Package: cd2030.core (via r-universe)

October 12, 2026

Title Data Loading, Validation, and Reporting for Countdown 2030

Version 1.4.4

Description cd2030.core is an R package for managing and validating data formatted according to Countdown 2030 standards. It provides tools for data loading, cleaning, and validation, detecting issues like missing values, inconsistencies, outliers, and ratio anomalies. The package ensures compliance with Countdown 2030 guidelines and generates standardized data quality reports, enabling users to focus on analysis and interpretation.

License AGPL (>= 3)

URL https://github.com/damurka/cd2030.core

Imports cli (>= 3.6.2), datasuite.ui (>= 0.5.2), dplyr (>= 1.1.4), flextable (>= 0.10.0), forcats (>= 1.0.1), future (>= 1.70.0), ggplot2 (>= 3.5.1), haven (>= 2.5.4), htmltools (>= 0.5.9), later (>= 1.4.8), lubridate (>= 1.9.4), openxlsx (>= 4.2.8), officer (>= 0.7.5), promises (>= 1.5.0), purrr (>= 1.2.2), RColorBrewer (>= 1.1.3), readr (>= 2.2.0), readxl (>= 1.4.3), rlang (>= 1.1.0), scales (>= 1.3.0), reactable (>= 0.4.5), sf (>= 1.1-1), shiny (>= 1.14.0), shiny.react (>= 0.4.0), stringdist (>= 0.9.15), stringi (>= 1.8.4), stringr (>= 1.6.0), tibble (>= 3.2.1), tidyr (>= 1.3.1), withr (>= 3.0.2)

Remotes damurka/datasuite.ui

Additional_repositories https://alkemalab.r-universe.dev

Suggests bayescoveragedeploy, bayescoveragemodel, ggrepel, jsonlite, knitr, magick, pdftools, rmarkdown, spelling, testthat (>= 3.0.0), xml2, zip

Config/datasuite/onDemand bayescoveragemodel, bayescoveragedeploy

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

LazyData true

BugReports https://github.com/damurka/cd2030.core/issues

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-10-12 09:34:40 UTC

RemoteUrl https://github.com/damurka/cd2030.core

RemoteRef main

RemoteSha c6ea74d66f3dac2b681dbb2697e8e1ba5478ba6f

Contents

add_mad_med_columns . . . 9
add_missing_column . . . 10
add_outlier5std_column . . . 11
adjust_service_data . . . 12
adjustment_changes_ui . . . 14
adjustment_evidence . . . 15
adjustment_settings_check . . . 16
adjustment_settings_default . . . 17
adjustment_settings_footnote . . . 17
adjustment_settings_from_k . . . 18
adjustment_steps_used . . . 19
bayes_model_estimates . . . 20
bayesian_page_server . . . 20
bayesian_ui . . . 21
cache_definition . . . 23
cache_manifest . . . 24
calculate_average_reporting_rate . . . 24
calculate_completeness_summary . . . 25
calculate_coverage . . . 26
calculate_derived_coverage . . . 27
calculate_district_completeness_summary . . . 28
calculate_district_outlier_summary . . . 28
calculate_district_ratios_summary . . . 29
calculate_district_reporting_rate . . . 30
calculate_health_system_comparison . . . 31
calculate_health_system_metrics . . . 32
calculate_indicator_coverage . . . 33

calculate_inequality . . . 35
calculate_outliers_summary . . . 36
calculate_overall_score . . . 37
calculate_overall_score1 . . . 38
calculate_ratios_and_adequacy . . . 39
calculate_ratios_summary . . . 41
calculate_ratios_ui . . . 42
calculate_threshold . . . 43
cd_admin_level_choices . . . 44
cd_admin_level_ui . . . 45
cd_admin_parts . . . 47
cd_app . . . 48
cd_bayes_fit_task . . . 50
cd_bayes_indicators . . . 51
cd_bayes_install_command . . . 51
cd_bayes_packages_missing . . . 52
cd_cfg . . . 52
cd_chart_catalog . . . 54
cd_chart_id . . . 54
cd_chartable_members . . . 55
cd_coverage_plot_ui . . . 56
cd_custom_chart_data . . . 57
cd_default_indicator_set . . . 58
cd_default_indicators . . . 59
cd_denominator_chip_keys . . . 60
cd_denominator_choices . . . 60
cd_denominator_header . . . 61
cd_denominator_labels . . . 62
cd_denominator_options . . . 63
cd_denominator_row . . . 63
cd_denominator_ui . . . 64
cd_describe_columns . . . 66
cd_dictionary . . . 66
cd_finish_plot . . . 67
cd_has_maternal . . . 68
cd_indicator_labels . . . 68
cd_indicator_ui . . . 69
cd_member_data . . . 70
cd_methodology_defaults . . . 71
cd_minimal_theme . . . 72
cd_nav_start . . . 72
cd_only_denominators . . . 74
cd_palette_chip . . . 75
cd_plot_arguments . . . 76
cd_population_ui . . . 77
cd_rate_status_cell . . . 78
cd_report_context . . . 79
cd_request_bayes_packages . . . 80

cd_saved_copy_name . . . 80
cd_scope . . . 81
cd_scope_filters . . . 82
cd_scope_server . . . 83
cd_scoped_page_ui . . . 84
cd_start_bayes_workers . . . 85
cd_tab_indicator . . . 86
cd_tabbed_charts_ui . . . 87
cd_table_card_ui . . . 89
cd_table_ui . . . 91
cd_type_scale . . . 94
cd_upload_data_ui . . . 94
cd_wizard_check_group . . . 96
cd_wizard_config . . . 97
cd_wizard_field_group . . . 98
cd_wizard_indicator_group . . . 99
cd_wizard_national_rate_fields . . . 99
cd_wizard_survey_field . . . 100
cd_years_input . . . 101
cd_years_sync . . . 101
check_admin_columns . . . 102
check_admin_pairing_consistency . . . 103
check_country_recognized . . . 104
check_district_consistency . . . 105
check_district_cross_sheet . . . 106
check_district_year_completeness . . . 107
check_indicator_emptiness . . . 107
check_indicator_emptiness_presheet . . . 108
check_month_language_consistency . . . 109
check_month_presence . . . 110
check_month_presence_presheet . . . 110
check_month_validity . . . 111
check_month_validity_presheet . . . 112
check_population_service_collision . . . 113
check_population_service_collision_presheet . . . 114
check_population_vs_births . . . 115
check_required_columns_presheet . . . 116
check_shapefile_admin_names . . . 117
check_single_country . . . 118
check_survey_admin_names . . . 119
clean_error_message . . . 120
compute_indicator_numerator . . . 121
compute_service_utilization . . . 122
compute_step_states . . . 123
consistency_check_ui . . . 125
consistency_pair_keys . . . 126
consistency_pair_label . . . 126
countdown-pages . . . 127

coverage_trends_ui . . . 129
coverage_ui . . . 131
create_mortality_ratios . . . 132
create_mortality_summary . . . 133
data_adjustment_ui . . . 134
data_completeness_ui . . . 135
data_quality_ui . . . 136
den_indicators . . . 137
denominator_assessment_ui . . . 138
denominator_selection_ui . . . 139
detect_indicator_group . . . 140
equiplot . . . 141
equiplot_area . . . 142
equiplot_education . . . 143
equiplot_wealth . . . 145
equity_ui . . . 146
filter_high_performers . . . 147
filter_out_years . . . 148
generate_adjustment_values . . . 149
generate_admin1_keys . . . 151
generate_admin1_mch_curative_index . . . 152
generate_admin1_service_utilization . . . 152
generate_bayes_model . . . 153
generate_coverage_data . . . 154
generate_fpet_summary . . . 156
generate_health_system_table . . . 156
generate_phc_scatter_data . . . 157
generate_private_sector_data . . . 158
generate_service_dqa_summary . . . 159
get_admin_columns . . . 160
get_all_indicators . . . 161
get_analysis_indicators . . . 162
get_country_iso3 . . . 162
get_country_name . . . 163
get_country_shapefile . . . 163
get_coverage_indicators . . . 164
get_excel_version . . . 165
get_indicator_group_names . . . 166
get_indicator_groups . . . 166
get_mapping_data . . . 167
get_named_indicators . . . 167
get_national_rates . . . 168
get_plot_admin_column . . . 169
get_population_column . . . 170
get_selected_group . . . 171
inequality_ui . . . 171
init_CacheConnection . . . 173
internal_consistency_ui . . . 174

interpret . . . 175
interpret.cd_fpet_data . . . 176
introduction_ui . . . 176
is_maternal_indicator . . . 177
list_indicator_groups . . . 178
list_missing_units . . . 178
list_outlier_units . . . 179
list_tracer_vaccines . . . 180
list_vaccine_indicators . . . 180
load_cache_data . . . 181
load_csection_estimates . . . 182
load_data . . . 184
load_equity_data . . . 186
load_excel_parts . . . 186
load_fpet_data . . . 188
load_private_sector_data . . . 189
load_survey_data . . . 190
load_un_estimates . . . 191
load_un_mortality_data . . . 192
load_wuenic_data . . . 193
map_shapefile_ui . . . 193
map_survey_ui . . . 194
mapping_modal_ui . . . 195
mapping_provenance_banner . . . 197
merge_and_standardize . . . 198
national_coverage_ui . . . 199
national_inequality_ui . . . 200
national_rates_required_fields . . . 201
national_rates_ui . . . 202
national_target_ui . . . 203
new_countdown . . . 204
notebook_data . . . 205
nr_field . . . 207
nr_group . . . 208
outlier_detection_ui . . . 209
overall_score_ui . . . 210
plot.cd_adjustment_values_filtered . . . 211
plot.cd_average_reporting_rate . . . 213
plot.cd_bayes_model . . . 215
plot.cd_completeness_summary . . . 216
plot.cd_coverage . . . 218
plot.cd_coverage_filtered . . . 219
plot.cd_coverage_selected . . . 221
plot.cd_derived_coverage . . . 222
plot.cd_district_reporting_rate . . . 223
plot.cd_fpet_data . . . 225
plot.cd_health_system_comparison . . . 227
plot.cd_health_system_metric . . . 228

plot.cd_health_system_table . . . 229
plot.cd_indicator_coverage . . . 231
plot.cd_inequality_filtered . . . 232
plot.cd_mapping_filtered . . . 234
plot.cd_mch_curative_index . . . 235
plot.cd_missing_district . . . 236
plot.cd_mortality_ratio_summarised . . . 238
plot.cd_mortality_summary . . . 239
plot.cd_mortality_summary_filtered . . . 240
plot.cd_outlier . . . 242
plot.cd_outlier_list . . . 243
plot.cd_overall_score . . . 245
plot.cd_phc_scatter . . . 246
plot.cd_population_metrics . . . 247
plot.cd_private_sector . . . 249
plot.cd_private_sector_plot_data . . . 250
plot.cd_ratios_summary . . . 252
plot.cd_service_utilization_admin1 . . . 253
plot.cd_service_utilization_filtered . . . 254
plot.cd_service_utilization_prepared . . . 256
plot.cd_threshold . . . 257
plot.cd_utilization_dqa . . . 258
plot_comparison . . . 259
plot_comparison.cd_data . . . 262
plot_line_graph . . . 264
plot_mortality_plausibility . . . 265
plot_national_health_metric . . . 266
prepare_mapping_service_utlization . . . 267
prepare_population_metrics . . . 268
prepare_private_sector_plot_data . . . 269
print.cd_data . . . 270
push_upload_on_remount . . . 271
quality_issues_table . . . 273
read_shapefile_folder . . . 273
reference_estimates_ui . . . 274
register_indicator_group . . . 275
report_block_kinds . . . 277
report_presets . . . 278
reporting_rate_ui . . . 278
reset_indicator_group . . . 280
resolve_country_best_effort . . . 280
restore_default_control . . . 281
robust_max . . . 282
rr_indicators . . . 283
run_all_quality_checks . . . 284
save_data . . . 285
save_dhis2_excel . . . 286
save_dhis2_master_data . . . 287

select_adjustment_value 288
select_coverage 288
select_inequality 289
select_mapping_data 290
select_mortality_summary 291
select_service_utilization 292
set_selected_group 293
shapefile_step_ui 294
slim_bayes_model 295
step_upload_complete 295
subnational_coverage_ui 297
subnational_denominator_ui 298
subnational_inequality_ui 299
subnational_mapping_ui 300
subnational_target_ui 301
summarise_completeness_ratio 302
survey_comparison_ui 303
survey_fields 304
survey_upload_ui 305
target_ui 306
tbl_sum.cd_adjustment_values 307
tbl_sum.cd_average_reporting_rate 308
tbl_sum.cd_completeness_summary 309
tbl_sum.cd_district_outliers_summary 310
tbl_sum.cd_district_ratios_summary 310
tbl_sum.cd_district_reporting_rate 311
tbl_sum.cd_indicator_coverage 312
tbl_sum.cd_missing_district 313
tbl_sum.cd_outlier 314
tbl_sum.cd_population_metrics 314
tbl_sum.cd_ratios_summary 315
upload_box_ui 316
with_cd_quiet 318
wizard_fields 319
wizard_landing_ui 319
wizard_step_defs 321
wizard_steps_ui 322
write_quality_issues 324

Index **325**

`add_mad_med_columns` *Add median and MAD columns for indicators*

Description

`add_mad_med_columns` calculates and appends the median and Median Absolute Deviation (MAD) for specified indicators in a dataset. These statistics are calculated per group, allowing the user to dynamically group data (e.g., by `district`) and exclude data from the most recent year in the dataset.

Usage

```
add_mad_med_columns(.data, indicators, group_by = "district")
```

Arguments

`.data`	A `cd_data` tibble containing health indicator data.
`indicators`	A character vector specifying the names of the indicator columns for which the median and MAD should be calculated.
`group_by`	A character string or vector specifying the column(s) to group by when calculating the median and MAD (default is `'district'`).

Details

- The median and MAD are calculated for each group specified by `group_by`.
- Only data from years prior to the most recent year in the dataset are considered.
- Missing values in the calculated statistics are replaced using `robust_max()`, ensuring that meaningful fallback values are provided.

Value

A tibble with additional columns for each indicator, named `{indicator}_med` and `{indicator}_mad`, containing the calculated median and MAD, respectively.

See Also

- `add_outlier5std_column()` for generating outlier flags based on these columns.
- `robust_max()` for calculating max value

Examples

```
## Not run:
# Add median and MAD columns for 'indicator1'
add_mad_med_columns(data,
  indicators = "indicator1",
  group_by = "district"
)
```

```
## End(Not run)
```

`add_missing_column` *Add missing value flags to health indicators*

Description

`add_missing_column` assesses data quality by identifying missing values for specified health indicators. It generates new columns for each indicator, flagging values as 'Missing' (`1`) or 'Non-Missing' (`0`) to support downstream quality control and analysis.

Usage

```
add_missing_column(.data, indicators)
```

Arguments

`.data`	A data frame containing health indicator data.
`indicators`	A character vector specifying the names of the indicator columns to analyze for outliers.

Details

- **Missingness Assessment**: Each indicator is processed to flag missing values (`NA`) with `1` (missing) or `0` (non-missing). The resulting columns are prefixed with `mis_` followed by the original indicator name (e.g., `mis_indicator1`).
- **Group Preservation**: The output retains the `district` and `year` columns to ensure grouping information remains intact.

Value

A data frame containing:

- **Missingness Flags**: For each indicator, a column prefixed with `mis_` indicating missingness (`1` for missing, `0` otherwise).
- `district` and `year` columns to maintain grouping information.

Examples

```
## Not run:
# dd missing value flags for all indicators
add_missing_column(data)

## End(Not run)
```

`add_outlier5std_column` *Add outlier flags based on 5-MAD bounds*

Description

`add_outlier5std_column` calculates and appends outlier flags for specified indicators in a dataset. An outlier is defined as a value falling outside the range defined by the median ± 5 times the Median Absolute Deviation (MAD). This method is applied on a per-group basis, allowing flexibility for grouping by columns like `district`.

Usage

```
add_outlier5std_column(.data, indicators, group_by = "district")
```

Arguments

`.data`	A `cd_data` tibble containing health indicator data.
`indicators`	A character vector specifying the names of the indicator columns to analyze for outliers.
`group_by`	A character string or vector specifying the column(s) to group by when calculating the median and MAD (default is `'district'`).

Details

- **Median and MAD Calculation**: For each indicator, the median and MAD are calculated using data from years prior to the most recent year in the dataset. The last year is excluded to prevent potential contamination of the baseline statistics.
- **Outlier Definition**: A value is flagged as an outlier if it falls outside the range: Lower Bound = median - 5 * MAD Lower Bound = median + 5 * MAD
- **Grouping**: Calculations are performed separately for each group specified by the `group_by` parameter.
- **Generated Columns**: For each indicator, the function generates an outlier flag column named `{indicator}_outlier5std`. The column contains `1` if the value is an outlier and `0` otherwise.

Value

A tibble with additional columns for each indicator, named `{indicator}_outlier5std`, containing a value of `1` if the observation is an outlier and `0` otherwise.

See Also

`add_mad_med_columns()` for computing and appending the median and MAD columns.

Examples

```
## Not run:
# Add missing value flags for all indicators
add_outlier5std_column(data,
  indicators = "indicator1",
  group_by = "district"
)

## End(Not run)
```

`adjust_service_data` *Adjust service data for coverage analysis*

Description

The `adjust_service_data` function processes DHIS-2 health service data, correcting for incomplete reporting, applying k-factor adjustments for accurate scaling, and managing outliers for consistency in data analysis. This standardization supports reliable comparisons in immunization coverage and health service utilization studies.

Usage

```
adjust_service_data(
  .data,
  adjustment = c("default", "custom", "none"),
  k_factors = NULL,
  settings = NULL,
  steps = FALSE
)
```

Arguments

`.data`	A `cd_data` dataframe, typically containing health facility data from DHIS-2 with monthly service counts by district.
`adjustment`	A character string specifying the type of adjustment to apply: • **"default"**: Applies a default k-factor of 0.25 across all indicator groups. • **"custom"**: Uses user-defined `k_factors` for different indicator groups, with values between 0 and 1. • **"none"**: Returns the data without any adjustments.
`k_factors`	A named numeric vector of custom k-factor values between 0 and 1 for each indicator group (e.g., `c(anc = 0.3, idelv = 0.2, ...)`). Used only if `adjustment = "custom"`.
`settings`	The adjustment settings (`adjustment_settings_default()`; the Data Adjustment page's): the years and areas removed, and each indicator's k, outlier and missing-value switches, everywhere or for an area. When given, `adjustment` and `k_factors` are not used.

`steps` `TRUE`: a list of the data after each step (`completeness`, `outliers`, `missing`), for showing what each changed (see `generate_adjustment_values()`).

Details

Prepares service data through a series of steps to ensure data quality and consistency:

1. **Validation**: Checks the structure of `.data` and ensures that the `adjustment` argument is valid. For `custom` adjustments, `k_factors` must be specified and contain valid values.
2. **k-Factor Defaults**: Default k-factor values are set to 0.25 for each indicator group, unless overridden by user-provided values in `k_factors`.
3. **Reporting Completeness**: Flags any district-year reporting rates below 75% and imputes missing data using district-level medians to account for reporting inconsistencies.
4. **k-Factor Scaling**: Adjusts service counts based on the k-factor and reporting rate using the following scaling formula:

$$AdjustedValue = Value \times \left(1 + \left(\frac{1}{ReportingRate/100} - 1\right) \times k\right)$$

   where `ReportingRate` is the reporting rate in percent, and `k` is the k-factor for the indicator group.
5. **Outlier Detection**: Identifies and flags extreme outliers using Hampel's X84 method, marking values that exceed 5 Median Absolute Deviations (MAD) from the median.
6. **Data Imputation**: Replaces remaining missing values with district-level medians to ensure data completeness.

Value

A `cd_adjusted_data` object containing adjusted service data, where outliers are flagged and managed, and missing values are imputed.

See Also

`new_countdown()` for creating `cd_data` objects and `generate_adjustment_values()` for generating adjustment summaries.

Examples

```
## Not run:
# Default adjustment
adjusted_data <- adjust_service_data(data, adjustment = "default")

# Custom adjustment with specific k-factors
custom_k <- c(
  anc = 0.3, idelv = 0.2, pnc = 0.35, vacc = 0.4,
  opd = 0.3, ipd = 0.25
)
adjusted_data_custom <- adjust_service_data(data,
  adjustment = "custom",
```

```
  k_factors = custom_k
)

# No adjustment
unadjusted_data <- adjust_service_data(data, adjustment = "none")

## End(Not run)
```

`adjustment_changes_ui` *Data adjustment changes page*

Description

The page that shows what the adjustment changed, one indicator at a time. A chart sets each year's reported count beside the adjusted count, which is made of its parts (completeness, outliers and missing values), and a table gives the same numbers with the change in percent. A step the indicator does not get is marked "Off". Above the chart, the page lists the adjustment's rules as the reports' footnote gives them (`adjustment_settings_footnote()`) and says whether the data has been adjusted.

Usage

```
adjustment_changes_ui(id, i18n)

adjustment_changes_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does not work the changes out before that.

Details

The indicator is chosen with the card's tabs (the app's `adjustment_indicators` setting) or with the filter for the other indicators, which wins while it has a value. The area filter is every area, one region or one district. The page reads `data_with_excluded_years`, `adjustment_settings`, `countdown_data` and `adjusted_flag` from the dataset, and works the numbers out with `generate_adjustment_values()`.

Value

`adjustment_changes_ui()` returns the page's UI as Shiny tags. `adjustment_changes_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "data_adjustment_changes",
  ui = adjustment_changes_ui,
  server = adjustment_changes_server,
  title = "title_adjust_changes",
  section = "lbl_nav_section_quality",
  subtitle = "sub_adjust_changes"
)

## End(Not run)
```

---

adjustment_evidence *What the data quality checks found, for the Data Adjustment page*

Description

The numbers the Data Adjustment page shows beside its settings, from the same computations as the check pages (Reporting rate, Outlier detection, Data completeness): each indicator group's national reporting rate in `year` and how many districts report below `threshold`; each indicator's outlier months (more than 5 x MAD from the district's median) and empty months; each region's and district's lowest group reporting rate in `year`.

Usage

```
adjustment_evidence(
  .data,
  threshold = .cd_method$data_quality$reporting_threshold,
  year = NULL
)
```

Arguments

`.data`	A `cd_data` tibble (the data before adjustment, its removed years already left out).
`threshold`	Numeric. The performance threshold (in percent) of the Reporting rate page.
`year`	The year of the reporting rates (by default the latest).

Value

A list: `year`, `threshold`, `groups` (named by group: `rr`, `below`), `indicators` (named by indicator: `outliers`, `missing`, months), `areas` (a list of `list(region, rr, districts = list(list(name, rr)))`).

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
evidence <- adjustment_evidence(data)
str(evidence$groups$opd)
str(evidence$indicators$penta3)

# Against a stricter threshold, for an earlier year
adjustment_evidence(data, threshold = 95, year = 2021)$groups$opd
```

---

adjustment_settings_check

Check and tidy adjustment settings

Description

Validates settings as the Data Adjustment page sends them (JSON read with `simplifyVector = FALSE`) and returns them in one shape: numbers as numbers, flags as logicals, empty maps as empty lists, unknown groups, indicators and levels refused.

Usage

```
adjustment_settings_check(settings)
```

Arguments

`settings`	The settings list.

Value

The settings, tidied. An error says what is wrong.

Examples

```
# Settings as they arrive from JSON: numbers and flags inside lists
settings <- list(
  removed_years = list(2020, 2021),
  everywhere = list(group_k = list(anc = 0.1), outliers = list(penta3 = list(FALSE))),
  areas = list(list(area = "Coast", level = "adminlevel_1", all_missing = list(FALSE)))
)
str(adjustment_settings_check(settings))

# What is wrong is refused with an error
try(adjustment_settings_check(list(everywhere = list(group_k = list(anc = 2)))))
```

adjustment_settings_default

The Countdown default adjustment settings

Description

Every indicator group adjusted for completeness with the method's k (0.25), outliers corrected and missing values filled everywhere, no year or area removed.

Usage

```
adjustment_settings_default(k = .cd_method$adjustment$k, groups = NULL)
```

Arguments

k	The k of every group (by default the method's, `0.25`).
groups	The groups that have a k (by default the app's, `cd_cfg("k_factors")`, else the method's).

Value

The settings list (see `adjust_service_data()`).

Examples

```
settings <- adjustment_settings_default()
str(settings)

# No completeness adjustment for any group
adjustment_settings_default(k = 0)$everywhere$group_k
```

adjustment_settings_footnote

Describe the adjustment settings as a report footnote

Description

Writes the adjustment settings out in sentences, for the footnote of a report: the years and areas removed, then the rules that apply everywhere and those of each area (k factors, and whether outliers are corrected and missing values filled).

Usage

```
adjustment_settings_footnote(
  settings,
  group_labels = NULL,
  indicator_labels = NULL
)
```

Arguments

`settings`	The settings list.
`group_labels, indicator_labels`	Named character: how groups and indicators are called (by default their ids).

Value

A character vector, one sentence per rule. The Countdown default is one rule (each group's k, everywhere); settings that change nothing give none.

Examples

```
settings <- adjustment_settings_default(k = 0.1, groups = c("anc", "vacc"))
settings$removed_years <- 2020
settings$everywhere$outliers <- list(penta3 = FALSE)
settings$areas <- list(list(area = "Coast", level = "adminlevel_1", all_missing = FALSE))
adjustment_settings_footnote(settings)

# With the names a report uses
adjustment_settings_footnote(
  settings,
  group_labels = c(anc = "Antenatal care", vacc = "Immunization"),
  indicator_labels = c(penta3 = "Penta 3")
)
```

`adjustment_settings_from_k` *Adjustment settings from the older k-factors and excluded years*

Description

A dataset saved before the settings existed has only its k per group and its removed years: they become the settings, everything else as the default.

Usage

```
adjustment_settings_from_k(k_factors = NULL, excluded_years = numeric())
```

Arguments

`k_factors`	Named numeric k per group.
`excluded_years`	Numeric years removed.

Value

The settings list.

Examples

```
settings <- adjustment_settings_from_k(c(anc = 0.1, vacc = 0), excluded_years = 2020)
settings$everywhere$group_k
settings$removed_years
```

`adjustment_steps_used` *Which adjustment steps an indicator gets*

Description

Whether completeness (a k above 0), outlier correction and filling missing values apply to an indicator in at least one district of the data (or of one area), with the settings: a step no district gets is off.

Usage

```
adjustment_steps_used(
  settings,
  .data,
  indicator,
  area = NULL,
  level = c("adminlevel_1", "district")
)
```

Arguments

`settings`	The settings list.
`.data`	A `cd_data` tibble (its districts and regions).
`indicator`	The indicator.
`area`, `level`	One area only (as `generate_adjustment_values()`); `NULL`, all.

Value

A named logical: `completeness`, `outliers`, `missing`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
settings <- adjustment_settings_default()
adjustment_steps_used(settings, data, "penta3")

# A region with its own settings: no completeness adjustment, outliers left as they are
settings$areas <- list(
  list(area = "Coast", level = "adminlevel_1", group_k = list(vacc = 0), all_outliers = FALSE)
)
adjustment_steps_used(settings, data, "penta3", area = "Coast")
adjustment_steps_used(settings, data, "penta3")
```

bayes_model_estimates *A Bayesian model's estimates as a table*

Description

The fitted coverage by year (2010 to 2030), in percent like the other coverage tables: the median with its 95% and 80% intervals. A sub-national model gives a row per region and year.

Usage

```
bayes_model_estimates(model)
```

Arguments

model	A model from generate_bayes_model().

Value

A tibble: indicator, adminlevel_1 (a sub-national model only), year, estimate, lower and upper (95%), lower_80 and upper_80.

Examples

```
## Not run:
# `cache` is a dataset with adjusted data; fitting needs the Bayesian model packages
model <- cache$get_bayes_model("national", "penta3")
estimates <- bayes_model_estimates(model)
estimates[estimates$year >= 2020, c("year", "estimate", "lower", "upper")]

# a sub-national model has a row per region and year
regions <- bayes_model_estimates(cache$get_bayes_model("adminlevel_1", "penta3"))
regions[regions$year == 2023, ]

## End(Not run)
```

bayesian_page_server *Bayesian page server for the page registry*

Description

The server an app lists for its Bayesian pages in its page registry. It calls bayesian_server() with an active that also starts the background workers (cd_start_bayes_workers()) each time it is read as TRUE, which happens first when the page is opened with data loaded. Starting them a second time does nothing, so the workers are started once and only for sessions that open the page.

Usage

```
bayesian_page_server(id, cache, i18n, admin_level, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`cache`	A reactive that returns the dataset, a CacheConnection.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`admin_level`	A string, not a reactive: the level the models are fitted at, `"national"` or `"adminlevel_1"` (admin level 1). The registry gives it through the page's `server_args`.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

What `bayesian_server()` returns: a reactive that holds the key of the tab shown (an indicator's name, or `"custom"`).

Examples

```
## Not run:
# the page's entry in an app's page registry; server_args gives the admin level
cd_page_def(
  id = "bayesian_national",
  ui = bayesian_ui,
  server = bayesian_page_server,
  title = "title_bayesian_analysis",
  section = "title_bayesian_analysis",
  subtitle = "sub_bayesian",
  server_args = list("national")
)

# the same page at admin level 1 is a second entry: id = "bayesian_subnational" and
# server_args = list("adminlevel_1")

## End(Not run)
```

`bayesian_ui` *Bayesian coverage model page*

Description

The Bayesian coverage model page, shared by the apps and used at national level and at admin level 1. It has one card of charts with a tab for each indicator the app lists (`cd_cfg("bayes_indicators")`, by default cd_bayes_indicators), the only ones the model takes; each tab plots the model's coverage estimates by year. A model takes minutes to fit, so the fit runs in a background worker (`cd_bayes_fit_task()`) and the chart shows its loader meanwhile. A fit starts only when its tab is shown, from the dataset's `bayes_model_inputs()`; the result is kept in the dataset (`keep_bayes_model()`) and read back with `bayes_model_cached()`, so it is fitted once for an admin level, indicator and denominator. While the model's packages are missing (`cd_bayes_packages_missing()`) the page shows a notice and fits nothing: inside DataSuite the notice has a button that asks it to install them, elsewhere it shows the R command to run.

Usage

```
bayesian_ui(id, i18n, label)

bayesian_server(id, cache, i18n, admin_level, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`label`	Not used.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A string, not a reactive: the level the models are fitted at, `"national"` or `"adminlevel_1"` (admin level 1).
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Details

At admin level 1 the page has two filters: the region (every region compared in one year, each with its model estimate and its facility-data value, or one region followed over the years as the national page follows the country) and the year the regions are compared in. The national page has none. Apps register the page with `bayesian_page_server()`, which also starts the workers.

Value

`bayesian_ui()` returns the page's UI as Shiny tags. `bayesian_server()` returns a reactive that holds the key of the tab shown: an indicator's name, or `"custom"`.

Examples

```
## Not run:
# a page of your own with the same charts: start the workers, then call the server.
# The id must be a page in the app's page registry.
```

```
my_bayes_ui <- function(id, i18n) bayesian_ui(id, i18n)
my_bayes_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  cd_start_bayes_workers()
  bayesian_server(id, cache, i18n, "adminlevel_1", active = active)
}

## End(Not run)
```

`cache_definition` *What a CacheConnection data point is*

Description

The definition of one public member of CacheConnection: what it holds or returns in the methodology's terms, its kind (`data`, `setting`, `reference`, `mapping`, `check`, `value`, `store`, `state`, `method`, `action`), its grain (what one row is) and key columns, its unit, where the user sets it in the app (`set_on`) and which pages show it (`shown_on`), the methodology docs section it comes from (`method`), the defaults it follows (ids of `cd_methodology_defaults()`), and what it is computed from (`depends_on`). Columns are described from their names by the data dictionary, not here. `status` is `"draft"` (with a `note`) where the definition is not certain. `cache_manifest()` includes every member's definition.

Usage

```
cache_definition(name = NULL)
```

Arguments

`name`	A member name, e.g. `"adjusted_data"`; `NULL` for all of them.

Value

A list (one definition), a named list of all of them, or `NULL` when `name` is not a public member.

Examples

```
cache_definition("adjusted_data")$what
```

cache_manifest *What the Countdown AI knows of CacheConnection*

Description

Every public member of `CacheConnection` – method or active binding – with its arguments (defaults, and the values an argument may take when the code says), its documentation, whether it writes to the dataset, whether a custom chart may draw from it, and its `definition` (what the data point is, see `cache_definition()`); plus the report kinds. Read from the installed package, so it always matches the code; countdown-analytics builds its guide for the AI from it.

Usage

```
cache_manifest()
```

Value

A list (JSON-able with `jsonlite::toJSON(auto_unbox = TRUE)`): `package`, `version`, `generatedAt`, `members` and `reportKinds`.

Examples

```
m <- cache_manifest()
length(m$members)
```

calculate_average_reporting_rate *Reporting rate summary by administrative level*

Description

`calculate_average_reporting_rate()` computes the average reporting rate for each indicator by year and administrative level. For `adminlevel_1`, you can limit the results to a single region.

Usage

```
calculate_average_reporting_rate(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  region = NULL
)
```

Arguments

.data	A tibble of class cd_data.
admin_level	Character. One of "national", "adminlevel_1", or "district".
region	Optional. Name of a specific region (adminlevel_1) to filter results. Only used when admin_level = "adminlevel_1".

Value

A tibble of class cd_average_reporting_rate.

Examples

```
## Not run:
calculate_average_reporting_rate(data, admin_level = "adminlevel_1")
calculate_average_reporting_rate(data, admin_level = "adminlevel_1", region = "Nairobi")

## End(Not run)
```

---

calculate_completeness_summary

Summarize data completeness by year

Description

Computes yearly percentage of non-missing values for each indicator, along with overall completeness summaries.

Usage

```
calculate_completeness_summary(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  threshold = .cd_method$data_quality$reporting_threshold,
  region = NULL
)
```

Arguments

.data	A cd_data object.
admin_level	One of 'national', 'adminlevel_1', or 'district'.
threshold	Numeric. The performance threshold (in percent) stored as an attribute on the result and used when plotting. Default is 90.
region	Optional name of an adminlevel_1 region. If supplied, only that region's rows are kept and results are shown by district. Only valid with admin_level = 'adminlevel_1'. Default is NULL.

Value

A `cd_completeness_summary` tibble:

- One row per year and group
- `mis_<indicator>` columns with % non-missing values
- `mean_mis_all` summarizing all indicators

Examples

```
## Not run:
calculate_completeness_summary(data)

## End(Not run)
```

`calculate_coverage` *Combine immunization coverage data*

Description

`calculate_coverage` integrates immunization coverage data from three sources: DHIS2 (District Health Information Software), survey-based estimates, and WUENIC (WHO-UNICEF estimates). The combined dataset is prepared for analysis at various administrative levels.

Usage

```
calculate_coverage(.data, survey_data, wuenic_data, subnational_map = NULL)
```

Arguments

`.data`	A `cd_indicator_coverage` data frame with DHIS2 coverage metrics (as returned by `calculate_indicator_coverage()`). Its `admin_level` and `region` attributes determine the level of the result.
`survey_data`	A data frame containing survey-based immunization estimates.
`wuenic_data`	A data frame containing WHO-UNICEF (WUENIC) coverage estimates.
`subnational_map`	(Optional) A data frame mapping subnational regions to parent regions, required for subnational-level analyses. Default is `NULL`.

Value

A `cd_coverage` data frame containing harmonized coverage estimates for each year from DHIS2, WUENIC, and survey data. Includes metadata for administrative levels, regions, and denominators.

Examples

```
## Not run:
calculate_coverage(precomputed_data, survey_df, wuenic_df)

## End(Not run)
```

`calculate_derived_coverage`
Generate coverage data with derived denominators

Description

Calculates trend-adjusted and subnationally-redistributed coverage estimates using the DTP1-derived denominator logic described in CD2030. It estimates coverage over time based on changes in DHIS2 population counts, while preserving subnational proportions from the base year.

Usage

```
calculate_derived_coverage(.data, indicator)
```

Arguments

`.data`	A `cd_population_metrics` object containing indicator values and DHIS2 population.
`indicator`	A character string specifying the indicator to calculate coverage for.

Details

This allows estimating:

- Trends over time in coverage
- Subnational inequities

Value

A `cd_derived_coverage` tibble with columns for old and new coverage estimates.

Examples

```
## Not run:
calculate_derived_coverage(population_metrics, "penta1")

## End(Not run)
```

calculate_district_completeness_summary

District-level completeness summary

Description

Calculates yearly % of districts with complete data for each indicator.

Usage

```
calculate_district_completeness_summary(.data, region = NULL)
```

Arguments

`.data`	A `cd_data` object.
`region`	Optional name of an `adminlevel_1` region. If supplied, only districts in that region are included. Default is `NULL` (all districts).

Value

A `cd_missing_district` tibble:

- One row per year
- `% non-missing` per indicator
- `mean_mis_all`, `mean_mis_four` summaries

Examples

```
## Not run:
calculate_district_completeness_summary(data)

## End(Not run)
```

calculate_district_outlier_summary

District-level outlier summary by year

Description

Calculates the yearly percentage of districts without extreme outliers for each health indicator. Flags are based on the Hampel X84 method (beyond 5 MAD from median).

Usage

```
calculate_district_outlier_summary(.data, region = NULL)
```

Arguments

`.data`	A `cd_data` object with outlier flags (`_outlier5std` columns).
`region`	Optional. Filter results to a specific `adminlevel_1`.

Details

- Aggregates by `district` and `year`, using the max flag per group.
- Computes percent of non-outliers per indicator.
- Returns yearly summaries including:
  - Per-indicator non-outlier rates
  - `mean_out_all`: all indicators
  - `mean_out_four`: excludes IPD

All results are rounded to two decimals.

Value

A `cd_district_outliers_summary` tibble.

Examples

```
## Not run:
calculate_district_outlier_summary(data)

## End(Not run)
```

`calculate_district_ratios_summary` *Calculate district adequacy summary*

Description

Calculates and summarizes the adequacy of specified indicator ratios by year, evaluating the consistency of key health indicators across districts. It computes the percentage of districts meeting the adequacy criteria for each indicator ratio, based on the specified range.

Usage

```
calculate_district_ratios_summary(.data)
```

Arguments

`.data`	A data frame containing indicator data by district and year. The data frame should include all indicators specified in `ratio_pairs`.

Details

- The function dynamically calculates ratios between the specified indicator pairs.
- For each year, it computes the percentage of districts where the calculated ratio for each indicator pair falls within the adequate range.
- The summary table aids in monitoring indicator consistency across districts and over time, providing insights into areas where health service delivery may vary.

Value

A tibble of class `cd_district_ratios_summary`, containing the summary of adequacy checks by year. Each column represents the percentage of districts meeting the adequacy criteria for the specified ratio, with overall metrics included for data interpretation and quality monitoring.

Examples

```
## Not run:
# Calculate adequacy summary for ANC1/Penta1 and other indicator ratios
calculate_district_ratios_summary(cd_data)

## End(Not run)
```

`calculate_district_reporting_rate` *District-level reporting rates by year*

Description

`calculate_district_reporting_rate` Calculates the percentage of districts that meet or exceed a specified reporting rate threshold for each indicator, by year. Also computes a yearly average across all indicators.

Usage

```
calculate_district_reporting_rate(
  .data,
  threshold = .cd_method$data_quality$reporting_threshold,
  region = NULL
)
```

Arguments

`.data`	A tibble of class `cd_data`.
`threshold`	Numeric. The performance threshold: the minimum reporting rate (in percent) for a district to be considered compliant. Default is 90.
`region`	Optional. Filter results for a specific region (admin level 1 name).

Value

A tibble of class `cd_district_reporting_rate`.

Examples

```
## Not run:
calculate_district_reporting_rate(data, threshold = 90)
calculate_district_reporting_rate(data, threshold = 85, region = "Eastern")

## End(Not run)
```

`calculate_health_system_comparison` *Compare health system metrics across levels*

Description

Returns a merged dataset of health coverage and system metrics at district and admin level 1 for the latest year.

Usage

```
calculate_health_system_comparison(
  .data,
  admin1_coverage_data,
  admin2_coverage_data
)
```

Arguments

`.data`	A `cd_data` object containing raw health system inputs including year, population, and facility indicators.
`admin1_coverage_data`	A coverage data frame at admin level 1 (with `year`, `adminlevel_1` and `cov_*` columns), such as the output of `calculate_indicator_coverage()` for `admin_level = "adminlevel_1"`. Its latest year defines the year kept in the result.
`admin2_coverage_data`	A coverage data frame at district level (with `year`, `adminlevel_1`, `district` and `cov_*` columns), such as the output of `calculate_indicator_coverage()` for `admin_level = "district"`.

Value

A data frame with admin level 1 and district metrics joined side by side for comparison.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

comparison <- calculate_health_system_comparison(
  cache$adjusted_data,
  admin1_coverage_data = cache$indicator_coverage_admin1,
  admin2_coverage_data = cache$indicator_coverage_district
)
comparison[c("adminlevel_1", "district", "ad1_ratio_fac_pop", "dis_ratio_fac_pop")]
```

---

calculate_health_system_metrics

Calculate health system metrics

Description

Aggregates data at national, admin level 1, or district level and computes ratios and performance scores.

Usage

```
calculate_health_system_metrics(
  .data,
  admin_level = c("national", "adminlevel_1", "district")
)
```

Arguments

`.data`	A data frame with health system indicators and population data.
`admin_level`	One of 'national', 'adminlevel_1', or 'district'.

Value

A summarized data frame with calculated metrics: ratios (facility, beds, workforce, service use) and performance scores.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

national <- calculate_health_system_metrics(data, admin_level = "national")
national[c("year", "ratio_fac_pop", "ratio_bed_pop", "ratio_hstaff_pop", "score_total")]

regions <- calculate_health_system_metrics(data, admin_level = "adminlevel_1")
```

```
regions[c("adminlevel_1", "ratio_fac_pop", "ratio_hstaff_pop", "score_total")]
```

`calculate_indicator_coverage`
Calculate health coverage indicators

Description

`calculate_indicator_coverage` computes key health coverage indicators across specified administrative levels (national, adminlevel_1, and district). The function integrates data from multiple sources, including DHIS-2, UN estimates, ANC-1, and Penta-1 survey data. It calculates coverage rates for a variety of vaccinations and health metrics based on projected, survey-derived, and estimated denominators.

Usage

```
calculate_indicator_coverage(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  derivation_population = c("totbirths_dhis2", "totlivebirths_dhis2", "totunder1_dhis2",
    "totpop_dhis2", "un_population", "un_births", "un_under1"),
  un_estimates = NULL,
  survey_estimates = NULL,
  subnational_map = NULL,
  anc1survey = .cd_method$denominators$anc1survey,
  dpt1survey = .cd_method$denominators$dpt1survey,
  survey_year = .cd_method$denominators$survey_year,
  region = NULL,
  show_district = TRUE,
  sbr = .cd_method$denominators$sbr,
  nmr = .cd_method$denominators$nmr,
  pnmr = .cd_method$denominators$pnmr,
  twin = .cd_method$denominators$twin,
  preg_loss = .cd_method$denominators$preg_loss
)
```

Arguments

`.data`	A `cd_data` tibble containing DHIS-2, UN, ANC-1, and Penta-1 data. This dataset must include columns for key population and vaccination metrics.
`admin_level`	Character. Specifies the administrative level for calculations. Options include:`"national", "adminlevel_1"`, and `"district"`.
`derivation_population`	Character. The population column used as the base for the DTP1/ANC1-derived denominators (`*derived` columns). One of `"totbirths_dhis2"`, `"totlivebirths_dhis2"`, `"totunder1_dhis2"`, `"totpop_dhis2"`, `"un_population"`, `"un_births"` or `"un_under1"`. Defaults to the first.

`un_estimates` Optional. A tibble containing UN population estimates. Required for national-level calculations.

`survey_estimates` Optional. A data frame of subnational survey estimates (`r_*` columns such as `r_anc1` and `r_penta1`, and optionally mortality rates). Used for subnational levels to replace the default rates below where survey values are available. Default is `NULL`.

`subnational_map` Optional. A data frame mapping survey regions to `adminlevel_1` units, used when joining `survey_estimates`. Default is `NULL`.

`anc1survey` Numeric. Survey-derived coverage rate for ANC-1 (antenatal care, first visit). Default is `0.98`.

`dpt1survey` Numeric. Survey-derived coverage rate for Penta-1 (DPT1 vaccination). Default is `0.97`.

`survey_year` Interger. The year of Penta-1 survey provided

`region` Optional name of an `adminlevel_1` region to restrict the calculation to. Only valid with `admin_level = "adminlevel_1"`. Default is `NULL`.

`show_district` Logical. When `region` is supplied, whether to return results by district within that region (`TRUE`, the default) or aggregated for the region.

`sbr` Numeric. The stillbirth rate. Default is `0.02`.

`nmr` Numeric. Neonatal mortality rate. Default is `0.025`.

`pnmr` Numeric. Post-neonatal mortality rate. Default is `0.024`.

`twin` Numeric. Twin birth rate. Default is `0.015`.

`preg_loss` Numeric. Pregnancy loss rate. Default is `0.03`.

Value

A tibble of class `cd_indicator_coverage` containing calculated coverage indicators for the specified administrative level.

Examples

```
## Not run:
# Calculate coverage indicators at the national level
coverage_data <- calculate_indicator_coverage(
  .data = dhis2_data,
  admin_level = "national",
  un_estimates = un_data
)

# Calculate coverage indicators at the district level
coverage_data <- calculate_indicator_coverage(
  .data = dhis2_data,
  admin_level = "district"
)

## End(Not run)
```

`calculate_inequality` *Analyze subnational health coverage data with inequality metrics*

Description

`calculate_inequality` computes subnational health coverage metrics and evaluates disparities compared to reference averages (either national or regional).

Usage

```
calculate_inequality(subnational_data, reference_data)
```

Arguments

`subnational_data`
: A data frame containing pre-calculated subnational health coverage data.

`reference_data`
: A data frame containing pre-calculated reference health coverage data (e.g., national data if analyzing `adminlevel_1`, or `adminlevel_1` data if analyzing `district`).

Value

A tibble (`cd_inequality` object) containing:

- Subnational health coverage metrics.
- Population shares.
- MADM, MRDM, and related disparity metrics.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

# regions against the national coverage
inequality <- calculate_inequality(
  subnational_data = cache$indicator_coverage_admin1,
  reference_data = cache$indicator_coverage_national
)
select_inequality(inequality, indicator = "instlivebirths", denominator = "dhis2")
```

`calculate_outliers_summary` *Annual summary of outlier-free reporting rates*

Description

`calculate_outliers_summary` computes yearly percentages of non-outlier values for each health indicator, based on precomputed 5-MAD outlier flags. Summarizes data quality by administrative level and year.

Usage

```
calculate_outliers_summary(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  region = NULL
)
```

Arguments

`.data`	A `cd_data` object with `_outlier5std` flag columns (0 = valid, 1 = outlier).
`admin_level`	Character. Aggregation level. One of: • `"national"` • `"adminlevel_1"` • `"district"`
`region`	Optional. Restrict to a specific region (only when `admin_level = "adminlevel_1"`).

Details

Outliers are defined using a Hampel filter with a 5-MAD threshold.

Aggregates the outlier flags by year and administrative level and computes the share of valid (non-outlier) values per indicator. The IPD indicators are left out. Adds `mean_out_all`, the average non-outlier rate across the indicators summarised.

Values are expressed as percentages (0-100).

Value

A tibble of class `cd_outlier`.

Examples

```
## Not run:
calculate_outliers_summary(data, admin_level = "district")

## End(Not run)
```

`calculate_overall_score` *Calculate overall quality score for data quality metrics*

Description

Calculates an overall quality score based on various data quality metrics. It summarizes completeness, outlier presence, and consistency of reporting in immunization health facility data.

Usage

```
calculate_overall_score(
  .data,
  threshold,
  ratio_pairs = NULL,
  region = NULL,
  labels = NULL
)
```

Arguments

`.data`	A data frame of type `cd_data` containing facility data including annual reporting rates, completeness, and consistency indicators.
`threshold`	Numeric. The performance threshold (in percent): the reporting rate and completeness a district is expected to reach. Required: it has no default.
`ratio_pairs`	Named list of indicator pairs whose ratios are scored (the default pairs when `NULL`).
`region`	Optional name of an `adminlevel_1` region. If supplied, all metrics are calculated for that region only. Default is `NULL`.
`labels`	Optional named list overriding the default row labels. May contain `header` (`h1`, `h2`, `h3`), `section` (`r1a`, `r1b`, `r1c`, `r2a`, `r2b`, `score`) and `metric` (for example `r_anc1_penta1`, `ok_anc1_penta1`) sub-lists; only the supplied entries are replaced. Default is `NULL`.

Details

`calculate_overall_score` processes multiple data quality indicators:

- **Completeness metrics**: Percentage of expected reports, districts with complete reporting, and districts with no missing values.
- **Outlier metrics**: Percentage of monthly values and districts without extreme outliers.
- **Consistency ratios**: Ratios between different immunization indicators to ensure internal consistency.

The function calculates averages for the selected metrics and includes a row summarizing the annual data quality score.

Value

A tibble with calculated scores for each metric, including a summary row for the annual quality score. The result is ordered by metric codes and ready for reporting in tabular or graphical form.

Examples

```
## Not run:
calculate_overall_score(.data = my_data)

## End(Not run)
```

`calculate_overall_score1` *Calculate overall quality score from summaries*

Description

Builds the same score table as `calculate_overall_score()`, but from summaries that have already been calculated.

Usage

```
calculate_overall_score1(
  average_reporting_rate,
  district_reporting_rate,
  district_completeness,
  outliers_summary,
  district_outliers_summary,
  ratios_summary,
  labels = NULL,
  threshold = .cd_method$data_quality$reporting_threshold
)
```

Arguments

`average_reporting_rate`
: A `cd_average_reporting_rate` object with a `mean_rr` column by year.

`district_reporting_rate`
: A `cd_district_reporting_rate` object.

`district_completeness`
: A `cd_missing_district` object, as returned by `calculate_district_completeness_summary(`

`outliers_summary`
: A `cd_outlier` object with a `mean_out_all` column by year.

`district_outliers_summary`
: A `cd_district_outliers_summary` object.

`ratios_summary`
: A `cd_ratios_and_adequacy` object.

`labels`
: Optional named list overriding the default row labels, with `header`, `section` and `metric` sub-lists (see `calculate_overall_score()`). Default is `NULL`.

`threshold`
: Numeric. The performance threshold (in percent) shown in the label of row 1b. Default is `90`.

Value

A tibble with calculated scores for each metric, including a summary row for the annual quality score.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
calculate_overall_score1(
  average_reporting_rate = calculate_average_reporting_rate(data),
  district_reporting_rate = calculate_district_reporting_rate(data, threshold = 90),
  district_completeness = calculate_district_completeness_summary(data),
  outliers_summary = calculate_outliers_summary(data),
  district_outliers_summary = calculate_district_outlier_summary(data),
  ratios_summary = calculate_ratios_and_adequacy(data),
  threshold = 90
)
```

---

calculate_ratios_and_adequacy

Calculate indicator ratios and adequacy flags

Description

Computes ratios for specified pairs of indicators dynamically and flags whether each calculated ratio falls within an adequate range, per district and year. This function is an intermediate utility used to standardize ratio calculations and adequacy checks across indicator pairs.

