Function List:

build_initial_state()

Start a new CLIParseState filled with sensible defaults and environment values so later steps only need to tweak what the user actually changed.

apply_env_language!(state)

Check well-known environment variables for language preferences and store the first valid hit inside the parse state.

apply_force_prompt_env!(state)

Read FORCE_WEIGHT_PROMPT from the environment and decide whether the weight prompt should always appear. Prints a warning when the value is unclear.

register_language_choice!(state, token; mark_explicit=false)

Try to switch to the language given by token. When mark_explicit is true we also flag that choice so automatic detection will not change it later.

process_cli_arguments!(state)

Run through every command-line argument, updating the parse state as we go. Flags that consume an extra value are handled automatically.

handle_cli_argument!(state, arg, index)

Update the parse state for one command-line token. The return value tells the caller how many additional tokens were consumed (for example when a flag needs a value).

apply_weight_override!(state, flag, value)

Update an individual metric weight based on the CLI flag/value pair, validating both the flag name and the provided value.

parse_date_value(raw, warn_key)

Attempt to parse a date string, emitting a localised warning on failure and returning nothing so invalid input can be ignored gracefully.

finalize_cli_config(state)

Normalise and validate the accumulated parse state, producing a ready-to-use CLIConfig plus updated speech-command wiring.

parse_cli()

Read all command-line options and environment variables and turn them into a tidy CLIConfig object.

main()

Entry point used by the executable script. Reads the CLI, loads the data, runs the requested command, and says goodbye when finished.

language_display_name(lang)

Return the translated display name for a language symbol using the localisation table.

with_language(config, lang; explicit)

Construct a new CLIConfig with the supplied language while copying all other fields, optionally flagging the choice as explicit.

apply_language_choice(token, fallback)

Normalise a language token and attempt to activate it, returning the effective language plus a flag indicating whether the change actually took place.

maybe_prompt_language(config)

Offer the interactive language prompt when no explicit language was chosen, returning an updated CLIConfig (or the original when prompting is skipped).

prompt_country_choice(df)

Show a numbered list of countries and let the user pick one by typing the number or the name. Returns the selection or nothing if they cancel.

ask_adjust_weights!(weights)

Ask the user whether they want to change the metric weights. If they say yes the weight prompt runs and the dictionary is updated in place.

report_config_for_menu(base, weights; country=nothing)

Create a copy of the current configuration that uses the supplied weights and optional country filter before calling the reporting workflow.

run_menu(df, config)

Display the looping main menu. Users can explore all regions, focus on one country, adjust weights, or exit the program from here.

build_translation_strings()

Parse the raw translation table and return a dictionary for each supported language.

normalize_language(lang)

Tidy up a language token (such as "de-DE") and return a symbol like :de or :en.

is_supported_language(lang)

Return true when the language token maps to one of the bundles we ship.

available_languages()

Return a list of language codes that can be selected.

current_language()

Return the language symbol that is currently active.

localized_country_name(country; lang=current_language())

Return the localized display name for a country, falling back to the canonical dataset label when no translation is available.

canonical_country_name(input)

Map a country label entered by the user to the dataset's canonical spelling. Returns nothing when no mapping is known.

string_map(lang)

Fetch the translations dictionary for the requested language, falling back to the default language when needed.

set_language!(lang)

Activate a language by symbol or string. Falls back to the default language when the requested language is unknown.

localize(key; kwargs...)

Look up the translation for key in the current language and replace any {{placeholder}} values using the provided keyword arguments.

add_weighted_score!(df, weights)

Create or update a WeightedScore column by mixing the metrics according to the chosen weights. Missing data is treated gently so gaps do not skew the result.

build_monthly_overview(df; weights=DEFAULT_METRIC_WEIGHTS)

Collect the latest month of data, calculate averages for each region, and return both the table and a handy month label. A status flag explains why the table might be empty.

print_monthly_overview_for_all_regions(df; weights=DEFAULT_METRIC_WEIGHTS, display=true)

Prepare the monthly overview table for every region. By default it prints a nicely formatted table, but it can also return the data silently for reuse elsewhere.

print_weighted_ranking(monthly_table, month_label; top_n=10)

Sort the monthly overview by weighted score, list the top regions, and show the rank table to the user. Returns the table so it can be inspected again later.

print_active_filters(config, df)

Tell the user which filters are currently active (season, date range, region, country) and how many rows remain after filtering.

print_active_weights(weights)

Show each metric weight in an easy-to-read format so the user sees how much influence each factor has at the moment.

print_decision_hints(scoreboard, monthly_table, weights)

Print a short list of helpful suggestions, such as which region has the most powder or the calmest winds based on the latest ranking tables.

