Package 'TSAR'

January 10, 2025

Type Package

Title Thermal Shift Analysis in R

Version 1.4.0

Year 2023

Description This package automates analysis workflow for Thermal Shift Analysis (TSA) data. Processing, analyzing, and visualizing data through both shiny applications and command lines. Package aims to simplify data analysis and offer front to end workflow, from raw data to multiple trial analysis.

License AGPL-3

Encoding UTF-8

LazyData false

Testthat true

RoxygenNote 7.2.3

Imports dplyr (>= 1.0.7), ggplot2 (>= 3.3.5), ggpubr (>= 0.4.0), magrittr (>= 2.0.3), mgcv (>= 1.8.38), readxl (>= 1.4.0), stringr (>= 1.4.0), tidyr (>= 1.1.4), utils (>= 4.3.1), shiny (>= 1.7.4.1), plotly (>= 4.10.2), shinyjs (>= 2.1.0), jsonlite (>= 1.8.7), rhandsontable (>= 0.3.8), openxlsx (>= 4.2.5.2), shinyWidgets (>= 0.7.6), minpack.lm (>= 1.2.3)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews Software, ShinyApps, Visualization, qPCR

DataRaw data/qPCR_data1.rda

DataRaw2 data/qPCR_data2.rda

DataRaw3 data/Well_Information.rda

DataRaw4 data/Well_Information_Template.rda

DataRawr data/example_tsar_data.rda

Depends R (>= 4.3.0)

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/TSAR

git_branch RELEASE_3_20

git_last_commit 5d98ac1

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-01-09

Author Xinlin Gao [aut, cre] (<https://orcid.org/0009-0002-2518-235X>),
William M. McFadden [aut, fnd]
(<https://orcid.org/0000-0001-6911-2172>),
Stefan G. Sarafianos [fnd, aut, ths]
(<https://orcid.org/0000-0002-5840-154X>)

Maintainer Xinlin Gao <candygao2015@outlook.com>

Contents

`analyze_norm` *Analyze to Normalize*

Description

The analyze_norm function allows users to process analysis through an UI interface. Function wraps together all functions with in TSA_analysis family and read_write_analysis family.

Usage

```
analyze_norm(raw_data)
```

Arguments

`raw_data`	The raw data for analysis.

Value

shiny application

See Also

`gam_analysis`, `read_tsar`, `write_tsar`, `join_well_info`

Examples

```
if (interactive()) {
    data("qPCR_data1")
    shiny::runApp(analyze_norm(qPCR_data1))
}
```

`condition_IDs` *TSAR Condition IDs*

Description

This function is used to extract information of the condition IDs from a loaded TSA Analysis Data file. Condition IDs are automatically generated by the read_analysis function in the automated workflow. This returns either a character vector of unique IDs present or a numeric value of the number of unique IDs.

Usage

```
condition_IDs(analysis_data, n = FALSE)
```

Arguments

analysis_data	a data frame that is unmerged and generated by TSAR::read_analysis() or a merged TSA data frame generated by TSAR::merge_TSA(). Data frames require a column named 'condition_ID'.
n	logical value; n = FALSE by default. When TRUE, a numeric value of unique IDs is returned. When FALSE, a character vector of unique IDs are returned.

Value

Either a character vector of condition_IDs or a numeric value.

See Also

merge_TSA and read_analysis for preparing input.

Other TSA Summary Functions: TSA_ligands(), TSA_proteins(), well_IDs()

Examples

```
data("example_tsar_data")
condition_IDs(example_tsar_data)
```

---

example_tsar_data *Example tsar_data file*

Description

Dataset Description: This is an example dataset of the tsar_data strucutre. The data frame contains well ID, conditions, and experimental details.

Usage

```
data(example_tsar_data)
```

Format

A data frame with the following columns:

Well Well position

Temperature Temperature in degrees

Fluorescence Fluorescence reading

Normalized Normalized value

norm_deriv Calculated first derivative

Tm Tm value

Protein Protein information

Ligand Ligand information

ExperimentFileName Experiment file name

well_ID Well ID

condition_ID Condition ID

Value

example tsar_data in data frame

Source

experimentally obtained

`gam_analysis` *Analysis of all 96 wells through gam modeling*

Description

Function pipeline that combines separated functions and iterate through each well to estimate the Tm.

