Package: dbreg (via r-universe)

May 21, 2026

Type Package

Title Fast Regressions on Database Backends

Version 0.0.3.99

Date 2026-03-16

Description Leverages database backends to run regressions on very large datasets, which may not fit into R's memory. Various acceleration strategies---e.g., Wong et al. (2021) <doi:10.48550/arXiv.2102.11297> and Arkhangelsky & Imbens (2024) <https://doi.org/10.1093/restud/rdad089>---allow for highly efficient computation, while robust standard errors are computed from sufficient statistics.

Imports DBI, duckdb, Formula, generics, glue, MASS, Matrix, stats, tinyplot

Suggests binsreg, here, fixest (>= 0.13.0), knitr, quarto, tinytest

Enhances dbplyr

Config/Needs/website etiennebacher/altdoc

Encoding UTF-8

License MIT + file LICENSE

URL https://grantmcdermott.com/dbreg/

BugReports https://github.com/grantmcdermott/dbreg/issues

VignetteBuilder quarto

Config/roxygen2/version 8.0.0

Config/pak/sysreqs xz-utils

Repository https://grantmcdermott.r-universe.dev

Date/Publication 2026-05-21 19:39:46 UTC

RemoteUrl https://github.com/grantmcdermott/dbreg

RemoteRef HEAD

RemoteSha 26ab3226aebb324a9f9d6943fafb6366d21e2042

Contents

`coef.dbreg` *Extract coefficients from dbreg objects*

Description

Extract coefficients from dbreg objects

Usage

```
## S3 method for class 'dbreg'
coef(object, fe = FALSE, ...)
```

Arguments

`object`	A ‘dbreg‘ object.
`fe`	Should the fixed effects be included? Default is ‘FALSE‘.
`...`	Additional arguments. Currently unused, except to capture superseded arguments.

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)

# coefficients
coef(mod)
coef(mod, fe = TRUE)  # include fixed effects

# confidence intervals
confint(mod)

# variance-covariance matrix
vcov(mod)
```

```
# predictions
head(predict(mod, newdata = airquality))
head(predict(mod, newdata = airquality, interval = "confidence"))
```

`confint.dbreg` *Confidence intervals for dbreg objects*

Description

Confidence intervals for dbreg objects

Usage

```
## S3 method for class 'dbreg'
confint(object, parm, level = 0.95, fe = FALSE, ...)
```

Arguments

`object`	A ‘dbreg‘ object.
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	the confidence level required. Default is 0.95.
`fe`	Should the fixed effects be included? Default is ‘FALSE‘.
`...`	Additional arguments. Currently unused, except to capture superseded arguments.

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)

# coefficients
coef(mod)
coef(mod, fe = TRUE)  # include fixed effects

# confidence intervals
confint(mod)

# variance-covariance matrix
vcov(mod)

# predictions
head(predict(mod, newdata = airquality))
head(predict(mod, newdata = airquality, interval = "confidence"))
```

dbbinsreg *Run a binscatter regression on a database backend and plot the result*

Description

Performs binned regression entirely in SQL, returning plot-ready data with estimated bin means or piecewise polynomial fits. The API is designed to be compatible with the **binsreg** package by Cattaneo, Crump, Farrell, and Feng (2024). Supports unconditional and conditional models (with controls and/or fixed effects).

Usage

```
dbbinsreg(
  fml,
  conn = NULL,
  table = NULL,
  data = NULL,
  path = NULL,
  points = c(0, 0),
  line = NULL,
  linegrid = 20,
  nbins = 20,
  binspos = "qs",
  randcut = NULL,
  sample_fit = NULL,
  ci = TRUE,
  cb = FALSE,
  vcov = NULL,
  level = 0.95,
  nsims = 500,
  strategy = c("auto", "compress"),
  plot = TRUE,
  verbose = getOption("dbreg.verbose", FALSE),
  dots = NULL,
  ...
)
```

Arguments

`fml` A `formula` representing the binscatter relation. The first variable on the RHS is the running variable; additional variables are controls. Fixed effects go after `|`. Examples:

- `y ~ x`: simple binscatter
- `y ~ x + w1 + w2`: binscatter with controls
- `y ~ x | fe`: binscatter with fixed effects
- `y ~ x + w1 + w2 | fe`: binscatter with controls and fixed effects

conn — Database connection, e.g. created with dbConnect. Can be either persistent (disk-backed) or ephemeral (in-memory). If no connection is provided, then an ephemeral duckdb connection will be created automatically and closed before the function exits. Note that a persistent (disk-backed) database connection is required for larger-than-RAM datasets in order to take advantage of out-of-core functionality like streaming (where supported).

table, data, path — Mutually exclusive arguments for specifying the data table (object) to be queried. In order of precedence:

