Package: palettome (via r-universe)

October 21, 2026

Title Hierarchy-Aware, Interactively Editable Color Palettes for Clusters

Version 1.0.1

Description Assigns perceptually-informed color palettes to clustered data (e.g. scRNA-seq clusters / cell subtypes from 'Seurat' or plain UMAP coordinate tables), where clusters that belong to the same ``family'' (e.g. subtypes of T cells) can be colored either harmoniously (shared hue, varying shade) or with maximal contrast (with family membership encoded via a secondary channel). Families can be auto-detected from embedding-space hierarchical clustering or supplied explicitly, and every generated color remains individually overridable, with manual overrides preserved across regeneration. Colors are constructed in the HCL space as in Zeileis et al. (2020) <doi:10.18637/jss.v096.i01>, separated using the CIEDE2000 color difference of Sharma, Wu and Dalal (2005) <doi:10.1002/col.20070>, and checked for color-vision deficiency using the simulation of Machado, Oliveira and Fernandes (2009) <doi:10.1109/TVCG.2009.113>. Core algorithms have no UI dependencies; an optional interactive 'shiny' UI is provided separately.

License Apache License (== 2.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports colorspace, farver, stats, graphics, utils, jsonlite

Suggests knitr, rmarkdown, testthat (>= 3.0.0), ggplot2, Seurat, shiny, plotly, colourpicker, shinyjs, dendextend

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/thisishcb/Palettome

BugReports https://github.com/thisishcb/Palettome/issues

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

Date/Publication 2026-10-20 19:57:32 UTC

RemoteUrl https://github.com/thisishcb/palettome

RemoteRef HEAD

RemoteSha 6683a9e84e971f5a33c7ee1c645fd25265701bb5

Contents

as_cluster_data . . . 2
cluster_colors . . . 4
color_distance . . . 5
compute_centroids . . . 5
compute_family_dendrogram . . . 6
cut_families . . . 7
detect_families . . . 8
downsample_stratified . . . 9
export_palette_csv . . . 10
export_palette_json . . . 11
family_colors . . . 11
generate_palette . . . 12
import_palette_json . . . 14
launch_palettome_ui . . . 15
optimize_color . . . 16
plot_dendrogram . . . 17
plot_palette_static . . . 18
plot_swatches . . . 19
reset_overrides . . . 20
set_manual_color . . . 20
simulate_cvd . . . 21

Index **23**

`as_cluster_data` *Build a palettome data object from a data frame or Seurat object*

Description

`as_cluster_data()` normalizes either a plain data frame of embedding coordinates + cluster labels, or a `Seurat` object, into the common `palettome_data` structure that every other core function (`detect_families()`, `generate_palette()`, `plot_palette_static()`, ...) consumes. This is the only place that needs to know about Seurat, and Seurat is never required to be installed to use the rest of the package.

Usage

```
as_cluster_data(x, ...)

## S3 method for class 'data.frame'
as_cluster_data(
  x,
  coord_cols,
  cluster_col,
  family_col = NULL,
  cell_id_col = NULL,
  ...
)

## S3 method for class 'Seurat'
as_cluster_data(
  x,
  reduction = "umap",
  dims = 1:2,
  cluster_col = "seurat_clusters",
  family_col = NULL,
  ...
)
```

Arguments

`x`	A `data.frame` (or object coercible to one) or a `Seurat` object.
`...`	Passed to methods.
`coord_cols`	Character vector of 2 or 3 column names holding embedding coordinates (e.g. `c("UMAP_1", "UMAP_2")`).
`cluster_col`	Name of the column holding cluster/subtype labels.
`family_col`	Optional name of a column holding an explicit parent/family label. When `NULL` (default), families are left undetected and must be filled in by `detect_families()`.
`cell_id_col`	Optional name of a column with unique per-row/cell identifiers. When `NULL`, row numbers are used.
`reduction`	Name of the dimensional reduction to pull coordinates from (passed to `Seurat::Embeddings()`), e.g. `"umap"`.
`dims`	Integer vector of length 2 or 3 selecting which dimensions of `reduction` to use.