prompt_region_choice(df, scoreboard, config)

Help the user choose a region by offering sensible suggestions. Returns the region name when recognised, otherwise nothing.

print_region_history(df, region_name; months=12)

Show a month-by-month summary (averages and totals) for the chosen region so users can spot longer trends.

region_top_snow_events(df; top_n=5)

Return the days with the biggest fresh-snow gains for a region, plus helpful context such as snow depth and temperature when available.

run_list(df)

Print every region name in alphabetical order. Useful when a user is unsure about the exact spelling.

resolve_region_name(df, name)

Try to match a user-entered region to the dataset. When no exact match is found, offer up to five similar suggestions.

run_region(df, region_name; weights=DEFAULT_METRIC_WEIGHTS, monthly_table=nothing)

Drive the detailed region view: show headline stats, monthly summaries, recent conditions, history tables, and create plots if possible.

styled_table(data; kwargs...)

Print tables with the same colours and highlights everywhere. Handy for keeping reports easy to read.

lookup_column(df, col)

Find a column by name, accepting either symbols or strings. Returns nothing when the column cannot be found.

qc_checks(df)

Print friendly warnings when the dataset looks suspicious (missing days, negative snow depth, unrealistic temperatures, and so on).

print_data_preview(df; limit=5)

Show the first and last few rows of the table so the user can confirm the data looks right before diving into bigger reports.

current_month_subset(df)

Return the rows that belong to the current month (or the most recent month with data) along with the date label we should display.

print_current_month_overview(df)

Summarise the current month's temperature, snow, wind, and precipitation. Tells the user when no data is available.

safe_stat(values, reducer)

Run a summary function (like mean) on the values that can be converted to numbers. Returns missing when nothing usable is found.

metric_group_summary(df; groupcol, ycol)

Group the table by region or country and calculate how many values exist, plus the average, median, min, and max for the selected metric.

recent_conditions(df; recent_days=14)

Pick the most recent rows (two weeks by default) and return just the columns that are useful for a quick daily conditions table.

print_daily_scoreboard(df; top_n=5)

Show which regions picked up the most new snow within the last year, highlighting the top rows for easy reading. Returns the table for reuse.

save_region_snow_plot(region_df, region_name; recent_days=90)

Draw a snow trend chart (depth plus new snow bars) for a region and save it as a PNG file. Returns the file path or nothing when plotting is not possible.

save_region_score_trend(region_monthly, region_name)

Plot how the weighted score changed over time for a region and save the image. Returns nothing when there is not enough information.

save_region_metric_trend(region_df, region_name, option)

Create a chart for a chosen metric (for example wind or precipitation) and save it to the plots folder. Skips the chart when the data is missing.

region_metric_options(df)

List which metric plots make sense for the given region, based on the data columns that actually contain values.

resolve_metric_tokens(tokens, options)

Turn user input (numbers or keywords) into actual metric selections and tell the caller which tokens were not understood.

generate_metric_plots(region_df, region_name, selections)

Create each selected metric plot in turn and report whether it was saved or skipped.

prompt_region_metric_plots(region_df, region_name; env_selection=nothing)

Interactively ask which extra metric charts to generate for a region. Supports preset lists through the REGION_METRICS environment variable.

prompt_region_details(df, ranking; config, weights, monthly_table)

Let the user pick regions from the ranking table for a deeper dive. Works interactively when a terminal is available and prints hints otherwise.

run_report(df, config, weights)

The main reporting pipeline: show the active filters, display leaderboards, and then offer detailed region exploration.

search_for_csv(filename)

Walk through the project folders to find a CSV with the given name. Used as a last resort when other lookup methods fail.

normalize_path(path)

Trim extra spaces from a path string. If the result is empty we return nothing so callers know there was no real value.

resolve_csv_path(csv_path)

Figure out which CSV file to load by checking command-line values, environment variables, and common fallbacks. Returns either a usable path or nothing.

ensure_local_csv(path)

Make sure the CSV file exists locally. Remote files are downloaded to a cache folder so later reads are fast and offline-friendly.

normalize_columns!(df)

Rename columns that refer to temperature, snow, wind, and similar topics so their names match the labels we use everywhere else.

load_data([csv_path])

Read the ski dataset from a CSV file, tidy up the columns, and make sure dates are in order. Throws a user-friendly error when no data file can be found.

canonical_country(value)

Translate different spellings or codes for Austria, Germany, and Switzerland into a single clean country name. Unknown values are returned as-is.

add_newsnow!(df)

Add a column that records how much the snow depth grew from one day to the next for each region. Drops in snow depth are treated as zero new snow.

in_season(date, season)

Return true when the given date falls inside the named season. WINTER covers November–April, SUMMER covers May–October, and ALL keeps every date.