- table: Character string giving the name of the data table in an existing (open) database connection.
- data: R dataframe that can be copied over to conn as a temporary table for querying via the DuckDB query engine. Ignored if table is provided.
- path: Character string giving a path to the data file(s) on disk, which will be read into conn. Internally, this string is passed to the FROM query statement, so could (should) include file globbing for Hive-partitioned datasets, e.g. "mydata/**/.*parquet". For more precision, however, it is recommended to pass the desired database reader function as part of this string, e.g. "read_parquet('mydata/**/*.parquet')" for DuckDB; note the use of single quotes. Ignored if either table or data is provided.

points — A vector c(p, s) specifying the polynomial degree p and smoothness s for the points (point estimates at bin means). Default is c(0, 0) for canonical binscatter (bin means). Set to NULL or FALSE to suppress points. The smoothness s must satisfy s <= p.

line — A vector c(p, s) specifying the polynomial degree p and smoothness s for the line (evaluated on a grid within bins). Default is NULL (no line). Set to TRUE for c(0, 0) or a vector like c(1, 1) for piecewise linear with continuity constraints. The smoothness s must satisfy s <= p.

linegrid — Number of evaluation points per bin for the line. Default is 20.

nbins — Integer number of bins. Default is 20.

binspos — Bin positioning method. One of either "qs" (quantile-spaced, equal-count bins, the default), "es" (evenly-spaced, equal-width bins), or a numeric vector of knot positions for manual specification.

randcut — Numeric in the range (0,1]. Controls the random sampling fraction for bin boundary computation on large datasets. If NULL (the default), then sampling is automatic: 0.01 (1%) for datasets exceeding 1 million rows and 1 (100%) otherwise. Note that sampling is only used for computing the bin boundaries, since this requires an expensive ranking operation. The subsequent, primary regression operations use all of the data (unless sample_fit is enabled).

sample_fit — Logical or NULL. Controls whether the spline regression (s > 0) re-uses the same random sample (controlled by randcut) that is used for computing the bin boundaries. This trades off some precision for major speed gains on big datasets; see the Smoothness Constraints section below. If NULL (the default), sampling is enabled automatically when applicable, with a message. Explicitly set to TRUE to enable the same sampling behaviour, but without the message. Alternatively, set to FALSE to use the full dataset. Ignored when s = 0, since the "compress" strategy already handles these aggregation cases efficiently.

ci	Logical. Calculate standard errors and confidence intervals for points? Default is TRUE.
cb	Logical. Calculate simultaneous confidence bands using simulation? Default is FALSE.
vcov	Character string or formula for standard errors. Options are "HC1" (default, heteroskedasticity-robust, matches binsreg), "iid", or a one-sided formula for clustered standard errors (e.g., ~cluster_var).
level	Numeric in the range [0,1], giving the confidence level for the confidence levels and/or bands. Default is 0.95.
nsims	Number of simulation draws for confidence band computation. Default is 500. Only used when cb = TRUE.
strategy	Acceleration strategy passed to dbreg. Only "compress" is currently supported; "auto" (the default) maps to "compress". Included for API consistency with dbreg. Ignored when smoothness s > 0, since spline basis construction requires row-level data (i.e., no pre-aggregation).
plot	Logical. If TRUE (default), a plot is produced as a side effect. Set to FALSE to suppress plotting.
verbose	Logical. Print auto strategy and progress messages to the console? Defaults to FALSE. This can be overridden for a single call by supplying verbose = TRUE, or set globally via options(dbreg.verbose = TRUE).
dots	Alias for points for binsreg compatibility. If not NULL, overrides the points argument.
...	Additional arguments passed to plot.dbbinsreg, which may in turn be passed to tinyplot.

Value

A list of class "dbbinsreg" containing:

points Data frame of the point estimates (one row per bin): x (bin mean), bin, and fit (fitted value). If ci=TRUE in the original call, then also includes the columns: se, lwr, and upr. Similarly, if cb=TRUE, then includes the columns: cb_lwr and cb_upr.

line Data frame of the line estimates (multiple rows per bin): x, bin, fit. Only present if line is specified.

bins Data frame with bin boundaries: id (bin number), left (left endpoint), right (right endpoint).

model The fitted dbreg model object (for points).

opt List of options used: points, line, nbins, binspos, etc.

If plot = TRUE (the default), a binscatter plot is also produced as a side effect. See plot.dbbinsreg for plot customization.

Comparison with binsreg

The dbbinsreg function is deeply inspired by the **binsreg** package (Cattaneo et al., 2024). The main difference is that dbbinsreg performs most of its computation on a database backend, employing various acceleration strategies, which makes it particularly suitable for large datasets (which may not fit in memory). At the same time, the database backend introduces its own set of tradeoffs. We cover the most important points of similarity and difference below.

