

# Package: pkmapr (via r-universe)

May 17, 2026

**Title** Pakistan Spatial Data Toolkit

**Version** 1.4.0

**Depends** R (>= 4.1.0)

**Description** Provides a tidy interface to Pakistan's official administrative boundary data from the United Nations Office for the Coordination of Humanitarian Affairs (OCHA). Spatial data at country, province, district, and tehsil levels are embedded in the package as 'sf' objects compatible with the 'tidyverse' and geospatial ecosystem. Includes utilities for geographic dictionary lookup, coordinate reference system selection, spatial measurement, and neighbour structure construction for use with 'spdep', 'ggplot2', 'leaflet', and related packages.

**License** GPL-3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Imports** sf, dplyr, rlang, cli, spdep, jsonlite

**Suggests** ggplot2, leaflet, rmapshaper, httptest2, testthat (>= 3.0.0), tmap, knitr, readr, rmarkdown, spelling, PakPC2023

**VignetteBuilder** knitr

**URL** <https://cran.r-project.org/package=pkmapr>,  
<https://abdullahumer1101.github.io/pkmapr/>,  
<https://github.com/abdullahumer1101/pkmapr/>

**BugReports** <https://github.com/abdullahumer1101/pkmapr/issues>

**Config/pak/sysreqs** libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

**Repository** <https://abdullahumer1101.r-universe.dev>

**Date/Publication** 2026-05-17 22:15:26 UTC

**RemoteUrl** <https://github.com/abdullahumer1101/pkmapr>

**RemoteRef** HEAD

**RemoteSha** bf8ca2cae2c860ef22f0a492da6756f0e0c94c3c

## Contents

get_country . . . . .	2
get_districts . . . . .	3
get_provinces . . . . .	4
get_tehsils . . . . .	5
pk_area . . . . .	6
pk_basemap . . . . .	6
pk_bbox . . . . .	7
pk_buffer . . . . .	8
pk_centroid . . . . .	9
pk_crs_suggest . . . . .	9
pk_dictionary . . . . .	10
pk_distance . . . . .	11
pk_intersect . . . . .	12
pk_join . . . . .	13
pk_map . . . . .	14
pk_map_interactive . . . . .	15
pk_neighbors . . . . .	16
pk_points_in . . . . .	17
pk_project . . . . .	18
pk_search . . . . .	19
pk_union . . . . .	20
pk_version . . . . .	21
<b>Index</b>	<b>22</b>

---

get_country	<i>Get national boundary for Pakistan.</i>
-------------	--

---

### Description

Get national boundary for Pakistan.

### Usage

```
get_country(crs = 4326)
```

### Arguments

crs	Integer EPSG code. Default 4326 (WGS84). Use 32642 for distance and area calculations. See <a href="#">pk_crs_suggest()</a> for guidance.
-----	---

**Value**

Returns an sf object (class "sf" and "data.frame") with:

country_name	Character. Name of the country ("Pakistan")
country_code	Character. ISO country code
area_km2	Numeric. Area in square kilometres
geometry	MULTIPOLYGON. The national boundary geometry

The output represents the complete national boundary of Pakistan.

**Examples**

```
pakistan <- get_country()
plot(sf::st_geometry(pakistan))
print(pakistan$area_km2)
```

---

get_districts	<i>Get Pakistan district boundaries</i>
---------------	---

---

**Description**

Get Pakistan district boundaries

**Usage**

```
get_districts(province = NULL, crs = 4326)
```

**Arguments**

province	Character. Filter to one province by exact name. Matching is case-insensitive. Run <code>pk_dictionary("provinces")</code> to see valid names. NULL (default) returns all districts.
crs	Integer EPSG code. Default 4326 (WGS84). Use 32642 for distance and area calculations. See <a href="#">pk_crs_suggest()</a> for guidance.

**Value**

Returns an sf object (class "sf" and "data.frame") with:

province_name	Character. Parent province name
district_name	Character. District name
district_code	Character. Unique district identifier code (e.g., "PK603")
area_km2	Numeric. Area in square kilometres for each district
geometry	MULTIPOLYGON. District boundary geometries

When province is specified, the output contains only districts within that province. The output represents administrative boundaries at the district level.