Usage

```
calculate_ratios_and_adequacy(
  .data,
  ratio_pairs = NULL,
  adequate_range = .cd_method$data_quality$ratio_adequate_range,
  region = NULL
)
```

Arguments

`.data`	A data frame of type `cd_data`.
`ratio_pairs`	A named list of indicator pairs for which ratios should be calculated. • Each pair should be a character vector of length two, with the first element as the numerator and the second as the denominator. Default is: `list("ratioAP" = c("anc1", "penta1"), "ratioPP" = c("penta1", "penta3"), "ratioOO" = c("opv1", "opv3"))`. • Each name in the list becomes the name of the resulting ratio column.
`adequate_range`	A numeric vector of length two specifying the inclusive lower and upper bounds for the adequate range, typically between `1` and `1.5`. Ratios within this range are flagged as adequate.
`region`	Optional. Restrict analysis to one region (`adminlevel_1`).

Details

This function standardizes ratio calculations by dynamically computing ratios and adequacy flags for specified pairs of indicators. It operates in three main stages:

1. **Summing Indicators**: Calculates the sum of each indicator by `district` and `year`.
2. **Calculating Ratios**: Divides the summed numerator by the summed denominator for each pair in `ratio_pairs`, creating a column for each ratio.
3. **Checking Adequacy**: Flags each ratio as adequate if it falls within the specified `adequate_range`, returning `1` for adequate and `0` otherwise.

Value

A data frame with columns for:

- **Summed indicators by district and year**: Summed values for each indicator across districts within the specified `year`, based on input in `ratio_pairs`.
- **Ratios**: Each ratio is calculated as the sum of the specified numerator divided by the sum of the specified denominator for each pair.
- **Adequacy flags**: For each ratio, an adequacy flag (e.g., `adeq_ratioAP`) is generated, set to `1` if the ratio falls within the `adequate_range`, and `0` otherwise.
- The output is aggregated by year and includes means of ratios and adequacy flags.

See Also

`calculate_ratios_summary()` and `calculate_district_ratios_summary()`, which summarise the result.

Examples

```
## Not run:
# Calculate ratios and adequacy checks using custom ranges
calculate_ratios_and_adequacy(data,
  ratio_pairs = list(
    "ratioAP" = c("anc1", "penta1"),
    "ratioPP" = c("penta1", "penta3")
  ),
  adequate_range = c(1, 1.5)
)

## End(Not run)
```

---

calculate_ratios_summary

Calculate yearly indicator ratios summary with expected ratios

---

Description

Computes yearly summaries for specified indicator ratios, assessing trends across years and providing both actual and expected ratios for each indicator. It evaluates specified indicator pairs to calculate ratios and compares actual values with expected ratios, based on predefined survey coverage values. The summary facilitates monitoring consistency of key health indicators over time.

Usage

```
calculate_ratios_summary(
  .data,
  survey_coverage = .cd_method$data_quality$ratio_expected_coverage,
  anc1_penta1_mortality = .cd_method$data_quality$ratio_anc1_penta1_mortality
)
```

Arguments

`.data`	A data frame containing yearly data for specified indicators. Expected columns include `district`, `year`, and additional numeric columns for health indicators, such as `anc1` and `penta1`.
`survey_coverage`	A named vector of assumed coverage rates for each indicator. Default is `c(anc1 = 0.98, penta1 = 0.97, penta3 = 0.89, opv1 = 0.97, opv3 = 0.78, pcv1 = 0.97, rota1 = 0.96)`.
`anc1_penta1_mortality`	A numeric multiplier applied specifically to the `"ratioAP"` ratio of `anc1` to `penta1`, to account for assumed mortality between `anc1` and `penta1`. Default is `1.07`.

Details

The function provides insights into indicator relationships over time and evaluates deviations from expected values. It enables data quality checks and trend analysis by summarizing ratios yearly and including expected values for comparison. The `"Expected Ratio"` row incorporates predefined survey coverage and serves as a reference for assessing the adequacy of each indicator's ratio.

Value

A `cd_ratios_summary` object, a data frame containing:

- `year`: Year of each calculated ratio, with an additional row `"Expected Ratio"` to represent expected values for each indicator ratio based on `survey_coverage`.
- **Indicator Ratios**: Columns for each ratio, showing actual yearly values and an expected value row based on `survey_coverage`.
- **Additional Columns**: Health indicator values for each year.

Examples

```
## Not run:
# Basic usage with default parameters
calculate_ratios_summary(cd_data)

# Custom survey coverage and mortality adjustments for "ratioAP"
calculate_ratios_summary(cd_data,
  survey_coverage = c(anc1 = 0.95, penta1 = 0.92, penta3 = 0.85),
  anc1_penta1_mortality = 1.05
)

## End(Not run)
```

`calculate_ratios_ui` *Indicator ratios card*

Description

A card with one chart of the ratios between related indicators (ANC1 to Penta1, OPV1 to OPV3 and Penta1 to Penta3 have translated labels). It is the first card of the Internal consistency page (`internal_consistency_ui()`) and has no filters. It reads `ratios_summary` from the dataset.

Usage

```
calculate_ratios_ui(id, i18n)

calculate_ratios_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The card does no work before that.

Value

`calculate_ratios_ui()` returns the card's UI as Shiny tags. `calculate_ratios_server()` is called for its side effects.

Examples

```
## Not run:
# the card in a page module, as the Internal consistency page uses it
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, calculate_ratios_ui(ns("ratios"), i18n))
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    calculate_ratios_server("ratios", cache, i18n, active = active)
  })
}

## End(Not run)
```

`calculate_threshold` *Calculate coverage/dropout threshold attainment*

Description

Evaluates administrative regions to determine the percentage that meet specific coverage targets or fall below specific dropout thresholds for a given year.

Usage

```
calculate_threshold(
  .data,
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived"),
  indicator = c("anc4", "instlivebirths", "vaccine", "dropout")
)
```

Arguments

`.data`	A tibble of class `cd_indicator_coverage`.
`denominator`	Character. The denominator used (e.g., `'dhis2'`, `'anc1'`).
`indicator`	Character. The health indicator group to evaluate (`'anc4'`, `'instlivebirths'`, `'vaccine'`, `'dropout'`).

Value

A `cd_threshold` object summarizing the percentage of regions meeting the criteria by year.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
coverage <- cache$indicator_coverage_admin1

# the percentage of regions that reach the target for institutional live births, by year
calculate_threshold(coverage, denominator = "dhis2", indicator = "instlivebirths")

# for the three tracer vaccines, with the penta1 denominator
calculate_threshold(coverage, denominator = "penta1", indicator = "vaccine")
```

`cd_admin_level_choices` *Choices of the admin level chip*

Description

The admin levels a page can be shown at, in the form a filter chip takes them. `cd_admin_level_ui()` builds its level chip from this, and a page can use it to offer the same levels in a control of its own.

Usage

```
cd_admin_level_choices(include_national = FALSE)
```

Arguments

`include_national`	Logical. `TRUE` puts the national level first, before admin level 1 and district. Default `FALSE`.

Value

A named character vector. The values are the levels (`"national"` when asked for, `"adminlevel_1"`, `"district"`) and the names are the translation keys of their labels.

Examples

```
cd_admin_level_choices()
cd_admin_level_choices(include_national = TRUE)
```

---

cd_admin_level_ui *Admin level and region filter chips*

Description

Two filter chips for a page's filter bar: one chooses the admin level (admin level 1 or district, and national when asked for), the other a single area at that level. A page puts `cd_admin_level_ui()` in its filter bar, calls `cd_admin_level_server()` in its server, and reads the chosen level and area from the reactive the server returns. `cd_admin_parts()` splits that reactive into one reactive for the level and one for the area.

Usage

```
cd_admin_level_ui(id, i18n, include_national = FALSE, show_admin_level = TRUE)

cd_admin_level_server(
  id,
  cache,
  i18n,
  allow_select_all = FALSE,
  show_district = TRUE,
  show_region = TRUE,
  show_admin_level = TRUE,
  selected_admin1 = reactive(NULL)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`include_national`	Logical. `TRUE` adds the national level to the level chip, as its first choice and so the one it starts on. Default `FALSE`.
`show_admin_level`	Logical. `FALSE` leaves the level chip out, for a page that only lets the reader choose an area. Default `TRUE`.
`cache`	A reactive that returns the dataset, a CacheConnection.
`allow_select_all`	Logical. `TRUE` adds an "all" option to the region chip and starts the chip on it. Default `FALSE`.

`show_district` Logical. Whether an area can be chosen at the district level. With a level chip, `FALSE` hides the region chip while the level is district. Without a level chip, it sets the fixed level (see Details). Default `TRUE`.

`show_region` Logical. `FALSE` never shows the region chip, so `region` is always `NULL`. Default `TRUE`.

`selected_admin1`
A reactive that returns the name of an admin level 1 area, or `NULL`. When it returns a name and the level is district, the region chip lists only the districts of that area. Default `reactive(NULL)`, every district.

Details

The region chip is drawn by the server, so it appears only where there are areas to choose from: never at the national level, never when `show_region` is `FALSE`, and at the district level only when `show_district` is `TRUE`. Its options are the areas of `cache()$subnational_regions` at the chosen level, and districts are listed under the admin level 1 area they belong to. Changing the level draws a new region chip, so an area of the old level is never kept. The list of areas is debounced by 300 milliseconds.

With `allow_select_all = TRUE` the region chip starts on an "all" option, which the server reports as a `NULL` region. Without it the chip starts on the first area in the list.

When `show_admin_level` is `FALSE` there is no level chip, and the level is fixed: `"district"` when `show_district` is `TRUE` (the default), `"adminlevel_1"` otherwise. Give `show_admin_level` the same value in the ui and the server call.

Value

`cd_admin_level_ui()` returns the UI as Shiny tags: the level chip (when shown) and the place where the server draws the region chip.

`cd_admin_level_server()` returns a reactive holding a list with two elements:

- `admin_level`: `"national"`, `"adminlevel_1"` or `"district"`.
- `region`: the name of the chosen area, or `NULL` when the region chip is not shown or "all" is chosen.

While the region chip is shown but has not yet reported a value, the reactive stops quietly (as `shiny::req()` does), so what depends on it waits.

Examples

```
## Not run:
# in a page's ui
cd_filter_bar(cd_admin_level_ui(ns("admin_level"), i18n))

# in the page's server
admin <- cd_admin_level_server("admin_level", cache, i18n, allow_select_all = TRUE)
observe({
  admin()$admin_level
  admin()$region
})
```

```
## End(Not run)
```

`cd_admin_parts` *Split the admin level filter's value into two reactives*

Description

`cd_admin_level_server()` returns one reactive holding both the admin level and the region. Pages usually hand the two to their charts and tables as separate reactives, and this gives one reactive for each.

Usage

```
cd_admin_parts(admin)
```

Arguments

`admin`	The reactive `cd_admin_level_server()` returned.

Value

A list of two reactives:

- `admin_level`: the chosen admin level.
- `region`: the chosen area's name, or `NULL` when the region chip is not shown or "all" is chosen.

Both wait (as `shiny::req()` does) until `admin()` has a value.

Examples

```
## Not run:
admin <- cd_admin_level_server("admin_level", cache, i18n)
parts <- cd_admin_parts(admin)
parts$admin_level()
parts$region()

## End(Not run)
```

`cd_app` *Build a Countdown app*

Description

Puts a whole Countdown app together and returns it as a Shiny app: the frame every app shares (header, sidebar, language picker), the Introduction and Load Data screens, and the app's own pages. An app's `run_app()` sets its options, builds its translator, lists its pages and its nav tree, and then calls `cd_app()`. See `cd2030.rmncah::run_app()` for a complete one.

Usage

```
cd_app(
  app_name,
  app_version,
  theme,
  nav_sections,
  registry,
  i18n,
  language,
  selected_file,
  upload_ui = get0("upload_data_ui", envir = parent.frame()) %||% cd_upload_data_ui,
  upload_server = get0("upload_data_server", envir = parent.frame()) %||%
    cd_upload_data_server
)
```

Arguments

`app_name`	The app's name, shown in the header.
`app_version`	The app's version, shown in the header.
`theme`	The app's colour theme: `"rmncah"` or `NULL` (maroon), `"vaccine"` (blue) or `"pooled"` (green).
`nav_sections`	The sidebar's nav tree: a list of nav sections, made with `cd_nav_start()` and the functions beside it, and with the kit's `cd_nav_section()` and `cd_nav_item()`.
`registry`	The app's pages: a list of page definitions made with the kit's `cd_page_def()`, the same list the app gives to `cd_use_pages()`.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`language`	The language the app starts in, for example `"en"`, `"fr"` or `"pt"`.
`selected_file`	The path of a dataset to open when the app starts, or `NA` for none. DataSuite gives it when the app is started on a file.
`upload_ui`	The ui function of the Load Data screen, called as `upload_ui("upload_data", i18n = i18n, is_electron = )`, where `is_electron` is `TRUE` when `selected_file` is given. By default it is the function named `upload_data_ui` where

`cd_app()` is called from, when there is one, and otherwise the shared screen, `cd_upload_data_ui()`, which the app sets up with `options(cd2030.wizard = )`.

`upload_server` The server function of the Load Data screen, called as `upload_server("upload_data", i18n, selected_file, active = )`. It must return a list with `cache` (a reactive that returns the dataset, a CacheConnection, or `NULL` before one is loaded) and `requires_walkthrough` (a reactive that is `TRUE` for a new upload and `FALSE` for a dataset opened again). By default it is the function named `upload_data_server` where `cd_app()` is called from, when there is one, and otherwise `cd_upload_data_server()`.

Details

`cd_app()` calls the kit's `datasuite.ui::app_frame()` and adds what is the same in every Countdown app:

- The Introduction screen and the Load Data screen. The app opens on Load Data, and these two screens are never locked.
- The dataset. It is the reactive the Load Data screen returns, and every page receives it as `cache`. The pages unlock once the dataset holds Countdown data. The nav items marked `requires_adjustment = TRUE` unlock once the data is also adjusted.
- The row naming the denominators in use, in every page header (`cd_denominator_header()`).
- The language. A dataset that has just been uploaded takes the language on screen, and a dataset that is opened again shows the language it was saved with.
- The reports' colours, which follow `theme`.
- What DataSuite's chat can read and do in the app: the filters of the page on show, the dataset, and actions that set filters and save, read, change and write reports.

Before calling it, the app registers its translator with `cd_use_i18n()` and its pages with `cd_use_pages()`.

Value

A Shiny app object. Printing it, or passing it to `shiny::runApp()`, runs the app.

Examples

```
## Not run:
i18n <- shiny.i18n::init_i18n(translation_json_path = cd_translations("translation.json"))
cd_use_i18n(i18n)

pages <- my_pages()
cd_use_pages(pages)

cd_app(
  app_name = "My Countdown app", app_version = "1.0.0", theme = "vaccine",
  nav_sections = list(cd_nav_start(), cd_nav_quality(), cd_nav_denominators()),
  registry = pages, i18n = i18n, language = "en", selected_file = NA
)
```

```
## End(Not run)
```

`cd_bayes_fit_task` *Fit one Bayesian coverage model in a background worker*

Description

Hands one fit to a `future` worker and returns at once. The worker runs `generate_bayes_model()` on the coverage data, overall score, indicator and denominator in `inputs`, with `cd2030.core` attached and a parallel-safe random seed. The Bayesian page uses this function as the task of its `shiny::ExtendedTask`. With the default `sequential` plan the fit runs in the calling process; see `cd_start_bayes_workers()`.

Usage

```
cd_bayes_fit_task(inputs)
```

Arguments

`inputs`	A list with `key`, `coverage_data`, `overall_score`, `indicator` and `denominator`, as the dataset's `bayes_model_inputs()` method returns (CacheConnection).

Value

A promise of a list with two elements: `key`, which is `inputs$key` unchanged, and `model`, the fitted model. The key lets the caller keep the model for the request it was fitted for.

Examples

```
## Not run:
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

# one fit, kept in the dataset when it ends (it takes minutes)
inputs <- cache$bayes_model_inputs("national", "penta3")
promises::then(cd_bayes_fit_task(inputs), function(out) cache$keep_bayes_model(out$key, out$model))

# in a Shiny server: the task of an ExtendedTask
fit <- ExtendedTask$new(cd_bayes_fit_task)
fit$invoke(inputs)

## End(Not run)
```

cd_bayes_indicators *Indicators the Bayesian coverage model can be fitted for*

Description

The indicators the Bayesian coverage model has a survey series for. They are the default tabs of the Bayesian page (`bayesian_ui()`). An app shows only the ones its data has by setting `bayes_indicators` in `options(cd2030.config = )`, which the page reads with `cd_cfg("bayes_indicators")` (the vaccine app: penta3 and measles1).

Usage

```
cd_bayes_indicators
```

Format

A character vector of five indicator names: `"anc4"`, `"anc_1trimester"`, `"ideliv"`, `"measles1"` and `"penta3"`.

Examples

```
cd_bayes_indicators

# the tabs an app shows: its own setting, or these
cd_cfg("bayes_indicators", cd_bayes_indicators)
```

cd_bayes_install_command *The R command that installs the Bayesian model's packages*

Description

For R outside DataSuite.

Usage

```
cd_bayes_install_command(packages = cd_bayes_packages_missing())
```

Arguments

`packages`	The packages to install, by default the missing ones.

Value

The command, as a string.

Examples

```
cd_bayes_install_command(c("bayescoveragemodel", "bayescoveragedeploy"))

# only what is missing
missing <- cd_bayes_packages_missing()
if (length(missing) > 0) cd_bayes_install_command(missing)
```

---

cd_bayes_packages_missing

The Bayesian model's packages that are not installed

Description

Checks without loading them (loading brings in Stan).

Usage

```
cd_bayes_packages_missing()
```

Value

A character vector: the packages `generate_bayes_model()` needs and that are not installed, empty when the model can run.

Examples

```
cd_bayes_packages_missing()

if (length(cd_bayes_packages_missing()) == 0) {
  message("The Bayesian model can run.")
}
```

---

cd_cfg

Read one of the app's settings

Description

The shared pages are the same in every Countdown app, and what differs between apps is kept in one list of settings. An app sets the list once, in its `run_app()`, with `options(cd2030.config = list(...))`, and the pages read the setting they need with `cd_cfg()`. This way an app (RMNCAH, vaccine, or one for a custom indicator group) says what is particular to it, and no shared page depends on a group's name.

Usage

```
cd_cfg(key, default = NULL)
```

Arguments

`key`	The setting's name, a string.
`default`	What to return when the app has not set `key`, or has set it to `NULL`. Default `NULL`.

Details

A setting may be a value or a function with no arguments that returns one. A function is called each time the setting is read, so its answer can follow something that is only known later, such as the indicator group.

The settings the shared pages read (each page gives its own `default`):

- `nat_cov_indicators`: the tabs of the national and sub-national coverage pages. It may include `"fpet"`, which adds the family planning chart.
- `bayes_indicators`: the tabs of the Bayesian coverage model pages.
- `target_indicators`: the tabs of the coverage target pages.
- `equity_indicators`: the tabs of the Equity Assessment page. `equity_custom_exclude`: the indicators its Custom tab leaves out.
- `cov_trend_indicators`, `sub_derived_indicators`, `survey_comp_indicators`: the tabs of the three cards of the Denominator Selection page.
- `adjustment_indicators`: the tabs of the Data Adjustment Changes page.
- `k_factors`: the adjustment factors, as `list(name = list(id, label))`.
- `reporting_rate_indicators`, `reporting_rate_facet_ncol`: the Reporting Rate page's service chips and the number of columns of its national chart.
- `consistency_pairs`: a list of `c(x, y)` pairs of indicators, one tab of the Consistency Checks card each.
- `has_maternal`: whether the indicator group has a maternal denominator (see `cd_has_maternal()`).

Value

The setting's value. When the setting is a function, the value that function returns. When the setting is not set, `default`.

Examples

```
old <- options(cd2030.config = list(
  target_indicators = c("penta3", "measles1"),
  equity_indicators = function() c("anc4", "penta3")
))

cd_cfg("target_indicators")
cd_cfg("equity_indicators")
cd_cfg("consistency_pairs", default = list())

options(old)
```

cd_chart_catalog *The charts that have saved options*

Description

A table of every chart with options saved in a dataset's cache: which chart (its `cd_chart_id()`, or for the screen the chart's place in the app), whether they are for the screen or for reports, how many options are set, and the group it belongs to (the first two parts of its id, e.g. `coverage_filtered.national`). Use it to see which chart ids a report will pick options up for.

Usage

```
cd_chart_catalog(cache)
```

Arguments

`cache`	A CacheConnection.

Value

A tibble with `id`, `target` (`"screen"`, `"report"` or `"dataset"`), `group`, `n_options` and `options` (a one-line summary).

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_chart_options("default", cd_chart_options(legend_position = "bottom"))
cache$set_chart_options(
  "report/coverage_filtered.national.penta3",
  cd_chart_options(title = "Penta3 coverage")
)

cd_chart_catalog(cache)
```

cd_chart_id *The id of a chart*

Description

A stable, readable name for the chart a plot draws, made from the data it is drawn from: the kind of data, the admin level and the indicator (`"coverage_filtered.national.anc4"`). The app's customize panel and the reports build it the same way, so options saved for a chart in the app (`"report/<id>"`) reach the same chart in a generated report. Ids are grouped by their parts: everything under `coverage_filtered.national` is the national coverage charts.

Usage

```
cd_chart_id(x)
```

Arguments

x	The data a plot method was given.

Value

A string, or NULL when the data does not say what it is.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cd_chart_id(calculate_average_reporting_rate(data))

# plain data does not say what it is
cd_chart_id(mtcars)
```

---

cd_chartable_members *The CacheConnection members a custom chart may draw from*

Description

Members that return a table: the precomputed results (active bindings such as reporting_rate_district) and the methods that compute one (calculate_coverage(), get_filtered_coverage(), decompose_change()...). Raw data (countdown_data, adjusted_data) and settings are left out.

Usage

```
cd_chartable_members()
```

Value

A character vector of member names.

Examples

```
head(cd_chartable_members())
```

`cd_coverage_plot_ui` *Chart with its tools and downloads*

Description

One chart with the tools every Countdown chart has: a panel to customize how it looks, a download of the chart as a picture, and a download of its data as an Excel workbook. The page gives the data and says how to draw it, and this module does the rest. It is the chart that goes in the tabs of `cd_tabbed_charts_ui()`, and it works as well on its own in a card.

Usage

```
cd_coverage_plot_ui(id, toolbar_inline = FALSE)

cd_coverage_plot_toolbar_ui(id)

cd_coverage_plot_server(
  id,
  filename,
  data_fn,
  ...,
  sheet_name,
  i18n,
  plot_fun = NULL,
  about = NULL
)
```

Arguments

`id`	The module's id.
`toolbar_inline`	Logical. `FALSE` (the default) draws the tools over the chart. `TRUE` leaves them out, for a page that places them with `cd_coverage_plot_toolbar_ui()`.
`filename`	A reactive that returns the first part of the downloaded files' name. The date and time and the extension are added to it.
`data_fn`	A reactive that returns the data the chart draws. It is also what the Excel download holds.
`...`	Arguments for `plot()`, used only when `plot_fun` is `NULL`.
`sheet_name`	A reactive that returns the name of the sheet in the Excel download. It comes after `...`, so it must be given by name, as must `i18n`, `plot_fun` and `about`.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`plot_fun`	A function `function(d)` that takes the data and draws the chart or returns a ggplot. `NULL` (the default) draws `plot(d, ...)`.
`about`	What the chart is, for the app's AI: `NULL` (the default), a list, or a function that returns a list. See `datasuite.ui::ai_register_component()`.

Details

The chart is drawn with `plot_fun(data_fn())` once `data_fn()` has a value. When `plot_fun` is `NULL` it is drawn with `plot(data_fn(), ...)`, so the data must then have a `plot()` method.

The Excel download holds `data_fn()` on one sheet named `sheet_name()`. The picture is the chart as it is on screen, as a PNG file. Both files are named `filename()` followed by the date and time.

In an app made with `cd_app()`, what the reader changes in the customize panel is kept in the dataset. The chart is also registered with the app's AI bridge, which can then list it and read its data.

By default the tools sit over a corner of the chart. With `toolbar_inline = TRUE`, `cd_coverage_plot_ui()` leaves them out, and the page puts them somewhere else (a card's header, for example) with `cd_coverage_plot_toolbar_ui()`, called with the same `id`.

Value

`cd_coverage_plot_ui()` returns the UI as Shiny tags: the chart, with its tools unless `toolbar_inline` is `TRUE`.

`cd_coverage_plot_toolbar_ui()` returns the chart's tools alone, as Shiny tags.

`cd_coverage_plot_server()` returns nothing useful. It is called for what it does.

Examples

```
## Not run:
# in a page's ui
cd_coverage_plot_ui(ns("coverage"))

# in the page's server
cd_coverage_plot_server("coverage",
  filename = reactive("penta3_coverage"),
  data_fn = reactive(my_coverage_data(cache())),
  sheet_name = reactive(i18n$t("opt_penta3")),
  i18n = i18n,
  plot_fun = function(d) plot(d, title = "Penta3 coverage")
)

## End(Not run)
```

`cd_custom_chart_data` *The table a custom chart draws*

Description

Checks a custom chart description, reads the data of the dataset member it names and applies the transforms it lists. The result is the table the chart is drawn from.

Usage

```
cd_custom_chart_data(cache, spec)
```

Arguments

cache	A CacheConnection.
spec	A custom chart description (report kind custom_chart, see datasuite.ui::report_validate_s

Value

The data frame after the spec's transforms.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

spec <- list(
  title = "Antenatal care reporting rate by region, 2023",
  data = list(member = "reporting_rate_admin1"),
  transform = list(
    list(filter = list(year = 2023)),
    list(select = c("adminlevel_1", "anc_rr"))
  ),
  plot = list(geom = "col", x = "adminlevel_1", y = "anc_rr")
)
cd_custom_chart_data(cache, spec)
```

---

cd_default_indicator_set

The indicators this app's tabbed chart cards show

Description

The indicators a tabbed chart card (cd_tabbed_charts_ui()) gives a tab each when the page names none.

Usage

```
cd_default_indicator_set()
```

Details

Reads options(cd2030.default_indicators =). An app sets it to a character vector of indicator codes, or to a function with no arguments that returns one. A function is called each time, so its answer can follow something that is only known later, such as the indicator group. When the option is not set, the result is cd_default_indicators.

Value

A character vector of indicator codes.

Examples

```
cd_default_indicator_set()

old <- options(cd2030.default_indicators = function() c("penta3", "measles1"))
cd_default_indicator_set()
options(old)
```

---

cd_default_indicators

The standard indicators of a tabbed chart card

Description

The indicators a tabbed chart card (`cd_tabbed_charts_ui()`) gives a tab each when the page names none and the app has not set its own. This is the RMNCAH app's set. Read the indicators in use with `cd_default_indicator_set()`, which also takes the app's option into account.

Usage

```
cd_default_indicators
```

Format

A character vector of five indicator codes: `"anc4"`, `"instlivebirths"`, `"low_bweight"`, `"penta3"` and `"measles1"`.

Examples

```
cd_default_indicators
```

`cd_denominator_chip_keys`
Label and hint of a denominator chip

Description

The translation keys of the label and of the hint a denominator chip shows. `cd_denominator_ui()` uses them, and a page that draws a denominator chip of its own can use them to match it.

Usage

```
cd_denominator_chip_keys(is_maternal)
```

Arguments

`is_maternal`	Logical. `TRUE` for the maternal denominator's chip, `FALSE` for the vaccination denominator's.

Value

A list of two translation keys, `label` and `hint`.

Examples

```
cd_denominator_chip_keys(is_maternal = TRUE)
cd_denominator_chip_keys(is_maternal = FALSE)
```

`cd_denominator_choices`
The standard denominator choices

Description

Every denominator the denominator chip can offer, in the form a filter chip takes them. This is the full set, which the RMNCAH app uses. An app with fewer denominators sets its own (see `cd_denominator_options()`).

Usage

```
cd_denominator_choices
```

Format

A named character vector of length 5. The values are the denominator ids and the names are the translation keys of their labels:

- `"dhis2"`: the DHIS2 population projection.
- `"anc1"` and `"penta1"`: the denominators derived from ANC1 and from Penta1.
- `"penta1derived"` and `"anc1derived"`: the growth-projected denominators (Penta1 and ANC1 population growth).

Examples

```
cd_denominator_choices
```

`cd_denominator_header` *Draw the denominator row in a page header*

Description

What a Countdown app adds to the server of every page header: it fills the header's `denominator` output with `cd_denominator_row()` for the denominators the dataset uses. `cd_app()` passes it to the app frame (`app_frame(page_header_extra = )`), which runs it inside each page header's module, so a page never calls it itself.

Usage

```
cd_denominator_header(input, output, session, cache, i18n)
```

Arguments

`input`, `output`, `session`	The `input`, `output` and `session` of the page header's module.
`cache`	A reactive that returns the dataset, a CacheConnection.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.

Details

It reads `cache()$denominator` and, when the app has a maternal denominator (`cd_has_maternal()`), `cache()$maternal_denominator`. The row is drawn once the values it needs are set, and again when one of them changes.

Value

Nothing useful. It is called for what it does.

Examples

```
## Not run:
# cd_app() gives it to the app frame, which calls it in each page header's module
app_frame(app_name, app_version, theme, nav_sections, registry, i18n, language, selected_file,
  data_server = data_server, page_header_extra = cd_denominator_header
)

# the call the frame makes, where the header has uiOutput(ns("denominator"))
moduleServer("header", function(input, output, session) {
  cd_denominator_header(input, output, session, cache = cache, i18n = i18n)
})

## End(Not run)
```

---

cd_denominator_labels

The denominator options' labels

Description

Gives the display label of each denominator option, keyed by its id. With a translator the labels are translated; a label that has no translation stays in English.

Usage

```
cd_denominator_labels(i18n = NULL, ids = .cd_dict_denominators$id)
```

Arguments

`i18n`	Optional translator (`i18n$t(key)`); without it, the English labels.
`ids`	Which denominators, in this order. Default: all six.

Value

A named character vector, id -> label.

Examples

```
cd_denominator_labels()
```

cd_denominator_options

The denominators this app offers

Description

The choices the denominator chip shows in the running app. Use it wherever a page must list or check the app's denominators, so that a page shared between apps never shows one the app does not have.

Usage

```
cd_denominator_options()
```

Details

Reads `options(cd2030.denominator_choices = )`. An app whose indicator group has fewer denominators sets that option to its own named vector (the vaccine app leaves out `"anc1derived"`). When the option is not set, the result is cd_denominator_choices.

Value

A named character vector: the values are the denominator ids and the names are the translation keys of their labels.

Examples

```
cd_denominator_options()

old <- options(cd2030.denominator_choices = c("lbl_denom_dhis2_proj" = "dhis2"))
cd_denominator_options()
options(old)
```

cd_denominator_row

Row naming the denominators in use

Description

The row a Countdown page header shows under its title: a label, one chip naming the vaccination denominator, one naming the maternal denominator when there is one, and a note saying where the denominators are set. `cd_denominator_header()` draws it on every page. Call it directly to show the same row somewhere else.

Usage

```
cd_denominator_row(vaccination, maternal = NULL, i18n)
```

Arguments

`vaccination`	The id of the vaccination denominator in use, for example `"penta1"`. The chip shows the translation of `opt_` followed by the id, or the id in capitals when there is no such translation.
`maternal`	The id of the maternal denominator in use, shown the same way. `NULL` (the default) leaves its chip out, for an app with no maternal denominator.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.

Value

The UI as Shiny tags (a `div`).

Examples

```
## Not run:
cd_denominator_row("penta1", "anc1", i18n)
cd_denominator_row("penta1", i18n = i18n)

## End(Not run)
```

`cd_denominator_ui` *Denominator filter chip*

Description

A filter chip that chooses the denominator the dataset uses, either the vaccination denominator or the maternal one. The choice is kept in the dataset, so every page that reads the denominator follows it. The Denominator Selection page uses one chip for each.

Usage

```
cd_denominator_ui(id, i18n, allow_input = FALSE, is_maternal = FALSE)

cd_denominator_server(
  id,
  cache,
  i18n,
  label = "title_denom_select_best",
  allowInput = FALSE,
  is_maternal = FALSE,
  display_mode = "combined"
)
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label. The server does not use it.
allow_input	Logical. TRUE draws the chip. Default FALSE, which draws nothing.
is_maternal	Logical. TRUE for the maternal denominator, FALSE (the default) for the vaccination denominator. It decides the chip's label and hint, and which value of the dataset the server reads and saves.
cache	A reactive that returns the dataset, a CacheConnection.
label	Not used.
allowInput	Logical. TRUE keeps the chip and the dataset in step. Default FALSE, which does nothing. Give it the value the ui's allow_input has.
display_mode	Not used.

Details

The chip starts empty and the server fills it from the dataset once the chip is on the page: cache()$denominator, or cache()$maternal_denominator when is_maternal is TRUE. When the value changes in the dataset, the chip follows. When the reader chooses a denominator, the server saves it with the dataset's set_denominator() or set_maternal_denominator(). An empty chip saves nothing.

The chip is drawn, and the dataset kept in step, only when asked for: pass allow_input = TRUE to the ui and allowInput = TRUE to the server (the two arguments are spelled differently). With the defaults the ui is NULL and the server does nothing.

The chip's choices are cd_denominator_options().

Value

cd_denominator_ui() returns the UI as Shiny tags (the chip), or NULL when allow_input is FALSE.

cd_denominator_server() returns nothing useful. It is called for what it does.

Examples

```
## Not run:
# in a page's ui
cd_denominator_ui(ns("vaxx_denominator"), i18n, allow_input = TRUE)

# in the page's server
cd_denominator_server("vaxx_denominator", cache, i18n, allowInput = TRUE)

## End(Not run)
```

cd_describe_columns *Read column names with the data dictionary*

Description

Splits each name into its parts (`cd_dictionary()`) and says in plain words what the column holds – e.g. `cov_penta3_penta1derived`: "Coverage of Pentavalent 3 (%), denominator: Penta1 population growth". A name it cannot read gets type `"unknown"` and no description: it never guesses.

Usage

```
cd_describe_columns(names)
```

Arguments

`names`	Column names.

Value

A data frame, one row per name: `column`, `type` (count, coverage, population, population_growth, survey, survey_upper, survey_lower, reporting_rate, fixed, unknown), `indicator`, `denominator`, `population` (the parts' ids, or `NA`) and `description`.

Examples

```
cd_describe_columns(c("cov_penta3_penta1derived", "totinftpenta_anc1", "r_penta1",
                      "vacc_rr", "year", "xyz"))
```

cd_dictionary *The data dictionary: what the ids and column names in the package's data mean*

Description

The package names its data from a few parts: an indicator (`penta1`), a denominator option (`anc1derived`), a target population (`totinftpenta`) and prefixes such as `cov_` (coverage) or `r_` (survey estimate). This is what each part means, and how they combine. Read meanings from here, not from the spelling: `anc1` and `penta1` are the **ANC1-derived** and **Penta1-derived** denominators, while `anc1derived` and `penta1derived` are the **population growth** ones.

Usage

```
cd_dictionary()
```

Value

A list of data frames: `denominators` (id, label, meaning, levels, key – the label's translation key), `indicators` (id, label, group, computed), `populations` (id, label, meaning, unit), `reporting_rates` (id, meaning), `columns` (fixed column names: id, meaning) and `grammar` (pattern, meaning, example).

See Also

`cd_describe_columns()` to read column names with it.

Examples

```
cd_dictionary()$denominators[, c("id", "label")]
```

`cd_finish_plot` *Finish a plot: apply the chart options*

Description

The last step of every plot method: `resolve_chart_options()` then `apply_chart_options()`. While a report renders (`export_report()`) the report's saved options sit underneath: the dataset-wide ones, then those saved for this type of graph (`cd_chart_type()`), then those saved for this very chart (`cd_chart_id()`); what the plot method or the template passes wins over all of them.

Usage

```
cd_finish_plot(p, options = NULL, ..., .source = NULL)
```

Arguments

`p`	A ggplot.
`options, ...`	See `resolve_chart_options()`.
`.source`	The data the plot was made from; its `cd_chart_id()` says which chart this is.

Value

The ggplot with the options applied.

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) + ggplot2::geom_point()
cd_finish_plot(p, title = "Weight and fuel use", x_title = "Weight", y_title = "Miles per gallon")

# the same as one options object, the way a plot method receives it
cd_finish_plot(p, cd_chart_options(title = "Weight and fuel use", legend_position = "bottom"))
```

cd_has_maternal *Whether this app has a maternal denominator*

Description

Tells a shared page whether to show what belongs to the maternal denominator: its chip on the Denominator Selection page and its entry in the page header's denominator row.

Usage

```
cd_has_maternal()
```

Details

The answer is the app's `has_maternal` setting (see `cd_cfg()`). When the app does not set it, the answer is `TRUE` unless `getOption("cd2030.selected_group")` is `"vaccine"`.

Value

`TRUE` or `FALSE`.

Examples

```
old <- options(cd2030.config = list(has_maternal = FALSE))
cd_has_maternal()
options(old)
```

cd_indicator_labels *The indicators' labels*

Description

Gives the display label of each indicator, keyed by its code (`anc1` -> "ANC 1"). Charts use it to name indicators on an axis, in place of the column names. With a translator the labels are translated (the key of an indicator is `opt_<code>`); a label that has no translation stays in English, and a code the package does not know is returned as it is.

Usage

```
cd_indicator_labels(i18n = NULL, indicators = names(.cd_dict_indicator_labels))
```

Arguments

`i18n`	Optional translator (`i18n$t(key)`); without it, the English labels.
`indicators`	Which indicators, in this order. Default: every indicator the package knows.

Value

A named character vector, code -> label.

Examples

```
cd_indicator_labels(indicators = c("anc1", "penta3", "instlivebirths"))
```

---

cd_indicator_ui *Indicator filter chip*

Description

A filter chip that chooses one indicator. A page puts `cd_indicator_ui()` in its filter bar or in a card, calls `cd_indicator_server()` in its server, and reads the chosen indicator from the reactive the server returns. The chip starts on its first choice.

Usage

```
cd_indicator_ui(
  id,
  i18n,
  label = NULL,
  tooltip = NULL,
  indicators = NULL,
  select_all = FALSE
)

cd_indicator_server(id)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`label`	The translation key of the chip's label. `NULL` (the default) uses `"title_global_indicator"`.
`tooltip`	The translation key of a hint shown with the chip, or `NULL` (the default) for none.
`indicators`	A character vector of indicator codes, the chip's choices. Without names, each indicator's label is the translation of `opt_` followed by its code (`"opt_penta3"`). With names, the names are the translation keys of the labels. `NULL` (the default) offers every indicator, as `get_all_indicators()` lists them.
`select_all`	Logical. `TRUE` adds an "all" choice first, whose value is the empty string `""`. Default `FALSE`.

Value

`cd_indicator_ui()` returns the UI as Shiny tags (the chip).

`cd_indicator_server()` returns a reactive holding the chosen indicator's code as a string. It holds `""` when "all" is chosen, and `NULL` until the chip has reported its first value.

Examples

```
## Not run:
# in a page's ui
cd_indicator_ui(ns("indicator"), i18n, indicators = c("penta3", "measles1"), select_all = TRUE)

# in the page's server
indicator <- cd_indicator_server("indicator")
observe(print(indicator()))

## End(Not run)
```

`cd_member_data` *The table a CacheConnection member gives*

Description

Reads an active binding, or calls a method with `args` (arguments it doesn't take are left out), and returns the result as a plain data frame (a map's geometry dropped).

Usage

```
cd_member_data(cache, member, args = list())
```

Arguments

`cache`	A `CacheConnection`.
`member`	A member name, one of `cd_chartable_members()`.
`args`	A named list of arguments for a method.

Value

A data frame.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

# a precomputed result
cd_member_data(cache, "reporting_rate_national")
```

```
# a method, with its arguments
rates <- cd_member_data(cache, "calculate_reporting_rate", args = list(admin_level = "adminlevel_1"))
head(rates)
```

`cd_methodology_defaults` *The constants of the Countdown methodology*

Description

Lists the constants the package's methodology uses (cutoffs, thresholds, default rates and targets) with the values the code runs with, so the documentation can be generated from them.

Usage

```
cd_methodology_defaults(group = c("all", "rmncah", "vaccine"))
```

Arguments

`group`	Which entries to return: `"all"` (the default), or those that apply to the `"rmncah"` or the `"vaccine"` indicator group (entries for `"both"` are included in each).

Value

A tibble with one row per constant and the columns:

- `id`: a stable snake_case identifier.
- `step`: one of `"data_quality"`, `"adjustment"`, `"denominators"`, `"coverage"`, `"subnational"`, `"mortality"`.
- `group`: `"both"`, `"rmncah"` or `"vaccine"`.
- `value`: a list column; each element is a number, a string or a vector.
- `unit`: for example `"%"`, `"x MAD"`, `"ratio"`, `"proportion"`, or `""`.
- `label`: a short English label.
- `note`: one English sentence on how the app uses it.
- `source`: the file of the package (within `R/`) that uses it.

Examples

```
cd_methodology_defaults()
cd_methodology_defaults("vaccine")
```

cd_minimal_theme *A minimal chart theme*

Description

ggplot2::theme_minimal() with a choice of grid lines, for the apps' light charts (cd2030.pooled's charts, the subnational coverage dot plot and heat map).

Usage

```
cd_minimal_theme(
  base_size = 11,
  grid = c("all", "major", "x", "y", "none"),
  grid_colour = NULL
)
```

Arguments

base_size	The base font size, in points. The default is the size the charts of the apps are written in (see cd_type_scale()): another size makes every text of the chart larger or smaller by the same ratio.
grid	Which grid lines to keep: "all" (the theme's own), "major" (no minor lines), "x" (only the vertical major lines, for horizontal bars and dot plots), "y" (only the horizontal major lines) or "none".
grid_colour	The colour of the lines "x" or "y" keeps, or NULL for the theme's own.

Value

A ggplot2 theme.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, factor(cyl))) +
  geom_point() +
  cd_minimal_theme(grid = "x")
```

cd_nav_start *The standard sidebar sections of a Countdown app*

Description

The parts of the sidebar every Countdown app shares. An app builds its nav tree from them, adds its own groups, and passes the list to cd_app() as nav_sections. Each link opens the screen whose id is the link's tab name (the names are listed below), so the app must have a page with that id.

Usage

```
cd_nav_start()

cd_nav_quality()

cd_nav_denominators()

cd_nav_national(extra = list())

cd_nav_subnational()
```

Arguments

`extra`	A list of more nav items (`cd_nav_item()`) for the National Analysis group, placed after the coverage target link and before the inequality group. The RMNCAH app adds its Continuum of Care page this way. Default `list()`, none.

Details

- `cd_nav_start()`: the "start" section, with the Introduction (`introduction`) and Load Data (`upload_data`) screens.
- `cd_nav_quality()`: the data quality section. One group holds `reporting_rate`, `outlier_detection`, `data_completeness`, `internal_consistency` and `overall_score`. A second group holds `data_adjustment` and `data_adjustment_changes`.
- `cd_nav_denominators()`: the denominators section, with `denominator_assessment` and `denominator_selection`.
- `cd_nav_national()`: the National Analysis group, with `national_coverage`, `national_target`, the items in `extra`, and a group for inequality holding `national_inequality` and `equity_assessment`.
- `cd_nav_subnational()`: the Sub-National Analysis group, with `subnational_coverage`, `subnational_inequality` and `subnational_target`.

The first three are whole sections. The last two are groups of links: put them inside a section of the app's own, made with `cd_nav_section()`.

Everything from the denominators down is marked `requires_adjustment = TRUE`: those links stay locked until the data is adjusted.

The labels are translated with the translator the app registered with `cd_use_i18n()`, so call these after it.

Value

`cd_nav_start()`, `cd_nav_quality()` and `cd_nav_denominators()` each return a nav section, as `cd_nav_section()` makes one: a list with the section's `label` and its `items`.

`cd_nav_national()` and `cd_nav_subnational()` each return a nav item, as `cd_nav_item()` makes one: a group whose `children` are the links.

Examples

```
## Not run:
nav <- list(
  cd_nav_start(),
  cd_nav_quality(),
  cd_nav_denominators(),
  cd_nav_section("lbl_nav_section_analysis", cd_nav_national(), cd_nav_subnational())
)

## End(Not run)
```

---

cd_only_denominators *Keep only the denominators this app offers*

Description

Filters a named list or vector, such as a chart's legend labels or category labels, down to the entries named after a denominator the app offers. Labels are usually written for every denominator, and one the app does not offer must not appear in a chart.

Usage

```
cd_only_denominators(x)
```

Arguments

`x`	A list or vector named by denominator id.

Value

`x` without the entries whose name is neither `"un"` (the UN projection, always kept) nor one of the ids in `cd_denominator_options()`. The order and the type of `x` are kept.

Examples

```
labels <- c(un = "UN projection", dhis2 = "DHIS2 projection", other = "Not a denominator")
cd_only_denominators(labels)
```

`cd_palette_chip` *Colour palette chip for a page's maps*

Description

A filter chip that chooses the colour palette of the maps on a page. It belongs in the bar of map options above the maps (`cd_map_options()`). The page reads the choice as `input$<inputId>` and passes it to the map plots, which accept these palette names.

Usage

```
cd_palette_chip(inputId, i18n, first = "Greens")
```

Arguments

`inputId`	The input's id. Inside a module, pass it through the module's `ns()`.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`first`	The palette listed first, and so the one the page opens with: `"Greens"` (the default), `"Blues"`, `"Reds"` or `"Purples"`. Any other value leaves the order as it is, with `"Greens"` first.

Details

The choices are four sequential RColorBrewer palettes: `"Greens"`, `"Blues"`, `"Reds"` and `"Purples"`. The chip starts on its first choice, so `first` decides which palette a page opens with. This is a plain input, not a module, so there is no server function to call.

Value

The UI as Shiny tags (the chip). Its value, `input$<inputId>`, is the chosen palette's name.

Examples

```
## Not run:
cd_map_options(cd_palette_chip(ns("palette"), i18n, first = "Blues"))

## End(Not run)
```

`cd_plot_arguments` *Arguments every chart shares*

Description

Every chart in the package (each `plot()` method, the equiplots and `plot_comparison()`) changes its text and its look in the same way, with the arguments below. They come after the arguments that say what to draw, and they have the same names as the chart options (`cd_chart_options()`), so what is given for one chart can be given for any other.

Arguments

`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Text

`title`, `subtitle`, `caption`, `x_title`, `y_title` and `legend_title` replace a chart's own text. `NULL`, the default, keeps the text the chart writes itself. To remove a text, hide it: `show_title = FALSE`, `show_x_title = FALSE`, `show_legend_title = FALSE`.

Look

Everything else is a chart option, given by name in `...`: the background (`background_color`, `panel_color`), fonts and sizes (`font_family`, `text_scale`, `title_size`, `axis_text_size`), the axes (`x_limits`, `y_limits`, `x_text_angle`, `flip`), the grid (`grid`, `grid_color`), the legend (`legend_position`, `legend_labels`, `show_legend`) and the colours (`colors`). `cd_chart_options()` lists them all.

One look for several charts

Make the options once and give them to each chart as `options`. A chart option named in the call wins over `options`.

Which text is used

The six text arguments say what the chart should read; `options` holds what someone changed afterwards (in the apps, with the chart's customize panel), so a text in `options` wins over the text argument. From least to most: the chart's own text, the text arguments, `options`, and a chart option named in `...`.

Names that are gone

`x_axis` and `x_label` were other names for `x_title`, `y_axis` and `y_label` for `y_title`, and `legend` and `fill_label` for `legend_title`. They are no longer accepted: a chart given one stops and names the argument to use.

Examples

```
## Not run:
rates <- calculate_average_reporting_rate(data, admin_level = "adminlevel_1")

# the same arguments on any chart
plot(rates, title = "Reporting by region", y_title = "Region", legend_title = "Rate")
plot(calculate_district_reporting_rate(data, 90), title = "Districts reporting", y_title = "Percent")

# background and fonts
plot(rates, background_color = "grey95", font_family = "serif", text_scale = 1.2)

# one look for several charts
look <- cd_chart_options(font_family = "serif", background_color = "grey95", legend_position = "right")
plot(rates, options = look)
plot(calculate_district_reporting_rate(data, 90), options = look)

## End(Not run)
```

`cd_population_ui` *Population filter chip*

Description

A filter chip that chooses the DHIS2 population the growth-projected denominators (ANC1 and Penta1 population growth) are built from. The choice is kept in the dataset.

Usage

```
cd_population_ui(id)

cd_population_server(id, cache)
```

Arguments

`id`	The module's id.
`cache`	A reactive that returns the dataset, a CacheConnection.

Details

The choices are `"totlivebirths_dhis2"` (live births), `"totbirths_dhis2"` (total births), `"totpop_dhis2"` (total population) and `"totunder1_dhis2"` (population under one year).

Once the chip is on the page, the server sets it to the dataset's value, `cache()$derivation_population`, when there is one, and does so again whenever `cache()` changes. Each choice the chip reports is saved with the dataset's `set_derivation_population()`.

The ui takes no translator: its labels are translated with the translator the app registered with `cd_use_i18n()`.

Value

`cd_population_ui()` returns the UI as Shiny tags (the chip).

`cd_population_server()` returns a reactive holding the chosen population's id as a string (`NULL` until the chip has reported a value).

Examples

```
## Not run:
# in a page's ui
cd_population_ui(ns("population"))

# in the page's server
population <- cd_population_server("population", cache)

## End(Not run)
```

`cd_rate_status_cell` *Table cell showing a rate with its status*

Description

Draws one cell of a table of rates, such as reporting rates: a small coloured shape that says how good the rate is, followed by the rate rounded to a whole number. The shape differs with the status as well as the colour, so the status never depends on colour alone. Use it as the `cell` of a `reactable::colDef()`, for example in the `columns` of `cd_table_server()`.

Usage

```
cd_rate_status_cell(value)
```

Arguments

`value`	The rate, a single number on a scale of 0 to 100. `NULL` or `NA` gives an empty cell.

Details

The status has fixed limits: a red triangle below 70, an amber diamond from 70 to below 90, and a green circle from 90.

Value

The cell as a Shiny tag (a `span` holding the shape and the number), or `""` when `value` is `NULL` or `NA`.

Examples

```
cd_rate_status_cell(64.2)
cd_rate_status_cell(95)
cd_rate_status_cell(NA)

# as the cell of a table column
reactable::colDef(cell = cd_rate_status_cell)
```

---

cd_report_context *The report context of a dataset*

Description

What the report builder (datasuite.ui) needs from a Countdown dataset: its charts and tables drawn in the report's look, the pictures and templates kept in it, its saved chart options, its fields (country, coverage values), years, regions and flag. Every report function accepts the dataset itself too (`datasuite.ui::as_report_context()`).

Usage

```
cd_report_context(cache)
```

Arguments

`cache`	A CacheConnection.

Value

A report context (`datasuite.ui::report_context()`).

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

context <- cd_report_context(cache)
context$years()
context$fields(NULL, "en")$country
```

cd_request_bayes_packages *Ask DataSuite to install the Bayesian model's packages*

Description

Inside DataSuite (an app it started), asks it to install them: it shows the progress in its status bar and then offers to restart the app. Outside DataSuite it does nothing; install them with `cd_bayes_install_command()`.