Core API and bin selection:

We aim to mimic the binsreg API as much as possible. Key parameter mappings include:

- points (alias dots): Point estimates at bin means
  - c(0,0): Canonical binscatter (bin means)
  - c(p,0): Piecewise polynomial of degree p, no smoothness
  - c(p,s): Piecewise polynomial with s smoothness constraints
- line: Same as points but evaluated on a finer grid for smooth visualization
- binspos: Bin positioning
  - "qs": Quantile-spaced (equal count)
  - "es": Evenly-spaced (equal width)

Important: Unlike binsreg, dbbinsreg does not automatically select the IMSE-optimal number of bins. Rather, users must specify nbins manually (with a default of value of 20). For guidance on bin selection, see binsregselect or Cattaneo et al. (2024).

Smoothness constraints:

When s > 0, the function fits a regression spline using a truncated power basis. For degree p and smoothness s, the basis includes global polynomial terms $(x, x^2, \ldots, x^p)$ plus truncated power terms $(x - \kappa_j)_+^r$ at each interior knot κ_j for $r = s, \ldots, p$. This enforces C^{s-1} continuity (continuous derivatives up to order $s - 1$) at bin boundaries. For example, c(1,1) gives a piecewise linear fit that is continuous; c(2,2) gives a piecewise quadratic with continuous first derivatives.

Important: Unlike s = 0 (which uses the "compress" strategy for fast aggregation), s > 0 requires row-level spline basis construction and can be very slow on large datasets. As a result, dbbinsreg re-uses the random sample (used to compute the bin boundaries) for estimating the spline fits in these cases, ensuring much faster computation at the cost of reduced precision. Users can override this behaviour by passing the sample_fit = FALSE argument to rather estimate the spline regressions on the full dataset.

Confidence intervals vs confidence bands:

When ci = TRUE (default), pointwise confidence intervals (CIs) are computed at each bin mean using standard asymptotic theory. When cb = TRUE, simultaneous confidence bands (CBs) are computed using a simulation-based sup-t procedure:

1. Draw nsims samples from the asymptotic distribution of the estimator
2. Compute the supremum of the t-statistics across all bins for each draw
3. Use the $(1 - \alpha)$ quantile of these suprema as the critical value

The confidence band is wider than pointwise CIs and provides simultaneous coverage: with $(1-\alpha)$ probability, the entire true function lies within the band. This is useful for making statements about the overall shape of the relationship rather than individual point estimates.

There are two important caveats, regarding dbbinsreg's CB support:

- Unlike binsreg, which evaluates CB on a fine grid within each bin, dbbinsreg computes CB only at bin means (same points as CI). This is much simpler for our backend SQL implementation and should be sufficient for most applications.
- CBs are currently only supported for unconstrained estimation (smoothness s = 0). When cb = TRUE with s > 0, a warning is issued and CB is skipped.

Note on quantile bin boundaries:

When using quantile-spaced bins (binspos = "qs"), dbbinsreg uses SQL's NTILE() window function, while binsreg uses R's quantile with type = 2. These algorithms have slightly different tie-breaking behavior, which can cause small differences in bin assignments at boundaries. In practice, differences are typically <1% and become negligible with larger datasets. To match binsreg exactly, compute quantile breaks on a subset of data in R and pass them via the binspos argument as a numeric vector.

References

Cattaneo, M. D., R. K. Crump, M. H. Farrell, and Y. Feng (2024). On Binscatter. *American Economic Review*, 114(5): 1488-1514.

See Also

plot.dbbinsreg for plot customization, dbreg for the underlying regression engine, binsreg for the original implementation.

Examples

```
#
## In-memory data ----

# Like `dbreg`, we can pass in-memory R data frames to an ephemeral DuckDB
# connection via the `data` argument.

# Canonical binscatter: bin means (default)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10)

# You can pass additional plotting arguments via ... to (tiny)plot.dbbinsreg
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10,
          main = "A simple binscatter example", theme = "clean")

# Alternatively, save the object and plot later
bs = dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, plot = FALSE)
plot(bs, main = "Same example, different theme", theme = "classic")

# Piecewise linear (p = 1), no smoothness (s = 0)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 0))

# Piecewise linear (p = 1) with continuity (s = 1)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 1))

# With line overlay for smooth visualization
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 1), line = TRUE)
```

```
# Different line smoothness to points
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(0, 0), line = c(1, 1))

# With uniform confidence bands (much greater uncertainty)
set.seed(99)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, cb = TRUE)

# Accounting for Diet "fixed effects" helps to resolve the situation
# (We'll also add a line and change the theme for a nicer plot)
dbbinsreg(weight ~ Time | Diet, data = ChickWeight, nbins = 10, cb = TRUE,
          line = c(1, 1), theme = "clean")

#
## DBI connection ----

library(DBI)
con = dbConnect(duckdb::duckdb())
dbWriteTable(con, "cw", as.data.frame(ChickWeight))

dbbinsreg(weight ~ Time | Diet, conn = con, table = "cw", nbins = 10,
          theme = "clean")
# etc.

# See ?dbreg for more connection examples

# Clean up
dbDisconnect(con)
```

---

dbreg *Run a regression on a database backend*

Description

Leverages the power of databases to run regressions on very large datasets, which may not fit into R's memory. Various acceleration strategies allow for highly efficient computation, while robust standard errors are computed from sufficient statistics.