**Examples**

```
# All districts
all_districts <- get_districts()

# Filter to Punjab province (case-insensitive)
punjab_districts <- get_districts(province = "Punjab")
punjab_districts <- get_districts(province = "punjab") # Same result

plot(sf::st_geometry(punjab_districts))
```

---

get_provinces	<i>Get Pakistan province boundaries</i>
---------------	---

---

**Description**

Get Pakistan province boundaries

**Usage**

```
get_provinces(crs = 4326)
```

**Arguments**

crs	Integer EPSG code. Default 4326 (WGS84). Use 32642 for distance and area calculations. See <a href="#">pk_crs_suggest()</a> for guidance.
-----	---

**Value**

Returns an sf object (class "sf" and "data.frame") with:

province_name	Character. Name of the province (e.g., "Punjab", "Sindh")
province_code	Character. Unique province identifier code
area_km2	Numeric. Area in square kilometres for each province
geometry	MULTIPOLYGON. Province boundary geometries

The output represents the administrative boundaries of Pakistan's provinces and territories.

**Examples**

```
provinces <- get_provinces()
plot(sf::st_geometry(provinces))
head(provinces)
```

---

get_tehsils	<i>Get Pakistan tehsil boundaries</i>
-------------	---------------------------------------

---

**Description**

Get Pakistan tehsil boundaries

**Usage**

```
get_tehsils(district = NULL, province = NULL, crs = 4326)
```

**Arguments**

district	Character. Filter to one district by exact name. Matching is case-insensitive. NULL returns all.
province	Character. Filter to one province by exact name. Matching is case-insensitive. NULL returns all. If both district and province are supplied, district takes precedence.
crs	Integer EPSG code. Default 4326 (WGS84). Use 32642 for distance and area calculations. See <a href="#">pk_crs_suggest()</a> for guidance.

**Value**

Returns an sf object (class "sf" and "data.frame") with:

province_name	Character. Parent province name
district_name	Character. Parent district name
tehsil_name	Character. Tehsil name
tehsil_code	Character. Unique tehsil identifier code
area_km2	Numeric. Area in square kilometres for each tehsil
geometry	MULTIPOLYGON. Tehsil boundary geometries

The output represents the finest available administrative boundaries in pkmpr, suitable for high-resolution spatial analysis, local-level mapping, and joining with tehsil-level census or survey data.

**Examples**

```
# All tehsils
all_tehsils <- get_tehsils()

# Filter to Sindh province
sindh_tehsils <- get_tehsils(province = "Sindh")
sindh_tehsils <- get_tehsils(province = "sindh") # Case-insensitive

# Filter to Lahore district
lahore_tehsils <- get_tehsils(district = "Lahore")
lahore_tehsils <- get_tehsils(district = "lahore") # Case-insensitive
```

```
plot(sf::st_geometry(lahore_tehsils))
```

---

pk\_area                      *Recalculate area in km<sup>2</sup> for an sf object*

---

### Description

Computes accurate areas by reprojecting to UTM Zone 42N before measurement. Returns the input object with area\_km2 added or updated.

### Usage

```
pk_area(x)
```

### Arguments

x                              An sf object with polygon geometries.

### Value

Returns the input sf object (class "sf" and "data.frame") with the

area\_km2                      Numeric column added or updated, representing the area of each polygon in square kilometres.

### Examples

```
districts <- get_districts()
districts <- pk_area(districts)
head(districts$area_km2)
```

---

pk\_basemap                      *Leaflet basemap centred on Pakistan*

---

### Description

Returns a leaflet map pre-zoomed to Pakistan's extent as a starting point for building custom interactive maps.

### Usage

```
pk_basemap(provider = "CartoDB.Positron")
```

### Arguments

provider                      Character. Tile provider. Default "CartoDB.Positron".

**Value**

Returns a leaflet object (class "leaflet" and "htmlwidget") with the following configuration:

Provider tiles	Specified tile provider (default: CartoDB.Positron)
Bounds	Pre-zoomed to Pakistan's extent: longitude 60.9°E to 77.8°E, latitude 23.5°N to 37.1°N

The output is a fully customizable leaflet basemap ready for adding layers (e.g., administrative boundaries, point data, markers). Use this as a starting point for building custom interactive maps.

**Examples**

```
# Default basemap with CartoDB.Positron tiles
pk_basemap()

# Alternative tile provider
pk_basemap(provider = "OpenStreetMap")
```

---

pk_bbox	<i>Get a bounding box for a named administrative unit</i>
---------	---

---

**Description**

Retrieves the spatial extent (bounding box) of a specific administrative unit by name and level. Useful for setting map views or cropping other spatial data.