Usage

```
gam_analysis(
  raw_data,
  keep = TRUE,
  fit = FALSE,
  smoothed = FALSE,
  boltzmann = FALSE,
  fluo_col = NA,
  selections = c("Well.Position", "Temperature", "Fluorescence", "Normalized")
)
```

Arguments

`raw_data`	data frame; raw data frame
`keep`	Boolean; set to `keep = TRUE` by default to return normalized data and fitted data
`fit`	Boolean; set to `fit = FALSE` by default, `fit = TRUE` returns access to information of each model fit. Not accessible in shiny.
`smoothed`	Boolean; set to `smoothed = FALSE` by default, if data is already smoothed, set smoothed to true
`boltzmann`	Boolean; set to `boltzmann = FALSE` by default. Set to `boltzmann = TRUE` if a botlzmann fit is preferred.
`fluo_col`	integer; the Fluorescence variable column id (e.g. fluo = 5 when 5th column of data frame is the Fluorescence value) if fluorescence variable is named exactly as "Fluorescence", fluo does not need to be specified.
`selections`	list of characters; the variables in raw data user intends to keep. It is set, by default, to `c("Well.Position","Temperature", "Fluorescence", "Normalized")`.

Value

List of data frames, list of three data frame outputs, Tm estimation by well, data set, fit of model by well.

See Also

Other tsa_analysis: `Tm_est()`

Examples

```
data("qPCR_data1")
gam_analysis(qPCR_data1,
    smoothed = TRUE, boltzmann = FALSE, fluo_col = 5,
    selections = c(
        "Well.Position", "Temperature", "Fluorescence",
        "Normalized"
    )
)
model <- gam_analysis(qPCR_data1, smoothed = FALSE, boltzmann = TRUE)
```

---

get_legend *Extract ggplot2 legend*

Description

Function enables separation of legends from plots within the TSAR package.

Usage

```
get_legend(input_plot)
```

Arguments

`input_plot`	a ggplot2 object

Value

two ggplots, one containing the legend and another containing all else.

See Also

Other TSA Plots: `TSA_boxplot()`, `TSA_compare_plot()`, `TSA_wells_plot()`, `graph_tsar()`, `view_deriv()`

Examples

```
data("example_tsar_data")
boxplot <- TSA_boxplot(example_tsar_data,
    color_by = "Protein",
    label_by = "Ligand", separate_legend = FALSE
)
get_legend(boxplot)
```

graph_tsar *Graph tsar_data*

Description

Function allows users to graph out tsar_data, building boxplot, compare plots, and curves by condition. Input of data as parameter is optional. graph_tsar wraps together all graphing functions and relative helper functions.

Usage

```
graph_tsar(tsar_data = data.frame())
```

Arguments

tsar_data	tsar data outputted by merge_norm or merge_tsa. Parameter is optional. If no data is passed, access the merge panel to merge norm_data into tsar_data.

Value

prompts separate app window for user interaction, does not return specific value; generates boxplot and compare plots according to user input

See Also

TSA_boxplot, TSA_compare_plot, condition_IDs, well_IDs, merge_norm, TSA_Tms, Tm_difference

Other TSA Plots: TSA_boxplot(), TSA_compare_plot(), TSA_wells_plot(), get_legend(), view_deriv()

Examples

```
if (interactive()) {
    data("example_tsar_data")
    shiny::runApp(graph_tsar(example_tsar_data))
}
```

join_well_info *Well information input function*

Description

Reads in the ligand and protein information and joins them accordingly to the big data frame for graphing purposes.