Usage

```
dbreg(
  fml,
  conn = NULL,
  table = NULL,
  data = NULL,
  path = NULL,
  weights = NULL,
  vcov = c("iid", "hc1"),
```

```
  strategy = c("auto", "compress", "moments", "demean", "within", "mundlak"),
  compress_ratio = NULL,
  compress_nmax = 1e+06,
  cluster = NULL,
  ssc = c("full", "nested"),
  sql_only = FALSE,
  data_only = FALSE,
  drop_missings = TRUE,
  verbose = getOption("dbreg.verbose", FALSE),
  ...
)
```

Arguments

`fml`	A `formula` representing the relation to be estimated. Fixed effects should be included after a pipe, e.g `fml = y ~ x1 + x2 | fe1 + f2`. Interaction terms are supported using standard R syntax (`:` for interactions, `*` for main effects plus interaction). Transformations and literals are not yet supported.
`conn`	Database connection, e.g. created with `dbConnect`. Can be either persistent (disk-backed) or ephemeral (in-memory). If no connection is provided, then an ephemeral `duckdb` connection will be created automatically and closed before the function exits. Note that a persistent (disk-backed) database connection is required for larger-than-RAM datasets in order to take advantage of out-of-core functionality like streaming (where supported).
`table, data, path`	Mutually exclusive arguments for specifying the data table (object) to be queried. In order of precedence: • `table`: Character string giving the name of the data table in an existing (open) database connection. • `data`: R dataframe that can be copied over to `conn` as a temporary table for querying via the DuckDB query engine. Ignored if `table` is provided. • `path`: Character string giving a path to the data file(s) on disk, which will be read into `conn`. Internally, this string is passed to the `FROM` query statement, so could (should) include file globbing for Hive-partitioned datasets, e.g. `"mydata/**/.*parquet"`. For more precision, however, it is recommended to pass the desired database reader function as part of this string, e.g. `"read_parquet('mydata/**/*.parquet')"` for DuckDB; note the use of single quotes. Ignored if either `table` or `data` is provided.
`weights`	Character string specifying the column name to use as weights, or NULL (default) for unweighted regression. Weights must be non-negative; rows with zero weight are dropped. Weighted regressions support `"iid"`, `"hc1"`, and clustered SEs, and are compatible with all strategies.
`vcov`	Character string or formula denoting the desired type of variance- covariance correction / standard errors. Options are `"iid"` (default), `"hc1"` (heteroskedasticity-consistent), or a one-sided formula like `~cluster_var` for cluster-robust standard errors. Note that `"hc1"` and clustered SEs require a second pass over the data unless `strategy = "compress"` to construct the residuals.

`strategy` Character string indicating the preferred acceleration strategy. The default `"auto"` will pick an optimal strategy based on internal heuristics. Users can also override with one of the following explicit strategies: `"compress"`, `"demean"` (alias: `"within"`), `"mundlak"`, or `"moments"`. See the Acceleration Strategies section below for details.

`compress_ratio, compress_nmax`

Numeric(s). Parameters that help to determine the acceleration `strategy` under the default `"auto"` option.

- `compress_ratio` defines the compression ratio threshold, i.e. numeric in the range `[0,1]` defining the minimum acceptable compressed versus the original data size. Default value of `NULL` means that the threshold will be automatically determined based on some internal heuristic (e.g., 0.01 for models without fixed effects).
- `compress_nmax` defines the maximum allowable size (in rows) of the compressed dataset that can be serialized into R. Pays heed to the idea that big data serialization can be costly (esp. for remote databases), even if we have achieved good compression on top of the original dataset. Default value is 1e6 (i.e., a million rows).

See the Acceleration Strategies section below for further details.

`cluster` Optional. Provides an alternative way to specify cluster-robust standard errors (i.e., instead of `vcov = ~cluster_var`). Either a one-sided formula (e.g., `~firm`) or character string giving the variable name. Only single-variable clustering is currently supported.

`ssc` Character string controlling the small-sample correction for clustered standard errors. Options are `"full"` (default) or `"nested"`. With `"full"`, all parameters (including fixed effect dummies) are counted in K for the CR1 correction. With `"nested"`, fixed effects that are nested within the cluster variable are excluded from K, matching the default behavior of `fixest::feols`. Only applies to `"compress"` and `"demean"` strategies (Mundlak uses explicit group mean regressors, not FE dummies). This distinction only matters for small samples. For large datasets (`dbreg`'s target use case), the difference is negligible and hence we default to the simple `"full"` option.

`sql_only` Logical indicating whether only the underlying compression SQL query should be returned (i.e., no computation will be performed). Default is `FALSE`.

`data_only` Logical indicating whether only the compressed dataset should be returned (i.e., no regression is run). Default is `FALSE`.

