Package: sideview (via r-universe)

November 1, 2026

Title Explore and Join Many Tables on a Zoomable Canvas

Version 0.4.0

Description Lays out many data frames side by side as cards on a pan-and-zoom canvas, either as a 'shiny' gadget for 'RStudio' or as an 'htmlwidgets' widget for 'R Markdown' and 'Quarto' pages. Drawing a line between two columns writes the 'dplyr' join and checks it first: match rates, unmatched keys and the key relationship. Sorts, filters and other steps build up a readable pipeline of R code, and a column inspector exports a data dictionary. An optional large language model (LLM) assistant, using 'Ollama' or an 'OpenAI'-compatible service, suggests 'tidyverse' code from the table schema only, never the data rows.

URL https://lauyeehow1986-hub.github.io/sideview/, https://github.com/lauyeehow1986-hub/sideview

BugReports https://github.com/lauyeehow1986-hub/sideview/issues

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports shiny, htmltools, htmlwidgets, dplyr (>= 1.1.0), rlang, httr2, jsonlite, later, rstudioapi

Suggests testthat (>= 3.0.0), shinytest2, arrow, duckdb, DBI, gt, flextable, knitr, rmarkdown, writexl, withr, bslib, dbplyr, stringr, callr, chromote, httpuv, pkgload, lubridate

Config/testthat/edition 3

Config/pak/sysreqs cmake make libuv1-dev libssl-dev zlib1g-dev

Repository https://lauyeehow1986-hub.r-universe.dev

Date/Publication 2026-11-01 03:24:54 UTC

RemoteUrl https://github.com/lauyeehow1986-hub/sideview

RemoteRef main

RemoteSha 68869b2c4a209fb2b369937b402b9ffa30424501

Contents

new_sideview_state . . . 3
print.sideview_state . . . 4
renderSideview . . . 4
save_sideview . . . 5
sideview . . . 6
sideview_addin . . . 8
sideview_server . . . 9
sideview_ui . . . 10
sideview_widget . . . 11
sideviewOutput . . . 12
sideviewProxy . . . 12
sideviewUI . . . 13
sv_add_card . . . 14
sv_auto_clean_rules . . . 15
sv_build_chat_prompt . . . 17
sv_build_prompt . . . 18
sv_build_schema_context . . . 18
sv_call_llm . . . 19
sv_code_arrange . . . 20
sv_code_filter . . . 20
sv_code_group_summarize . . . 21
sv_code_join . . . 21
sv_code_mutate . . . 22
sv_code_pivot_longer . . . 23
sv_code_pivot_wider . . . 23
sv_code_recode . . . 24
sv_code_rename . . . 25
sv_code_select . . . 26
sv_codebook . . . 26
sv_column_profile . . . 27
sv_detect_discrepancies . . . 28
sv_escape_ident . . . 29
sv_extract_code . . . 29
sv_find_tables . . . 30
sv_insert_code . . . 30
sv_is_rstudio_available . . . 31
sv_join_diagnostics . . . 31
sv_llm_models . . . 33
sv_load_state . . . 34
sv_normalize . . . 35
sv_save_state . . . 35
sv_send_to_console . . . 36
sv_suggest_mappings . . . 37
sv_theme . . . 38

Index **39**

new_sideview_state *Create a New Sideview Canvas State Object*

Description

Constructs a structured `sideview_state` object containing canvas transform, card configurations, table join connections, and pipeline script state.

Usage

```
new_sideview_state(
  version = "0.3.0",
  canvas = list(zoom = 1, pan_x = 0, pan_y = 0),
  cards = list(),
  connections = list(),
  pipeline = list(code = "", active_tab = "pipeline")
)
```

Arguments

`version`	Character string representing state schema version (default `"0.3.0"`).
`canvas`	Named list containing canvas settings: `zoom` (numeric), `pan_x` (numeric), `pan_y` (numeric).
`cards`	List of card state objects (each specifying `id`, `name`, `x`, `y`, `width`, `height`, `minimized`, `sorts` and `filters`; a sort is `list(col, dir)` with `dir` `"asc"` or `"desc"`, a filter is `list(col, op, val)`).
`connections`	List of join connection objects (each specifying `from_table`, `from_col`, `to_table`, `to_col`, `join_type`).
`pipeline`	Named list containing `code` (character), `active_tab` (character) and, from version 0.3.0, the pipeline `steps` and `pipe` (`"%>%"` or `"|>"`). Without `steps`, `code` is restored as one custom step.

Details

The canvas writes this shape when you click **Save** in the toolbar, and `sideview(..., state = )` / `sideview_widget(..., state = )` restore it. Cards are matched to tables by `name`.

Value

An object of class `c("sideview_state", "list")`.

Examples

```
st <- new_sideview_state()
print(st)
```

print.sideview_state *Print Method for Sideview State Objects*

Description

Displays a concise summary of the canvas state including version, card count, and active connections.

Usage

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

Arguments

x	A sideview_state object.
...	Unused additional arguments.

Value

Invisibly returns x.

renderSideview *Shiny Render Function for Sideview Widget*

Description

Shiny Render Function for Sideview Widget

Usage

```
renderSideview(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that generates a sideview_widget.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? Default FALSE.

Value

An S3 object of class c("shiny.render.function", "function").

save_sideview *Save Sideview Widget to a Standalone HTML File*

Description

Exports a `sideview` htmlwidget or table objects to a self-contained HTML file with all styles, fonts, and JavaScript embedded.

Usage

```
save_sideview(
  widget,
  file,
  selfcontained = TRUE,
  title = "Sideview",
  background = "white",
  max_rows = 5000,
  ...
)
```

Arguments

`widget`	A `sideview` htmlwidget object, or tabular data objects that can be converted to one via `sideview_widget`.
`file`	Path to the output HTML file (e.g. `"report.html"`).
`selfcontained`	Logical; whether to produce a standalone HTML file with all CSS and JavaScript dependencies inlined. Defaults to `TRUE`.
`title`	HTML title for the output page. Default `"Sideview"`.
`background`	Background color for the document body. Default `"white"`.
`max_rows`	Row cap used when `widget` is table data rather than a widget (see `sideview_widget()`; default 5000). A widget you built yourself keeps the cap it was built with.
`...`	Additional arguments passed to `htmlwidgets::saveWidget()`.

Value

The normalized path to the saved file (invisibly).

Examples

```
out <- tempfile(fileext = ".html")
save_sideview(mtcars, file = out)
unlink(out)
```

`sideview` *Multi-Table 2D Infinite Canvas Interactive Gadget*

Description

Launches an interactive Shiny gadget containing an infinite 2D canvas with mouse-wheel zoom, pan, draggable table cards, interactive column mapping, discrepancy recoding, and LLM-assisted tidyverse code generation.

Usage