Usage

```
join_well_info(
  file_path,
  file = NULL,
  analysis_file,
  skips = 0,
  nrows = 96,
  type
)
```

Arguments

file_path	string; file path to read in the file
file	object; use file to override the need of file_path if information is already read in
analysis_file	data frame; data frame containing smoothed fluorescence data and tm values
skips	integer; number indicating the number of headers present in input file, default set to 0 when file input is "by_well" If the input follows the excel template, this parameter does not apply.
nrows	integer; number indicating the number of rows the data is. Default set to 96 assuming analysis on 96 well plate. Parameter is only applicable when file input is "by_well". If inputting by excel template, this parameter does not apply, please ignore.
type	string; variable specifies the type of input read in. type = "by_well" requires input of csv or txt files of three variables: Well, Protein, Ligand. type = "by_template" requires input of excel file following the template format provided

Value

outputs data frame joining data information with well information

See Also

Other read_write_analysis: `read_tsar()`, `write_tsar()`

Examples

```
data("qPCR_data1")
result <- gam_analysis(qPCR_data1, smoothed = TRUE, fluo = 5)
data("well_information")
join_well_info(
    file_path = NULL, file = well_information,
    read_tsar(result, output_content = 2), type = "by_template"
)
```

merge_norm *Merge and format norm_data into tsar_data*

Description

This function merges data of experiment replicates across different dates. It merges and produces information variables used to group wells of same set up.

Usage

```
merge_norm(data, name, date)
```

Arguments

data	list, a character vector specifying the file paths of the data files or data frame objects of analysis data set. For example, given data frames named "data1" and "data2", specify parameter as `data = list(data1, data2)`.
name	list, character vector specifying the experiment names.
date	list, character vector specifying the dates. Does not require any date format restrictions.

Details

This function merges and normalizes test data from multiple files. The lengths of the `data`, `name`, and `date` vectors must match, otherwise an error is thrown.

Value

data frame in the format of tsar_data

See Also

Other TSAR Formatting: `TSA_Tms()`, `TSA_average()`, `Tm_difference()`, `merge_TSA()`, `normalize_fluorescence()` `rescale()`

Examples

```
data("qPCR_data1")
result <- gam_analysis(qPCR_data1, smoothed = TRUE, fluo = 5)
data("well_information")
norm_data <- join_well_info(
    file_path = NULL, file = well_information,
    read_tsar(result, output_content = 2), type = "by_template"
)
norm_data <- na.omit(norm_data)
data("qPCR_data2")
result2 <- gam_analysis(qPCR_data1, smoothed = TRUE, fluo = 5)
norm_data2 <- join_well_info(
    file_path = NULL, file = well_information,
    read_tsar(result2, output_content = 2), type = "by_template"
)
norm_data2 <- na.omit(norm_data2)
```

```
tsar_data <- merge_norm(
    data = list(norm_data, norm_data2),
    name = c("Thermal Shift_162.eds.csv", "Thermal Shift_168.eds.csv"),
    date = c("20230203", "20230209")
)
```

`merge_TSA` *Merge TSA Raw Data and Analysis Files*

Description

This function is used to load both the Raw Data and the Analysis Results which are returned by the TSA software. Both output files have unique information regarding the experiment, and these need reunited for downstream analysis. Automatically generated Well IDs are use to merge similar data within the same experiment from the different files. Both Raw Data and the Analysis Results files must be specified. The returned, merged results from this function are required for downstream analysis as the format is set up for the automated workflow.

Usage

```
merge_TSA(analysis_file_path, raw_data_path, protein = NA, ligand = NA)
```

Arguments

`raw_data_path`, `analysis_file_path`
: a character string or vector of character strings; the path or the name of the file which the 'RawData' or "AnalysisData' are to be read from. Raw data and Analysis Data are to be loaded as a pair of results from the same experiment. When loading multiple files, the index/position of the pairs are merged where the first file specified by raw_data_path is to be merged with the first file specified by analysis_file_path. The same is done for the second, third, .. etc.
Either a .txt or .csv file; file type can vary between file pairs.

raw_data_path path must contain the term *RawData* and analysis_file_path must contain the term *AnalysisResults* as the TSA software automatically assigns this when exporting data. Data is loaded from the `read_raw_data` and `read_analysis` functions within this merge_TSA function.

`protein`
: can be used to select for an individual or multiple protein(s) as a character string matching protein names assigned in the TSA software. NA by default.