`drop_missings` Logical indicating whether incomplete cases (i.e., rows where any of the dependent, independent or FE variables are missing) should be dropped. The default is `TRUE`, according with standard regression software. It is *strongly* recommended not to change this value unless you are absolutely sure that your data have no missings and you wish to skip some internal checks. (Even then, it probably isn't worth it.)

`verbose` Logical. Print auto strategy and progress messages to the console? Defaults to `FALSE`. This can be overridden for a single call by supplying `verbose = TRUE`, or set globally via `options(dbreg.verbose = TRUE)`.

`...` Additional arguments. Currently ignored, except to handle superseded arguments for backwards compatibility.

Value

A list of class "dbreg" containing various slots, including a table of coefficients (which the associated `print` method will display).

Acceleration Strategies

`dbreg` offers four primary acceleration strategies for estimating regression results from simplified data representations. Below we use the shorthand Y (outcome), X (explanatory variables), and FE (fixed effects) for exposition purposes:

1. `"compress"`: compresses the data via a `GROUP BY` operation (using X and the FE as groups), before running weighted least squares on this much smaller dataset:

$$\hat{\beta} = (X_c' W X_c)^{-1} X_c' W Y_c$$

   where $W = \text{diag}(n_g)$ are the group frequencies. This procedure follows Wong et al. (2021).

2. `"moments"`: computes sufficient statistics $(X'X, X'y)$ directly via SQL aggregation, returning a single-row result. This solves the standard OLS normal equations $\hat{\beta} = (X'X)^{-1}X'y$. Limited to cases without FE.

3. `"demean"` (alias `"within"`): subtracts group-level means from both Y and X before computing sufficient statistics (per the `"moments"` strategy). For example, given unit i and time t FE, we apply double demeaning:

$$\ddot{Y}_{it} = \beta \ddot{X}_{it} + \varepsilon_{it}$$

   where $\ddot{X} = X - \bar{X}_i - \bar{X}_t + \bar{X}$. This (single-pass) within transformation is algebraically equivalent to the fixed effects projection—i.e., Frisch-Waugh-Lovell partialling out—in the presence of a single FE. It is also identical for the two-way FE (TWFE) case if your panel is balanced. For unbalanced two-way panels, however, the double demeaning strategy is not algebraically equivalent to the fixed effects projection and therefore does not recover the exact TWFE coefficients. In such cases, and also for weighted two-way FE, `dbreg` uses alternating projections (AP) to recover the exact FE coefficients, at the cost of extra passes over the data. AP also generalizes the `"demean"` strategy to three or more FE.

4. `"mundlak"`: a generalized Mundlak (1978), or correlated random effects (CRE) estimator that regresses Y on X plus group means of X:

$$Y_{it} = \alpha + \beta X_{it} + \gamma \bar{X}_i + \varepsilon_{it} \quad \text{(one-way)}$$

$$Y_{it} = \alpha + \beta X_{it} + \gamma \bar{X}_i + \delta \bar{X}_t + \varepsilon_{it} \quad \text{(two-way, etc.)}$$

   Unlike `"demean"`, Y is not transformed, so predictions are on the original scale. Supports any number of FE and works correctly for any panel structure (balanced or unbalanced). However, note that CRE is a *different model* from FE: while coefficients are asymptotically equivalent under certain assumptions, they will generally differ in finite samples.

The relative efficiency of each of these strategies depends on the size and structure of the data, as well as the number of unique regressors and FE. For (quote unquote) "standard" cases, the `"compress"` strategy can yield remarkable performance gains and should justifiably be viewed as a good default. However, the compression approach tends to be less efficient for true panels (repeated cross-sections over time), where N » T. In such cases, it can be more efficient to use a demeaning strategy that first controls for (e.g. subtracts) group means, before computing sufficient statistics on

the aggregated data. The reason for this is that time and unit FE are typically high dimensional, but covariate averages are not; see Arkhangelsky & Imbens (2024).

However, the demeaning approaches invite tradeoffs of their own. For example, the single-pass double demeaning transformation only obtains exact TWFE results for balanced panels with two FE. For unbalanced panels, weighted regressions, or three or more FE, `dbreg` uses alternating projections (iterative demeaning) which is exact but requires multiple passes and may be slower to converge on very large datasets. In such cases, `"mundlak"` (CRE) may be preferable as it is a single-pass estimator that obtains consistent coefficients regardless of panel structure and FE count, but at the "cost" of recovering a different estimand. (It is a different model to TWFE, after all.) See Wooldridge (2025) for an extended discussion of these issues.

Users should weigh these tradeoffs when choosing their acceleration strategy. Summarising, we can provide a few guiding principles. `"compress"` is a good default that guarantees the "exact" FE estimates and is usually very efficient (barring data I/O costs and high FE dimensionality). `"mundlak"` is another efficient alternative provided that the CRE estimand is acceptable (don't be alarmed if your coefficients are not identical). Finally, the `"demean"` and `"moments"` strategies are great for particular use cases (i.e., balanced panels and cases without FE, respectively).