**Usage**

```
pk_bbox(name, level = c("province", "district", "tehsil"))
```

**Arguments**

name	Character. Name of the administrative unit (e.g., "Punjab", "Lahore").
level	Character. One of "province", "district", or "tehsil".

**Value**

Returns a bbox object (class "bbox") with named elements:

xmin	Minimum x-coordinate (longitude or easting)
ymin	Minimum y-coordinate (latitude or northing)
xmax	Maximum x-coordinate
ymax	Maximum y-coordinate

The output is suitable for use with `ggplot2::coord_sf(xlim = c(xmin, xmax), ylim = c(ymin, ymax))` or `leaflet::fitBounds(lng1 = xmin, lat1 = ymin, lng2 = xmax, lat2 = ymax)`. It represents the rectangular extent that exactly contains the requested administrative unit.

**Note**

If you see an error like object 'xxx' not found when using this function, the issue is likely in your data preparation, not `pk_bbox()`. Test the function directly: `pk_bbox("Punjab", level = "province")`. If that works, check that your data has the expected column names.

**Examples**

```
# Get bounding box for Lahore district
bb <- pk_bbox("Lahore", level = "district")
print(bb)

# Use with ggplot2

library(ggplot2)
districts <- get_districts()
bb_punjab <- pk_bbox("Punjab", level = "province")
ggplot() +
  geom_sf(data = districts) +
  coord_sf(xlim = c(bb_punjab["xmin"], bb_punjab["xmax"]),
           ylim = c(bb_punjab["ymin"], bb_punjab["ymax"]))
```

---

pk\_buffer

*Create buffers around sf geometries in km*

---

**Description**

Reprojects to UTM Zone 42N internally so buffer distances are in kilometres, then returns buffers in the original CRS.

**Usage**

```
pk_buffer(x, dist_km)
```

**Arguments**

x	An sf object.
dist_km	Numeric. Buffer distance in kilometres.

**Value**

Returns an sf object with the same attributes as x but with geometries transformed to buffers of radius `dist_km` kilometres.

The output CRS is identical to the input CRS (reprojected back from UTM Zone 42N after buffering). This ensures buffers are circular in projected space with accurate kilometre distances.

**Examples**

```
districts <- get_districts()
buffered <- pk_buffer(districts, dist_km = 10)
```

---

pk_centroid	<i>Extract centroids from an sf object</i>
-------------	--

---

**Description**

Returns polygon centroids as a point sf object with only geometry (attributes are dropped to avoid warnings about constant attributes).

**Usage**

```
pk_centroid(x)
```

**Arguments**

x An sf object with polygon geometries.

**Value**

Returns an sf point object (class "sf") with:

geometry Point geometries representing the centroids of input polygons

Centroids represent the geometric center of each polygon, useful for point-based visualisation, distance calculations, or as nodes for spatial network analysis.

**Examples**

```
districts <- get_districts()
centres <- pk_centroid(districts)
plot(centres)
```

---

pk_crs_suggest	<i>Suggest an appropriate projected CRS for a Pakistan sf object</i>
----------------	--

---

**Description**

Examines the geographic extent of an sf object and recommends the most appropriate projected coordinate reference system for metric operations. Pakistan spans UTM zones 41N, 42N, and 43N; national-level analyses benefit from an equal-area projection.

**Usage**

```
pk_crs_suggest(x)
```

**Arguments**

x                    An sf object.

**Details**

Using WGS84 (EPSG:4326) for distance, area, or buffer operations produces inaccurate results as it measures in degrees rather than metres.

**Value**

A named list:

**epsg** Integer EPSG code for the recommended CRS.

**name** Human-readable CRS name.

**rationale** One-sentence explanation of the recommendation.

**Examples**

```
pk_crs_suggest(get_country())
pk_crs_suggest(get_districts(province = "Balochistan"))
```

---

pk\_dictionary

*Pakistan Administrative Boundaries Dictionary*

---

**Description**

Search and explore administrative names and codes for Pakistan.

**Usage**

```
pk_dictionary(level = NULL, name = NULL, code = NULL)
```

**Arguments**

level                Character. One of: "provinces", "districts", "tehsils". NULL (default) returns all levels.

name                 Character. Filter by partial name (case-insensitive).

code                 Character. Filter by partial P-code (case-insensitive).