Value

A `palettome_data` object: a list with elements

cells A data frame with columns `cell_id`, the coordinate columns (as named in `coord_cols`), `cluster`, and optionally `family`.

coord_cols Character vector of coordinate column names in `cells`, in order (length 2 or 3).

has_family Logical; whether an explicit family column was supplied.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)

pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
str(pdata$cells)

# With an explicit family column
cells$lineage <- substr(cells$cluster, 1, 1)
pdata_fam <- as_cluster_data(cells, coord_cols = c("x", "y"),
                             cluster_col = "cluster", family_col = "lineage")
pdata_fam$has_family
```

`cluster_colors` *Get cluster colors as a named vector or a plain data frame*

Description

The most common thing to actually do with a `palettome_session` is hand its colors to something else's plotting call (e.g. `Seurat::DimPlot(cols = cluster_colors(session))`, `ggplot2::scale_color_manual(values = cluster_colors(session))`). This pulls just the `cluster -> color` mapping out of the fuller session object in whichever shape is convenient, without needing to know the session's internal structure.

Usage

```
cluster_colors(session, format = c("vector", "data.frame"))
```

Arguments

`session`	A `palettome_session` (from `generate_palette()`).
`format`	`"vector"` (default) for a named character vector keyed by cluster id, or `"data.frame"` for a `cluster/color/family_id/ family_color` data frame (the family's own representative color merged in alongside its id, so grouping/faceting/legend code doesn't need a separate join against `family_colors()`).

Value

A named character vector, or a data frame, depending on `format`.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
cluster_colors(session)
cluster_colors(session, format = "data.frame")
```

color_distance — *Pairwise perceptual color distance*

Description

Thin wrapper over `farver::compare_colour()` in CIE Lab space, used both by the contrast-mode selector and by the package tests to check that generated colors respect a minimum perceptual separation.

Usage

```
color_distance(hex, method = "CIE2000")
```

Arguments

`hex`	Character vector of hex colors.
`method`	Distance method passed to `farver::compare_colour()` (`"CIE2000"`, `"CIE94"`, `"CIE1976"`/`"euclidean"`, ...).

Value

A symmetric numeric matrix of pairwise distances, named by `hex`.

Examples

```
d <- color_distance(c("#1B9E77", "#D95F02", "#7570B3"))
round(d, 1)
```

compute_centroids — *Compute per-cluster centroids and cell counts*

Description

Precomputes cluster centroids (mean coordinate per cluster), cell counts, and (if present) the family each cluster belongs to. Every downstream interactive operation (recoloring, regrouping) should be able to work off this small summary rather than touching `pdata$cells` again.

Usage

```
compute_centroids(pdata)
```

Arguments

`pdata`	A `palettome_data` object from `as_cluster_data()`.

Value

A data frame with one row per cluster: `cluster`, one column per coordinate named `<coord>_centroid`, `n_cells`, and `family` if `pdata$has_family` is `TRUE`.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
compute_centroids(pdata)
```

---

compute_family_dendrogram

Build a cluster-distance dendrogram for family detection

Description

Computes cluster centroids (or uses a supplied connectivity/distance matrix) and runs hierarchical clustering on them. Kept separate from `detect_families()` so a UI can render a dendrogram preview once and then re-cut it at different k/h cheaply via `cut_families()`, without recomputing distances every time the user drags a cut-height slider.

Usage

```
compute_family_dendrogram(
  pdata,
  dist_method = "euclidean",
  hclust_method = "average",
  connectivity = NULL
)
```

Arguments

`pdata`	A `palettome_data` object from `as_cluster_data()`.
`dist_method`	Distance metric for centroid-based clustering, passed to `stats::dist()`. Ignored if `connectivity` is supplied.
`hclust_method`	Agglomeration method, passed to `stats::hclust()`.
`connectivity`	Optional square cluster-by-cluster connectivity or similarity matrix (e.g. from a shared-nearest-neighbor graph), with row and column names matching cluster ids. When supplied, it is converted to a distance matrix (`1 - normalized similarity`) and used instead of Euclidean centroid distance.

Value

A list with `hclust` (an stats::hclust object), `centroids` (from `compute_centroids()`), and `clusters` (the cluster id order used).

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
dendro <- compute_family_dendrogram(pdata)
dendro$hclust
```

`cut_families` *Cut a precomputed dendrogram into families*

Description

Cut a precomputed dendrogram into families

Usage

```
cut_families(dendro, k = NULL, h = NULL)
```

Arguments

`dendro`	A list as returned by `compute_family_dendrogram()`.
`k`	Desired number of families (mutually exclusive with `h`).
`h`	Height at which to cut the dendrogram (mutually exclusive with `k`). If both are `NULL`, `k` defaults to the square root of the number of clusters, rounded, as a reasonable starting point.