`ligand`
: can be used to select for an individual or multiple ligand(s) as a character string matching ligand names assigned in the TSA software. NA by default.

Value

A data frame of merged TSA data.

IDs

The TSAR package relies on matching conditions and file names for each well and for each set of conditions between multiple files output by the TSA software. Conditions are assigned to individual wells within the TSA software; these assigned values are detected by `read_analysis` and `read_raw_data` then are converted into IDs. Ensure your labeling of values within the TSA software is consistent so that similar values can be merged - typos or varying terms will be treated as distinct values within TSAR unless the values are manually specified by the user. Automatically generated well IDs within a TSA file can be found using the `well_IDs` function; condition IDs can be found using the `condition_IDs` function.

Condition IDs are generated only in the `read_analysis`, see that function's documentation for more details. Condition IDs are assigned to raw data in the `merge_TSA` function.

Well IDs are similar to Condition IDs, as they are generated from columns in TSA output. Well IDs are used to match the analysis and raw data files for the same experiment, as both files contain unique, useful information for each well. The well ID includes the .eds file name saved from the PCR machine to match equivalent wells between files of the same experiment. Each well on all plates should have a unique well ID. If you wish to change or specify the file name used for the well ID, a new name can be manually assigned with the "manual_file" argument.

See Also

`read_raw_data` and `read_analysis` for loading data.

Other TSAR Formatting: `TSA_Tms()`, `TSA_average()`, `Tm_difference()`, `merge_norm()`, `normalize_fluorescence()`, `rescale()`

Examples

```
# note: example does not contain example data to run
# merge_TSA(analysis_file_path, raw_data_path)
```

`model_boltzmann` *Boltzmann Modeling on TSA data*

Description

Function finds fitted fluorescence values by imposing Boltzmann function.

Usage

```
model_boltzmann(norm_data)
```

Arguments

`norm_data`	data frame input, preferably normalized using `normalize`.

Value

dtaa frame containing gam model fitted values

See Also

Other data_preprocess: model_fit(), model_gam(), normalize(), remove_raw(), run_boltzmann(), screen(), view_model(), weed_raw()

Examples

```
data("qPCR_data1")
A01 <- subset(qPCR_data1, Well.Position == "A01")
A01 <- normalize(A01)
model_boltzmann(A01)
```

---

model_fit *Refit and calculate derivative function*

Description

Model_fit calculates derivatives by refitting model onto data. Only runs on data of a single well.

Usage

```
model_fit(norm_data, model, smoothed)
```

Arguments

norm_data	data frame; the raw data set input
model	fitted model containing fitted values
smoothed	inform whether data already contains a smoothed model; Input the column name of the smoothed data to override values of gam model fitting. For example, existing "Fluorescence" column contains data already smoothed, set smoothed = "Flourescence" to calculate derivative function upon the called smoothed data.

Value

data frame; with calculated derivative columns

See Also

Other data_preprocess: model_boltzmann(), model_gam(), normalize(), remove_raw(), run_boltzmann(), screen(), view_model(), weed_raw()

Examples

```
data("qPCR_data1")
test <- subset(qPCR_data1, Well.Position == "A01")
test <- normalize(test, fluo = 5, selected = c(
    "Well.Position", "Temperature",
    "Fluorescence", "Normalized"
))
gammodel <- model_gam(test, x = test$Temperature, y = test$Normalized)
model_fit(test, model = gammodel)
# if data come smoothed, run ...
model_fit(test, smoothed = "Fluorescence")
```

`model_gam` *Generalized Addidtive Modeling on TSA data*

Description

Function finds fitted fluorescence values by imposing generalized additive model on fluorescence data by temperature. Model assumes method = "GACV.Cp" and sets to `formula = y ~ s(x, bs = "ad")`. Function inherits function from gam package, `gam()`.