```
sideview(
  ...,
  title = "Sideview Canvas",
  viewer = sv_default_viewer(),
  theme = "auto",
  state = NULL,
  env = parent.frame(),
  run = interactive()
)
```

Arguments

`...`	Arbitrary table objects (`data.frame`, `tibble`, `matrix`, `table`, `gt`, `flextable`, `htmlwidgets`), named or unnamed.
`title`	Title for the gadget viewer.
`viewer`	Shiny viewer gadget display mode (e.g. `shiny::paneViewer()`, `shiny::dialogViewer()`, `shiny::browserViewer()`), or a character shorthand (`"browser"`, `"dialog"`, or `"pane"`). Default is `shiny::dialogViewer()` (width 1400, height 900) in RStudio, `shiny::browserViewer()` in Positron (it has no dialog viewer; `"dialog"` there also opens the browser, with one message) and `shiny::paneViewer()` elsewhere.
`theme`	Canvas theme: `"auto"` (default; follows the RStudio editor theme), `"light"`, `"dark"`, an `sv_theme()` or a `bslib::bs_theme()`.
`state`	A saved workspace to restore: a `sideview_state` (see `new_sideview_state()`) or the path to a file written by the toolbar's **Save** button or `sv_save_state()`. Card positions and sizes, zoom and pan, sorts, filters, joins and the pipeline come back; cards are matched to tables by name.
`env`	Environment where **Run in R** evaluates the generated code and stores its result (as `sv_result` unless you name it). Defaults to the environment `sideview()` is called from, which is your workspace when you call it from the console.
`run`	Logical; if `TRUE`, launches the interactive gadget via `shiny::runGadget()`. If `FALSE`, returns the `shiny.appobj` definition without launching (useful for testing or embedding). Defaults to `interactive()`.

Value

If `run = TRUE`, the generated code (invisibly) when you click **Done**, or `NULL` after **Cancel** or closing the window. If `run = FALSE`, returns a `shiny.appobj` with the normalized cards attached.

Big tables (arrow and duckdb)

Arrow tables, datasets and queries and duckdb tables (`dplyr::tbl()`) go on the canvas as cards that show a 50-row preview. When the duckdb, dbplyr and DBI packages are installed, sorting, filtering, scrolling, the join check, the column inspector and the codebook use the whole table: duckdb runs them and only summaries and row windows come back into R. The table is never collected. Without duckdb, or when the link to a table is lost, the card says "First 50 of N rows" and the checks use the preview.

Generated code stays lazy and ends in one `collect()`. Known differences from an in-memory card:

- "Contains" on a number matches its text. The grid filter drops a trailing `.0`, but in generated code duckdb writes a double as `"130.0"` where R writes `"130"`, and float noise or scientific notation can differ. The generated filter is a literal match, `stringr::str_detect(col, "<escaped literal>")`.
- POSIXct values at exactly midnight, and `-0`, can print differently.
- Join keys of different types compare as text, and double keys compare exactly.
- Arrow time-zone offsets are computed for 1900 to 2100; local times outside that range may be off. duckdb `TIMESTAMPTZ` columns show in UTC.
- Joining arrow with duckdb copies the arrow side into a duckdb temporary table (`arrow::to_duckdb()` then `dplyr::compute()`), which costs memory; a data frame joined into duckdb is copied as `sideview_<table>`.
- duckdb 1.4.3 can crash when it scans an arrow factor column that contains `NA`. sideview decodes such columns itself, but generated code that calls `arrow::to_duckdb()` on one may hit it: convert the factor to character first.

If **Run in R** fails on a lazy pipeline, the message suggests moving `collect()` above the step that needs R.

Known differences

Date rules on joins (the calendar button on a join step) give the same rows in R, in the saved widget and on duckdb/arrow cards, with these exceptions:

- The saved widget drops fractions of a second from date-times, so a boundary that falls inside a second can differ from R.
- The generated duckdb code for a date-time window uses the session time zone when the ICU extension is loaded. With a non-UTC `TimeZone` setting, a boundary can move.
- Running the code in R on a mixed arrow/data-frame and duckdb chain may leave temporary `dbplyr_*` tables on your connection until it closes.
- sideview writes `na_matches = "never"` on every rule join (dplyr's default would match `NA` to `NA`).

Keyboard and navigation

Single-key shortcuts work when the canvas has the keyboard and you are not typing: `F` fit all cards, `0` reset zoom, `+/-` zoom, `M` show or hide the minimap, `D` the drawer, `T` light/dark, `/` the AI prompt and `?` the full list. `Ctrl+K` finds a card or column, `Ctrl+Z`/`Ctrl+Y` undo and redo a pipeline step, and `Esc` cancels a connection or closes a popover. Double-click a card title, a minimap rectangle or a search result to zoom to that card.

Options

`sideview.session_grace` Seconds the gadget waits after its last browser session closes before stopping (default `5`), so reloading the page does not end the app.

`sideview.llm_backend` `"ollama"` (default, or the `SIDEVIEW_LLM_BACKEND` variable) or `"openai"` for any OpenAI-compatible server.

`sideview.llm_model` Model name. Defaults to `SIDEVIEW_OLLAMA_MODEL` or `"llama3.2"` for Ollama, and `SIDEVIEW_OPENAI_MODEL` or `"gpt-4o-mini"` otherwise. `sv_llm_models()` lists what is available.

`sideview.llm_endpoint` Base URL. Defaults to `OLLAMA_HOST` or `http://localhost:11434`, and `OPENAI_BASE_URL` or `https://api.openai.com/v1`.

`sideview.llm_api_key` API key for OpenAI-compatible servers (default: the `OPENAI_API_KEY` variable).

`sideview.llm_think` `TRUE` lets Ollama reasoning models "think" before answering. Default `FALSE`, so they answer promptly.

Only schema summaries (column names, types and simple profiles) are sent to the model, never raw rows.

Examples

```
if (interactive()) {
  sideview(mtcars, iris)
}
```

`sideview_addin` *RStudio Addin for Sideview*

Description

Automatically discovers tabular datasets in the user's active environment and launches the Sideview interactive multi-table canvas gadget.

Usage

```
sideview_addin(
  env = .GlobalEnv,
  viewer = sv_default_viewer(),
  run = interactive()
)
```

Arguments

env	Environment to search for tables, and where **Run in R** stores its result (defaults to .GlobalEnv, where RStudio addins run).
viewer	Shiny viewer gadget display mode, or character shorthand ("browser", "dialog", or "pane"). Default is shiny::dialogViewer() (width 1400, height 900) in RStudio, shiny::browserViewer() in Positron (it has no dialog viewer; "dialog" there also opens the browser, with one message) and shiny::paneViewer() elsewhere.
run	Logical indicating whether to execute the gadget. Default interactive().

Value

Result of sideview(...), invisibly when the gadget was run.

---

sideview_server *Sideview Server Logic*

Description

Orchestrates bi-directional Shiny reactivity between the infinite 2D canvas and the R session. Handles field mapping (column_mapped), column renaming (column_renamed), discrepancy calculations (discrepancy_check), natural language AI prompts (run_ai_prompt), RStudio script injection (inject_script), and RStudio console execution (send_console).

Usage

