Package: plnr (via r-universe)

June 9, 2026

Title A Framework for Planning and Executing Analyses

Version 2025.11.22

Description A comprehensive framework for planning and executing analyses in R. It provides a structured approach to running the same function multiple times with different arguments, executing multiple functions on the same datasets, and creating systematic analyses across multiple strata or variables. The framework is particularly useful for applying the same analysis across multiple strata (e.g., locations, age groups), running statistical methods on multiple variables (e.g., exposures, outcomes), generating multiple tables or graphs for reports, and creating systematic surveillance analyses. Key features include efficient data management, structured analysis planning, flexible execution options, built-in debugging tools, and hash-based caching.

License MIT + file LICENSE

URL https://www.rwhite.no/plnr/, https://github.com/raubreywhite/plnr

BugReports https://github.com/raubreywhite/plnr/issues

Encoding UTF-8

LazyData true

Depends R (>= 3.3.0)

Imports data.table, digest, fs, foreach, glue, pbmcapply, purrr, R6, stats, tidyr, usethis, utils, uuid

Suggests testthat, knitr, rmarkdown, progressr, ggplot2, readxl, magrittr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/pak/sysreqs cmake git make libgit2-dev libicu-dev libuv1-dev libssl-dev libx11-dev

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

Date/Publication 2026-05-10 16:50:25 UTC

RemoteUrl https://github.com/raubreywhite/plnr

RemoteRef HEAD

RemoteSha 3c073ffee5246f7df3e92cbce346880decb174a2

Contents

`create_rmarkdown` *Create an example R Markdown project structure*

Description

This function creates a complete example project structure for an R Markdown analysis using the `plnr` framework. It sets up a standardized directory structure and creates example files demonstrating how to use `plnr` for data analysis and report generation.

Usage

```
create_rmarkdown(home)
```

Arguments

`home`	Character string, the path where the project should be created

Details

The created project includes:

- A main `run.R` script that initializes the project and demonstrates `plnr` usage
- Example analysis functions in the `R` directory
- A template R Markdown document
- Standard project directories (results, paper, raw)

Value

NULL, creates files and directories in the specified location

Examples

```
## Not run:
# Create a temporary directory for the example
temp_dir <- tempfile("plnr_example_")
create_rmarkdown(temp_dir)

# View the created structure
list.files(temp_dir, recursive = TRUE)

## End(Not run)
```

`example_action_fn` *Example action function for demonstrating analysis structure*

Description

This function serves as an example of how to structure an action function for use with the `Plan` class. It simply prints the names of the data and argset components it receives.

Usage

```
example_action_fn(data, argset)
```

Arguments

`data`	A named list containing the datasets for the analysis
`argset`	A named list containing the arguments for the analysis

Value

NULL, prints information about the input data and argset

Examples

```
# Create a new plan
p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Create batch of argsets
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
```

```
)

# Add analyses to plan
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)

# View argsets and run example
p$get_argsets_as_dt()
p$run_one("analysis_1")
```

`example_data_fn_nor_covid19_cases_by_time_location` *An example data_fn that returns a data set*

Description

An example data_fn that returns a data set

Usage

```
example_data_fn_nor_covid19_cases_by_time_location()
```

`expand_list` *Create a cross product of lists*

Description

This function creates a cross product of multiple lists, similar to `tidyr::expand_grid()` but with a more convenient interface that doesn't require wrapping arguments in an extra `list()`. It's useful for generating combinations of parameters for analysis.

Usage

```
expand_list(...)
```

Arguments

`...`	Named arguments, each containing a vector or list of values to combine

Value

A list of lists, where each inner list contains one combination of values from the input arguments

Examples

```
# Create combinations of parameters
combinations <- plnr::expand_list(
  a = 1:2,
  b = c("a", "b")
)

# View the combinations
str(combinations)

# Compare with tidyr::expand_grid
tidyr::expand_grid(list(
  a = 1:2,
  b = c("a", "b")
))
```

`get_anything` *Get objects with package namespace support*

Description

This function extends `base::get()` to support package namespace scoping (e.g., "pkg::var"). It's particularly useful when working with package exports and namespace-qualified objects.

