by the sum of all weights per claim). Normalised weights are then multiplied by the corresponding individual's best estimates  $B_{i,c}$  and summed together into a single confidence score (Equation 4).



$$w_{Interval_{i,c}} = \frac{1}{U_{i,c} - L_{i,c}} \quad (3)$$

$$\hat{p}_c(IntWAgg) = \sum_{i=1}^N \tilde{w}_{Interval_{i,c}} B_{i,c} \quad (4)$$

### 3.3.2 Judgement-level weights rescaled by participant-level behaviour

To account for differences in interval widths across forecasts, judgement-level weights can be rescaled. In `IndIntWAgg` each participant's best estimate  $B_{i,c}$  is weighted by their interval width for forecast  $c$  relative to their average interval width across all forecasts. When the interval width for a given forecast is narrower than the participant's average interval width across forecasts, it implies greater confidence and potentially better forecasting performance.

`weight_nIndivInterval()` computes this rescaled weight (Equation 5). The weights are then normalised within each forecast, multiplied by  $B_{i,c}$ , and summed to give the aggregated confidence score (Equation 6).

$$w_{nIndivInterval_{i,c}} = \frac{1}{\frac{U_{i,c} - L_{i,c}}{\max((U_{i,d} - L_{i,d}) : d=1, \dots, C)}} \quad (5)$$

$$\hat{p}_c(IndIntWAgg) = \sum_{i=1}^N \tilde{w}_{nIndivInterval_{i,c}} B_{i,c} \quad (6)$$

The `IntervalWAgg()` wrapper function implements several methods that calculate different types of weighted linear combinations of best estimates with weights calculated from uncertainty bounds (see `?IntervalWAgg`). Let's apply both the `IntWAgg` and `IndIntWAgg` methods to the focal claim 108 (Table 1) using `IntervalWAgg()`, by specifying the `type` argument accordingly:

```
R> bind_rows(
+   focal_claim_108 |>
+   IntervalWAgg(type = "IntWAgg"),
+   focal_claim_108 |>
+   IntervalWAgg(type = "IndIntWAgg")
+ )
```

# A tibble: 2 x 4

	method	paper_id	cs	n_experts
	<chr>	<chr>	<dbl>	<int>
1	IntWAgg	108	74.8	5
2	IndIntWAgg	108	74	5

Other wrapper functions that weight estimates at the judgement-level include: `ShiftingWAgg()`, which weights estimates based on shifts in expert opinions between rounds when judgements are elicited using the IDEA protocol (or similar, Box 1); `ReasoningWAgg()`, which weights estimates by the breadth of reasoning provided in support of the estimate (see Section 3.4.2); and several methods implemented by the `LinearWAgg()` function, including `DistLimitWAgg`, which weights judgements based on the distance of estimates from the limits of the interval, `GranWAgg`, which calculates weights based on the granularity of estimates, and `OutWAgg`, which downweights outlier estimates.

### 3.3.3 Participant-level weights

VarIndIntWAgg uses participant-level weights by weighting each participant's best estimate  $B_{i,c}$  by the variability of their interval widths across forecasts relative to other participants. This method assumes that participants who show greater variability in their interval widths across forecasts are more responsive to the supporting evidence of different claims, and therefore more likely to be accurate forecasters.

$$w\_varIndivInterval_i = var(U_{i,c} - L_{i,c}) : c = 1, \dots, C \quad (7)$$

Where the variance `var` is calculated across claims for each participant, and the weights are then normalised across participants, multiplied by  $B_{i,c}$ , and summed to give the aggregated confidence score (Equation 8):

$$\hat{p}_c(VarIndIntWAgg) = \sum_{i=1}^N \tilde{w}_varIndivInterval_i B_{i,c} \quad (8)$$

```
R> focal_claims |>
+   IntervalWAgg(type = "VarIndIntWAgg")
```

```
# A tibble: 4 x 4
  method      paper_id    cs n_experts
  <chr>      <chr>    <dbl>    <int>
1 VarIndIntWAgg 108      74.6      5
2 VarIndIntWAgg 138      68.8      5
3 VarIndIntWAgg 186      57.5      5
4 VarIndIntWAgg 24       21.1      5
```

See `?IntervalWAgg` for other methods that calculate different types of weighted linear combinations of best estimates with weights calculated from uncertainty bounds, and Table 5 for a summary of all methods that use participant-level weights.

## 3.4 Weighted linear combination of judgements with user-supplied weights

**aggreCAT** provides several methods that utilise external data to construct weights for the aggregation of expert judgements. `LinearWAgg()` provides a flexible wrapper function for *applying* user-supplied weights to the best estimates  $B_{i,c}$  at either the judgement-level or participant-level. While, some aggregators utilise external data to *construct* weights internally within the wrapper functions, such as `ReasoningWAgg()` and `BayesianWAgg()` (see Section 3.4.2 and Section 3.5).

### 3.4.1 User-supplied weights

Using the `LinearWAgg()` wrapper function, any user-supplied weight can be applied at either the participant-level (`type = "Participant"`) or the judgement-level (`type = "Judgement"`). We show how to apply the `QuizWAgg` method described in (Hanea et al. 2021) to the focal claim with `LinearWAgg()`, which weights each participant's best estimate  $B_{i,c}$  by their score on a statistical knowledge quiz under the assumption that participants with higher statistical knowledge are more likely to be accurate forecasters, and therefore should be weighted more heavily in the aggregation than those with lower statistical knowledge.

`weights` are constructed from the `data_supp_quiz` dataset, which is a dataset of quiz scores for each participant in the repliCATS pilot study (See `?data_supp_quiz` for details about the dataset). The quiz dataset contains multiple score variables to choose as weights for the `QuizWAgg` method. We rename our chosen quiz score variable to `weight` to meet the requirements of the `LinearWAgg()`. The weights are then normalised across participants, multiplied by  $B_{i,c}$ , and summed to give the aggregated confidence score. To customise the name of the method returned in our output, we set the `name` argument to `QuizWAgg`:

```
focal_claim_108 |>
  LinearWAgg(
    type = "Participant",
    weights = data_supp_quiz |>
      rename(weight = quiz_score),
    name = "QuizWAgg"
  )
```

```
# A tibble: 1 x 4
  method  paper_id    cs n_experts
  <chr>    <chr>    <dbl>    <int>
1 QuizWAgg 108      73.4      5
```

In this way users can specify their own custom weights in addition to those described by Hanea et al. (2021) by preparing a dataset of weights and parsing this to `LinearWAgg()`'s `weights` argument. The dataset of weights must contain a column named `weight` containing the weight values, and a column that can be used to join the weights to the `expert_judgements` dataset by participant (e.g. `user_name`) or by judgement (e.g. `paper_id` and `user_name`).

### 3.4.2 Reasoning-based weights `ReasonWAgg()`

The `ReasoningWAgg()` function, which implements the `ReasonWAgg` method, requires a dataset of qualitatively coded reasons that experts provided to justify their numeric estimates. `ReasonWAgg` weights each participant's best estimate  $B_{i,c}$  by the number of unique reasoning categories they used to justify their estimates for the target forecast  $c$ , assuming that participants who use a greater breadth of reasoning categories to justify their estimates are more likely to be accurate forecasters.

The weighting function `weight_reason()` counts the number of unique reasons used by each participant for each claim, and uses this as a proxy for forecasting performance (Equation 9). Here,  $r = 1, \dots, R$  indexes the distinct reason categories available in the coded dataset, and  $R$  is the total number of unique reason categories that any participant could use in justifying their estimates. The more unique a participant uses to justify their estimates, the higher their weight in the aggregation. The weights are then normalised across the claim and multiplied by the corresponding individual's best estimates  $B_{i,c}$  and summed together into a single confidence score (Equation 10).

$$w_{reason_{i,c}} = \sum_{r=1}^R I(\text{reason}_{i,c} = r) \quad (9)$$

$$\hat{p}_c(\text{ReasonWAgg}) = \frac{1}{N} \sum_{i=1}^N \tilde{w}_{reason_{i,c}} B_{i,c} \quad (10)$$

Below we apply the `ReasonWAgg` method to the focal claim 108 using the `ReasoningWAgg()` wrapper function. First, we prepare the dataset of reasons by filtering `data_supp_reasons` to include only the focal users who assessed claim 108.

We reshape the output to illustrate the number of unique reasoning categories used by each participant to justify their estimates for claim 108 (Table 2). Each column corresponds to a unique reason category, and the value in each cell indicates whether a participant used that reason category (1) or not (0) to justify their estimates for claim 108. The total number of unique reasons used by each participant is then calculated by summing across the reason category columns.

Then we apply the `ReasonWAgg` method to compute a confidence score for claim 108 weighting participants' best estimates by the number of unique reasoning categories they used to justify their estimates for that claim.