Usage

```
cd_request_bayes_packages(packages = cd_bayes_packages_missing())
```

Arguments

`packages`	The packages to install, by default the missing ones.

Value

`TRUE` when DataSuite was asked, invisibly.

Examples

```
## Not run:
# in an app DataSuite started, before the model is first used
if (length(cd_bayes_packages_missing()) > 0) {
  cd_request_bayes_packages()
}

# one package only
cd_request_bayes_packages("bayescoveragedeploy")

## End(Not run)
```

cd_saved_copy_name *File name of a dataset's saved copy*

Description

Builds the name of the `.rds` file a finished dataset is saved to: `<stem>_<group>.rds`, for example `Benin_rmncah.rds` or `Benin_vaccine.rds`. One source file can be loaded by more than one app, and the group in the name keeps each app's finished copy apart.

Usage

```
cd_saved_copy_name(stem)
```

Arguments

`stem`	The name of the source file without its extension, a string.

Value

The file name (not a full path), a string. The group is the one `cd_wizard_indicator_group()` returns.

Examples

```
old <- options(cd2030.app_group = "rmncah")
cd_saved_copy_name("Kenya")
cd_saved_copy_name(tools::file_path_sans_ext(basename("C:/data/Kenya_2024.xlsx")))
options(old)
```

`cd_scope` *Define the scope of a scoped page*

Description

A scope says at which geography a page runs its analysis: which admin level and region filters the page shows, and what admin level and region it hands to the analysis. Pages that run the same analysis at a different geography (national coverage and sub-national coverage, for example) differ only in their scope. Pass the result to `cd_scoped_page_ui()` and `cd_scoped_page_server()`, or to `cd_scope_filters()` and `cd_scope_server()`.

Usage

```
cd_scope(
  kind = c("national", "level", "region", "level_region"),
  fixed_level = NULL,
  show_district = FALSE
)
```

Arguments

`kind`	The kind of scope: `"national"` (the default), `"level"`, `"region"` or `"level_region"`.
`fixed_level`	An admin level (for example `"district"`) handed to the analysis in place of the level of the filter bar. `NULL` (the default) hands over the level of the filter bar. A `"national"` scope ignores it.
`show_district`	Logical. For `kind = "level_region"`, whether a district can be chosen as the region. Other kinds ignore it. Default `FALSE`.

Details

The kinds of scope:

- `"national"`: no filters. The analysis gets the admin level `"national"` and no region.
- `"level"`: an admin level chip only (admin level 1 or district). The analysis gets the chosen level and no region.
- `"region"`: a region chip only, listing the admin level 1 areas. The analysis gets the chosen area and the admin level `"adminlevel_1"`, or `fixed_level` when it is given.
- `"level_region"`: an admin level chip and a region chip. The region chip is shown at admin level 1, and also at the district level when `show_district` is `TRUE`.

Value

A list with the elements `kind`, `fixed_level` and `show_district`, as given.

Examples

```
cd_scope()
cd_scope("region", fixed_level = "district")
cd_scope("level_region", show_district = TRUE)
```

---

cd_scope_filters *Filter bar of a scope*

Description

The filter bar holding the admin level and region chips a scope asks for. `cd_scoped_page_ui()` uses it. Call it yourself in a page that lays out its own body but still takes its geography from a scope, and pair it with `cd_scope_server()`.

Usage

```
cd_scope_filters(ns, i18n, scope)
```

Arguments

`ns`	The page's namespace function, `NS(id)`.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`scope`	The page's scope, from `cd_scope()`.

Details

The chips are an admin level filter (`cd_admin_level_ui()`) with the id `ns("scope")`. So in the page's module server, call `cd_scope_server("scope", ...)` with the same scope.

Value

The UI as Shiny tags (a filter bar), or `NULL` for a `"national"` scope, which has no filters.

Examples

```
## Not run:
scope <- cd_scope("level_region")

my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, filters = cd_scope_filters(ns, i18n, scope), my_cards(ns))
}

## End(Not run)
```

`cd_scope_server` *Admin level and region of a scope*

Description

The server half of `cd_scope_filters()`: it gives the admin level and the region a scope resolves to, as two reactives ready to hand to an analysis. Call it inside the page's module server. `cd_scoped_page_server()` does this for a whole scoped page.

Usage

```
cd_scope_server(id, cache, i18n, scope)
```

Arguments

`id`	The id of the scope's filter within the page's module. Use `"scope"`, the id `cd_scope_filters()` gives the chips.
`cache`	A reactive that returns the dataset, a CacheConnection.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`scope`	The page's scope, from `cd_scope()`.

Details

For a `"national"` scope nothing is read from the page. For the other kinds it runs the admin level filter (`cd_admin_level_server()`) with the settings the scope asks for, so the two reactives wait until that filter has a value.

Value

A list of two reactives:

- `admin_level`: `"national"` for a national scope. Otherwise the scope's `fixed_level` when it has one, else the level of the filter (`"adminlevel_1"` or `"district"`).
- `region`: the name of the chosen area, or `NULL` when the scope has no region chip.

Examples

```
## Not run:
my_page_server <- function(id, cache, i18n) {
  moduleServer(id, function(input, output, session) {
    geo <- cd_scope_server("scope", cache, i18n, cd_scope("level_region"))
    observe(print(list(geo$admin_level(), geo$region())))
  })
}

## End(Not run)
```

---

cd_scoped_page_ui *A whole page that runs an analysis at a scope*

Description

Builds a page from an analysis module and a scope: the page's header and body, the scope's filter bar, and the analysis module inside. Pages that run the same analysis at a different geography then differ only in the scope they pass (see `cd_scope()`), and the filter bar and its wiring are written once.

Usage

```
cd_scoped_page_ui(id, i18n, scope, inner_ui, ...)

cd_scoped_page_server(
  id,
  cache,
  i18n,
  scope,
  inner_server,
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id, which is also the page's id in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`scope`	The page's scope, from `cd_scope()`. Give the ui and the server the same one.
`inner_ui`	The analysis module's ui function, called as `inner_ui(id, i18n, ...)`.
`...`	More arguments for `inner_ui`.
`cache`	A reactive that returns the dataset, a CacheConnection.
`inner_server`	The analysis module's server function, called as `inner_server(id, cache, i18n, admin_level, region, active = active)`.

`active` A reactive that is `TRUE` once the page has been opened with data loaded. It is passed to `inner_server`, which is expected to do no work before that. Default `reactive(TRUE)`.

Details

The analysis module is the app's own pair of functions:

- `inner_ui(id, i18n, ...)` returns the page's cards.
- `inner_server(id, cache, i18n, admin_level, region, active = )` takes the admin level and the region as two reactives (see `cd_scope_server()` for what they hold).

Both are called with the id `"body"` inside the page's namespace. The page's title, section and subtitle come from the app's page registry, so `id` must be the id of a registered page.

Value

`cd_scoped_page_ui()` returns the page's UI as Shiny tags.

`cd_scoped_page_server()` returns what `inner_server` returns.

Examples

```
## Not run:
scope <- cd_scope("region", fixed_level = "district")

subnational_target_ui <- function(id, i18n) {
  cd_scoped_page_ui(id, i18n, scope, target_ui)
}
subnational_target_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  cd_scoped_page_server(id, cache, i18n, scope, target_server, active)
}

## End(Not run)
```

---

cd_start_bayes_workers

Start the background workers that fit the Bayesian models

Description

Sets the R session's `future` plan to `multisession`, so that the Bayesian page fits its models in other R processes and the app stays usable meanwhile. It changes the plan only while the plan is the default (`sequential`): a session that already has a plan of its own keeps it, and a second call does nothing. There is one worker by default, so a fit asked for while another runs waits its turn. `options(cd2030.bayes_workers = )` sets another number (at least one is always used).

Usage

```
cd_start_bayes_workers()
```

Value

The session's `future` plan, invisibly.

Examples

```
## Not run:
cd_start_bayes_workers()
future::nbrOfWorkers()

# two fits at a time: set the option before the first call
options(cd2030.bayes_workers = 2)
cd_start_bayes_workers()

## End(Not run)
```

---

cd_tab_indicator *The indicator of a tab of charts*

Description

The `serverInput` function given to `cd_tabbed_charts_server()` receives the indicator of its tab as a string, and for the Custom tab as a reactive that returns the indicator the reader chose. `cd_tab_indicator()` gives one way to read both.

Usage

```
cd_tab_indicator(indicator)
```

Arguments

`indicator`	What `serverInput` received: an indicator's code, or a reactive that returns one.

Value

A function with no arguments that returns the indicator's code. For the Custom tab it must be called where a reactive can be read (in a `reactive()`, a render function or a `plot_fun`).

Examples

```
ind <- cd_tab_indicator("penta3")
ind()

## Not run:
# in a page's server: one serverInput for the indicators' tabs and the Custom tab
cd_tabbed_charts_server("panel",
  serverInput = function(id, current_indicator) {
    ind <- cd_tab_indicator(current_indicator)
    cd_coverage_plot_server(
      id = id,
      filename = reactive(paste0(ind(), "_coverage")),
      data_fn = reactive(cache()$get_filtered_coverage(ind(), "national")),
      sheet_name = reactive(i18n$t(paste0("opt_", ind()))),
      i18n = i18n
    )
  }
)

## End(Not run)
```

---

cd_tabbed_charts_ui *Chart card with one tab for each indicator*

Description

A card that shows the same chart for several indicators, one tab each, with a "Custom" tab where the reader chooses any other indicator. The page says how one chart is drawn (`uiInput` and `serverInput`), and this module makes the tabs, switches between them and puts the chart tools of the tab on show in the card's header. Many analysis pages are built from it.

Usage

```
cd_tabbed_charts_ui(
  id,
  i18n,
  title_key,
  uiInput,
  indicators = NULL,
  customIndicators = get_analysis_indicators(),
  showCustom = TRUE,
  width = 12
)

cd_tabbed_charts_server(
  id,
```

```
  serverInput,
  indicators = NULL,
  customIndicators = get_analysis_indicators(),
  showCustom = TRUE,
  selected_tab = reactive(NULL),
  i18n = cd_i18n()
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label. In `cd_tabbed_charts_server()` it is the last argument and defaults to the translator the app registered with `cd_use_i18n()`.
`title_key`	The translation key of the card's title.
`uiInput`	A function `function(id, toolbar_inline)` that returns the UI of one tab's chart (see Details).
`indicators`	A character vector of indicator codes, one tab each, in this order. `NULL` (the default) uses `cd_default_indicator_set()`.
`customIndicators`	The indicators the Custom tab's chip offers, as `cd_indicator_ui()` takes them. Default `get_analysis_indicators()`. Only the ui uses it. The server accepts it and ignores it.
`showCustom`	Logical. `TRUE` (the default) adds the Custom tab after the indicators' tabs.
`width`	The card's width, passed to the kit's `cd_chart_card()`. Default `12`. The kit's card takes a width and ignores it: a card is as wide as the place it is put in.
`serverInput`	A function `function(id, indicator)` that starts the server of one tab's chart (see Details).
`selected_tab`	A reactive that returns the key of a tab to show (an indicator's code or `"custom"`), or `NULL`. When its value changes to a key, the card switches to that tab. It lets another card's tabs drive this card's. Default `reactive(NULL)`, which never switches.

Details

`uiInput` and `serverInput` are the two halves of one chart, usually `cd_coverage_plot_ui()` and a function that calls `cd_coverage_plot_server()`:

- `uiInput(id, toolbar_inline = TRUE)` is called once for each tab and returns that tab's chart. It must leave the chart's tools out when `toolbar_inline` is `TRUE`, because the card's header shows them. The header draws the tools with `cd_coverage_plot_toolbar_ui()`, so the chart must be a `cd_coverage_plot_ui()` or use the same ids.
- `serverInput(id, indicator)` starts the chart of one tab. It is called once for each indicator, with the indicator's code as both arguments. For the Custom tab it is called once, with the id `"custom"` and a reactive that returns the indicator the reader chose;

read either with cd_tab_indicator(). It runs inside this module, so id is used as it is, without ns().

A tab's label is the translation of opt_ followed by the indicator's code. The tabs that are not on show are hidden, and Shiny does not draw a hidden chart.

Give indicators and showCustom the same values in the ui and the server call.

Value

cd_tabbed_charts_ui() returns the UI as Shiny tags (the card).

cd_tabbed_charts_server() returns a reactive holding the key of the tab on show: an indicator's code, or "custom".

Examples

```
## Not run:
# in a page's ui
cd_tabbed_charts_ui(ns("panel"), i18n, "title_coverage_national", cd_coverage_plot_ui,
  indicators = c("penta3", "measles1")
)

# in the page's server
cd_tabbed_charts_server("panel",
  serverInput = function(id, indicator) {
    cd_coverage_plot_server(
      id = id,
      filename = reactive(paste0(indicator, "_coverage")),
      data_fn = reactive(my_coverage_data(cache(), indicator)),
      sheet_name = reactive(i18n$t(paste0("opt_", indicator))),
      i18n = i18n
    )
  },
  indicators = c("penta3", "measles1")
)

## End(Not run)
```

---

cd_table_card_ui *Flextable in a card with picture and Excel downloads*

Description

A card that shows a formatted table (a flextable) on screen, with two downloads in the card's header: the table as a picture and its data as an Excel workbook. Use it for a table whose layout matters, such as a scorecard. The Overall Score page uses it. For a plain table of data, use cd_table_ui().

Usage

```
cd_table_card_ui(id, i18n, title, ..., width = 12)

cd_table_card_server(
  id,
  i18n,
  data,
  table_fun,
  filename,
  sheet = "lbl_score_metric_header",
  about = NULL
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`title`	The card's title: text that is already translated (`i18n$t("...")`), or a tag.
`...`	More arguments for the kit's `cd_chart_card()`, which passes them to `cd_card()`: for example `status = "success"`.
`width`	The card's width, passed to the kit's `cd_chart_card()`. Default `12`. The kit's card takes a width and ignores it: a card is as wide as the place it is put in.
`data`	A reactive that returns what the table shows. It is given to `table_fun` and written to the Excel workbook.
`table_fun`	A function `function(data)` that returns the flextable.
`filename`	A reactive that returns the first part of the downloaded files' name. The date and time and the extension are added to it.
`sheet`	The translation key of the workbook's sheet name. Default `"lbl_score_metric_header"`.
`about`	What the table is, for the app's AI: `NULL` (the default), a list, or a function that returns a list. See `datasuite.ui::ai_register_component()`.

Details

The page gives the data and one function that turns it into a flextable. The table on screen and the picture are drawn with the same function, so they always match. The picture is a PNG file. The workbook holds `data()` on one sheet. Both files are named `filename()` followed by the date and time.

The table is drawn once `data()` has a value. It is also registered with the app's AI bridge, which can then list it and read its data.

Value

`cd_table_card_ui()` returns the UI as Shiny tags (the card).

`cd_table_card_server()` returns nothing useful. It is called for what it does.

Examples

```
## Not run:
# in a page's ui
cd_table_card_ui(ns("score"), i18n, i18n$t("title_score_main"), status = "success")

# in the page's server
cd_table_card_server("score", i18n,
  data = reactive(my_scores(cache())),
  table_fun = function(d) flextable::flextable(d),
  filename = reactive("overall_score")
)

## End(Not run)
```

---

cd_table_ui *Table card with a year or indicator control*

Description

A card that shows a data frame as a table, with a control above it (a year chip or an indicator chip) and a download of the data as an Excel workbook in the card's header. The module translates the column headings it knows, rounds the numbers and aligns them. Use it for the table that goes with a page's charts. For a formatted table (a flextable), use `cd_table_card_ui()`.

Usage

```
cd_table_ui(
  id,
  i18n,
  title_key,
  control_type = c("year", "indicator"),
  width = 6
)

cd_table_server(
  id,
  cache,
  i18n,
  data,
  columns = NULL,
  control_type = c("year", "indicator"),
  filename = reactive("download"),
  label_key = "btn_global_download_data",
  extension = reactive("xlsx"),
  data_transform = NULL,
  excel_write_fun = NULL,
```

```
  excel_sheet = NULL,
  excel_title = NULL,
  about = NULL
)
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label.
title_key	The translation key of the card's title.
control_type	The control above the table: "year" (the default) or "indicator".
width	The card's width, passed to the kit's cd_table_card(). Default 6. The kit's card takes a width and ignores it: a card is as wide as the place it is put in.
cache	A reactive that returns the dataset, a CacheConnection.
data	A reactive that returns the data frame the table shows.
columns	A named list of reactable::colDef(), named by column. Each one replaces or adds to what the module sets for that column (heading, cell, alignment). Entries for columns the data does not have are dropped. NULL (the default) changes nothing.
filename	A reactive that returns the first part of the downloaded file's name. Default reactive("download").
label_key	The translation key of the download button's label. Default "btn_global_download_data".
extension	A reactive that returns the downloaded file's extension. Default reactive("xlsx"). The file is an Excel workbook whatever the extension says.
data_transform	A function function(data) applied to data() before the table is drawn, for example to choose or rename columns. NULL (the default) shows data() as it is.
excel_write_fun	A function function(wb, data) that writes data() into the openxlsx workbook wb, adding the sheets it needs. NULL (the default) uses excel_sheet, or the plain download described in Details.
excel_sheet	The translation key of the sheet's name, for a download that holds data() on one sheet. Used only when excel_write_fun is NULL. Default NULL.
excel_title	The translation key of a title written above the data on that sheet. Used only with excel_sheet. Default NULL, no title.
about	What the table is, for the app's AI: NULL (the default), a list, or a function that returns a list. See datasuite.ui::ai_register_component().

Details

The control. Give control_type the same value in the ui and the server call.

- `"year"`: a chip listing the dataset's years (`cache()$data_years`), which starts on the first of them. When the data has a `year` column, the table shows only the rows of the chosen year.
- `"indicator"`: an indicator chip (`cd_indicator_ui()`) offering every indicator. The module does not filter by it. The page reads the chosen indicator from the reactive the server returns and builds `data` from it.

The columns. These headings are translated without the page doing anything: `year`, `adminlevel_1`, `district` and `month`; columns whose name starts with `cov_` (coverage); columns whose name ends with `_med` or `_mad` (median and MAD of an indicator) and the indicator's own column beside them; and, with the indicator control, the column named after the chosen indicator. Every other column keeps its name unless `columns` gives it a heading. Numbers are shown rounded to whole numbers and aligned to the right, except `year` and `month`.

The download. The file is an Excel workbook named `filename()`, then the control's value, then the date and time. What it holds depends on the arguments:

- With `excel_write_fun`, that function writes the workbook from `data()`.
- Else with `excel_sheet`, `data()` is written on one sheet with that name, under `excel_title` when it is given.
- With neither, the data the table is drawn from (`data()` after `data_transform`) is written on one sheet named "Data".

With a year control the download holds the year the control is on, as the table does and as the file's name says.

The table is also registered with the app's AI bridge, which can then list it and read its data.

Value

`cd_table_ui()` returns the UI as Shiny tags (the card).

`cd_table_server()` returns a reactive holding the control's value: the chosen year as a string, or the chosen indicator's code.

Examples

```
## Not run:
# in a page's ui
cd_table_ui(ns("rates"), i18n, "title_rr_main", control_type = "year")

# in the page's server
year <- cd_table_server("rates", cache, i18n,
  data = reactive(my_rates(cache())),
  columns = list(anc_rr = reactable::colDef(name = "ANC", cell = cd_rate_status_cell)),
  filename = reactive("reporting_rate"),
  excel_sheet = "title_rr_main"
)

## End(Not run)
```

cd_type_scale *The sizes of the text of a chart*

Description

The one scale the charts of the Countdown apps are written in, for a chart drawn outside the package that is to look like them.

Usage

```
cd_type_scale()
```

Value

A named list: `title`, `subtitle`, `axis_title`, `axis_text`, `legend`, `strip` and `caption` in points (for `ggplot2::element_text()`); `label` and `label_small` in millimetres (the `size` of `ggplot2::geom_text()`).

Examples

```
cd_type_scale()$title
```

cd_upload_data_ui *The Load Data screen*

Description

The whole Load Data screen of a Countdown app: the page header, with its help button, and the Load Data wizard with all seven steps. The server starts every step's server, builds the steps' pages and hands them to `wizard_steps_server()`. The screen is the same for every app. What differs between apps (the fields of the National Rates step and the reference-data upload zones) comes from the app's `options(cd2030.wizard = ...)`, which must be set before the screen is used (see `cd_wizard_config()`). `cd_app()` uses this screen unless the app passes its own `upload_ui` and `upload_server`.

Usage

```
cd_upload_data_ui(id, i18n, is_electron = FALSE)

cd_upload_data_server(id, i18n, cdsuite_file, active)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`is_electron`	`TRUE` when the app was started with a file to load (the desktop app), `FALSE` (the default) when the user picks the file in the page. The ui function does not use it: the server works it out from `cdsuite_file`.
`cdsuite_file`	The path of the file the app was started with, a string, or `NA_character_` when it was started without one. With a path, the file is loaded at startup and the Upload Data step shows no upload zone.
`active`	A reactive that returns `TRUE` while the Load Data screen is the page shown.

Details

The steps and their modules, in wizard order: Upload Data (`upload_box_ui()` and `reference_estimates_ui()` in one card), Data Quality (`data_quality_ui()`), Survey Files (`survey_upload_ui()`), National Rates (`national_rates_ui()`), Shapefile (`shapefile_step_ui()`), Map Survey Files (`map_survey_ui()`) and Map Shapefile Names (`map_shapefile_ui()`).

Value

`cd_upload_data_ui()` returns the UI as Shiny tags: the page header and the wizard. `cd_upload_data_server()` returns a list with two elements:

- `cache`: a reactive value that holds the dataset, a CacheConnection, or `NULL` when none is loaded;
- `requires_walkthrough`: a reactive value that is `TRUE` when the wizard must be walked through step by step, and `FALSE` when a saved dataset (`.rds`) was loaded.

Examples

```
## Not run:
# the app describes its part of the wizard once, before the screen is used
options(cd2030.app_group = "rmncah", cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))

# in the app's ui
cd_upload_data_ui("upload_data", i18n = i18n)

# in the app's server: the dataset every page reads
loaded <- cd_upload_data_server("upload_data", i18n, NA_character_,
  active = reactive(identical(input$tabs, "upload_data"))
)
cache <- loaded$cache
```

```
# cd_app() does both when it is given the pair
cd_app(app_name, app_version, theme, nav_sections, registry, i18n, language, selected_file,
  upload_ui = cd_upload_data_ui, upload_server = cd_upload_data_server
)

## End(Not run)
```

`cd_wizard_check_group` *Check that a loaded dataset belongs to the app's indicator group*

Description

Sets the package's selected indicator group back to the app's group (with `set_selected_group()`), then stops with an error when the dataset was built for a different group. The group a dataset was built for is the `indicator_group` attribute of its `countdown_data`. A dataset without that attribute, or a `NULL` dataset, passes.

Usage

```
cd_wizard_check_group(cache)
```

Arguments

`cache`	The dataset, a CacheConnection object (not a reactive), or `NULL`.

Value

`TRUE`, invisibly. The function stops with an error when the dataset was built for another indicator group.

Examples

```
old <- options(cd2030.app_group = "rmncah")
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cd_wizard_check_group(cache)

# no dataset passes too
cd_wizard_check_group(NULL)
options(old)
```

`cd_wizard_config` *The app's Load Data wizard configuration*

Description

Returns what an app has set for its own part of the Load Data wizard. The wizard is the same for every indicator group except for two things, which an app describes once with `options(cd2030.wizard = list(...))` before the wizard is used: the fields of the National Rates step and which reference-data upload zones the Upload Data step shows. The indicator group is not part of this configuration (see `cd_wizard_indicator_group()`).

Usage

```
cd_wizard_config()
```

Details

An app sets the option like this:

```
options(cd2030.wizard = list(
  national_rates_groups = list(cd_wizard_field_group(...), ...),
  reference_uploads     = c("un_estimates", "wuenic_estimates", "un_mortality_estimates")
))
```

- `national_rates_groups` (required): the groups of fields of the National Rates step, in display order. Each element is made with `cd_wizard_field_group()`, and each field in it with `cd_wizard_survey_field()` (a survey coverage estimate, in percent) or `cd_wizard_rate_field()` (a national rate, as a proportion). The wizard takes from these the inputs it shows, what it saves to and reads back from the dataset, and which values the step needs before it counts as complete: every field listed is required.
- `reference_uploads` (optional): which reference-data zones the Upload Data step shows, any of `"un_estimates"`, `"wuenic_estimates"` and `"un_mortality_estimates"`. The default is all three.

Value

The value of `getOption("cd2030.wizard")`, a list, with `reference_uploads` filled in with its default when the app did not set it. The function stops with an error when the option, or its `national_rates_groups` element, is not set.

Examples

```
old <- options(cd2030.wizard = list(
  national_rates_groups = list(
    cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
      cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
      cd_wizard_national_rate_fields()
    )
```

```
  ),
  reference_uploads = c("un_estimates", "wuenic_estimates")
))

cfg <- cd_wizard_config()
cfg$reference_uploads
length(cfg$national_rates_groups[[1]]$fields)

options(old)
```

`cd_wizard_field_group` *Describe one group of fields of the National Rates step*

Description

Puts fields under one heading for an app's wizard configuration. The National Rates step shows each group as a titled block of number fields, in the order given (see `cd_wizard_config()`).

Usage

```
cd_wizard_field_group(title_key, subtitle_key, ...)
```

Arguments

`title_key`	The translation key of the group's title, a string.
`subtitle_key`	The translation key of the group's subtitle, a string.
`...`	The fields of the group, in display order. Each argument is one field (from `cd_wizard_survey_field()` or `cd_wizard_rate_field()`) or a list of fields (such as `cd_wizard_national_rate_fields()`), whose fields are added one by one.

Value

A list with elements `title_key`, `subtitle_key` and `fields` (a flat list of the fields).

Examples

```
group <- cd_wizard_field_group("title_upload_group_immunization", "sub_upload_group_immunization",
  cd_wizard_survey_field("penta3_prop", "penta3", "title_upload_penta3_survey"),
  cd_wizard_survey_field("measles1_prop", "measles1", "title_upload_measles1_survey")
)
vapply(group$fields, function(f) f$key, character(1))

# a list of fields is added field by field
group <- cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
  cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
  cd_wizard_national_rate_fields()
)
length(group$fields)
```

cd_wizard_indicator_group

The indicator group the running app is for

Description

Returns the indicator group of the app, from `options(cd2030.app_group = ...)`. When that option is not set it falls back to `getOption("cd2030.selected_group")`. The wizard uses it to build a dataset for the app's group and to name the dataset's saved copy.

Usage

```
cd_wizard_indicator_group()
```

Value

The indicator group, as the app set it (a string such as `"rmncah"` or `"vaccine"`). The function stops with an error when neither option is set.

Examples

```
old <- options(cd2030.app_group = "rmncah")
cd_wizard_indicator_group()
options(old)
```

cd_wizard_national_rate_fields

The five national rate fields every indicator group needs

Description

Returns the descriptions of the national rate fields that every app's National Rates step has: pregnancy loss, twin rate, stillbirth rate, neonatal mortality rate and post-neonatal mortality rate. Apps share this list so that their input ids, dataset keys and labels stay the same. Pass it to `cd_wizard_field_group()`.

Usage

```
cd_wizard_national_rate_fields()
```

Value

A list of five field descriptions, each made with `cd_wizard_rate_field()`, for the `national_estimates` keys `preg_loss`, `twin_rate`, `sbr`, `nmr` and `pnmr`.

Examples

```
fields <- cd_wizard_national_rate_fields()
vapply(fields, function(f) f$key, character(1))
vapply(fields, function(f) f$id, character(1))
```

`cd_wizard_survey_field`

Describe one field of the National Rates step

Description

Each function describes one number field for an app's wizard configuration (see `cd_wizard_config()`). `cd_wizard_survey_field()` describes a survey coverage estimate, entered in percent (0 to 100, in steps of 1) and saved in the dataset's `survey_estimates`. `cd_wizard_rate_field()` describes a national rate, entered as a proportion (0 to 0.05, in steps of 0.001) and saved in the dataset's `national_estimates`. Pass the results to `cd_wizard_field_group()`.

Usage

```
cd_wizard_survey_field(id, key, label_key)

cd_wizard_rate_field(id, key, label_key)
```

Arguments

`id`	The input id of the field, a string.
`key`	The name of the value in the dataset, a string: an element of `survey_estimates` for a survey field (for example `"anc4"`), or of `national_estimates` for a rate field (`"nmr"`, `"pnmr"`, `"sbr"`, `"twin_rate"` or `"preg_loss"`).
`label_key`	The translation key of the field's label, a string.

Value

A list that describes the field, with elements `kind` (`"survey"` or `"rate"`), `id`, `key`, `label_key`, `min`, `max`, `step` and `hint` (the translation key of the hint under the field).

Examples

```
str(cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"))
str(cd_wizard_rate_field("stillbirth_rate", "sbr", "title_upload_stillbirth"))
```

`cd_years_input` *The years a years chip means*

Description

Turns the value of a years chip (see `cd_years_sync()`) into the years to use. An empty chip means "all years", and this returns every year for it, so the result can be saved in the dataset and used to filter data as it is.

Usage

```
cd_years_input(value, all_years)
```

Arguments

`value`	The chip's value (`input$years`): a character vector of years, or `""` when nothing is chosen. Entries that are empty or not a number are dropped.
`all_years`	What "all years" is: the years in the dataset.

Value

An integer vector: the years in `value`, or `all_years` (as integers) when `value` holds no year.

Examples

```
cd_years_input(c("2021", "2023"), all_years = 2019:2023)
cd_years_input("", all_years = 2019:2023)
```

`cd_years_sync` *Keep a years chip in step with the dataset*

Description

The server half of a filter chip that chooses several years. The chip itself is a multi-choice chip the page draws (`cd_chip_multi()`). This function gives the chip its options, the years in the data, and sets it to the years already chosen in the dataset. Choosing nothing in the chip means "all years".

Usage

```
cd_years_sync(input, session, id = "years", years, selected)
```

Arguments

`input`	The module's `input`.
`session`	The module's `session`.
`id`	The chip's input id within the module. Default `"years"`.
`years`	A reactive that returns the years the chip offers, the years in the data.
`selected`	A reactive that returns the years chosen in the dataset. An empty or all-`NA` value means no choice, which leaves the chip empty ("all years").

Details

Call it once inside a module server. It sends the options and the chosen years to the chip once the chip is on the page, and again when `years()` or `selected()` changes. It does not send the chip what the chip already holds.

It only goes from the dataset to the chip. The page saves the reader's choice itself, with its own `observeEvent(input$years, ...)`, where `cd_years_input()` turns the chip's value into the years it means.

Value

Nothing useful (the observer it creates). It is called for what it does.

Examples

```
## Not run:
# in a page's ui
cd_chip_multi(ns("years"), "title_global_select_years", i18n = i18n)

# in the page's server
cd_years_sync(input, session, "years",
  years = reactive(cache()$data_years),
  selected = reactive(cache()$mapping_years)
)
observeEvent(input$years, {
  cache()$set_mapping_years(cd_years_input(input$years, cache()$data_years))
})

## End(Not run)
```

`check_admin_columns` *Check that the Admin sheet has its required columns*

Description

Checks that the Admin sheet has the `country` and `first_admin_level` columns. Run it on the sheet before the sheets are merged, so that a missing column is reported by name.

Usage

```
check_admin_columns(admin_data)
```

Arguments

`admin_data`	A data frame. The cleaned Admin sheet, before the sheets are merged (the admin element of `load_excel_parts()$parts`).

Value

`NULL` if both columns are present. Otherwise a character vector of length one, named `"x"`, with a message that lists the missing columns.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
admin <- sheets$parts[[sheets$admin_sheet_name]]
check_admin_columns(admin)

# A sheet without its country column
check_admin_columns(admin[setdiff(names(admin), "country")])
```

---

check_admin_pairing_consistency

Check that each district has one admin level 1 name in the Admin sheet

---

Description

Checks that no district is listed in the Admin sheet under more than one admin level 1 unit. This is the part of `check_district_consistency()` that can run before the sheets are merged.

Usage

```
check_admin_pairing_consistency(admin_data)
```

Arguments

`admin_data`	A data frame. The cleaned Admin sheet, before the sheets are merged, with `district` and `first_admin_level` columns.

Details

Admin level 1 names are compared without regard to case or extra spaces.

Value

NULL if every district has one admin level 1 name, or if the sheet lacks either column. Otherwise a character vector of length one, named "x", with a message that lists the districts.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
admin <- sheets$parts[[sheets$admin_sheet_name]]
check_admin_pairing_consistency(admin)

# The first district listed a second time, under another region
again <- admin[1, ]
again$first_admin_level <- "Western"
check_admin_pairing_consistency(rbind(admin, again))
```

---

check_country_recognized

Check that the country in the Admin sheet is recognized

Description

Checks that the country named in the Admin sheet is one the package supports. Run it before the sheets are merged to report an unrecognized country early. new_countdown() stops with an error on the same problem.

Usage

```
check_country_recognized(parts, admin_sheet_name)
```

Arguments

parts	A named list of cleaned per-sheet tibbles (load_excel_parts()$parts).
admin_sheet_name	A string. The name of the Admin sheet within parts.

Details

The check applies only when the country column holds exactly one distinct value, ignoring missing values. More than one value is reported by check_single_country(). The name is recognized when it is close to a name in the package's list of countries.

Value

NULL if the country is recognized, if the Admin sheet has no country column, or if the column does not hold exactly one value. Otherwise a character vector of length one, named "x", with a message that gives the unrecognized name.

See Also

check_required_columns_presheet()

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_country_recognized(sheets$parts, sheets$admin_sheet_name)

# A country the package has no reference data for
parts <- sheets$parts
parts[[sheets$admin_sheet_name]]$country <- "Atlantis"
check_country_recognized(parts, sheets$admin_sheet_name)
```

---

check_district_consistency

Check that districts are consistent across the dataset

Description

Checks two things in the merged data: that each district belongs to one admin level 1 unit, and that each district appears in every year of the data. A district that fails either test has usually been renamed or misspelled in part of the workbook.

Usage

```
check_district_consistency(.data)
```

Arguments

`.data`	A data frame. The merged and standardized dataset, with `district`, `adminlevel_1` and `year` columns.

Details

Admin level 1 names are compared without regard to case or extra spaces, so `"North East"` and `"north east"` count as the same unit. The years a district must appear in are all the distinct years in `.data`.

Value

`NULL` if no problem is found. Otherwise a character vector with one or two messages, each named `"x"`: one that lists the districts with more than one admin level 1 name, and one that lists the districts missing from at least one year.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
check_district_consistency(data)

# A district spelled differently in one year
renamed <- data
renamed$district[renamed$district == "Baringo" & renamed$year == 2023] <- "Baringo County"
check_district_consistency(renamed)
```

---

check_district_cross_sheet

Check that districts match between the Admin sheet and the others

---

Description

Compares the districts in the Admin sheet with the districts in all the other sheets, in both directions. Run it on the unmerged sheets: merging keeps only the districts of one sheet, so a district that is missing from that sheet would be dropped without notice.

Usage

```
check_district_cross_sheet(parts, admin_sheet_name)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`admin_sheet_name`	A string. The name of the Admin sheet within `parts`.

Value

`NULL` if the districts match, or if the Admin sheet has no `district` column. Otherwise a character vector with one or two messages, each named `"x"`: one that lists the districts found in another sheet but not in the Admin sheet, each with the sheets it was found in, and one that lists the Admin sheet districts found in no other sheet.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_district_cross_sheet(sheets$parts, sheets$admin_sheet_name)

# A district renamed in the Admin sheet only
parts <- sheets$parts
parts[[sheets$admin_sheet_name]]$district[1] <- "Baringo County"
check_district_cross_sheet(parts, sheets$admin_sheet_name)
```

`check_district_year_completeness`
Check that every district has every year in the Population sheet

---

Description

Checks that each district in the Population sheet has a row for every year the sheet covers. This is the part of `check_district_consistency()` that can run before the sheets are merged.

Usage

```
check_district_year_completeness(population_data)
```

Arguments

`population_data`
A data frame. The cleaned Population sheet, before the sheets are merged, with `district` and `year` columns.

Value

`NULL` if every district appears in every year, or if the sheet lacks either column. Otherwise a character vector of length one, named `"x"`, with a message that lists the districts missing from at least one year.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
population <- sheets$parts[[sheets$population_sheet_name]]
check_district_year_completeness(population)

# A district with no population row for one year
gap <- population$district == "Baringo" & population$year == 2023
check_district_year_completeness(population[!gap, ])
```

`check_indicator_emptiness`
Flag indicators that have no data

Description

Finds the indicators of the selected indicator group whose column is missing in every row of the dataset. An indicator with no data cannot be analysed.

Usage

```
check_indicator_emptiness(.data)
```

Arguments

`.data`	A data frame. The merged dataset, with one column per indicator.

Details

Only indicator columns that are present in `.data` are examined. An indicator whose column is absent is not reported here.

Value

A character vector of the names of the empty indicators. It has length zero if there are none.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
check_indicator_emptiness(data)

# An indicator that was never reported
data$penta3 <- NA_real_
check_indicator_emptiness(data)
```

`check_indicator_emptiness_presheet` *Flag indicators that have no data, before merging*

Description

Runs `check_indicator_emptiness()` on the service sheets, stacked together, before the sheets are merged. An indicator is flagged when it is missing in every row of every service sheet.

Usage

```
check_indicator_emptiness_presheet(parts, service_sheet_names)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.

Value

A character vector of the names of the empty indicators. It has length zero if there are none.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_indicator_emptiness_presheet(sheets$parts, sheets$service_sheet_names)
```

---

check_month_language_consistency

Flag months that are written in more than one way

Description

Finds the calendar months that the service sheets label with more than one text, for example `"January"` in one sheet and `"Janvier"` in another. The data loads correctly either way, because the month names are standardized. The result shows that the workbook mixes languages or spellings, which is worth knowing about how it was put together.

Usage

```
check_month_language_consistency(parts, service_sheet_names)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.

Value

A tibble with one row per month that has more than one distinct original text, and columns `month` (the standardized month name) and `variants` (the distinct original texts, sorted and separated by commas). It has no rows if no month qualifies, or if the service sheets lack a `month` or `raw_month` column.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_month_language_consistency(sheets$parts, sheets$service_sheet_names)

# One service sheet that names January in French
parts <- sheets$parts
sheet <- sheets$service_sheet_names[2]
parts[[sheet]]$raw_month[parts[[sheet]]$month == "January"] <- "Janvier"
check_month_language_consistency(parts, sheets$service_sheet_names)
```

check_month_presence *Check that every district-year has all 12 months*

Description

Checks that each district has a row for every calendar month of every year it appears in. A missing month leaves a gap in the monthly series the analysis relies on.

Usage

```
check_month_presence(.data)
```

Arguments

`.data`	A data frame with `district`, `year` and `month` columns, where `month` holds English month names. This is usually the merged and standardized dataset.

Value

`NULL` if no month is missing. Otherwise a character vector of length one, named `"x"`, with a message that lists the missing months as `district year/month`. The message lists the first 10 and gives a count of the rest.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
check_month_presence(data)

# One month of one district-year dropped
gap <- data$district == "Baringo" & data$year == 2023 & data$month == "March"
check_month_presence(data[!gap, ])
```

check_month_presence_presheet *Check that every district-year has all 12 months, before merging*

Description

Runs `check_month_presence()` on the service sheets, stacked together, before the sheets are merged. A district-year passes when its 12 months are present across the service sheets taken together.

Usage

```
check_month_presence_presheet(parts, service_sheet_names)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.

Value

`NULL` if no month is missing. Otherwise a character vector of length one, named `"x"`, with a message that lists the missing months as `district year/month`. The message lists the first 10 and gives a count of the rest.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_month_presence_presheet(sheets$parts, sheets$service_sheet_names)

# March left out of every service sheet
parts <- sheets$parts
for (sheet in sheets$service_sheet_names) {
  parts[[sheet]] <- parts[[sheet]][parts[[sheet]]$month != "March", ]
}
check_month_presence_presheet(parts, sheets$service_sheet_names)
```

`check_month_validity` *Check that every month value is a recognized month*

Description

Checks the merged data for rows whose month could not be read as a calendar month, for example because of a typo or a month name in a language that is not supported. Such rows have a missing `month` and would otherwise stay in the data unnoticed.

Usage

```
check_month_validity(.data)
```

Arguments

`.data`	A data frame. The merged and standardized dataset, with a `month` column and optionally a `raw_month` column.

Details

A row is a problem when its `month` is `NA`. If `.data` has a `raw_month` column (the month text as it was in the workbook), the message lists the distinct unrecognized values. If it does not, as with a Stata master dataset, or if those values are all missing, the message gives the number of affected rows instead.

Value

`NULL` if every month is recognized, or if `.data` has no `month` column. Otherwise a character vector of length one, named `"x"`, with the message.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
check_month_validity(data)

# Months that could not be read are missing after loading
data$month[1:3] <- NA
check_month_validity(data)
```

---

check_month_validity_presheet

Check that every month value is a recognized month, before merging

---

Description

Checks the service sheets, before they are merged, for rows whose month could not be read as a calendar month, for example because of a typo or a month name in a language that is not supported.

Usage

```
check_month_validity_presheet(parts, service_sheet_names)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.

Details

A row is a problem when its `month` is `NA`. If the sheets have a `raw_month` column (the month text as it was in the workbook, which `load_excel_parts()` keeps), the message lists the distinct unrecognized values. If they do not, or if those values are all missing, the message gives the number of affected rows instead.

Value

`NULL` if every month is recognized, or if the service sheets have no `month` column. Otherwise a character vector of length one, named `"x"`, with the message.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_month_validity_presheet(sheets$parts, sheets$service_sheet_names)

# A month the workbook misspells is read as missing
parts <- sheets$parts
sheet <- sheets$service_sheet_names[1]
typo <- parts[[sheet]]$month == "March"
parts[[sheet]]$raw_month[typo] <- "Mrch"
parts[[sheet]]$month[typo] <- NA
check_month_validity_presheet(parts, sheets$service_sheet_names)
```

---

check_population_service_collision

Flag district-years where service births equal population births

---

Description

Finds district-years where the monthly institutional live births (`instlivebirths`) equal the annual population live births (`live_births`) in every month. Monthly counts normally vary, so this pattern suggests that population figures were entered in a service data sheet.

Usage

```
check_population_service_collision(.data)
```

Arguments

`.data`	A data frame. The merged dataset, with `district`, `year`, `instlivebirths` and `live_births` columns.

Details

Only months where both values are present are compared. A district-year is flagged when it has at least three such months and the two values are equal in all of them.

Value

A tibble with columns `district` and `year`, one row per flagged district-year. It has no rows if nothing is flagged or if `.data` lacks `instlivebirths` or `live_births`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
check_population_service_collision(data)

# Annual population births entered as the monthly service figure
```

```
copied <- data$district == "Baringo" & data$year == 2023
data$instlivebirths[copied] <- data$live_births[copied]
check_population_service_collision(data)
```

`check_population_service_collision_presheet`
Flag service births that equal population births, before merging

---

Description

Runs `check_population_service_collision()` on the unmerged sheets. It finds district-years where the monthly institutional live births in the service sheets equal the annual live births in the Population sheet in every month, which suggests that population figures were entered in a service data sheet.

Usage

```
check_population_service_collision_presheet(
  parts,
  population_sheet_name,
  service_sheet_names
)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`population_sheet_name`	A string. The name of the Population sheet within `parts`.
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.

Details

The service sheets are stacked and the Population sheet's `live_births` is added to each service row by `district` and `year`. Every service row is kept. A district-year is flagged when it has at least three months with both values present and the two values are equal in all of them.

Value

A tibble with columns `district` and `year`, one row per flagged district-year. It has no rows if nothing is flagged, if the Population sheet lacks `district`, `year` or `live_births`, or if the service sheets lack `instlivebirths`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_population_service_collision_presheet(
  sheets$parts, sheets$population_sheet_name, sheets$service_sheet_names
)

# Baringo's 2023 population births entered in every month of the service sheet
parts <- sheets$parts
population <- parts[[sheets$population_sheet_name]]
births <- population$live_births[population$district == "Baringo" & population$year == 2023]
service <- parts[[sheets$service_sheet_names[1]]]
service$instlivebirths[service$district == "Baringo" & service$year == 2023] <- births
parts[[sheets$service_sheet_names[1]]] <- service
check_population_service_collision_presheet(
  parts, sheets$population_sheet_name, sheets$service_sheet_names
)
```

---

check_population_vs_births

Check that population live births are not below reported births

---

Description

Compares, for each district-year, the estimated live births in the Population sheet with the institutional births reported in the service sheets. Not every birth happens in a facility, so an estimate that is lower than what facilities reported points to a problem in one of the two figures.

Usage

```
check_population_vs_births(
  parts,
  population_sheet_name,
  service_sheet_names,
  margin = 0.1
)
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`population_sheet_name`	A string. The name of the Population sheet within `parts`.
`service_sheet_names`	A character vector. The names of the service data sheets within `parts`.
`margin`	A number between 0 and 1. The fraction by which the population figure may fall short of the reported figure before it is flagged. Default is `0.10` (10%).

Details

The reported figure is the sum over the months of a district-year of `instlivebirths`, or of `instdeliveries` if the service sheets have no `instlivebirths` column. A district-year is flagged when `live_births` is less than `(1 - margin)` times that sum. Only district-years that have both figures are compared.

Value

`NULL` if no district-year is flagged, if the Population sheet lacks `district`, `year` or `live_births`, or if the service sheets have neither births column. Otherwise a character vector of length one, named `"x"`, with a message that lists the first 10 flagged district-years and gives a count of the rest. Its `"rows"` attribute is a tibble of all the flagged district-years, with columns `district`, `year`, `live_births` and `total_delivered` (the reported sum).

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
found <- check_population_vs_births(
  sheets$parts, sheets$population_sheet_name, sheets$service_sheet_names
)
found
attr(found, "rows")

# Flag any shortfall, however small
check_population_vs_births(
  sheets$parts, sheets$population_sheet_name, sheets$service_sheet_names,
  margin = 0
)
```

---

check_required_columns_presheet

Check that the sheets have the columns an indicator group needs

---

Description

Checks the unmerged sheets for every column the indicator group requires: the group's indicators, a reporting rate for each of its categories, and the population columns. It reports the same missing columns that `new_countdown()` stops on, before the sheets are merged.

Usage

```
check_required_columns_presheet(parts, group = get_selected_group())
```

Arguments

`parts`	A named list of cleaned per-sheet tibbles (`load_excel_parts()$parts`).
`group`	A string. The indicator group to check against. Default is the selected group.

Details

Only the column names are read. They are compared as they will be once the sheets are merged and standardized, so a column that standardization renames or computes is not reported as missing. The messages use the names the workbook uses.

Value

`NULL` if no required column is missing, or if the sheets have no columns. Otherwise a character vector with up to three messages, each named `"x"`: one for the population columns, one for the reporting rate columns and one for the indicator columns.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
check_required_columns_presheet(sheets$parts, group = "rmncah")

# A Population sheet without its growth rate column
parts <- sheets$parts
parts[[sheets$population_sheet_name]]$pop_growth_rate <- NULL
check_required_columns_presheet(parts, group = "rmncah")
```

---

check_shapefile_admin_names

Flag admin level 1 names with no matching shapefile region

Description

Finds the admin level 1 names in the dataset that have no close match among the regions of the shapefile. A dataset region without a shapefile match cannot be shown on a map.

Usage

```
check_shapefile_admin_names(shapefile, countdown_data, name_field = "NAME_1")
```

Arguments

`shapefile`	An `sf` object, or `NULL`. The country's shapefile: the one bundled with the package or one the user uploaded.
`countdown_data`	A data frame. The dataset, with an `adminlevel_1` column.
`name_field`	A string. The column of `shapefile` that holds the admin level 1 names. Default is `"NAME_1"`, the column the bundled shapefiles use.

Details

Names are compared without regard to case or accents, using the Jaro-Winkler distance. A name is unmatched when its distance to the closest shapefile name is 0.25 or more. Shapefile regions that match no dataset name are not reported.

Value

A tibble with one row per unmatched dataset name and columns `name` (the dataset name), `suggestion` (the closest shapefile name) and `distance` (the distance between them). It has no rows if every name matches, or if `shapefile` is `NULL` or has no `name_field` column.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

# The bundled shapefile has the counties, the dataset's regions are the former provinces
check_shapefile_admin_names(cache$shapefile, cache$countdown_data)

# With no shapefile there is nothing to compare
check_shapefile_admin_names(NULL, cache$countdown_data)
```

`check_single_country` *Check that the Admin sheet names one country*

Description

Checks that the `country` column of the Admin sheet holds no more than one distinct value. A dataset must belong to one country, so run this before the sheets are merged to get a message that lists the values found.

