



1 · SETUP

pak: `pak ("PrigasG/dragmapr")`
Install from GitHub (CRAN release coming).

library (`dragmapr`)

prepare_dragmapr_sf (`x`)
Converts lon/lat data to a projected CRS (EPSG:3857 by default), repairs invalid geometry, and keeps only polygon features. **Run this first** if your data uses degrees.

```
prepare_dragmapr_sf(my_sf)
prepare_dragmapr_sf(my_sf, target_crs = 32632)
```

CORE WORKFLOW



🔗 Using Spatial Studio? Skip ③–④ and call `render_dragmapr_project("project.zip")` instead.

READ & APPLY OFFSETS

read_offsets (`file`)
Load a region offset CSV downloaded from the browser.

apply_offsets (`x`, `offsets`)
Shift an `sf` object's geometry by the saved region offsets.

read_label_state (`file`)
Load a label offset CSV.

apply_label_state (`labels`, `file`)
Apply saved label positions to a label table.

```
# Typical round-trip
reg <- read_offsets("drag_region_offsets.csv")
labs <- read_label_state("drag_label_offsets.csv")
render_dragged_map(my_sf, region_col = "region",
  region_offsets = reg, label_offsets = labs)
```

2 · OPEN THE INTERACTIVE MAP

drag_map_prototype (`x`, `region_col`, ...)
Writes a self-contained HTML file and (optionally) opens it in your browser. Drag regions and labels, then download the two offset CSVs from the side panel.

Argument	What it does
<code>x</code>	Your projected <code>sf</code> object
<code>region_col</code>	Column that groups regions (required)
<code>label_col</code>	Column to use as label text (default = <code>region_col</code>)
<code>labels</code>	TRUE auto-labels · FALSE no labels · data frame for custom
<code>open</code>	TRUE opens browser immediately
<code>show_legend</code>	TRUE adds a region colour legend
<code>map_background</code>	"white" · "light_grid" · "dark" · "transparent"
<code>region_palette</code>	Named vector of hex colours per region
<code>file</code>	Where to save the HTML (temp file if NULL)
<code>side_panel</code>	FALSE hides panel (use in Shiny apps)
<code>transition</code>	Bloom config for parent/child groupings adv

```
# Minimal — opens browser
drag_map_prototype(my_sf, region_col = "region", open = TRUE)

# With legend and dark background
drag_map_prototype(my_sf, "region",
  show_legend = TRUE, map_background = "dark", open = TRUE)
```

RSTUDIO ADDIN

dragmapr_addin ()
Or use **Addins** › **Launch dragmapr** in the RStudio toolbar. Point-and-click column selection, drag the map, click *Done* — saves `region_offsets` and `label_offsets` to `.GlobalEnv`.

```
# Load your sf first, then launch
regions <- prepare_dragmapr_sf(sf::st_read("map.shp"))
dragmapr_addin()

# After clicking Done:
render_dragged_map(regions, region_col = "name",
  region_offsets = region_offsets,
  label_offsets = label_offsets,
  file = "map.png")
```

3 · LABELS

make_region_labels (`x`, `region_col`, `label_col`)
Derive one label per region from your data — the easiest starting point.

as_drag_labels (`df`)
Validate and wrap a custom label data frame. Needs columns: `label_id`, `region`, `label`, `x`, `y`.

as_drag_annotations (`df`, `connector`, `connector_type`)
Turns label rows into draggable callout boxes with optional connector lines. Types: "straight" · "elbow" · "curve" · "squiggle".

select_label_ids (`labels`, `ids`)
Show only the listed label IDs (e.g. for Shiny filter controls).

```
# Auto labels
lbs <- make_region_labels(my_sf, region_col = "region")

# Custom callout box with a curved connector
note <- as_drag_annotations(
  data.frame(label_id="n1", region="North",
    label="Key area", x=5e4, y=1.5e5),
  connector=TRUE, connector_type="curve")
```

☑ Save and restore label positions across sessions with `read_label_state()` / `apply_label_state()`.

BUILT-IN EXAMPLES

`basic_draggable_map.R` — four regions, zero setup
`shiny_draggable_export.R` — live preview + PNG export
`shiny_spatial_studio.R` — full Spatial Studio app
`branch-bloom-tester.R` — parent/child bloom demo
`explodemap_hhs_labels.R` — HHS-style exploded layout

```
source(system.file(
  "examples", "basic_draggable_map.R",
  package = "dragmapr"))
```



4 · STATIC EXPORT

render_dragged_map(x, region_col, ...)
Applies saved offsets and renders a ggplot2 image. Returns the ggplot object; saves to disk when file is set.