Value

A data frame with columns `cluster` and `family_id` (character, `"F1"`, `"F2"`, ...).

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
dendro <- compute_family_dendrogram(pdata)
cut_families(dendro, k = 3)
```

detect_families — *Detect (or pass through) cluster families*

Description

If pdata carries an explicit family column (i.e. pdata$has_family is TRUE), that mapping is used as-is and no clustering is performed. Otherwise, families are auto-detected by hierarchical clustering of cluster centroids (or a supplied connectivity graph). Either way, the result is meant to be a *starting point*: it is a plain cluster -> family_id data frame that a UI can freely rewrite by hand (drag-to-regroup) afterward.

Usage

```
detect_families(
  pdata,
  k = NULL,
  h = NULL,
  dist_method = "euclidean",
  hclust_method = "average",
  connectivity = NULL
)
```

Arguments

pdata	A palettome_data object from as_cluster_data().
k, h	See cut_families(). Ignored when families are explicit.
dist_method	Distance metric for centroid-based clustering, passed to stats::dist(). Ignored if connectivity is supplied.
hclust_method	Agglomeration method, passed to stats::hclust().
connectivity	Optional square cluster-by-cluster connectivity or similarity matrix (e.g. from a shared-nearest-neighbor graph), with row and column names matching cluster ids. When supplied, it is converted to a distance matrix (1 - normalized similarity) and used instead of Euclidean centroid distance.

Value

A palettome_families object: a list with

assignment data frame with columns cluster, family_id.

method "explicit", "hclust", or "graph".

dendro The compute_family_dendrogram() result, or NULL if method == "explicit"; kept so a UI can preview/re-cut the dendrogram without recomputation.

neighbors Neighbor information ready to pass to generate_palette() – the supplied connectivity matrix if any, otherwise the cluster centroids – so neighbor-aware coloring works without recomputing anything.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
fam <- detect_families(pdata, k = 3)
fam$method
fam$assignment
```

---

downsample_stratified *Stratified per-cluster subsample for interactive preview*

Description

Precomputing a small, cluster-stratified subsample is the core trick that keeps hierarchy editing and recoloring responsive on multi-million-cell datasets: interactive redraws use this subsample, and only a final export needs to touch every cell. See `inst/benchmark/` for a benchmark justifying the approach and default threshold.

Usage

```
downsample_stratified(pdata, max_n = 50000, min_per_cluster = 20, seed = 1)
```

Arguments

`pdata`	A `palettome_data` object.
`max_n`	Target total number of points after subsampling. If `nrow(pdata$cells) <= max_n`, `pdata` is returned unchanged.
`min_per_cluster`	Minimum points kept per cluster even if that over-represents small clusters relative to a uniform per-cluster share, so rare clusters stay visible.
`seed`	Random seed for reproducible subsampling. The caller's random number state is restored on exit.

Value

A `palettome_data` object with a (possibly) reduced `cells` table.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
```

```
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
small <- downsample_stratified(pdata, max_n = 60, min_per_cluster = 5)
table(small$cells$cluster)
```

`export_palette_csv` *Export cluster and/or family color tables as CSV*

Description

Export cluster and/or family color tables as CSV

Usage

```
export_palette_csv(session, clusters_file, families_file = NULL)
```

Arguments

`session`	A `palettome_session`.
`clusters_file`	Path to write the cluster -> color table to (columns `cluster`, `family_id`, `color`, `manual_color`, `n_cells`), or `NULL` to skip.
`families_file`	Optional path to also write the family -> color table to (columns `family_id`, `color`, `manual_color`).

Value

Invisibly, `TRUE`.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
clusters_file <- file.path(tempdir(), "clusters.csv")
families_file <- file.path(tempdir(), "families.csv")
export_palette_csv(session, clusters_file, families_file)
read.csv(clusters_file)
unlink(c(clusters_file, families_file))
```

export_palette_json *Export a palette session to JSON*

Description

Writes the stable, language-agnostic JSON contract described in the package README (`schema_version`, `clusters`, `families`, `params`) so a future non-R (e.g. Python/Scanpy) frontend could read/write the same file.