Usage

```
model_gam(norm_data, x, y)
```

Arguments

`norm_data`	data frame input of only one well's reading, preferably normalized using `normalize`.
`x`	temperature column
`y`	normalized fluorescence column

Value

data frame containing gam model fitted values

See Also

Other data_preprocess: `model_boltzmann()`, `model_fit()`, `normalize()`, `remove_raw()`, `run_boltzmann()`, `screen()`, `view_model()`, `weed_raw()`

Examples

```
data("qPCR_data1")
test <- subset(qPCR_data1, Well.Position == "A01")
test <- normalize(test, fluo = 5, selected = c(
    "Well.Position", "Temperature",
    "Fluorescence", "Normalized"
))
model_gam(test, x = test$Temperature, y = test$Normalized)
```

normalize *Normalize Fluorescence*

Description

normalize() reads in raw_data. This function normalizes data by standardizing them according to maximum and minimum fluorescence per well, with maximum set to 1 and minimum set to 0. It also reformats data types by checking for potential error. i.e. a string specifying 100,000 will be read in as number, 100000, without issue. Function is applicable only to data of a single well, do not call on an entire data frame of all 96 well data. It is intended for single well screening purposes.

Usage

```
normalize(
  raw_data,
  fluo = NA,
  selected = c("Well.Position", "Temperature", "Fluorescence", "Normalized")
)
```

Arguments

raw_data	data frame; raw dataset input, should be of only one well. If multiple wells need to be normalized, use gam_analysis() for 96 well application. If only preliminary screening is needed, use screen().
fluo	integer; the Fluorescence variable column id (e.g. fluo = 5 when 5th column of the data frame is the Fluorescence value) if fluorescence variable is named exactly as "Fluorescence", fluo does not need to be specified. i.e. fluo is set to NA by default, suggesting the variable is named "Fluorescence".
selected	list of character strings; variables from the original data set users intend to keep. Variable default set to c("Well.Position", "Temperature", "Fluorescence", "Normalized") if not otherwise specified. If data frame variables are named differently, user needs to specify what column variables to keep.

Value

cleaned up data framed with selected columns

See Also

Other data_preprocess: model_boltzmann(), model_fit(), model_gam(), remove_raw(), run_boltzmann(), screen(), view_model(), weed_raw()

Examples

```
data("qPCR_data1")
test <- subset(qPCR_data1, Well.Position == "A01")
normalize(test)
```

`normalize_fluorescence` *Normalize Fluorescence Curve*

Description

This function will take the TSA data and normalize the arbitrary fluorescence measurements based on the specified method. Each well, determined by a unique well ID, is normalized independently. All measurements can be normalized to the minimum or maximum value. Alternatively, setting by = "rescale" (the default) will cause all values to be normalized between the minimum and maximum values, with the maximum = 1 and the minimum = 0 and all other values normalized in-between. Finally, the user can supply a single value or vector of values to normalize the data to. The returned data frame will be the input tsa data frame with a new column named "RFU" containing the normalized TSA data.

Usage

```
normalize_fluorescence(tsa_data = tsa_data, by = "rescale", control_vect = NA)
```

Arguments

tsa_data	a data frame that is unmerged and generated by TSAR::read_raw_data() or a merged TSA data frame generated by TSAR::merge_TSA(). Data frames require a column named "Fluorescence" containing numeric values for normalizing.
by	character string; either c("rescale", "min", "max", "control"). by = "rescale" by default, scaling Fluorescence values in-between the minimum and maximum observation. Each well can be normalized to either the minimum or maximum value with by = "min" or by = "max", respectively. To normalize all values to a numeric value or vector, set by = "control".
control_vect	numeric vector to normalize the column "Fluorescence" to. An individual number will normalize all measurements to be normalized to it. The vector will need to align with tsa_data$Fluorescence. Ensure by = "control", else the supplied vector will be ignored.

Value

a data frame identical to the tsa_data input with a new column named "RFU" containing the normalized values

See Also

read_raw_data and merge_TSA for loading data.

Other TSAR Formatting: TSA_Tms(), TSA_average(), Tm_difference(), merge_TSA(), merge_norm(), rescale()

Examples

```
# examples not ran without example dataset
# raw_data <- read_raw_data(raw_data_path)
# normalize_fluorescence(raw_data, by == "control)
```

qPCR_data1 *qPCR_data1 Dataset*

Description

Dataset Description: This dataset contains qPCR data for the CA121 protein and common vitamins. It provides fluorescence measurements obtained using QuantStudio3. Dataset is experimentally obtained by author of this package.