**Value**

Returns a data frame (class "data.frame") with the following columns:

name	Administrative unit name (e.g., "Lahore")
level	Administrative level: "provinces", "districts", or "tehsils"
code	P-code (unique identifier for the administrative unit)
parent	Parent administrative unit (province for districts, district for tehsils)

The output represents a searchable dictionary of all administrative units in Pakistan. Use this function to explore available units, find codes, or validate input for other pkmpr functions.

**Case Insensitivity**

All matching in `pk_dictionary()` is **case-insensitive**. "Lahore", "lahore", and "LAHORE" all return the same results.

**Note**

It is recommended to use admin codes, rather than district or tehsil names for instance, as some units especially at tehsil level are almost or completely similar.

**Examples**

```
# All provinces
pk_dictionary(level = "provinces")

# Case-insensitive search for districts containing "lahore"
pk_dictionary(level = "districts", name = "lahore")
pk_dictionary(level = "districts", name = "LAHORE") # Same result

# Search by code
pk_dictionary(code = "PK6")
```

---

pk_distance	<i>Compute distances between two sf objects in km</i>
-------------	---

---

**Description**

Reprojects internally to UTM Zone 42N for accurate metric distances.

**Usage**

```
pk_distance(x, y, by = c("centroid", "edge"))
```

**Arguments**

x	An sf object.
y	An sf object.
by	Character. "centroid" (default) for centroid-to-centroid distances. "edge" for nearest-point-on-boundary distances.

**Value**

Returns a numeric matrix (class "matrix") of distances in kilometres, with dimensions nrow(x) by nrow(y).

When by = "centroid", distances are measured between polygon centroids. When by = "edge", distances are measured between the closest points on polygon boundaries. The output represents the shortest straight-line distance between features.

**Examples**

```
provinces <- get_provinces()
d <- pk_distance(provinces, provinces)
print(d)

# Edge-to-edge distances
d_edge <- pk_distance(provinces, provinces, by = "edge")
```

---

pk\_intersect                      *Intersect two sf objects*

---

**Description**

Returns the geometric intersection of two sf objects with CRS alignment handled automatically.

**Usage**

```
pk_intersect(x, y)
```

**Arguments**

x	An sf object.
y	An sf object.

**Value**

Returns an sf object containing the geometric intersection of x and y:

geometry	Points, lines, or polygons where x and y overlap
...	All attribute columns from both x and y (with suffixes if names conflict)

The output CRS matches x. Empty geometries (no intersection) are dropped. Output represents areas/features common to both input layers.

**Examples**

```
districts <- get_districts()
buffered <- pk_buffer(districts, dist_km = 10)
intersected <- pk_intersect(districts, buffered)
```

---

pk_join	<i>Join data to a pkmpr sf object with match checking</i>
---------	---

---

**Description**

Performs a left join of external data to a pkmpr spatial object, with automatic validation of matching keys. Uses code columns (e.g., district\_code, tehsil\_code, province\_code) wherever possible to ensure reliable joins even when names change or have spelling variations.

**Usage**

```
pk_join(spatial, data, by)
```

**Arguments**

spatial	An sf object from a pkmpr geometry function (e.g., get_districts(), get_tehsils(), get_provinces()).
data	A data frame to join. Must contain the column specified in by.
by	Character. Column name present in both spatial and data. Recommended to use code columns (district_code, tehsil_code, province_code) rather than name columns for more reliable matching.

**Value**

Returns the spatial sf object (class "sf" and "data.frame") with all columns from data joined to the matching rows.

The output preserves:

geometry	The original spatial geometries unchanged
spatial_attributes	All original columns from the spatial object
data_columns	All columns from data appended to matching rows

Rows in data that do not match any spatial unit generate a warning and receive NA values for spatial attributes in the joined result. Rows in the spatial object that have no match in data retain their original attributes but receive NA for the joined data columns.

**Note**

After joining, always inspect names(result) to check for column name conflicts. If your data shares column names with the spatial object (e.g., province\_name, district\_name), both versions will be preserved with .x and .y suffixes. Rename or select the appropriate columns before further analysis.