Usage

```
get_anything(x)
```

Arguments

`x`	Character string specifying the object to retrieve. Can be either a simple object name or a namespace-qualified name (e.g., "pkg::var")

Value

The requested object

Examples

```
# Get a namespace-qualified object
plnr::get_anything("plnr::nor_covid19_cases_by_time_location")

# Get a simple object (same as base::get)
x <- 1
get_anything("x")
```

is_run_directly *Check if code is being run directly or from within a function*

Description

This function determines whether code is being executed directly in the global environment or from within a function call. It's particularly useful for development and debugging purposes, allowing functions to behave differently when run directly versus when called as part of a larger analysis plan.

Usage

```
is_run_directly()
```

Value

A logical value: TRUE if the code is being run directly (i.e., from the global environment), FALSE if it's being run from within a function call

Examples

```
# When run directly
is_run_directly()  # TRUE

# When run from within a function
test_fn <- function() {
  is_run_directly()  # FALSE
}
test_fn()
```

nor_covid19_cases_by_time_location *Covid-19 data for PCR-confirmed cases in Norway (nation and county)*

Description

This data comes from the Norwegian Surveillance System for Communicable Diseases (MSIS). The date corresponds to when the PCR-test was taken.

Usage

```
nor_covid19_cases_by_time_location
```

Format

A csfmt_rts_data_v1 with 11028 rows and 18 variables:

granularity_time day/isoweek

granularity_geo nation, county

country_iso3 nor

location_code norge, 11 counties

border 2020

age total

isoyear Isoyear of event

isoweek Isoweek of event

isoyearweek Isoyearweek of event

season Season of event

seasonweek Seasonweek of event

calyear Calyear of event

calmonth Calmonth of event

calyearmonth Calyearmonth of event

date Date of event

covid19_cases_testdate_n Number of confirmed covid19 cases

covid19_cases_testdate_pr100000 Number of confirmed covid19 cases per 100.000 population

Details

The raw number of cases and cases per 100.000 population are recorded.

This data was extracted on 2022-05-04.

Source

https://github.com/folkehelseinstituttet/surveillance_data/blob/master/covid19/_DOCUMENTATION_data_covid19_msis_by_time_location.txt

Description

The `Plan` class provides a framework for organizing and executing multiple analyses on one or more datasets. It enforces a structured approach to analysis by:

1. **Data Management**:
   - Loading data once and reusing across analyses
   - Separating data cleaning from analysis
   - Providing hash-based tracking of data changes
2. **Analysis Structure**:
   - Requiring all analyses to use the same data sources
   - Standardizing analysis functions to accept only `data` and `argset` parameters
   - Organizing analyses into clear, maintainable plans
3. **Execution Control**:
   - Supporting both single-function and multi-function analysis plans
   - Providing flexible execution options (sequential or parallel)
   - Including built-in debugging tools

Details

The framework uses three main concepts:

- **Argset**: A named list containing a set of arguments for an analysis
- **Analysis**: A combination of one argset and one action function
- **Plan**: A container that holds one data pull and a list of analyses

Public fields

`analyses` List of analyses, each containing an argset and action function

Methods

Public methods:

- `Plan$new()`
- `Plan$add_data()`
- `Plan$add_argset()`
- `Plan$add_argset_from_df()`
- `Plan$add_argset_from_list()`
- `Plan$add_analysis()`
- `Plan$add_analysis_from_df()`
- `Plan$add_analysis_from_list()`
- `Plan$apply_action_fn_to_all_argsets()`
- `Plan$apply_analysis_fn_to_all()`
- `Plan$x_length()`
- `Plan$x_seq_along()`

- Plan$set_progress()
- Plan$set_progressor()
- Plan$set_verbose()
- Plan$set_use_foreach()
- Plan$get_data()
- Plan$get_analysis()
- Plan$get_argset()
- Plan$get_argsets_as_dt()
- Plan$run_one_with_data()
- Plan$run_one()
- Plan$run_all_with_data()
- Plan$run_all()
- Plan$run_all_progress()
- Plan$run_all_parallel()
- Plan$clone()

Method `new()`**:** Create a new Plan instance

Usage:

```
Plan$new(verbose = interactive() | config$force_verbose, use_foreach = FALSE)
```