```
sideview_server(
  cards = list(),
  ...,
  llm_fn = sv_call_llm,
  insert_fn = sv_insert_code,
  console_fn = sv_send_to_console,
  gadget = FALSE,
  run_env = parent.frame()
)
```

Arguments

cards	A sideview_card object, a list of sideview_card objects, or tabular objects to be normalized.
...	Additional arguments.
llm_fn	Function used to execute LLM queries (default sv_call_llm).
insert_fn	Function used to insert code into active RStudio document (default sv_insert_code).
console_fn	Function used to send code to RStudio console (default sv_send_to_console).

gadget	If TRUE, run as a gadget: the Done and Cancel buttons, closing the RStudio dialog, and closing the last browser tab all stop the app. After the last tab closes the app waits getOption("sideview.session_grace", 5) seconds so a page reload does not end it. Leave FALSE when embedding the server in your own multi-user Shiny app.
run_env	Environment where **Run in R** evaluates the pipeline and stores its result (as sv_result unless another name is given). Defaults to the environment sideview_server() is called from.

Value

A Shiny server function function(input, output, session).

---

sideview_ui *Sideview User Interface*

Description

Builds the complete 2D infinite canvas user interface for Sideview, containing the draggable table cards, panzoom canvas, SVG connector overlay, action drawer, and natural language AI bar with all HTML dependencies attached.

Usage

```
sideview_ui(
  cards = list(),
  title = "Sideview Canvas",
  gadget = FALSE,
  theme = "auto",
  state = NULL
)
```

Arguments

cards	A sideview_card object, a list of sideview_card objects, or tabular objects to be normalized.
title	Display title for the application toolbar. Default is "Sideview Canvas".
gadget	If TRUE, add Done and Cancel buttons to the toolbar (used by sideview()).
theme	"auto", "light", "dark", an sv_theme() or a bslib::bs_theme(). See sv_theme().
state	Optional saved workspace to restore on load (see sideview()).

Value

A shiny.tag.list UI definition.

sideview_widget *Create a Standalone Sideview HTMLWidget*

Description

Normalizes passed table objects (`data.frame`, `tibble`, `table`, `gt`, `flextable`, `htmlwidgets`) and creates an interactive 2D infinite canvas `htmlwidget` suitable for rendering in R Markdown, Quarto, Shiny, or exporting via `save_sideview()`.

Usage

```
sideview_widget(
  ...,
  width = NULL,
  height = NULL,
  elementId = NULL,
  options = list(),
  theme = "auto",
  state = NULL,
  max_rows = 5000
)
```

Arguments

`...`	Arbitrary table objects (`data.frame`, `tibble`, `table`, `gt`, `flextable`, `htmlwidgets`), named or unnamed, or a single list of tables.
`width`	Optional widget width (e.g. `"100%"`).
`height`	Optional widget height (e.g. `"600px"`).
`elementId`	Optional HTML element identifier.
`options`	Named list of canvas options (e.g. `snapToEdge`, `snapThreshold`, `minScale`, `maxScale`).
`theme`	Canvas theme: `"auto"` (default; follows the reader's `prefers-color-scheme`), `"light"`, `"dark"`, an `sv_theme()` or a `bslib::bs_theme()`.
`state`	A saved workspace to restore (see `sideview()`): a `sideview_state` or the path to a saved state file.
`max_rows`	Most rows of each data-frame table to embed in the widget (default 5000). A longer table is cut to its first `max_rows` rows, its card says "First N of M rows", and one message names the tables cut. The widget has no R session behind it, so every row would travel in the HTML. Use `Inf` to embed everything. Lazy (arrow / duckdb) tables already carry only a preview and are not affected.

Value

An S3 object of class `c("sideview", "htmlwidget")`.

Examples

```
if (interactive()) {
  sideview_widget(mtcars, iris)
}
```

`sideviewOutput` *Shiny Output Container for Sideview Widget*

Description

Shiny Output Container for Sideview Widget

Usage

```
sideviewOutput(outputId, width = "100%", height = "600px")
```

Arguments

`outputId`	Output variable to read from.
`width, height`	Must be a valid CSS unit (e.g. `'100%'`, `'600px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.

Value

A `shiny.tag` object representing the widget container.

`sideviewProxy` *Reach the proxy of a canvas from elsewhere in the session*

Description

Returns the same proxy `sideviewServer(id)` returned, so a canvas started in one module can be driven from another. Inside a nested module, `id` is relative to the calling module, as for `moduleServer()`.

Usage

```
sideviewProxy(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

`id`	The id given to `sideviewServer()`.
`session`	The Shiny session (default: the current one).

Value

A `sideview_proxy`.

Examples

```
if (interactive()) {
  # inside a server function:
  # sv_replace_card(sideviewProxy("review"), "labs", labs_now())
}
```

`sideviewUI` *Sideview canvas as a Shiny module*

Description

`sideviewUI()` and `sideviewServer()` put an R-backed sideview canvas into your own Shiny app (several per page). Row sorting and filtering, lazy arrow/duckdb cards, the join check and the column inspector all run in R, as in `sideview()`. Tables are added, replaced and removed from the server with the verbs `sv_add_card()`, `sv_replace_card()` and `sv_remove_card()`; the user's work is read back with the reactives of the returned proxy.

Usage

```
sideviewUI(id, height = "600px", title = NULL, theme = "auto")

sideviewServer(id, cards = list(), state = NULL, llm_fn = NULL)
```

Arguments

`id`	Module id.
`height`	CSS height of the canvas, e.g. `"600px"`.
`title`	Text shown in the toolbar; `NULL` (the default) shows none.
`theme`	`"auto"`, `"light"`, `"dark"`, an `sv_theme()` or a `bslib::bs_theme()`.
`cards`	Tables for the canvas: a (named) list as `sideview()` takes, or a **reactive** returning one. A reactive is compared with the canvas by name each time it fires: new names are added, names that are gone are removed, and names still there are replaced unless the table is `identical()` to the one on the canvas.
`state`	A saved workspace applied once after the first cards arrive: a state object, or its JSON text (a string whose first non-blank character is an opening curly brace).
`llm_fn`	Function used for the AI bar and the copilot (as for `sideview()`). With none, the server tells the canvas to hide the AI bar and the Copilot tab. `sideviewUI()` always renders them, so they show for a moment before the first message arrives.

Details

Code execution is off in a module (Insert Script, Run & Exit, Run in R and Run Console are not rendered, and their inputs are ignored by the server). The composed `$code()` is built in the browser: treat it as untrusted text to show, save or review, never `eval()` it unattended.

Value

`sideviewUI()` returns a `shiny.tag.list`. `sideviewServer()` returns a `sideview_proxy`: a list with the reactives `code()` (the composed code, `""` for none), `pipeline()` (the typed steps as a list), `state()` (a state object, or `NULL` before the first change) and `cards()` (the card names), plus `id`, `ns` and `session`. The values update after the user's changes settle (300 ms).

One server per id

Calling `sideviewServer()` twice with the same `id` in one session replaces the first canvas's entry in the proxy registry: `sideviewProxy()` then reaches the second. Give each canvas its own id.