Usage

```
export_palette_json(session, file)
```

Arguments

`session`	A `palettome_session`.
`file`	Path to write to.

Value

Invisibly, the JSON string that was written.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
path <- file.path(tempdir(), "palette.json")
export_palette_json(session, path)
unlink(path)
```

family_colors *Get family colors as a named vector or a plain data frame*

Description

Same as `cluster_colors()`, but for each family's representative/accent color (the harmonious-mode base hue, or the contrast-mode secondary channel) rather than per-cluster fill colors.

Usage

```
family_colors(session, format = c("vector", "data.frame"))
```

Arguments

session	A palettome_session (from generate_palette()).
format	"vector" (default) for a named character vector keyed by cluster id, or "data.frame" for a cluster/color/family_id/ family_color data frame (the family's own representative color merged in alongside its id, so grouping/faceting/legend code doesn't need a separate join against family_colors()).

Value

A named character vector, or a data frame, depending on format.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
family_colors(session)
family_colors(session, format = "data.frame")
```

---

generate_palette *Generate a cluster/family color palette*

Description

The core color-generation entry point. Given a cluster -> family assignment, it places one base hue per family and then colors each cluster, in one of two modes:

Usage

```
generate_palette(
  assignment,
  session = NULL,
  mode = c("harmonious", "contrast"),
  harmonious_style = c("sweep", "per_family"),
  sweep_anchors = NULL,
  neighbor_hues = NULL,
  neighbors = NULL,
  seed = 1,
  lightness_range = "auto",
  chroma_range = "auto",
  hue_range = c(0, 360),
  cvd = c("none", "deutan", "protan", "tritan"),
  n_cells = NULL
)
```

Arguments

- `assignment` — A data frame with columns `cluster`, `family_id`.
- `session` — An existing `palettome_session` to inherit manual overrides from, or `NULL`.
- `mode` — `"harmonious"` or `"contrast"`.
- `harmonious_style` — `"sweep"` or `"per_family"` (only used when `mode = "harmonious"`).
- `sweep_anchors` — Optional character vector of 2+ hex colors (only used when `mode = "harmonious"` and `harmonious_style = "sweep"`): a gradient to sweep the palette along, interpolated through in HCL space in the order given, instead of the automatically-placed analogous arc. Handy when the auto arc's seed-to-seed variety (see generate_palette examples) still isn't the specific look you want – e.g. brand colors, or literally the colors from a reference gradient.
- `neighbor_hues` — `"contrast"`, `"coherent"`, or `NULL` to follow the mode (contrast -> `"contrast"`, harmonious -> `"coherent"`).
- `neighbors` — Optional neighbor information used to order families and clusters in space: a data frame of centroids (`cluster` + `*_centroid` or coordinate columns, e.g. `compute_centroids()`) or a square cluster-by-cluster connectivity/distance matrix. `NULL` disables neighbor-aware placement (falls back to even hue spacing).
- `seed` — Random seed, so the same inputs always give the same colors. The caller's random number state is restored on exit.
- `lightness_range`, `chroma_range` — Length-2 numeric ranges in HCL space, or `"auto"` (default) to derive them from manual overrides when present and from mode/style presets otherwise.
- `hue_range` — Length-2 numeric range in degrees to draw hues from.
- `cvd` — `"none"`, `"deutan"`, `"protan"`, or `"tritan"`.
- `n_cells` — Optional named integer vector or `cluster`/`n_cells` data frame, stored on the session for convenience.

Details

- `mode = "harmonious"` – a coordinated palette in HCL space. `harmonious_style = "sweep"` (default) lays families along a single analogous hue arc with a shared monotone lightness ramp and muted chroma (a smooth blue -> purple -> rose -> tan style sweep); `harmonious_style = "per_family"` gives each family its own hue and colors its members as graded shades of it.
- `mode = "contrast"` – every cluster gets a maximally-distinguishable color via greedy max-min selection in Lab space; family membership is carried by `$families$color` as a secondary channel.