Usage

```
check_single_country(admin_data)
```

Arguments

`admin_data`	A data frame. The cleaned Admin sheet, with a `country` column.

Details

Missing values are ignored. A sheet with no country value at all passes this check.

Value

`NULL` if the sheet has at most one country value. Otherwise a character vector of length one, named `"x"`, with a message that lists the values.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
admin <- sheets$parts[[sheets$admin_sheet_name]]
check_single_country(admin)

# A row that belongs to another country
admin$country[1] <- "Uganda"
check_single_country(admin)
```

---

check_survey_admin_names

Flag admin level 1 names with no matching survey region

Description

Finds the admin level 1 names in the dataset that have no close match among the regions of the regional survey. A dataset region without a survey match cannot get a regional or equity estimate.

Usage

```
check_survey_admin_names(regional_survey, countdown_data)
```

Arguments

`regional_survey`
: A data frame of regional survey data with an `adminlevel_1` column, or `NULL`.

`countdown_data`
: A data frame. The dataset, with an `adminlevel_1` column.

Details

Names are compared without regard to case or accents, using the Jaro-Winkler distance. A name is unmatched when its distance to the closest survey name is 0.25 or more. Survey regions that match no dataset name are not reported.

Value

A tibble with one row per unmatched dataset name and columns `name` (the dataset name), `suggestion` (the closest survey name) and `distance` (the distance between them). It has no rows if every name matches, or if `regional_survey` is `NULL` or has no `adminlevel_1` column.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
data <- cache$countdown_data
check_survey_admin_names(cache$regional_survey, data)

# A region the survey calls something else
data$adminlevel_1[data$adminlevel_1 == "Coast"] <- "Pwani"
check_survey_admin_names(cache$regional_survey, data)
```

---

clean_error_message *Clean and validate error messages*

Description

`clean_error_message` processes and cleans error messages to remove unnecessary details such as ANSI escape codes, CLI-specific bullets, stack traces, and technical context. It ensures the input is a valid error object before processing and returns a blank string (`""`) if the input is not an error.

Usage

```
clean_error_message(error_message)
```

Arguments

`error_message`	The input to be processed. This should be an object of class `"error"`. Non-error inputs will result in a blank string being returned.

Value

A cleaned, user-friendly error message as a string. Returns an empty string if the input is not an error object.

Examples

```
# Processing a Valid Error Message
tryCatch(
  {
    stop("Something went wrong.")
  },
  error = function(e) {
    cleaned <- clean_error_message(e)
    cleaned
  }
)

# Handling Non-Error Input
cleaned <- clean_error_message("This is not an error object")
```

cleaned

---

compute_indicator_numerator
Compute aggregated numerators for health indicators

Description

compute_indicator_numerator aggregates the numerators for specified health indicators across different administrative levels, such as national, and and subnational The function groups data by year and the specified administrative level and calculates the sum of the selected indicators.

Usage

```
compute_indicator_numerator(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  region = NULL,
  show_district = TRUE
)
```

Arguments

.data	A tibble of class cd_data containing health indicator data. The dataset must include columns for the relevant indicators, which are organized by the indicator_groups attribute.
admin_level	Character. Specifies the administrative level for aggregation. Available options are: "national", "adminlevel_1" and "district". Default is "national".
region	Optional name of an adminlevel_1 region. If supplied, only that region's rows are aggregated. Only valid with admin_level = "adminlevel_1". Default is NULL.
show_district	Logical. When region is supplied, whether to aggregate by district within the region (TRUE, the default) or for the region as a whole.

Value

A tibble containing yearly aggregated totals for each indicator at the specified administrative level.

Examples

```
## Not run:
# Calculate national-level totals for health indicators
national_totals <- compute_indicator_numerator(dhis2_data, admin_level = "national")

# Calculate administrative-level 1 totals for health indicators
region_totals <- compute_indicator_numerator(dhis2_data, admin_level = "adminlevel_1")

# Calculate district-level totals for health indicators
district_totals <- compute_indicator_numerator(dhis2_data, admin_level = "district")

## End(Not run)
```

---

compute_service_utilization

Compute service utilization metrics

Description

Calculates annualized service utilization statistics including outpatient (OPD), inpatient (IPD), case fatality rates, and under-5 service use, aggregated at national, admin level 1 or district level.

Usage

```
compute_service_utilization(
  .data,
  admin_level = c("national", "adminlevel_1", "district")
)
```

Arguments

`.data`	A data frame containing service and population indicators.
`admin_level`	One of `"national"`, `"adminlevel_1"` or `"district"` to define the aggregation level.

Value

A `cd_service_utilization` object (tibble subclass) with summarized and derived indicators:

- OPD/IPD totals and rates
- Under-5 service utilization
- Case fatality rates
- Proportion of under-5 deaths

Examples

```
## Not run:
  compute_service_utilization(dat, admin_level = "national")

## End(Not run)
```

`compute_step_states` *Work out the state of every Load Data wizard step*

Description

Applies the step definitions to a dataset and returns, for each step, whether it is complete, locked, required and how it should show in the wizard's row of steps. The wizard's row of steps, its footer and the summary page all read this result.

Usage

```
compute_step_states(
  cd,
  current,
  requires_walkthrough = TRUE,
  step_defs = wizard_step_defs,
  acknowledged = character(0)
)
```

Arguments

`cd`	The dataset, a CacheConnection object (the value of `cache()`, not the reactive), or `NULL` when no dataset is loaded.
`current`	The key of the step the user is on, a string, or `NULL` when there is none.
`requires_walkthrough`	`TRUE` (the default) when the steps must be gone through in order, as for a newly loaded Excel or Stata file. `FALSE` for a finished dataset, which locks nothing.
`step_defs`	The steps, a list with the structure of wizard_step_defs, which is the default. A step may also have a `required_fn`, a function of the dataset that is used in place of its `required` value.
`acknowledged`	The keys of the steps the user has moved on from with Continue or Skip, a character vector.

Details

On a walkthrough (`requires_walkthrough = TRUE`) a step is locked when the step before it is not in `acknowledged`, or when any earlier required step is incomplete. Without a walkthrough nothing is locked.

`status` is only for display. It is `"locked"` for a locked step, otherwise `"current"` for the step named by `current`, otherwise `"complete"` when the step is complete, touched and in `acknowledged`, and otherwise `"available"`. To know whether a step's data is done, read `complete`, not `status`: the current step never has the status `"complete"`.

Value

A list with one element per step, in the order of `step_defs`. Each element is a list with:

- `key`, `title_key`: copied from the step's definition.
- `status`: `"locked"`, `"current"`, `"complete"` or `"available"` (see Details).
- `complete`: `TRUE` when the step's `complete_fn` returns `TRUE` for `cd`.
- `touched`: the result of the step's `touched_fn`, or the same as `complete` when the step has none.
- `locked`: `TRUE` when the user may not open the step yet.
- `relevant`: the result of the step's `relevant_fn`, or `TRUE` when the step has none.
- `required`: `TRUE` when the step must be complete before the user can go on.

Examples

```
old <- options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# on a walkthrough, with the first two steps passed
states <- compute_step_states(cache,
  current = "survey_files", acknowledged = c("upload", "quality")
)
vapply(states, function(s) s$status, character(1))

# a finished dataset: nothing is locked
states <- compute_step_states(cache, current = NULL, requires_walkthrough = FALSE)
vapply(states, function(s) s$complete, logical(1))

options(old)
```

`consistency_check_ui` *Consistency checks card*

Description

A card of charts that each compare two indicators, drawn by `plot_comparison()`. It has one tab for each indicator pair the app sets (the `consistency_pairs` setting, see `consistency_pair_keys()`), for example ANC1 against Penta1, and a Custom tab where the person picks any two indicators to compare. It is the second card of the Internal consistency page (`internal_consistency_ui()`). It reads `countdown_data` from the dataset.

Usage

```
consistency_check_ui(id, i18n)

consistency_check_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The card does no work before that.

Value

`consistency_check_ui()` returns the card's UI as Shiny tags. `consistency_check_server()` is called for its side effects.

Examples

```
## Not run:
# the card in a page module, as the Internal consistency page uses it
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, consistency_check_ui(ns("consistency"), i18n))
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    consistency_check_server("consistency", cache, i18n, active = active)
  })
}

## End(Not run)
```

consistency_pair_keys

Tab keys of the consistency check pairs

Description

Gives the key of each indicator pair the Consistency checks card (`consistency_check_ui()`) compares. The pairs come from the app's `consistency_pairs` setting (see `cd_cfg()`). Without that setting they are ANC1 with Penta1 and Penta1 with Penta3. The card has one tab per key, followed by its `"custom"` tab, which is not in this list.

Usage

```
consistency_pair_keys()
```

Value

A character vector with one key per pair, the two indicator codes joined by an underscore (`"anc1_penta1"`).

Examples

```
consistency_pair_keys()

# the pairs an app sets
old <- options(cd2030.config = list(consistency_pairs = list(c("anc1", "penta1"), c("opv1", "opv3"))))
consistency_pair_keys()
options(old)
```

consistency_pair_label

Tab label of a consistency check pair

Description

Gives the label of one indicator pair's tab on the Consistency checks card (`consistency_check_ui()`). The label is the pair's translation (the key `opt_consist_<x>_<y>`) when the translator has one. Otherwise it is the two indicator codes in capitals joined by "and" (`"ANC1 and PENTA1"`).

Usage

```
consistency_pair_label(i18n, pair)
```

Arguments

`i18n`	The app's translator (a `shiny.i18n` translator).
`pair`	A character vector of two indicator codes, the first for the x axis and the second for the y axis (`c("anc1", "penta1")`).

Value

The label: what `i18n$t()` gives for the pair's key when it is translated, otherwise a character string.

Examples

```
## Not run:
# i18n is the app's translator, as a page receives it
consistency_pair_label(i18n, c("anc1", "penta1"))

# the labels of every pair the app compares, named by tab key
pairs <- cd_cfg("consistency_pairs", list(c("anc1", "penta1"), c("penta1", "penta3")))
labels <- vapply(pairs, function(pair) consistency_pair_label(i18n, pair), character(1))
setNames(labels, consistency_pair_keys())

## End(Not run)
```

`countdown-pages` *The Countdown pages, wizard, filters and app frame*

Description

The Shiny pieces every Countdown app (cd2030.rmncah, cd2030.vaxx, and apps on a custom indicator group) is built from, on datasuite.ui's interface kit. This page is the guide to them: what there is and how the pieces fit. Each function has its own help page, with its arguments and what it returns, next to its code: pages in `R/page-<page id>.R` (a page's part in `R/page-<page id>--<part>.R`), the Load Data wizard in `R/wizard-*.R`, filters in `R/filter-*.R`, layout in `R/layout-*.R`, chart and table cards in `R/chart-*.R` and the app itself in `R/app-*.R`. The kit's own components are described in datasuite.ui's `docs/COMPONENTS.md`.

Details

The app. `cd_app(app_name, app_version, theme, nav_sections, registry, i18n, language, selected_file, upload_ui, upload_server)` returns the Shiny app: datasuite.ui's `app_frame()` with the Introduction and Load Data screens, the dataset header (with its denominator row, `cd_denominator_header()`) and the app's pages. An app's `run_app()` sets its options, builds its translator, lists its pages and nav tree, and calls it – see `cd2030.rmncah::run_app()` for a complete one. The standard sidebar sections are `cd_nav_start()`, `cd_nav_quality()`, `cd_nav_denominators()`, `cd_nav_national()` and `cd_nav_subnational()`.

Per-app settings. An app sets `options(cd2030.config = list(...))` once; the pages read it with `cd_cfg(key, default)`. Keys: `nat_cov_indicators`, `target_indicators`, `equity_indicators`, `cov_trend_indicators`, `sub_derived_indicators`, `survey_comp_indicators`, `adjustment_indicators`, `k_factors`, `reporting_rate_indicators`, `reporting_rate_facet_ncol`, `consistency_pairs`, `has_maternal` (see `R/app-config.R`). The indicator group is pinned with `set_selected_group()` and `options(cd2030.app_group = ...)`; the Introduction page's help comes from `options(cd2030.help_dir = ...)`.

Pages. Each is a `<stem>_ui(id, i18n)` / `<stem>_server(id, cache, i18n, ...)` pair taking the dataset (CacheConnection) as a reactive:

- data quality: `data_quality`, `reporting_rate`, `data_completeness`, `outlier_detection`, `consistency_check`, `internal_consistency`, `overall_score`, `calculate_ratios`;
- adjustment: `data_adjustment` (the years and areas removed are its first section), `adjustment_changes`;
- denominators: `denominator_assessment`, `denominator_selection`, `subnational_denominator`;
- national and sub-national analysis: `national_coverage`, `subnational_coverage`, `coverage`, `coverage_trends`, `national_inequality`, `subnational_inequality`, `inequality`, `equity`, `national_target`, `subnational_target`, `target`, `survey_comparison`, `subnational_mapping`.

The Introduction page (`introduction_ui(id, i18n)`, `introduction_server(id, selected_language)`) shows the app's help in the chosen language; `cd_app()` adds it.

The Load Data wizard. `cd_upload_data_ui()` / `cd_upload_data_server()` are the Load Data screen (`cd_app()`'s default): the page header and the wizard, the same for every app, which describes its own part of it with `options(cd2030.wizard = list(national_rates_groups, reference_uploads))` (see `R/wizard-config.R`). `upload_box_*`, `wizard_steps_*` and `wizard_landing_*` make the screen; its steps are `wizard_step_defs` (upload, data quality, survey files, national rates, shapefile, survey and map mappings), each with a `step_*_complete()` test, and `compute_step_states()` works out which are done, blocked or optional. The steps' pages are `national_rates_*`, `survey_upload_*`, `reference_estimates_*`, `shapefile_step_*`, `map_survey_*` and `map_shapefile_*`; `cd_wizard_*` build their fields.

Filters and building blocks. `cd_admin_level_*`, `cd_indicator_*`, `cd_denominator_*`, `cd_population_*`, `cd_years_input()` / `cd_years_sync()`, the scoped page (`cd_scope()`, `cd_scoped_page_*`, `cd_scope_filters()`), tabbed charts (`cd_tabbed_charts_*`), tables (`cd_table_*`), a flextable in a card with its picture and Excel downloads (`cd_table_card_*`) and the coverage chart card (`cd_coverage_plot_*`).

See Also

`cd_app()` to build an app; `reporting_rate_ui()` for a page; `cd_upload_data_ui()` for the Load Data wizard; `cd_table_ui()`, `cd_scope()` and `cd_admin_level_ui()` for building blocks.

Examples

```
## Not run:
# an app's run_app(): its options, its translator, its pages and nav tree, then cd_app()
options(cd2030.selected_group = "rmncah", cd2030.app_group = "rmncah")
set_selected_group("rmncah")
```

```
options(cd2030.config = list(
  reporting_rate_indicators = c("opt_anc" = "anc_rr", "opt_vacc" = "vacc_rr"),
  has_maternal = TRUE
))
options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))

i18n <- shiny.i18n::init_i18n(translation_json_path = cd_translations("translation.json"))
i18n$set_translation_language("en")
cd_use_i18n(i18n)

pages <- list(
  cd_page_def(
    id = "reporting_rate", ui = reporting_rate_ui, server = reporting_rate_server,
    title = "title_rr_main", section = "lbl_nav_section_quality", subtitle = "sub_rr_main",
    help = c("apps/countdown/data-quality", "reporting-rate")
  )
)
cd_use_pages(pages)

cd_app(
  app_name = "My Countdown app", app_version = "1.0.0", theme = "rmncah",
  nav_sections = list(cd_nav_start(), cd_nav_quality()),
  registry = pages, i18n = i18n, language = "en", selected_file = NA,
  upload_ui = cd_upload_data_ui, upload_server = cd_upload_data_server
)

## End(Not run)
```

`coverage_trends_ui` *Coverage trends by denominator card*

Description

A card of the Denominator selection page (`denominator_selection_ui()`) with one tab per indicator (the app's `cov_trend_indicators` setting). Each tab has a chart of the indicator's coverage over the years, with one line per denominator the app offers, for the country or for the chosen region. The card has no filters of its own: the page passes it the admin level and the region. It calls the dataset's `calculate_derived_coverage()`, and draws nothing until the dataset's `check_inequality_params` is `TRUE`.

Usage

```
coverage_trends_ui(id, i18n)
```

```
coverage_trends_server(
  id,
  cache,
  admin_level,
  region,
  i18n,
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A reactive that returns the admin level of the chart: `"national"`, `"adminlevel_1"` or `"district"`.
`region`	A reactive that returns the name of the chosen region, or `NULL` when none is chosen. For an admin level other than national, the chart is drawn only once a region is chosen.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The card does no work before that.

Value

`coverage_trends_ui()` returns the card's UI as Shiny tags. `coverage_trends_server()` is called for its side effects.

Examples

```
## Not run:
# the card in a page that has an admin level filter, as the Denominator selection page does
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n,
    filters = cd_filter_bar(cd_admin_level_ui(ns("admin_level"), i18n, include_national = TRUE)),
    coverage_trends_ui(ns("coverage"), i18n)
  )
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    admin <- cd_admin_parts(cd_admin_level_server("admin_level", cache, i18n))
    coverage_trends_server("coverage", cache, admin$admin_level, admin$region, i18n, active = active)
  })
}

## End(Not run)
```

`coverage_ui` *Coverage charts for one geography*

Description

The body of the national and sub-national coverage pages: one card of charts with a tab for each indicator the app lists (`cd_cfg("nat_cov_indicators")`) and a Custom tab for any other analysis indicator. A tab plots the indicator's coverage by year from the health facility data, with the denominator selected for that indicator, beside the survey and WUENIC estimates. It reads the dataset's `get_filtered_coverage()`, which takes the indicator's rows from `calculate_coverage()` for the admin level and region. The tab `"fpet"`, when an app lists it, shows the family planning estimates instead (`cache()$fpet_data`, from 2010 onwards). The piece has no filters of its own: the page gives it the admin level and region.

Usage

```
coverage_ui(id, i18n, key_label)

coverage_server(
  id,
  cache,
  i18n,
  admin_level,
  region = reactive(NULL),
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`key_label`	The translation key of the card's title.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A reactive that returns the level the coverage is worked out at: `"national"`, `"adminlevel_1"` (admin level 1) or `"district"`.
`region`	A reactive that returns the name of the region (or district) to show, or `NULL` for none.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The charts do no work before that.

Value

`coverage_ui()` returns the card's UI as Shiny tags. `coverage_server()` returns a reactive that holds the key of the tab shown: an indicator's name, or `"custom"`.

Examples

```
## Not run:
# the charts in a page module, at national level
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, coverage_ui(ns("body"), i18n, "title_coverage_national"))
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    coverage_server("body", cache, i18n, reactive("national"), active = active)
  })
}

# for one region: the admin level and the region are reactives
coverage_server("body", cache, i18n, reactive("adminlevel_1"), reactive("Nairobi"), active = active)

## End(Not run)
```

---

create_mortality_ratios

Extract latest mortality ratios for UN comparison

Description

Joins computed national mortality indicators with UN estimates, returns latest value per indicator.

Usage

```
create_mortality_ratios(.data, mortality_data)
```

Arguments

`.data`	The main mortality dataset.
`mortality_data`	A `cd_un_mortality` object.

Value

A tibble of class `cd_mortality_ratio` with latest available values and UN means.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

mortality <- create_mortality_summary(cache$countdown_data)
ratios <- create_mortality_ratios(mortality, cache$un_mortality_estimates)
```

```
ratios

# the institutional ratio against the UN estimate, by assumed completeness
summarise_completeness_ratio(ratios, plot_type = "mmr")
```

`create_mortality_summary` *Create mortality summary object*

Description

Combines national and admin level 1 summaries and formats as a custom class for plotting.

Usage

```
create_mortality_summary(.data)
```

Arguments

`.data`	An object of class `cd_data`.

Value

A `cd_mortality_summary` tibble.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

mortality <- create_mortality_summary(data)
mortality

# the national rates
mortality[mortality$adminlevel_1 == "National", c("year", "mmr_inst", "sbr_inst", "nn_inst")]
```

`data_adjustment_ui` *Data adjustment page*

Description

The page where the person sets which data is kept for analysis and how it is corrected. Years can be removed everywhere, and an area's data can be removed for some or every year. The corrections are for completeness (by each indicator group's k factor), outliers and missing values, set for everywhere or with a region's or district's own settings. The page is one editor (`datasuite.ui::cd_adjustment_editor()`), which shows beside each setting what the check pages found (`adjustment_evidence()`). Below it are a button that downloads the adjusted data as a Stata file and a link to the Data adjustment changes page (`adjustment_changes_ui()`).

Usage

```
data_adjustment_ui(id, i18n)

data_adjustment_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	Not used. It is accepted so that the app's page registry can pass it as it does to every other page.

Details

The Adjust data button saves the settings in the dataset (`set_adjustment_settings()`) and adjusts it (`adjust_data()`). The page is then read-only until the person chooses Edit. The page reads `countdown_data`, `performance_threshold`, `adjustment_settings`, `adjusted_flag` and `adjusted_data` from the dataset.

Value

`data_adjustment_ui()` returns the page's UI as Shiny tags. `data_adjustment_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "data_adjustment",
  ui = data_adjustment_ui,
```

```
  server = data_adjustment_server,
  title = "title_adjust_main",
  section = "lbl_nav_section_quality",
  subtitle = "sub_adjust_main"
)

## End(Not run)
```

`data_completeness_ui` *Data completeness page*

Description

The data quality page on missing values. Its first card has three tabs: a heat map of completeness by admin level 1 area or district, a chart of district completeness for the chosen indicator, and a line chart of the indicator's completeness over the years. Below it is a table of the units with missing values in the year picked on the table. The filters are the indicator, which can also be set to all indicators, and the admin level with its region. With all indicators chosen, the table lists the missing units of every indicator, with a column that names the indicator.

Usage

```
data_completeness_ui(id, i18n)

data_completeness_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Details

The page reads `completeness_admin1`, `completeness_district` and `district_completeness` from the dataset, and calls its `list_missing_units()`, and its `calculate_completeness_summary()` and `calculate_district_completeness_summary()` when a region is chosen. The heat map's Excel download writes `completeness_national` and `district_completeness`.

Value

`data_completeness_ui()` returns the page's UI as Shiny tags. `data_completeness_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "data_completeness",
  ui = data_completeness_ui,
  server = data_completeness_server,
  title = "title_complete_main",
  section = "lbl_nav_section_quality",
  subtitle = "sub_complete_main"
)

## End(Not run)
```

---

data_quality_ui *Data quality step of the Load Data wizard*

Description

Shows the result of every data quality check on the loaded dataset as a list of rows, each marked as passed or as a warning, with an explanation in a tooltip. The checks come from `run_all_quality_checks()`. The user enters nothing here and nothing is saved to the dataset. A button in the card's toolbar downloads every issue found as an Excel workbook (`write_quality_issues()`). The step counts as complete when every check of severity `"blocking"` has passed (see `step_quality_complete()`).

Usage

```
data_quality_ui(id, i18n)

data_quality_server(id, cache, i18n)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.

Details

The rows are in three parts:

- the blocking checks: the file and its sheets (always shown as passed, because the wizard cannot reach this step when the file could not be read), the admin sheet's columns, the required columns, a single country, a recognised country, districts across sheets, district consistency, and missing or invalid months;

- the informational checks, which never block: population against reported births, reported live births equal to the population's live births in every month, indicators with no data, and month names written in more than one way;
- the mapping checks, which never block: admin level 1 names that do not match the survey or the shapefile.

A row is left out when the check did not run. A dataset loaded already merged (a Stata file, or a saved copy without stored results) only has the checks that run on merged data.

One check is not part of this step: that every indicator the group needs is present. It runs when the user presses Finish, because it reads columns that only exist after the sheets are merged.

Value

`data_quality_ui()` returns the UI as Shiny tags: a card that holds the list of checks and the download button. `data_quality_server()` is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "quality", ui = data_quality_ui(ns("data_quality"), i18n))
data_quality_server("data_quality", cache, i18n)

## End(Not run)
```

`den_indicators` *Tabs of the denominator assessment chart*

Description

The keys of the tabs on the denominator assessment page (`denominator_assessment_ui()`): total population, births and children under one.

Usage

```
den_indicators
```

Format

A character vector.

Examples

```
den_indicators
```

`denominator_assessment_ui` *Denominator assessment page*

Description

The page that compares the population figures in the health data (DHIS2) with the United Nations estimates over the years. One card has three tabs, each with a chart: total population, births (the DHIS2 live births and total births) and children under one. The one filter is the population chip (`cd_population_ui()`), which shows the dataset's `derivation_population` and saves a change to it there. The charts read `denominator_metrics` from the dataset.

Usage

```
denominator_assessment_ui(id, i18n)

denominator_assessment_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

`denominator_assessment_ui()` returns the page's UI as Shiny tags. `denominator_assessment_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "denominator_assessment",
  ui = denominator_assessment_ui,
  server = denominator_assessment_server,
  title = "title_denom_assessment",
  section = "lbl_nav_section_denominators",
  subtitle = "sub_denom_assessment"
)

## End(Not run)
```

`denominator_selection_ui` *Denominator selection page*

Description

The page where the person compares the candidate denominators and chooses the ones the analysis uses. It has three cards: coverage from each denominator beside the survey estimate (`survey_comparison_ui()`), coverage by admin level 1 area (`subnational_denominator_ui()`) and coverage over the years (`coverage_trends_ui()`). The filters are the denominator for the vaccination indicators, the denominator for the maternal indicators (only in an app that has them, see `cd_has_maternal()`) and the admin level, which includes national, with its region. The denominator filters are also where the choice is made (`cd_denominator_ui()`): they show the dataset's `denominator` and `maternal_denominator`, and a change is saved there.

Usage

```
denominator_selection_ui(id, i18n)

denominator_selection_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page's three cards do no work before that.

Value

`denominator_selection_ui()` returns the page's UI as Shiny tags. `denominator_selection_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "denominator_selection",
  ui = denominator_selection_ui,
  server = denominator_selection_server,
  title = "title_denom_selection",
  section = "lbl_nav_section_denominators",
  subtitle = "sub_denom_selection"
)
```

```
## End(Not run)
```

`detect_indicator_group` *Auto-detect the best-matching indicator group from a vector of indicators*

Description

Given a character vector of indicator codes present in the dataset, selects the group whose **entire universe** is present in the data (strict coverage: `group  data`). If multiple groups are fully covered, the tie is broken by choosing the **largest** universe (most specific), then alphabetically.

Usage

```
detect_indicator_group(indicators)
```

Arguments

`indicators`	Character vector of indicator codes present in the data.

Details

Detection considers the merged view from `.get_all_groups()` (i.e., built-ins plus any user overrides registered via `register_indicator_group()`).

Value

The detected group name (character scalar).

Examples

```
## Not run:
inds <- c("anc1","penta1","penta3","measles1","measles2","bcg")
detect_indicator_group(inds)

## End(Not run)
```

`equiplot` *Create dot plots for equity analysis*

Description

`equiplot` generates a dot plot to visualize the distribution of variables across different groups (e.g., countries, regions, or interventions). It is designed for equity analysis by plotting values for variables like health intervention coverage across subgroups, allowing insights into disparities.

Usage

```
equiplot(
  .data,
  variables,
  group_by,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  legend_title = NULL,
  legend_labels = NULL,
  reverse_y_axis = FALSE,
  connect_dots = TRUE,
  dot_size = NULL
)
```

Arguments

`.data`	A data frame with one row per group and one column per variable (values in percent, plotted on a 0-100 scale).
`variables`	Character vector of the column names to plot as dots; each becomes a coloured series.
`group_by`	The column (unquoted or as a string) whose values form the rows of the plot (y axis).
`title`, `subtitle`, `caption`	Optional plot title, subtitle and caption. Each must be a single string or `NULL`.
`x_title`	Optional x-axis title. Defaults to a percentage label.
`legend_title`	Optional title of the colour legend.
`legend_labels`	Optional named vector or list mapping variable names to display labels (for example translated labels).
`reverse_y_axis`	Logical. Whether to reverse the order of the groups on the y axis. Default is `FALSE`.

connect_dots	Logical. Whether to draw a line connecting the dots of each group. Default is TRUE.
dot_size	Optional dot size, clamped to 1-5. Defaults to 4.

Value

A ggplot object.

Examples

```
coverage <- data.frame(
  region = c("North", "Centre", "South"),
  poorest = c(42, 55, 61),
  richest = c(78, 83, 90)
)
equiplot(coverage, variables = c("poorest", "richest"), group_by = region)

equiplot(coverage, c("poorest", "richest"), group_by = region,
         title = "Skilled birth attendance by wealth", legend_title = "Wealth quintile",
         legend_labels = c(poorest = "Poorest", richest = "Richest"), connect_dots = FALSE)
```

---

equiplot_area *A specialized dot plot for area of residence analysis*

Description

Plots rural and urban survey estimates of an indicator by year.

Usage

```
equiplot_area(
  .data,
  indicator,
  dot_size = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

.data	Equity survey data with year, level and r_<indicator> columns.
indicator	Character. The indicator to plot; its r_<indicator> column is used.
dot_size	Optional dot size, clamped to 1-5. Defaults to 4.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	Optional named vector or list mapping group names to display labels.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object: the dot plot drawn by equiplot(), with one row per survey year and one dot each for rural and urban, and the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the dataset object gives the package's survey estimates for the country, by residence
equiplot_area(cache$area_survey, "anc4")

equiplot_area(cache$area_survey, "anc4", title = "Four antenatal visits by residence",
              legend_title = "Residence", legend_labels = list(Rural = "Rural areas", Urban = "Towns"))
```

---

equiplot_education *A specialized dot plot for maternal education analysis*

Description

Plots survey estimates of an indicator by year for each level of maternal education.

Usage

```
equiplot_education(
  .data,
  indicator,
  dot_size = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`.data`	Equity survey data with `year`, `level` and `r_<indicator>` columns.
`indicator`	Character. The indicator to plot; its `r_<indicator>` column is used.
`dot_size`	Optional dot size, clamped to 1-5. Defaults to 4.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional named vector or list mapping group names to display labels.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object: the dot plot drawn by `equiplot()`, with one row per survey year and one dot for each level of maternal education, and the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the dataset object gives the package's survey estimates for the country, by maternal education
equiplot_education(cache$education_survey, "penta3")
```

```
equiplot_education(cache$education_survey, "penta3", title = "Penta3 by education of the mother",
                   x_title = "Penta3 coverage (%)", dot_size = 3, legend_position = "bottom")
```

`equiplot_wealth` *A specialized dot plot for wealth quintile analysis*

Description

Plots survey estimates of an indicator by year for each wealth quintile.

Usage

```
equiplot_wealth(
  .data,
  indicator,
  dot_size = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`.data`	Equity survey data with `year`, `level` and `r_<indicator>` columns.
`indicator`	Character. The indicator to plot; its `r_<indicator>` column is used.
`dot_size`	Optional dot size, clamped to 1-5. Defaults to 4.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional named vector or list mapping group names to display labels.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object: the dot plot drawn by equiplot(), with one row per survey year and one dot for each wealth quintile (Q1 to Q5), and the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the dataset object gives the package's survey estimates for the country, by wealth quintile
equiplot_wealth(cache$wiq_survey, "instlivebirths")

equiplot_wealth(cache$wiq_survey, "instlivebirths", title = "Institutional live births by wealth",
                legend_title = "Quintile", caption = "Source: household surveys")
```

---

equity_ui *Equity assessment page*

Description

Shows how survey coverage of an indicator differs between groups of the population, as an equiplot for each survey. One filter chooses the grouping: place of residence (urban and rural), mother's education or wealth quintile; the page reads the matching survey table of the dataset (cache()$area_survey, cache()$education_survey or cache()$wiq_survey). The charts sit in one card with a tab for each indicator the app lists (cd_cfg("equity_indicators")) and a Custom tab, where any analysis indicator can be picked except those in cd_cfg("equity_custom_exclude").

Usage

```
equity_ui(id, i18n)

equity_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

id	The module's id. A page's id is also its entry in the app's page registry.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.
active	A reactive that is TRUE once the page has been opened with data loaded. The page does no work before that.

Value

equity_ui() returns the page's UI as Shiny tags. equity_server() returns a reactive that holds the key of the tab shown: an indicator's name, or "custom".

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "equity_assessment",
  ui = equity_ui,
  server = equity_server,
  title = "title_nav_equity",
  section = "title_nav_national_analysis",
  subtitle = "sub_equity"
)

## End(Not run)
```

---

filter_high_performers

Filter high-performing areas by coverage

Description

Filters a dataset to retain administrative areas with coverage values above a specified threshold for a given indicator and denominator.

Usage

```
filter_high_performers(
  .data,
  indicator,
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived"),
  threshold = 90
)
```

Arguments

`.data`	An object of class `cd_indicator_coverage`.
`indicator`	A string specifying the indicator.
`denominator`	A string. The denominator used in coverage calculation.
`threshold`	Numeric. Minimum coverage (in percent, compared after rounding) an area must reach to be kept. Default is `90`.

Value

A filtered data frame retaining regions meeting the threshold.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
coverage <- cache$indicator_coverage_admin1

filter_high_performers(coverage, indicator = "instlivebirths", denominator = "dhis2")
filter_high_performers(coverage, indicator = "instlivebirths", denominator = "dhis2", threshold = 100)
```

---

filter_out_years *Filter dataset by removing specified years*

Description

The `filter_out_years` function is used to exclude specified years from a DHIS-2 dataset, refining the data for more targeted analysis or reporting. This is particularly useful when focusing on specific timeframes, such as recent years only or a custom-defined analysis period. By removing data from unwanted years, you can create a tailored dataset aligned with analytical goals.

Usage

```
filter_out_years(.data, years_to_exclude)
```

Arguments

`.data`	A `cd_data` object, which contains a dataset with DHIS-2 indicators and demographic data organized by year.
`years_to_exclude`	An integer vector that specifies the years to exclude from the dataset.

Details

- **Purpose**: Use this function when specific years are unnecessary for analysis, such as baseline years or years with incomplete data.
- **Validation**:
  - Ensures `.data` is a valid `cd_data` object for DHIS-2 analysis.
  - Confirms `years_to_exclude` is a vector of integers; raises an error if not properly specified.
- **Functionality**: Filters `.data` to retain only the rows where the `year` does not match any value in `years_to_exclude`.

Value

A `cd_data` object with rows filtered to exclude those matching the specified years in `years_to_exclude`. The resulting data set is intended for focused analysis across the remaining years.

Examples

```
## Not run:
# Example: Remove data for 2017 and 2018 from a DHIS-2 dataset
filtered_data <- filter_out_years(dhis_data, years_to_exclude = c(2017, 2018))

## End(Not run)
```

---

generate_adjustment_values

Generate adjusted and unadjusted service counts by year

Description

`generate_adjustment_values` calculates yearly unadjusted and adjusted service counts for each indicator within a `cd_data` dataset, applying the specified adjustment type. It provides a comparison of raw and adjusted values for analysis purposes.

Usage

```
generate_adjustment_values(
  .data,
  adjustment = c("default", "custom", "none"),
  k_factors = NULL,
  settings = NULL,
  area = NULL,
  level = c("adminlevel_1", "district")
)
```

Arguments

`.data`	A `cd_data` dataframe containing service data for adjustments.
`adjustment`	A character string specifying the adjustment type: • `"default"`: Applies preset k-factors (e.g., 0.25) for each indicator group. • `"custom"`: Uses user-specified `k_factors` values for each indicator group. • `"none"`: Skips adjustments, returning only the raw data.
`k_factors`	A named numeric vector of custom k-factor values between 0 and 1 for each indicator group (e.g., `c(anc = 0.3, idelv = 0.2, ...)`). Required if `adjustment = "custom"`.

`settings` The adjustment settings (`adjust_service_data()`): when given, the counts after each step are given too (`_completeness`, `_outliers`), and the reported counts are those of the data kept (the removed years and areas left out).

`area`, `level` One area only: a region (`level = "adminlevel_1"`) or a district (`"district"`); `NULL`, all.

Details

Performs the following steps:

1. **Data Validation**: Ensures `.data` is of the `cd_data` class and `adjustment` is correctly specified.
2. **Unadjusted Summation**: Calculates the yearly sums of unadjusted service counts.
3. **Adjusted Summation**: Applies `adjust_service_data()` to compute adjusted values, then calculates the yearly sums.
4. **Combining Results**: Merges unadjusted and adjusted yearly counts for comparison.

Value

A `cd_adjustment_values` tibble containing:

- Columns for unadjusted values, suffixed with `_raw`.
- Columns for adjusted values, suffixed with `_adj`.
- A `year` column indicating the year of each count.

See Also

`adjust_service_data()` for the detailed adjustment function.

Examples

```
## Not run:
# Generate adjustment values with default k-factors
adjustment_values_default <- generate_adjustment_values(data, adjustment = "default")

# Generate adjustment values with custom k-factors
custom_k <- c(anc = 0.3, idelv = 0.2, pnc = 0.35, vacc = 0.4, opd = 0.3, ipd = 0.25)
adjustment_values_custom <-
  generate_adjustment_values(data, adjustment = "custom", k_factors = custom_k)

# Generate unadjusted values only
unadjusted_values <- generate_adjustment_values(data, adjustment = "none")

## End(Not run)
```

`generate_admin1_keys` *Generate a key for each admin level 1 unit*

Description

Gives each distinct admin level 1 name a short key, so that the same unit can be identified in several tables (for example the dataset, the survey mapping and the map mapping). Generate the keys once from the dataset's names, then join them to the other tables by name.

Usage

```
generate_admin1_keys(admin1_names)
```

Arguments

`admin1_names`	A character vector of admin level 1 names, usually the `adminlevel_1` column of the dataset.

Details

The keys are arbitrary: the distinct names are sorted and numbered in that order. They are not standard codes, and they are not stable between datasets. A dataset with a different set of names can give a different key to the same unit. Missing names get no key.

Value

A tibble with one row per distinct name, sorted by name, and columns `adminlevel_1` (the name) and `admin1_key` (the key: `"A1-001"`, `"A1-002"` and so on).

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
generate_admin1_keys(data$adminlevel_1)

# Repeated and missing names get no key of their own
generate_admin1_keys(c("Western", "Coast", "Western", NA))
```

generate_admin1_mch_curative_index

Generate MCH vs curative index data

Description

Keeps the latest year of subnational service utilization data and its two service indices: the MCH preventive services index and the curative services index. The result is ready for `plot.cd_mch_curative_index()`.

Usage

```
generate_admin1_mch_curative_index(service_utilization)
```

Arguments

`service_utilization`
: Dataframe containing admin level 1 service utilization.

Value

A tibble with class `cd_mch_curative_index`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
utilization <- compute_service_utilization(data, admin_level = "adminlevel_1")

index <- generate_admin1_mch_curative_index(utilization)
index
plot(index)
```

generate_admin1_service_utilization

Generate service utilization admin level 1 data

Description

Keeps the latest year of subnational service utilization data and the under-five utilization rate of one service: OPD visits per child (`ratio_opd_u5_pop`) or IPD admissions per 100 children (`ratio_ipd_u5_pop`). The result is ready for `plot.cd_service_utilization_admin1()`.

Usage

```
generate_admin1_service_utilization(
  service_utilization,
  metric_type = c("opd", "ipd")
)
```

Arguments

`service_utilization`	Dataframe containing admin level 1 service utilization.
`metric_type`	Character. Either "opd" or "ipd".

Value

A tibble with class `cd_service_utilization_admin1` and the columns `adminlevel_1`, `year` and the selected rate. The attribute `metric_type` holds `metric_type`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
utilization <- compute_service_utilization(data, admin_level = "adminlevel_1")

# outpatient visits per child under five, by region, in the latest year
generate_admin1_service_utilization(utilization, metric_type = "opd")

# inpatient admissions per 100 children under five
admissions <- generate_admin1_service_utilization(utilization, metric_type = "ipd")
plot(admissions)
```

`generate_bayes_model` *Generate Bayesian coverage model object*

Description

Fits a Bayesian coverage model for one indicator, combining the survey estimates and the routine (DHIS2) coverage in `coverage_data` with the yearly data quality score in `overall_score`. The model is national or sub-national (admin level 1), following the admin level of `coverage_data`.

Usage

```
generate_bayes_model(
  coverage_data,
  overall_score,
  indicator = c("anc4", "anc_1trimester", "ideliv", "measles1", "penta3"),
  denominator = c("anc1", "dhis2", "penta1", "penta1derived", "anc1derived"),
  keep_samples = FALSE
)
```

Arguments

`coverage_data`	Survey data frame of class 'cd_coverage'.
`overall_score`	DHIS2 data frame of class 'cd_overall_score'.
`indicator`	Indicator character code (e.g., 'anc4', 'penta3').
`denominator`	The denominator type to use from DHIS2 (e.g., 'penta1', 'dhis2').
`keep_samples`	Whether the model keeps the sampler's draws and the compiled Stan model (`samples`, `stan_model`). Default `FALSE`: the charts and `bayes_model_estimates()` read the summary of the draws, which is kept either way, and a model without the draws is much smaller (tens to hundreds of KB instead of tens of MB), so it is quicker to save and to pass between processes.

Value

An object of class `cd_bayes_model`: the fitted model returned by `bayescoveragedeploy::fit_local_model()` (a list, with its `posteriors` summary), without `samples` and `stan_model` unless `keep_samples = TRUE`. It carries the attributes `indicator`, `model_indicator` (the name the model uses for the indicator), `iso` and `is_national`.

Examples

```
## Not run:
# needs the Bayesian model packages (see cd_bayes_packages_missing()); a fit takes minutes
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

model <- generate_bayes_model(
  coverage_data = cache$calculate_coverage("national"),
  overall_score = cache$overall_score,
  indicator = "penta3",
  denominator = "penta1"
)
bayes_model_estimates(model)
plot(model)

## End(Not run)
```

---

generate_coverage_data

Generate coverage data for continuum of care

Description

Processes and shapes raw coverage data into a clean, long format optimized for continuum-of-care plotting. It dynamically builds regex patterns based on the spatial level and data type to extract relevant columns:

- **National**: Extracts Facility, Survey, and WUENIC data.
- **Subnational**: Extracts Facility data only.

The function automatically filters for the absolute latest available data point per indicator-source combination and converts the `indicator` column into an ordered factor representing the lifecycle stages to ensure plots are always ordered correctly.

Usage

```
generate_coverage_data(
  .data,
  type = c("maternal", "child"),
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived")
)
```

Arguments

`.data`	A data frame/tibble originating from `calculate_coverage()`. Must contain an `admin_level` attribute.
`type`	Character. Specifies whether to extract 'maternal' or 'child' indicators.
`denominator`	Character. The denominator to use for the specified indicator type. Options include "dhis2", "anc1", "penta1", "penta1derived", or "anc1derived".

Value

A tibble of class `cd_coverage_selected` containing the latest coverage values, categorized sources, and ordered indicators. Retains `admin_level` and `admin_col` attributes for downstream plotting.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
coverage <- cache$calculate_coverage("national")

generate_coverage_data(coverage, type = "maternal", denominator = "anc1")
generate_coverage_data(coverage, type = "child", denominator = "penta1")
```

generate_fpet_summary

Load and generate FPET output

Description

Processes raw FPET data (e.g., from Track20 or UNFPA) into a structured tibble for visualization and interpretation of modern contraceptive prevalence and demand satisfied.

Usage

```
generate_fpet_summary(.data)
```

Arguments

`.data`	An object of class `cd_fpet_data`.

Value

A `cd_fpet_data` object.

Examples

```
## Not run:
# an FPET results file (.csv) for the country
fpet <- load_fpet_data(path = "fpet_kenya.csv", country_iso = "KEN")

summary <- generate_fpet_summary(fpet)
plot(summary)
interpret(summary)

## End(Not run)
```

generate_health_system_table

Generate health system metrics data

Description

Arranges national health system metrics as a table of eight indicators in three sections: health infrastructure, health workforce and the role of the private sector. Each row has the indicator's label, its value and its unit, ready for `plot.cd_health_system_table()`.

Usage

```
generate_health_system_table(metric_data, labels = NULL)
```

Arguments

`metric_data`	A `cd_health_system_metric` object for the national level, as returned by `calculate_health_system_metrics()` with `admin_level = "national"`.
`labels`	(Optional) A nested list of localized labels for sections, indicators, and units (elements `section`, `indicator` and `unit`). English labels are used for whatever it does not set.

Value

A tibble of class `cd_health_system_table` with one row per indicator and the columns `section`, `indicator`, `value` and `unit` (`NA` where the indicator has no unit).

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
metrics <- calculate_health_system_metrics(data, admin_level = "national")

generate_health_system_table(metrics)

# labels in another language: what is not given stays in English
generate_health_system_table(metrics, labels = list(section = list(workforce = "Personnel de sante")))
```

---

generate_phc_scatter_data

Generate PHC performance scatter data

Description

Pairs a health system input (facility density or health workforce density) with a service coverage index for each admin level 1 unit, for `plot.cd_phc_scatter()`. The service coverage index is the mean of the MCH preventive services index and the curative services index.

Usage

```
generate_phc_scatter_data(
  .data,
  x_indicator = c("ratio_fac_pop", "ratio_hstaff_pop")
)
```

Arguments

`.data`	A `cd_health_system_metric` object, as returned by `calculate_health_system_metrics()` with `admin_level = "adminlevel_1"`.
`x_indicator`	Independent variable ('ratio_fac_pop' or 'ratio_hstaff_pop').

Value

A tibble of class `cd_phc_scatter` with one row per admin level 1 unit and the columns `adminlevel_1`, the column named by `x_indicator`, `service_coverage` and `is_outlier` (`TRUE` when `service_coverage` is below the first quartile or above the third quartile). The attribute `indicator` holds `x_indicator`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
metrics <- calculate_health_system_metrics(data, admin_level = "adminlevel_1")

generate_phc_scatter_data(metrics, x_indicator = "ratio_fac_pop")

# against the density of health workers
workforce <- generate_phc_scatter_data(metrics, x_indicator = "ratio_hstaff_pop")
plot(workforce)
```

---

generate_private_sector_data

Generate private sector ownership data

Description

Reshapes health system metrics into the share of health facilities that are private, NGO and public in each admin level 1 unit, for `plot.cd_private_sector()`. The public share is what remains after the private and NGO shares.

Usage

```
generate_private_sector_data(.data, legend_labels = NULL)
```

Arguments

`.data`	Data frame containing subnational health indicators, with the columns `adminlevel_1`, `private_facility_share` and `ngo_facility_share`, such as the output of `calculate_health_system_metrics()` with `admin_level = "adminlevel_1"`.
`legend_labels`	(Optional) A named list to override default legend labels (elements `Private`, `NGO` and `Public`).

Value

A tibble of class `cd_private_sector` with one row per admin level 1 unit and facility type, and the columns `adminlevel_1`, `facility_type` (a factor ordered private, NGO, public, with the labels as its levels) and `share` (percent). The attributes `lbl_prv`, `lbl_ngo` and `lbl_pub` hold the labels used.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
metrics <- calculate_health_system_metrics(data, admin_level = "adminlevel_1")

generate_private_sector_data(metrics)

# with another label in the legend
shares <- generate_private_sector_data(metrics, legend_labels = list(Private = "For profit"))
plot(shares)
```

---

generate_service_dqa_summary

Generate service DQA summary

Description

Combines the reporting rate, completeness, outlier and service utilization results for outpatient (OPD) and inpatient (IPD) services into one data quality table, with a row per indicator and a column per year. The rows are numbered and grouped under three section headers, ready for `plot.cd_utilization_dqa()`.

Usage

```
generate_service_dqa_summary(
  average_reporting_rate,
  district_reporting_rate,
  completeness_national,
  district_completeness,
  outliers_summary,
  district_outliers_summary,
  service_utilization,
  threshold = .cd_method$data_quality$reporting_threshold,
  labels = NULL
)
```

Arguments

`average_reporting_rate`
: Dataframe. Reporting rate data.

`district_reporting_rate`
: Dataframe. District reporting rate data.

`completeness_national`
: Dataframe. National completeness data.

`district_completeness`
: Dataframe. District completeness data.

`outliers_summary`
: Dataframe. National outliers data.

`district_outliers_summary`
: Dataframe. District outliers summary data.

`service_utilization`
: Dataframe. Output of compute_service_utilization('national').

`threshold`
: Numeric. The performance threshold (in percent) shown in the label. Default is `90`.

`labels`
: List. Optional custom labels for sections and indicators.

Value

A wide-format tibble with summarized DQA indicators grouped by section.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

generate_service_dqa_summary(
  average_reporting_rate = calculate_average_reporting_rate(data),
  district_reporting_rate = calculate_district_reporting_rate(data),
  completeness_national = calculate_completeness_summary(data),
  district_completeness = calculate_district_completeness_summary(data),
  outliers_summary = calculate_outliers_summary(data),
  district_outliers_summary = calculate_district_outlier_summary(data),
  service_utilization = compute_service_utilization(data, admin_level = "national")
)
```

`get_admin_columns` *Determine grouping columns for administrative levels*

Description

Returns the appropriate grouping column(s) based on the specified administrative level. If a region is specified and `admin_level = "adminlevel_1"`, districts are included to allow further disaggregation within that region.

Usage

```
get_admin_columns(admin_level, region = NULL, show_district = TRUE)
```

Arguments

`admin_level`
: A string. One of `"national"`, `"adminlevel_1"`, or `"district"`.

`region`
: Optional. If provided, must be used only with `admin_level = "adminlevel_1"`.

`show_district`
: Optional. If show_district is true `adminlevel_1` will include `district` column

Value

A character vector of column names to use in grouping. Returns:

- `NULL` for `"national"`
- `"adminlevel_1"` or `c("adminlevel_1", "district")` for `"adminlevel_1"`
- `c("adminlevel_1", "district")` for `"district"`

Examples

```
get_admin_columns("national")
#> NULL

get_admin_columns("adminlevel_1")
#> "adminlevel_1"

get_admin_columns("adminlevel_1", region = "Eastern")
#> c("adminlevel_1", "district")

get_admin_columns("district")
#> c("adminlevel_1", "district")
```

---

get_all_indicators *Get All Indicators*

Description

Flatten all indicators from all groups

Usage

```
get_all_indicators()
```

Value

Character vector of all indicators

Examples

```
get_all_indicators()

# the indicators of another group (the selected group is put back afterwards)
old <- get_selected_group()
set_selected_group("vaccine")
get_all_indicators()
set_selected_group(old)
```

get_analysis_indicators *Get Indicators excluding indicators without denominator*

Description

Flatten all indicators from all groups

Usage

```
get_analysis_indicators()
```

Value

Character vector of all indicators

Examples

```
get_analysis_indicators()

# what the selected group has and the analysis leaves out
setdiff(get_all_indicators(), get_analysis_indicators())
```

get_country_iso3 *Get country ISO3 code*

Description

Reads the ISO3 code of the country a Countdown dataset is for. The code is stored with the data when it is loaded.