If this all sounds like too much to think about, don't fret. The good news is that `dbreg` can do a lot (all?) of the deciding for you. Specifically, it will invoke an `"auto"` heuristic behind the scenes if a user does not provide an explicit acceleration strategy. Working through the heuristic logic does impose some additional overhead, but this should be negligible in most cases (certainly compared to the overall time savings). The `"auto"` heuristic is as follows:

- IF no FE AND (any continuous regressor OR poor compression ratio OR too big compressed data) THEN `"moments"`.
- ELSE IF 1 FE AND (poor compression ratio OR too big compressed data) THEN `"demean"`.
- ELSE IF 2 FE AND (poor compression ratio OR too big compressed data):
  - IF balanced panel THEN `"demean"`.
  - ELSE `"demean"` via alternating projections.
- ELSE IF 3+ FE AND (poor compression ratio OR too big compressed data) THEN `"demean"` via alternating projections.
- ELSE THEN `"compress"`.

Tip: set `dbreg(..., verbose = TRUE)` *to print information about the auto strategy decision criteria.*

References

Arkhangelsky, D. & Imbens, G. (2024) *Fixed Effects and the Generalized Mundlak Estimator.* The Review of Economic Studies, 91(5), pp. 2545–2571. Available: https://doi.org/10.1093/restud/rdad089

Mundlak, Y. (1978) *On the Pooling of Time Series and Cross Section Data.* Econometrica, 46(1), pp. 69–85. Available: https://doi.org/10.2307/1913646

Wong, J., Forsell, E., Lewis, R., Mao, T., & Wardrop, M. (2021). *You Only Compress Once: Optimal Data Compression for Estimating Linear Models.* arXiv preprint arXiv:2102.11297. Available: https://doi.org/10.48550/arXiv.2102.11297

Wooldridge, J.M. (2025) *Two-way fixed effects, the two-way mundlak regression, and difference-in-differences estimators.* Empirical Economics, 69, pp. 2545–2587. Available: https://doi.org/10.1007/s00181-025-02807-z

See Also

dbConnect for creating database connections, duckdb for DuckDB-specific connections

Examples

```
## In-memory data ----

# We can pass in-memory R data frames to an ephemeral DuckDB connection via
# the `data` argument. This is convenient for small(er) datasets and demos.

# Default "compress" strategy reduces the data to 4 rows before running OLS
dbreg(weight ~ Diet, data = ChickWeight)

# Compare with lm
summary(lm(weight ~ Diet, data = ChickWeight))$coefficients

# Add "fixed effects" after a `|`
dbreg(weight ~ Time | Diet, data = ChickWeight)

# "robust" SEs can also be computed using a sufficient statistics approach
dbreg(weight ~ Time | Diet, data = ChickWeight, vcov = "hc1")
dbreg(weight ~ Time | Diet, data = ChickWeight, vcov = ~Chick)

# Different acceleration strategies + specifications
dbreg(weight ~ Time | Diet, data = ChickWeight, strategy = "demean")
dbreg(weight ~ Time | Diet, data = ChickWeight, strategy = "mundlak")
dbreg(weight ~ Time | Diet + Chick, data = ChickWeight, strategy = "mundlak") # two-way Mundlak
dbreg(weight ~ Time, data = ChickWeight, strategy = "moments") # no FEs
# etc.

# Interactions: does the effect of Time vary by Diet?
# (Diet main effects are collinear with Chick FE, so these drop out)
dbreg(weight ~ Time * Diet | Chick, data = ChickWeight)

#
## DBI connection ----

# For persistent databases or more control, use the `conn` + `table` args.
# Again, we use DuckDB below but any other DBI-supported backend should work
# too (e.g., odbc, bigrquery, noctua (AWS Athena),  etc.) See:
# https://r-dbi.org/backends/

library(DBI)
con = dbConnect(duckdb::duckdb())
dbWriteTable(con, "cw", as.data.frame(ChickWeight))

dbreg(weight ~ Time | Diet, conn = con, table = "cw")

# Tip: Rather than creating or writing (temp) tables, use CREATE VIEW to
# define subsets or computed columns without materializing data. This is more
# efficient and especially useful for filtering or adding variables.
dbExecute(
```

```
  con,
  "
  CREATE VIEW cw1 AS
  SELECT *
  FROM cw
  WHERE Diet = 1
  "
)
dbreg(weight ~ Time | Chick, conn = con, table = "cw1")

#
## Path to file ----
#
# For file-based data (e.g., parquet), use the path argument.

tmp = tempfile(fileext = ".parquet")
dbExecute(con, sprintf("COPY cw TO '%s' (FORMAT PARQUET)", tmp))

dbreg(weight ~ Time | Diet, path = tmp)

# Cleanup
dbDisconnect(con)
unlink(tmp)

#
## Big dataset ----

# For a more compelling and appropriate dbreg use-case, i.e. regression on a
# big (~180 million row) dataset of Hive-partioned parquet files, see the
# package website:
# https://grantmcdermott.com/dbreg/
```

