Package 'dbplot'

March 13, 2026

Version 0.4.0

Title Simplifies Plotting Data Inside Databases

Description Leverages 'dplyr' to process the calculations of a plot inside a database. This package provides helper functions that abstract the work at three levels: outputs a 'ggplot', outputs the calculations, outputs the formula needed to calculate bins.

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.0), rlang (>= 1.0.0), ggplot2 (>= 3.3.0), purrr

Suggests DBI, dbplyr (>= 2.0.0), testthat (>= 3.0.0), tidyr, covr, lifecycle, duckdb, RSQLite, nycflights13

License MIT + file LICENSE

URL https://github.com/edgararuiz/dbplot

BugReports https://github.com/edgararuiz/dbplot/issues

RoxygenNote 7.3.3

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation no

Author Edgar Ruiz [aut, cre]

Maintainer Edgar Ruiz <edgararuiz@gmail.com>

Repository CRAN

Date/Publication 2026-03-13 16:30:02 UTC

Contents

dbplot_bar 2
dbplot_boxplot 3
dbplot_histogram 4
dbplot_line 5
dbplot_raster 6
db_bin 8

db_compute_bins 9
db_compute_boxplot 10
db_compute_count 11
db_compute_raster 12

Index **14**

---

dbplot_bar *Bar plot*

Description

Uses dplyr operations to aggregate data and then 'ggplot2' to create the plot. Because of this approach, the calculations automatically run inside the database if 'data' has a database or sparklyr connection. The 'class()' of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
dbplot_bar(data, x, ..., y = n())
```

Arguments

data	A table (tbl)
x	A discrete variable
...	A set of named or unnamed aggregations
y	The aggregation formula. Defaults to count (n)

Value

A ggplot object with a bar plot. If multiple aggregations are provided, returns a list of ggplot objects, one for each aggregation.

See Also

dbplot_line , dbplot_histogram, dbplot_raster

Examples

```
## Not run:
library(DBI)
library(dplyr)
library(ggplot2)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# Returns a plot of the row count per am
db_mtcars |>
  dbplot_bar(am)
```

```
# Returns a plot of the average mpg per am
db_mtcars |>
  dbplot_bar(am, mean(mpg))

# Returns the average and sum of mpg per am
db_mtcars |>
  dbplot_bar(am, avg_mpg = mean(mpg), sum_mpg = sum(mpg))

dbDisconnect(con)

## End(Not run)
```

`dbplot_boxplot` *Boxplot*

Description

Uses dplyr operations to aggregate data and then 'ggplot2' to create the boxplot. Because of this approach, the calculations automatically run inside the database if 'data' has a database or sparklyr connection. The 'class()' of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Requires database support for percentile/quantile functions. See `db_compute_boxplot` for supported database backends.

Usage

```
dbplot_boxplot(data, x, var, coef = 1.5)
```

Arguments

`data`	A table (tbl)
`x`	A discrete variable in which to group the boxplots
`var`	A continuous variable
`coef`	Length of the whiskers as multiple of IQR. Defaults to 1.5

Value

A ggplot object displaying boxplots for the specified variable grouped by x.

See Also

`dbplot_bar`, `dbplot_line` , `dbplot_raster`, `dbplot_histogram`

Examples

```
## Not run:
library(DBI)
library(dplyr)
library(ggplot2)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

db_mtcars |>
  dbplot_boxplot(am, mpg)

dbDisconnect(con)

## End(Not run)
```

`dbplot_histogram` *Histogram*

Description

Uses dplyr operations to aggregate data and then ‘ggplot2‘ to create the histogram. Because of this approach, the calculations automatically run inside the database if ‘data‘ has a database or sparklyr connection. The ‘class()‘ of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
dbplot_histogram(data, x, bins = 30, binwidth = NULL)
```

Arguments

`data`	A table (tbl)
`x`	A continuous variable
`bins`	Number of bins. Defaults to 30.
`binwidth`	Fixed width for each bin, in the same units as the data. Overrides bins when specified

Value

A ggplot object displaying a histogram of the specified variable.

See Also

`dbplot_bar`, `dbplot_line` , `dbplot_raster`

Examples

```
## Not run:
library(DBI)
library(dplyr)
library(ggplot2)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# A ggplot histogram with 30 bins
db_mtcars |>
  dbplot_histogram(mpg)

# A ggplot histogram with bins of size 10
db_mtcars |>
  dbplot_histogram(mpg, binwidth = 10)

dbDisconnect(con)

## End(Not run)
```

`dbplot_line` *Line plot*

Description

Uses dplyr operations to aggregate data and then 'ggplot2' to create a line plot. Because of this approach, the calculations automatically run inside the database if 'data' has a database or sparklyr connection. The 'class()' of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

If multiple named aggregations are passed, 'dbplot' will only use one SQL query to perform all of the operations. The purpose is to increase efficiency, and only make one "trip" to the database in order to obtains multiple, related, plots.