Arguments:

`verbose` Logical, whether to show verbose output. Defaults to TRUE in interactive mode or when config$force_verbose is TRUE

`use_foreach` Logical, whether to use foreach for parallel processing. NULL = program decides, FALSE = use loop, TRUE = use foreach

Returns: A new Plan instance

Method `add_data()`**:** Add a new dataset to the plan

Usage:

```
Plan$add_data(name, fn = NULL, fn_name = NULL, direct = NULL)
```

Arguments:

`name` Character string, name of the dataset

`fn` Function that returns the dataset (optional)

`fn_name` Character string, name of a function that returns the dataset (optional)

`direct` Direct dataset object (optional)

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Add data using a function
data_fn <- function() { return(plnr::nor_covid19_cases_by_time_location) }
p$add_data("data_1", fn = data_fn)

# Add data using a function name
```

```
p$add_data("data_2", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Add data directly
p$add_data("data_3", direct = plnr::nor_covid19_cases_by_time_location)

# View added data
p$get_data()
```

Method `add_argset()`**:** Add a new argset to the plan

Usage:

```
Plan$add_argset(name = uuid::UUIDgenerate(), ...)
```

Arguments:

`name` Character string, name of the argset (defaults to a UUID)

`...` Named arguments that will comprise the argset

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Add argsets with different arguments
p$add_argset("argset_1", var_1 = 3, var_b = "hello")
p$add_argset("argset_2", var_1 = 8, var_c = "hello2")

# View added argsets
p$get_argsets_as_dt()
```

Method `add_argset_from_df()`**:** Add multiple argsets from a data frame

Usage:

```
Plan$add_argset_from_df(df)
```

Arguments:

`df` Data frame where each row represents a new argset

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Create data frame of argsets
batch_argset_df <- data.frame(
  name = c("a", "b", "c"),
  var_1 = c(1, 2, 3),
  var_2 = c("i", "j", "k")
)

# Add argsets from data frame
p$add_argset_from_df(batch_argset_df)

# View added argsets
p$get_argsets_as_dt()
```

Method `add_argset_from_list()`**:** Add multiple argsets from a list

Usage:

```
Plan$add_argset_from_list(l)
```

Arguments:

`l` List of lists, where each inner list represents a new argset

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Create list of argsets
batch_argset_list <- list(
  list(name = "a", var_1 = 1, var_2 = "i"),
  list(name = "b", var_1 = 2, var_2 = "j"),
  list(name = "c", var_1 = 3, var_2 = "k")
)

# Add argsets from list
p$add_argset_from_list(batch_argset_list)

# View added argsets
p$get_argsets_as_dt()
```

Method `add_analysis()`**:** Add a new analysis to the plan

Usage:

```
Plan$add_analysis(name = uuid::UUIDgenerate(), fn = NULL, fn_name = NULL, ...)
```

Arguments:

`name` Character string, name of the analysis (defaults to a UUID)

`fn` Function to use for the analysis (optional)

`fn_name` Character string, name of the function to use (optional)

`...` Additional arguments to be added to the argset

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Add analysis
p$add_analysis(
  name = "analysis_1",
  fn_name = "plnr::example_action_fn"
)

# View argsets and run analysis
p$get_argsets_as_dt()
p$run_one("analysis_1")
```

Method `add_analysis_from_df()`**:** Add multiple analyses from a data frame

Usage:

```
Plan$add_analysis_from_df(fn = NULL, fn_name = NULL, df)
```

Arguments:

`fn` Function to use for all analyses (optional)

`fn_name` Character string, name of the function to use (optional)

`df` Data frame where each row represents a new analysis

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Create data frame of analyses
batch_argset_df <- data.frame(
  name = c("a", "b", "c"),
  var_1 = c(1, 2, 3),
  var_2 = c("i", "j", "k")
)

# Add analyses from data frame
p$add_analysis_from_df(
  fn_name = "plnr::example_action_fn",
  df = batch_argset_df
)

# View argsets and run example
p$get_argsets_as_dt()
p$run_one(1)
```