**Table 2.** Reasoning data for focal claim 108 for a sample of reasoning categories. The dataset of reasons is used to construct weights for the ReasonWAgg method, which weights each participant’s best estimate by the number of unique reasoning categories they used to justify their estimates for the target forecast (see ?data\_supp\_reasons for details). Each column corresponds to a unique reason category, and the value in each cell indicates whether a participant used that reason category (1) or not (0). The total number of unique reasons used by each participant is calculated by summing across the reason category columns.

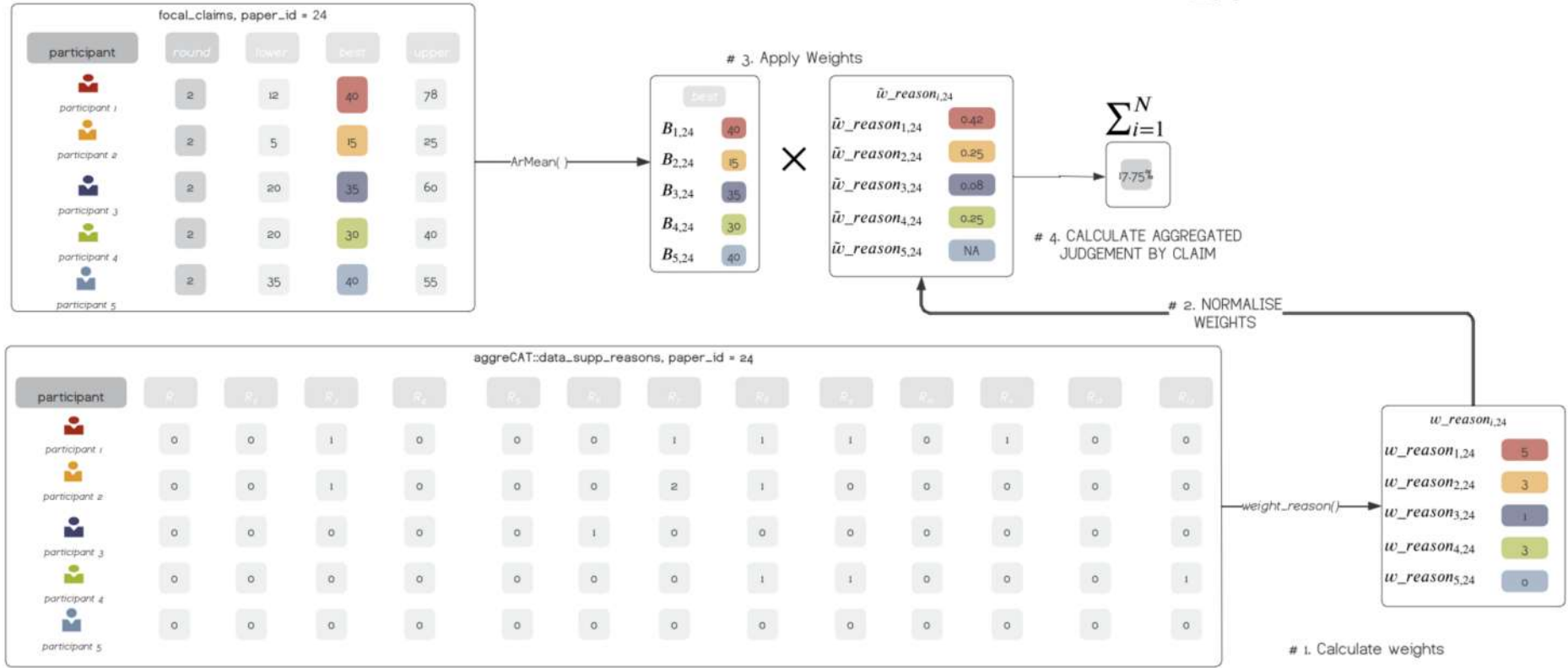
Participant	Plausibility	Population or subject characteristics (sampling practices)	Revision state-ments	Significance statisti-cal (p-value etc )
5nuvx01cj5	1	0	1	1
f35wd3nhdz	1	0	0	0
g31jx5dffb	1	0	0	0
kf7fxabzu0	1	0	0	0
lvr6dwbqag	1	1	0	0

```
R> focal_claims |>
+   dplyr::filter(paper_id == "108") |>
+   ReasoningWAgg(type = "ReasonWAgg", reasons = data_supp_reasons)
```

```
# A tibble: 1 x 4
  method    paper_id    cs n_experts
  <chr>      <chr>    <dbl>    <int>
1 ReasonWAgg 108      74.4      5
```

ReasoningWAgg(type = "ReasonWAgg" )

$$\hat{p}_c(\text{ReasonWAgg}) = \sum_{i=1}^N \tilde{w}_{\text{reason}_{i,c}} B_{i,c}$$



**Figure 3.** Illustration of the ReasonWAgg aggregation method for a subset of five participants who assessed claim 108. ReasonWAgg is applied using the wrapper function ReasoningWAgg() and exemplifies aggregation methods that use supplementary data (data\_supp\_ReasonWAgg) collected externally to the IDEA protocol in the construction of weights and subsequent calculation of confidence scores. Weights are constructed by taking the sum of the number of unique reasons made in support of quantitative estimates for each participant, for the target forecast.

---

### 3.5 Bayesian aggregation methods

While most aggregation methods in **aggreCAT** are weighted linear combinations of best estimates, the `BayesianWAgg()` aggregation family takes a model-based approach to computing confidence scores. It implements two variants that use elicited best estimates as data with which to update prior distributions of forecasts: `BayTriVar` and `BayPriorsAgg`. Both methods use the same underlying Bayesian model, but differ in how the prior distributions are specified. The `BayTriVar` method assumes forecast-specific prior means, while `BayPriorsAgg` allows users to specify their own priors.

The model in `BayesianWAgg` incorporates uncertainty from three sources: (i) *judgement-level variability*, based on the width of each participant's uncertainty interval for the target forecast; (ii) *participant-level variability*, defined as variation in each participant's best estimates across all forecasts they assessed; and (iii) *forecast-level variability*, defined as variation in best estimates across all participants for a given forecast. The confidence score is then taken as the median of the posterior distribution.

Below we apply the Bayesian Triple Variability method `BayTriVar` to the focal claim, which expects best estimates in the form of probabilities. We toggle the argument `percent_toggle` to `TRUE` to convert percentage inputs to probabilities in the data parsed to the `expert_judgements` argument. Although we compute the confidence score for a single claim only, the model computes participant-level weights using data from all forecasts within `expert_judgements`. Consequently we filter the aggregated confidence scores for claim 108 *after* applying the aggregation method to the full dataset of claims:

```
R> focal_claims |>
+   BayesianWAgg(type = "BayTriVar", percent_toggle = TRUE) |>
+   dplyr::filter(paper_id == "108")
```

```
# A tibble: 1 x 4
  method  paper_id    cs n_experts
  <chr>    <chr>    <dbl>   <int>
1 BayTriVar 108      0.699     5
```

Because `BayesianWAgg()` computes participant-level weights using data from all forecasts, the aggregated estimates are sensitive to the input data. This is also true for any aggregators that re-scale judgement-level weights with participant-level values, such as `IndIntWAgg` (see Section 3.3.1). As such, when calculating the confidence score for a single forecast, the user should ensure that the all available forecasts are parsed to `expert_judgements` before applying the aggregation method, and then filter the aggregated estimates for the target forecast after aggregation.

Below we illustrate this point by calculating the confidence score for claim 108 using `BayTriVar` with a different input dataset, `data_ratings`, which contains judgements for all claims in the `repliCATS` pilot study. We apply `BayTriVar` to the full dataset of claims, and then filter the aggregated confidence scores for claim 108 *after* applying the aggregation method to the full dataset of claims:

```
data_ratings |>
  BayesianWAgg(type = "BayTriVar", percent_toggle = TRUE) |>
  dplyr::filter(paper_id == "108")
```

```
# A tibble: 1 x 4
  method  paper_id    cs n_experts
  <chr>    <chr>    <dbl>   <int>
1 BayTriVar 108      0.748    25
```

Notice that the confidence score for claim 108 differs between the two applications of `BayTriVar`, depending on which input data was used to compute participant-level weights. This illustrates the importance of ensuring that all available forecasts are parsed to `expert_judgements` when applying aggregation methods that use participant-level weights, and then filtering the aggregated estimates for



---

the target forecast after aggregation.

## 4 An illustrative workflow for use in real study contexts

Throughout the SCORE program, 752 participants assessed more than 4000 unique claims using the repliCATS IDEA protocol, between 7th July 2019 and 25 November 2021 (Mody et al. 2026). This required batch aggregation over multiple claims, and to generate confidence scores for multiple claims. We also applied multiple aggregation methods to the same claim so that we could compare and evaluate the different aggregation methods. We expect that these are not uncommon use-cases, therefore we now demonstrate a general workflow for using the **aggreCAT** package to aggregate expert judgements using pilot data from DARPA SCORE program generated by the repliCATS project.