Neighbor-awareness: when `neighbors` is supplied, families that are adjacent in the embedding/graph get **contrasting** hues by default (`neighbor_hues = "contrast"`, the default in contrast mode) or **analogous** hues (`neighbor_hues = "coherent"`, the default in harmonious mode); and within a family, spatially-adjacent clusters are given far-apart lightness for local contrast.

Passing a previous `session` makes this incremental: any cluster or family with `manual_color == TRUE` keeps its color, the generator treats those as fixed and avoids clashing with them, and with `lightness_range`/`chroma_range` left at `"auto"` the non-manual colors are regenerated inside the L/C envelope implied by the manual picks.

Value

A `palettome_session` object (see the package README for the JSON schema).

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")

fam <- detect_families(pdata, k = 3)

# Harmonious: one coordinated hue sweep, neighbor-aware
session <- generate_palette(fam$assignment, mode = "harmonious",
                            neighbors = fam$neighbors, seed = 1)
cluster_colors(session)

# Different seeds give different sweeps
cluster_colors(generate_palette(fam$assignment, seed = 7))

# Contrast: every cluster maximally distinguishable
session_c <- generate_palette(fam$assignment, mode = "contrast", seed = 1)
cluster_colors(session_c)

# Sweep along a gradient you choose
cluster_colors(generate_palette(fam$assignment,
                                sweep_anchors = c("#0B3D91", "#FF5733")))
```

`import_palette_json` *Import a palette session from JSON*

Description

Import a palette session from JSON

Usage

```
import_palette_json(file)
```

Arguments

`file`	Path to a JSON file written by `export_palette_json()`.

Value

A `palettome_session`.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
path <- file.path(tempdir(), "palette.json")
export_palette_json(session, path)
session_back <- import_palette_json(path)
identical(cluster_colors(session_back), cluster_colors(session))
unlink(path)
```

---

launch_palettome_ui *Launch the interactive Palettome Shiny UI*

Description

This is the *only* function in the package that requires UI packages (shiny, plotly, colourpicker, shinyjs). All of them are Suggests-only, so installing/loading the palettome package itself never requires them, and every other function (`detect_families()`, `generate_palette()`, `plot_palette_static()`, `export_palette_json()`, ...) works without them. If any are missing, this fails with a friendly message rather than an obscure error, and points out that the rest of the package still works.

Usage

```
launch_palettome_ui(
  pdata,
  families = NULL,
  session = NULL,
  k = NULL,
  h = NULL,
  connectivity = NULL,
  preview_max_n = 50000,
  ...
)
```

Arguments

`pdata`	A `palettome_data` object from `as_cluster_data()`.
`families`	Optional: a `palettome_families` object (from `detect_families()`) or a plain `cluster`/`family_id` data frame to seed the initial grouping. If `NULL`, `detect_families()` is run with k/h.
`session`	Optional: a `palettome_session` (e.g. loaded via `import_palette_json()`) to resume editing, preserving prior manual overrides. If `NULL`, a fresh harmonious palette is generated.
`k, h`	Passed to `detect_families()` when `families` is `NULL`.
`connectivity`	Optional cluster-by-cluster connectivity/distance matrix passed to `detect_families()` and used for neighbor-aware coloring; when `NULL`, cluster centroids are used instead.

preview_max_n	Passed to downsample_stratified() for the live scatter plot; the full dataset is only used for the final PNG export.
...	Passed on to shiny::runApp() (e.g. launch.browser, port).