See Also

`sideviewProxy()`, `sv_add_card()`

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(sideviewUI("cv"), verbatimTextOutput("code"))
  server <- function(input, output, session) {
    px <- sideviewServer("cv", cards = list(cars = mtcars, flowers = iris))
    output$code <- renderText(px$code())
  }
  shinyApp(ui, server)
}
```

---

sv_add_card *Add, replace or remove a card, set the workspace or highlight columns*

Description

Verbs for a canvas started with `sideviewServer()`. Each updates the R side first and then tells the browser, and returns the proxy invisibly so they pipe.

Usage

```
sv_add_card(proxy, x, name)

sv_replace_card(proxy, name, x)

sv_remove_card(proxy, name)

sv_set_state(proxy, state)

sv_highlight(proxy, name, cols)
```

Arguments

proxy	A sideview_proxy from sideviewServer() or sideviewProxy().
x	A table: anything sideview() accepts.
name	Card name (a single string).
state	A state object or its JSON text (e.g. proxy$state()); text is recognised by its leading curly brace.
cols	Column names (a character vector); columns not on the card are ignored.

Details

- sv_add_card() adds an auto-placed card; a name already on the canvas is an error.
- sv_replace_card() keeps the card's place and size and swaps its data. Pipeline steps that name a column the new table lacks are **disabled, not deleted**, and one toast says so.
- sv_remove_card() removes a card and disables the steps that used it.
- sv_set_state() replaces layout, joins and pipeline, as the Open button does.
- sv_highlight() highlights columns of a card, as a header click would.

Value

proxy, invisibly.

See Also

sideviewServer()

Examples

```
if (interactive()) {
  # px <- sideviewProxy("review")
  # sv_replace_card(px, "labs", labs_now()); sv_highlight(px, "labs", "lab_date")
}
```

---

sv_auto_clean_rules *Generate tidyverse Code for Automatic Discrepancy Cleaning and Crosswalks*

Description

Inspects discrepancy detection output or pairwise column discrepancies (exact matches, fuzzy near-matches, and missing levels), automatically synthesizing clean recoding rules using dplyr::case_when(). Handles casing differences, leading/trailing whitespace discrepancies, and fuzzy string distance variations (Levenshtein distance <= 2).

Usage

```
sv_auto_clean_rules(
  discrepancy_result,
  df_name = "df",
  col = "status",
  default = col,
  pipe = FALSE,
  lazy = FALSE,
  df_is_code = FALSE
)

sv_code_auto_clean(
  discrepancy_result,
  df_name = "df",
  col = "status",
  default = col,
  pipe = FALSE,
  lazy = FALSE,
  df_is_code = FALSE
)
```

Arguments

discrepancy_result	A discrepancy result list (such as returned by `sv_detect_discrepancies()`) or an object containing `fuzzy_matches`, `only_in_x`, `only_in_y`, and `exact_matches`.
df_name	Name of the data frame to mutate. Defaults to `"df"`. Can be `NULL` for piped or standalone expressions.
col	Column name being recoded. Defaults to `"status"`. If `NULL`, attempts to extract from `discrepancy_result$from_col` or defaults to `"status"`.
default	Fallback value for unmatched records. If equal to `col`, preserves unmatched records via `.default = <col>`. Set to `NULL` to omit the default clause.
pipe	Logical; if TRUE, emits piped syntax `df %>% dplyr::mutate(...)`.
lazy	Logical; TRUE when `df_name` is an arrow or duckdb table. The code then ends in `dplyr::collect()` (the one collect of a lazy pipeline), the recode closes with `TRUE ~ <col>` (arrow cannot take `.default = <col>`), and `df_name` may be an expression such as `dplyr::tbl(con, "visits")` when `df_is_code` is TRUE. The matching of values is done in R on the values compared, so the generated code uses no `trimws()` or `tolower()`.
df_is_code	Logical; TRUE when `df_name` is R code (such as `dplyr::tbl(con, "visits")`) to use as written. FALSE (the default) treats it as a table name and backtick-quotes it when it is not syntactic, as in `visits (2020)`. Only used when `lazy` is TRUE.

Value

A character string of tidyverse code, or `NULL` if no discrepancies require cleaning.

Examples

```
disc <- sv_detect_discrepancies(c("male", "Female"), c("Male", "Female"))
sv_auto_clean_rules(disc, df_name = "patients", col = "gender")
```

`sv_build_chat_prompt` *Build Structured Multi-Turn Prompt for Conversational Copilot*

Description

Formats a multi-turn conversational prompt for the Sideview AI Copilot drawer. Preserves database/table schema context, active pipeline R code, and previous dialogue turns (user/assistant). Instructs the model to output updated executable tidyverse R code within code fences.

Usage

```
sv_build_chat_prompt(
  user_message,
  history = list(),
  current_code = NULL,
  schema_context = NULL
)
```

Arguments

`user_message`	Natural language query or instruction from the user.
`history`	A list of previous conversation turns, where each turn is a list containing `role` ("user" or "assistant") and `content` (or `text`/`message`).
`current_code`	Optional character string containing the currently active pipeline code in the drawer.
`schema_context`	Optional privacy-safe schema string built via `sv_build_schema_context`.

Value

A list of class `c("sideview_chat_prompt", "sideview_prompt", "list")` containing `system_message`, `user_message`, `history`, `messages`, and `prompt`.

sv_build_prompt *Build Structured Prompt for Tidyverse LLM Code Generation*

Description

Formats system instructions emphasizing pure, executable tidyverse R code enclosed in markdown code fences (```r ... ```) and packages user task description alongside privacy-safe table schemas.

Usage

```
sv_build_prompt(user_prompt, schema_context = NULL)
```

Arguments

user_prompt	Natural language description of desired data transformation.
schema_context	Optional privacy-safe schema string built via sv_build_schema_context.

Value

A list of class sideview_prompt containing system_message, user_message, and prompt.

sv_build_schema_context *Build a Privacy-Safe Schema Context from Sideview Cards*

Description

Summarizes the metadata of sideview cards (table names, dimensions, column names, types, value ranges for numerics/dates, sample levels for factors, and unique samples for characters) without leaking full raw row data or records. Non-tabular widget cards and NULL data are handled safely.

Usage

```
sv_build_schema_context(cards, max_categories = 5L)
```

Arguments

cards	A sideview_card, a list of sideview_cards, or tabular data.
max_categories	Maximum sample unique categories to report for categorical columns. Default is 5.

Value

A character string containing the privacy-safe schema summary.

sv_call_llm *Call Local Ollama or OpenAI-Compatible LLM Endpoint*

Description

Sends prompt requests to local Ollama (`http://localhost:11434`) or OpenAI-compatible endpoints via `httr2`. Supports model selection, customizable endpoints, API key authentication, and clean error reporting for network or HTTP status failures.

Usage