### 4.1 Generating multiple forecasts

The modular and tidy design of the aggregation functions in **aggreCAT** (see Box 2 and Section 2.2.1) supports batch aggregation of judgements across multiple forecasts using multiple methods so that the user is free to focus their attention on the interpretation and analysis of the forecasts, rather than on data processing and implementation of the aggregation methods. Below we apply the `ArMean` aggregation method to 25 claims evaluated by 25 participants simultaneously:

```
AverageWAgg(data_ratings, type = "ArMean")
```

```
# A tibble: 25 x 4
  method paper_id    cs n_experts
  <chr>   <chr>   <dbl>   <int>
1 ArMean 100     70.6     25
2 ArMean 102     30.8     25
3 ArMean 103     62.5     25
4 ArMean 104     47.1     25
5 ArMean 106     36.5     25
6 ArMean 108     71.8     25
7 ArMean 109     72.5     25
8 ArMean 116     62.6     25
9 ArMean 118     54.8     25
10 ArMean 133     59.9     25
# i 15 more rows
```

### 4.2 Comparing and evaluating aggregation methods

In real study contexts, such as that of the repliCATS project, it is of interest to compute confidence scores using multiple aggregation methods so that their performance might be evaluated and compared. Since different methods offer different mathematical properties, and therefore might be more or less appropriate depending on the purpose of the aggregation and forecasting, a researcher or analyst might want to check how the different assumptions embedded in different aggregation methods influence the final confidence scores for a forecast – i.e. how robust are the results to different methods and therefore to different assumptions?

From a computational perspective, multiple aggregation methods must first be applied to the forecast prior to comparison and evaluation. This can be achieved by applying each different aggregation method to `focal_claims`, and binding the results together with **dplyr**'s `bind_rows()`. However, more elegant and succinct solutions can be implemented using **purrr**'s `map_dfr()` function (Henry and Wickham (2020), see Listing A1 and Listing A2).

```
R> confidenceSCOREs <-
+   dplyr::bind_rows(
```

```

+   AverageWAgg(data_ratings,
+     "ArMean",
+     percent_toggle = TRUE
+   ),
+   IntervalWAgg(data_ratings,
+     "IndIntWAgg",
+     percent_toggle = TRUE
+   ),
+   IntervalWAgg(data_ratings,
+     "IntWAgg",
+     percent_toggle = TRUE
+   ),
+   ShiftingWAgg(data_ratings,
+     "ShiftWAgg",
+     percent_toggle = TRUE
+   ),
+   BayesianWAgg(data_ratings,
+     "BayTriVar",
+     percent_toggle = TRUE
+   ),
+   ReasoningWAgg(data_ratings,
+     reasons = data_supp_reasons,
+     percent_toggle = TRUE
+   )
+ )
R> confidenceSCOREs

```

```

# A tibble: 150 x 4
  method paper_id    cs n_experts
  <chr>   <chr>    <dbl>    <int>
1 ArMean 100      0.706      25
2 ArMean 102      0.308      25
3 ArMean 103      0.625      25
4 ArMean 104      0.471      25
5 ArMean 106      0.365      25
6 ArMean 108      0.718      25
7 ArMean 109      0.725      25
8 ArMean 116      0.626      25
9 ArMean 118      0.548      25
10 ArMean 133     0.599      25
# i 140 more rows

```

After generating confidence scores using various aggregation methods, we then evaluate the forecasts. We evaluated the repliCATS pilot study forecasts against the outcomes of previous, high-powered replication studies (Hanea et al. 2021), which are contained in the `data_outcomes` dataset published with **aggreCAT**. `data_outcomes` records the binary replication outcome (outcome: 1 if the claim successfully replicated, 0 if not) for each `paper_id`, sourced from previous large-scale replication projects (Klein et al. 2014, 2018; Ebersole et al. 2016; Camerer et al. 2018; Open Science Collaboration 2015):

```

R> data_outcomes |>
+   head()

```

```
# A tibble: 6 x 2
  paper_id outcome
  <chr>      <dbl>
1 100          1
2 102          0
3 103          0
4 104          1
5 106          0
6 108          1
```

The function `confidence_score_evaluation()` evaluates a set of aggregated forecasts or confidence scores against a set of known or observed outcomes, returning the Area Under the ROC Curve (AUC), the Brier score, and classification accuracy and correlation of each method (Table 3):

```
R> confidence_score_evaluation(
+   confidenceSCOREs,
+   data_outcomes
+ )
```

**Table 3.** AUC and Classification Accuracy for forecasts from the aggregation methods ShiftWAgg, ArMean, IntWAgg, IndIntWAgg, ReasonWAgg and BayTriVar for a subset of the repliCATS pilot study claims (`focal_claims`) and known outcomes.

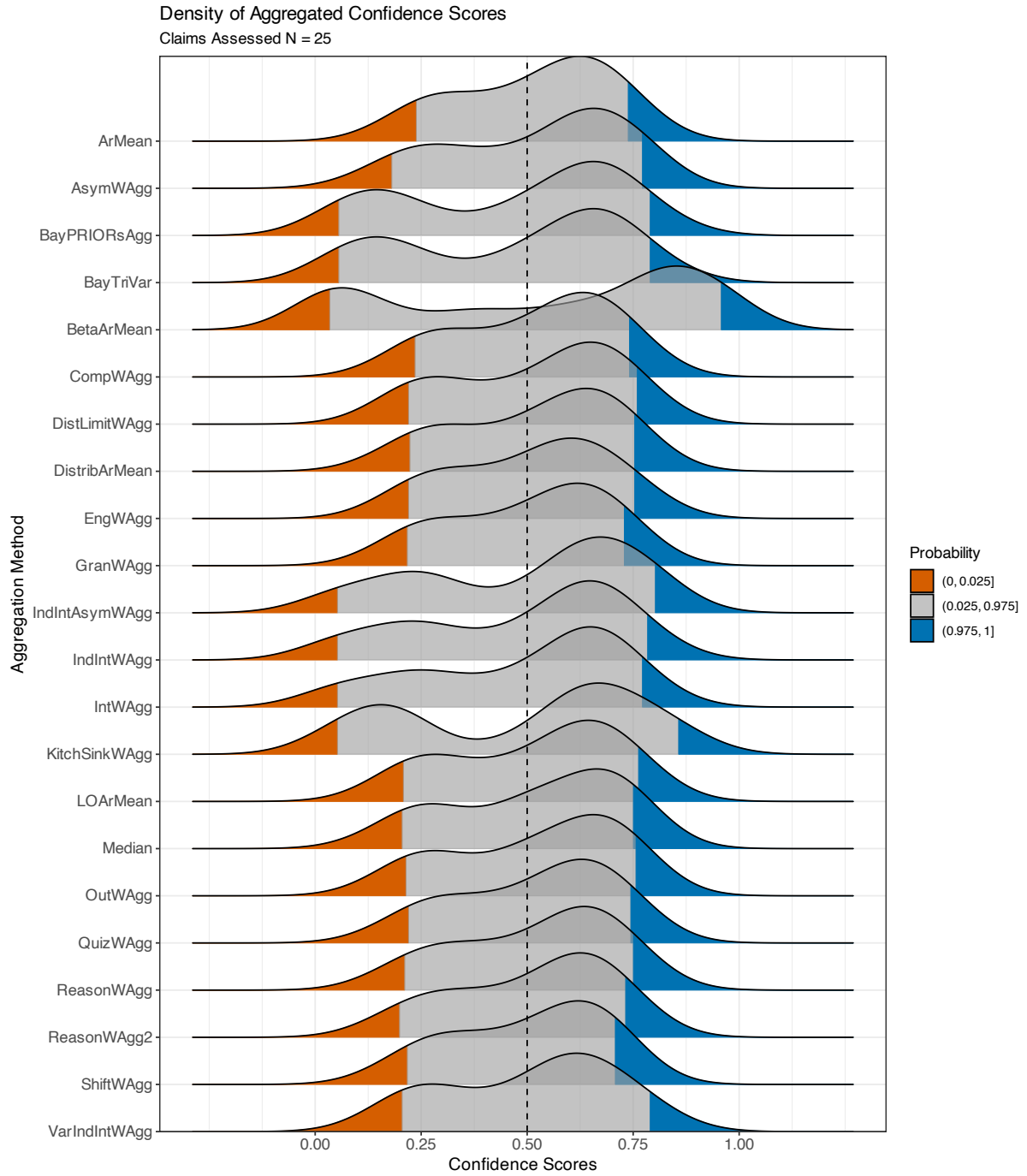
Method	AUC	Brier Score	Classification Accuracy	Correlation
ArMean	0.94	0.15	0.84	0.7472928
BayTriVar	0.87	0.14	0.80	0.7032517
IndIntWAgg	0.93	0.14	0.84	0.7222541
IntWAgg	0.93	0.14	0.84	0.7250046
ReasonWAgg	0.89	0.15	0.84	0.7191740
ShiftWAgg	0.96	0.15	0.88	0.7921471

### 4.3 Visualising judgements, confidence scores and forecast performance

We include three functions for visualising comparison and evaluation of confidence scores across multiple forecasts elicited from multiple experts using multiple aggregation methods.