Value

Invisibly, the edited palettome_session once the app closes (via the "Done" button, or by interrupting R with Esc / Ctrl+C), so session <- launch_palettome_ui(...) keeps every change made in the UI. The app also shows the current colors as pasteable named-vector R code.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
if (interactive()) {
  session <- launch_palettome_ui(pdata, k = 3)
  cluster_colors(session)
}
```

---

optimize_color *Snap a hand-picked color into the palette's aesthetic envelope*

Description

Feature companion to manual recoloring: keeps the hue you chose but pulls its lightness and chroma to something in-gamut and coherent – either matched to the lightness/chroma distribution of an existing session (so a hand-picked color sits with the rest), or, with no session, toward the natural lightness for that hue and a pleasing (non-neon) chroma.

Usage

```
optimize_color(color, session = NULL, target = c("auto", "muted", "vivid"))
```

Arguments

color	A hex color string.
session	Optional palettome_session whose current colors define the target L/C register.
target	"auto" (keep near the input chroma), "muted", or "vivid".

Value

A hex color string with the same hue, adjusted L/C.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
optimize_color("#7B00FF")
optimize_color("#7B00FF", session = session, target = "muted")
```

---

plot_dendrogram *Plot a dendrogram preview of detected families*

Description

A dependency-light preview: uses dendextend to color branches by family when it is installed, and falls back to base `stats::plot.hclust()` with `stats::rect.hclust()` otherwise. Either way this only needs base R + stats to run.

Usage

```
plot_dendrogram(dendro, k = NULL, h = NULL)
```

Arguments

dendro	A list from `compute_family_dendrogram()` (or the $dendro element of a `detect_families()` result).
k, h	Passed to `cut_families()` / `stats::rect.hclust()` to draw the family cut.

Value

Invisibly, the (possibly colored) dendrogram object that was plotted.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
fam <- detect_families(pdata, k = 3)
plot_dendrogram(fam$dendro, k = 3)
```

plot_palette_static *Static plot of an embedding colored by the current palette*

Description

Works with zero UI packages installed: uses ggplot2 for a nicer plot if it happens to be available (it is Suggests-only, never required), and otherwise falls back to base `graphics::plot()`. Either way, no Shiny/UI dependency is ever touched.

Usage

```
plot_palette_static(
  pdata,
  session,
  cvd = "none",
  point_size = 0.6,
  alpha = 0.7
)
```

Arguments

`pdata`	A `palettome_data` object from `as_cluster_data()`.
`session`	A `palettome_session` (from `generate_palette()`) supplying the cluster -> color mapping.
`cvd`	`"none"` or a CVD type; when set, points are drawn with `simulate_cvd()`-adjusted colors as a colorblind-safe preview.
`point_size, alpha`	Passed through to the plotting backend.

Details

For large datasets, consider subsampling `pdata$cells` (e.g. stratified per cluster) before calling this for interactive-style previews, and only plotting the full data for a final export; see `inst/benchmark/` for the rendering-strategy discussion behind this recommendation.

Value

A `ggplot` object if ggplot2 is installed, otherwise `NULL` (invisibly) after drawing directly to the current base graphics device.

Examples

```
cells <- data.frame(
  x = rnorm(120, rep(c(0, 1, 6, 7, 3, 4), each = 20)),
  y = rnorm(120, rep(c(0, 1, 0, 1, 6, 7), each = 20)),
  cluster = rep(c("T1", "T2", "B1", "B2", "M1", "M2"), each = 20)
)
```

```
pdata <- as_cluster_data(cells, coord_cols = c("x", "y"), cluster_col = "cluster")
fam <- detect_families(pdata, k = 3)
session <- generate_palette(fam$assignment, neighbors = fam$neighbors, seed = 1)
plot_palette_static(pdata, session)
```

`plot_swatches` *Standalone swatch/legend figure, grouped by family*

Description

A base-graphics-only (no ggplot2/Shiny needed) legend figure: one row per cluster, colored swatch + label, visually grouped and separated by family, with the family's own color shown as a header bar. Suitable for exporting as PNG/SVG alongside the main plot.

Usage

```
plot_swatches(session, cvd = "none", order_by = c("lightness", "id"))
```

Arguments

`session`	A `palettome_session`.
`cvd`	`"none"` or a CVD type to preview the swatches under.
`order_by`	`"lightness"` (default) sorts families, and clusters within each family, by lightness so a harmonious sweep reads as a continuous ramp; `"id"` keeps `family_id` / `cluster` order.

Value

Invisibly, `NULL`; draws to the current graphics device.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
plot_swatches(session)
plot_swatches(session, cvd = "deutan")
```

reset_overrides *Reset manual color overrides*

Description

Reset manual color overrides

Usage

```
reset_overrides(session, cluster = TRUE, family = TRUE)
```

Arguments

session	A palettome_session.
cluster, family	Character vector of ids to reset, or TRUE to reset all clusters/families respectively. Defaults to resetting everything.