```
sv_call_llm(
  prompt,
  backend = c("ollama", "openai"),
  model = NULL,
  endpoint = NULL,
  api_key = NULL,
  timeout = 60,
  extract_code = FALSE,
  resolve_model = TRUE,
  ...
)
```

Arguments

`prompt`	A character string or `sideview_prompt` object from `sv_build_prompt`.
`backend`	Character string: either `"ollama"` or `"openai"`.
`model`	Model name (e.g., `"llama3.2"`, `"llama3"`, `"gpt-4o-mini"`).
`endpoint`	Base URL or full chat endpoint for the API.
`api_key`	API key (for OpenAI or authenticated endpoints).
`timeout`	Request timeout in seconds. Default 60.
`extract_code`	Logical; if TRUE, runs `sv_extract_code()` on the response text. Default FALSE.
`resolve_model`	Logical; for Ollama, if an untagged model such as `"llama3.2"` is not installed, retry once with an installed tag of it (e.g. `"llama3.2:3b"`). Default TRUE.
`...`	Additional options added to the request body (e.g. `temperature`). For Ollama, `think = FALSE` is sent unless you pass `think` or set `options(sideview.llm_think = TRUE)`, so reasoning models answer promptly.

Value

Character string containing the LLM response text (or extracted code if `extract_code = TRUE`).

sv_code_arrange *Generate tidyverse Code for Arranging (Sorting) Rows*

Description

Generate tidyverse Code for Arranging (Sorting) Rows

Usage

```
sv_code_arrange(df_name, col, desc = FALSE, pipe = FALSE)
```

Arguments

df_name	Name of the data frame.
col	Character vector of column name(s) to sort by.
desc	Logical vector indicating descending order for each column. Recycled to match the length of col. Defaults to FALSE.
pipe	Logical; if TRUE, uses pipe syntax df %>% dplyr::arrange(...).

Value

A character string of R code.

sv_code_filter *Generate tidyverse Code for Filtering Rows*

Description

Generate tidyverse Code for Filtering Rows

Usage

```
sv_code_filter(df_name, condition, pipe = FALSE)
```

Arguments

df_name	Name of the data frame.
condition	Character vector of boolean filter expressions.
pipe	Logical; if TRUE, uses pipe syntax df %>% dplyr::filter(...).

Value

A character string of R code.

sv_code_group_summarize *Generate tidyverse Code for Grouping and Aggregating Data*

Description

Constructs syntactically valid `dplyr::group_by()` and `dplyr::summarise()` pipelines with identifier escaping, customizable summary metrics, and reproducible grouping behavior via `.groups = 'drop'`.

Usage

```
sv_code_group_summarize(
  df_name = NULL,
  group_cols,
  summary_exprs,
  pipe = FALSE
)
```

Arguments

`df_name`	Name of the data frame (optional; can be NULL for intermediate piped pipeline segments).
`group_cols`	Character vector of grouping column name(s).
`summary_exprs`	Named character vector or named list of aggregation expressions (e.g., `c(total = "dplyr::n()", avg = "mean(val, na.rm = TRUE)")`).
`pipe`	Logical; if TRUE, uses pipe syntax `df %>% dplyr::group_by(...) %>% dplyr::summarise(...)`. Defaults to `FALSE`.

Value

A character string of R code.

sv_code_join *Generate tidyverse Code for Relational Table Joins*

Description

Generate tidyverse Code for Relational Table Joins

Usage

```
sv_code_join(df1, df2, col1, col2 = col1, join_type = "left", pipe = FALSE)
```

Arguments

df1	Name of the first data frame (optional / can be NULL when constructing intermediate piped join segments).
df2	Name of the second data frame.
col1	Key column name(s) in df1.
col2	Key column name(s) in df2 (defaults to col1).
join_type	Type of join: "left", "inner", "anti", "right", "full", "semi".
pipe	Logical; if TRUE, uses pipe syntax `df1 %>% dplyr::<join_type>(df2, ...)`.

Value

A character string of R code.

---

sv_code_mutate *Generate tidyverse Code for Column Mutation*

Description

Generate tidyverse Code for Column Mutation

Usage

```
sv_code_mutate(df_name, new_col, expr, pipe = FALSE)
```

Arguments

df_name	Name of the data frame.
new_col	Name of the column(s) being created or transformed.
expr	R expression string(s) to evaluate.
pipe	Logical; if TRUE, uses pipe syntax `df %>% dplyr::mutate(...)`.

Value

A character string of R code.

sv_code_pivot_longer *Generate tidyverse Code for Reshaping Data from Wide to Long*

Description

Constructs syntactically valid `tidyr::pivot_longer()` calls with proper identifier escaping and support for both piped workflows and direct calls.

Usage

```
sv_code_pivot_longer(
  df_name = NULL,
  cols,
  names_to = "name",
  values_to = "value",
  pipe = FALSE
)
```

Arguments

`df_name`	Name of the data frame (optional; can be NULL for piped workflows).
`cols`	Character vector of column name(s) to pivot into longer format.
`names_to`	Character string specifying the name of the column to create from the data stored in the column names of `cols`. Defaults to `"name"`.
`values_to`	Character string specifying the name of the column to create from the data stored in cell values. Defaults to `"value"`.
`pipe`	Logical; if TRUE, uses pipe syntax `df %>% tidyr::pivot_longer(...)`. Defaults to `FALSE`.

Value

A character string of R code.

sv_code_pivot_wider *Generate tidyverse Code for Reshaping Data from Long to Wide*

Description

Constructs syntactically valid `tidyr::pivot_wider()` calls with proper identifier escaping and optional aggregation functions.

Usage

```
sv_code_pivot_wider(
  df_name = NULL,
  names_from,
  values_from,
  values_fn = NULL,
  pipe = FALSE
)
```

Arguments

df_name	Name of the data frame (optional; can be NULL for piped workflows).
names_from	Character vector of column name(s) to get the name of the output column from.
values_from	Character vector of column name(s) to get the cell values from.
values_fn	Optional function name (character string or named vector/list) to apply to values. Defaults to NULL.
pipe	Logical; if TRUE, uses pipe syntax df %>% tidyr::pivot_wider(...). Defaults to FALSE.

Value

A character string of R code.

---

sv_code_recode *Generate tidyverse Code for Categorical Recoding and Discrepancy Crosswalks*

Description

Generate tidyverse Code for Categorical Recoding and Discrepancy Crosswalks

Usage

```
sv_code_recode(
  df_name,
  col,
  crosswalk,
  default = NULL,
  method = c("case_when", "case_match", "fct_recode"),
  pipe = FALSE,
  lazy = FALSE
)
```

Arguments

df_name	Name of the data frame.
col	Column name to recode.
crosswalk	Mapping between source and target values. Can be: • A data.frame with source values in column 1 and target values in column 2. • A named character vector (names = source values, values = target values). • A named list where names are target values and elements are vectors of source values.
default	Fallback value for unmatched records. If NULL, omitted. If equal to col, emits unquoted column name .default = <col>.
method	Recoding function: "case_when" (dplyr, the default), "case_match" (dplyr; deprecated in dplyr 1.2.0, kept for older pipelines) or "fct_recode" (forcats).
pipe	Logical; if TRUE, uses pipe syntax df %>% dplyr::mutate(...).
lazy	Logical; TRUE for a table that lives in arrow or duckdb. case_when() then closes with a TRUE ~ <col> arm instead of .default = <col>, which arrow cannot take.

Value

A character string of R code.

---

sv_code_rename *Generate tidyverse Code for Column Renaming*

Description

Generate tidyverse Code for Column Renaming

Usage