Argument	What it does
region_offsets	Data frame or CSV path from browser download
label_offsets	Label CSV or data frame
labels	Label table · FALSE to skip labels
show_legend	Show region fill legend
legend_title	Legend heading text
map_background	"white" · "light_grid" · "dark"
connector_endpoint	"none" or "arrow" on label connectors
show_origin_outlines	Ghost outlines of original positions
show_movement_connectors	Lines from original → dragged position
file	Output path (.png / .pdf / .svg)
width, height, dpi	Output dimensions (inches & resolution)

```
render_dragged_map(my_sf,
  region_col    = "region",
  region_offsets = "drag_region_offsets.csv",
  label_offsets = "drag_label_offsets.csv",
  show_legend   = TRUE,
  file          = "map.png",
  width=10, height=8, dpi=300)
```

render_dragmapr_project(zip, ...)
Reads a Spatial Studio project ZIP and renders the final image in one call. No separate CSVs needed.

```
render_dragmapr_project("project.zip",
  file="map.png", width=10, height=8)
```

read_dragmapr_project(zip)
Inspect a saved project bundle: source geometry, offsets, labels, palette, and metadata.

write_dragmapr_project(x, ..., file)
Bundle source geometry + offsets + metadata into a ZIP for sharing.

SPATIAL STUDIO & SHINY

shiny_spatial_studio.R example
Full-featured Shiny app: upload shapefiles/GeoJSON/GPKG, pick grouping and label columns, drag the map, switch columns while keeping layout, export PNG/PDF/ZIP.

```
source(system.file("examples",
  "shiny_spatial_studio.R",
  package="dragmapr"))
```

read_dragmapr_sf_upload(input\$file)
Read a user-uploaded spatial file (ZIP shapefile, GeoJSON, GPKG) inside a Shiny server function.

read_dragmapr_sf_url(url)
Download and read a spatial file from a URL.

dragmapr_iframe_bridge()
JavaScript bridge that relays drag state from the helper <iframe> to Shiny input values. Add to your UI once.

```
# Minimal Shiny integration
ui <- fluidPage(
  dragmapr_iframe_bridge(),
  uiOutput("map_iframe")

server <- function(input, output, session) {
  sf_data <- reactive({
    read_dragmapr_sf_upload(input$file)
  })
}
```

suggest_offsets(x, region_col, method)
Generate automatic starting offsets — useful as a Spatial Studio default before any dragging. Methods: "radial" · "grid" · "directional".

HIERARCHY & BLOOM **ADVANCED**

Parent/child bloom lets you drill into sub-regions with a leaf-flip animation while keeping the rest of the map at parent level.

detect_hierarchy_columns(x)
Scan all columns and return candidate parent/child pairs based on nesting structure.

recommend_dragmapr_hierarchy(x)
Recommend the best grouping column and its child column, with a data-driven reason.

validate_bloom_hierarchy(x, parent_col, child_col)
Check that the child column genuinely nests inside the parent — required before arming a bloom transition.

make_hierarchy_key(x, cols)
Build stable composite keys when child names repeat across parents.

build_branch_transition_data(x, parent_col, child_col)
Build the transition list to pass to drag_map_prototype(), including dissolved parent shells and bloom group mappings.

make_branch_bloom_labels(x, ...)
Build a combined label table covering both parent and child levels for a bloom helper.

inherit_layout(offsets, x, from_col, to_col)
Carry drag offsets from one grouping column to another (e.g. parent → child positions on first visit).

transition_options() / build_elastic_transition()
Create static transition frames and dotted group boundaries for package-level transition testing.

make_group_boundaries(x, group_col)
Return the padded dotted boundary geometry used as a group-drag frame.

```
# Typical bloom setup
td <- build_branch_transition_data(
  my_sf, parent_col="region", child_col="state")
drag_map_prototype(td$data, "region",
  transition=td$transition, open=TRUE)
```

UTILITIES & DIAGNOSTICS

dragmapr_diagnostics(x)
Report CRS, geometry type, column types, and potential issues before you start dragging.

profile_spatial_upload(x)
Summarise an uploaded file: row count, CRS, hierarchy candidates, recommended grouping.

summarise_spatial_crs(x)
Human-readable description of the CRS and whether reprojection is needed.

create_layout_snapshot() / restore_layout_snapshot()
Capture and restore offsets for a grouping while switching columns or sessions.

layout_metrics(before, after, id_col)
Measure drift and stability between two map layouts.

example_hhs_layout() / example_panel_layout()
Return small built-in layouts for testing and examples — no external data needed.

🔗 Aliases `make_labels()`, `read_label_offsets()`, and `apply_label_offsets()` are kept for older scripts but emit deprecation warnings.