Method `add_analysis_from_list()`**:** Add multiple analyses from a list

Usage:

```
Plan$add_analysis_from_list(fn = NULL, fn_name = NULL, l)
```

Arguments:

`fn` Function to use for all analyses (optional)

`fn_name` Character string, name of the function to use (optional)

`l` List of lists, where each inner list represents a new analysis

Returns: NULL, modifies the plan in place

Examples:

```
p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
```

```
# Create list of analyses
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)

# Add analyses from list
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)

# View argsets and run example
p$get_argsets_as_dt()
p$run_one("analysis_1")
```

Method `apply_action_fn_to_all_argsets()`**:** Applies an action function to all the argsets

Usage:

```
Plan$apply_action_fn_to_all_argsets(fn = NULL, fn_name = NULL)
```

Arguments:

`fn` Action function.

`fn_name` Action function name. p <- plnr::Plan$new() p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location") batch_argset_list <- list(list(name = "analysis_1", var_1 = 1, var_2 = "i"), list(name = "analysis_2", var_1 = 2, var_2 = "j"), list(name = "analysis_3", var_1 = 3, var_2 = "k")) p$add_argset_from_list(fn_name = "plnr::example_action_fn", l = batch_argset_list) p$get_argsets_as_dt() p$apply_action_fn_to_all_argsets(fn_name = "plnr::example_action_fn") p$run_one("analysis_1")

Method `apply_analysis_fn_to_all()`**:** Deprecated. Use apply_action_fn_to_all_argsets.

Usage:

```
Plan$apply_analysis_fn_to_all(fn = NULL, fn_name = NULL)
```

Arguments:

`fn` Action function.

`fn_name` Action function name.

Method `x_length()`**:** Number of analyses in the plan.

Usage:

```
Plan$x_length()
```

Method `x_seq_along()`**:** Generate a regular sequence from 1 to the length of the analyses in the plan.

Usage:

```
Plan$x_seq_along()
```

Method `set_progress()`**:** Set an internal progress bar

Usage:

```
Plan$set_progress(pb)
```

Arguments:

`pb` Progress bar.

Method `set_progressor()`**:** Set an internal progressor progress bar

Usage:

```
Plan$set_progressor(pb)
```

Arguments:

`pb` progressor progress bar.

Method `set_verbose()`**:** Set verbose flag

Usage:

```
Plan$set_verbose(x)
```

Arguments:

`x` Boolean.

Method `set_use_foreach()`**:** Set use_foreach flag

Usage:

```
Plan$set_use_foreach(x)
```

Arguments:

`x` Boolean.

Method `get_data()`**:** Extracts the data provided via 'add_data' and returns it as a named list.

Usage:

```
Plan$get_data()
```

Returns: Named list, where most elements have been added via `add_data`.
One extra named element is called 'hash'. 'hash' contains the data hashes of particular datasets/variables, as calculated using the 'spookyhash' algorithm via digest::digest. 'hash' contains two named elements:

- current (the hash of the entire named list)
- current_elements (the hash of the named elements within the named list)

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
p$get_data()
```

Method `get_analysis()`**:** Extracts an analysis from the plan.

Usage:

```
Plan$get_analysis(index_analysis)
```

Arguments:

index_analysis Either an integer (1:length(analyses)) or a character string representing the name of the analysis.

Returns: An analysis.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_analysis("analysis_1")
```

Method `get_argset()`**:** Extracts an argset from the plan.

Usage:

```
Plan$get_argset(index_analysis)
```

Arguments:

index_analysis Either an integer (1:length(analyses)) or a character string representing the name of the analysis.

Returns: An argset

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_argset("analysis_1")
```

Method `get_argsets_as_dt()`**:** Gets all argsets and presents them as a data.table.

Usage:

```
Plan$get_argsets_as_dt()
```

Returns: Data.table that contains all the argsets within a plan.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_argsets_as_dt()
```

Method `run_one_with_data()`**:** Run one analysis (data is provided by user).

Usage:

```
Plan$run_one_with_data(index_analysis, data, ...)
```

Arguments:

`index_analysis` Either an integer (1:length(analyses)) or a character string representing the name of the analysis.

`data` Named list (generally obtained from p$get_data()).

`...` Not used.