Usage

```
data(qPCR_data1)
```

Format

A data frame with the following columns:

Well Well Count, not required for user

Well.Position Well Label, i.e. A01; required input

Reading reading count in time series, not required for user

Temperature temperature reading, required input

Fluorescence fluorescence reading, required input

Value

qPCR_data1 data frame

Source

experimentally obtained

qPCR_data2 *qPCR_data2 Dataset*

Description

Dataset Description: This dataset contains qPCR data for the CA121 protein and common vitamins. It provides fluorescence measurements obtained using QuantStudio3. A different experiemnt trial containing data of similar property as data, qPCR_data1. Dataset is experimentally obtained by author of this package.

Usage

```
data(qPCR_data2)
```

Format

A data frame with the following columns:

Well Well Count, not required for user

Well.Position Well Label, i.e. A01; required input

Reading reading count in time series, not required for user

Temperature temperature reading, required input

Fluorescence fluorescence reading, required input

Value

qPCR_data2 data frame

---

read_analysis *Read TSA Analysis Data*

Description

Open TSA Analysis files. This function is used to load data output from the thermal shift software analysis tab. Can be either .txt or .csv file with a path / file name including the string "Analysis-Results" due to its automatic naming from the software. The values assigned to wells within the TSA software are automatically extracted from the loaded file; values must be assigned within the TSA software for the automated workflow (See IDs Section Below). **Note:** Wells that do not have an Analysis Group assigned are removed. The TSA software automatically assigns all wells to Analysis Group 1 by default, and can be changed but not removed by the software.

Usage

```
read_analysis(
  path,
  type = "derivative",
  conditions = c("Protein", "Ligand"),
  manual_conditions = NA,
  manual_wells = NA,
  skip_flags = FALSE,
  manual_file = NA
)
```

Arguments

path	a character string; the path or the name of the file which the 'AnalysisResults' data are to be read from. Either a .txt or .csv file. The path must contain the term *AnalysisResults* as the TSA software automatically assigns this when exporting data.
type	either c("boltzmann", "derivative"); `type = "derivative")` by default. Determines what model of Tm estimation to load from the TSA software. Loads Tms as 'Tm B' when `type = "boltzmann")`; loads Tms as 'Tm D' when `type = "derivative")`.

`conditions` A character vector of condition types assigned within the TSA software to load. `conditions = c("Protein", "Ligand")` by default. These conditions are used to generate the IDs discussed.

`manual_conditions, manual_wells` NA by default, enabling automated analysis. A character vector of Condition IDs and Well IDs to manually assign each row of the read data.

`skip_flags` logical value; `type = FALSE` by default. When `type = TRUE`, wells that have flags reported by TSA software are removed.

`manual_file` NA by default. User can specify .eds for merging if needed for Well IDs if needed with a character string.

Value

A data frame of TSA analysis data.

IDs

The TSAR package relies on matching conditions and file names for each well and for each set of conditions between multiple files output by the TSA software. Conditions are assigned to individual wells within the TSA software; these assigned values are detected by `read_analysis` and `read_raw_data` then are converted into IDs. Ensure your labeling of values within the TSA software is consistent so that similar values can be merged - typos or varying terms will be treated as distinct values within TSAR unless the values are manually specified by the user. Automatically generated well IDs within a TSA file can be found using the `well_IDs` function; condition IDs can be found using the `condition_IDs` function.

Condition IDs are generated from columns in TSA output specified by the 'conditions' argument. Protein and Ligand values, the default conditions within the TSA software, are the values used to create these IDs. You can manually specify the condition categories from the TSA software, including user-made conditions. Condition IDs are used to match equivalent observations between technical and biological replicates. Wells with identical condition IDs, specified by the 'conditions' argument, will be aggregated in down-stream analysis; user-specified conditions must remain consistent in use and order to create compatible IDs between TSA files from the same experiment and between replicates.