```
sv_code_rename(df_name, old_col, new_col, pipe = FALSE)
```

Arguments

df_name	Name of the data frame.
old_col	Existing column name(s).
new_col	New column name(s).
pipe	Logical; if TRUE, uses pipe syntax df %>% dplyr::rename(...).

Value

A character string of R code.

sv_code_select *Generate tidyverse Code for Selecting Columns*

Description

Generate tidyverse Code for Selecting Columns

Usage

```
sv_code_select(df_name, cols, pipe = FALSE)
```

Arguments

df_name	Name of the data frame.
cols	Character vector of column names to select.
pipe	Logical; if TRUE, uses pipe syntax `df %>% dplyr::select(...)`.

Value

A character string of R code.

sv_codebook *Codebook (Data Dictionary)*

Description

One row per column across one or more tables: type, label, missingness, distinct values, whether the column is a unique key, range and quartiles, mean and SD, the most common values and data-quality flags. Write it to CSV or Excel to document a dataset or check it before analysis.

Usage

```
sv_codebook(..., top_n = 5L, file = NULL, progress = NULL)
```

Arguments

...	Data frames, duckdb tables or Arrow tables, or one list of them. Names default to the argument expressions, as in `sideview()`. A duckdb or Arrow table is profiled in full inside the database; it is never collected.
top_n	Number of most common values listed in `top_values`.
file	Optional path ending in `.csv` or `.xlsx` (needs the `writexl` package). The codebook is still returned, invisibly.
progress	Optional `function(table, j, n_cols)`, called before each column is profiled.

Value

A data frame with columns `table`, `column`, `position`, `class`, `label`, `n`, `rows_profiled`, `n_missing`, `pct_missing`, `n_distinct`, `unique_key`, `min`, `q1`, `median`, `q3`, `max` (as text, so dates fit), `mean`, `sd`, `top_values` and `issues`. `rows_profiled` is the number of rows the profile used; it equals `n` except for a preview of a larger table.

Duckdb and Arrow tables

Each column of a duckdb or Arrow table is profiled in full by duckdb, one column at a time, so `n` and `rows_profiled` are the table's row count, and a very wide table takes longer rather than more memory; use `progress` to report on it. In the gadget, a card that only has a preview (duckdb is not installed, or the link to the table was lost) is profiled from its first 50 rows: `n` is still the true row count, `rows_profiled` is 50, and the other columns describe those rows.

Examples

```
patients <- data.frame(id = 1:4, sex = c("M", "F", "f", NA), age = c(40, 51, 62, 999))
sv_codebook(patients)
visits <- data.frame(id = c(1, 1, 2, 5), sbp = c(120, 135, 128, 141))
out <- tempfile(fileext = ".csv")
sv_codebook(patients, visits, file = out)
unlink(out)
```

`sv_column_profile` *Column Profile*

Description

Summarises one column for the column inspector: counts, missingness, distinct values, whether it could serve as a unique key, quartiles and a histogram for numbers, the most common values for text, and data-quality flags (stray spaces, values that differ only in case, numbers stored as text, empty strings, extreme values).

Usage

```
sv_column_profile(x, top_n = 10L, bins = 20L, column = NULL)
```

Arguments

`x`	A vector (a data frame column), or a lazy table (a duckdb table or an Arrow table / dataset) together with `column`.
`top_n`	Number of most common values to keep.
`bins`	Number of equal-width histogram bins for numeric columns.
`column`	The column to profile when `x` is a lazy table. The whole table is profiled by the database, and the result equals the one for the same column held in memory. Not used for a vector.

Value

A list with `kind` (`"numeric"`, `"character"`, `"logical"`, `"date"`, `"datetime"` or `"other"`), `class`, `label` (the `"label"` attribute, if any), `n`, `n_missing`, `pct_missing`, `n_distinct` (non-missing), `unique_key`, `stats` (named list: `min`, `q1`, `median`, `q3`, `max`, and `mean`, `sd` for numbers), `histogram` (`breaks`, `counts`), `top` (data frame of `value`, `n`), `n_other` (distinct values not in `top`) and `issues` (character).

Duckdb and Arrow tables

For a duckdb table or an Arrow table, give the table as `x` and the column name as `column`. The database profiles the whole table in a few queries and only the summary comes back; the result equals the one for the same column held in memory. Arrow tables are profiled through a private duckdb connection that is closed afterwards. duckdb ordered-factor columns report class `"factor"`.

Examples

```
sv_column_profile(c(" Male", "male", "Female", NA))
sv_column_profile(c(54, 61, 47, 999, NA))$issues
```

---

sv_detect_discrepancies

Detect Exact Matches, Discrepancies, and Fuzzy Near-Matches

Description

Compares distinct values between two vectors (e.g. columns mapped across tables), computing exact matches, values only in x, values only in y, fuzzy/near-matches using Levenshtein distance (adist), and comprehensive match statistics.

Usage

```
sv_detect_discrepancies(x, y, max_distance = 2)
```

Arguments

`x`	Vector of values from the first column.
`y`	Vector of values from the second column.
`max_distance`	Maximum Levenshtein edit distance for fuzzy matching (default 2).

Value

A list with elements:

- `exact_matches`: character vector of values present in both x and y.
- `fuzzy_matches`: data.frame with columns `x`, `y`, `distance`.
- `only_in_x`: character vector of values present only in x.

- `only_in_y`: character vector of values present only in y.
- `stats`: list of match statistics (counts, NA counts, match rate).
- `match_stats`: alias to `stats`.

`sv_escape_ident` *Escape R Identifiers with Backticks If Non-Syntactic*

Description

Escape R Identifiers with Backticks If Non-Syntactic

Usage

```
sv_escape_ident(x)
```

Arguments

`x`	Character vector of identifier names.

Value

Character vector of identifiers, escaped with backticks if necessary.

`sv_extract_code` *Extract Executable R Code from LLM Response Text*

Description

Parses code fence blocks (```` ```r ... ``` ````, ```` ```R ... ``` ````, ```` ``` ... ``` ````) out of LLM responses. If multiple fences exist, concatenates them. If no fence is found, falls back to the trimmed raw text. Strips extraneous whitespace and returns valid R code.

Usage

```
sv_extract_code(response_text)
```

Arguments

`response_text`	Character vector containing the LLM response text.

Value

A character string of extracted R code.

sv_find_tables *Find Tabular Data Objects in an Environment*

Description

Scans an environment for tabular objects, including `data.frame`, `tibble`, `matrix`, `table`, `gt`, `flextable`, and `sideview_card` objects.

Usage