Here, we use `confidence_score_ridgeplot()` to generate ridgeline plots using **ggbridges** (Wilke 2021). The plot displays the distribution of predicted outcomes across a collection of forecasts for each aggregation method, grouped into separate ‘mountain ranges’ according to the mathematical properties of the aggregation method (Figure 4).

```
library(ggbridges)
confidence_score_ridgeplot(confidence_scores = data_confidence_scores) +
  theme(
    axis.text = element_text(size = 11),
    axis.title = element_text(size = 12)
  )
```



**Figure 4.** Ridgeline plots illustrating the distribution of confidence scores for 22 aggregation methods on all 25 pilot data claims.

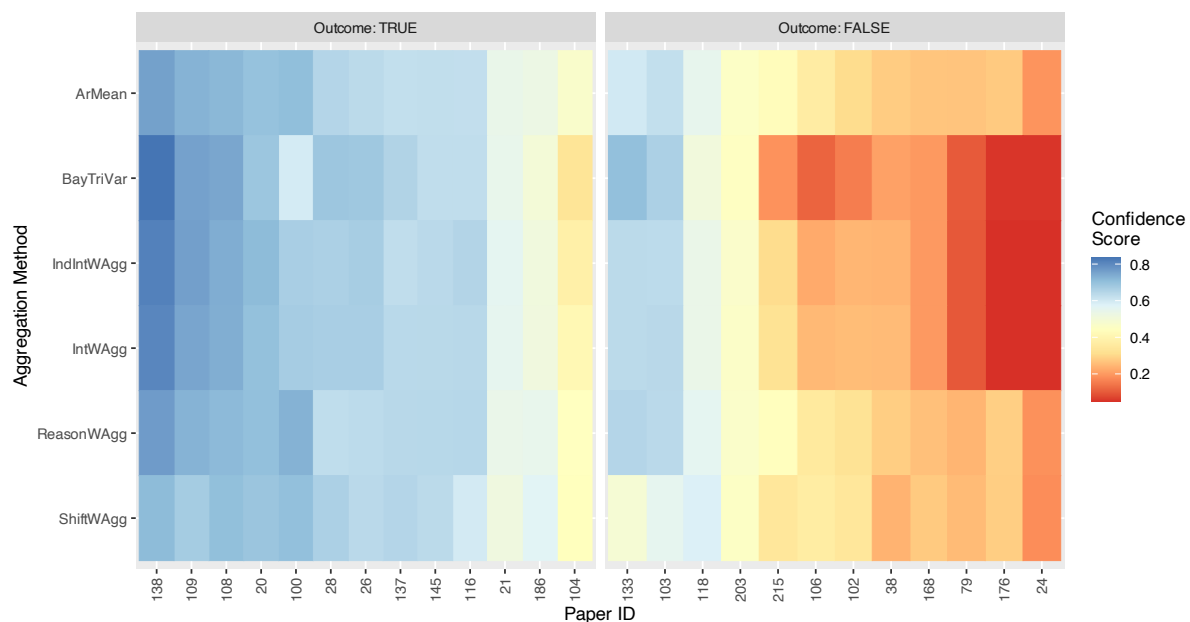
While `confidence_score_ridgeplot()` is useful for comparison of aggregated forecasts among methods, `confidence_score_heatmap()` facilitates comparative *evaluation* of the aggregation methods against a set of known outcomes. Next we use `confidence_score_heatmap()` to generate a blocked heatmap of confidence scores for each aggregation method for each claim in the replicATS pilot study, grouped horizontally according to the binary outcomes in `data_outcomes` (see Section 2.2.2) and vertically according to the mathematical characteristics of each aggregation method (Figure 5).

`confidence_score_heatmap()` provides a visual summary of the performance of different aggregation methods for different claims. The heatmap can be used to quickly identify which methods performed better than others for different claims, depending on the outcome, and for identifying

which methods might be more appropriate for different forecasting contexts. Under perfect forecasting we could expect a blocked heatmap in which the left block of claims with known outcomes of TRUE (i.e. successful replication) would be dominated by dark blue squares, indicating accurate forecasts of successful replication (confidence scores  $> 0.5$ ), and the right block of claims with known outcomes of FALSE (i.e. failed replication) would be dominated by dark red squares, indicating accurate forecasts of failed replication (confidence scores  $< 0.5$ ). Deviation from this expectation indicates which aggregation methods were inaccurate for a given forecast, and to what degree.

In the case of the repliCATS pilot study, the heatmap reveals that the majority of methods accurately forecasted the successful replication of most claims (Figure 5). The dominance of yellow/orange tiles on the right block of the heatmap indicates that most methods struggled to accurately forecast failed replication across most claims. For claims 176 and 24, which successfully replicated, IndIntWAgg and IntWAgg had confidence scores that were better calibrated to the observed outcomes, while all methods accurately predicted the outcome for claim 215, except for ReasonWAgg.

```
library(ggforce)
library(ggpubr)
confidence_score_heatmap(
  confidence_scores = confidenceSCOREs,
  data_outcomes = data_outcomes
)
```



**Figure 5.** Blocked heatmap of confidence scores for 25 claims generated by six different aggregation methods for the repliCATS pilot study. Claims where known outcomes successfully replicated (`outcome == TRUE`) are presented on the left block, and claims that did not replicate (`outcome == FALSE`) are presented on the right block. Confidence scores from different aggregation methods are grouped vertically according to the methods' mathematical properties. Colour and intensity of cells indicates the direction and degree of deviation of the confidence scores from the known outcomes, respectively. where the outcome was TRUE, dark blue cells (confidence score  $> 0.5$ ) indicate *accurate* forecasts of successful replication, and dark red cells (confidence score  $< 0.5$ ) indicate *inaccurate* forecasts of successful replication. The inverse is true for claims where the known outcome was FALSE, i.e. the right heatmap block.

#### 4.4 Extending aggreCAT to other datasets and problems

The modular, tidy design of the aggregation workflow in **aggreCAT** allows users to easily apply the aggregation methods to their own datasets and forecasting contexts, and to extend the package by

creating their own bespoke aggregation methods (e.g., Section 3.4.1) or by applying custom plots. Because the aggregation functions in **aggreCAT** return tidy data.frames and tibbles (Box 2), users can easily manipulate the raw judgements, aggregated confidence scores and outcome data to prepare them for subsequent analysis and visualisation. The modular design of the aggregation functions in **aggreCAT** allows users to leverage the pre- and post-processing functions within their own custom aggregation functions. The pre-processing function `preprocess_judgements()` can be used to prepare the data for aggregation, while the post-processing function `postprocess_judgements()` can be used to prepare the output of the aggregation for subsequent analysis and visualisation. In Listing A4 we illustrate how to use these functions to prepare the data for plotting with **ggplot2**.

The aggregation methods supplied by the **aggreCAT** package can easily be applied to various forecasting problems as long as the data inputs adhere to the required format. Depending on the elicitation format used to generate the forecasts, different aggregators may be more or less appropriate for use with the elicited judgements. We summarise the different aggregation methods and their requirements for use with different elicitation formats in Table 5, and we recommend that users consult the documentation for each aggregator to determine which method is most appropriate for their data and forecasting context.

#### 4.4.1 Preparing Elicitation Data for Aggregation

The wrapper functions for each method are designed to be flexible and user-friendly, allowing users to easily apply the methods to their own datasets with minimal data processing and manipulation. We demonstrate how to prepare data for applying the **aggreCAT** aggregation methods using data collected using the IDEA protocol for an environmental conservation problem (Arlidge et al. 2020). Participants were asked “How many green turtles in winter per month would be saved using a total gillnet ban, with gear switching to lobster potting or hand line fishing required?”. We take the data that will be parsed to the `expert_judgements` argument in the wrapper functions from Arlidge et al. (2020, Table S51), make the data long instead of wide, and then add the required columns `paper_id` and `question`:

```
R> green_turtles <-
+   dplyr::tribble(
+     ~user_name, ~round, ~three_point_lower,
+     ~three_point_upper, ~three_point_best,
+     "L01", 1, 10.00, 16.43, 10.00,
+     "L01", 2, 10.00, 16.43, 10.00,
+     "L02", 1, 500.00, 522.50, 500.00,
+     "L02", 2, 293.75, 406.25, 350.00,
+     "L03", 1, 400.00, 512.50, 400.00,
+     "L03", 2, 300.00, 356.25, 300.00,
+     "L04", 1, 32.29, 65.10, 41.67,
+     "L04", 2, 32.29, 65.10, 41.67,
+     "L05", 1, 6.67, 7.74, 6.67,
+     "L05", 2, 6.67, 7.74, 6.67
+   ) |>
+   dplyr::group_by(user_name) |> # pivot longer
+   tidyr::pivot_longer(
+     cols = tidyr::contains("three_point"),
+     names_to = "element", values_to = "value"
+   ) |>
+   dplyr::mutate(
+     paper_id = 1,
+     round = ifelse(round == 1, "round_1", "round_2"),
+     question = "direct_replication"
+   )
```



We can then apply multiple aggregation methods, using the same approach implemented for aggregation of the `focal_claims` dataset (Listing A3), with aggregated confidence scores for the green turtle dataset shown in Table 4. Note that some aggregators require probabilistic inputs, like `BaysianWAgg`, so are not applicable to the green turtle dataset, which contains judgements as point estimates rather than probabilities. In Listings A1 and A2 we illustrate how to apply several aggregation methods with different data input requirements to a single dataset simultaneously.