Usage

```
dbplot_line(data, x, ..., y = n())
```

Arguments

`data`	A table (tbl)
`x`	A discrete variable
`...`	A set of named or unnamed aggregations
`y`	The aggregation formula. Defaults to count (n)

Value

A ggplot object with a line plot. If multiple aggregations are provided, returns a list of ggplot objects, one for each aggregation.

See Also

dbplot_bar, dbplot_histogram, dbplot_raster

Examples

```
## Not run:
library(DBI)
library(dplyr)
library(ggplot2)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# Returns a plot of the row count per cyl
db_mtcars |>
  dbplot_line(cyl)

# Returns a plot of the average mpg per cyl
db_mtcars |>
  dbplot_line(cyl, mean(mpg))

# Returns the average and sum of mpg per am
db_mtcars |>
  dbplot_line(am, avg_mpg = mean(mpg), sum_mpg = sum(mpg))

dbDisconnect(con)

## End(Not run)
```

---

dbplot_raster *Raster plot*

Description

To visualize two continuous variables, we typically resort to a Scatter plot. However, this may not be practical when visualizing millions or billions of dots representing the intersections of the two variables. A Raster plot may be a better option, because it concentrates the intersections into squares that are easier to parse visually.

Uses dplyr operations to aggregate data and ggplot2 to create a raster plot. Because of this approach, the calculations automatically run inside the database if ‘data‘ has a database or sparklyr connection. The ‘class()‘ of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
dbplot_raster(data, x, y, fill = n(), resolution = 100, complete = FALSE)
```

Arguments

data	A table (tbl)
x	A continuous variable
y	A continuous variable
fill	The aggregation formula. Defaults to count (n)
resolution	The number of bins created per variable. The higher the number, the more records will be imported from the source
complete	Uses tidyr::complete to include empty bins. Inserts value of 0.

Details

There are two considerations when using a Raster plot with a database. Both considerations are related to the size of the results downloaded from the database:

- The number of bins requested: The higher the bins value is, the more data is downloaded from the database.

- How concentrated the data is: This refers to how many intersections return a value. The more intersections without a value, the less data is downloaded from the database.

Value

A ggplot object displaying a raster/heatmap plot of the aggregated data across the two continuous variables.

See Also

dbplot_bar, dbplot_line , dbplot_histogram

Examples

```
## Not run:
library(DBI)
library(dplyr)
library(ggplot2)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_faithful <- copy_to(con, faithful, "faithful")

# Returns a 100x100 raster plot of record count of intersections of eruptions and waiting
db_faithful |>
  dbplot_raster(eruptions, waiting)

# Returns a 50x50 raster plot of eruption averages of intersections of eruptions and waiting
db_faithful |>
  dbplot_raster(eruptions, waiting, fill = mean(eruptions), resolution = 50)

dbDisconnect(con)

## End(Not run)
```

db_bin *Bin formula*

Description

Uses the rlang package to build the formula needed to create the bins of a numeric variable in an unevaluated fashion. This way, the formula can be then passed inside a dplyr verb.

Usage

```
db_bin(var, bins = 30, binwidth = NULL)
```

Arguments

var	Variable name or formula
bins	Number of bins. Defaults to 30.
binwidth	Fixed width for each bin, in the same units as the data. Overrides bins when specified

Value

An unevaluated expression (rlang quosure) that calculates bin membership for the specified variable. This expression is designed to be used within dplyr verbs using the '!¡ operator.

Examples

```
## Not run:
library(DBI)
library(dplyr)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# Important: Always name the field and
# prefix the function with `!!` (See Details)

# Uses the default 30 bins
db_mtcars |>
  group_by(x = !!db_bin(mpg)) |>
  count()

# Uses binwidth which overrides bins
db_mtcars |>
  group_by(x = !!db_bin(mpg, binwidth = 10)) |>
  count()

dbDisconnect(con)

## End(Not run)
```

db_compute_bins *Calculate histogram bins*

Description

Uses dplyr operations to create histogram bins. Because of this approach, the calculations automatically run inside the database if ‘data‘ has a database or sparklyr connection. The ‘class()‘ of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
db_compute_bins(data, x, bins = 30, binwidth = NULL)
```

Arguments

data	A table (tbl)
x	A continuous variable
bins	Number of bins. Defaults to 30.
binwidth	Fixed width for each bin, in the same units as the data. Overrides bins when specified

Value

An ungrouped data.frame with two columns: the bin values for the x variable and the count of observations in each bin.

See Also

db_bin,

Examples

```
## Not run:
library(DBI)
library(dplyr)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# Returns record count for 30 bins in mpg
db_mtcars |>
  db_compute_bins(mpg)

# Returns record count for bins of size 10
db_mtcars |>
  db_compute_bins(mpg, binwidth = 10)

dbDisconnect(con)

## End(Not run)
```

`db_compute_boxplot` *Returns a dataframe with boxplot calculations*

Description

Uses dplyr operations to create boxplot calculations. Because of this approach, the calculations automatically run inside the database if ‘data‘ has a database or sparklyr connection. The ‘class()‘ of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Requires database support for percentile/quantile functions. Confirmed to work with:

- DuckDB (recommended for local examples) - uses quantile()
- Spark/Hive (via sparklyr) - uses percentile_approx()
- SQL Server (2012+) - uses PERCENTILE_CONT()
- PostgreSQL (9.4+) - uses percentile_cont()
- Oracle (9i+) - uses PERCENTILE_CONT()

Does NOT work with SQLite, MySQL < 8.0, or MariaDB (no percentile support).