**Examples**

```

districts <- get_districts()
my_data <- data.frame(district_code = "PK603", value = 42)
joined <- pk_join(districts, my_data, by = "district_code")
print(names(joined))

# Example with missing match (generates warning)

bad_data <- data.frame(district_code = c("PK603", "INVALID_CODE"), value = c(42, 99))
joined_bad <- pk_join(districts, bad_data, by = "district_code")

```

---

pk\_map

*Produces a ggplot2 map for rapid exploratory visualisation.*


---

**Description**

Produces a ggplot2 map for rapid exploratory visualisation.

**Usage**

```
pk_map(x, fill = NULL, title = NULL, palette = "viridis", breaks = NULL, ...)
```

**Arguments**

x	An sf object.
fill	Character. Column name to use as fill variable. NULL (default) produces an outline map.
title	Character. Map title. NULL for no title.
palette	Character. Viridis palette option. One of "viridis" (default), "magma", "plasma", "inferno", or "cividis".
breaks	Numeric vector. Custom legend break points. NULL (default) lets ggplot2 choose automatically.
...	Additional arguments passed to ggplot2::geom_sf().

**Value**

Returns a ggplot object (class "gg" and "ggplot") representing a choropleth map.

When fill = NULL, the output is an outline map with grey90 fill and white borders, useful for context or reference.

When a fill variable is provided, the output uses a viridis color scale with automatic legend, for visualizing spatial distributions of continuous variables (e.g., area, population, density).

The returned ggplot object can be extended with additional layers, themes, or scales using standard ggplot2 syntax.

**Examples**

```
# Outline map of provinces
pk_map(get_provinces())

# Choropleth map with fill variable
pk_map(get_provinces(), fill = "area_km2", title = "Province areas")
```

---

pk\_map\_interactive      *Interactive choropleth map of a pkmapr sf object*

---

**Description**

Produces an interactive leaflet map. Returns a leaflet object that can be extended with standard leaflet functions.

**Usage**

```
pk_map_interactive(x, fill = NULL, popup = NULL, ...)
```

**Arguments**

x	An sf object.
fill	Character. Column name for choropleth fill. NULL produces an outline map.
popup	Character vector. Column names to display in click popups.
...	Additional arguments passed to <code>leaflet::addPolygons()</code> .

**Value**

Returns a leaflet object (class "leaflet" and "htmlwidget") representing an interactive web map.

When `fill = NULL`, the output shows polygon outlines only.

When a fill variable is provided, the output renders polygons with:

fillColor	Color-coded by the fill variable using the viridis palette
fillOpacity	0.7 (semi-transparent for layer visibility)
color	White borders for polygon boundaries
popup	HTML popups showing selected attributes on click

The map includes a legend for the fill variable and uses the CartoDB.Positron tile provider as the basemap.

**Examples**

```
districts <- get_districts()
pk_map_interactive(districts,
  fill = "area_km2",
  popup = c("district_name", "area_km2"))
```

---

 pk\_neighbors

 Construct a spatial neighbours list for Pakistan administrative units
 

---

### Description

Builds a contiguity or distance-based spatial neighbours structure for direct use with `spdep` and `spatialreg`.

### Usage

```
pk_neighbors(
  x,
  style = c("queen", "rook", "knn", "idw"),
  k = NULL,
  disputed = c("include", "exclude_both", "exclude_gb", "exclude_ajk")
)
```

### Arguments

<code>x</code>	An <code>sf</code> object with polygon geometries, typically the output of <code>get_districts()</code> or <code>get_provinces()</code> .
<code>style</code>	Character. Neighbour definition: "queen" (shared boundary point, default), "rook" (shared edge), "knn" (k nearest centroids), or "idw" (inverse distance weights).
<code>k</code>	Integer. Number of nearest neighbours. Required when <code>style = "knn"</code> .
<code>disputed</code>	Character. Controls whether Gilgit-Baltistan and/or Azad Jammu & Kashmir are included in the spatial weights structure: "include" Default. Both units are treated as normal administrative units and participate in the neighbour graph. "exclude_both" Both GB and AJK are dropped from <code>x</code> before building the weights. The returned data element contains the subsetted <code>sf</code> object. "exclude_gb" Only Gilgit-Baltistan is dropped. "exclude_ajk" Only Azad Jammu & Kashmir is dropped.