**Table 4.** Example aggregation of non-percentage / non-probabilistic estimates with several aggregation methods using Green Turtle dataset (Arlidge *et al.* 2020).

Method	Question ID	Confidence Score	N (experts)
ArMean	1	141.67	5
ShiftWAgg	1	328.85	5
IntWAgg	1	15.26	5
ShiftWAgg	1	328.85	5

## 5 Summary and discussion

The **aggreCAT** package provides a diverse suite of methods for mathematically aggregating judgements elicited from groups of experts using structured procedures such as the IDEA protocol. There are very few open-source tools for this purpose, and **aggreCAT** is distinctive both in the range of aggregation methods it implements—including methods that use proxies of forecasting accuracy via weights—and in its computational approach. To our knowledge, no other R package or other software offers such a broad collection of aggregation methods, including those that incorporate performance-based weights.

**aggreCAT** is designed for practical use in both one-off workshops and larger programs in which data collection is ongoing and aggregation needs to be automated. It follows tidy data principles: users supply `data.frames` of elicited judgements, and the aggregation functions return `data.frames` of aggregated forecasts. This has several advantages. First, data-wrangling and method application are handled internally, allowing researchers to focus on analysing and interpreting aggregation outputs—particularly important in data-deficient contexts where rapid expert assessments are needed. Second, because inputs and outputs are tidy, **aggreCAT** pairs naturally with other tidyverse tools such as **purrr**, **dplyr**, and **ggplot2**, as illustrated in the `repliCATS` workflow (Section 4). Third, this design scales readily to settings with many forecasts, many experts, and multiple aggregation methods, as demonstrated by its application in the `repliCATS` program to forecasts for more than 4000 research claims (Wilkinson *et al.* 2026; Mody *et al.* 2026).

The package also includes built-in tools for performance evaluation, enabling analysts to “ground-truth” forecasts against known outcomes or compare them with alternative forecasting approaches (Section 4.2). These tools compute standard accuracy metrics for confidence scores derived from different aggregation methods. The tidy design of the aggregation wrappers (Section 2.2.1) makes it straightforward to apply multiple methods in parallel and use the built-in performance evaluation tools to compare their accuracy against known outcomes, facilitating systematic comparison of methods and helping users to assess how well particular aggregation choices align with their forecasting goals.

A further strength of **aggreCAT** is its extensibility. Each aggregation function follows a consistent modular pattern, with most input and output wrangling handled by generic pre- and post-processing functions. This structure simplifies debugging, makes it easier to identify the source of errors, and lowers the barrier for users who wish to implement custom aggregation methods on top of the existing framework.

The aggregation methods implemented in **aggreCAT** can be applied to a variety of forecasting problems,

---

data types and elicitation protocols. We illustrated this flexibility in our application of **aggreCAT** to both probabilistic forecasts of replicability (repliCATS, Section 3) and to count data in a fisheries and conservation problem (Section 4.4). While **aggreCAT** includes specialised methods that exploit multi-round elicitation structures, such as those implemented in `ShiftingWAgg()`, the majority of methods require only a single round of single-point best-estimates. Thus, **aggreCAT** can accommodate a range of structured elicitation designs that yield individual estimates without enforcing consensus.

Currently, the package expects data inputs to follow nomenclature inherited from the repliCATS project. Future releases will relax these requirements to be more domain-agnostic, and we regard the current constraints as a minimal barrier to adoption. Similar naming conventions are already familiar to many users from other R packages. We have demonstrated that, despite this constraint, **aggreCAT** can already be extended and applied to domains beyond forecasting the replicability of research claims.

At the same time, there are important limitations and practical considerations. Different aggregation methods have specific data requirements: some require two rounds of elicitation (for example, methods that weight by shifts between rounds), some require full three-point judgements (lower, best, upper), and some require probabilistic inputs bounded between 0 and 1 and are therefore unsuitable for generic point estimates or unbounded quantities. A subset of methods also depends on supplementary data, such as coded reasoning categories or claim-level priors. Analysts must therefore align their choice of aggregation method with the design of their elicitation protocol and the data they have available. Where probabilistic judgements are not available, or where only a single point estimate is elicited, only a subset of the implemented methods are applicable, as illustrated by the green turtle example.

Currently, the package expects data inputs to follow nomenclature inherited from the repliCATS project. Future releases will relax these requirements to be more domain-agnostic, and we regard the current constraints as a minimal barrier to adoption. Similar naming conventions are already familiar to many users from other R packages (e.g. the **vegan** package, Oksanen et al. 2020). We have demonstrated that, despite this constraint, **aggreCAT** can already be extended and applied to domains beyond forecasting the replicability of research claims.

In this paper we have described the computational implementation of the aggregation methods and supporting tools in **aggreCAT**, and provided usage examples and workflows for both simple and complex research contexts. Our aim is to equip analysts to apply these aggregation functions to their own elicitation data. When users are unsure which aggregation method best suits their goals, they can consult existing methodological work on these aggregation methods (Hanea et al. 2021) for detailed discussion of their mathematical principles, underlying hypotheses, and comparative performance, and they can exploit **aggreCAT**'s built-in evaluation tools to assess performance in their own applications. Overall, **aggreCAT** is intended to support researchers and decision analysts in rapidly and rigorously analysing outcomes from the IDEA protocol and other structured elicitation procedures where mathematical aggregation of human forecasts is required.

**Table 5.** Summary of aggregation methods and functions, including data requirements and sources.

Method	Description	Data Requirements	Weighting Function	Min. Number Elicitation Rounds Required	Elicitation Method	Data Sources
<b>AverageWagg(): Averaged best estimates</b>						
ArMean	Arithmetic mean of the best estimates.		NA - Estimates are equally weighted	1	Single-point	$B_{i,c}$
Median	Median of the best estimates.		NA - Estimates are equally weighted	1	Single-point	$B_{i,c}$
GeoMean	Geometric mean of the best estimates.		NA - Estimates are equally weighted	1	Single-point	$B_{i,c}$
LOArMean	Arithmetic mean of the log odds transformed best estimates.		NA - Estimates are equally weighted prior to transformation	1	Single-point	$B_{i,c}$
ProbitArMean	Arithmetic mean of the probit transformed best estimates.		NA - Estimates are equally weighted prior to transformation	1	Single-point	$B_{i,c}$
<b>LinearWagg() Linearly-weighted best estimates</b>						
DistLimitWagg	Weighted by the distance of the best estimate from the closest certainty limit. Best-estimates closest to certainty limits are more strongly weighted.		Calculated internally	1	Single-point	$B_{i,c}$
GranWagg	Weighted by the granularity of best estimates.		Calculated internally	1	Single-point	$B_{i,c}$
Judgement	Weighted by user-supplied weights at the judgement level.	'data.frame/tibble' with three columns ('paper_id', 'user_name', 'weight')	Calculated by the user	1	Single-point	$B_{i,c}$
Participant	Weighted by user-supplied weights at the participant level.	'data.frame'/'tibble' with two columns ('user_name', 'weight')	Calculated by the user	1	Single-point	$B_{i,c}$
OutWagg	Outliers are down-weighted.		'weight_outlier()'	1	Single-point	$B_{i,c}$
<b>IntervalWagg() Linearly-weighted best estimates, with weights determined by interval widths</b>						
IntWagg	Weighted by interval width.		'weight_interval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$
IndIntWagg	Weighted by the re-scaled interval width (interval width relative to largest interval width provided by individual $i$ ).		'weight_nIndivInterval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}, U_{i,d}, L_{i,d}$
AsymWagg	Weighted by asymmetry of intervals.		'weight_asym()', 'weight_nIndivInterval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$
IndIntAsymWagg	Weighted by individuals' interval widths and their asymmetry.		'weight_asym()', 'weight_nIndivInterval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}, U_{i,d}, L_{i,d}$
VarIndIntWagg	Weighted by the variation in individuals' interval widths across estimates.		'weight_varIndivInterval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}, U_{i,d}, L_{i,d}$
KitchSinkWagg	Weighted by everything but the kitchen sink - rewards narrow and assymetric intervals as well as the variability of individuals' interval widths across estimates.		'weight_asym()', 'weight_nIndivInterval()', 'weight_varIndivInterval()'	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}, U_{i,d}, L_{i,d}$
<b>ShiftingWagg() Weighted by judgements that shift most after discussion</b>						
ShiftWagg	Accounts for shifts in individuals' best-estimates, upper and lower bounds between rounds.		Calculated internally	2	IDEA protocol or other structured protocol that generates multiple rounds of judgements using three-point elicitation	$B1_{i,c}, U1_{i,c}, L1_{i,c}, B1_{i,c}, U1_{i,c}, L1_{i,c}$
BestShiftWagg	Weights constructed from shifts in best-estimates.		Calculated internally	2	IDEA protocol or other structured protocol that generates multiple rounds of judgements of single point-estimates	$B_{i,c}$
IntShiftWagg	Weights constructed from shifts in interval widths.		Calculated internally	2	IDEA protocol or other structured protocol that generates multiple rounds of judgements using three-point elicitation	$B_{i,c}, U_{i,c}, L_{i,c}$