Note that this function supports input tbl that already contains grouping variables. This can be useful when creating faceted boxplots.

Usage

```
db_compute_boxplot(data, x, var, coef = 1.5)
```

Arguments

`data`	A table (tbl) that can already contain grouping variables
`x`	A discrete variable in which to group the boxplots
`var`	A continuous variable
`coef`	Length of the whiskers as multiple of IQR. Defaults to 1.5

Value

An ungrouped data.frame with boxplot statistics including columns for the grouping variable(s), quartiles (lower, middle, upper), whisker bounds (ymin, ymax), and the count of observations per group.

Examples

```
## Not run:
library(DBI)
library(dplyr)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

db_mtcars |>
```

```
  db_compute_boxplot(am, mpg)

dbDisconnect(con)

## End(Not run)
```

---

db_compute_count *Aggregates over a discrete field*

Description

Uses dplyr operations to aggregate data. Because of this approach, the calculations automatically run inside the database if 'data' has a database or sparklyr connection. The 'class()' of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
db_compute_count(data, x, ..., y = n())
```

Arguments

data	A table (tbl)
x	A discrete variable
...	A set of named or unnamed aggregations
y	The aggregation formula. Defaults to count (n)

Value

An ungrouped data.frame with the discrete variable and one or more aggregation columns. The first column is the grouping variable (x), followed by the aggregated values.

Examples

```
## Not run:
library(DBI)
library(dplyr)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_mtcars <- copy_to(con, mtcars, "mtcars")

# Returns the row count per am
db_mtcars |>
  db_compute_count(am)

# Returns the average mpg per am
db_mtcars |>
  db_compute_count(am, mean(mpg))

# Returns the average and sum of mpg per am
```

```
db_mtcars |>
  db_compute_count(am, mean(mpg), sum(mpg))

dbDisconnect(con)

## End(Not run)
```

`db_compute_raster` *Aggregates intersections of two variables*

Description

To visualize two continuous variables, we typically resort to a Scatter plot. However, this may not be practical when visualizing millions or billions of dots representing the intersections of the two variables. A Raster plot may be a better option, because it concentrates the intersections into squares that are easier to parse visually.

Uses dplyr operations to aggregate data. Because of this approach, the calculations automatically run inside the database if 'data' has a database or sparklyr connection. The 'class()' of such tables in R are: tbl_sql, tbl_dbi, tbl_spark

Usage

```
db_compute_raster(data, x, y, fill = n(), resolution = 100, complete = FALSE)

db_compute_raster2(data, x, y, fill = n(), resolution = 100, complete = FALSE)
```

Arguments

`data`	A table (tbl)
`x`	A continuous variable
`y`	A continuous variable
`fill`	The aggregation formula. Defaults to count (n)
`resolution`	The number of bins created per variable. The higher the number, the more records will be imported from the source
`complete`	Uses tidyr::complete to include empty bins. Inserts value of 0.

Details

There are two considerations when using a Raster plot with a database. Both considerations are related to the size of the results downloaded from the database:

- The number of bins requested: The higher the bins value is, the more data is downloaded from the database.

- How concentrated the data is: This refers to how many intersections return a value. The more intersections without a value, the less data is downloaded from the database.

Value

An ungrouped data.frame with three columns: the x variable bins, the y variable bins, and the aggregated fill values for each x-y intersection.

For ‘db_compute_raster2‘: A data.frame with five columns - the x and y variable bins, the fill values, and additional columns for the upper bounds of each bin (x_2 and y_2), useful for defining precise tile boundaries.

Examples

```
## Not run:
library(DBI)
library(dplyr)
con <- dbConnect(duckdb::duckdb(), ":memory:")
db_faithful <- copy_to(con, faithful, "faithful")

# Returns a 100x100 grid of record count of intersections of eruptions and waiting
db_faithful |>
  db_compute_raster(eruptions, waiting)

# Returns a 50x50 grid of eruption averages of intersections of eruptions and waiting
db_faithful |>
  db_compute_raster(eruptions, waiting, fill = mean(eruptions), resolution = 50)

dbDisconnect(con)

## End(Not run)
```

Index

db_bin, 8, *9*
db_compute_bins, 9
db_compute_boxplot, *3*, 10
db_compute_count, 11
db_compute_raster, 12
db_compute_raster2 *(*db_compute_raster*)*, 12
dbplot_bar, 2, *3, 4*, *6, 7*
dbplot_boxplot, 3
dbplot_histogram, *2, 3*, 4, *6, 7*
dbplot_line, *2–4*, 5, *7*
dbplot_raster, *2–4*, *6*, 6