apply_filters(df, runargs)

Filter the dataset by region, country, date range, and season based on the values in runargs and environment variables. Returns the filtered copy.

find_date_column(df)

Look through the table headers and return the column that most likely stores dates. If nothing matches typical date names we return nothing.

lininterp!(v)

Fill in missing numbers in a vector by connecting the known values with straight lines. Values at the start or end copy the nearest known value.

rolling_mean(v, window)

Return a smoothed copy of a numeric vector by averaging neighbouring values. When the window is too small we just give back the original data.

available_regions(df)

Return all distinct region names found in the data, sorted alphabetically.

available_countries(df)

Return all distinct country names found in the data, sorted alphabetically.

filter_country(df, country)

Return only the rows that belong to country, ignoring case differences. If no country is provided the original table is returned.

print_available_regions(regions)

Print a simple bulleted list of region names. When the list is empty a friendly message explains that no regions were found.

stdin_is_tty()

Return true when the program is connected to a real terminal window. Interactive prompts should only appear in that case.

normalize_speech_cmd(cmd)

Clean up a speech command string. Empty or whitespace-only input returns nothing, which signals that speech capture is disabled.

current_speech_cmd()

Return the speech command stored for the current session, or nothing when speech input is off.

set_speech_cmd!(cmd)

Store the active speech command after cleaning it with normalize_speech_cmd. The effective command (or nothing) is returned.

run_speech_capture(cmd)

Run the external speech command and return its trimmed output. Any problem results in nothing plus a warning so keyboard input can take over.

readline_with_speech(prompt="> "; ...)

Ask the user for input. If a speech command is configured we try it first and fall back to keyboard input when nothing useful is heard.

prompt_yes_no(; default=false, invalid_key=:prompt_yes_no_retry, prompt="> ")

Ask the user for a yes/no answer and only accept y or n (case-insensitive). An empty response returns the provided default. Any other input prints a reminder and the question repeats.

detect_speech_command()

Look for known helper scripts (transcribe.sh, etc.) and return the first command we find. If no helper is present, nothing is returned.

maybe_prompt_speech_cmd!(current)

Offer the user a choice to enable speech input when we are running in a terminal. The resolved command (or nothing) is returned.

slower(x)

Turn any value into a lowercase string. This helps us compare names without worrying about uppercase or lowercase letters.

Base.lowercase(x::Symbol)

Allow symbols (like :Region) to be lowered directly so callers do not need to convert them to strings first.

slugify(name)

Create a safe filename from a region name. We keep letters and numbers, replace other characters with underscores, and remove extras so saved plots have tidy names.

ensure_plot_dir()

Create the plots/ folder if it does not already exist and return the full path. We call this right before saving images so saving never fails because of a missing folder.

clean_numeric_series(dates, values)

Walk two matching lists of dates and numbers, skipping entries that are empty or not convertible to numbers. The clean pairs are returned so plots and stats can rely on valid data only.

collect_valid(values)

Collect all usable numbers from an input list. Entries that are missing, nothing, or not real numbers are ignored so callers receive a clean vector of Float64 values.

clone_metric_weights()

Return a copy of the default weights so callers can tweak the values without touching the shared defaults.

parse_weight_value(raw)

Turn a user-supplied weight (like "25" or "25%") into a number. Returns nothing when the text cannot be understood.

parse_bool(value)

Read common yes/no strings and return true, false, or nothing if the value is ambiguous.

apply_weight_env_overrides!(weights)

Check environment variables like WEIGHT_SNOW_NEW and update the weight map when the values look valid. Warnings are printed otherwise.

apply_weight_preset!(weights, preset)

Copy the numbers from a preset into the weight map and normalise the result so the weights still add up to 100.

prompt_weight_profile!(weights; force=false)

Show the preset list (balanced, powder, family, sunny) and let the user pick. Returns one of four symbols: • :skip – nothing was changed, move along • :manual – user wants to enter every weight manually • :preset – preset applied, no further adjustments requested
• :preset_manual – preset applied, user wants to tweak numbers afterwards

prompt_metric_weights!(weights; force=false)

Ask the user for each metric weight one by one. Pressing Enter keeps the current value. The function loops until the total equals 100, otherwise it restarts the questions with a friendly reminder.

normalize_weights!(weights)

Rescale all weights so they sum to 100. If the sum is zero or negative we warn the user, restore the defaults, and return those.

prepare_weights!(weights; force, prompt)

High-level helper used by the CLI/menu. Steps:

> Optionally show the preset picker (forced when force=true without a TTY).
> If the user wants manual input, call prompt_metric_weights!.
> Normalise weights before returning them. Always returns the updated dictionary.