(continued)

Method	Description	Data Requirements	Weighting Function	Min. Number Elicitation Rounds Required	Elicitation Method	Data Sources
DistShiftWAgg	Weights constructed from degree of extrimisation shift between rounds.		Calculated internally	2	IDEA protocol or other structured protocol that generates multiple rounds of judgements of single point-estimates	$B_{i,c}$
DistIntShiftWAgg	Weights constructed by degree of interval narrowing and shift towards certainty bounds between rounds.		Calculated internally	2	IDEA protocol or other structured protocol that generates multiple rounds of judgements using three-point elicitation	$B_{i,c}, U_{i,c}, L_{i,c}$
<b>ReasoningWAgg()</b> Linearly-weighted best estimates, with weights constructed from supplementary reasoning data						
ReasonWAgg	Weighted by the breadth of reasoning (number of supplied reasons) provided to support the individuals' estimate.	'data_supp_ReasonWAgg'	'weight_reason()'	1	IDEA protocol or other structured protocol to elicit reasoning, but only sinle round (round 2) of data used in aggregation calculation.	$B_{i,c}, w_{reason_{i,c}}$
ReasonWAgg2	Weighted by the breadth of reasoning provided to support the individuals' estimate, rescaled by breadth of reasoning across all claims.	'data_supp_ReasonWAgg'	'weight_reason2()'	1	IDEA protocol or other structured protocol to elicit reasoning, but only sinle round (round 2) of data used in aggregation calculation.	$B_{i,c}, w_{reason_{i,c}}, U_{i,d}, L_{i,d}$
<b>ExtremisationWAgg()</b> Takes the average of best-estimates and transforms it using the cumulative distribution function of a beta distribution						
BetaArMean	Beta-transformed arithmetic mean of the best-estimates.		NA - Estimates are equally weighted	1	Single-point	$B_{i,c}$
BetaArMean2	Beta-transformed arithmetic mean of the best-estimates, but only to confidence scores outside a specified middle range.		NA - Estimates are equally weighted	1	Single-point	$B_{i,c}$
<b>DistributionWAgg()</b> Calculates the arithmetic mean of distributions created from expert judgements						
DistribArMean	Applies a non-parametric distribution evenly across upper, lower and best-estimates.		NA - Estimates are equally weighted	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$
TriDistribArMean	Applies a triangular distribution to the upper, lower and best-estimates.		NA - Estimates are equally weighted	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$
<b>BayesianWAgg()</b> Bayesian aggregation methods with either uninformative or informative prior distributions						
BayTriVar	Bayesian tripple variability method.		NA - Estimates are equally weighted	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$
BayPRIORsAgg	As per 'BayTriVar' but with priors derived from external predictive models, updated with individuals' best-estimates.	'data_supp_BayPRIORsAgg'	NA - Estimates are equally weighted	1	Three-point	$B_{i,c}, U_{i,c}, L_{i,c}$

---

## Acknowledgments

This project is sponsored by the Defense Advanced Research Projects Agency (DARPA) under cooperative agreement No.HR001118S0047. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

## References

- Alipourfard, Nazanin, Beatrix Arendt, Daniel M Benjamin, et al. 2021. *Systematizing Confidence in Open Research and Evidence (SCORE)*. SocArXiv. <https://doi.org/10.31235/osf.io/46mnb>.
- Arlidge, William N. S., Joanna Alfaro-Shigueto, Bruno Ibanez-Erquiaga, Jeffrey C. Mangel, Dale Squires, and Eleanor J. Milner-Gulland. 2020. "Evaluating Elicited Judgments of Turtle Captures for Data-limited Fisheries Management." *Conservation Science and Practice* 2 (5).
- Bache, Stefan Milton, and Hadley Wickham. 2020. *Magrittr: A Forward-Pipe Operator for r*. <https://CRAN.R-project.org/package=magrittr>.
- Camerer, CF, Anna Dreber, Felix Holzmeister, et al. 2018. "Evaluating the Replicability of Social Science Experiments in Nature and Science Between 2010 and 2015." *Naturecom*.
- Cooney, Philip, and Arthur White. 2023. "Direct Incorporation of Expert Opinion into Parametric Survival Models to Inform Survival Extrapolation." *Medical Decision Making*, 1–12. <https://journals.sagepub.com/doi/epub/10.1177/0272989X221150212>.
- Ebersole, Charles R., Olivia E. Atherton, Aimee L. Belanger, et al. 2016. "Many Labs 3: Evaluating Participant Pool Quality Across the Academic Semester via Replication." *Journal of Experimental Social Psychology* 67: 68–82. <https://doi.org/https://doi.org/10.1016/j.jesp.2015.10.012>.
- Fraser, Hannah, Martin Bush, Bonnie C. Wintle, et al. 2023. "Predicting Reliability Through Structured Expert Elicitation with the repliCATS (Collaborative Assessments for Trustworthy Science) Process." *PLOS One* 18: e0274429. <https://doi.org/10.1371/journal.pone.0274429>.
- Gaillard, Pierre, and Yannig Goude. 2026. *Opera: Online Prediction by Expert Aggregation*. <https://github.com/dralliag/opera>.
- Gao, Meixiang, Yanyan Ye, Ye Zheng, and Jiangshan Lai. 2025. "A Comprehensive Analysis of r's Application in Ecological Research from 2008 to 2023." *Journal of Plant Ecology* 18 (1): rtaf010. <https://doi.org/10.1093/jpe/rtaf010>.
- Goossens, L. H. J., R. M. Cooke, A. R. Hale, and L. J. Rodic-Wiersma. 2008. "Fifteen Years of Expert Judgement at TUDelft." *Safety Science* 46 (2): 234–44.
- Hanea, Anca, David P Wilkinson, Marissa McBride, et al. 2021. "Mathematically Aggregating Experts' Predictions of Possible Futures." *PLoS ONE* 16 (9). <https://doi.org/https://doi.org/10.1371/journal.pone.0256919>.
- Hemming, Victoria, Mark A. Burgman, Anca M. Hanea, Marissa F. McBride, and Bonnie C. Wintle. 2017. "A Practical Guide to Structured Expert Elicitation Using the IDEA Protocol." *Methods in Ecology and Evolution* 9 (1): 169–80. <https://doi.org/10.1111/2041-210x.12857>.