Value

The updated palettome_session (colors are left as-is; only the manual_color flags are cleared – call generate_palette() again to recompute the now-unlocked colors).

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
session <- set_manual_color(session, cluster = "c1", color = "#2E5A87")
session <- reset_overrides(session)
any(session$clusters$manual_color)
```

set_manual_color *Manually set a cluster's or family's color*

Description

Sets color and flags manual_color = TRUE for the given cluster and/or family in a session. Manually-colored entries are left untouched by later generate_palette() calls (until reset with reset_overrides()).

Usage

```
set_manual_color(session, cluster = NULL, family = NULL, color)
```

Arguments

session	A palettome_session.
cluster, family	Cluster ids / family ids to recolor (either or both; at least one required). May be vectors to recolor several at once.
color	Hex color string(s): either a single color applied to every target, or one color per id (recycled separately for cluster and family). Ids not present in the session are dropped, together with their paired colors, with a warning.

Value

The updated palettome_session.

Examples

```
assignment <- data.frame(cluster = paste0("c", 1:4), family_id = rep(c("F1", "F2"), each = 2))
session <- generate_palette(assignment, mode = "harmonious", seed = 1)
session <- set_manual_color(session, cluster = "c1", color = "#2E5A87")
session$clusters[, c("cluster", "color", "manual_color")]

# Several clusters at once, one color each
session <- set_manual_color(session, cluster = c("c3", "c4"),
                            color = c("#2E5A87", "#6E6E6E"))

# Manual picks survive regeneration
session <- generate_palette(assignment, session = session, mode = "contrast")
cluster_colors(session)[["c1"]]
```

---

simulate_cvd *Simulate color-vision deficiency on a palette*

Description

Preview-only helper: does not mutate a session, just maps colors through colorspace::deutan()/colorspace::protan()/c
so a UI can toggle a colorblind-safe preview of an already-generated palette.

Usage

```
simulate_cvd(hex, type = c("deutan", "protan", "tritan"), severity = 1)
```

Arguments

hex	Character vector of hex colors.
type	One of "deutan", "protan", "tritan".
severity	Severity in [0, 1], passed through to colorspace.

Value

Character vector of simulated hex colors, same length as hex.

Examples

```
pal <- c("#1B9E77", "#D95F02", "#7570B3", "#E7298A")
simulate_cvd(pal, type = "deutan")
simulate_cvd(pal, type = "tritan", severity = 0.5)
```

Index

as_cluster_data, 2
as_cluster_data(), *5, 6*, *8*, *15*, *18*

cluster_colors, 4
cluster_colors(), *11*
color_distance, 5
colorspace::deutan(), *21*
colorspace::protan(), *21*
colorspace::tritan(), *21*
compute_centroids, 5
compute_centroids(), *7*, *13*
compute_family_dendrogram, 6
compute_family_dendrogram(), *7*, *17*
cut_families, 7
cut_families(), *6*, *8*, *17*

detect_families, 8
detect_families(), *2, 3*, *6*, *15*, *17*
downsample_stratified, 9
downsample_stratified(), *16*

export_palette_csv, 10
export_palette_json, 11
export_palette_json(), *14, 15*

family_colors, 11
family_colors(), *4*, *12*
farver::compare_colour(), *5*

generate_palette, 12, *13*
generate_palette(), *2*, *4*, *8*, *12*, *15*, *18*, *20*
graphics::plot(), *18*

import_palette_json, 14
import_palette_json(), *15*

launch_palettome_ui, 15

optimize_color, 16

plot_dendrogram, 17
plot_palette_static, 18
plot_palette_static(), *2*, *15*
plot_swatches, 19

reset_overrides, 20
reset_overrides(), *20*

set_manual_color, 20
simulate_cvd, 21
simulate_cvd(), *18*
stats::dist(), *6*, *8*
stats::hclust, *7*
stats::hclust(), *6*, *8*
stats::plot.hclust(), *17*
stats::rect.hclust(), *17*