`gof` *Calculate goodness-of-fit metrics for dbreg objects*

Description

Calculate goodness-of-fit metrics for dbreg objects

Usage

```
gof(object, ...)
```

Arguments

`object`	A ‘dbreg‘ object.
`...`	Additional arguments (currently unused)

Value

Named vector with r2, adj_r2, and rmse

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)
gof(mod)
```

`plot.dbbinsreg` *Plot method for dbbinsreg objects*

Description

Visualizes binned regression results from `dbbinsreg`. Plots dots at bin means with optional confidence intervals and/or confidence bands, and optionally overlays a smooth line if computed. Uses tinyplot for rendering but works with both `plot()` and `tinyplot()` generics.

Usage

```
## S3 method for class 'dbbinsreg'
plot(
  x,
  type = NULL,
  ci = TRUE,
  cb = TRUE,
  line = TRUE,
  lty = 1,
  theme = "basic",
  ...
)

## S3 method for class 'dbbinsreg'
tinyplot(
  x,
  type = NULL,
  ci = TRUE,
  cb = TRUE,
  line = TRUE,
  lty = 1,
  theme = "basic",
  ...
)
```

Arguments

x	A dbbinsreg object
type	The type of plot. If NULL (the default), then the type will be inferred based on the underlying object (e.g, "pointrange" for points with confidence intervals).
ci	Logical. Show confidence intervals for dots? Default is TRUE.
cb	Logical. Show confidence bands as a ribbon? Default is TRUE if available in the object.
line	Logical. Show the line overlay if available? Default is TRUE.
lty	Integer or character string. Line type for line overlay.
theme	Character string. One of the valid plot themes supported by tinytheme. The default "basic" theme is a light adaptation of the standard base graphics aesthetic, featuring filled points and a background grid. Various other themes are supported (e.g., "clean", "minimal", "classic", etc.), while passing NULL switches the theme off entirely.
...	Additional arguments passed to tinyplot, e.g. main, sub, file, etc.

Examples

```
#
## In-memory data ----

# Like `dbreg`, we can pass in-memory R data frames to an ephemeral DuckDB
# connection via the `data` argument.

# Canonical binscatter: bin means (default)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10)

# You can pass additional plotting arguments via ... to (tiny)plot.dbbinsreg
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10,
          main = "A simple binscatter example", theme = "clean")

# Alternatively, save the object and plot later
bs = dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, plot = FALSE)
plot(bs, main = "Same example, different theme", theme = "classic")

# Piecewise linear (p = 1), no smoothness (s = 0)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 0))

# Piecewise linear (p = 1) with continuity (s = 1)
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 1))

# With line overlay for smooth visualization
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(1, 1), line = TRUE)

# Different line smoothness to points
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, points = c(0, 0), line = c(1, 1))

# With uniform confidence bands (much greater uncertainty)
set.seed(99)
```

```
dbbinsreg(weight ~ Time, data = ChickWeight, nbins = 10, cb = TRUE)

# Accounting for Diet "fixed effects" helps to resolve the situation
# (We'll also add a line and change the theme for a nicer plot)
dbbinsreg(weight ~ Time | Diet, data = ChickWeight, nbins = 10, cb = TRUE,
          line = c(1, 1), theme = "clean")

#
## DBI connection ----

library(DBI)
con = dbConnect(duckdb::duckdb())
dbWriteTable(con, "cw", as.data.frame(ChickWeight))

dbbinsreg(weight ~ Time | Diet, conn = con, table = "cw", nbins = 10,
          theme = "clean")
# etc.

# See ?dbreg for more connection examples

# Clean up
dbDisconnect(con)
```

`predict.dbreg` *Predict method for dbreg objects*

Description

Predict method for dbreg objects

Usage

```
## S3 method for class 'dbreg'
predict(
  object,
  newdata = NULL,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

`object`	A ‘dbreg‘ object.
`newdata`	Data frame for predictions. Required for objects that were estimated using the ‘"mundlak"‘ and ‘"moments"‘ strategies, since ‘dbreg‘ does not retain any data for these estimations.

`interval`	Type of interval to compute: ‘"none"‘ (default), ‘"confidence"‘, or ‘"prediction"‘. Note that ‘"confidence"‘ intervals reflect uncertainty in the estimated mean, while ‘"prediction"‘ intervals additionally account for residual variance. See `predict.lm` for details.
`level`	Confidence level for intervals. Default is 0.95.
`...`	Additional arguments (currently unused).