Returns: Returned value from the action function.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
data <- p$get_data()
p$run_one_with_data("analysis_1", data)
```

Method `run_one()`**:** Run one analysis (data is obtained automatically from self$get_data()).

Usage:

```
Plan$run_one(index_analysis, ...)
```

Arguments:

`index_analysis` Either an integer (1:length(analyses)) or a character string representing the name of the analysis.

`...` Not used.

Returns: Returned value from the action function.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_one("analysis_1")
```

Method `run_all_with_data()`**:** Run all analyses (data is provided by user).

Usage:

```
Plan$run_all_with_data(data, ...)
```

Arguments:

`data` Named list (generally obtained from p$get_data()).

`...` Not used.

Returns: List where each element contains the returned value from the action function.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
data <- p$get_data()
p$run_all_with_data(data)
```

Method `run_all()`**:** Run all analyses (data is obtained automatically from self$get_data()).

Usage:

```
Plan$run_all(...)
```

Arguments:

`...` Not used.

Returns: List where each element contains the returned value from the action function.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_all()
```

Method `run_all_progress()`**:** Run all analyses with a progress bar (data is obtained automatically from self$get_data()).

Usage:

```
Plan$run_all_progress(...)
```

Arguments:

`...` Not used.

Returns: List where each element contains the returned value from the action function.

Examples:

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_all_progress()
```

Method `run_all_parallel()`**:** Run all analyses in parallel (data is obtained automatically from self$get_data()).
This function only works on linux computers and uses `pbmcapply` as the parallel backend.

Usage:

```
Plan$run_all_parallel(mc.cores = getOption("mc.cores", 2L), ...)
```

Arguments:

`mc.cores` Number of cores to be used.

`...` Not used.

Returns: List where each element contains the returned value from the action function.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Plan$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Plan$add_data`
## ------------------------------------------------

p <- plnr::Plan$new()

# Add data using a function
data_fn <- function() { return(plnr::nor_covid19_cases_by_time_location) }
p$add_data("data_1", fn = data_fn)

# Add data using a function name
p$add_data("data_2", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Add data directly
p$add_data("data_3", direct = plnr::nor_covid19_cases_by_time_location)

# View added data
p$get_data()

## ------------------------------------------------
## Method `Plan$add_argset`
## ------------------------------------------------

p <- plnr::Plan$new()

# Add argsets with different arguments
p$add_argset("argset_1", var_1 = 3, var_b = "hello")
p$add_argset("argset_2", var_1 = 8, var_c = "hello2")

# View added argsets
p$get_argsets_as_dt()

## ------------------------------------------------
## Method `Plan$add_argset_from_df`
## ------------------------------------------------

p <- plnr::Plan$new()

# Create data frame of argsets
batch_argset_df <- data.frame(
  name = c("a", "b", "c"),
  var_1 = c(1, 2, 3),
  var_2 = c("i", "j", "k")
)
```

```
# Add argsets from data frame
p$add_argset_from_df(batch_argset_df)

# View added argsets
p$get_argsets_as_dt()

## ------------------------------------------------
## Method `Plan$add_argset_from_list`
## ------------------------------------------------

p <- plnr::Plan$new()

# Create list of argsets
batch_argset_list <- list(
  list(name = "a", var_1 = 1, var_2 = "i"),
  list(name = "b", var_1 = 2, var_2 = "j"),
  list(name = "c", var_1 = 3, var_2 = "k")
)

# Add argsets from list
p$add_argset_from_list(batch_argset_list)

# View added argsets
p$get_argsets_as_dt()

## ------------------------------------------------
## Method `Plan$add_analysis`
## ------------------------------------------------

p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Add analysis
p$add_analysis(
  name = "analysis_1",
  fn_name = "plnr::example_action_fn"
)

# View argsets and run analysis
p$get_argsets_as_dt()
p$run_one("analysis_1")

## ------------------------------------------------
## Method `Plan$add_analysis_from_df`
## ------------------------------------------------

p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
```

```
# Create data frame of analyses
batch_argset_df <- data.frame(
  name = c("a", "b", "c"),
  var_1 = c(1, 2, 3),
  var_2 = c("i", "j", "k")
)

# Add analyses from data frame
p$add_analysis_from_df(
  fn_name = "plnr::example_action_fn",
  df = batch_argset_df
)

# View argsets and run example
p$get_argsets_as_dt()
p$run_one(1)

## ------------------------------------------------
## Method `Plan$add_analysis_from_list`
## ------------------------------------------------

p <- plnr::Plan$new()

# Add example data
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")

# Create list of analyses
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)

# Add analyses from list
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)

# View argsets and run example
p$get_argsets_as_dt()
p$run_one("analysis_1")

## ------------------------------------------------
## Method `Plan$get_data`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
p$get_data()

## ------------------------------------------------
## Method `Plan$get_analysis`
```

```
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_analysis("analysis_1")

## ------------------------------------------------
## Method `Plan$get_argset`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_argset("analysis_1")

## ------------------------------------------------
## Method `Plan$get_argsets_as_dt`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$get_argsets_as_dt()

## ------------------------------------------------
## Method `Plan$run_one_with_data`
## ------------------------------------------------
```