```
sv_find_tables(env = parent.frame())
```

Arguments

`env`	Environment to scan (defaults to `parent.frame()`).

Value

A named list of tabular objects found in the environment.

sv_insert_code *Insert Generated Code into the Active RStudio Editor*

Description

Inserts tidyverse pipelines or generated code directly into the user's active document buffer at the current cursor location via `rstudioapi::insertText()`. If RStudio is unavailable, running headless, or if no document is currently open, falls back gracefully by returning a structured status list containing the code.

Usage

```
sv_insert_code(code, location = NULL, ...)
```

Arguments

`code`	Character vector or scalar string containing the code to insert. Multiple elements are joined by newline characters.
`location`	Optional cursor position or range for `rstudioapi::insertText()`. Defaults to `NULL` (current selection/cursor).
`...`	Additional arguments passed to `rstudioapi::insertText()`.

Value

A named list with:

success	Logical indicating whether insertion into editor succeeded.
destination	Character "editor" on success, "fallback" on fallback/error.
code	Character string of the normalized code.
message	Character description of the result or error.

Examples

```
res <- sv_insert_code("library(dplyr)")
```

---

sv_is_rstudio_available *Check if RStudio API is Available*

Description

Safely tests whether the RStudio API is running and operational in the current session without raising errors in headless, testing, or CI environments.

Usage

```
sv_is_rstudio_available()
```

Value

Logical scalar (TRUE if RStudio is running and accessible, FALSE otherwise).

Examples

```
sv_is_rstudio_available()
```

---

sv_join_diagnostics *Join Diagnostics*

Description

Checks a join before you run it: how many rows match in each direction, the key relationship (one-to-one, one-to-many, many-to-many), how many rows each join type returns, missing and duplicated keys, a sample of unmatched keys, and fixes that would match more rows, each measured on its own (trimming whitespace, ignoring case, dropping leading zeros). Keys are compared the way dplyr joins compare them by default, so NA matches NA.

Usage

```
sv_join_diagnostics(
  x,
  y,
  by,
  sample_n = 5L,
  x_name = "x",
  y_name = "y",
  rule = NULL
)
```

Arguments

x, y	Tables to join (x is the left table): data frames, or duckdb tables and Arrow tables, which are checked in full without loading them into R.
by	Join columns: a character vector as in dplyr::left_join(), where names are columns of x and values columns of y (c("id", "site" = "site_code")). Unnamed entries use the same name on both sides.
sample_n	Number of unmatched keys to list for each side.
x_name, y_name	Table names used in messages.
rule	Optional date rule for the join, a list with the fields of a join step's rule: shape ("between", "window", "before", "after" or "overlaps") and its columns (xCol, yCol, yFrom, yTo, xFrom, xTo), a window's whole-number from and to (days, -3650 to 3650) and, for "before"/"after", inclusive. The check then counts the rows the rule keeps, with na_matches = "never": under a rule a missing key matches nothing. A rule that does not hold (see the message) is an error.

Details

The canvas runs this whenever two columns are connected and shows the result on the join step and in the Discrepancy tab.

Value

An object of class sv_join_diagnostics: a list with by (data frame of x/y columns), n_x, n_y, distinct_x, distinct_y, distinct_matched, matched_x, matched_y (rows with a partner), match_rate_x, match_rate_y, na_x, na_y (rows with a missing key), dup_keys_x, dup_keys_y (keys that occur more than once), relationship, rows (named list of row counts per join type), unmatched_x, unmatched_y (sample keys), type_issues, hints (list of fix, label, extra_x, extra_y), strict_args, warnings and status ("ok", "warn" or "error").

strict_args is a named list with an entry per join type (left_join, inner_join, right_join, full_join) holding the arguments a Strict join writes (relationship, and unmatched = "error" when every row of the checked side has a partner); an entry is absent when Strict would write nothing for that type (no matches; right and full joins under a nearest-match rule). With a rule the result also has rule, rule_kind (the kind of the rule's first x column), rule_dropped_x and rule_dropped_y (rows that have a key partner but no pair that satisfies the rule), and rows$right_join / rows$full_join are NA under a nearest-match rule.

Duckdb and Arrow tables

When `x` or `y` is a duckdb table or an Arrow table, the check runs inside duckdb on the whole table and only the counts come back into R; the result equals the one for the same data held in memory. If the tables are on two different duckdb connections the check does not run. Arrow tables are handed to duckdb with `arrow::to_duckdb()`, and a data frame is registered temporarily; both are removed afterwards. Known differences from the in-memory check: keys of different types are compared as text, double keys are compared exactly, and an in-memory POSIXct key is compared as text where a database key is compared as an instant. A date rule runs in duckdb too; its date-time columns are compared as UTC instants (taken with `epoch_us()`, so the session `TimeZone` and the ICU extension play no part), and a date-time window adds whole days of 24 hours, as the in-memory check does.

Examples

```
patients <- data.frame(id = c(1, 2, 3, 4), age = c(40, 51, 62, 33))
visits <- data.frame(id = c(1, 1, 2, 5), visit = 1:4)
sv_join_diagnostics(patients, visits, by = "id")
```

`sv_llm_models` *List the models available from an LLM backend*

Description

Queries a local Ollama server (`/api/tags`) or an OpenAI-compatible server (`/models`) for the installed or available model names. Useful for choosing `options(sideview.llm_model = ...)`.

Usage

```
sv_llm_models(
  backend = c("ollama", "openai"),
  endpoint = NULL,
  api_key = NULL,
  timeout = 5
)
```

Arguments

`backend`	`"ollama"` or `"openai"`.
`endpoint`	Base URL. Defaults as in `sv_call_llm()`; a chat endpoint (`.../api/chat`, `.../chat/completions`) is also accepted.
`api_key`	API key for OpenAI-compatible servers.
`timeout`	Request timeout in seconds. Default 5.

Value

A character vector of model names.

`sv_load_state` *Load Canvas State from File*

Description

Deserializes a previously saved Sideview canvas workspace from a `.json` or `.rds` file and returns a `sideview_state` object. Optionally launches an interactive session.

Usage

```
sv_load_state(file, tables = NULL, launch = FALSE, format = NULL, ...)
```

Arguments

`file`	Path to the saved state file (`.json` or `.rds`).
`tables`	Optional named list of data frames or tables to associate with loaded cards. If `NULL`, the tables named by the saved cards are looked up in the calling environment (then every table there, as `sv_find_tables()` finds them).
`launch`	Logical; if `TRUE` and the session is interactive, launches `sideview(tables, state = <loaded state>, .` so cards, zoom, joins and the pipeline come back. If non-interactive, returns the loaded state object.
`format`	Optional format specification (`"json"` or `"rds"`). If `NULL` (default), the format is inferred from the file extension (`.json` or `.rds`).
`...`	Further arguments for `sideview()` when `launch = TRUE` (e.g. `viewer`).

Value

A `sideview_state` object of class `c("sideview_state", "list")`, or the gadget result if `launch = TRUE` in an interactive session.

Examples

```
st <- new_sideview_state()
tf <- tempfile(fileext = ".json")
sv_save_state(st, tf)
loaded <- sv_load_state(tf)
print(loaded)
unlink(tf)
```

sv_normalize *Universal Normalization Engine for Sideview Cards*

Description

Converts various R table formats (data.frame, tibble, table, gt, flextable, htmlwidgets) into a standardized `sideview_card` S3 object for rendering on the 2D infinite canvas.

Usage