- 
- Henry, Lionel, and Hadley Wickham. 2020. *Purrr: Functional Programming Tools*. <https://CRAN.R-project.org/package=purrr>.
- Klein, Richard A., Kate A. Ratliff, Michelangelo Vianello, et al. 2014. "Investigating Variation in Replicability." *Social Psychology* 45 (3): 142–52.
- Klein, Richard A., Michelangelo Vianello, Fred Hasselman, et al. 2018. "Many Labs 2: Investigating Variation in Replicability Across Samples and Settings." *Advances in Methods and Practices in Psychological Science* 1 (4): 443–90. <https://doi.org/10.1177/2515245918810225>.
- Legge, Sarah, Libby Rumpff, John C. Z. Woinarski, et al. 2022. "The Conservation Impacts of Ecological Disturbance: Time-Bound Estimates of Population Loss and Recovery for Fauna Affected by the 2019–2020 Australian Megafires." *Global Ecology and Biogeography* 31 (10): 2085–104. <https://doi.org/10.1111/geb.13473>.
- Mody, Fallon, David Peter Wilkinson, Hannah Fraser, et al. 2026. *Large-Scale Human Predictions of the Replicability of Published Social and Behavioural Science Papers – a Multi- Study Analysis*. April. [https://doi.org/10.31222/osf.io/vgyed\\_v1](https://doi.org/10.31222/osf.io/vgyed_v1).
- Oakley, Jeremy. 2026. *SHELF: Tools to Support the Sheffield Elicitation Framework*. <https://doi.org/10.32614/CRAN.package.SHELF>.
- Oksanen, Jari, F. Guillaume Blanchet, Michael Friendly, et al. 2020. *Vegan: Community Ecology Package*. <https://CRAN.R-project.org/package=vegan>.
- Open Science Collaboration. 2015. "Estimating the Reproducibility of Psychological Science." *Science* 349 (6251): aac4716. <https://doi.org/10.1126/science.aac4716>.
- Sutherland, William J., Phoebe Barnard, Steven Broad, et al. 2017. "A 2017 Horizon Scan of Emerging Issues for Global Conservation and Biological Diversity." *Trends in Ecology and Evolution* 32 (1): 31–40. <https://doi.org/10.1016/j.tree.2016.11.005>.
- Wickham, H. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10).
- Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2021. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st ed. Paperback; O'Reilly Media. <https://r4ds.had.co.nz/>.
- Wilke, Claus O. 2021. *Ggbridges: Ridgeline Plots in "Ggplot2"*. <https://CRAN.R-project.org/package=ggbridges>.
- Wilkinson, David Peter, Fallon Mody, Mark Burgman, et al. 2026. *repliCATS-SCORE: Elicited Human Predictions of Social and Behavioural Science Replicability*. April. [https://doi.org/10.31222/osf.io/hj9ex\\_v1](https://doi.org/10.31222/osf.io/hj9ex_v1).
- Wintle, Bonnie C, Eden T Smith, Martin Bush, et al. 2023. "Predicting and Reasoning about Replicability Using Structured Groups." *Royal Society Open Science*, 1–24. <https://doi.org/10.1098/rsos.221553>.
- Wittmann, Marion E., Roger M. Cooke, John D. Rothlisberger, et al. 2015. "Use of Structured Expert



---

Judgment to Forecast Invasions by Bighead and Silver Carp in Lake Erie.” *Conservation Biology* 29 (1): 187–97. <https://doi.org/10.1111/cobi.12369>.

## A Appendix

### A.1 Computational details

The analyses and results in this paper were obtained using the following computing environment, versions of R and R packages:

```
- Session info -----
setting  value
version  R version 4.5.2 (2025-10-31)
os       macOS Tahoe 26.3.1
system   aarch64, darwin20
ui       X11
language (EN)
collate  en_US.UTF-8
ctype    en_US.UTF-8
tz       Australia/Melbourne
date     2026-05-03
pandoc   3.8.3 @ /Applications/Positron.app/Contents/Resources/app/quarto/bin/tools/aarch
quarto   1.9.37 @ /Applications/quarto/bin/quarto

- Packages -----
! package      * version      date (UTC) lib source
abind          1.4-8         2024-09-12 [1] CRAN (R 4.5.0)
VP aggregCAT   * 1.0.1.9000   2025-05-28 [?] CRAN (R 4.5.0) (on disk 1.0.0)
assertthat     0.2.1        2019-03-21 [1] CRAN (R 4.5.0)
backports     1.5.0         2024-05-23 [1] CRAN (R 4.5.0)
boot          1.3-32        2025-08-29 [1] CRAN (R 4.5.2)
brio          1.1.5         2024-04-24 [1] CRAN (R 4.5.0)
broom         1.0.12        2026-01-27 [1] CRAN (R 4.5.2)
cachem        1.1.0         2024-05-16 [1] CRAN (R 4.5.0)
car           3.1-5         2026-02-03 [1] CRAN (R 4.5.2)
carData       3.0-6         2026-01-30 [1] CRAN (R 4.5.2)
cellranger    1.1.0         2016-07-27 [1] CRAN (R 4.5.0)
class         7.3-23        2025-01-01 [1] CRAN (R 4.5.2)
cli           3.6.5         2025-04-23 [1] CRAN (R 4.5.0)
coda          0.19-4.1      2024-01-31 [1] CRAN (R 4.5.0)
crayon        1.5.3         2024-06-20 [1] CRAN (R 4.5.0)
data.table    1.18.2.1      2026-01-27 [1] CRAN (R 4.5.2)
desc          1.4.3         2023-12-10 [1] CRAN (R 4.5.0)
DescTools     0.99.60       2025-03-28 [1] CRAN (R 4.5.0)
devtools      2.4.6         2025-10-03 [1] CRAN (R 4.5.0)
digest        0.6.39        2025-11-19 [1] CRAN (R 4.5.2)
dplyr         * 1.2.0        2026-02-03 [1] CRAN (R 4.5.2)
e1071         1.7-17        2025-12-18 [1] CRAN (R 4.5.2)
ellipsis      0.3.2         2021-04-29 [1] CRAN (R 4.5.0)
evaluate      1.0.5         2025-08-27 [1] CRAN (R 4.5.0)
Exact         3.3           2024-07-21 [1] CRAN (R 4.5.0)
expm          1.0-0         2024-08-19 [1] CRAN (R 4.5.0)
farver        2.1.2         2024-05-13 [1] CRAN (R 4.5.0)
fastmap       1.2.0         2024-05-15 [1] CRAN (R 4.5.0)
forcats       * 1.0.1        2025-09-25 [1] CRAN (R 4.5.0)
```

Formula	1.2-5	2023-02-24	[1]	CRAN	(R 4.5.0)
fs	2.0.1	2026-03-24	[1]	CRAN	(R 4.5.2)
generics	0.1.4	2025-05-09	[1]	CRAN	(R 4.5.0)
ggforce	* 0.5.0	2025-06-18	[1]	CRAN	(R 4.5.0)
ggplot2	* 4.0.2	2026-02-03	[1]	CRAN	(R 4.5.2)
ggpubr	* 0.6.3	2026-02-24	[1]	CRAN	(R 4.5.2)
ggridges	* 0.5.7	2025-08-27	[1]	CRAN	(R 4.5.0)
ggsignif	0.6.4	2022-10-13	[1]	CRAN	(R 4.5.0)
gld	2.6.8	2025-09-14	[1]	CRAN	(R 4.5.0)
glue	1.8.0	2024-09-30	[1]	CRAN	(R 4.5.0)
GoFKernel	2.1-3	2024-12-06	[1]	CRAN	(R 4.5.0)
gtable	0.3.6	2024-10-25	[1]	CRAN	(R 4.5.0)
haven	2.5.5	2025-05-30	[1]	CRAN	(R 4.5.0)
here	1.0.2	2025-09-15	[1]	CRAN	(R 4.5.0)
hms	1.1.4	2025-10-17	[1]	CRAN	(R 4.5.0)
htmltools	0.5.9	2025-12-04	[1]	CRAN	(R 4.5.2)
httr	1.4.7	2023-08-15	[1]	CRAN	(R 4.5.0)
insight	1.4.6	2026-02-04	[1]	CRAN	(R 4.5.2)
jsonlite	2.0.0	2025-03-27	[1]	CRAN	(R 4.5.0)
kableExtra	* 1.4.0	2024-01-24	[1]	CRAN	(R 4.5.0)
KernSmooth	2.23-26	2025-01-01	[1]	CRAN	(R 4.5.2)
knitr	* 1.51	2025-12-20	[1]	CRAN	(R 4.5.2)
lattice	0.22-7	2025-04-02	[1]	CRAN	(R 4.5.2)
lifecycle	1.0.5	2026-01-08	[1]	CRAN	(R 4.5.2)
litedown	0.9	2025-12-18	[1]	CRAN	(R 4.5.2)
lmom	3.2	2024-09-30	[1]	CRAN	(R 4.5.0)
lubridate	* 1.9.5	2026-02-04	[1]	CRAN	(R 4.5.2)
magick	2.9.0	2025-09-08	[1]	CRAN	(R 4.5.0)
magrittr	2.0.4	2025-09-12	[1]	CRAN	(R 4.5.0)
MASS	7.3-65	2025-02-28	[1]	CRAN	(R 4.5.2)
mathjaxr	2.0-0	2025-12-01	[1]	CRAN	(R 4.5.2)
Matrix	1.7-4	2025-08-28	[1]	CRAN	(R 4.5.2)
memoise	2.0.1	2021-11-26	[1]	CRAN	(R 4.5.0)
MLmetrics	1.1.3	2024-04-13	[1]	CRAN	(R 4.5.0)
mvtnorm	1.3-5	2026-03-11	[1]	CRAN	(R 4.5.2)
otel	0.2.0	2025-08-29	[1]	CRAN	(R 4.5.0)
pillar	1.11.1	2025-09-17	[1]	CRAN	(R 4.5.0)
pkgbuild	1.4.8	2025-05-26	[1]	CRAN	(R 4.5.0)
pkgconfig	2.0.3	2019-09-22	[1]	CRAN	(R 4.5.0)
pkgload	1.5.0	2026-02-03	[1]	CRAN	(R 4.5.2)
png	0.1-8	2022-11-29	[1]	CRAN	(R 4.5.0)
polyclip	1.10-7	2024-07-23	[1]	CRAN	(R 4.5.0)
precrec	0.14.5	2025-05-15	[1]	CRAN	(R 4.5.0)
proxy	0.4-29	2025-12-29	[1]	CRAN	(R 4.5.2)
purrr	* 1.2.1	2026-01-09	[1]	CRAN	(R 4.5.2)
qdapRegex	0.7.10	2025-03-24	[1]	CRAN	(R 4.5.0)
R2jags	0.8-9	2024-10-13	[1]	CRAN	(R 4.5.0)
R2WinBUGS	2.1-24	2026-02-23	[1]	CRAN	(R 4.5.2)
R6	2.6.1	2025-02-15	[1]	CRAN	(R 4.5.0)
RColorBrewer	1.1-3	2022-04-03	[1]	CRAN	(R 4.5.0)