### Value

A named list with the following elements:

<code>nb</code>	An <code>spdep nb</code> object containing the neighbour relationships. Each element is an integer vector of neighbour indices.
<code>listw</code>	A row-standardised <code>spdep listw</code> object ( <code>style = "W"</code> , <code>zero.policy = TRUE</code> ), ready for <code>spdep::moran.test()</code> , <code>spdep::localmoran()</code> , <code>spatialreg::lagsarlm()</code> , and related functions.
<code>data</code>	The <code>sf</code> object used to build the weights. Identical to the input <code>x</code> when <code>disputed = "include"</code> . When any units are excluded this is the subsetted <code>sf</code> , always in row-for-row alignment with <code>nb</code> and <code>listw</code> . Always use <code>w\$data</code> rather than the original input when mapping or attaching analysis results.

**Disputed units**

GB and AJK are present in the OCHA/HDX boundary data and share polygon edges with KP and Punjab, so under `disputed = "include"` (the default) they participate in the neighbour graph as normal administrative units. Use `"exclude_gb"`, `"exclude_ajk"`, or `"exclude_both"` to drop one or both units before building the weights. The returned data element is always the `sf` object actually used to build the weights — identical to the input when `disputed = "include"`, subsetted otherwise. Always use `w$data` rather than the original input for all subsequent analysis and mapping, regardless of which disputed option is chosen.

**Examples**

```
districts <- get_districts()

# Default: all units included
w <- pk_neighbors(districts)
moran_result <- spdep::moran.test(w$data$area_km2, w$listw)
print(moran_result)

# Rook contiguity
w_rook <- pk_neighbors(districts, style = "rook")

# K-nearest neighbours (k = 5)
w_knn <- pk_neighbors(districts, style = "knn", k = 5)

# Inverse distance weights
w_idw <- pk_neighbors(districts, style = "idw")

# Exclude both GB and AJK for sensitivity analysis
w_excl <- pk_neighbors(districts, disputed = "exclude_both")
# Use w_excl$data – not the original districts – for subsequent analysis
moran_excl <- spdep::moran.test(w_excl$data$area_km2, w_excl$listw)

# Exclude only Gilgit-Baltistan
w_no_gb <- pk_neighbors(districts, disputed = "exclude_gb")

# Exclude only Azad Jammu & Kashmir
w_no_ajk <- pk_neighbors(districts, disputed = "exclude_ajk")
```

---

pk\_points\_in

*Assign points to administrative units (point-in-polygon)*

---

**Description**

For each point in `points`, identifies which polygon it falls within and joins that polygon's attributes to the point record.

**Usage**

```
pk_points_in(points, polygons, return_all = TRUE)
```

**Arguments**

points	An sf object with point geometries.
polygons	An sf object with polygon geometries.
return_all	Logical. Keep unmatched points with NA polygon attributes (TRUE, default) or drop them (FALSE).

**Value**

Returns an sf object of the same class as points with:

geometry	Point geometries (unchanged)
...	Attribute columns from polygons joined to matching points

Points that fall outside all polygons receive NA values for all polygon attributes when return\_all = TRUE. When return\_all = FALSE, such points are removed entirely.

The output represents point locations with their containing administrative unit attributes attached, allowing spatial aggregations and unit-based analyses.

**Examples**

```
# Get district boundaries
districts <- get_districts()

# Create sample points (or use your own sf object)
set.seed(123)
sample_points <- sf::st_sample(districts, size = 50)
sample_points_sf <- sf::st_sf(geometry = sample_points)

# Assign points to districts
points_with_districts <- pk_points_in(sample_points_sf, districts)
print(head(points_with_districts))
```

---

pk\_project

*Project an sf object to a Pakistan-appropriate CRS*


---

**Description**

Convenience wrapper around `sf::st_transform()` with a default of UTM Zone 42N, appropriate for most Pakistan analyses requiring accurate distance, area, or buffer operations.

**Usage**

```
pk_project(x, crs = 32642)
```

**Arguments**

x	An sf object.
crs	Integer EPSG code. Default 32642 (WGS84 / UTM Zone 42N).

**Value**

Returns the sf object reprojected to the specified CRS. The output has the same attributes and geometry type as the input, but coordinates are transformed to the new projection. UTM Zone 42N (EPSG:32642) preserves distances and areas accurately across most but not all of Pakistan.

**Examples**

```
districts <- get_districts()
projected <- pk_project(districts)
sf::st_crs(projected)$epsg # Should be 32642
```

---

pk_search	<i>Search Across All Administrative Levels</i>
-----------	--

---

**Description**

Search for administrative units by partial name or code across all levels (provinces, districts, tehsils).