```
sv_normalize(x, name = NULL, id = NULL, ...)
```

Arguments

`x`	An object to normalize (data.frame, tibble, table, gt, flextable, htmlwidget).
`name`	Optional display name for the card.
`id`	Optional unique identifier for the card.
`...`	Additional arguments passed to methods.

Value

An object of class `sideview_card`.

sv_save_state *Save Canvas State to File*

Description

Serializes the active Sideview canvas state (cards, positions, filters, connections, and pipeline code) to either `.json` or `.rds` format.

Usage

```
sv_save_state(state, file, format = NULL)
```

Arguments

`state`	A `sideview_state` object or structured state list.
`file`	Path to the destination file.
`format`	Optional format specification (`"json"` or `"rds"`). If `NULL` (default), the format is inferred from the file extension (`.json` or `.rds`).

Value

Invisibly returns the path of the saved file.

Examples

```
st <- new_sideview_state()
tf <- tempfile(fileext = ".json")
sv_save_state(st, tf)
unlink(tf)
```

`sv_send_to_console` *Send Generated Code to the RStudio Console*

Description

Sends R code to the RStudio console for immediate evaluation or user inspection. Falls back gracefully without crashing if RStudio is unavailable or running in a headless environment.

Usage

```
sv_send_to_console(code, execute = TRUE, ...)
```

Arguments

`code`	Character vector or scalar string containing the code to execute. Multiple elements are joined by newline characters.
`execute`	Logical indicating whether to execute the code immediately (`TRUE`) or simply populate the console line without executing (`FALSE`). Default `TRUE`.
`...`	Additional arguments passed to `rstudioapi::sendToConsole()`.

Value

A named list with:

`success`	Logical indicating whether console transmission succeeded.
`destination`	Character `"console"` on success, `"fallback"` on fallback/error.
`code`	Character string of the normalized code.
`message`	Character description of the result or error.

Examples

```
res <- sv_send_to_console("1 + 1", execute = FALSE)
```

`sv_suggest_mappings` *Suggest Foreign Key Mappings Across Canvas Cards*

Description

Evaluates pairwise heuristic column relationships across loaded `sideview_card` objects or tabular data structures, computing a 3-part composite confidence score based on name similarity (40%), data type compatibility (20%), and Jaccard value overlap (40%).

Usage

```
sv_suggest_mappings(cards, ..., min_confidence = 0.6)
```

Arguments

`cards`	A list of `sideview_card` objects or tabular data frames. Alternatively, individual cards or tables can be passed via `...`.
`...`	Additional cards or tables passed as positional arguments.
`min_confidence`	Minimum confidence threshold (numeric between 0 and 1) for returned suggestions (default `0.6`).

Value

A `data.frame` with columns:

`from_table` Source table or card name.

`from_col` Source column name.

`to_table` Destination table or card name.

`to_col` Destination column name.

`confidence` Composite confidence score in `[0, 1]`.

`reason` Human-readable justification describing similarity components.

Examples

```
df1 <- data.frame(patient_id = 1:5, age = c(20, 30, 40, 50, 60))
df2 <- data.frame(patient_id = 3:7, score = c(10, 20, 30, 40, 50))
sv_suggest_mappings(list(patients = df1, visits = df2))
```

sv_theme *Canvas Theme*

Description

Builds a theme for `sideview()` and `sideview_widget()`: a light/dark mode plus optional overrides of the canvas design tokens (the `--sv-*` CSS custom properties in `inst/www/sideview.css`).

Usage

```
sv_theme(
  mode = c("auto", "light", "dark"),
  primary = NULL,
  font = NULL,
  font_mono = NULL,
  radius = NULL,
  ...
)
```

Arguments

`mode`	`"auto"` (default), `"light"` or `"dark"`. With `"auto"` the gadget follows the RStudio editor theme when it runs inside RStudio, and otherwise (and in the widget) follows the operating system's `prefers-color-scheme` setting. The toolbar button or the `T` key switches theme while the canvas is open.
`primary`	Accent colour for buttons, selections and splines (any CSS colour that `grDevices::col2rgb()` understands, e.g. `"#0d6efd"`). Hover, focus and tint shades are derived from it.
`font`, `font_mono`	Font stacks for text and code.
`radius`	Card corner radius (CSS length, e.g. `"4px"`).
`...`	Further token overrides by name, e.g. `canvas_bg = "#fafafa"` sets `--sv-canvas-bg`. Underscores become hyphens.

Value

An object of class `sv_theme`.

Examples

```
sv_theme("dark")
sv_theme(primary = "#0f766e", radius = "4px")
if (interactive()) sideview(mtcars, theme = sv_theme("dark", primary = "#f59e0b"))
```

Index

dplyr::left_join(), *32*

grDevices::col2rgb(), *38*

new_sideview_state, 3
new_sideview_state(), *6*

print.sideview_state, 4

renderSideview, 4

save_sideview, 5
sideview, 6
sideview(), *10, 11*, *13*, *15*, *26*, *34*, *38*
sideview_addin, 8
sideview_server, 9
sideview_ui, 10
sideview_widget, *4, 5*, 11
sideview_widget(), *5*, *38*
sideviewOutput, 12
sideviewProxy, 12
sideviewProxy(), *14, 15*
sideviewServer *(*sideviewUI*)*, 13
sideviewServer(), *12*, *14, 15*
sideviewUI, 13
sv_add_card, 14
sv_add_card(), *13, 14*
sv_auto_clean_rules, 15
sv_build_chat_prompt, 17
sv_build_prompt, 18
sv_build_schema_context, 18
sv_call_llm, 19
sv_call_llm(), *33*
sv_code_arrange, 20
sv_code_auto_clean
*(*sv_auto_clean_rules*)*, 15
sv_code_filter, 20
sv_code_group_summarize, 21
sv_code_join, 21
sv_code_mutate, 22
sv_code_pivot_longer, 23
sv_code_pivot_wider, 23
sv_code_recode, 24
sv_code_rename, 25
sv_code_select, 26
sv_codebook, 26
sv_column_profile, 27
sv_detect_discrepancies, 28
sv_escape_ident, 29
sv_extract_code, 29
sv_find_tables, 30
sv_find_tables(), *34*
sv_highlight *(*sv_add_card*)*, 14
sv_insert_code, 30
sv_is_rstudio_available, 31
sv_join_diagnostics, 31
sv_llm_models, 33
sv_llm_models(), *8*
sv_load_state, 34
sv_normalize, 35
sv_remove_card *(*sv_add_card*)*, 14
sv_remove_card(), *13*
sv_replace_card *(*sv_add_card*)*, 14
sv_replace_card(), *13*
sv_save_state, 35
sv_save_state(), *6*
sv_send_to_console, 36
sv_set_state *(*sv_add_card*)*, 14
sv_suggest_mappings, 37
sv_theme, 38
sv_theme(), *6*, *10, 11*, *13*