Usage

```
get_country_iso3(.data)
```

Arguments

.data	A cd_data object

Value

The ISO3 country code as a character string

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

get_country_iso3(data)
```

---

get_country_name *Get country name*

Description

Reads the name of the country a Countdown dataset is for. The name is stored with the data when it is loaded.

Usage

```
get_country_name(.data)
```

Arguments

.data	A cd_data object

Value

The country name as a character string

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

get_country_name(data)
```

---

get_country_shapefile *Load country shapefile for mapping*

Description

Loads the admin level 1 boundaries of a country from the shapefiles that come with the package. Ensures geometries are valid and standardized to WGS84 (EPSG:4326).

Usage

```
get_country_shapefile(country_iso, level = c("admin_level_1", "district"))
```

Arguments

`country_iso`	A string with the ISO3 code of the country (e.g., `"KEN"`).
`level`	A character string specifying the admin level: `"admin_level_1"` (default) or `"district"`. The package has admin level 1 boundaries only, so both values give the admin level 1 areas.

Value

An `sf` object containing valid geometries of the country's admin level 1 areas.

Examples

```
shapefile <- get_country_shapefile("KEN", level = "admin_level_1")
```

---

get_coverage_indicators

Get Indicators for coverage calculation

Description

Flatten all indicators from all groups

Usage

```
get_coverage_indicators()
```

Value

Character vector of all indicators

Examples

```
get_coverage_indicators()

# the vaccine group calculates coverage for every indicator it has
old <- get_selected_group()
set_selected_group("vaccine")
setequal(get_coverage_indicators(), get_all_indicators())
set_selected_group(old)
```

`get_excel_version` *Format service utilization data for Excel export*

Description

Transforms a `cd_service_utilization` object into a wide-format table with labeled indicators, suitable for export to Excel.

Usage

```
get_excel_version(.data)
```

Arguments

`.data`	A `cd_service_utilization` object produced by `compute_service_utilization()`.

Details

The output table includes population, OPD/IPD service utilization, under-5 service metrics, reporting completeness, and case fatality rates, reshaped to facilitate Excel review or reporting.

Value

A tibble in wide format with:

- One row per indicator per admin unit
- One column per year
- A human-readable label for each indicator (`indiclabel`)

Examples

```
## Not run:
  x <- compute_service_utilization(dat, admin_level = "adminlevel_1")
  get_excel_version(x)

## End(Not run)
```

get_indicator_group_names *Get indicator group names*

Description

Lists the names of the categories (such as `anc`, `idelv` or `vacc`) that the indicators of the selected indicator group are arranged in. These are the names of `get_indicator_groups()`.

Usage

```
get_indicator_group_names()
```

Value

Character vector of indicator group names

Examples

```
get_indicator_group_names()

# the indicators of one of them
get_indicator_groups()[["vacc"]]
```

get_indicator_groups *Get the current indicator group definition*

Description

Returns the **merged** definition for a group (built-in overlaid with any override). If `group` is omitted, uses `get_selected_group()`.

Usage

```
get_indicator_groups(group = get_selected_group())
```

Arguments

`group`	Group name. Defaults to the globally selected group.

Value

A named list of categories -> character vector of indicators.

Examples

```
get_indicator_groups("rmncah")
```

get_mapping_data *Get mapping data for subnational coverage analysis*

Description

Merges coverage data with a country's shapefile to enable spatial visualization of health indicators. It joins subnational (admin level 1) indicator coverage that has already been calculated with the geographic features of each unit.

Usage

```
get_mapping_data(.data, subnational_map = NULL)
```

Arguments

`.data`	A `cd_population` object (for example the output of `calculate_indicator_coverage()`) calculated at `admin_level = "adminlevel_1"`. Must include metadata attributes `iso3` (ISO3 country code) and `admin_level`.
`subnational_map`	Optional. A data frame to join with the shapefile to add metadata.

Value

A `tibble` of class `"cd_mapping"` that combines the input data with shapefile geometries.

Examples

```
## Not run:
coverage <- calculate_indicator_coverage(data, admin_level = "adminlevel_1")
get_mapping_data(coverage)

## End(Not run)
```

get_named_indicators *Get Named Indicator Vector*

Description

Each indicator is named by its group

Usage

```
get_named_indicators()
```

Value

Named character vector of indicators

Examples

```
indicators <- get_named_indicators()
head(indicators)

# the indicators of one category
unname(indicators[names(indicators) == "vacc"])
```

---

get_national_rates *Get the rates the denominators use from survey data*

Description

Takes the most recent survey value of ANC1 coverage, Penta1 coverage and the stillbirth, neonatal and post-neonatal mortality rates, and converts each to a proportion. A rate the survey data does not have is filled with the value given in the matching argument. The result has the rates that `calculate_indicator_coverage()` takes as arguments.

Usage

```
get_national_rates(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  anc1survey = .cd_method$denominators$anc1survey,
  dpt1survey = .cd_method$denominators$dpt1survey,
  sbr = .cd_method$denominators$sbr,
  nmr = .cd_method$denominators$nmr,
  pnmr = .cd_method$denominators$pnmr,
  twin = .cd_method$denominators$twin,
  preg_loss = .cd_method$denominators$preg_loss,
  subnational_map = NULL
)
```

Arguments

`.data`	A `cd_survey_data` object with `year` and `r_*` columns (`r_anc1` and `r_penta1` in percent, and optionally `r_sbr`, `r_nmr` and `r_pnmr` per 1,000). National survey data for `admin_level = "national"`, regional survey data (with an `adminlevel_1` column) otherwise.
`admin_level`	Character. The administrative level of the survey data: `"national"`, `"adminlevel_1"` or `"district"`.
`anc1survey`	Numeric. ANC1 coverage (a proportion) to use where the survey has no value.

`dpt1survey` Numeric. Penta1 coverage (a proportion) to use where the survey has no value.

`sbr` Numeric. Stillbirth rate (a proportion) to use where the survey has no value.

`nmr` Numeric. Neonatal mortality rate (a proportion) to use where the survey has no value.

`pnmr` Numeric. Post-neonatal mortality rate (a proportion) to use where the survey has no value.

`twin` Numeric. Twin birth rate (a proportion). It is not read from the survey data, so this value is always the one returned.

`preg_loss` Numeric. Pregnancy loss rate (a proportion). It is not read from the survey data, so this value is always the one returned.

`subnational_map` Optional. A data frame mapping survey regions to `adminlevel_1` units, used for subnational levels. Default is `NULL`.

Value

A tibble with one row for national data, or one row per unit for subnational data, and the columns `pnmr`, `nmr`, `sbr`, `preg_loss`, `twin`, `anc1survey` and `dpt1survey`, all as proportions. Subnational results also have the admin column or columns.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
rates <- get_national_rates(cache$national_survey)
rates

# A stillbirth rate of your own, used because the survey has none
get_national_rates(cache$national_survey, sbr = 0.015)$sbr

# The rates of each survey region
regional <- get_national_rates(cache$regional_survey, admin_level = "adminlevel_1")
regional
```

`get_plot_admin_column` *Determine the administrative column to use for plotting*

Description

Returns the appropriate administrative unit column to use on plots based on the selected administrative level and whether a region is specified. Used to decide whether to facet or label plots by district or by adminlevel_1.

Usage

```
get_plot_admin_column(admin_level, region = NULL)
```

Arguments

admin_level	A string. Either "adminlevel_1" or "district".
region	Optional. A region name or NULL. If specified, districts will be used for plotting.

Value

A string: either "district" or "adminlevel_1".

Examples

```
get_plot_admin_column("adminlevel_1")
#> "adminlevel_1"

get_plot_admin_column("adminlevel_1", region = "Nairobi")
#> "district"

get_plot_admin_column("district")
#> "district"
```

---

get_population_column

Get the population column for an indicator and denominator

Description

Names the column that holds the target population of an indicator for a denominator option. The name is the indicator's target population followed by the denominator, for example "totinftpenta_penta1" for penta3 with the penta1 denominator.

Usage

```
get_population_column(indicator, denominator)
```

Arguments

indicator	Character scalar. Indicator name (e.g., 'penta1', 'measles1')
denominator	Character scalar. The denominator for the indicator: one of "dhis2", "anc1", "penta1", "penta1derived" or "anc1derived".

Value

A string naming the corresponding population column, or NA when the indicator has no target population defined.

Examples

```
get_population_column("penta3", "penta1")
get_population_column("anc4", "dhis2")

# births are total births with the DHIS2 denominator, live births with the others
get_population_column("bcg", "dhis2")
get_population_column("bcg", "anc1")
```

---

get_selected_group *Get the globally selected indicator group*

Description

Returns the last group set via set_selected_group(). If none is set for the session, falls back to getOption("cd2030.selected_group", "rmncah") and caches it.

Usage

```
get_selected_group()
```

Value

A length-1 character vector (group name).

Examples

```
get_selected_group()
```

---

inequality_ui *Sub-national inequality charts*

Description

The body of the inequality pages: one card of charts with a tab for each of the app's default indicators (cd_default_indicator_set()) and a Custom tab for any other analysis indicator. A tab plots, by year, the coverage of every sub-national unit beside the reference coverage held in the inequality table, with the denominator selected for the indicator. It reads the dataset's get_filtered_inequality(): cache()$inequality_admin1 for admin level 1 with no region, and cache()$inequality_district otherwise, limited to the region's districts when a region is given. Both tables measure units against the national value; when a region is given the legend names the reference after the region. The piece has no filters of its own: the page gives it the admin level and region.

Usage

```
inequality_ui(id, i18n, title_key = "title_inequ_national")

inequality_server(
  id,
  cache,
  i18n,
  admin_level,
  region = reactive(NULL),
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`title_key`	The translation key of the card's title. By default the national page's.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A reactive that returns the level of the units compared: `"adminlevel_1"` (admin level 1) or `"district"`.
`region`	A reactive that returns the name of an admin level 1 region, to compare its districts, or `NULL` to compare every unit of the country. A region is accepted only with `"adminlevel_1"`.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The charts do no work before that.

Value

`inequality_ui()` returns the card's UI as Shiny tags. `inequality_server()` returns a reactive that holds the key of the tab shown: an indicator's name, or `"custom"`. The national inequality page uses it to show the same tab on its maps.

Examples

```
## Not run:
# the charts in a page module, for the admin level 1 areas of the country
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, inequality_ui(ns("inequality"), i18n, title_key = "title_inequ_national"))
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    shown <- inequality_server("inequality", cache, i18n, reactive("adminlevel_1"), active = active)
    observe(message("Tab shown: ", shown()))
  })
}

## End(Not run)
```

`init_CacheConnection` *Create a CacheConnection object*

Description

`init_CacheConnection` initializes a `CacheConnection` either from a provided `.rds` file path or directly from a `cd_data` object. Only one of the arguments should be non-NULL.

Usage

```
init_CacheConnection(
  rds_path = NULL,
  countdown_data = NULL,
  data_path = NULL,
  wizard_parts = NULL,
  indicator_group = c("auto", "rmncah", "vaccine", "custom"),
  read_only = FALSE
)
```

Arguments

`rds_path`	Optional character. Path to an RDS file to load the cache from.
`countdown_data`	Optional `cd_data` object to initialize in-memory cache.
`data_path`	Optional character. Path to the original source file (e.g. the `.xlsx` `countdown_data` was parsed from). The cache is kept beside it, as a `.rds` file with the same name.
`wizard_parts`	Optional `load_excel_parts()` result: the sheets of an upload that the Load Data wizard has read but not yet merged. Use it instead of `rds_path` or `countdown_data`.
`indicator_group`	Character. Specifies the indicator group to use (auto, rmncah, vaccine, custom).
`read_only`	Logical. `TRUE` never writes the cache back to its `.rds`: for reading a dataset another process (the app) owns, e.g. the AI's copy.

Value

An instance of the `CacheConnection` class.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)

# in memory: no file is written
cache <- init_CacheConnection(countdown_data = data)
```

```
cache$country
cache$data_years

## Not run:
# the dataset an app saved, opened without ever writing to it
cache <- init_CacheConnection(rds_path = "kenya_rmncah.rds", read_only = TRUE)

## End(Not run)
```

`internal_consistency_ui` *Internal consistency page*

Description

The data quality page on how well related indicators agree with each other. It has two cards and no filters: the ratio chart (`calculate_ratios_ui()`) and the consistency checks, which compare pairs of indicators (`consistency_check_ui()`). Those two pieces read `ratios_summary` and `countdown_data` from the dataset.

Usage

```
internal_consistency_ui(id, i18n)

internal_consistency_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

`internal_consistency_ui()` returns the page's UI as Shiny tags. `internal_consistency_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "internal_consistency",
  ui = internal_consistency_ui,
  server = internal_consistency_server,
```

```
  title = "title_consist_main",
  section = "lbl_nav_section_quality",
  subtitle = "sub_consist_main"
)

## End(Not run)
```

`interpret` *Generic interpretation method*

Description

A generic method to extract narrative interpretation from structured data objects.

Usage

```
interpret(x, ...)
```

Arguments

`x`	An object.
`...`	Additional arguments passed to methods.

Value

What the method for the class of `x` returns. The method for `cd_fpet_data` (`interpret.cd_fpet_data()`) returns a character string.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

# the family planning estimates that come with the dataset
interpret(cache$fpet_data)
```

interpret.cd_fpet_data

Interpret FPET results with emphasis on current estimates

Description

Summarizes 2020 and 2024 estimates, and briefly mentions FPET projections to 2030.

Usage

```
## S3 method for class 'cd_fpet_data'
interpret(x, ...)
```

Arguments

x	A cd_fpet_data object.
...	Unused.

Value

A character string with interpretation summary.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)

text <- interpret(cache$fpet_data)
cat(text)
```

introduction_ui

Introduction page

Description

The app's opening screen: one card that shows the app's introduction text in the chosen language. The text is the Markdown file 0_intro_<language>.md in the app's help folder, which an app sets with options(cd2030.help_dir =) (by default "help"). When there is no file for the language, the English one (0_intro_en.md) is shown. The page reads nothing from the dataset.

Usage

```
introduction_ui(id, i18n)

introduction_server(id, selected_language)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for the card's title.
`selected_language`	A reactive that returns the code of the language picked in the app (for example `"en"`). While it does not hold one non-empty string, the page uses the language the app's translator is set to, and English if that cannot be read.

Value

`introduction_ui()` returns the page's UI as Shiny tags. `introduction_server()` is called for its side effects.

Examples

```
## Not run:
# the folder that holds 0_intro_en.md, 0_intro_fr.md and so on
options(cd2030.help_dir = system.file("intro", package = "cd2030.rmncah"))

# in the app's UI, and in its server function
introduction_ui("introduction", i18n = i18n)
introduction_server("introduction", selected_language = reactive(input$selected_language))

## End(Not run)
```

`is_maternal_indicator` *Detemine if an indicator is maternal indicator*

Description

Default grouping of indicators used in CD2030 coverage framework

Usage

```
is_maternal_indicator(indicator)
```

Arguments

`indicator`	Character. Indicator name(s) to check, for example `"anc1"`. Always `FALSE` unless the selected group is `"rmncah"`.

Value

TRUE if and indicator is maternal

Examples

```
is_maternal_indicator("anc4")
is_maternal_indicator(c("anc4", "ideliv", "penta3"))
```

---

list_indicator_groups

List available indicator group names

Description

Returns all **current** groups (built-ins + any overrides).

Usage

```
list_indicator_groups()
```

Value

A sorted character vector of group names.

Examples

```
list_indicator_groups()
```

---

list_missing_units

List the districts and months with a missing value

Description

Lists every district and month in which the value of one indicator is missing. Use it to see where the gaps behind a completeness summary are.

Usage

```
list_missing_units(.data, indicator, region = NULL)
```

Arguments

`.data`	A `cd_data` object.
`indicator`	Character. The indicator to check: one of the indicators of the dataset (for example `"penta3"`).
`region`	Optional name of an `adminlevel_1` region. If supplied, only the districts of that region are listed. Default is `NULL`.

Value

A tibble of class `cd_missing_list` with one row per district and month where the indicator is missing, and the columns `adminlevel_1`, `district`, `year` and `month`. The attribute `indicator` holds the indicator.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
data <- load_data(path)
list_missing_units(data, "measles2")

# Within one region
list_missing_units(data, "measles2", region = "Rift Valley")
```

`list_outlier_units` *Identify monthly outliers by district*

Description

Flags extreme monthly values for every indicator at district level using the Hampel method.

Usage

```
list_outlier_units(.data)
```

Arguments

`.data`	A `cd_data` object with monthly data.

Details

- Aggregates by `year`, `month`, and district.
- Computes median and MAD (Median Absolute Deviation).
- Flags outliers when values are more than 5 MAD from the median.

Value

A `cd_outlier_list` tibble with:

- Grouping columns
- Raw values for each indicator
- Median (`<indicator>_med`)
- MAD (`<indicator>_mad`)
- Outlier flag (`<indicator>_outlier5std`)

Examples

```
## Not run:
list_outlier_units(cd_data)

## End(Not run)
```

---

list_tracer_vaccines *List tracer vaccines*

Description

Returns a subset of vaccines used in tracer metrics

Usage

```
list_tracer_vaccines()
```

Value

A character vector of tracer vaccines

Examples

```
list_tracer_vaccines()

# those the selected indicator group has
intersect(list_tracer_vaccines(), get_all_indicators())
```

---

list_vaccine_indicators *List all vaccine and related coverage indicators*

Description

Returns a vector of standard vaccine and related indicators used in coverage and dropout analysis. This includes routine immunizations, tracer indicators, and derived dropout/coverage metrics commonly used in DHIS2-based health reporting.

Usage

```
list_vaccine_indicators()
```

Value

A character vector of vaccine indicator names

Examples

```
list_vaccine_indicators()
```

`load_cache_data` *Initialize or load a cached Countdown 2030 connection*

Description

Loads data via `load_data()` for Excel/Stata inputs, or initializes a cache from an `.rds`. Returns a cache/connection object.

Usage

```
load_cache_data(
  path,
  indicator_group = c("auto", "vaccine", "rmncah", "custom"),
  profile = NULL,
  on_conflict = c("replace", "merge", "error"),
  create_cache = FALSE,
  start_year = NULL,
  admin_sheet_name = NULL,
  population_sheet_name = NULL,
  reporting_sheet_name = NULL,
  service_sheet_names = NULL,
  validate = TRUE
)
```

Arguments

`path`	Path to `.xlsx`, `.xls`, `.dta`, or `.rds`.
`indicator_group`	One of `"auto"`, `"vaccine"`, `"rmncah"`, `"custom"`. Default `"auto"`.
`profile`	Optional. Either a **name** (string) of an existing group, or a **definition** to register inline before loading: • explicit form: `list(name = "rmncah", value = list(hiv = c("hiv_test","pmtct1")))` • compact form: `list(hiv_profile = list(testing = c("hiv_test"), care = c("art_new")))` For `"custom"`, the profile must resolve to **one** concrete name.
`on_conflict`	Conflict policy when `profile` is a definition: one of `"replace"`, `"merge"`, or `"error"`. Default `"replace"`. `"merge"` unions indicators per category.
`create_cache`	Logical. For Excel/Stata inputs, whether to pass `path` to the connection as its data path so a cache file is created or loaded next to it. Default is `FALSE` (no cache file). Ignored for `.rds` inputs.
`start_year`	Optional integer filter for minimum year.

`admin_sheet_name`
: Excel sheet with admin data. Default `"Admin_data"`.

`population_sheet_name`
: Excel sheet with population data. Default `"Population_data"`.

`reporting_sheet_name`
: Excel sheet with reporting completeness. Default `"Reporting_completeness"`.

`service_sheet_names`
: Character vector of service-data sheets. Default: all sheets matching `"Service_data"` if not supplied.

`validate`
: Logical. Whether to stop with an error when the merged data fails a data quality check: `check_district_consistency()`, `check_month_presence()` or `check_month_validity()`. Default is `TRUE`. With `FALSE` a file that has such problems still loads, so the caller can run the checks later with `run_all_quality_checks()`. Structural checks, such as a missing Admin sheet column, always run.

Value

A cache/connection object as returned by `init_CacheConnection()`.

Examples

```
## Not run:
# From Excel
con <- load_cache_data("data/country.xlsx", indicator_group = "auto")

# From Stata
con <- load_cache_data("data/country.dta", indicator_group = "rmncah")

# From pre-saved RDS
con <- load_cache_data("data/cd_data.rds", indicator_group = "rmncah")

## End(Not run)
```

---

load_csection_estimates

Load C-section share estimates

Description

Loads c-section correction estimates for national or area-level adjustment.

Usage

```
load_csection_estimates(
  path = NULL,
  .data = NULL,
```

```
  country_iso = NULL,
  level = c("national", "area")
)
```

Arguments

`path`	Optional. File path to a `.dta` file. Required if `.data` is not provided.
`.data`	Optional. A pre-loaded dataset. Used directly if provided.
`country_iso`	ISO3 country code. When the data holds several countries (an `iso` column with more than one code), the rows of this one are kept; data for one country is used as it is.
`level`	Either `"national"` or `"area"` indicating the data level.

Details

- Filters data for the specified `country_iso`.
- Validates whether the data level matches `"national"` or `"area"`.
- For `"area"` level, parses the `level` column to extract `area`.
- Returns only relevant columns for merging (`iso`, `year`, `indic`, and either `national` or `area_est`).

Value

A tibble of class `"cd_csection_estimates"` with a `"level"` attribute.

See Also

`load_private_sector_data()`, `prepare_private_sector_plot_data()`

Examples

```
## Not run:
cs <- load_csection_estimates(
  path = "csection_national.dta",
  country_iso = "KEN",
  level = "national"
)

## End(Not run)
```

`load_data` *Load Countdown 2030 data (Excel or Stata)*

Description

Loads a cleaned dataset from an Excel (.xlsx/.xls) or Stata (.dta) file, optionally **registers** a profile override, and returns a tibble of class `cd_data`. Group selection is resolved from the data inside `new_countdown()`:

- `"auto"` detects the best-matching group (built-ins + any overrides)
- `"vaccine"` or `"rmncah"` select built-ins (overrides still apply)
- `"custom"` uses `profile` (must exist after any inline registration)

Usage

```
load_data(
  path,
  indicator_group = c("auto", "vaccine", "rmncah", "custom"),
  profile = NULL,
  on_conflict = c("replace", "merge", "error"),
  start_year = NULL,
  admin_sheet_name = NULL,
  population_sheet_name = NULL,
  reporting_sheet_name = NULL,
  service_sheet_names = NULL,
  validate = TRUE
)
```

Arguments

`path` Path to `.xlsx`, `.xls`, or `.dta`.

`indicator_group` One of `"auto"`, `"vaccine"`, `"rmncah"`, `"custom"`. Default `"auto"`.

`profile` Optional. Either a **name** (string) of an existing group, or a **definition** to register inline before loading:

- explicit form: `list(name = "rmncah", value = list(hiv = c("hiv_test","pmtct1")))`
- compact form: `list(hiv_profile = list(testing = c("hiv_test"), care = c("art_new")))`

For `"custom"`, the profile must resolve to **one** concrete name.

`on_conflict` Conflict policy when `profile` is a definition: one of `"replace"`, `"merge"`, or `"error"`. Default `"replace"`. `"merge"` unions indicators per category.

`start_year` Optional integer filter for minimum year.

`admin_sheet_name` Excel sheet with admin data. Default `"Admin_data"`.

`population_sheet_name`
: Excel sheet with population data. Default `"Population_data"`.

`reporting_sheet_name`
: Excel sheet with reporting completeness. Default `"Reporting_completeness"`.

`service_sheet_names`
: Character vector of service-data sheets. Default: all sheets matching `"Service_data"` if not supplied.

`validate`
: Logical. Whether to stop with an error when the merged data fails a data quality check: `check_district_consistency()`, `check_month_presence()` or `check_month_validity()`. Default is `TRUE`. With `FALSE` a file that has such problems still loads, so the caller can run the checks later with `run_all_quality_checks()`. Structural checks, such as a missing Admin sheet column, always run.

Details

The actual **group resolution** occurs in `new_countdown()`, using indicators **from the loaded data**. If `profile` is a definition, it is registered via `register_indicator_group()` **before** loading, so auto-detection considers it.

Value

A tibble of class `cd_data`.

Examples

```
## Not run:
# Built-in group (explicit)
cd <- load_data("data/country.dta", indicator_group = "rmncah")
attr(cd, "indicator_group")  # "rmncah"

# Auto-detect among built-ins
cd <- load_data("data/country.xlsx", indicator_group = "auto")

# Inline extend rmncah, then auto-detect (merge by category)
cd <- load_data(
  "data/country.xlsx",
  indicator_group = "auto",
  profile = list(name = "rmncah", value = list(hiv = c("hiv_test","pmtct1"))),
  on_conflict = "merge"
)

# Define a brand-new custom profile and select it
cd <- load_data(
  "data/country.xlsx",
  indicator_group = "custom",
  profile = list(my_profile = list(anc = c("anc1","anc4")))
)

## End(Not run)
```

load_equity_data *Load and filter equity data*

Description

Loads survey equity data, removes unnecessary variables, and filters by country.

Usage

```
load_equity_data(path = NULL, .data = NULL, country_iso)
```

Arguments

path	Optional. File path to a .dta file.
.data	Optional. A preloaded data frame.
country_iso	Character. ISO3 country code to filter by.

Value

A tibble of class cd_equity_data.

Examples

```
## Not run:
wealth <- load_equity_data(path = "wiq_data.dta", country_iso = "KEN")
cache$set_wiq_survey(wealth)

# The same for the survey by area of residence and by mother's education
cache$set_area_survey(load_equity_data(path = "area_data.dta", country_iso = "KEN"))
cache$set_education_survey(load_equity_data(path = "meduc_data.dta", country_iso = "KEN"))

## End(Not run)
```

load_excel_parts *Read and clean the sheets of a Countdown 2030 workbook*

Description

Reads and cleans each sheet of a Countdown 2030 Excel workbook and returns the sheets separately, without merging them. Use it to run data quality checks on each sheet before the merge, then pass the result to merge_and_standardize() to finish loading.

Usage

```
load_excel_parts(
  path,
  start_year = NULL,
  admin_sheet_name = NULL,
  population_sheet_name = NULL,
  reporting_sheet_name = NULL,
  service_sheet_names = NULL
)
```

Arguments

`path`	Path to an `.xlsx` or `.xls` file.
`start_year`	Optional integer filter for minimum year.
`admin_sheet_name`	Excel sheet with admin data. Default `"Admin_data"`.
`population_sheet_name`	Excel sheet with population data. Default `"Population_data"`.
`reporting_sheet_name`	Excel sheet with reporting completeness. Default `"Reporting_completeness"`.
`service_sheet_names`	Character vector of service-data sheets. Default: all sheets matching `"Service_data"` if not supplied.

Details

Month names are standardized to English on every sheet that has a `month` column, and the original text is kept in a `raw_month` column. The sheets can then be joined on `month` even if they name the months in different languages.

The function stops with an error if a named sheet is not in the workbook, if a sheet is empty, or if a sheet lacks its key columns (`district` for the Admin sheet, `district` and `year` for the Population sheet, and `district`, `year` and `month` for the others). Other problems with the data are left to the data quality checks.

Value

A list with these elements:

- `parts`: a named list of cleaned tibbles, one per sheet, named by sheet.
- `sheet_names`: a character vector of the sheet names, in the order the sheets are merged.
- `sheet_ids`: a named list of the key columns of the Admin sheet, the Population sheet and (as `service_data`) every other sheet.
- `admin_sheet_name`, `population_sheet_name`, `reporting_sheet_name`, `service_sheet_names`: the sheet names used, with defaults filled in.
- `path`, `start_year`: the arguments as given.

See Also

merge_and_standardize(), run_all_quality_checks()

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
names(sheets$parts)
sheets$service_sheet_names
sheets$parts[[sheets$population_sheet_name]]

# Month names are in English, the workbook's own text is kept beside them
unique(sheets$parts[[sheets$service_sheet_names[1]]][c("month", "raw_month")])
```

---

load_fpet_data *Load FPET data from CSV file or use provided data*

Description

load_fpet_data loads Family Planning Estimation Tool (FPET) data from a .csv file, or uses a pre-supplied dataset. It assigns a country ISO code if specified, and returns a tibble with a custom class cd_fpet_data.

Usage

```
load_fpet_data(path = NULL, .data = NULL, country_iso = NULL)
```

Arguments

path	Optional character. File path to a .csv file. If provided, data is read from the file.
.data	Optional data.frame or tibble. Used directly if provided instead of loading from file.
country_iso	Optional character. ISO3 country code. Used to tag or filter the dataset.

Value

A tibble with class cd_fpet_data. If country_iso is provided, the data is filtered by iso3 == country_iso.

Examples

```
## Not run:
# Load FPET data from file
fpet <- load_fpet_data(path = "path/to/fpet.csv", country_iso = "NGA")

# Use existing data
fpet <- load_fpet_data(.data = some_fpet_data, country_iso = "ETH")
```

```
## End(Not run)
```

`load_private_sector_data`
Load and prepare private sector data

Description

Loads raw prevalence data for public/private sector analysis and applies internal preprocessing for c-section share calculation.

Usage

```
load_private_sector_data(
  path = NULL,
  .data = NULL,
  country_iso = NULL,
  level = c("national", "area")
)
```

Arguments

`path`	Optional. File path to a `.dta` file. Required if `.data` is not provided.
`.data`	Optional. A pre-loaded dataset. Used directly if provided.
`country_iso`	ISO3 country code. When the data holds several countries (an `iso` column with more than one code), the rows of this one are kept; data for one country is used as it is.
`level`	Either `"national"` or `"area"` indicating the data level.

Details

Loads the data from `path`, or uses `.data`, and checks that it matches `level`: area data has a `level` column and national data does not. Then:

- Assigns `"Public"` or `"Private"` to the `sector` column based on `indic` suffix.
- Removes sector suffix from `indic`.
- Computes number of c-sections (`num_csection`) and sector share of c-sections (`share_csection`) within each country and survey year, and within each value of the `level` column for area data.
- If level is `"area"`, parses `level` string into `area = "Urban"` or `"Rural"`.

Value

A tibble of class `"cd_private_sector_data"` with a `"level"` attribute.

Output columns

- `indic`: cleaned indicator name (without "pub"/"priv")
- `sector`: `"Public"` or `"Private"`
- `share_csection`: share of c-sections attributable to each sector
- `area`: (only if `level == "area"`)

See Also

`load_csection_estimates()`, `prepare_private_sector_plot_data()`

Examples

```
## Not run:
dt <- load_private_sector_data(
  path = "National estimates.dta",
  country_iso = "KEN",
  level = "national"
)

## End(Not run)
```

---

load_survey_data *Load and process survey data*

Description

Loads national or subnational survey data, cleans labels and codes, and applies country-specific adjustments for admin-level naming.

Usage

```
load_survey_data(
  path = NULL,
  .data = NULL,
  country_iso,
  admin_level = c("national", "adminlevel_1")
)
```

Arguments

`path`	Optional. File path to a `.dta` file.
`.data`	Optional. A preloaded data frame.
`country_iso`	Character. Country ISO3 code.
`admin_level`	Character. One of `'national'` or `'adminlevel_1'`.

Value

A tibble of class `cd_survey_data`.

Examples

```
## Not run:
national <- load_survey_data(path = "all_data.dta", country_iso = "KEN")
cache$set_national_survey(national)

# The survey by region
regional <- load_survey_data(
  path = "gregion_data.dta", country_iso = "KEN", admin_level = "adminlevel_1"
)
cache$set_regional_survey(regional)

## End(Not run)
```

`load_un_estimates` *Load UN estimates*

Description

Loads and processes UN estimates from a `.dta` file or in-memory data. Filters by country and year, renames columns, and assigns a custom class.

Usage

```
load_un_estimates(path = NULL, .data = NULL, country_iso, start_year, end_year)
```

Arguments

`path`	Optional. File path to a `.dta` file.
`.data`	Optional. A preloaded data frame.
`country_iso`	Character. ISO3 code of the country.
`start_year`, `end_year`	Integers. First and last year (inclusive) to include.

Value

A tibble of class `cd_un_estimates`.

Examples

```
## Not run:
un <- load_un_estimates(
  path = "un_estimates.dta", country_iso = "KEN", start_year = 2019, end_year = 2023
)

# From data already read, then kept with the dataset
un <- load_un_estimates(.data = all_un, country_iso = "KEN", start_year = 2019, end_year = 2023)
cache$set_un_estimates(un)

## End(Not run)
```

---

load_un_mortality_data

Load and filter UN mortality estimates

Description

Loads UN mortality data from a file path or existing data frame, filters it by the given country ISO3 code, and returns cleaned mortality data sorted by year.

Usage

```
load_un_mortality_data(path = NULL, .data = NULL, country_iso)
```

Arguments

`path`	Optional. Path to an Excel or CSV file containing UN mortality estimates.
`.data`	Optional. A pre-loaded data frame with UN mortality estimates.
`country_iso`	Required. A 3-letter ISO country code used to filter the dataset.

Value

A tibble of class `cd_un_mortality`.

Examples

```
## Not run:
  load_un_mortality_data("data/un_mortality.csv", country_iso = "KEN")
  load_un_mortality_data(.data = df, country_iso = "UGA")

## End(Not run)
```

load_wuenic_data *Load WUENIC immunization data*

Description

Loads and processes WUENIC estimates, computes dropout and zero-dose indicators, and standardizes naming conventions.

Usage

```
load_wuenic_data(path = NULL, .data = NULL, country_iso)
```

Arguments

path	Optional. File path to a .dta file.
.data	Optional. A preloaded data frame.
country_iso	Character. ISO3 country code.

Value

A tibble of class cd_wuenic_data.

Examples

```
## Not run:
wuenic <- load_wuenic_data(path = "wuenic.dta", country_iso = "KEN")
wuenic[c("year", "cov_penta3_wuenic", "cov_dropout_penta13_wuenic")]
cache$set_wuenic_estimates(wuenic)

## End(Not run)
```

map_shapefile_ui *Map Shapefile Names step of the Load Data wizard*

Description

The user matches each admin level 1 name of the dataset to the name the shapefile uses for the same area. A button opens the matching dialog (mapping_modal_ui()); saving it stores the mapping with cache()$set_map_mapping(). The names come from the shapefile in use (the one the user uploaded, or the built-in one for the country), in the column cache()$shapefile_name_field. Once a mapping is saved, a "clear mapping" button removes it (cache()$clear_map_mapping()) and a banner says how the names were matched (mapping_provenance_banner()). The step counts as complete when a mapping is saved (see step_map_mapping_complete()), which needs the user to save the dialog at least once.

Usage

```
map_shapefile_ui(id, i18n)

map_shapefile_server(id, cache, i18n)
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.

Value

map_shapefile_ui() returns the UI as Shiny tags: a card. map_shapefile_server() is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "map_mapping", ui = map_shapefile_ui(ns("map_shapefile"), i18n))
map_shapefile_server("map_shapefile", cache, i18n)

## End(Not run)
```

---

map_survey_ui *Map Survey Files step of the Load Data wizard*

Description

The user matches each admin level 1 name of the dataset to the name the regional survey uses for the same area. A button opens the matching dialog (mapping_modal_ui()); saving it stores the mapping with cache()$set_survey_mapping(). The regional survey is the one the user uploaded, or the built-in one for the country, so the step can always be done. Once a mapping is saved, a "clear mapping" button removes it (cache()$clear_survey_mapping()) and a banner says how the names were matched (mapping_provenance_banner()). The step counts as complete when a mapping is saved (see step_survey_mapping_complete()), which needs the user to save the dialog at least once.

Usage

```
map_survey_ui(id, i18n)

map_survey_server(id, cache, i18n)
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.

Value

map_survey_ui() returns the UI as Shiny tags: a card. map_survey_server() is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "survey_mapping", ui = map_survey_ui(ns("map_survey"), i18n))
map_survey_server("map_survey", cache, i18n)

## End(Not run)
```

---

mapping_modal_ui *Dialog that matches the dataset's region names to another source*

Description

A button that opens a dialog with one row per admin level 1 name of the dataset (cache()$subnational_regions). In each row the user picks the matching name from another source: the regional survey or the shapefile. The dialog suggests matches itself and keeps the matches saved before. When the user saves, the server stores the mapping in the dataset with cache()$set_survey_mapping() or cache()$set_map_mapping(), depending on type. The two mapping steps of the wizard use it (map_survey_ui(), map_shapefile_ui()).

Usage

```
mapping_modal_ui(id)

mapping_modal_server(
  id,
  cache,
  survey_data,
  survey_map,
  label,
  title,
  show_col,
  type,
  i18n
)
```

Arguments

id	The module's id.
cache	A reactive that returns the dataset, a CacheConnection.
survey_data	A reactive that returns the other source, a data frame (the regional survey, or the shapefile). Its show_col column holds the names the user chooses from.
survey_map	A reactive that returns the mapping saved so far (a data frame as described in Details), or NULL when none is saved.
label	The translation key of the button's text, a string.
title	The translation key of the dialog's title, a string.
show_col	The column of survey_data() that holds the names to match: a string, or a function without arguments (or a reactive) that returns one. A function is called each time the value is needed, so the column can change while the app runs, as an uploaded shapefile's name column does.
type	Where the mapping is saved: "survey_mapping" (with cache()$set_survey_mapping()) or "map_mapping" (with cache()$set_map_mapping()). Any other value saves nothing.
i18n	The app's translator (a shiny.i18n translator), used for every label.

Details

The saved mapping is a tibble with one row per matched name and three columns: admin_level_1 (the dataset's name), a column named after show_col (the matching name in the other source) and source ("auto" when the row is the dialog's own suggestion, left unchanged, and "user" otherwise). Rows the user left without a match are not saved.

The button and dialog appear once the dataset has its admin level 1 names and survey_data() has a value.

Value

mapping_modal_ui() returns the UI as Shiny tags: the placeholder the server fills with the button and its dialog. mapping_modal_server() is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in a module's ui
mapping_modal_ui(ns("survey_data"))

# in its server: match the dataset's names to the regional survey's
mapping_modal_server(
  id = "survey_data",
  cache = cache,
  survey_data = reactive(cache()$regional_survey),
  survey_map = reactive(cache()$survey_mapping),
  label = "btn_upload_map_survey",
```

```
  title = "msg_upload_manual_mapping",
  show_col = "adminlevel_1",
  type = "survey_mapping",
  i18n = i18n
)

## End(Not run)
```

`mapping_provenance_banner` *Banner that says how a saved name mapping was matched*

Description

Builds the information banner the two mapping steps show under their status message once a mapping is saved. It says how many of the saved matches were made automatically and how many the user chose, from the mapping's `source` column (`"auto"` or `"user"` per row, written by `mapping_modal_server()`).

Usage

```
mapping_provenance_banner(mapping_df, i18n)
```

Arguments

`mapping_df`	The saved mapping, a data frame with a `source` column, or `NULL`.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.

Value

The banner as Shiny tags, or `NULL` when `mapping_df` is `NULL`, has no rows or has no `source` column.

Examples

```
## Not run:
# in a module's server: under the mapping's button, say how the saved names were matched
output$survey_map_summary <- renderUI(mapping_provenance_banner(cache()$survey_mapping, i18n))

# the banner for a mapping of three names, two matched by the dialog and one by the user
mapping <- data.frame(
  admin_level_1 = c("Nairobi", "Mombasa", "Kisumu"),
  adminlevel_1 = c("Nairobi", "Mombasa", "Kisumu"),
  source = c("auto", "auto", "user")
)
mapping_provenance_banner(mapping, i18n)

## End(Not run)
```

`merge_and_standardize` *Merge and standardize sheets read by* `load_excel_parts()`

Description

Merges the separate sheets returned by `load_excel_parts()` into one dataset, standardizes its columns and returns it as a `cd_data` tibble. This is the second half of loading an Excel workbook.

Usage

```
merge_and_standardize(
  parts_result,
  indicator_group = c("auto", "vaccine", "rmncah", "custom"),
  profile = NULL,
  profile_name = NULL,
  validate = TRUE
)
```

Arguments

`parts_result`	A list. What `load_excel_parts()` returns.
`indicator_group`	One of `"auto"`, `"vaccine"`, `"rmncah"`, `"custom"`. Default `"auto"`.
`profile`	Optional. Either a **name** (string) of an existing group, or a **definition** to register inline before loading: • explicit form: `list(name = "rmncah", value = list(hiv = c("hiv_test","pmtct1")))` • compact form: `list(hiv_profile = list(testing = c("hiv_test"), care = c("art_new")))` For `"custom"`, the profile must resolve to **one** concrete name.
`profile_name`	Optional. The resolved profile (indicator group) name, used when resolving the indicator group (for example with `indicator_group = "custom"`). Default is `NULL`.
`validate`	Logical. Whether to stop with an error when the merged data fails a data quality check: `check_district_consistency()`, `check_month_presence()` or `check_month_validity()`. Default is `TRUE`. With `FALSE` a file that has such problems still loads, so the caller can run the checks later with `run_all_quality_checks()`. Structural checks, such as a missing Admin sheet column, always run.

Details

Before merging, the function stops with an error if the Admin sheet fails `check_admin_columns()` or `check_single_country()`. The merged data then goes to `new_countdown()`, which resolves the indicator group and, when `validate` is `TRUE`, stops on a data quality problem.

Value

A tibble of class `cd_data`.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
data <- merge_and_standardize(sheets)
data

# Load even if a data quality check fails, to run the checks afterwards
data <- merge_and_standardize(sheets, validate = FALSE)
```

`national_coverage_ui` *National coverage page*

Description

Shows national coverage by year for each indicator: the estimate from the health facility data, with the denominator selected for the indicator, beside the survey and WUENIC estimates. The page is the coverage charts (`coverage_ui()`) at the national level: one card with a tab for each indicator in `cd_cfg("nat_cov_indicators")` and a Custom tab. It has no filters. The data come from the dataset's `get_filtered_coverage()`, which reads `cache()$calculate_coverage("national")`.

Usage

```
national_coverage_ui(id, i18n)

national_coverage_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

`national_coverage_ui()` returns the page's UI as Shiny tags. `national_coverage_server()` returns what `coverage_server()` returns: a reactive that holds the key of the tab shown.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "national_coverage",
  ui = national_coverage_ui,
  server = national_coverage_server,
  title = "title_coverage_national",
  section = "title_nav_national_analysis",
  subtitle = "sub_cov_national"
)

## End(Not run)
```

`national_inequality_ui` *National inequality page*

Description

Shows how coverage differs between the admin level 1 areas of the country, in two cards. The first is the inequality charts (`inequality_ui()`): each area's coverage by year against national coverage, from `cache()$inequality_admin1`. The second is the maps of the same coverage (`subnational_mapping_ui()`), from the dataset's `get_filtered_mapping_data()`; changing the tab of the charts changes the tab of the maps too. Two controls above the maps apply to the maps only: the years to map, which are saved in the dataset (`set_mapping_years()`), and the colour palette.

Usage

```
national_inequality_ui(id, i18n)

national_inequality_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The charts and maps do no work before that.

Value

`national_inequality_ui()` returns the page's UI as Shiny tags. `national_inequality_server()` is called for its side effects; its value (the observer that saves the chosen years) is of no use to a caller.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "national_inequality",
  ui = national_inequality_ui,
  server = national_inequality_server,
  title = "title_inequ_national",
  section = "title_nav_national_analysis",
  subtitle = "sub_inequ_national"
)

## End(Not run)
```

---

national_rates_required_fields

The values the National Rates step requires

Description

Return the names of the values that must be set before the National Rates step counts as complete: every field of the app's wizard configuration (cd_wizard_config()). national_rates_required_field gives the national rates, which are elements of the dataset's national_estimates. survey_estimates_required_f gives the survey coverage estimates, which are elements of the dataset's survey_estimates. They are functions because the list differs between apps and is read from the configuration when called.

Usage

```
national_rates_required_fields()

survey_estimates_required_fields()
```

Value

A character vector of names, in display order: the key of every "rate" field for national_rates_required_fields(and of every "survey" field for survey_estimates_required_fields().

See Also

step_national_rates_complete(), which uses both.

Examples

```
old <- options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_survey_field("ideliv_prop", "instlivebirths", "title_upload_ideliv_survey"),
```

```
    cd_wizard_national_rate_fields()
  )
)))

national_rates_required_fields()
survey_estimates_required_fields()

options(old)
```

`national_rates_ui` *National rates step of the Load Data wizard*

Description

The user enters the national figures the denominators are calculated from: the survey coverage estimates (in percent) and national rates (as proportions) listed in the app's wizard configuration (`cd_wizard_config()`), the year of the most recent survey, and the year the surveys start from. The fields show what the dataset already holds, so values taken from an uploaded national survey appear filled in; a banner says so when the user has uploaded a national survey. Each change is saved to the dataset shortly after it is made. The step counts as complete when every listed field has a value and the survey year is set (see `step_national_rates_complete()`).

Usage

```
national_rates_ui(id, i18n)

national_rates_server(id, cache, i18n)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.

Details

What is saved:

- survey coverage estimates, rounded to one decimal place, with `cache()$set_survey_estimates()`;
- national rates, rounded to three decimal places, with `cache()$set_national_estimates()`;
- the most recent survey year (2015 to 2030) with `cache()$set_survey_year()`, or `cache()$clear_survey_year()` when the user empties the field;
- the survey start year with `cache()$set_start_survey_year()`. This field is a list of `cache()$survey_years` and only shows when the dataset has survey years. When the saved start year is not one of them, the earliest is saved in its place.

Emptying an estimate or a rate field saves a missing value (`NA`) for it, which makes the step incomplete again.

Value

`national_rates_ui()` returns the UI as Shiny tags: a card with the groups of fields. `national_rates_server()` is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# the app describes the step's fields once, before the wizard is used
options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))

# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "national_rates", ui = national_rates_ui(ns("national_rates"), i18n))
national_rates_server("national_rates", cache, i18n)

## End(Not run)
```

`national_target_ui` *National coverage target page*

Description

Shows how much of the country meets the coverage targets. One filter chooses the admin level (admin level 1 or district). The page is the target charts and table (`target_ui()`) for the whole country: by year, the percentage of regions or districts that meet each target group's target (the dataset's `get_filtered_threshold()`), and a table of the areas whose coverage of a chosen indicator reaches the target (the dataset's `get_high_performers()`).

Usage

```
national_target_ui(id, i18n)

national_target_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

`national_target_ui()` returns the page's UI as Shiny tags. `national_target_server()` returns what `target_server()` returns: a reactive that holds the key of the tab shown.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "national_target",
  ui = national_target_ui,
  server = national_target_server,
  title = "title_nav_global_coverage",
  section = "title_nav_national_analysis",
  subtitle = "sub_target_national"
)

## End(Not run)
```

---

new_countdown *Create a* cd_data *object from cleaned data and resolve the indicator group*

Description

Validates required columns, resolves the concrete indicator group name from the **data** (`"auto"` detection supported), ensures the selected group's indicators are fully present, sets the global selection, and returns a `cd_data`.

Usage

```
new_countdown(
  .data,
  class = NULL,
  indicator_group = c("auto", "vaccine", "rmncah", "custom"),
  profile_name = NULL,
  profile = NULL,
  validate = TRUE
)
```

Arguments

`.data`	Tibble after cleaning/merge.
`class`	Optional character vector of extra classes to add in front of `"cd_data"`. Default is `NULL`.
`indicator_group`	One of `"auto"`, `"vaccine"`, `"rmncah"`, `"custom"`.

`profile_name`	Optional. The resolved profile (indicator group) name, used when resolving the indicator group (for example with `indicator_group = "custom"`). Default is `NULL`.
`profile`	For `"custom"`, the group name to select (must exist). Ignored otherwise. Stored as the `profile` attribute of the result.
`validate`	Logical. Whether to run the Tier B quality checks (district consistency, month presence and month validity). Default is `TRUE`.

Value

A tibble with class `cd_data` and `attr(, "indicator_group")` set to the resolved name. Also sets `options(cd2030.selected_group)` via `set_selected_group()`.

Examples

```
## Not run:
x <- .load_excel_data("data/country.xlsx", indicator_group = "auto")
cd <- new_countdown(x, indicator_group = "auto")
attr(cd, "indicator_group")

## End(Not run)
```

`notebook_data` *Give DataSuite's notebooks a folder's Countdown datasets*

Description

Lists, prepares, attaches or describes the datasets of a folder for notebooks in R, Python and Stata. A dataset is the `.rds` file an app saved next to a country's Excel file, together with its workspace. DataSuite calls this function for the apps that declare it as their `notebookData`. It does not write to an `.xlsx` or `.rds` file: it writes only to the workspace's `data/` folder.

Usage

```
notebook_data(
  action = c("list", "prepare", "attach", "describe"),
  folder,
  workspace = NULL,
  datasets = NULL,
  force = FALSE,
  period = 60,
  tables = NULL
)
```

Arguments

action	"list" (the datasets and their tables, from the file names alone), "prepare" (Stata files of the tables of datasets, for Python and Stata, made or remade when their .rds changed) or "attach" (in a notebook's R session: the tables by name, and ds_list(), ds_use(), ds_save(), ds_reload()) or "describe" (the own dataset's tables with their columns and what each column holds, for an assistant writing code; read from the .rds, which is not changed).
folder	The folder holding the Excel files, the .rds files and their workspaces.
workspace	The notebook's workspace folder (<stem>.shiny-workspace). The .rds file with the same stem is the dataset the notebook belongs to.
datasets	For "prepare": the datasets (their .rds file names without .rds, or "ref") to prepare. The dataset of workspace is always included.
force	For "prepare": check the .rds files now, not only when the last check is older than period.
period	Seconds between checks of an .rds for changes (default 60).
tables	For "prepare": the tables to write (names from "list"); the usual ones when NULL. Tables already written are kept, and written again once their .rds has changed. For "describe": the tables to describe; the usual ones when NULL.

Value

For "list", "prepare" and "describe", a list (DataSuite reads it as JSON); for "attach", invisibly the names attached.