**Usage**

```
pk_search(query, fuzzy = FALSE)
```

**Arguments**

query	Character. Search term (partial match, case-insensitive).
fuzzy	Logical. If TRUE, uses fuzzy matching for typos. Default FALSE. Warning: Fuzzy matching can be slower and may return unexpected matches for short or common queries.

**Value**

Returns a data frame (class "data.frame") with the following columns:

name	Matching administrative unit name(s)
level	Administrative level of each match
code	P-code of each match
parent	Parent unit of each match

The output represents all administrative units that match the search query. When no matches are found, returns invisible NULL with a warning message. When fuzzy matching is enabled, the output may include approximate matches that could be useful for handling typos or spelling variations.

**Case Insensitivity**

All matching in `pk_search()` is **case-insensitive** by default. "Lahore", "lahore", and "LAHORE" all return the same results.

## Fuzzy Matching

When `fuzzy = TRUE`, the function uses approximate string matching to handle typos and spelling variations. For example, "lahor" will match "Lahore". This is useful for interactive exploration but may return unexpected results for ambiguous queries.

## Examples

```
# Case-insensitive search
pk_search("lahore") # Returns Lahore district and tehsils
pk_search("LAHORE") # Same result

# Fuzzy search for misspelled "lahore"
pk_search("lahor", fuzzy = TRUE)
pk_search("lahre", fuzzy = TRUE)

# Search by code
pk_search("PK6")
```

---

pk\_union

*Dissolve sf polygons by a grouping column*

---

## Description

Merges polygons sharing the same value in a grouping column and recalculates `area_km2`.

## Usage

```
pk_union(x, by)
```

## Arguments

<code>x</code>	An sf object.
<code>by</code>	Character. Column name to group by.

## Value

Returns a dissolved sf object (class "sf" and "data.frame") with:

<code>geometry</code>	MULTIPOLYGON geometries created by merging adjacent polygons
<code>by_column</code>	The unique grouping values (one row per group)
<code>area_km2</code>	Recalculated area in square kilometres for each dissolved polygon

Polygons that are not spatially adjacent but share the same group value will become MULTIPOLYGON objects. Invalid geometries are repaired automatically using `sf::st_make_valid()`.

## Examples

```
tehsils <- get_tehsils()
by_district <- pk_union(tehsils, by = "district_name")
```

---

pk_version	<i>Check package version and update status</i>
------------	--

---

**Description**

Compares the installed version of pkmapr with the latest release available on GitHub.

**Usage**

```
pk_version(quiet = FALSE)
```

**Arguments**

quiet	Logical. If FALSE (default), prints status messages to console. If TRUE, returns the version information silently.
-------	--

**Value**

Returns invisibly a list (class "list") with the following components:

installed	Character string. The currently installed version number.
latest	Character string or NA. The latest version number from GitHub, or NA if the check failed (e.g., no internet connection).

**Examples**

```
pk_version()

# Silent mode for programmatic use
vers <- pk_version(quiet = TRUE)
if (!is.na(vers$latest) && vers$installed != vers$latest) {
  message("Update recommended!")
}
```

# Index

[get\\_country](#), [2](#)  
[get\\_districts](#), [3](#)  
[get\\_provinces](#), [4](#)  
[get\\_tehsils](#), [5](#)

[pk\\_area](#), [6](#)  
[pk\\_basemap](#), [6](#)  
[pk\\_bbox](#), [7](#)  
[pk\\_buffer](#), [8](#)  
[pk\\_centroid](#), [9](#)  
[pk\\_crs\\_suggest](#), [9](#)  
[pk\\_crs\\_suggest\(\)](#), [2–5](#)  
[pk\\_dictionary](#), [10](#)  
[pk\\_distance](#), [11](#)  
[pk\\_intersect](#), [12](#)  
[pk\\_join](#), [13](#)  
[pk\\_map](#), [14](#)  
[pk\\_map\\_interactive](#), [15](#)  
[pk\\_neighbors](#), [16](#)  
[pk\\_points\\_in](#), [17](#)  
[pk\\_project](#), [18](#)  
[pk\\_search](#), [19](#)  
[pk\\_union](#), [20](#)  
[pk\\_version](#), [21](#)