Rcpp	1.1.1	2026-01-10	[1]	CRAN	(R 4.5.2)
readr	* 2.1.6	2025-11-14	[1]	CRAN	(R 4.5.2)
readxl	1.4.5	2025-03-07	[1]	CRAN	(R 4.5.0)
remotes	2.5.0	2024-03-17	[1]	CRAN	(R 4.5.0)
rjags	4-17	2025-03-24	[1]	CRAN	(R 4.5.0)
rlang	1.1.7	2026-01-09	[1]	CRAN	(R 4.5.2)
rmarkdown	2.31	2026-03-26	[1]	CRAN	(R 4.5.2)
rootSolve	1.8.2.4	2023-09-21	[1]	CRAN	(R 4.5.0)
rprojroot	2.1.1	2025-08-26	[1]	CRAN	(R 4.5.0)
rstatix	0.7.3	2025-10-18	[1]	CRAN	(R 4.5.0)
rstudioapi	0.18.0	2026-01-16	[1]	CRAN	(R 4.5.2)
S7	0.2.1	2025-11-14	[1]	CRAN	(R 4.5.2)
scales	1.4.0	2025-04-24	[1]	CRAN	(R 4.5.0)
sessioninfo	1.2.3	2025-02-05	[1]	CRAN	(R 4.5.0)
stringi	1.8.7	2025-03-27	[1]	CRAN	(R 4.5.0)
stringr	* 1.6.0	2025-11-04	[1]	CRAN	(R 4.5.0)
svglite	2.2.2	2025-10-21	[1]	CRAN	(R 4.5.0)
systemfonts	1.3.1	2025-10-01	[1]	CRAN	(R 4.5.0)
testthat	* 3.3.2	2026-01-11	[1]	CRAN	(R 4.5.2)
textclean	0.9.7	2026-03-05	[1]	CRAN	(R 4.5.2)
textshaping	1.0.4	2025-10-10	[1]	CRAN	(R 4.5.0)
tibble	* 3.3.1	2026-01-11	[1]	CRAN	(R 4.5.2)
tidyr	* 1.3.2	2025-12-19	[1]	CRAN	(R 4.5.2)
tidyselect	1.2.1	2024-03-11	[1]	CRAN	(R 4.5.0)
tidyverse	* 2.0.0	2023-02-22	[1]	CRAN	(R 4.5.0)
timechange	0.4.0	2026-01-29	[1]	CRAN	(R 4.5.2)
tinytable	* 0.16.0	2026-02-11	[1]	CRAN	(R 4.5.2)
tinytex	* 0.59	2026-03-28	[1]	CRAN	(R 4.5.2)
tweenr	2.0.3	2024-02-26	[1]	CRAN	(R 4.5.0)
tzdb	0.5.0	2025-03-15	[1]	CRAN	(R 4.5.0)
usethis	3.2.1	2025-09-06	[1]	CRAN	(R 4.5.0)
utf8	1.2.6	2025-06-08	[1]	CRAN	(R 4.5.0)
vctrs	0.7.1	2026-01-23	[1]	CRAN	(R 4.5.2)
VGAM	1.1-14	2025-12-04	[1]	CRAN	(R 4.5.2)
viridisLite	0.4.3	2026-02-04	[1]	CRAN	(R 4.5.2)
withr	3.0.2	2024-10-28	[1]	CRAN	(R 4.5.0)
xfun	0.57	2026-03-20	[1]	CRAN	(R 4.5.2)
xml2	1.5.2	2026-01-17	[1]	CRAN	(R 4.5.2)
yaml	2.3.12	2025-12-10	[1]	CRAN	(R 4.5.2)

[1] /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library

\* -- Packages attached to the search path.  
V -- Loaded and on-disk version mismatch.  
P -- Loaded and on-disk path mismatch.

---

## A.2 Listings

**Listing A1** Multiple aggregation methods can be applied by binding rows rather than using the `purrr` package, if preferred.

```
purrr::map2_dfr(.x = list(AverageWAgg,
                          IntervalWAgg,
                          IntervalWAgg,
                          ShiftingWAgg,
                          BayesianWAgg),
               .y = list("ArMean",
                         "IndIntWAgg",
                         "IntWAgg",
                         "ShiftWAgg",
                         "BayTriVar"),
               .f = ~ .x(focal_claims,
                         type = .y,
                         percent_toggle = TRUE)
)
```

---

**Listing A2** Example of approach for applying a mixture of aggregation methods that do and do not require supplementary data. To batch aggregate claims using a combination of aggregation methods that do and do not require supplementary data, they must be aggregated separately, since the wrapper functions applying methods that require supplementary data have method-specific additional arguments. We can chain the aggregation of the methods that do not require supplementary data, and the methods that do require supplementary data together very neatly using `dplyr::bind_rows` function (Wickham et al. 2021) and the `magrittr` pipe `%>%` (Bache and Wickham 2020). We implement this approach while applying the aggregation methods `ArMean`, `IntWAgg`, `IndIntWAgg`, `ShiftWAgg` and `BayTriVar` to the `repliCATS` pilot program dataset `data_ratings`.

---

```
confidenceSCOREs <-
  list(
    AverageWAgg,
    IntervalWAgg,
    IntervalWAgg,
    ShiftingWAgg,
    BayesianWAgg
  ) %>%
  purrr::map2_dfr(
    .y = list("ArMean",
              "IndIntWAgg",
              "IntWAgg",
              "ShiftWAgg",
              "BayTriVar"),
    .f = ~ .x(data_ratings, type = .y, percent_toggle = TRUE)
  ) %>%
  dplyr::bind_rows(
    ReasoningWAgg(data_ratings,
                  reasons = data_supp_reasons,
                  percent_toggle = TRUE)
  )
```

---

**Listing A3** Bring your own data: non-probabilistic values

---

```
turtle_CS <-
  list(
    AverageWAgg,
    IntervalWAgg,
    IntervalWAgg,
    ShiftingWAgg
  ) %>%
  purrr::map2_dfr(.y = list("ArMean",
                            "IndIntWAgg",
                            "IntWAgg",
                            "ShiftWAgg"),
    .f = ~ .x(green_turtles, type = .y,
              percent_toggle = FALSE)
  )
```

---



---

**Listing A4** Visualising confidence scores

---

```
plot_cs <-
  confidenceSCOREs |>
  dplyr::left_join(data_outcomes) |>
  dplyr::mutate(data_type = "Confidence Scores") |>
  dplyr::rename(x_vals = cs,
                y_vals = method) |>
  dplyr::select(y_vals, paper_id, data_type, outcome, x_vals)

plot_judgements <-
  preprocess_judgements(focal_claims,
                        percent_toggle = TRUE) |>
  tidyr::pivot_wider(names_from = element,
                    values_from = value) |>
  dplyr::left_join(data_outcomes) |>
  dplyr::rename(x_vals = three_point_best,
                y_vals = user_name) |>
  dplyr::select(paper_id,
                y_vals,
                x_vals,
                tidyr::contains("three_point"),
                outcome) |>
  dplyr::mutate(data_type = "Elicited Probabilities")

p <- plot_judgements |>
  dplyr::bind_rows(., {dplyr::semi_join(plot_cs, plot_judgements,
                                       by = "paper_id")}) |>
  ggplot2::ggplot(ggplot2::aes(x = x_vals, y = y_vals)) +
  ggplot2::geom_pointrange(ggplot2::aes(xmin = three_point_lower,
                                       xmax = three_point_upper)) +
  ggplot2::facet_grid(data_type ~ paper_id, scales = "free_y") +
  ggplot2::theme_classic() +
  ggplot2::theme(legend.position = "none") +
  ggplot2::geom_vline(aes(xintercept = 0.5, colour = as.logical(outcome))) +
  ggplot2::xlab("Probability of Replication") +
  ggplot2::ylab(ggplot2::element_blank()) +
  ggplot2::scale_colour_brewer(palette = "Set1")
```

---