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
data <- p$get_data()
p$run_one_with_data("analysis_1", data)

## ------------------------------------------------
## Method `Plan$run_one`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_one("analysis_1")

## ------------------------------------------------
## Method `Plan$run_all_with_data`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
data <- p$get_data()
p$run_all_with_data(data)

## ------------------------------------------------
## Method `Plan$run_all`
## ------------------------------------------------
```

```
p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_all()

## ------------------------------------------------
## Method `Plan$run_all_progress`
## ------------------------------------------------

p <- plnr::Plan$new()
p$add_data("covid_data", fn_name = "plnr::example_data_fn_nor_covid19_cases_by_time_location")
batch_argset_list <- list(
  list(name = "analysis_1", var_1 = 1, var_2 = "i"),
  list(name = "analysis_2", var_1 = 2, var_2 = "j"),
  list(name = "analysis_3", var_1 = 3, var_2 = "k")
)
p$add_analysis_from_list(
  fn_name = "plnr::example_action_fn",
  l = batch_argset_list
)
p$run_all_progress()
```

`set_opts` *Set package configuration options*

Description

This function allows users to configure package-wide options, such as verbosity of output messages. It modifies the package's internal configuration state.

Usage

```
set_opts(force_verbose = FALSE)
```

Arguments

`force_verbose`	Logical, whether to force verbose output messages regardless of the interactive state. Defaults to `FALSE`

Value

NULL, modifies the package's internal configuration

Examples

```
# Enable verbose output
set_opts(force_verbose = TRUE)

# Disable verbose output
set_opts(force_verbose = FALSE)
```

---

test_action_fn *Test action function that returns a constant value*

Description

A simple test function that always returns 1, useful for testing the Plan framework's functionality.

Usage

```
test_action_fn(data, argset)
```

Arguments

data	A named list containing the datasets (unused in this example)
argset	A named list containing the arguments (unused in this example)

Value

The integer 1

---

try_again *Retry code execution with exponential backoff*

Description

This function attempts to execute code multiple times with random delays between attempts. It's particularly useful for handling transient failures in operations that may succeed on subsequent attempts, such as network requests or file operations.

Usage

```
try_again(
  x,
  times = 2,
  delay_seconds_min = 5,
  delay_seconds_max = 10,
  verbose = FALSE
)
```

Arguments

x	The code to execute (as an expression)
times	Integer, the maximum number of attempts to make. Defaults to 2
delay_seconds_min	Numeric, the minimum delay in seconds between attempts. Defaults to 5
delay_seconds_max	Numeric, the maximum delay in seconds between attempts. Defaults to 10
verbose	Logical, whether to show progress information. Defaults to FALSE

Details

The function is adapted from the try_again function in the testthat package, but with additional features for controlling retry behavior and verbosity.

Value

TRUE invisibly if successful, otherwise throws an error with the last error message

Examples

```
## Not run:
# Try a simple operation
try_again({
  # Your code here
  stop("Simulated error")
}, times = 3, verbose = TRUE)

# Try with custom delays
try_again({
  # Your code here
  stop("Simulated error")
}, delay_seconds_min = 1, delay_seconds_max = 3)

## End(Not run)
```

Index