Predicting on "demean" strategy objects

Predicting on ‘dbreg‘ objects should generally work as expected. However, predictions from ‘"demean"‘ strategy models carry two important caveats:

1. Predictions require group means to transform back to the original scale. If ‘newdata‘ contains the outcome variable, group means are computed from ‘newdata‘ and used to return level predictions. If the outcome is absent, within-group predictions (deviations from group means) are returned instead, with a message.

2. Confidence/prediction intervals are not supported. A demeaned model cannot account for uncertainty in the fixed-effects (since these were absorbed at estimation time), which in turn would yield intervals that are too narrow. Requesting intervals for ‘"demean"‘ strategy models will return point predictions with a message. Users should re-estimate with a different strategy if intervals are needed.

See Also

[dbreg()] for examples.

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)

# coefficients
coef(mod)
coef(mod, fe = TRUE)  # include fixed effects

# confidence intervals
confint(mod)

# variance-covariance matrix
vcov(mod)

# predictions
head(predict(mod, newdata = airquality))
head(predict(mod, newdata = airquality, interval = "confidence"))
```

print.dbbinsreg *Print method for dbbinsreg objects (binsreg-compatible format)*

Description

Print method for dbbinsreg objects (binsreg-compatible format)

Usage

```
## S3 method for class 'dbbinsreg'
print(x, ...)
```

Arguments

x	A dbbinsreg object
...	Additional arguments passed to print

print.dbreg *Print method for dbreg objects*

Description

Print method for dbreg objects

Usage

```
## S3 method for class 'dbreg'
print(x, fe = FALSE, ...)
```

Arguments

x	‘dbreg‘ object.
fe	Should the fixed effects be displayed? Default is ‘FALSE‘.
...	Other arguments passed to print. Currently unused, except to capture superseded arguments.

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)
# mod # same as below
print(mod)
print(mod, fe = TRUE)  # include fixed effects
```

`sql_model_matrix` *Construct SQL expressions for a design matrix*

Description

Expands formula terms into SQL SELECT expressions, handling factor one-hot encoding and interaction terms. Only the right-hand side of the formula is processed; the response variable (LHS) is ignored.

Usage

```
sql_model_matrix(
  formula,
  conn,
  table,
  expand = c("all", "interactions"),
  sep = "_x_",
  fe_vars = character()
)
```

Arguments

`formula`	A formula (or Formula) object. Only RHS terms are expanded; the LHS (response variable) is ignored.
`conn`	Database connection
`table`	Table name or FROM clause
`expand`	Character: ‘"all"‘ expands factors and interactions, ‘"interactions"‘ only expands interaction terms (factors in main effects kept as-is for grouping)
`sep`	Character separator for interaction term names. Default is ‘"_x_"‘. Use ‘":"‘ for standard R naming convention.
`fe_vars`	Character vector of fixed effect variable names. These are treated as "in the model" for determining whether to drop reference levels in interactions.

Value

List with: - ‘select_exprs‘: character vector of SQL expressions - ‘col_names‘: corresponding column names - ‘factor_levels‘: list of factor levels by variable (for reference)

Examples

```
library(DBI)
library(duckdb)
con = dbConnect(duckdb())
duckdb_register(con, "test", data.frame(x1 = 1:3, x2 = c("a", "b", "c")))
sql_model_matrix(~ x1 + x2, con, "test")
sql_model_matrix(~ x1:x2, con, "test")
sql_model_matrix(~ x1:x2, con, "test", sep = ":")
dbDisconnect(con)
```

tidiers *Tidiers for 'dbreg' objects*

Description

Provides `broom::tidy` and `broom::glance` methods for "dbreg" objects.

Usage

```
## S3 method for class 'dbreg'
tidy(x, conf.int = FALSE, conf.level = 0.95, fe = FALSE, ...)

## S3 method for class 'dbreg'
glance(x, ...)
```

Arguments

`x`	a model of class 'dbreg' produced by the `dbreg` function.
`conf.int`	Logical indicating whether to include confidence intervals. Default is 'FALSE'.
`conf.level`	Confidence level for intervals. Default is 0.95.
`fe`	Should the fixed effects be tidied too? Default is 'FALSE'.
`...`	Additional arguments to tidying method. Currently unused except to handle superseded arguments.

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)
tidy(mod, conf.int = TRUE)
tidy(mod, conf.int = TRUE, fe = TRUE)
glance(mod)
```

vcov.dbreg *Variance-covariance matrix for dbreg objects*

Description

Variance-covariance matrix for dbreg objects

Usage

```
## S3 method for class 'dbreg'
vcov(object, ...)
```

Arguments

object	A ‘dbreg‘ object.
...	Additional arguments (currently unused).

Examples

```
mod = dbreg(Temp ~ Wind | Month, data = airquality)

# coefficients
coef(mod)
coef(mod, fe = TRUE)  # include fixed effects

# confidence intervals
confint(mod)

# variance-covariance matrix
vcov(mod)

# predictions
head(predict(mod, newdata = airquality))
head(predict(mod, newdata = airquality, interval = "confidence"))
```

Index