Examples

```
# a folder with a country's Excel file that no app has opened yet
folder <- tempfile("countdown_")
dir.create(folder)
file.copy(system.file("extdata", "kenya.xlsx", package = "cd2030.core"), folder)

listing <- notebook_data("list", folder = folder)
listing$datasets[[1]]$status
vapply(listing$ref$tables, function(t) t$name, character(1))
unlink(folder, recursive = TRUE)

## Not run:
# in a notebook of the workspace of kenya_rmncah.rds: its tables by name, and ds_list()
workspace <- file.path("data", "kenya_rmncah.shiny-workspace")
notebook_data("attach", folder = "data", workspace = workspace)
head(adjusted_data)

# for Python and Stata: Stata files of two of its tables in the workspace's data/ folder
notebook_data("prepare", folder = "data", workspace = workspace,
              tables = c("adjusted_data", "national_survey"))

## End(Not run)
```

`nr_field` *One number field of the wizard's forms*

Description

Builds one labelled number field with the interface kit's `cd_field_number()`. The field checks what is typed against `min` and `max`, shows an error under the field for a value out of range, and sends only a valid value to the server. A required field that is empty shows a "not set" warning. A field whose hint is `"hint_upload_percent"` shows a "%" unit at its end.

Usage

```
nr_field(ns, i18n, inputId, labelKey, min, max, step, hintKey, required = TRUE)
```

Arguments

`ns`	The namespace function of the module the field belongs to (from `NS(id)` or `session$ns`).
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`inputId`	The input id of the field, without the namespace, a string.
`labelKey`	The translation key of the field's label, a string.
`min`, `max`	The smallest and largest value the field accepts, numbers.
`step`	The amount one step of the field changes the value by, a number.
`hintKey`	The translation key of the hint under the field, a string, or `NULL` for no hint.
`required`	`TRUE` (the default) when the field must have a value; an empty required field shows the "not set" warning.

Value

The field as Shiny tags.

Examples

```
## Not run:
# in a module's ui, where ns <- NS(id)
nr_field(ns, i18n, "anc4_prop", "title_upload_anc4_survey", 0, 100, 1, "hint_upload_percent")

# a field without a hint, which may be left empty
nr_field(ns, i18n, "survey_year", "title_upload_recent_survey_year", 2015, 2030, 1, NULL,
  required = FALSE
)

# from a field description of the app's wizard configuration
f <- cd_wizard_rate_field("stillbirth_rate", "sbr", "title_upload_stillbirth")
```

```
nr_field(ns, i18n, f$id, f$label_key, f$min, f$max, f$step, f$hint)

## End(Not run)
```

`nr_group` *A titled group of fields in the wizard's forms*

Description

Lays out fields in a grid under a title and a subtitle. The National Rates step uses it for each group of fields, and the Upload Data step for its reference-data upload zones.

Usage

```
nr_group(i18n, titleKey, subtitleKey, ..., cols = 4)
```

Arguments

`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`titleKey`	The translation key of the group's title, a string.
`subtitleKey`	The translation key of the group's subtitle, a string.
`...`	The fields of the group, as Shiny tags (for example from `nr_field()`), in display order.
`cols`	The number of columns of the grid, a whole number. The default is 4.

Value

The group as Shiny tags.

Examples

```
## Not run:
# in a module's ui, where ns <- NS(id)
nr_group(i18n, "title_upload_group_survey_ref", "sub_upload_group_survey_ref",
  nr_field(ns, i18n, "survey_year", "title_upload_recent_survey_year", 2015, 2030, 1, NULL),
  uiOutput(ns("survey_start_ui"))
)

# two columns in place of four
nr_group(i18n, "title_upload_group_immunization", "sub_upload_group_immunization",
  nr_field(ns, i18n, "penta3_prop", "title_upload_penta3_survey", 0, 100, 1, "hint_upload_percent"),
  nr_field(ns, i18n, "bcg_prop", "title_upload_bcg_survey", 0, 100, 1, "hint_upload_percent"),
  cols = 2
)

## End(Not run)
```

`outlier_detection_ui` *Outlier detection page*

Description

The data quality page on extreme values in the monthly data. Its first card has three tabs for the chosen indicator: a heat map of the outlier summary by admin level 1 area or district, a bar chart by indicator and a bar chart by area. Below it are a table of the districts and months flagged as outliers in the year picked on the table (the reported value with its median and median absolute deviation), and a trend chart of those outliers for one region and that year. The filters are the indicator and the admin level with its region. The trend chart has its own region filter, which follows the page's region.

Usage

```
outlier_detection_ui(id, i18n)

outlier_detection_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Details

The page reads `outliers_admin1`, `outliers_district` and `list_outlier_units` from the dataset, and calls its `calculate_outliers_summary()` when a region is chosen. The heat map's Excel download also writes `outliers_national` and `district_outliers_summary`.

Value

`outlier_detection_ui()` returns the page's UI as Shiny tags. `outlier_detection_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "outlier_detection",
  ui = outlier_detection_ui,
  server = outlier_detection_server,
  title = "title_outlier_main",
  section = "lbl_nav_section_quality",
```

```
  subtitle = "sub_outlier_main"
)

## End(Not run)
```

`overall_score_ui` *Overall score page*

Description

The data quality page that brings the checks together in one table: the completeness of monthly reporting, extreme outliers and the consistency of annual reporting, with a score for each year of the data. The one filter is the region: with none chosen the table is for the country, otherwise it is for that admin level 1 area. The table can be downloaded.

Usage

```
overall_score_ui(id, i18n)

overall_score_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Details

The page calls the dataset's `calculate_overall_score()`, and reads its `performance_threshold` (shown in a row label) and `data_years`.

Value

`overall_score_ui()` returns the page's UI as Shiny tags. `overall_score_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "overall_score",
  ui = overall_score_ui,
  server = overall_score_server,
```

```
  title = "title_score_main",
  section = "lbl_nav_section_quality",
  subtitle = "sub_score_main"
)

## End(Not run)
```

`plot.cd_adjustment_values_filtered` *Plot adjusted vs. unadjusted data for health indicators*

Description

`plot.cd_adjustment_values_filtered` creates a bar plot to compare the unadjusted (raw) and adjusted values of health indicators over time. It allows users to specify the indicator prefix and customize legend labels for flexibility across different health data.

Usage

```
## S3 method for class 'cd_adjustment_values_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  type = c("stacked", "change", "totals"),
  zoom = TRUE,
  options = NULL
)
```

Arguments

`x`	A data frame containing the `year` column and columns for the raw and adjusted values of health indicators (e.g., `ideliv_raw`, `ideliv_adj`). The indicator plotted is taken from its `indicator` attribute.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.

`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	A named list or vector of custom legend labels, with names `raw` (unadjusted data) and/or `adjusted`. Supplied entries replace the defaults ("N of `indicator` before adjustment" and "N of `indicator` after adjustment"). Default is `NULL`.
`type`	With the adjustment's steps in `x` (`generate_adjustment_values()` with `settings`): `"stacked"` (the default) draws, each year, the reported count beside the adjusted count made of its parts – the reported count, what completeness added, what filling missing values added, and the outliers' correction (a correction that lowers the total drawn faded above the bar: the part taken away); `"change"` what each step changed, labelled with its share of the reported count; `"totals"` the counts side by side. Legend labels: `raw`, `completeness`, `outliers`, `missing` (and `adjusted` for `"totals"`).
`zoom`	`"stacked"`: `TRUE` starts the axis near the smallest bar, so changes of a few percent can be seen (a note says where it starts); `FALSE` from zero.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

This function helps visualize the difference between raw and adjusted values of a given health indicator, aiding in the assessment of data completeness and adjustments. The difference and percentage difference are calculated within the function but are not directly shown on the plot. Instead, the plot shows the actual unadjusted and adjusted values side-by-side for each year.

Value

A ggplot2 object showing the comparison of unadjusted and adjusted data for the specified indicator over time.

Examples

```
## Not run:
# Using default legend labels and title
plot.cd_adjustment_values_filtered(adjustments, indicator = "ideliv")

# Custom legend labels and title
plot.cd_adjustment_values_filtered(adjustments,
  indicator = "instlivebirths",
  title = "Customized Title",
  legend_labels = c("Original", "Modified")
)

## End(Not run)
```

`plot.cd_average_reporting_rate` *Plot sub-national reporting rates by year and unit*

Description

Visualizes reporting rates for selected health service indicators at sub-national levels, using heat maps or bar plots. Intended for use with outputs from `calculate_average_reporting_rate()` at `"adminlevel_1"` or `"district"` level.

Usage

```
## S3 method for class 'cd_average_reporting_rate'
plot(
  x,
  plot_type = c("heat_map", "bar"),
  indicator = c("anc_rr", "idelv_rr", "vacc_rr", "opd_rr", "ipd_rr"),
  threshold = .cd_method$data_quality$reporting_threshold,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_average_reporting_rate` object, typically the output from `calculate_average_reporti` Must contain subnational data.
`plot_type`	Either `"heat_map"` or `"bar"`: • `"heat_map"`: Shows reporting rates using color-coded tiles by year and unit. • `"bar"`: Displays reporting rates as bars, grouped by year and faceted by unit.
`indicator`	One of the following: • `"anc_rr"`: Antenatal care • `"idelv_rr"`: Institutional deliveries • `"vacc_rr"`: Vaccination • `"opd_rr"`: Outpatient visits • `"ipd_rr"`: Inpatient admissions

`threshold`	Numeric value (default = 90). Used only in `"heat_map"` mode to define the boundary for high reporting rates.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

Only subnational objects are accepted. If the data was computed at `"adminlevel_1"` with a `region` argument, the object is automatically treated as `"district"` level and plotted by district name.

Heat Map Mode:

- Each tile represents a reporting rate for a unit and year.
- Colors:
  - Green: `>= threshold`
  - Orange: `70 <= value < threshold`
  - Red: `< 70`
- Labels display the actual percentage.

Bar Mode:

- Bars show values per year.
- Units (regions or districts) appear as facets.
- Color gradient shows low to high reporting.

Value

A `ggplot` object.

Examples

```
## Not run:
# Heat map for vaccination at district level
x <- calculate_average_reporting_rate(data, admin_level = "district")
plot(x, plot_type = "heat_map", indicator = "vacc_rr")

# Bar chart by district in selected region
```

```
x <- calculate_average_reporting_rate(data, admin_level = "adminlevel_1", region = "Nairobi")
plot(x, plot_type = "bar", indicator = "idelv_rr")

## End(Not run)
```

`plot.cd_bayes_model` *Plot S3 method for Bayesian coverage model*

Description

Leverages the bayescoveragemodel package's internal plotting function and applies the custom Countdown (CD) theme and translated labels.

Usage

```
## S3 method for class 'cd_bayes_model'
plot(
  x,
  region = NULL,
  year = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  national = NULL,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`x`	An object of class `cd_bayes_model` returned by `generate_bayes_model()`.
`region`	(Optional) A region of a sub-national model: its estimates by year are drawn, as a national model's are. `NULL` compares the regions in one year.
`year`	(Optional) For a sub-national model. The year the regions are compared in; by default the latest year the facility data has. With a `region`, the year is marked on its chart.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.

`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`national`	(Optional) For a sub-national model. The country's coverage from the facility data, with the same denominator: a data frame with `year` and `value` (a proportion), as `cache$bayes_national_routine()` gives. The comparison draws it as a line and a region's chart as a series.
`legend_labels`	(Optional) A named list to override the legend's entries, e.g. with translations: `model`, `routine` (the facility data) and `national`.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object with the chart options applied. For a national model, or the `region` of a sub-national model: the estimated coverage by year. For a sub-national model with no `region`: the regions compared in `year`, each with its model estimate and interval and its facility-data value.

Examples

```
## Not run:
# `cache` is a dataset with adjusted data; fitting needs the Bayesian model packages
national <- cache$get_bayes_model("national", "penta3")
plot(national)
plot(national, title = "Penta3 coverage, Kenya", y_title = "Coverage")

# a sub-national model: the regions compared, or the trend of one region
regional <- cache$get_bayes_model("adminlevel_1", "penta3")
country <- cache$bayes_national_routine("penta3")
plot(regional, national = country)
plot(regional, year = 2021, national = country)
plot(regional, region = "Nairobi", national = country)

## End(Not run)
```

`plot.cd_completeness_summary` *Plot missing summary*

Description

Draws the share of missing values in a completeness summary as a heat map by unit, for one indicator by year or for all indicators averaged over the years. With `plot_type = "trend"` it draws the completeness of one indicator over time, one panel per unit.

Usage

```
## S3 method for class 'cd_completeness_summary'
plot(
  x,
  plot_type = c("heat_map", "trend"),
  indicator = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  indicator_labels = NULL,
  options = NULL
)
```

Arguments

x	A `cd_completeness_summary` object, as returned by `calculate_completeness_summary()`.
plot_type	Either `"heat_map"` (missingness by unit, for one or all indicators) or `"trend"` (completeness over time for one indicator). Default is `"heat_map"`.
indicator	Optional. One of the supported indicators (`'opv1'`, `'penta3'`, etc.) to visualize in the plot. If `NULL` all indicators will be shown.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
indicator_labels	(Optional) A named vector or list, indicator code -> label, for the axis that names indicators (see `cd_indicator_labels()`, which gives them translated). By default the English labels.
options	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object: a heat map or a faceted line chart, depending on `plot_type`.

Examples

```
## Not run:
# Region-level summary
plot(missing_data, indicator = "penta3")

## End(Not run)
```

`plot.cd_coverage` *Plot coverage by region*

Description

Generates a bar chart comparing indicator coverage across regions, highlighting a specific region and categorizing performance as lower, average, or higher.

Usage

```
## S3 method for class 'cd_coverage'
plot(
  x,
  indicator = NULL,
  denominator = NULL,
  year = NULL,
  region = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`x`	Data frame containing the coverage data.
`indicator`	Character string specifying the indicator (e.g., 'penta1').
`denominator`	Character string specifying the denominator ('penta1', 'anc1', 'dhis2', 'penta1derived', 'anc1derived').
`year`	Integer representing the year to plot.
`region`	Character string for the specific region to highlight in yellow.
`title`	The chart's title.
`subtitle`	The line under the title.

`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	(Optional) Named list to override the default category labels (e.g., `list(lower = "Low", average = "Avg", higher = "High")`).
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object with the chart options applied: one bar per unit for the selected year, ordered by coverage and coloured by category (the lower, middle and upper third of the values).

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
coverage <- cache$calculate_coverage("adminlevel_1")

# `region` is the unit drawn in yellow
plot(coverage, indicator = "penta3", denominator = "penta1", year = 2023, region = "Nairobi")
plot(coverage, indicator = "penta3", denominator = "penta1", year = 2023, region = "Nairobi",
     title = "Penta3 coverage by region, 2023", y_title = "Coverage (%)",
     legend_labels = list(lower = "Lowest third", average = "Middle third", higher = "Highest third"))
```

---

plot.cd_coverage_filtered

Plot national coverage data

Description

Generates a line plot to visualize immunization coverage data across years, allowing comparison between different estimates (e.g., DHIS2 estimates, WUENIC estimates, and Survey estimates).

Usage

```
## S3 method for class 'cd_coverage_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	A data frame of type cd_coverage_filtered, containing year-wise coverage data for the specified indicator.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	(Optional) A named list or vector to override the legend keys for translation. Valid keys: dhis2, wuenic, survey, ci. Defaults to NULL.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object displaying a line plot of the coverage data.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
coverage <- cache$get_filtered_coverage("penta3", "national")

plot(coverage)
plot(coverage, title = "Penta3 coverage, Kenya", y_title = "Coverage (%)",
     legend_labels = list(dhis2 = "Facility data", survey = "Household surveys"))
```

`plot.cd_coverage_selected` *Plot S3 method for coverage data*

Description

Draws the latest coverage of a set of indicators. National data, or the data of one region, is drawn as a profile (bars by indicator and source) or as the gap between the facility and the survey estimate. Subnational data is drawn as dots by unit, one panel per indicator, or as a heatmap of units by indicator.

Usage

```
## S3 method for class 'cd_coverage_selected'
plot(
  x,
  type = NULL,
  source_labels = NULL,
  indicator_labels = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	An object of class `cd_coverage_selected`.
`type`	"profile" or "gap" (for national), "dot" or "heatmap" (for subnational).
`source_labels`	(Optional) Named list to translate 'facility', 'survey', 'wuenic'.
`indicator_labels`	(Optional) Named list to translate indicator names (e.g. 'anc4').
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.

`options` A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object with the chart options applied. Which chart it is depends on `type` and on the admin level of `x`.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

# national: a profile of the indicators, or the gap between facility data and the survey
national <- cache$generate_coverage_data("national", "maternal")
plot(national, type = "profile")
plot(national, type = "gap", title = "Facility data against the survey", legend_position = "bottom")

# by region: a heatmap (or "dot")
regions <- cache$generate_coverage_data("adminlevel_1", "child")
plot(regions, type = "heatmap", legend_title = "Coverage (%)")
```

---

plot.cd_derived_coverage

Plot derived vs traditional coverage over time

Description

Generates a line plot comparing traditional (`coverage_old`) and derived (`coverage_new`) coverage estimates over time for a single indicator. It supports both national and subnational views.

Usage

```
## S3 method for class 'cd_derived_coverage'
plot(
  x,
  type = c("bar", "trend"),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = list(),
```

```
  options = NULL
)
```

Arguments

x	A cd_derived_coverage object.
type	A character string specifying the plot type: "bar" or "trend". "trend" can only be applied nationally or to a specific region. Defaults to "bar".
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	(Optional) A named list of character strings to override specific default legend labels (e.g., list(penta1derived = "Custom Penta1")).
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object showing coverage trends over time.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
derived <- cache$calculate_derived_coverage("penta3", "national")

plot(derived)
plot(derived, type = "trend", title = "Penta3: usual and derived denominators",
     y_title = "Coverage (%)", legend_position = "bottom")
```

---

plot.cd_district_reporting_rate

Plot district reporting rate summary

Description

Generates a bar plot displaying the percentage of districts with reporting rates below the defined threshold for multiple indicators across various years. This plot provides a quick visual assessment of district-level reporting compliance across indicators like ANC, Institutional Delivery, PNC, Vaccination, OPD, and IPD.

Usage

```
## S3 method for class 'cd_district_reporting_rate'
plot(
  x,
  indicator_labels = NULL,
  facet_ncol = 3,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_district_reporting_rate` data frame containing reporting rate data, processed by `calculate_district_reporting_rate()`.
`indicator_labels`	Optional named character vector of panel titles, keyed by short indicator name (`anc`, `idelv`, `vacc`, `pnc`, `opd`, `ipd`). Supplied entries replace the defaults.
`facet_ncol`	Integer. The number of columns the service panels are laid out in. Default is `3`, so four services are drawn as a row of three and a row of one.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

Inverts the percentages in `x` (the percentage of districts at or above the threshold) to display the percentage of districts with rates below the threshold for each indicator (e.g., if 95% of districts reach the threshold, 5% are shown as below it). Each indicator is visualized in a separate panel with data grouped by year, providing an overview of performance trends over time.

Indicators plotted include:

- **Antenatal Care (ANC)** - Percentage of districts meeting or exceeding target rates
- **Institutional Delivery** - Institutional delivery compliance over time
- **Postnatal Care (PNC)** - Districts' PNC service rates against thresholds
- **Vaccination** - Vaccination service coverage at district level
- **Outpatient Department (OPD)** - OPD reporting rates in districts
- **Inpatient Department (IPD)** - IPD reporting compliance by district

The plot output includes a title, axis labels, and a legend for year, allowing users to identify service areas with low reporting compliance.

Value

A ggplot object visualizing reporting rates across indicators and years.

Examples

```
## Not run:
# Generate a plot of district reporting rates below a threshold of 90%
plot(cd_district_reporting_rate(data), threshold = 90)

## End(Not run)
```

---

plot.cd_fpet_data *Plot family planning estimation tool (FPET) data*

Description

Generates a line and ribbon plot for FPET indicators (e.g., mCPR, Demand Satisfied) displaying median estimates and 95% credible intervals.

Usage

```
## S3 method for class 'cd_fpet_data'
plot(
  x,
  indicator_labels = NULL,
  title = NULL,
  subtitle = NULL,
```

```
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

x	An object of class cd_fpet_data.
indicator_labels	(Optional) A named list mapping raw indicator names to translated legend labels.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object with the chart options applied: one line per indicator by year, with a shaded band for the credible interval.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
fpet <- cache$fpet_data

plot(fpet)
plot(
  fpet,
  indicator_labels = list(prevalence = "Modern contraceptive prevalence", demand = "Demand satisfied"),
  title = "Family planning in Kenya"
)
```

`plot.cd_health_system_comparison` *Compare district vs admin level 1 health metrics*

Description

Generates scatterplots with linear fit and identity line comparing district to admin level 1 metrics or coverage vs density.

Usage

```
## S3 method for class 'cd_health_system_comparison'
plot(
  x,
  indicator = c("cov_instdeliveries_hstaff", "ratio_opd_u5_hstaff", "ratio_ipd_u5_hos",
    "ratio_ipd_u5_bed"),
  denominator = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`x`	A data frame from `calculate_health_system_comparison()`.
`indicator`	One of ratio_* or cov_instdeliveries_* (requires `denominator`).
`denominator`	Optional for cov_ indicators. Must be one of 'dhis2', 'anc1', 'penta1'.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	(Optional) A named list to override default legend labels (keys: 'admin', 'linear').
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
comparison <- cache$health_system_comparison

plot(comparison, indicator = "ratio_opd_u5_hstaff")
plot(comparison, indicator = "ratio_ipd_u5_bed", title = "Admissions of children and hospital beds",
     legend_labels = list(admin = "Regions", linear = "Trend"))
```

---

plot.cd_health_system_metric

Plot health metrics for admin level 1 units

Description

Plots individual health system indicators by admin level 1 and compares with national value or target.

Usage

```
## S3 method for class 'cd_health_system_metric'
plot(
  x,
  indicator = c("ratio_fac_pop", "ratio_hos_pop", "ratio_bed_pop", "ratio_hstaff_pop",
    "skill_mix"),
  national_value = NA_real_,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	Data frame containing admin level 1 indicators.
indicator	A single metric to plot. Options include ratio_fac_pop, ratio_hos_pop, ratio_bed_pop, ratio_hstaff_pop, skill_mix.

`national_value`
: A numeric value representing the national average to plot as a vertical line. Can be NA.

`title`
: The chart's title.

`subtitle`
: The line under the title.

`caption`
: The note under the chart.

`x_title`
: The title of the horizontal axis.

`y_title`
: The title of the vertical axis.

`legend_title`
: The title of the legend.

`...`
: Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.

`legend_labels`
: (Optional) A named list to override default legend labels (e.g. for translation).

`options`
: A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
regions <- cache$health_system_metrics_admin1
national <- cache$health_system_metrics_national

plot(regions, indicator = "ratio_fac_pop", national_value = national$ratio_fac_pop)
plot(regions, indicator = "ratio_hstaff_pop", national_value = national$ratio_hstaff_pop,
     title = "Health workers per 10,000 people", x_title = "Health workers",
     legend_labels = list(threshold_line = "Benchmark (44.5)"))
```

---

plot.cd_health_system_table

Plot S3 method for health system metrics

Description

Draws the national health system metrics as a formatted table. The indicators are grouped under their section, each with its value and unit, and the year is shown in a line above the column headers.

Usage

```
## S3 method for class 'cd_health_system_table'
plot(
  x,
  year = 2024,
  width = NULL,
  indicator_label = "Indicator",
  value_label = "Value",
  unit_label = "Unit",
  ...,
  options = NULL
)
```

Arguments

`x`	An object of class `cd_health_system_table`, as returned by `generate_health_system_table()`
`year`	The year to display in the header (defaults to 2024)
`width`	(Optional) Total width in inches. If NULL, autofit is used.
`indicator_label`	(Optional) Translated label for the Indicator column header
`value_label`	(Optional) Translated label for the Value column header
`unit_label`	(Optional) Translated label for the Unit column header
`...`	Additional arguments
`options`	(Optional) A `cd_chart_options()` object: text, legend, fonts and sizes a user changed. Applied last, so it wins over the arguments above; the plot's own defaults are used for whatever it does not set. Any chart option can also be given by name in `....`

Value

A `flextable` object (from the flextable package), not a ggplot. Chart options change ggplots only, so `options` has no effect on this table.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
metrics <- cache$generate_health_system_table()

plot(metrics, year = 2023)
plot(metrics, year = 2023, width = 6, indicator_label = "Indicateur", value_label = "Valeur",
     unit_label = "Unite")
```

plot.cd_indicator_coverage *Plot national denominators and coverage indicators*

Description

Generates specific plots for national denominators and health coverage indicators. It supports dynamic customization of titles and labels.

Usage

```
## S3 method for class 'cd_indicator_coverage'
plot(
  x,
  plot_type = c("anc_coverage_dhis2", "delivery_coverage_dhis2",
    "immunization_coverage_dhis2", "anc_coverage_un", "delivery_coverage_un",
    "immunization_coverage_un", "anc_coverage_anc1", "delivery_coverage_anc1",
    "immunization_coverage_anc1", "anc_coverage_penta1", "delivery_coverage_penta1",
    "immunization_coverage_penta1"),
  admin_name = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	A `cd_indicator_coverage` object containing the calculated indicators.
plot_type	Character. The specific indicator set to plot. Valid options include: • **DHIS2 Denominators**: `"anc_coverage_dhis2"`, `"delivery_coverage_dhis2"`, `"immunization_coverage_dhis2"` • **UN Denominators**: `"anc_coverage_un"`, `"delivery_coverage_un"`, `"immunization_coverage_un"` • **ANC1 Derived**: `"anc_coverage_anc1"`, `"delivery_coverage_anc1"`, `"immunization_coverage_anc1"` • **Penta1 Derived**: `"anc_coverage_penta1"`, `"delivery_coverage_penta1"`, `"immunization_coverage_penta1"`
admin_name	Optional character. The specific admin area to filter by (required if data is subnational).
title	The chart's title.

`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional character vector. Custom labels for the legend items. Must match the number of lines in the selected plot type.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

plot(cache$indicator_coverage_national, plot_type = "anc_coverage_penta1")
plot(cache$indicator_coverage_national, plot_type = "immunization_coverage_dhis2",
     title = "Immunization coverage, DHIS2 population", y_title = "Coverage (%)", legend_position = "righ

# subnational data: name the unit
plot(cache$indicator_coverage_admin1, plot_type = "immunization_coverage_dhis2", admin_name = "Nairobi")
```

---

plot.cd_inequality_filtered

Plot subnational health coverage analysis

Description

Generates a plot to visualize health coverage data across subnational units, distinguishing between the national mean and subnational coverage. The Mean Absolute Difference to the Mean (MADM) is displayed as an indicator on the y-axis.

Usage

```
## S3 method for class 'cd_inequality_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	A `cd_inequality_filtered` object returned by `select_inequality()`.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	(Optional) A named list or vector to override the legend keys and MADM label for translation. Valid keys: `subnational`, `national`, `region` (the region's own coverage, drawn when the data is a region's districts) and `madm`. Defaults to `NULL`.
options	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object displaying the subnational health coverage plot.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
inequality <- cache$get_filtered_inequality("penta3", "adminlevel_1")

plot(inequality)
plot(inequality, title = "Penta3 coverage by region", y_title = "Coverage (%)",
```

```
  legend_labels = list(subnational = "Regions", national = "Kenya"))
```

`plot.cd_mapping_filtered` *Plot subnational coverage maps by indicator*

Description

Creates a faceted `ggplot2` map showing subnational coverage levels of a health indicator, colored using a gradient palette. Coverage values are drawn from a `cd_mapping_filtered` object, typically filtered for a specific indicator and denominator.

Usage

```
## S3 method for class 'cd_mapping_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_mapping_filtered` object. Created using `select_mapping_data()`, and must include spatial geometry and metadata attributes (`indicator`, `palette`, `column`).
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object visualizing the spatial distribution of the selected indicator by region and year.

See Also

`select_mapping_data()`, `get_mapping_data()`

Examples

```
## Not run:
# Assuming `map_data` is filtered with select_mapping_data()
plot(
  map_data,
  title = "Penta 1 Coverage by Region",
  legend = "Coverage (%)",
  caption = "Source: DHIS2 2024"
)

## End(Not run)
```

---

plot.cd_mch_curative_index *Plot S3 method for MCH vs curative index*

Description

Draws a scatter plot of the curative services index against the MCH preventive services index, with one point per admin level 1 unit. Dashed lines at the two medians split the plot into four labelled quadrants, and the units with the highest and lowest value of either index are named.

Usage

```
## S3 method for class 'cd_mch_curative_index'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  quad_labels = NULL,
  options = NULL
)
```

Arguments

x	Object of class cd_mch_curative_index, as returned by generate_admin1_mch_curative_index
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
quad_labels	(Optional) A named list of the texts written in the four quadrants: q_top_left, q_top_right, q_bottom_left and q_bottom_right.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object, with the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
index <- cache$generate_admin1_mch_curative_index()

plot(index)
plot(index, title = "Preventive and curative care", x_title = "Preventive index", y_title = "Curative ind
     quad_labels = list(q_top_right = "High on both", q_bottom_left = "Low on both"))
```

---

plot.cd_missing_district

Plot percent of districts with complete data

Description

Visualizes the percentage of districts with complete data for a selected indicator over time.

Usage

```
## S3 method for class 'cd_missing_district'
plot(
  x,
  indicator = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

x	A `cd_missing_district` (or compatible) object containing district-level completeness summaries. Must include a `year` column and corresponding `mis_{indicator}` columns.
indicator	A character string specifying the indicator to plot. Must be one of the values returned by `get_all_indicators()`.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
options	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

The function produces a column chart showing the percent of districts with complete data for the selected indicator by year.

Value

A `ggplot` object.

Examples

```
## Not run:
plot(missing_summary, indicator = "penta1")

## End(Not run)
```

`plot.cd_mortality_ratio_summarised` *Plot completeness of facility reporting ratios*

Description

Calculates and plots the estimated completeness of facility reporting for maternal deaths or stillbirths based on UN estimates and assumed community-to-institution ratios.

Usage

```
## S3 method for class 'cd_mortality_ratio_summarised'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`x`	A `cd_mortality_ratio_summarised` object from completeness estimation.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.

`legend_labels` (Optional) A named list to override the legend's entries, e.g. with translations: `lower`, `best` and `upper` (the UN lower bound, best estimate and upper bound). Defaults to `NULL`.

`options` A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object with ratio lines, labels, and reference points.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
completeness <- cache$summarise_completeness_ratio("mmr")

plot(completeness)
plot(completeness, title = "Completeness of maternal death reporting", y_title = "Completeness (%)",
     legend_labels = list(lower = "UN lower bound", best = "UN best estimate", upper = "UN upper bound"))
```

---

plot.cd_mortality_summary

Plot mortality rate indicators

Description

Produces line plots for national mortality indicators with regional points for comparison.

Usage

```
## S3 method for class 'cd_mortality_summary'
plot(
  x,
  indicator = c("mmr_inst", "ratio_md_sb", "sbr_inst", "nn_inst"),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	A cd_mortality_summary object.
indicator	One of "mmr_inst", "ratio_md_sb", "sbr_inst", or "nn_inst".
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	(Optional) A named list to override the legend's entries, e.g. with translations: national (the national line) and regions (the regional points). Defaults to NULL.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
mortality <- cache$mortality_summary

plot(mortality, indicator = "mmr_inst")
plot(mortality, indicator = "sbr_inst", title = "Institutional stillbirth rate",
     legend_labels = list(national = "Kenya", regions = "Regions"), legend_position = "bottom")
```

---

plot.cd_mortality_summary_filtered

Plot filtered institutional mortality rates

Description

Visualizes institutional mortality rates (MMR or SBR) across regions using filled geographic polygons. Facets the map by year and applies a color gradient by value.

Usage

```
## S3 method for class 'cd_mortality_summary_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_mortality_summary_filtered` object.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

Takes the default title and legend title from the `indicator` attribute of `x` and sets the geometry to WGS84 coordinates. Draws the regions as filled polygons with `geom_sf()`, one panel per year, on a sequential colour scale from the palette stored on `x` (`Reds` by default), with gray for missing values.

Value

A `ggplot2` object: the map, with the chart options applied.

Examples

```
## Not run:
filtered <- select_mortality_summary(mortality_data, "UGA", indicator = "mmr",
                                     plot_year = 2020:2022)
plot(filtered)
```

```
## End(Not run)
```

`plot.cd_outlier` *Visualize summary of outlier detection*

Description

Plots annual trends or heat maps of non-outlier rates for immunization indicators at sub-national levels.

Usage

```
## S3 method for class 'cd_outlier'
plot(
  x,
  selection_type = c("region", "indicator", "heat_map"),
  indicator = NULL,
  threshold = .cd_method$data_quality$reporting_threshold,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  indicator_labels = NULL,
  options = NULL
)
```

Arguments

`x`	A `cd_outlier` object with precomputed outlier flags.
`selection_type`	One of `"region"`, `"indicator"`, or `"heat_map"`: • `"region"`: Non-outlier rates by year and region. • `"indicator"`: Yearly average non-outlier rate per indicator. • `"heat_map"`: Year-by-unit heat map of all or selected indicators.
`indicator`	Optional. Specific indicator name (e.g., `"penta3"`). Required for `"region"` view.
`threshold`	Numeric. Upper cut-off (in percent) of the middle colour band; values above it are shown as good. Default is `90`.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.

`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`indicator_labels`	(Optional) A named vector or list, indicator code -> label, for the axis that names indicators (see `cd_indicator_labels()`, which gives them translated). By default the English labels.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

- Values are assumed to be percentages of non-outliers (i.e., 100 = no outliers).
- `"region"` and `"indicator"` use bar plots with gradient fill.
- `"heat_map"` shows indicator values by year and region.

Value

A `ggplot` object.

Examples

```
## Not run:
plot(outlier_data, selection_type = "region", indicator = "penta3")
plot(outlier_data, selection_type = "heat_map")

## End(Not run)
```

`plot.cd_outlier_list` *Plot outlier time series for a region*

Description

Displays a time-series plot of one indicator for a single region or district, with outlier highlights.

Usage

```
## S3 method for class 'cd_outlier_list'
plot(
  x,
  indicator = NULL,
  year = NULL,
```

```
  region = NULL,
  label = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

x	A `cd_outlier_list` object from `list_outlier_units()`.
indicator	The indicator to plot (for example `"penta1"`).
year	Optional single year to plot. If `NULL` (the default), all years are shown.
region	The name of the unit to plot.
label	Optional named character vector overriding the legend labels. Recognised names are `reported`, `median`, `bounds` and `outliers`.
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
options	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

- Plots observed values, median trend, and 5xMAD range.
- Flags outliers in red.

Value

A `ggplot` object.

Examples

```
## Not run:
list_outlier_units(cd_data, "penta1") %>%
  plot(region_name = "Nakuru")

## End(Not run)
```

`plot.cd_overall_score` *Plot S3 method for overall score*

Description

Draws the overall data quality score as a formatted table, with the metrics grouped by type and a column for each year. Percentage scores are shaded green at or above the threshold stored with the scores, yellow from 70 up to the threshold and red below 70.

Usage

```
## S3 method for class 'cd_overall_score'
plot(x, years = NULL, title = NULL, width = NULL, ..., options = NULL)
```

Arguments

`x`	The score dataframe (class `cd_overall_score`), as returned by `calculate_overall_score()`.
`years`	Vector of years to display. Required: each must be a year column of `x`.
`title`	(Optional) Custom title for the table
`width`	(Optional) Total width in inches. If NULL, autofit is used.
`...`	Additional arguments
`options`	(Optional) A `cd_chart_options()` object: text, legend, fonts and sizes a user changed. Applied last, so it wins over the arguments above; the plot's own defaults are used for whatever it does not set. Any chart option can also be given by name in `....`

Value

A `flextable` object (from the flextable package), not a ggplot. Chart options change ggplots only, so `options` has no effect on this table.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
score <- calculate_overall_score(data, threshold = 90)

plot(score, years = 2019:2023)
plot(score, years = 2021:2023, title = "Data quality, last three years", width = 7)
```

`plot.cd_phc_scatter` *Plot S3 method for PHC performance scatter*

Description

Draws a scatter plot of the service coverage index against facility density or health workforce density, with one point per admin level 1 unit. Dashed lines at the two medians split the plot into four labelled quadrants, and the units flagged as outliers are named.

Usage

```
## S3 method for class 'cd_phc_scatter'
plot(
  x,
  quad_labels = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	An object of class `cd_phc_scatter`, as returned by `generate_phc_scatter_data()`.
`quad_labels`	(Optional) Named list to override quadrant text. Expected keys: `high_high`, `low_high`, `high_low`, `low_low`.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object, with the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
scatter <- cache$generate_phc_scatter_data("ratio_fac_pop")

plot(scatter)
plot(scatter, title = "Facilities and services by region", y_title = "Service index",
     quad_labels = list(low_low = "Few facilities, few services"))
```

---

plot.cd_population_metrics

Plot national population or births metrics

Description

Generates a line graph to visualize national-level demographic data, comparing DHIS-2 data/projections against UN estimates. This function is restricted to national-level data.

Usage

```
## S3 method for class 'cd_population_metrics'
plot(
  x,
  metric = c("population", "births", "under1"),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`x`	A `cd_population_metrics` object containing national-level demographic data.
`metric`	A character string specifying the type of data to plot. Must be one of: • **'population'**: Total population estimates (in thousands). • **'births'**: Total live births (in thousands). • **'under1'**: Population under 1 year of age (in thousands).
`title`	The chart's title.
`subtitle`	The line under the title.

`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional. A character vector of custom labels for the legend items. Must match the number of lines plotted for the selected metric: • 'population': 2 labels • 'births': 3 labels • 'under1': 2 labels
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

The function compares local Health Management Information System (DHIS-2) data with United Nations (UN) estimates.

Default Configuration:

- **Population**: Compares `un_population` vs `totpop_dhis2`.
- **Births**: Compares `un_births` vs `totlivebirths_dhis2` vs `totbirths_dhis2`.
- **Under 1**: Compares `un_under1` vs `totunder1_dhis2`.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
population <- cache$denominator_metrics

plot(population, metric = "population")
plot(population, metric = "births", title = "Births: DHIS2 and the UN", y_title = "Thousands",
     legend_labels = c("UN live births", "DHIS2 live births", "DHIS2 births"))
```

`plot.cd_private_sector` *Plot S3 method for private sector ownership*

Description

Draws a stacked bar for each admin level 1 unit showing the percentage of health facilities that are private, NGO and public, with the percentage written inside each segment.

Usage

```
## S3 method for class 'cd_private_sector'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	An object of class `cd_private_sector`, as returned by `generate_private_sector_data()`.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A `ggplot` object, with the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
ownership <- cache$generate_private_sector_data()

plot(ownership)
plot(ownership, title = "Who owns the health facilities", y_title = "Facilities (%)",
     legend_title = "Owner", legend_position = "right")
```

---

plot.cd_private_sector_plot_data

Plot private sector prevalence by sector and area/national level

---

Description

Generates a stacked bar chart for RMNCAH indicators comparing public vs. private sector share by either national or area-level (urban/rural) estimates.

Usage

```
## S3 method for class 'cd_private_sector_plot_data'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  country_name = NULL,
  legend_labels = NULL,
  indicator_labels = NULL,
  area_labels = NULL,
  share_label = NULL,
  options = NULL
)
```

Arguments

x	A data frame of class "cd_private_sector_plot_data" generated by prepare_private_sector_plot_data().
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.

`x_title` The title of the horizontal axis.

`y_title` The title of the vertical axis.

`legend_title` The title of the legend.

`...` Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.

`country_name` Optional character string to override the default country name used in the plot title. If `NULL`, uses the unique value in the `country` column if present.

`legend_labels` (Optional) A named list to override the legend's entries, e.g. with translations: `public` and `private`.

`indicator_labels` (Optional) A named list of the indicators' names: `careany`, `csection` and `ideliv`.

`area_labels` (Optional) A named list of the areas' names, for area data: `rural` and `urban`.

`share_label` (Optional) The text written before the private share. By default `"Priv. share = "`.

`options` A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

The function automatically detects whether the data is at the `"national"` or `"area"` level using the `"level"` attribute. It:

- Maps indicator codes to descriptive labels
- Stacks `r_raw_Public` and `r_raw_Private` values
- Adds text annotations for private sector shares

For area-level data, bars are grouped by `area` and faceted by indicator. For national-level data, each indicator has a single stacked bar.

Value

A `ggplot` object with:

- Stacked bars for public/private sector share
- Facets by indicator for area-level plots
- Annotated private sector shares as text labels

Examples

```
## Not run:
dt <- load_private_sector_data("Area estimates.dta", country_iso = "TZA", level = "area")
cs <- load_csection_estimates("csection_area.dta", country_iso = "TZA", level = "area")
plot_data <- prepare_private_sector_plot_data(dt, cs)
plot(plot_data)

## End(Not run)
```

---

plot.cd_ratios_summary

Plot ratios summary for indicator ratios summary object

Description

Generates a bar plot for a `cd_ratios_summary` object, displaying the calculated indicator ratios for each year. It allows for visual comparison across years, showing how each indicator ratio changes over time.

Usage

```
## S3 method for class 'cd_ratios_summary'
plot(
  x,
  x_labels = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_ratios_summary` object created by the `calculate_ratios_summary` function. It should contain a `year` column and one or more columns with names starting with `"Ratio"`, representing the calculated indicator ratios.
`x_labels`	Optional named list or vector of x-axis category labels, keyed by ratio name (`anc1_penta1`, `opv1_opv3`, `penta1_penta3`). Supplied entries replace the defaults.
`title`	The chart's title.
`subtitle`	The line under the title.

`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

This function provides a visual summary of indicator ratios, with each ratio displayed as a bar for each year. The `Expected Ratio` row is included if available, allowing for easy comparison of actual ratios against expected values. The bars are grouped by year, with distinct colors representing each year for clear differentiation.

Value

A `ggplot` object representing a bar plot of indicator ratios by year.

Examples

```
## Not run:
# Assuming `cd_ratios_summary` is the object returned by `calculate_ratios_summary`
plot(cd_ratios_summary)

## End(Not run)
```

---

plot.cd_service_utilization_admin1

Plot S3 method for service utilization at admin level 1

Description

Draws a horizontal bar for each admin level 1 unit showing its under-five OPD visits per child or IPD admissions per 100 children in the latest year. The bars are ordered by value and labelled with it.

Usage

```
## S3 method for class 'cd_service_utilization_admin1'
plot(
  x,
  title = NULL,
  subtitle = NULL,
```

```
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

x	Object of class cd_service_utilization_admin1, as returned by generate_admin1_service_util
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object, with the chart options applied.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()

plot(cache$generate_admin1_service_utilization("opd"))
plot(cache$generate_admin1_service_utilization("ipd"), title = "Admissions of children under five",
     x_title = "Admissions per 100 children", show_legend = FALSE)
```

---

plot.cd_service_utilization_filtered

Plot service utilization indicators

Description

Visualizes service utilization over time from a cd_service_utilization_filtered object.

Usage

```
## S3 method for class 'cd_service_utilization_filtered'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

x	A cd_service_utilization_filtered object created by select_service_utilization().
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.
legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
legend_labels	(Optional) A named list to override the legend's entries, e.g. with translations: y1 and y2, the chart's two series (for opd and ipd: under-5 and all ages). Defaults to NULL.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot2 plot object.

Examples

```
## Not run:
plot(select_service_utilization(dat, indicator = 'opd'))

## End(Not run)
```

`plot.cd_service_utilization_prepared` *Plot filtered service utilization indicators*

Description

Generates a faceted map of service utilization metrics across subnational units (admin level 1) for each available year. Uses spatial polygons filled by the selected indicator.

Usage

```
## S3 method for class 'cd_service_utilization_prepared'
plot(
  x,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

`x`	A `cd_service_utilization_prepared` object returned by `prepare_mapping_service_utlizat`
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

Reads the indicator stored on `x` (`ratio_opd_u5_pop` or `ratio_ipd_u5_pop`) and sets the data to WGS84 coordinates. Draws the units with `geom_sf()`, filled on a sequential colour scale from the palette stored on `x` (purple by default), with one panel per year and the package's map theme.

Value

A ggplot2 object representing faceted service utilization maps by year.

Examples

```
## Not run:
plot(filtered)

## End(Not run)
```

---

plot.cd_threshold *Plot target threshold attainment over time*

Description

Generates a grouped bar chart showing the percentage of administrative regions that meet specific health coverage or dropout thresholds over time.

Usage

```
## S3 method for class 'cd_threshold'
plot(
  x,
  x_labels = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  options = NULL
)
```

Arguments

x	A cd_threshold object returned by calculate_threshold().
x_labels	(Optional) A named list or vector to translate the raw indicator names on the x-axis (e.g., list(penta1 = "Penta 1", dropout_penta13 = "Penta 1-3 Dropout")).
title	The chart's title.
subtitle	The line under the title.
caption	The note under the chart.
x_title	The title of the horizontal axis.
y_title	The title of the vertical axis.

legend_title	The title of the legend.
...	Chart options by name (see cd_chart_options()): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
options	A cd_chart_options() object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Value

A ggplot object displaying a grouped bar chart of threshold attainment.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
reached <- cache$get_filtered_threshold("vaccine", "district")

plot(reached)
plot(reached, x_labels = list(bcg = "BCG", penta3 = "Penta 3", measles1 = "Measles 1"),
     title = "Districts reaching the vaccination target", y_title = "Districts (%)", legend_title = "Year
```

---

plot.cd_utilization_dqa

Plot S3 method for service DQA summary

Description

Draws the service utilization data quality summary as a formatted table, with the indicators grouped by section and a column for each year. Completeness and outlier percentages are shaded green at or above the threshold stored with the summary, yellow from 70 up to the threshold and red below 70.

Usage

```
## S3 method for class 'cd_utilization_dqa'
plot(x, years = NULL, title = NULL, width = NULL, ..., options = NULL)
```

Arguments

x	The score dataframe (needs class 'cd_utilization_dqa'), as returned by generate_service_dqa_summary().
years	Vector of years to display (e.g., 2020:2024). Required.
title	(Optional) Custom title for the table
width	(Optional) Total width in inches. If NULL, autofit is used.
...	Additional arguments

`options` (Optional) A `cd_chart_options()` object: text, legend, fonts and sizes a user changed. Applied last, so it wins over the arguments above; the plot's own defaults are used for whatever it does not set. Any chart option can also be given by name in `...`.

Value

A `flextable` object (from the flextable package), not a ggplot. Chart options change ggplots only, so `options` has no effect on this table.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
dqa <- cache$calculate_service_dqa_summary("national")

plot(dqa, years = cache$data_years)
plot(dqa, years = 2021:2023, title = "Quality of OPD and IPD data", width = 7)
```

---

plot_comparison *Internal consistency plot functions for indicator comparisons*

Description

These functions generate scatter plots that compare two health indicators across years, assessing internal consistency through visualization of linear regression fits and R-squared values by year. Each function is designed to compare a specific pair of indicators, with options for customizing plot titles, axis labels, and the inclusion of trend information.

`plot_comparison_anc1_penta1()` compares ANC1 and Penta1. Its title is "Comparison of numbers of ANC1 and Penta1 by year" and its axes are titled "ANC1" and "Penta1" unless others are given; the two functions below are the same for their pairs.

`plot_comparison_penta1_penta3()` compares Penta1 and Penta3.

`plot_comparison_opv1_opv3()` compares OPV1 and OPV3.

Usage

```
plot_comparison(
  .data,
  x_var,
  y_var,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
```

```
  ...,
  legend_labels = NULL,
  call = caller_env()
)

plot_comparison_anc1_penta1(
  .data,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)

plot_comparison_penta1_penta3(
  .data,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)

plot_comparison_opv1_opv3(
  .data,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  options = NULL
)
```

Arguments

`.data`	A data frame of class `cd_data` that includes the relevant indicators for comparison.
`x_var`	Character. The name of the indicator variable to plot on the x-axis.

`y_var`	Character. The name of the indicator variable to plot on the y-axis.
`title`	The chart's title.
`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional named list or vector overriding the legend labels. Recognised names are `district`, `linear_fit` and `diagonale`. Default is `NULL`.
`call`	The calling environment, used for error handling. Default is `caller_env()`.
`options`	A `cd_chart_options()` object; see cd_plot_arguments.

Details

- The scatter plot displays the relationship between the two specified indicators over multiple years, with each year presented in a separate facet.
- A linear regression line is added to each year's plot to help visualize the trend.
- R-squared values, calculated per year, are optionally displayed on each facet, indicating the strength of the linear relationship between the indicators for that year.
- A diagonal reference line is also provided for visual comparison to a 1:1 relationship, where applicable.

This function is commonly used for consistency checks between indicators such as:

- ANC1 vs PENTA1
- PENTA1 vs PENTA3
- OPV1 vs OPV3

Value

A `ggplot2` object, comprising:

- A scatter plot comparing two specified indicators over multiple years.
- A linear regression line for each year, indicating the trend between the indicators.
- Optional display of R-squared values per facet, summarizing the correlation strength.
- Faceting by year to facilitate a detailed comparison of trends across years.

Users can render this `ggplot2` object directly, save it, or further modify it using the `+` operator.

Examples

```
## Not run:
# Plot a comparison between ANC1 and PENTA1 indicators
plot_comparison(cd_data, x_var = "anc1", y_var = "penta1")

# Dedicated function for comparing ANC1 and PENTA1 over time
plot_comparison_penta1_anc1(cd_data)

## End(Not run)
```

---

plot_comparison.cd_data

Plot comparison with linear fit and R-squared

Description

Creates a scatter plot to compare two indicators over multiple years, including a linear regression line and a diagonal reference line. The plot is faceted by year, and displays R-squared values for each year to assess the relationship between indicators.

Usage

```
## S3 method for class 'cd_data'
plot_comparison(
  .data,
  x_var,
  y_var,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_title = NULL,
  y_title = NULL,
  legend_title = NULL,
  ...,
  legend_labels = NULL,
  call = caller_env(),
  options = NULL
)
```

Arguments

.data	A cd_data object containing merged data for the specified indicators.
x_var	Character. Name of the variable to plot on the x-axis (e.g., 'anc1').
y_var	Character. Name of the variable to plot on the y-axis (e.g., 'penta1').
title	The chart's title.

`subtitle`	The line under the title.
`caption`	The note under the chart.
`x_title`	The title of the horizontal axis.
`y_title`	The title of the vertical axis.
`legend_title`	The title of the legend.
`...`	Chart options by name (see `cd_chart_options()`): background, fonts, sizes, axes, grid, legend, colours. Other arguments are ignored.
`legend_labels`	Optional named list or vector overriding the legend labels. Recognised names are `district`, `linear_fit` and `diagonale`; unnamed entries keep their defaults. Default is `NULL`.
`call`	The calling environment.
`options`	A `cd_chart_options()` object or a list of chart options: a look made once, to give to several charts. Applied after the text arguments, so a text it holds wins over them.

Details

This function calculates R-squared values for each year and displays them in each facet, helping assess the relationship between the two indicators over time. The diagonal reference line is added to aid in visualizing deviations from a perfect 1:1 relationship.

Common indicator comparisons include:

- ANC1 vs PENTA1
- PENTA1 vs PENTA3
- OPV1 vs OPV3

Value

A ggplot2 object showing the comparison plot of two indicators with linear regression and R-squared values.

Examples

```
## Not run:
plot_comparison(cd_data, x_var = "anc1", y_var = "penta1")

## End(Not run)
```

`plot_line_graph` *Plot a line graph of several series*

Description

Draws one line with points for each of several columns of a data frame against a shared x column, in the package's plot theme. The y axis is scaled to the data: it starts at zero when all values are positive and leaves some room above the highest value.

Usage

```
plot_line_graph(
  .data,
  x,
  y_vars,
  title,
  y_title,
  x_title,
  legend_labels,
  hline = NULL,
  hline_style = "dashed",
  options = NULL,
  ...
)
```

Arguments

`.data`	A data frame.
`x`	The column name for the x-axis (e.g., "year").
`y_vars`	Vector of column names for the y-axis data.
`title`	Plot title.
`y_title`	Label for the y-axis.
`x_title`	Label for the x-axis.
`legend_labels`	Vector of labels for the legend (must match y_vars length).
`hline`	Optional numeric value for a horizontal reference line.
`hline_style`	Style of the horizontal line.
`options`	(Optional) A `cd_chart_options()` object: text, legend, fonts and sizes a user changed. Applied last, so it wins over the arguments above. Default is `NULL`.
`...`	Chart options (see `cd_chart_options()`) given by name, applied as with `options`.

Value

A ggplot object.

Examples

```
births <- data.frame(
  year = 2019:2023,
  un = c(1445, 1451, 1469, 1483, 1500),
  dhis2 = c(1390, 1402, 1455, 1478, 1461)
)
plot_line_graph(births, x = "year", y_vars = c("un", "dhis2"), title = "Live births (thousands)",
                y_title = "Live births", x_title = "Year", legend_labels = c("UN estimate", "DHIS2"))

plot_line_graph(births, x = "year", y_vars = c("un", "dhis2"), title = "Live births (thousands)",
                y_title = "Live births", x_title = "Year", legend_labels = c("UN estimate", "DHIS2"),
                hline = 1500, legend_position = "right")
```

---

plot_mortality_plausibility

Plot subnational mortality plausibility

Description

Draws one institutional mortality indicator by year, with a point for each admin level 1 unit, the median of the units in each year and the range of values considered plausible for the indicator. Use it to see whether subnational values are realistic.

Usage

```
plot_mortality_plausibility(
  x,
  indicator = c("fresh_total_sb", "ratio_md_sb", "ratio_md_nd"),
  title = NULL,
  y_title = NULL,
  x_title = NULL,
  caption = NULL,
  legend_labels = NULL
)
```

Arguments

`x`	A `cd_mortality_summary` object.
`indicator`	One of `"fresh_total_sb"` (fresh stillbirths as a percentage of all stillbirths), `"ratio_md_sb"` (stillbirths per maternal death) or `"ratio_md_nd"` (neonatal deaths per maternal death).
`title`	(Optional) Custom translated title.
`y_title`	(Optional) Custom translated y-axis label.
`x_title`	(Optional) Custom translated x-axis label.
`caption`	(Optional) Custom translated footnote.
`legend_labels`	(Optional) A named list to override default legend labels (keys: 'median', 'plausible').

Value

A ggplot object. The function stops with an error when x has no subnational values for the indicator.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()
mortality <- cache$mortality_summary

plot_mortality_plausibility(mortality, indicator = "fresh_total_sb")
plot_mortality_plausibility(mortality, indicator = "ratio_md_sb", title = "Stillbirths per maternal death
                            x_title = "Year", legend_labels = list(median = "Median of the regions"))
```

---

plot_national_health_metric

Plot national health system metrics

Description

Creates a horizontal bar plot for either health system performance or density metrics at national level.

Usage

```
plot_national_health_metric(.data, metric = c("performance", "density"))
```

Arguments

.data	A one-row data frame containing national-level metrics.
metric	Either 'performance' or 'density'.

Value

A ggplot object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()
national <- cache$health_system_metrics_national

plot_national_health_metric(national, metric = "performance")
plot_national_health_metric(national, metric = "density")
```

`prepare_mapping_service_utlization` *Filter and prepare service utilization data for mapping*

Description

Selects one under-five indicator from a `cd_service_utilization` object, optionally for given years. Joins spatial data for subnational mapping and renames geometry for use with `ggplot2::geom_sf()`.

Usage

```
prepare_mapping_service_utlization(
  .data,
  indicator = c("ipd", "opd"),
  plot_years = NULL,
  subnational_map = NULL,
  palette = c("Purples", "Blues", "Greens", "Reds", "YlGnBu")
)
```

Arguments

`.data`	A `cd_service_utilization` object returned by `compute_service_utilization()`.
`indicator`	Character. Service indicator to map, either `"ipd"` or `"opd"`. Defaults to `"ipd"`.
`plot_years`	Optional. Integer or vector of years to include.
`subnational_map`	Optional. A mapping data frame to link `NAME_1` from the shapefile to internal admin labels.
`palette`	Character. RColorBrewer palette name stored on the result for the map fill: one of `"Purples"` (default), `"Blues"`, `"Greens"`, `"Reds"` or `"YlGnBu"`.

Details

Selects the under-five service indicator (`ratio_ipd_u5_pop` or `ratio_opd_u5_pop`) and joins the admin level 1 shapefile of the country whose ISO3 code is stored on `.data`. Keeps only `plot_years` when they are given, and renames the geometry column to `geometry` for `geom_sf()`.

Only `adminlevel_1` data is currently supported for mapping.

Value

A tibble of class `cd_service_utilization_prepared`, ready for faceted spatial plotting.

Examples

```
## Not run:
prepare_mapping_service_utlization(service_data, indicator = "opd", plot_years = 2019:2022)

## End(Not run)
```

---

prepare_population_metrics

Compute population metrics for DHIS-2 and UN data

Description

`prepare_population_metrics` calculates demographic metrics such as total population, live births, growth rates, and related indicators using DHIS-2 population data. For national-level data, it optionally integrates UN estimates to provide comparative metrics. The function supports aggregation at different administrative levels, including national, admin-level_1, and district levels.

Usage

```
prepare_population_metrics(
  .data,
  admin_level = c("national", "adminlevel_1", "district"),
  un_estimates = NULL,
  region = NULL,
  show_district = TRUE
)
```

Arguments

`.data`	A `cd_data` tibble containing processed DHIS-2 health facility data with population indicators. This dataset must include columns for metrics like `total_pop`, `live_births`, and `pop_rate`.
`admin_level`	Character. Specifies the administrative level for aggregation. Available options are: `'national'`, `'adminlevel_1'`, and `'district'`. Default is `'national'`.
`un_estimates`	Optional. A tibble containing UN population estimates with columns for `un_population`, `un_births`, `un_popgrowth`, and related metrics. This parameter is only required for national-level calculations.
`region`	Optional name of an `adminlevel_1` region. If supplied, only that region's rows are used. Only valid with `admin_level = 'adminlevel_1'`. Default is `NULL`.
`show_district`	Logical. When `region` is supplied, whether to return metrics by district within the region (`TRUE`, the default) or for the region as a whole.

Value

A tibble of class `cd_population_metrics` containing demographic metrics for the specified administrative level and years. Metrics include:

- **For all levels**: DHIS-2 metrics such as `totpop_dhis2` (total population) and `totlivebirths_dhis2` (total live births).
- **For national level**: Both DHIS-2 and UN metrics for comparison.

Examples

```
## Not run:
# National-level demographic metrics
population_data <- prepare_population_metrics(
  .data = dhis2_data,
  admin_level = "national",
  un_estimates = un_data
)

# District-level demographic metrics
population_data <- prepare_population_metrics(
  .data = dhis2_data,
  admin_level = "district"
)

## End(Not run)
```

---

prepare_private_sector_plot_data

Prepare private sector plot data

Description

Processes reshaped private sector data and c-section correction estimates to produce a tidy dataset for plotting public/private sector shares.

Usage

```
prepare_private_sector_plot_data(.data, csection_data)
```

Arguments

`.data`	A data frame of class `"cd_private_sector_data"` returned by `load_private_sector_data()`. Must include `share_csection`, `r_raw`, and `sector` columns.
`csection_data`	A data frame of class `"cd_csection_estimates"` returned by `load_csection_estimates()`. Must match the level of `.data` (`"national"` or `"area"`).

Details

Joins private sector data with correction estimates using `iso`, `year`, and `indic` (and `area` for area data). For `"csection"` rows, applies the adjustment `r_raw = estimate * share_csection`. Drops intermediate columns, reshapes the data to one row per indicator with a set of columns for each sector, and calculates private sector share and total prevalence.

Used as input to `plot.cd_private_sector_plot_data()` for visualisation.

Value

A tibble of class `"cd_private_sector_plot_data"` with attributes:

- `level`: `"national"` or `"area"`
- Columns reshaped from wide to include `r_raw_Public`, `r_raw_Private`, etc.
- Computed columns:
  - `privateShare`: percent share of private sector
  - `totalPrevalence`: total prevalence (public + private)

Examples

```
## Not run:
dt <- load_private_sector_data("National estimates.dta", country_iso = "KEN", level = "national")
cs <- load_csection_estimates("csection_national.dta", country_iso = "KEN", level = "national")
plot_data <- prepare_private_sector_plot_data(dt, cs)

## End(Not run)
```

---

print.cd_data *Print method for* `cd_data` *class*

Description

Prints a summary of an object of class `cd_data`, including data dimensions, available years, regions, variable groups, and any quality metrics associated with the data. It uses the `cli` package for enhanced, styled output.

Usage

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

Arguments

`x`	An object of class `cd_data`, which contains `merged_data` (a dataframe with demographic data) and optional `quality_metrics` (a dataframe or list of quality metrics).
`...`	Additional arguments (currently not used).

Details

This print method provides the following summary sections:

1. **Data Summary**: Shows the number of observations and variables in `merged_data`.
2. **Years Available**: Lists unique years present in `merged_data`.
3. **Regions Available**: Lists unique regions in the `adminlevel_1` column.
4. **Variable Groups**: Displays predefined groups of variables, including ANC, delivery, vaccination, PNC, OPD, and IPD groups.
5. **Quality Metrics**: If present, prints `quality_metrics` for additional data insights.

Value

Invisibly returns `x`. This function is called for its side effect of printing a formatted summary of the `cd_data` object.

Variable Groups

- `ancvargroup`: Variables related to antenatal care.
- `idelvvargroup`: Variables related to delivery.
- `vaccvargroup`: Variables related to vaccinations.
- `pncvargroup`: Variables related to postnatal care.
- `opdvargroup`: Variables related to outpatient care.
- `ipdvargroup`: Variables related to inpatient care.

Examples

```
## Not run:
# Assuming `cd_data_obj` is an object of class `cd_data`:
print(cd_data_obj)

## End(Not run)
```

`push_upload_on_remount` *Keep an upload zone showing the file the dataset already holds*

Description

Sets up, inside a module server, an observer that puts a file upload zone back in its "uploaded" state each time the zone appears on the page, when the dataset holds something the user uploaded for the zone's field (`cache()$is_default(field)` is `FALSE`). The zone shows the name of the file uploaded in this session, or a general "file on record" label when the name is not known (for example after loading a saved dataset).

Usage

```
push_upload_on_remount(
  input,
  session,
  i18n,
  input_id,
  field,
  cache,
  filename_val
)
```

Arguments

`input`, `session`	The `input` and `session` objects of the calling module server.
`i18n`	The app's translator (a `shiny.i18n` translator), used for the label shown when the file's name is not known.
`input_id`	The input id of the upload zone, without the namespace, a string.
`field`	The name of the dataset field the zone fills, a string (for example `"un_estimates"`), as `cache()$is_default()` takes it.
`cache`	A reactive that returns the dataset, a CacheConnection.
`filename_val`	A reactive value (`reactiveVal()`) that holds the name of the file uploaded in this session, or `NULL`. The caller sets it after an upload and sets it to `NULL` when the zone is reset.

Value

Nothing useful. The function is called for its side effect of registering the observer.

Examples

```
## Not run:
# in a module's ui: the upload zone
cd_file_upload(ns("un_data"), label = "title_upload_un_estimates", accept = ".dta", i18n = i18n)

# in its server: remember the uploaded file's name, and show it when the zone comes back
un_filename <- reactiveVal(NULL)
observeEvent(input$un_data, un_filename(input$un_data$name))
observeEvent(input$un_data_reset, un_filename(NULL))
push_upload_on_remount(input, session, i18n, "un_data", "un_estimates", cache, un_filename)

## End(Not run)
```

quality_issues_table *Tabulate data quality results*

Description

Turns the results of `run_all_quality_checks()` into two tables: a summary with one row per check, and a list with one row per issue found.

Usage

```
quality_issues_table(results, labels = NULL)
```

Arguments

`results`	A named list. What `run_all_quality_checks()` returns.
`labels`	An optional named character vector that maps a check's key (`names(results)`) to the name to show for it, for example a translated name. A check without a label gets a plain English name.

Value

A list of two tibbles. `summary` has columns `check`, `severity`, `status` (`"passed"` or `"issue"`) and `issues` (the number of issues). `issues` has columns `check`, `severity` and `issue`, with one row for each thing a check found (each district-year, each name, each message) and nothing shortened.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
results <- run_all_quality_checks(cache)
tables <- quality_issues_table(results)
tables$summary
tables$issues

# A check under a name of your own
quality_issues_table(results, labels = c(indicator_emptiness = "Indicators without data"))$summary
```

read_shapefile_folder *Read a user-uploaded shapefile folder*

Description

Reads a shapefile from the files of an uploaded folder, as a Shiny multi-file or folder `fileInput()` gives them: a data frame with each file's original `name` and the `datapath` it was saved to. The `.shp`, `.dbf`, `.shx` and `.prj` files are all required; they are copied under their original names into one temporary directory and read from there. Other files are ignored.

Usage

```
read_shapefile_folder(files_df, call = caller_env())
```

Arguments

`files_df`	A data frame with `name` and `datapath` columns, one row per uploaded file.
`call`	The calling environment, forwarded to `cd_abort()`.

Value

The uploaded shapefile as a valid `sf` object, reprojected to EPSG:4326. Column names are exactly as uploaded: the column holding the unit names is not renamed to `NAME_1`.

Examples

```
## Not run:
# in a Shiny server, for fileInput("shapefile", "Shapefile", multiple = TRUE) in the page
boundaries <- read_shapefile_folder(input$shapefile)

# outside Shiny: the parts of a shapefile kept in one folder
parts <- list.files("boundaries", full.names = TRUE)
boundaries <- read_shapefile_folder(data.frame(name = basename(parts), datapath = parts))
names(boundaries)

## End(Not run)
```

`reference_estimates_ui`

Reference-data uploads of the wizard's Upload Data step

Description

Optional upload zones for the reference estimates the analysis uses: UN estimates, WUENIC estimates and UN mortality estimates, each a Stata file (`.dta`). Which zones show is set by the app (`cd_wizard_config()$reference_uploads`). The dataset uses built-in estimates for its country until the user uploads a file. An uploaded file is read for the dataset's country and saved with `cache()$set_un_estimates()`, `cache()$set_wuenic_estimates()` or `cache()$set_un_mortality_estimates()`. A zone's reset button goes back to the built-in estimates (`cache()$clear_un_estimates()` and so on). A file that cannot be read shows an error banner under its zone. These uploads do not affect whether the Upload Data step counts as complete.

Usage

```
reference_estimates_ui(id, i18n)

reference_estimates_server(id, cache, i18n)
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.

Details

The UI has no card of its own: it is a divided section meant to sit in the same card as upload_box_ui(). UN estimates are read for the years of the dataset (cache()$data_years), so that zone does nothing until the dataset has years.

Value

reference_estimates_ui() returns the UI as Shiny tags: a divider and a group of upload zones. reference_estimates_server() is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen: the Upload Data step's page is one card
# holding the dataset's upload zone and these reference-data zones
panel <- list(key = "upload", ui = cd_card(
  title = i18n$t("title_upload_dataset_card"), width = 12,
  upload_box_ui(ns("upload_box"), i18n),
  reference_estimates_ui(ns("reference_estimates"), i18n)
))
reference_estimates_server("reference_estimates", cache, i18n)

## End(Not run)
```

---

register_indicator_group

Register or override an indicator group (profile)

Description

Adds or modifies a group definition by **name**. If the name already exists, use on_conflict = "replace" to replace the entire definition, or on_conflict = "merge" to **union** indicators *per category*.

Usage

```
register_indicator_group(
  name,
  value,
  on_conflict = c("replace", "merge", "error")
)
```

Arguments

`name`	A non-empty string: the group/profile name to create or modify.
`value`	A **named list**: each element is a category containing a character vector of indicator codes.
`on_conflict`	One of `"replace"`, `"merge"`, or `"error"`. `"merge"` performs a union per category (`sort(unique(c(old, new)))`).

Details

Reserved names `"auto"` and `"custom"` cannot be registered.

- If `name` matches a built-in (e.g. `"rmncah"`), the override will be layered on top of the built-in according to `on_conflict`.
- The merged, live view is available via `.get_all_groups()` or `get_indicator_groups()`.

Value

Invisibly returns `name`.

Examples

```
# Replace RMNCAH completely
register_indicator_group(
  "rmncah",
  list(anc = c("anc1","anc4"), idelv = c("sba")),
  on_conflict = "replace"
)

# Extend RMNCAH by merging into existing categories (union)
register_indicator_group(
  "rmncah",
  list(hiv = c("hiv_test","pmtct1")),
  on_conflict = "merge"
)

# Back to the built-in group
reset_indicator_group("rmncah")
```

`report_block_kinds` *The charts and tables a report can contain*

Description

What each kind draws and which settings it takes. The builder offers these; `render_report_block()` draws them.

Usage

```
report_block_kinds(group = get_selected_group())
```

Arguments

`group`	The indicator group; kinds that only make sense for one group (`groups`) are left out of the other.

Details

The kinds are listed in the order of the analysis (their `group`: data quality, adjustment, denominators, national, sub-national, region, mortality, service utilization, health system). The standard report files add their own kinds (see the top of the standard reports section).

Value

A named list; each entry has `type` (`"chart"` or `"table"`), `group`, `label`, `indicators` (`"analysis"`, a character vector, or `NULL` when the kind has no indicator), `levels`, `variants` (named character vector or `NULL`), `year` (whether a year is chosen), `regional` (whether it is drawn for one region) and `groups` (the indicator groups it is for).

Examples

```
kinds <- report_block_kinds("rmncah")
head(names(kinds))
str(kinds$reporting_rate)

# the kinds the vaccine group does not have
setdiff(names(kinds), names(report_block_kinds("vaccine")))
```

report_presets *The standard reports*

Description

One per analysis section (data quality, adjustment, denominators, national coverage, inequality, mortality, service utilization, health system, private sector), then the synthesis chartbook and the sub-national one-pager: the Countdown report templates rebuilt as blocks, which the builder opens as editable copies. The one-pager is about one region: its blocks use the report's region (`region = "@report"`). It is a slide deck of one dense slide.

Usage

```
report_presets(lang = "en", group = get_selected_group())
```

Arguments

`lang`	The language of the report's text: `"en"`, `"fr"` or `"pt"`.
`group`	The indicator group (`"rmncah"` or `"vaccine"`); each has its own list.

Value

A named list of reports in the order of the analysis, each `list(name, description, kind, design, cover, blocks)` for a document (`kind = "document"`), or `list(name, description, kind = "deck", design, cover, slides)` for a slide deck (see `export_deck()`). `report_project_blocks()` gives the blocks of either.

Examples

```
presets <- report_presets(lang = "en", group = "rmncah")
names(presets)
presets$mortality$name
length(presets$mortality$blocks)

# the vaccine group's reports, named in French
vapply(report_presets("fr", group = "vaccine"), function(p) p$name, character(1))
```

reporting_rate_ui *Reporting rate page*

Description

The data quality page on how completely facilities report. Its first card shows the reporting rate of the chosen indicator for each admin level 1 area or district, as a heat map or as a bar chart (two tabs). Below it are a chart of the district reporting rates, for the country or for the chosen region, and a table of the areas whose rate is below the performance threshold in the year picked on the table. The filters are the indicator (the app's `reporting_rate_indicators` setting), the performance threshold and the admin level with its region.

Usage

```
reporting_rate_ui(id, i18n)

reporting_rate_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Details

The page reads `reporting_rate_admin1`, `reporting_rate_district` and `district_reporting_rate` from the dataset, and calls its `calculate_reporting_rate()` and `calculate_district_reporting_rate()` when a region is chosen. The performance threshold comes from the dataset's `performance_threshold`, and a change to it is saved there with `set_performance_threshold()`.

Value

`reporting_rate_ui()` returns the page's UI as Shiny tags. `reporting_rate_server()` is called for its side effects.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "reporting_rate",
  ui = reporting_rate_ui,
  server = reporting_rate_server,
  title = "title_rr_main",
  section = "lbl_nav_section_quality",
  subtitle = "sub_rr_main"
)

## End(Not run)
```

`reset_indicator_group`
Reset (remove) a group override

Description

Removes a user override for `name`. If `name` matches a built-in group, the built-in remains intact; only the override layer is removed.

Usage

```
reset_indicator_group(name)
```

Arguments

`name`	Group name whose override should be removed.

Value

Invisibly returns `name`.

Examples

```
reset_indicator_group("rmncah")
```

`resolve_country_best_effort`
Find the country of an Admin sheet without raising an error

Description

Looks up the country named in the Admin sheet and returns its name and ISO3 code. Unlike the lookup in `new_countdown()`, it does not stop with an error when the country cannot be resolved. Use it to show the country before the sheets are merged.

Usage

```
resolve_country_best_effort(admin_data)
```

Arguments

`admin_data`	A data frame. The cleaned Admin sheet, before the sheets are merged.

Details

The country is resolved only when the `country` column holds exactly one distinct value, ignoring missing values, and that value is close to a name in the package's list of countries. This function does not validate the sheet. Use `check_single_country()` and `check_country_recognized()` for that.

Value

A list with two elements: `country` (the country's name as the package spells it) and `country_iso` (its ISO3 code), both strings. Both are `NULL` if the sheet has no `country` column or the country cannot be resolved.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
sheets <- load_excel_parts(path)
admin <- sheets$parts[[sheets$admin_sheet_name]]
resolve_country_best_effort(admin)

# Nothing is returned, and no error raised, for a country that is not recognized
admin$country <- "Atlantis"
resolve_country_best_effort(admin)
```

`restore_default_control` *A link that clears something the user set in the dataset*

Description

Sets up, inside a module server, a link button that shows only while a field of the dataset holds something the user set (`cache()$is_default(field)` is `FALSE`), and that calls `clear_fn()` when pressed. The two mapping steps use it for their "clear mapping" link. The module's UI needs `uiOutput(ns(output_id))` where the link goes.

Usage

```
restore_default_control(
  ns,
  output,
  input,
  output_id,
  field,
  cache,
  clear_fn,
  label_key,
  i18n
)
```

Arguments

`ns`	The namespace function of the calling module (`session$ns`).
`output`, `input`	The `output` and `input` objects of the calling module server.
`output_id`	The id of the output that shows the link, a string. The button's own input id is the same.
`field`	The name of the dataset field to watch, a string (for example `"survey_mapping"`), as `cache()$is_default()` takes it.
`cache`	A reactive that returns the dataset, a CacheConnection.
`clear_fn`	A function without arguments that clears the field, for example `function() cache()$clear_survey_mapping()`.
`label_key`	The translation key of the link's text, a string.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.

Value

Nothing useful. The function is called for its side effects: it sets `output[[output_id]]` and registers an observer.

Examples

```
## Not run:
# in a module's ui: where the link goes
uiOutput(ns("survey_map_restore"))

# in its server: a "clear mapping" link that shows while a survey mapping is saved
restore_default_control(
  session$ns, output, input, "survey_map_restore", "survey_mapping", cache,
  clear_fn = function() cache()$clear_survey_mapping(),
  label_key = "btn_upload_clear_mapping", i18n = i18n
)

## End(Not run)
```

`robust_max` *Robust maximum value calculation*

Description

`robust_max` calculates the maximum value of a numeric vector while handling missing values. If all values in the vector are `NA`, the function returns `NA` instead of raising an error.

Usage

```
robust_max(x, fallback = NA)
```

Arguments

x	A numeric vector.
fallback	argument allows you to specify a value to return when all elements are NA

Value

The maximum value of x, or NA if all values are NA.

Examples

```
# Example usage
robust_max(c(1, 2, 3, NA)) # Returns 3
robust_max(c(NA, NA)) # Returns NA
robust_max(c(-Inf, 0, 10)) # Returns 10
```

---

rr_indicators *Tabs of the reporting rate chart*

Description

The keys of the tabs of the first card on the reporting rate page (reporting_rate_ui()): the heat map and the bar chart.

Usage

```
rr_indicators
```

Format

A character vector.

Examples

```
rr_indicators
```

`run_all_quality_checks` *Run all data quality checks and report every result*

Description

Runs the data quality checks on the data held in a cache and returns the result of each one, whether it passed or not. It does not stop on a problem, so a caller can show the whole report.

Usage

```
run_all_quality_checks(cache)
```

Arguments

`cache`	A `CacheConnection`, or an object with the same `wizard_parts`, `wizard_quality_results` and `countdown_data` fields and `check_survey_admin_names()` and `check_shapefile_admin_na` methods.

Details

Which checks run depends on what the cache holds:

- If it holds unmerged sheets (`cache$wizard_parts`), the checks run on the separate sheets, so a problem can be traced to the sheet it comes from.
- Otherwise, if it holds results saved when the sheets were merged (`cache$wizard_quality_results`), those saved results are returned and no check is run.
- Otherwise the checks run on the merged dataset (`cache$countdown_data`). This set has fewer checks than the one for unmerged sheets.

Value

A named list with one element per check. Each element is a list with:

- `ok`: `TRUE` if the check passed.
- `severity`: `"blocking"` (the caller should not go on until the problem is fixed), `"informational"` (shown, but does not block) or `"mapping"` (an admin level 1 name with no match in the survey or the shapefile; shown, but does not block).
- `detail`: what the check returned, which is `NULL`, a character vector or a tibble, depending on the check.

See Also

`quality_issues_table()` and `write_quality_issues()` to tabulate or save the result.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")

# On a loaded dataset
cache <- load_cache_data(path)
results <- run_all_quality_checks(cache)
names(results)
results$indicator_emptiness

# On the sheets of a workbook, before they are merged
sheets <- load_excel_parts(path)
results <- run_all_quality_checks(init_CacheConnection(wizard_parts = sheets))
Filter(function(result) !result$ok, results)
```

---

save_data *Save processed Countdown 2030 data to file*

Description

`save_data` saves processed Countdown 2030 data to either an RDS or DTA file. It checks that the input data is of class `cd_data` before saving.

Usage

```
save_data(.data, file = "master_dataset.dta")
```

Arguments

`.data`	A tibble of class `cd_data` containing the processed data to be saved.
`file`	A string. The path to the file where the data should be saved, with extension `.rds` or `.dta` depending on the desired format. Default is `master_dataset.dta`

Details

The function verifies that the `data` argument is of class `cd_data`. It then saves the data in the specified file format based on the file extension.

Value

No return value. This function is called for its side effects (saving the data to an RDS or DTA file).

Examples

```
## Not run:
# Save processed Countdown 2030 data to an RDS file
save_data(cd_data, "processed_cd2030_data.rds")

# Save processed Countdown 2030 data to a DTA file
save_data(cd_data, "processed_cd2030_data.dta")

## End(Not run)
```

---

save_dhis2_excel *Save DHIS2 data to an Excel workbook*

Description

Writes the admin, population, completeness and service data downloaded from DHIS2 to one Excel workbook, with styled sheets for each type of data. A file that already exists at `filename` is overwritten.

Usage

```
save_dhis2_excel(.data, filename, last_org)
```

Arguments

`.data`	A list containing the structured DHIS2 data, including `admin`, `population`, `completeness`, and `service` datasets.
`filename`	A string specifying the file path and name where the Excel workbook will be saved.
`last_org`	Character. Name of the lowest organisation-unit column (for example `"district"`). Used as the key header column on every sheet, followed by `year` and `month` where relevant.

Details

The workbook contains sheets for the different types of DHIS2 data:

- **Admin Data**: Maps health districts to administrative units.
- **Population Data**: Contains official population data for denominators in health reporting.
- **Completeness Data**: Tracks reporting completeness metrics.
- **Service Data**: Records service delivery numbers.

Each sheet is styled, with column headers, instructions, and frozen panes. Instructions are added to guide users on how to input or interpret data.

Value

Saves the workbook to the specified filename.

Examples

```
## Not run:
save_dhis2_excel(data, "dhis2_data.xlsx")

## End(Not run)
```

---

save_dhis2_master_data

Create a master dataset from DHIS2 data

Description

Combines the service, completeness, population and admin data downloaded from DHIS2 into one Countdown dataset, with a row per district, year and month and a column per data element.

Usage

```
save_dhis2_master_data(.data)
```

Arguments

`.data`	A list containing the structured DHIS2 data, including `admin`, `population`, `completeness`, and `service` datasets.

Details

Pivots the service, completeness and population data to a wide format and merges them on their common keys (`district`, `year` and, where present, `month`), then adds the admin data by `district`. The merged data is standardized and made into a `cd_data` object, which checks that the required columns are present.

Value

A tibble of class `cd_data` holding the combined and standardized DHIS2 dataset.

Examples

```
## Not run:
master_dataset <- save_dhis2_master_data(data)

## End(Not run)
```

select_adjustment_value

Select one indicator's adjustment values

Description

Keeps `year` and the two columns for a single indicator's raw and adjusted values, e.g. `anc1_raw` and `anc1_adj`.

Usage

```
select_adjustment_value(.data, indicator)
```

Arguments

`.data`	A tibble of class `cd_adjustment_values`.
`indicator`	A single indicator name. Must be one of `get_all_indicators()`.

Value

A tibble with class `cd_adjustment_values_filtered` containing:

- `year`
- `<indicator>_raw`
- `<indicator>_adj`

The result carries an attribute `indicator` with the selected indicator.

Examples

```
## Not run:
x <- select_adjustment_value(adj_values, "anc1")
attr(x, "indicator")   # "anc1"

## End(Not run)
```

select_coverage

Select one indicator's coverage

Description

`select_coverage` extracts specific coverage indicators and denominators from a combined coverage dataset, applies regional filtering, and reshapes the data for analysis and visualisation.

Usage

```
select_coverage(
  .data,
  indicator,
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived"),
  region = NULL
)
```

Arguments

`.data`	A `cd_coverage` data frame with combined immunization coverage data.
`indicator`	Coverage indicators to include (e.g., `"penta3"`, `"measles1"`).
`denominator`	Denominator sources for coverage calculations (e.g., `"dhis2"`).
`region`	Name of the subnational unit (at the data's `admin_level`) to keep. Required for subnational data that was not already calculated for a single region; must be `NULL` for national data or data calculated for a region. Default is `NULL`.

Value

A reshaped data frame with filtered indicators and denominators.

Examples

```
## Not run:
select_coverage(combined_data, c("penta3", "measles1"), "dhis2", "Central Province")

## End(Not run)
```

`select_inequality` *Select one indicator's inequality metrics*

Description

`select_inequality` refines the output of `calculate_inequality` by selecting specific indicators and denominators for analysis. It extracts and renames relevant columns to streamline further analyses.

Usage

```
select_inequality(
  .data,
  indicator,
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived")
)
```

Arguments

.data	A cd_inequality tibble created by calculate_inequality.
indicator	A character vector of health indicators to include (e.g., "penta3", "measles1").
denominator	A character vector of denominators to filter by (e.g., "dhis2", "anc1").

Value

A tibble containing filtered subnational inequality metrics for the specified indicators and denominators.

Examples

```
## Not run:
# Filter for Penta-3 coverage using DHIS-2 denominator
filtered_data <- inequality_metrics %>%
  select_inequality(indicator = "penta3", denominator = "dhis2")

## End(Not run)
```

---

select_mapping_data *Select one indicator's mapping data*

Description

Extracts and filters indicator coverage data by year, denominator type, and color palette for visualization.

Usage

```
select_mapping_data(
  .data,
  indicator,
  denominator = c("dhis2", "anc1", "penta1", "penta1derived", "anc1derived"),
  palette = c("Reds", "Blues", "Greens", "Purples", "YlGnBu"),
  plot_year = NULL
)
```

Arguments

.data	A cd_mapping object returned by get_mapping_data().
indicator	Character. Indicator name (e.g., "anc4", "penta1").
denominator	Character. One of: "dhis2", "anc1", "penta1", "penta1derived", "anc1derived".
palette	Character. Color palette for mapping. One of: "Reds", "Blues", "Greens", "Purples", "YlGnBu".
plot_year	Optional integer or vector of years to filter.

Value

A `tibble` of class `"cd_mapping_filtered"`.

Examples

```
## Not run:
filtered <- select_mapping_data(
  data, indicator = "anc4", denominator = "anc1", palette = "Blues", plot_year = 2021
)

## End(Not run)
```

`select_mortality_summary` *Select one mortality rate, ready to map*

Description

Selects one institutional mortality rate from a `cd_mortality_summary` object, optionally for given years, and joins the admin level 1 boundaries of the country so the result can be mapped.

Usage

```
select_mortality_summary(
  .data,
  country_iso,
  indicator = c("mmr", "sbr", "nn"),
  plot_year = NULL,
  subnational_map = NULL,
  palette = c("Reds", "Blues", "Greens", "Purples", "YlGnBu")
)
```

Arguments

`.data`	A `cd_mortality_summary` object created by `create_mortality_summary()`.
`country_iso`	A character string. ISO3 code of the country, used to load its boundaries.
`indicator`	Character. Mortality indicator to select: `"mmr"`, `"sbr"` or `"nn"`. Defaults to `"mmr"`.
`plot_year`	Optional integer or vector of years to filter.
`subnational_map`	Optional. A data frame to join with the shapefile to add metadata.
`palette`	Character. RColorBrewer sequential palette for the map (default `"Reds"`).

Details

Selects the column of the mortality indicator (`mmr_inst`, `sbr_inst` or `nn_inst`) and keeps only `plot_year` when it is given. Joins the rates to the admin level 1 boundaries by unit name, or through `subnational_map` when the names in the data differ from those of the shapefile, and renames the geometry column to `geometry`.

Value

A tibble of class `cd_mortality_summary_filtered`, ready for geospatial plotting.

Examples

```
## Not run:
select_mortality_summary(
  mortality_data,
  country_iso = "NGA",
  indicator = "sbr",
  plot_year = 2021:2023
)

## End(Not run)
```

---

select_service_utilization

Select one service utilization indicator

Description

Prepares a `cd_service_utilization` object for non-spatial plotting (e.g., time series) by filtering by administrative region and indicator type.

Usage

```
select_service_utilization(
  .data,
  indicator = c("opd", "ipd", "under5", "cfr", "deaths"),
  region = NULL
)
```

Arguments

`.data`	A `cd_service_utilization` object created by `compute_service_utilization()`.
`indicator`	Character. The indicator to visualize. Options include `"opd"`, `"ipd"`, `"under5"` (proportion under 5), `"cfr"` (case fatality rate), or `"deaths"` (proportion of under-5 deaths).
`region`	A single unit name to filter by. Required when data is subnational (`adminlevel_1` or `district`); must be `NULL` for national data.

Details

Checks the class of `.data` and the `indicator` value. Subnational data is filtered to the unit named by `region`; for national data `region` must not be given. The result keeps every column of `.data` and is tagged with the `cd_service_utilization_filtered` class for plotting.

Value

A tibble of class `cd_service_utilization_filtered`, with an attached `indicator` attribute for plotting.

Examples

```
## Not run:
# Filter IPD indicator for Central region
filtered <- select_service_utilization(service_data, indicator = "ipd", region = "Central")

## End(Not run)
```

`set_selected_group` *Set the selected indicator group globally*

Description

Persists the selected group in session state and as an option (`options(cd2030.selected_group = <name>)`). Downstream helpers can retrieve it with `get_selected_group()`.

Usage

```
set_selected_group(group)
```

Arguments

`group`	Group name. Must exist in `.get_all_groups()`.

Value

Invisibly returns the resolved group name.

Examples

```
set_selected_group("rmncah")
get_selected_group()
```

`shapefile_step_ui` *Shapefile step of the Load Data wizard*

Description

An optional step where the user can replace the built-in map of the country with their own shapefile. The user uploads the shapefile's folder (its `.shp`, `.dbf`, `.shx` and `.prj` files), picks the column that holds the admin level 1 names (`NAME_1` is preselected when the file has it), and confirms. Confirming saves the shapefile with `cache()$set_shapefile()` and the column with `cache()$set_shapefile_name_field()`; nothing is saved before that. The zone's reset button goes back to the built-in shapefile (`cache()$clear_shapefile()`). The step never blocks the wizard while the built-in shapefile is in use (see `step_shapefile_complete()`).

Usage

```
shapefile_step_ui(id, i18n)

shapefile_step_server(id, cache, i18n)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.

Details

Under the upload zone the step shows: an error banner when the folder cannot be read; a banner that says whether the built-in or a custom shapefile is in use; and a warning with the number of admin level 1 names of the dataset that do not match the shapefile (`cache()$check_shapefile_admin_names()`), which the Map Shapefile Names step resolves.

Value

`shapefile_step_ui()` returns the UI as Shiny tags: a card with the upload zone. `shapefile_step_server()` is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "shapefile", ui = shapefile_step_ui(ns("shapefile_step"), i18n))
shapefile_step_server("shapefile_step", cache, i18n)

## End(Not run)
```

slim_bayes_model — *A Bayesian model without the sampler's draws and the compiled Stan model*

Description

What makes a fitted model large (`samples`, `stan_model`) and no chart or table reads: see `generate_bayes_model()`'s `keep_samples`. A model that has neither is returned as it is.

Usage

```
slim_bayes_model(model)
```

Arguments

`model`	A fitted model.

Value

The model without them, its class and attributes kept.

Examples

```
# a stand-in for a model fitted with keep_samples = TRUE
model <- structure(
  list(posteriors = list(), samples = matrix(0, 1000, 50), stan_model = "the compiled model"),
  class = c("cd_bayes_model", "list"),
  indicator = "penta3"
)
slim <- slim_bayes_model(model)
names(slim)
attr(slim, "indicator")
object.size(slim) < object.size(model)
```

step_upload_complete — *Tests of whether a Load Data wizard step is complete*

Description

Each function looks at a dataset and says whether one step of the Load Data wizard is done. The wizard uses them, through wizard_step_defs and `compute_step_states()`, to decide which steps are open and when Continue is enabled. `step_shapefile_touched()` is not a completion test: it says whether the user has done anything on the Shapefile step, which decides whether that step's button reads "Skip" or "Continue".

Usage

```
step_upload_complete(cd)

step_quality_complete(cd)

step_national_rates_complete(cd)

step_survey_files_complete(cd)

step_shapefile_complete(cd)

step_shapefile_touched(cd)

step_survey_mapping_complete(cd)

step_map_mapping_complete(cd)
```

Arguments

`cd`	The dataset, a CacheConnection object (the value of `cache()`, not the reactive), or `NULL` when no dataset is loaded.

Details

The survey files, the reference estimates and the shapefile have built-in data for the dataset's country, so reading them from the dataset never gives `NULL`. The tests for those steps therefore use `cd$is_default()`, which says whether the user has uploaded something. The two name mappings have no built-in data, so their tests check for `NULL`: a mapping exists only after the user has saved the matching dialog at least once.

Value

`TRUE` or `FALSE`. Every function returns `FALSE` for a `NULL` dataset, except `step_shapefile_complete()`, which returns `TRUE`. Otherwise each returns `TRUE` when:

- `step_upload_complete()`: the dataset holds unmerged sheets (`cd$wizard_parts`) or merged data with at least one row (`cd$countdown_data`).
- `step_quality_complete()`: the upload is complete and every check of severity `"blocking"` in `run_all_quality_checks()` has passed.
- `step_national_rates_complete()`: every national rate named by `national_rates_required_fields()` and every survey estimate named by `survey_estimates_required_fields()` has a value, and `cd$survey_year` is set.
- `step_survey_files_complete()`: the user has uploaded all five survey files (survey_fields).
- `step_shapefile_complete()`: the built-in shapefile is in use, or the user has uploaded one and `cd$shapefile_name_field` is set.
- `step_shapefile_touched()`: the user has uploaded a shapefile.

- `step_survey_mapping_complete()`: a survey name mapping is saved (`cd$survey_mapping`).
- `step_map_mapping_complete()`: a shapefile name mapping is saved (`cd$map_mapping`).

Examples

```
old <- options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

step_upload_complete(cache)
step_quality_complete(cache)
step_national_rates_complete(cache)
step_survey_files_complete(cache)
step_shapefile_complete(cache)
step_shapefile_touched(cache)
step_survey_mapping_complete(cache)
step_map_mapping_complete(cache)

# no dataset loaded
step_upload_complete(NULL)

options(old)
```

`subnational_coverage_ui` *Sub-national coverage page*

Description

Shows coverage by year for one region or one district: the estimate from the health facility data, with the denominator selected for the indicator, beside the survey and WUENIC estimates. Two filters choose the admin level (admin level 1 or district) and then the area at that level; the charts wait until an area is chosen. The page is the coverage charts (`coverage_ui()`) for that area: one card with a tab for each indicator in `cd_cfg("nat_cov_indicators")` and a Custom tab. The data come from the dataset's `get_filtered_coverage()`, which reads `cache()$calculate_coverage()` for the chosen level.

Usage

```
subnational_coverage_ui(id, i18n)

subnational_coverage_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

id	The module's id. A page's id is also its entry in the app's page registry.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.
active	A reactive that is TRUE once the page has been opened with data loaded. The page does no work before that.

Value

subnational_coverage_ui() returns the page's UI as Shiny tags. subnational_coverage_server() returns what coverage_server() returns: a reactive that holds the key of the tab shown.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "subnational_coverage",
  ui = subnational_coverage_ui,
  server = subnational_coverage_server,
  title = "title_nav_subnational_coverage",
  section = "title_nav_subnational_analysis",
  subtitle = "sub_cov_subnational"
)

## End(Not run)
```

---

subnational_denominator_ui

Sub-national coverage by denominator card

Description

A card of the Denominator selection page (denominator_selection_ui()) with one tab per indicator (the app's sub_derived_indicators setting). Each tab has a chart of the indicator's coverage in each admin level 1 area, for each denominator the app offers. The card has no filters. It calls the dataset's calculate_derived_coverage() for admin level 1 and reads its survey_year, which the chart's title names.

Usage

```
subnational_denominator_ui(id, i18n)

subnational_denominator_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

id	The module's id.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.
active	A reactive that is TRUE once the page has been opened with data loaded. The card does no work before that.

Value

subnational_denominator_ui() returns the card's UI as Shiny tags. subnational_denominator_server() is called for its side effects.

---

subnational_inequality_ui

Sub-national inequality page

Description

Shows how coverage differs between districts. A filter chooses the admin level. With admin level 1, a second filter chooses a region, and the charts show the coverage of that region's districts; they wait until a region is chosen. With district, there is no second filter and the charts show every district of the country. The page is the inequality charts (inequality_ui()), which read cache()$inequality_district through the dataset's get_filtered_inequality(); that table measures each district against the national value.

Usage

```
subnational_inequality_ui(id, i18n)

subnational_inequality_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

id	The module's id. A page's id is also its entry in the app's page registry.
i18n	The app's translator (a shiny.i18n translator), used for every label.
cache	A reactive that returns the dataset, a CacheConnection.
active	A reactive that is TRUE once the page has been opened with data loaded. The page does no work before that.

Value

subnational_inequality_ui() returns the page's UI as Shiny tags. subnational_inequality_server() returns what inequality_server() returns: a reactive that holds the key of the tab shown.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "subnational_inequality",
  ui = subnational_inequality_ui,
  server = subnational_inequality_server,
  title = "title_inequ_subnational",
  section = "title_nav_subnational_analysis",
  subtitle = "sub_inequ_subnational"
)

## End(Not run)
```

---

subnational_mapping_ui

Sub-national coverage maps

Description

The maps of the national inequality page: one card with a tab for each of the app's default indicators (`cd_default_indicator_set()`) and a Custom tab for any other analysis indicator. A tab draws maps of the indicator's coverage in each admin level 1 area, with the denominator selected for the indicator, for the years chosen for mapping (`cache()$mapping_years`). It reads the dataset's `get_filtered_mapping_data()`, which joins the admin level 1 coverage to the map through the dataset's map mapping. The piece has no filters of its own: the page gives it the colour palette and the tab to show.

Usage

```
subnational_mapping_ui(id, i18n)

subnational_mapping_server(
  id,
  cache,
  i18n,
  palette,
  selected_tab,
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.

`palette`	A reactive that returns the name of the maps' colour palette (for example `"Greens"`).
`selected_tab`	A reactive that returns the key of the tab to show, or `NULL` to leave the tabs as they are. When its value changes, the maps switch to that tab.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The maps do no work before that.

Value

`subnational_mapping_ui()` returns the card's UI as Shiny tags. `subnational_mapping_server()` returns a reactive that holds the key of the tab shown: an indicator's name, or `"custom"`.

Examples

```
## Not run:
# the maps below the inequality charts, showing the tab the charts show
my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n,
    inequality_ui(ns("inequality"), i18n),
    cd_map_options(cd_palette_chip(ns("palette"), i18n)),
    subnational_mapping_ui(ns("map"), i18n)
  )
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    selected_tab <- inequality_server("inequality", cache, i18n, reactive("adminlevel_1"), active = activ
    subnational_mapping_server("map", cache, i18n, reactive(input$palette), selected_tab, active = active
  })
}

## End(Not run)
```

---

subnational_target_ui

Sub-national coverage target page

Description

Shows how much of one region meets the coverage targets. One filter chooses an admin level 1 region; the page waits until one is chosen, and the unit counted is always the district. The page is the target charts and table (`target_ui()`) for that region: by year, the percentage of the region's districts that meet each target group's target (the dataset's `get_filtered_threshold()`), and a table of the areas of the region whose coverage of a chosen indicator reaches the target (the dataset's `get_high_performers()`).

Usage

```
subnational_target_ui(id, i18n)

subnational_target_server(id, cache, i18n, active = reactive(TRUE))
```

Arguments

`id`	The module's id. A page's id is also its entry in the app's page registry.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The page does no work before that.

Value

`subnational_target_ui()` returns the page's UI as Shiny tags. `subnational_target_server()` returns what `target_server()` returns: a reactive that holds the key of the tab shown.

Examples

```
## Not run:
# the page's entry in an app's page registry
cd_page_def(
  id = "subnational_target",
  ui = subnational_target_ui,
  server = subnational_target_server,
  title = "title_nav_global_coverage",
  section = "title_nav_subnational_analysis",
  subtitle = "sub_target_subnational"
)

## End(Not run)
```

---

summarise_completeness_ratio

Summarize completeness-adjusted mortality ratio

Description

Computes and summarizes adjusted mortality ratio estimates based on completeness correction factors for different completeness-to-incompleteness ratios (CI ratios).

Usage

```
summarise_completeness_ratio(
  .data,
  plot_type = c("mmr", "sbr", "nn"),
  lbr_mean = 0
)
```

Arguments

`.data`	A data frame of class `cd_mortality_ratio`, containing UN estimates and observed ratios.
`plot_type`	Character. The type of mortality ratio to summarize. Must be one of: • `"mmr"`: Maternal mortality ratio • `"sbr"`: Stillbirth rate
`lbr_mean`	Numeric. Mean live birth registration completeness, used for scaling completeness ratios. Default is `0`.

Value

A tibble of class `cd_mortality_ratio_summarised` with columns:

- `ciratio`: The completeness-to-incompleteness ratio used.
- `name`: Label for the UN estimate type (`lower bound`, `best estimate`, `upper bound`).
- `rat`: The adjusted mortality ratio value.

Examples

```
## Not run:
summarise_completeness_ratio(mortality_data, plot_type = "mmr", lbr_mean = 65)

## End(Not run)
```

---

survey_comparison_ui *Coverage against the survey card*

Description

A card of the Denominator selection page (`denominator_selection_ui()`) with one tab per indicator (the app's `survey_comp_indicators` setting). Each tab has a chart that sets the coverage worked out from the facility data with each denominator beside the survey's estimate. When the data has no survey coverage for the indicator, the tab shows a message with a button that opens the Load Data screen instead of the chart. The card has no filters of its own: the page passes it the admin level and the region. It calls the dataset's `calculate_derived_coverage()`, reads its `survey_year`, and draws nothing until its `check_inequality_params` is `TRUE`.

Usage

```
survey_comparison_ui(id, i18n)

survey_comparison_server(
  id,
  cache,
  admin_level,
  region,
  i18n,
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A reactive that returns the admin level the coverage is worked out for: `"national"`, `"adminlevel_1"` or `"district"`.
`region`	A reactive that returns the name of the chosen region, or `NULL` when none is chosen.
`active`	A reactive that is `TRUE` once the page has been opened with data loaded. The card does no work before that.

Value

`survey_comparison_ui()` returns the card's UI as Shiny tags. `survey_comparison_server()` is called for its side effects.

`survey_fields` *The dataset fields of the five survey files*

Description

The names of the dataset fields the Survey Files step fills, one per survey file. Each has built-in data for the dataset's country, so `cache()$is_default(field)` is the way to tell whether the user has uploaded a file for it.

Usage

```
survey_fields
```

Format

A character vector of five field names: `"national_survey"`, `"regional_survey"`, `"wiq_survey"` (wealth index quintiles), `"area_survey"` and `"education_survey"`.

Examples

```
survey_fields

# which survey files the user has uploaded
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
vapply(survey_fields, function(f) !cache$is_default(f), logical(1))
```

---

survey_upload_ui *Survey files step of the Load Data wizard*

Description

An optional step where the user can upload the country's survey data in place of the built-in survey data. It has one upload zone per file, each a Stata file (`.dta`): national, regional, area, maternal education and wealth index quintile. The files can be uploaded one at a time, and a file is recognised by the zone it is dropped into, not by its name. Each file is read for the dataset's country and saved with `cache()$set_national_survey()`, `cache()$set_regional_survey()`, `cache()$set_area_survey()`, `cache()$set_education_survey()` or `cache()$set_wiq_survey()`. A zone's reset button goes back to the built-in data (`cache()$clear_national_survey()` and so on). The step counts as complete when all five files are uploaded (see `step_survey_files_complete()`); being optional, it can be skipped.

Usage

```
survey_upload_ui(id, i18n)

survey_upload_server(id, cache, i18n)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.

Details

The national and regional files are read with `load_survey_data()`, the other three with `load_equity_data()`. A file that cannot be read shows an error banner under its zone. When the user has uploaded a regional survey whose region names do not all match the dataset's admin level 1 names (`cache()$check_survey_admin_names()`), a warning under the zones gives the number of names that do not match; the Map Survey Files step resolves them.

Value

`survey_upload_ui()` returns the UI as Shiny tags: a card with the five upload zones. `survey_upload_server()` is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in the server of the Load Data screen, where `cache` is a reactive holding the dataset:
# the step's page, for wizard_steps_server(), and its server
panel <- list(key = "survey_files", ui = survey_upload_ui(ns("survey_upload"), i18n))
survey_upload_server("survey_upload", cache, i18n)

## End(Not run)
```

`target_ui` *Coverage target charts and table*

Description

The body of the coverage target pages. A card of charts has a tab for each target group the app lists (`cd_cfg("target_indicators")`, from `"anc4"`, `"instlivebirths"`, `"vaccine"` and `"dropout"`) and no Custom tab; a tab plots, by year, the percentage of regions or districts that meet the group's target, from the dataset's `get_filtered_threshold()`. Below it, a table lists the areas whose coverage of one indicator reaches the target, by year, from the dataset's `get_high_performers()`; the indicator is picked above the table, and the table can be downloaded as an Excel file. The table's title names admin level 1 or districts, following the admin level. The piece has no page filters of its own: the page gives it the admin level and region.

Usage

```
target_ui(id, i18n)

target_server(
  id,
  cache,
  i18n,
  admin_level,
  region = reactive(NULL),
  active = reactive(TRUE)
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection.
`admin_level`	A reactive that returns the level of the units counted: `"adminlevel_1"` (admin level 1) or `"district"`.
`region`	A reactive that returns the name of an admin level 1 region, to count only its districts, or `NULL` for the whole country. A region is accepted only with `"district"`.

`active` A reactive that is `TRUE` once the page has been opened with data loaded. The charts and the table do no work before that.

Value

`target_ui()` returns the charts' card and the table's placeholder as Shiny tags. `target_server()` returns a reactive that holds the key of the tab shown (a target group's name).

Examples

```
## Not run:
# the charts and table in a page whose filter chooses the admin level
scope <- cd_scope("level")

my_page_ui <- function(id, i18n) {
  ns <- NS(id)
  cd_page_ui(id, i18n, filters = cd_scope_filters(ns, i18n, scope), target_ui(ns("body"), i18n))
}
my_page_server <- function(id, cache, i18n, active = reactive(TRUE)) {
  moduleServer(id, function(input, output, session) {
    geo <- cd_scope_server("scope", cache, i18n, scope)
    target_server("body", cache, i18n, geo$admin_level, geo$region, active = active)
  })
}

## End(Not run)
```

`tbl_sum.cd_adjustment_values`

Summary for `cd_adjustment_values` *Objects*

Description

Provides a custom summary for objects of class `cd_adjustment_values`, typically displaying unadjusted and adjusted values in a structured format.

Usage

```
## S3 method for class 'cd_adjustment_values'
tbl_sum(x, ...)
```

Arguments

`x` An object of class `cd_adjustment_values`, usually created by `generate_adjustment_values`, containing unadjusted and adjusted service counts.

`...` Additional arguments passed to or from other methods.

Details

This function overrides the default `tbl_sum` method for objects of class `cd_adjustment_values`. It returns a named character vector summarizing the table as "Unadjusted and adjusted values" along with any additional details provided by the `NextMethod()` function.

Value

A character vector with a custom table summary for `cd_adjustment_values` objects.

See Also

`generate_adjustment_values()` for generating the `cd_adjustment_values` object.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$adjust_data()

# the summary is the header printed above the table
cache$adjustment_values
```

---

tbl_sum.cd_average_reporting_rate

Summary for average reporting rates by year

Description

Provides a custom summary for the `cd_average_reporting_rate` object, indicating that the data represents annual average reporting rates across all indicators. This summary is useful for quickly verifying the consistency of indicator reporting rates.

Usage

```
## S3 method for class 'cd_average_reporting_rate'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_average_reporting_rate` object containing reporting rate data by year.
`...`	Additional arguments for compatibility with the S3 method.

Value

A character vector describing the content and purpose of the `cd_average_reporting_rate` data.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the summary is the header printed above the table
calculate_average_reporting_rate(data)
```

---

tbl_sum.cd_completeness_summary

Summary for `cd_completeness_summary`

Description

Provides the header shown when a `cd_completeness_summary` object is printed. It labels the table as the percentage of monthly values with complete (non-missing) data, by year, ahead of the usual tibble header.

Usage

```
## S3 method for class 'cd_completeness_summary'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_completeness_summary` object containing yearly data completeness metrics.
`...`	Additional arguments for compatibility with S3 methods.

Value

A character vector that summarizes the data content and purpose, specifically indicating the proportion of values that are complete across years.

Examples

```
## Not run:
# Generate a summary of data completeness by year
tbl_sum(cd_completeness_summary_object)

## End(Not run)
```

tbl_sum.cd_district_outliers_summary

Summary for `cd_district_outliers_summary`

Description

Provides a summary for the `cd_district_outliers_summary` object, indicating the percentage of districts without extreme outliers for each indicator by year. This summary supports monitoring data quality across districts by showing the proportion of non-extreme values across all indicators.

Usage

```
## S3 method for class 'cd_district_outliers_summary'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_district_outliers_summary` object containing the percentage of districts without extreme outliers for each indicator and year.
`...`	Additional arguments for compatibility with S3 methods.

Value

A character vector describing the purpose and content of the data, highlighting the percentage of districts with no extreme outliers for each indicator annually.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the summary is the header printed above the table
calculate_district_outlier_summary(data)
```

tbl_sum.cd_district_ratios_summary

Summary for `cd_district_ratios_summary` *Object*

Description

Provides a customized summary for a `cd_district_ratios_summary` object, which includes the percentage of districts meeting the adequacy criteria (e.g., ratios within the range of 1.0 to 1.5) across specified indicator pairs by year. This summary aids in monitoring health indicator alignment at a district level over time.

Usage

```
## S3 method for class 'cd_district_ratios_summary'
tbl_sum(x, ...)
```

Arguments

x	A `cd_district_ratios_summary` object, typically created by the `calculate_district_ratios_` function, containing adequacy checks for district-level indicator ratios by year.
...	Additional arguments to support S3 method compatibility.

Value

A character vector providing a descriptive title indicating the percentage of districts with indicator ratios that meet adequacy criteria, aiding in data quality interpretation.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
ratios <- calculate_ratios_and_adequacy(data)

# the summary is the header printed above the table
calculate_district_ratios_summary(ratios)
```

`tbl_sum.cd_district_reporting_rate`

Summary for district-level reporting rates by year

Description

Provides a custom summary for the `cd_district_reporting_rate` object, displaying a message with the threshold used in calculating district-level reporting rates by year. This summary serves to validate which districts meet the defined threshold for reporting completeness.

Usage

```
## S3 method for class 'cd_district_reporting_rate'
tbl_sum(x, ...)
```

Arguments

x	A `cd_district_reporting_rate` object containing district-level reporting rates.
...	Additional arguments for compatibility with the S3 method.

Value

A character vector describing the reporting threshold applied and summarizing district reporting rates by year.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the summary is the header printed above the table, with the threshold used
calculate_district_reporting_rate(data, threshold = 90)
```

---

tbl_sum.cd_indicator_coverage

Table summary for national coverage estimates

Description

`tbl_sum.cd_indicator_coverage` generates a summary label for national coverage estimates tables, providing a clear indication of the data contents. This function is designed to be used as a custom summary method for objects of class `cd_indicator_coverage`, creating a consistent table title for outputs.

Usage

```
## S3 method for class 'cd_indicator_coverage'
tbl_sum(x, ...)
```

Arguments

`x`	An object of class `cd_indicator_coverage`, typically a data frame or tibble containing national coverage estimates for various health indicators.
`...`	Additional arguments passed to `NextMethod()` for extending or customizing the summary method.

Details

This function is part of the summary method hierarchy for `cd_indicator_coverage` objects. By adding a specific table title (`"National Coverage Estimates"`), this function helps standardize output tables, ensuring consistency in reports and summaries for national health coverage indicators.

Value

A named character vector where the first element, `"Table"`, is titled `"National Coverage Estimates"`, followed by the summary generated by the next method in the class hierarchy.

Examples

```
## Not run:
# Assuming `data` is of class `cd_indicator_coverage`
tbl_sum.cd_indicator_coverage(data)

## End(Not run)
```

---

tbl_sum.cd_missing_district

Summary for `cd_missing_district`

Description

Provides a custom summary for the `cd_missing_district` object, showing the percentage of districts with complete (non-missing) data aggregated by year. This summary helps users understand the distribution of data completeness at the district level across time.

Usage

```
## S3 method for class 'cd_missing_district'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_missing_district` object containing yearly district-level data completeness metrics.
`...`	Additional arguments for compatibility with S3 methods.

Value

A character vector that summarizes the data content and purpose, indicating the percentage of districts with complete data across years.

Examples

```
## Not run:
# Generate a summary of district-level data completeness by year
tbl_sum(calculate_district_completeness_summary(data))

## End(Not run)
```

tbl_sum.cd_outlier *Summary for* cd_outlier

Description

Provides a custom summary for the `cd_outlier` object, displaying the percentage of monthly indicator values not considered extreme outliers, aggregated by year. This summary is essential for assessing data quality and trends across indicators.

Usage

```
## S3 method for class 'cd_outlier'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_outlier` object containing aggregated data quality metrics for monthly values.
`...`	Additional arguments for compatibility with S3 methods.

Value

A character vector summarizing the content and purpose of the data, indicating the proportion of values meeting the criteria for non-outliers by year.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))

# the summary is the header printed above the table
calculate_outliers_summary(data, admin_level = "national")
```

tbl_sum.cd_population_metrics *Summary for* cd_population_metrics

Description

The print header of the population table `prepare_population_metrics()` gives: the population, births and under-ones from DHIS2, beside the UN's at national level.

Usage

```
## S3 method for class 'cd_population_metrics'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_population_metrics` object.
`...`	Additional arguments (not used).

Value

A character vector: the table's title, the country and the default header.

Examples

```
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
cache$set_national_estimates(list(sbr = 0.02))
cache$adjust_data()

# the summary is the header printed above the table
cache$denominator_metrics
```

`tbl_sum.cd_ratios_summary`

Summary for `cd_ratios_summary` *Object*

Description

Provides a concise summary for a `cd_ratios_summary` object, which includes the calculated indicator ratios across specified pairs (e.g., ANC1/Penta1, Penta1/Penta3, OPV1/OPV3) by year. This summary facilitates understanding of inter-indicator consistency over time.

Usage

```
## S3 method for class 'cd_ratios_summary'
tbl_sum(x, ...)
```

Arguments

`x`	A `cd_ratios_summary` object, typically generated by the `calculate_ratios_summary` function. It contains year-wise ratios for specified indicator pairs.
`...`	Additional arguments to support S3 method compatibility.

Value

A character vector providing a descriptive title that clarifies the type of summary and the indicator ratios reported, to enhance interpretation when viewing the data.

Examples

```
data <- load_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
ratios <- calculate_ratios_and_adequacy(data)

# the summary is the header printed above the table
calculate_ratios_summary(ratios)
```

---

upload_box_ui *Main dataset upload of the Load Data wizard*

Description

The upload zone of the wizard's Upload Data step, where the user gives the health facility data the whole analysis runs on: an Excel file (`.xls`, `.xlsx`), a Stata file (`.dta`) or a saved dataset (`.rds`). The server reads the file and makes the dataset (a CacheConnection) that every other step works on. A file that cannot be read shows an error banner under the zone. The zone's reset button removes the dataset, which locks the later steps again. A button downloads the merged data as a Stata file. The Upload Data step counts as complete once a dataset is loaded (see `step_upload_complete()`).

Usage

```
upload_box_ui(id, i18n, is_electron = FALSE)

upload_box_server(id, i18n, cdsuite_file, is_electron = FALSE)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`is_electron`	`TRUE` when the app was started with a file to load (the desktop app), `FALSE` (the default) when the user picks the file in the page. The ui and the server must be given the same value.
`cdsuite_file`	The path of the file the app was started with, a string, or `NA_character_` when it was started without one.

Details

How a file is loaded depends on its type:

- Excel: the sheets are read with `load_excel_parts()` and kept unmerged in the dataset (`wizard_parts`) until the user presses Finish. The country is worked out from the admin sheet and stored with `set_wizard_country()`.
- Stata: the file is loaded with `load_cache_data()`, without validation. A copy is saved when the user presses Finish, as for Excel.

- `.rds`: a saved dataset is loaded with `load_cache_data()` too, and needs no walk-through.

A Stata file or a saved dataset built for another indicator group than the app's is refused (`cd_wizard_check_group()`).

An Excel or Stata file that has a saved copy from an earlier Finish (named by `cd_saved_copy_name()`, in the dataset folder) loads that copy in its place, so the user does not go through the walk-through again. When the copy cannot be read, the original file is loaded. The dataset folder is the folder of `cdsuite_file` when the app was started with a file, and otherwise `datasets` under `tools::R_user_dir("cd2030", "data")`.

When the app is started with a file (`is_electron = TRUE`) there is no upload zone: the server loads `cdsuite_file` once at startup and shows a success or an error banner.

Value

`upload_box_ui()` returns the UI as Shiny tags: the upload zone (left out when `is_electron` is `TRUE`), the placeholder for the status banner and the download button. It has no card of its own. `upload_box_server()` returns a list with three elements:

- `cache`: a reactive that returns the dataset, a CacheConnection, or `NULL` when none is loaded;
- `requires_walkthrough`: a reactive value that is `TRUE` when the wizard must be walked through step by step, and `FALSE` when a saved dataset (`.rds`) was loaded;
- `source_path`: a reactive value that holds the path Finish saves the dataset next to (the dataset folder and the file's name), or `NULL` when a saved dataset was loaded or nothing is loaded.

Examples

```
## Not run:
# in a module's ui: the user picks the file in the page
upload_box_ui(ns("upload_box"), i18n)

# in its server
upload <- upload_box_server("upload_box", i18n, cdsuite_file = NA_character_)
observe({
  req(upload$cache())
  message("Loaded ", upload$cache()$country, "; walkthrough: ", upload$requires_walkthrough())
})

# an app started with a file to load: no upload zone, the file is loaded at startup
upload_box_ui(ns("upload_box"), i18n, is_electron = TRUE)
upload <- upload_box_server("upload_box", i18n, "C:/data/Kenya.xlsx", is_electron = TRUE)

## End(Not run)
```

`with_cd_quiet` *Execute code in quiet mode*

Description

Temporarily suppress messages by enabling quiet mode within the provided code block.

Temporarily suppress messages within the specified environment.

Usage

```
with_cd_quiet(code)

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

Arguments

`code`	Code to execute quietly
`env`	The environment to use for scoping.

Value

No return value, called for side effects

No return value, called for side effects.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")

# no progress messages while the data loads
data <- with_cd_quiet(load_data(path))

# quiet until the function returns
load_quietly <- function(path) {
  local_cd_quiet()
  load_data(path)
}
data <- load_quietly(path)
```

wizard_fields *The National Rates fields of one kind*

Description

Returns every field of one kind from the app's wizard configuration (`cd_wizard_config()$national_rates_groups`) across all groups, in display order.

Usage

```
wizard_fields(kind)
```

Arguments

`kind`	The kind of field to return: `"survey"` (survey coverage estimates) or `"rate"` (national rates).

Value

A list of field descriptions, as made by `cd_wizard_survey_field()` or `cd_wizard_rate_field()`. The list is empty when the configuration has no field of that kind.

Examples

```
old <- options(cd2030.wizard = list(national_rates_groups = list(
  cd_wizard_field_group("title_upload_group_maternal", "sub_upload_group_maternal",
    cd_wizard_survey_field("anc4_prop", "anc4", "title_upload_anc4_survey"),
    cd_wizard_national_rate_fields()
  )
)))

vapply(wizard_fields("survey"), function(f) f$id, character(1))
vapply(wizard_fields("rate"), function(f) f$id, character(1))

options(old)
```

wizard_landing_ui *Summary page of the Load Data wizard*

Description

The page the wizard shows in place of its steps once a dataset is finished: after the user presses Finish, or when a saved dataset (`.rds`) is loaded. It has a banner that says the dataset is ready, with a button that opens the Reporting Rate page (`"reporting_rate"`), and one row per wizard step. Each row gives the step's title, an "optional" badge when the step is not required, a one-line summary of what the dataset holds for that step, and an Edit button that opens the step. Nothing is saved to the dataset here.

Usage

```
wizard_landing_ui(id)

wizard_landing_server(id, cache, step_defs, i18n, on_edit, just_finished)
```

Arguments

`id`	The module's id.
`cache`	A reactive that returns the dataset, a CacheConnection.
`step_defs`	The wizard's steps, a list with the structure of wizard_step_defs. The page shows one row per step, in this order.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`on_edit`	A function of one argument, the key of a step. The page calls it when the user presses that step's Edit button; the caller uses it to open the step.
`just_finished`	A reactive that returns `TRUE` when the page shows because the user has pressed Finish, and `FALSE` when it shows because a saved dataset was loaded. It only changes the banner's subtitle.

Details

`wizard_steps_server()` shows this page and starts its server; an app does not normally call either function itself.

Value

`wizard_landing_ui()` returns the UI as Shiny tags: the placeholders for the banner and the rows. `wizard_landing_server()` is called for its side effects and returns nothing useful.

Examples

```
## Not run:
# in a module's server, as wizard_steps_server() uses it
output$body <- renderUI(wizard_landing_ui(ns("landing")))

current_step <- reactiveVal("upload")
wizard_landing_server(
  "landing", cache = cache, step_defs = wizard_step_defs, i18n = i18n,
  on_edit = function(step_key) current_step(step_key),
  just_finished = reactive(FALSE)
)

## End(Not run)
```

`wizard_step_defs` *The steps of the Load Data wizard*

Description

The definition of the wizard's seven steps, in the order the user goes through them: Upload Data, Data Quality, Survey Files, National Rates, Shapefile, Map Survey Files and Map Shapefile Names. `compute_step_states()` turns it into the state of each step for a dataset. Survey Files and Shapefile are optional; the other five are required.

Usage

```
wizard_step_defs
```

Format

A list of seven lists, one per step. Each has:

- `key`: the step's key, a string: `"upload"`, `"quality"`, `"survey_files"`, `"national_rates"`, `"shapefile"`, `"survey_mapping"` or `"map_mapping"`.
- `title_key`: the translation key of the step's title.
- `required`: `TRUE` when the step must be complete before the user can go on.
- `complete_fn`: a function of the dataset that returns `TRUE` when the step is complete (see `step_upload_complete()`).
- `touched_fn` (the Shapefile step only): a function of the dataset that returns `TRUE` when the user has done something on the step.
- `relevant_fn` (the two mapping steps only): a function of the dataset whose result `compute_step_states()` reports as `relevant`.

Details

Survey Files comes before National Rates because an uploaded national survey fills in most of the National Rates fields; the user then only has to check them.

Examples

```
vapply(wizard_step_defs, function(s) s$key, character(1))
vapply(wizard_step_defs, function(s) s$required, logical(1))

# test one step on a dataset
cache <- load_cache_data(system.file("extdata", "kenya.xlsx", package = "cd2030.core"))
wizard_step_defs[[1]]$complete_fn(cache)
```

`wizard_steps_ui` *The Load Data wizard: its steps, navigation and Finish*

Description

The frame of the Load Data wizard. It shows the row of steps, the page of the current step, and a footer with Back and Continue buttons, and it decides which step the user may open (`compute_step_states()`). A newly loaded Excel or Stata file is walked through step by step: a step opens once the user has moved on from the one before it, and a required step must be complete before Continue is enabled. An optional step the user has not touched offers "Skip" in place of "Continue". The last step's button is Finish. A saved dataset (`.rds`) skips the walkthrough and opens on the summary page (`wizard_landing_ui()`), from which any step can be opened for editing.

Usage

```
wizard_steps_ui(id)

wizard_steps_server(
  id,
  i18n,
  cache,
  requires_walkthrough,
  panels,
  source_path,
  active,
  step_defs = wizard_step_defs
)
```

Arguments

`id`	The module's id.
`i18n`	The app's translator (a `shiny.i18n` translator), used for every label.
`cache`	A reactive that returns the dataset, a CacheConnection, or `NULL` when no dataset is loaded.
`requires_walkthrough`	A reactive that returns `TRUE` when the loaded file must be walked through step by step (an Excel or Stata file) and `FALSE` when it is a saved dataset (`.rds`). `upload_box_server()` returns it.
`panels`	The pages of the steps: a list with one element per step of `step_defs`. Each element is a list with `key` (the step's key) and `ui` (the step's page, as Shiny tags).
`source_path`	A reactive that returns the path Finish saves the dataset next to, a string, or `NULL`. `upload_box_server()` returns it.
`active`	A reactive that returns `TRUE` while the Load Data screen is the page shown.

`step_defs` The wizard's steps, a list with the structure of wizard_step_defs, which is the default.

Details

Finish. On a dataset whose sheets are not merged yet (`cache()$wizard_parts`), and whose blocking quality checks pass, Finish:

- merges the sheets with `merge_and_standardize()` for the app's indicator group, adds a key per admin level 1 area, and saves the result with `cache()$set_countdown_data()`;
- adds the same keys to the saved survey and shapefile name mappings (`cache()$set_survey_mapping()`, `cache()$set_map_mapping()`);
- marks the dataset as finished with `cache()$set_quality_confirmed(TRUE)`;
- sets where the dataset is saved, with `cache()$set_cache_path()`: the folder of `source_path()` and the name `cd_saved_copy_name()` gives;
- keeps the quality check results with `cache()$set_wizard_quality_results()`, then removes the unmerged sheets with `cache()$clear_wizard_parts()`.

When the merge fails, an error notification shows and the wizard stays on the last step. Otherwise the wizard shows the summary page. A dataset that has no unmerged sheets (one loaded from a Stata file) is merged already: Finish marks it finished and sets where it is saved, the same way.

Editing. A dataset that is finished (loaded from a saved copy, or `cache()$quality_confirmed` is `TRUE`) is in edit mode. The footer then has a button back to the summary page and a button that opens the Reporting Rate page (`"reporting_rate"`). Coming back to the Load Data screen from another page shows the summary page again when `cache()$quality_confirmed` is `TRUE`.

Value

`wizard_steps_ui()` returns the UI as Shiny tags: the placeholder the server fills with the wizard or the summary page. `wizard_steps_server()` returns a list with two elements:

- `edit_mode`: a reactive that is `TRUE` when a finished dataset is being edited;
- `current_step`: a reactive value that holds the key of the current step.

Examples

```
## Not run:
# in the ui of the Load Data screen
wizard_steps_ui(ns("wizard"))

# in its server: one page per step of wizard_step_defs, built with the screen's own ns()
upload <- upload_box_server("upload_box", i18n, cdsuite_file = NA_character_)
panels <- list(
  list(key = "upload", ui = upload_box_ui(ns("upload_box"), i18n)),
  list(key = "quality", ui = data_quality_ui(ns("data_quality"), i18n)),
  list(key = "survey_files", ui = survey_upload_ui(ns("survey_upload"), i18n)),
  list(key = "national_rates", ui = national_rates_ui(ns("national_rates"), i18n)),
  list(key = "shapefile", ui = shapefile_step_ui(ns("shapefile_step"), i18n)),
```

```
  list(key = "survey_mapping", ui = map_survey_ui(ns("map_survey"), i18n)),
  list(key = "map_mapping", ui = map_shapefile_ui(ns("map_shapefile"), i18n))
)
wizard <- wizard_steps_server(
  "wizard", i18n = i18n, cache = upload$cache,
  requires_walkthrough = upload$requires_walkthrough, panels = panels,
  source_path = upload$source_path, active = active
)
observe(message("On step ", wizard$current_step()))

## End(Not run)
```

`write_quality_issues` *Write every data quality result to an Excel workbook*

Description

Two sheets: `Summary` (each check, whether it passed, how many issues) and `Issues` (one row per issue).

Usage

```
write_quality_issues(results, path, labels = NULL)
```

Arguments

`results`	A named list. What `run_all_quality_checks()` returns.
`path`	Where to write the `.xlsx`.
`labels`	An optional named character vector that maps a check's key (`names(results)`) to the name to show for it, for example a translated name. A check without a label gets a plain English name.

Value

`path`, invisibly.

Examples

```
path <- system.file("extdata", "kenya.xlsx", package = "cd2030.core")
cache <- load_cache_data(path)
results <- run_all_quality_checks(cache)
file <- write_quality_issues(results, tempfile(fileext = ".xlsx"))
openxlsx::read.xlsx(file, sheet = "Issues")
```

Index

* **Countdown app charts and tables**
  - cd_coverage_plot_ui, 56
  - cd_rate_status_cell, 78
  - cd_table_card_ui, 89
  - cd_table_ui, 91
* **Countdown app filters**
  - cd_admin_level_choices, 44
  - cd_admin_level_ui, 45
  - cd_admin_parts, 47
  - cd_denominator_chip_keys, 60
  - cd_denominator_choices, 60
  - cd_denominator_options, 63
  - cd_denominator_ui, 64
  - cd_has_maternal, 68
  - cd_indicator_ui, 69
  - cd_only_denominators, 74
  - cd_palette_chip, 75
  - cd_population_ui, 77
  - cd_years_input, 101
  - cd_years_sync, 101
* **Countdown app frame**
  - cd_app, 48
  - cd_cfg, 52
  - countdown-pages, 127
* **Countdown app layout**
  - cd_default_indicator_set, 58
  - cd_default_indicators, 59
  - cd_denominator_header, 61
  - cd_denominator_row, 63
  - cd_nav_start, 72
  - cd_scope, 81
  - cd_scope_filters, 82
  - cd_scope_server, 83
  - cd_scoped_page_ui, 84
  - cd_tab_indicator, 86
  - cd_tabbed_charts_ui, 87
* **Countdown app pages**
  - adjustment_changes_ui, 14
  - bayesian_page_server, 20
  - bayesian_ui, 21
  - calculate_ratios_ui, 42
  - cd_bayes_fit_task, 50
  - cd_bayes_indicators, 51
  - cd_start_bayes_workers, 85
  - consistency_check_ui, 125
  - consistency_pair_keys, 126
  - consistency_pair_label, 126
  - coverage_trends_ui, 129
  - coverage_ui, 131
  - data_adjustment_ui, 134
  - data_completeness_ui, 135
  - den_indicators, 137
  - denominator_assessment_ui, 138
  - denominator_selection_ui, 139
  - equity_ui, 146
  - inequality_ui, 171
  - internal_consistency_ui, 174
  - introduction_ui, 176
  - national_coverage_ui, 199
  - national_inequality_ui, 200
  - national_target_ui, 203
  - outlier_detection_ui, 209
  - overall_score_ui, 210
  - reporting_rate_ui, 278
  - rr_indicators, 283
  - subnational_coverage_ui, 297
  - subnational_denominator_ui, 298
  - subnational_inequality_ui, 299
  - subnational_mapping_ui, 300
  - subnational_target_ui, 301
  - survey_comparison_ui, 303
  - target_ui, 306
* **Countdown app wizard**
  - cd_saved_copy_name, 80
  - cd_upload_data_ui, 94
  - cd_wizard_check_group, 96
  - cd_wizard_config, 97
  - cd_wizard_field_group, 98

cd_wizard_indicator_group, 99
cd_wizard_national_rate_fields, 99
cd_wizard_survey_field, 100
compute_step_states, 123
data_quality_ui, 136
map_shapefile_ui, 193
map_survey_ui, 194
mapping_modal_ui, 195
mapping_provenance_banner, 197
national_rates_required_fields, 201
national_rates_ui, 202
nr_field, 207
nr_group, 208
push_upload_on_remount, 271
reference_estimates_ui, 274
restore_default_control, 281
shapefile_step_ui, 294
step_upload_complete, 295
survey_fields, 304
survey_upload_ui, 305
upload_box_ui, 316
wizard_fields, 319
wizard_landing_ui, 319
wizard_step_defs, 321
wizard_steps_ui, 322

add_mad_med_columns, 9
add_mad_med_columns(), *11*
add_missing_column, 10
add_outlier5std_column, 11
add_outlier5std_column(), *9*
adjust_service_data, 12
adjust_service_data(), *17*, *150*
adjustment_changes_server *(adjustment_changes_ui)*, 14
adjustment_changes_ui, 14
adjustment_changes_ui(), *134*
adjustment_evidence, 15
adjustment_evidence(), *134*
adjustment_settings_check, 16
adjustment_settings_default, 17
adjustment_settings_default(), *12*
adjustment_settings_footnote, 17
adjustment_settings_footnote(), *14*
adjustment_settings_from_k, 18
adjustment_steps_used, 19
apply_chart_options(), *67*
bayes_model_estimates, 20
bayes_model_estimates(), *154*
bayesian_page_server, 20
bayesian_page_server(), *22*
bayesian_server *(bayesian_ui)*, 21
bayesian_server(), *20*, *21*
bayesian_ui, 21
bayesian_ui(), *51*

cache_definition, 23
cache_definition(), *24*
cache_manifest, 24
cache_manifest(), *23*
CacheConnection, *14*, *21–23*, *43*, *45*, *49*, *50*, *54*, *61*, *65*, *77*, *79*, *83*, *84*, *92*, *95*, *96*, *123*, *125*, *128*, *130*, *131*, *134–136*, *138*, *139*, *146*, *172*, *174*, *194–196*, *199*, *200*, *202*, *203*, *209*, *210*, *272*, *275*, *279*, *282*, *294*, *296*, *298–300*, *302*, *304–306*, *316*, *317*, *320*, *322*
calculate_average_reporting_rate, 24
calculate_completeness_summary, 25
calculate_completeness_summary(), *217*
calculate_coverage, 26
calculate_derived_coverage, 27
calculate_district_completeness_summary, 28
calculate_district_completeness_summary(), *38*
calculate_district_outlier_summary, 28
calculate_district_ratios_summary, 29
calculate_district_ratios_summary(), *40*
calculate_district_reporting_rate, 30
calculate_district_reporting_rate(), *224*
calculate_health_system_comparison, 31
calculate_health_system_metrics, 32
calculate_health_system_metrics(), *157*, *158*
calculate_indicator_coverage, 33
calculate_indicator_coverage(), *26*, *31*, *167*, *168*
calculate_inequality, 35
calculate_outliers_summary, 36
calculate_overall_score, 37

calculate_overall_score(), *38*, *39*, *245*
calculate_overall_score1, 38
calculate_ratios_and_adequacy, 39
calculate_ratios_server *(calculate_ratios_ui)*, 42
calculate_ratios_summary, 41
calculate_ratios_summary(), *40*
calculate_ratios_ui, 42
calculate_ratios_ui(), *174*
calculate_threshold, 43
calculate_threshold(), *257*
cd-plot-arguments *(cd_plot_arguments)*, 76
cd_admin_level_choices, 44
cd_admin_level_server *(cd_admin_level_ui)*, 45
cd_admin_level_server(), *47*, *83*
cd_admin_level_ui, 45
cd_admin_level_ui(), *44*, *82*, *128*
cd_admin_parts, 47
cd_admin_parts(), *45*
cd_app, 48
cd_app(), *57*, *61*, *72*, *128*
cd_bayes_fit_task, 50
cd_bayes_fit_task(), *22*
cd_bayes_indicators, *22*, 51
cd_bayes_install_command, 51
cd_bayes_install_command(), *80*
cd_bayes_packages_missing, 52
cd_bayes_packages_missing(), *22*
cd_cfg, 52
cd_cfg(), *68*, *126*
cd_chart_catalog, 54
cd_chart_id, 54
cd_chart_id(), *54*, *67*
cd_chart_options(), *76*, *143–145*, *212*, *214*, *216*, *217*, *219–224*, *226*, *227*, *229*, *230*, *232–234*, *236–241*, *243–246*, *248*, *249*, *251*, *253–256*, *258*, *259*, *261*, *263*, *264*
cd_chart_type(), *67*
cd_chartable_members, 55
cd_chartable_members(), *70*
cd_coverage_plot_server *(cd_coverage_plot_ui)*, 56
cd_coverage_plot_server(), *88*
cd_coverage_plot_toolbar_ui *(cd_coverage_plot_ui)*, 56
cd_coverage_plot_toolbar_ui(), *88*
cd_coverage_plot_ui, 56
cd_coverage_plot_ui(), *88*
cd_custom_chart_data, 57
cd_default_indicator_set, 58
cd_default_indicator_set(), *59*, *88*
cd_default_indicators, *58*, 59
cd_denominator_chip_keys, 60
cd_denominator_choices, 60, *63*
cd_denominator_header, 61
cd_denominator_header(), *49*, *63*
cd_denominator_labels, 62
cd_denominator_options, 63
cd_denominator_options(), *60*, *65*, *74*
cd_denominator_row, 63
cd_denominator_row(), *61*
cd_denominator_server *(cd_denominator_ui)*, 64
cd_denominator_ui, 64
cd_denominator_ui(), *60*, *139*
cd_describe_columns, 66
cd_describe_columns(), *67*
cd_dictionary, 66
cd_dictionary(), *66*
cd_finish_plot, 67
cd_has_maternal, 68
cd_has_maternal(), *53*, *61*, *139*
cd_indicator_labels, 68
cd_indicator_labels(), *217*, *243*
cd_indicator_server *(cd_indicator_ui)*, 69
cd_indicator_ui, 69
cd_indicator_ui(), *88*, *93*
cd_member_data, 70
cd_methodology_defaults, 71
cd_methodology_defaults(), *23*
cd_minimal_theme, 72
cd_nav_denominators *(cd_nav_start)*, 72
cd_nav_national *(cd_nav_start)*, 72
cd_nav_quality *(cd_nav_start)*, 72
cd_nav_start, 72
cd_nav_start(), *48*
cd_nav_subnational *(cd_nav_start)*, 72
cd_only_denominators, 74
cd_palette_chip, 75
cd_plot_arguments, 76, *261*
cd_population_server

(cd_population_ui), 77
cd_population_ui, 77
cd_population_ui(), *138*
cd_rate_status_cell, 78
cd_report_context, 79
cd_request_bayes_packages, 80
cd_saved_copy_name, 80
cd_saved_copy_name(), *317*, *323*
cd_scope, 81
cd_scope(), *82–84*, *128*
cd_scope_filters, 82
cd_scope_filters(), *81*, *83*
cd_scope_server, 83
cd_scope_server(), *81*, *82*, *85*
cd_scoped_page_server *(cd_scoped_page_ui)*, 84
cd_scoped_page_server(), *81*, *83*
cd_scoped_page_ui, 84
cd_scoped_page_ui(), *81*, *82*
cd_start_bayes_workers, 85
cd_start_bayes_workers(), *20*, *50*
cd_tab_indicator, 86
cd_tab_indicator(), *89*
cd_tabbed_charts_server *(cd_tabbed_charts_ui)*, 87
cd_tabbed_charts_server(), *86*
cd_tabbed_charts_ui, 87
cd_tabbed_charts_ui(), *56*, *58*, *59*
cd_table_card_server *(cd_table_card_ui)*, 89
cd_table_card_ui, 89
cd_table_card_ui(), *91*
cd_table_server *(cd_table_ui)*, 91
cd_table_server(), *78*
cd_table_ui, 91
cd_table_ui(), *89*, *128*
cd_type_scale, 94
cd_type_scale(), *72*
cd_upload_data_server *(cd_upload_data_ui)*, 94
cd_upload_data_ui, 94
cd_upload_data_ui(), *128*
cd_wizard_check_group, 96
cd_wizard_check_group(), *317*
cd_wizard_config, 97
cd_wizard_config(), *94*, *98*, *100*, *201*, *202*
cd_wizard_field_group, 98
cd_wizard_field_group(), *97*, *99*, *100*
cd_wizard_indicator_group, 99
cd_wizard_indicator_group(), *81*, *97*
cd_wizard_national_rate_fields, 99
cd_wizard_national_rate_fields(), *98*
cd_wizard_rate_field *(cd_wizard_survey_field)*, 100
cd_wizard_rate_field(), *97–99*, *319*
cd_wizard_survey_field, 100
cd_wizard_survey_field(), *97*, *98*, *319*
cd_years_input, 101
cd_years_input(), *102*
cd_years_sync, 101
cd_years_sync(), *101*
check_admin_columns, 102
check_admin_columns(), *198*
check_admin_pairing_consistency, 103
check_country_recognized, 104
check_country_recognized(), *281*
check_district_consistency, 105
check_district_consistency(), *103*, *107*, *182*, *185*, *198*
check_district_cross_sheet, 106
check_district_year_completeness, 107
check_indicator_emptiness, 107
check_indicator_emptiness(), *108*
check_indicator_emptiness_presheet, 108
check_month_language_consistency, 109
check_month_presence, 110
check_month_presence(), *110*, *182*, *185*, *198*
check_month_presence_presheet, 110
check_month_validity, 111
check_month_validity(), *182*, *185*, *198*
check_month_validity_presheet, 112
check_population_service_collision, 113
check_population_service_collision(), *114*
check_population_service_collision_presheet, 114
check_population_vs_births, 115
check_required_columns_presheet, 116
check_required_columns_presheet(), *105*
check_shapefile_admin_names, 117
check_single_country, 118

check_single_country(), *104*, *198*, *281*
check_survey_admin_names, 119
clean_error_message, 120
compute_indicator_numerator, 121
compute_service_utilization, 122
compute_service_utilization(), *165*, *267*, *292*
compute_step_states, 123
compute_step_states(), *295*, *321*, *322*
consistency_check_server *(consistency_check_ui)*, 125
consistency_check_ui, 125
consistency_check_ui(), *126*, *174*
consistency_pair_keys, 126
consistency_pair_keys(), *125*
consistency_pair_label, 126
countdown-pages, 127
coverage_server *(coverage_ui)*, 131
coverage_server(), *199*, *298*
coverage_trends_server *(coverage_trends_ui)*, 129
coverage_trends_ui, 129
coverage_trends_ui(), *139*
coverage_ui, 131
coverage_ui(), *199*, *297*
create_mortality_ratios, 132
create_mortality_summary, 133
create_mortality_summary(), *291*

data_adjustment_server *(data_adjustment_ui)*, 134
data_adjustment_ui, 134
data_completeness_server *(data_completeness_ui)*, 135
data_completeness_ui, 135
data_quality_server *(data_quality_ui)*, 136
data_quality_ui, 136
data_quality_ui(), *95*
datasuite.ui::as_report_context(), *79*
datasuite.ui::report_context(), *79*
den_indicators, 137
denominator_assessment_server *(denominator_assessment_ui)*, 138
denominator_assessment_ui, 138
denominator_assessment_ui(), *137*
denominator_selection_server *(denominator_selection_ui)*, 139
denominator_selection_ui, 139
denominator_selection_ui(), *129*, *298*, *303*
detect_indicator_group, 140

equiplot, 141
equiplot(), *143*, *144*, *146*
equiplot_area, 142
equiplot_education, 143
equiplot_wealth, 145
equity_server *(equity_ui)*, 146
equity_ui, 146
export_deck(), *278*
export_report(), *67*

filter_high_performers, 147
filter_out_years, 148

generate_adjustment_values, 149
generate_adjustment_values(), *13*, *14*, *19*, *212*, *308*
generate_admin1_keys, 151
generate_admin1_mch_curative_index, 152
generate_admin1_mch_curative_index(), *236*
generate_admin1_service_utilization, 152
generate_admin1_service_utilization(), *254*
generate_bayes_model, 153
generate_bayes_model(), *20*, *50*, *52*, *295*
generate_coverage_data, 154
generate_fpet_summary, 156
generate_health_system_table, 156
generate_health_system_table(), *230*
generate_phc_scatter_data, 157
generate_phc_scatter_data(), *246*
generate_private_sector_data, 158
generate_private_sector_data(), *249*
generate_service_dqa_summary, 159
generate_service_dqa_summary(), *258*
get_admin_columns, 160
get_all_indicators, 161
get_analysis_indicators, 162
get_country_iso3, 162

get_country_name, 163
get_country_shapefile, 163
get_coverage_indicators, 164
get_excel_version, 165
get_indicator_group_names, 166
get_indicator_groups, 166
get_indicator_groups(), *166*, *276*
get_mapping_data, 167
get_mapping_data(), *235*
get_named_indicators, 167
get_national_rates, 168
get_plot_admin_column, 169
get_population_column, 170
get_selected_group, 171
get_selected_group(), *166*, *293*
ggplot2::element_text(), *94*
ggplot2::geom_text(), *94*
ggplot2::theme_minimal(), *72*

inequality_server *(inequality_ui)*, 171
inequality_server(), *299*
inequality_ui, 171
inequality_ui(), *200*, *299*
init_CacheConnection, 173
internal_consistency_server *(internal_consistency_ui)*, 174
internal_consistency_ui, 174
internal_consistency_ui(), *42*, *125*
interpret, 175
interpret.cd_fpet_data, 176
interpret.cd_fpet_data(), *175*
introduction_server *(introduction_ui)*, 176
introduction_ui, 176
is_maternal_indicator, 177

list_indicator_groups, 178
list_missing_units, 178
list_outlier_units, 179
list_tracer_vaccines, 180
list_vaccine_indicators, 180
load_cache_data, 181
load_cache_data(), *316*, *317*
load_csection_estimates, 182
load_csection_estimates(), *190*, *269*
load_data, 184
load_data(), *181*
load_equity_data, 186
load_equity_data(), *305*
load_excel_parts, 186
load_excel_parts(), *112*, *198*, *316*
load_fpet_data, 188
load_private_sector_data, 189
load_private_sector_data(), *183*, *269*
load_survey_data, 190
load_survey_data(), *305*
load_un_estimates, 191
load_un_mortality_data, 192
load_wuenic_data, 193
local_cd_quiet *(with_cd_quiet)*, 318

map_shapefile_server *(map_shapefile_ui)*, 193
map_shapefile_ui, 193
map_shapefile_ui(), *95*, *195*
map_survey_server *(map_survey_ui)*, 194
map_survey_ui, 194
map_survey_ui(), *95*, *195*
mapping_modal_server *(mapping_modal_ui)*, 195
mapping_modal_server(), *197*
mapping_modal_ui, 195
mapping_modal_ui(), *193*, *194*
mapping_provenance_banner, 197
mapping_provenance_banner(), *193*, *194*
merge_and_standardize, 198
merge_and_standardize(), *186*, *188*, *323*

national_coverage_server *(national_coverage_ui)*, 199
national_coverage_ui, 199
national_inequality_server *(national_inequality_ui)*, 200
national_inequality_ui, 200
national_rates_required_fields, 201
national_rates_required_fields(), *296*
national_rates_server *(national_rates_ui)*, 202
national_rates_ui, 202
national_rates_ui(), *95*
national_target_server *(national_target_ui)*, 203
national_target_ui, 203
new_countdown, 204

new_countdown(), *13*, *104*, *116*, *184*, *185*, *198*, *280*
notebook_data, 205
nr_field, 207
nr_field(), *208*
nr_group, 208

outlier_detection_server *(outlier_detection_ui)*, 209
outlier_detection_ui, 209
overall_score_server *(overall_score_ui)*, 210
overall_score_ui, 210

plot.cd_adjustment_values_filtered, 211
plot.cd_average_reporting_rate, 213
plot.cd_bayes_model, 215
plot.cd_completeness_summary, 216
plot.cd_coverage, 218
plot.cd_coverage_filtered, 219
plot.cd_coverage_selected, 221
plot.cd_derived_coverage, 222
plot.cd_district_reporting_rate, 223
plot.cd_fpet_data, 225
plot.cd_health_system_comparison, 227
plot.cd_health_system_metric, 228
plot.cd_health_system_table, 229
plot.cd_health_system_table(), *156*
plot.cd_indicator_coverage, 231
plot.cd_inequality_filtered, 232
plot.cd_mapping_filtered, 234
plot.cd_mch_curative_index, 235
plot.cd_mch_curative_index(), *152*
plot.cd_missing_district, 236
plot.cd_mortality_ratio_summarised, 238
plot.cd_mortality_summary, 239
plot.cd_mortality_summary_filtered, 240
plot.cd_outlier, 242
plot.cd_outlier_list, 243
plot.cd_overall_score, 245
plot.cd_phc_scatter, 246
plot.cd_phc_scatter(), *157*
plot.cd_population_metrics, 247
plot.cd_private_sector, 249
plot.cd_private_sector(), *158*
plot.cd_private_sector_plot_data, 250
plot.cd_private_sector_plot_data(), *270*
plot.cd_ratios_summary, 252
plot.cd_service_utilization_admin1, 253
plot.cd_service_utilization_admin1(), *152*
plot.cd_service_utilization_filtered, 254
plot.cd_service_utilization_prepared, 256
plot.cd_threshold, 257
plot.cd_utilization_dqa, 258
plot.cd_utilization_dqa(), *159*
plot_comparison, 259
plot_comparison(), *76*, *125*
plot_comparison.cd_data, 262
plot_comparison_anc1_penta1 *(plot_comparison)*, 259
plot_comparison_opv1_opv3 *(plot_comparison)*, 259
plot_comparison_penta1_penta3 *(plot_comparison)*, 259
plot_line_graph, 264
plot_mortality_plausibility, 265
plot_national_health_metric, 266
prepare_mapping_service_utlization, 267
prepare_mapping_service_utlization(), *256*
prepare_population_metrics, 268
prepare_population_metrics(), *314*
prepare_private_sector_plot_data, 269
prepare_private_sector_plot_data(), *183*, *190*, *250*
print.cd_data, 270
push_upload_on_remount, 271

quality_issues_table, 273
quality_issues_table(), *284*

read_shapefile_folder, 273
reference_estimates_server *(reference_estimates_ui)*, 274
reference_estimates_ui, 274
reference_estimates_ui(), *95*
register_indicator_group, 275
register_indicator_group(), *140*, *185*
render_report_block(), *277*

report_block_kinds, 277
report_presets, 278
report_project_blocks(), *278*
reporting_rate_server
(*reporting_rate_ui*), 278
reporting_rate_ui, 278
reporting_rate_ui(), *128*, *283*
reset_indicator_group, 280
resolve_chart_options(), *67*
resolve_country_best_effort, 280
restore_default_control, 281
robust_max, 282
robust_max(), *9*
rr_indicators, 283
run_all_quality_checks, 284
run_all_quality_checks(), *136*, *182*, *185*, *188*, *198*, *273*, *296*, *324*

save_data, 285
save_dhis2_excel, 286
save_dhis2_master_data, 287
select_adjustment_value, 288
select_coverage, 288
select_inequality, 289
select_inequality(), *233*
select_mapping_data, 290
select_mapping_data(), *234*, *235*
select_mortality_summary, 291
select_service_utilization, 292
select_service_utilization(), *255*
set_selected_group, 293
set_selected_group(), *171*, *205*
shapefile_step_server
(*shapefile_step_ui*), 294
shapefile_step_ui, 294
shapefile_step_ui(), *95*
shiny::req(), *46*, *47*
shiny::runApp(), *49*
slim_bayes_model, 295
step_map_mapping_complete
(*step_upload_complete*), 295
step_map_mapping_complete(), *193*
step_national_rates_complete
(*step_upload_complete*), 295
step_national_rates_complete(), *201*, *202*
step_quality_complete
(*step_upload_complete*), 295
step_quality_complete(), *136*
step_shapefile_complete
(*step_upload_complete*), 295
step_shapefile_complete(), *294*
step_shapefile_touched
(*step_upload_complete*), 295
step_survey_files_complete
(*step_upload_complete*), 295
step_survey_files_complete(), *305*
step_survey_mapping_complete
(*step_upload_complete*), 295
step_survey_mapping_complete(), *194*
step_upload_complete, 295
step_upload_complete(), *316*, *321*
subnational_coverage_server
(*subnational_coverage_ui*), 297
subnational_coverage_ui, 297
subnational_denominator_server
(*subnational_denominator_ui*), 298
subnational_denominator_ui, 298
subnational_denominator_ui(), *139*
subnational_inequality_server
(*subnational_inequality_ui*), 299
subnational_inequality_ui, 299
subnational_mapping_server
(*subnational_mapping_ui*), 300
subnational_mapping_ui, 300
subnational_mapping_ui(), *200*
subnational_target_server
(*subnational_target_ui*), 301
subnational_target_ui, 301
summarise_completeness_ratio, 302
survey_comparison_server
(*survey_comparison_ui*), 303
survey_comparison_ui, 303
survey_comparison_ui(), *139*
survey_estimates_required_fields
(*national_rates_required_fields*), 201
survey_estimates_required_fields(), *296*
survey_fields, *296*, 304
survey_upload_server
(*survey_upload_ui*), 305
survey_upload_ui, 305
survey_upload_ui(), *95*

target_server *(target_ui)*, 306
target_server(), *204*, *302*
target_ui, 306
target_ui(), *203*, *301*
tbl_sum.cd_adjustment_values, 307
tbl_sum.cd_average_reporting_rate, 308
tbl_sum.cd_completeness_summary, 309
tbl_sum.cd_district_outliers_summary, 310
tbl_sum.cd_district_ratios_summary, 310
tbl_sum.cd_district_reporting_rate, 311
tbl_sum.cd_indicator_coverage, 312
tbl_sum.cd_missing_district, 313
tbl_sum.cd_outlier, 314
tbl_sum.cd_population_metrics, 314
tbl_sum.cd_ratios_summary, 315

upload_box_server *(upload_box_ui)*, 316
upload_box_server(), *322*
upload_box_ui, 316
upload_box_ui(), *95*, *275*

with_cd_quiet, 318
wizard_fields, 319
wizard_landing_server *(wizard_landing_ui)*, 319
wizard_landing_ui, 319
wizard_landing_ui(), *322*
wizard_step_defs, *123*, *295*, *320*, 321, *323*
wizard_steps_server *(wizard_steps_ui)*, 322
wizard_steps_server(), *94*, *320*
wizard_steps_ui, 322
write_quality_issues, 324
write_quality_issues(), *136*, *284*