Well IDs are similar to Condition IDs, as they are generated from columns in TSA output that are specified by the 'conditions' argument. Well IDs are used to match the analysis and raw data files for the same experiment, as both files contain unique, useful information for each well. In addition to the condition ID, the well ID includes the .eds file name saved from the PCR machine to match equivalent wells between files of the same experiment. Each well on all plates should have a unique well ID. If you wish to change or specify the file name used for the well ID, a new name can be manually assigned with the "manual_file" argument.

The user may manually assign condition IDs using the 'manual_conditions' argument rather than using the automatically generated IDs. The same is true for well IDs, which can be manually assigned with 'manual_wells'. This is not suggested, as there may be issues with matching if well/conditions are not properly matching. This gives the potential for errors in downstream applications as well.

See Also

read_raw_data for loading accompanying data. merge_TSA for joining Analysis Results and Raw Data files from the TSA software.

Other Read TSA Data: read_raw_data()

Examples

```
path <- "~/Desktop/analysis_data"
# note: example does not contain example data to run
# read_analysis(path)
```

---

read_raw_data *Read TSA Raw Data*

Description

Open TSA Raw Data files. This function is used to load data output from the thermal shift software Raw Data tab. Can be either .txt or .csv file with a path / file name including the string "Raw-Data" due to its automatic naming from the software. The values assigned to wells within the TSA software are automatically extracted from the loaded file; values must be assigned within the TSA software for the automated workflow (See IDs Section Below).

Usage

```
read_raw_data(path, manual_file = NA, type = "fluorescence")
```

Arguments

path	a character string; the path or the name of the file which the 'RawData' data are to be read from. Either a .txt or .csv file. The path must contain the term *RawData* as the TSA software automatically assigns this when exporting data.
manual_file	NA by default. User can specify .eds for merging if needed for Well IDs with a character string.
type	either c("boltzmann", "derivative", "fluorescence"); type = "fluorescence") by default. Determines what data track to load. When type = "fluorescence"), the arbitrary fluorescence of the TSA dye is loaded; this is the primary data. Alternately, derivatives van be loaded: Loads data as boltzman estimated tracks when type = "boltzmann"); loads the 2nd derivative of emissions when type = "derivative").

Value

A data frame of TSA raw data.

IDs

The TSAR package relies on matching conditions and file names for each well and for each set of conditions between multiple files output by the TSA software. Conditions are assigned to individual wells within the TSA software; these assigned values are detected by `read_analysis` and `read_raw_data` then are converted into IDs. Ensure your labeling of values within the TSA software is consistent so that similar values can be merged - typos or varying terms will be treated as distinct values within TSAR unless the values are manually specified by the user. Automatically generated well IDs within a TSA file can be found using the `well_IDs` function; condition IDs can be found using the `condition_IDs` function.

Condition IDs are generated only in the `read_analysis`, see that function's documentation for more details. Condition IDs are assigned to raw data in the `merge_TSA` function.

Well IDs are similar to Condition IDs, as they are generated from columns in TSA output. Well IDs are used to match the analysis and raw data files for the same experiment, as both files contain unique, useful information for each well. The well ID includes the .eds file name saved from the PCR machine to match equivalent wells between files of the same experiment. Each well on all plates should have a unique well ID. If you wish to change or specify the file name used for the well ID, a new name can be manually assigned with the "manual_file" argument.

See Also

`read_analysis` for loading accompanying data. `merge_TSA` for joining Analysis Results and Raw Data files from the TSA software.

Other Read TSA Data: `read_analysis()`

Examples

```
path <- "~/Desktop/raw_data"
# note: example does not contain example data to run
# read_raw_data(path)
```

`read_tsar` *Read analysis result*

Description

reads previous pipeline output lists from `gam_analysis()` and organizes them into separate data frames.

Usage

```
read_tsar(gam_result, output_content)
```

Arguments

gam_result	list; input uses resulting output of gam_analysis() function
output_content	integer; output_content = 0 returns only the tm value by wells output_content = 1 returns data table with fitted values output_content = 2 returns the combination of 0 and 1

Value

output files with select dataset

See Also

Other read_write_analysis: join_well_info(), write_tsar()

Examples

```
data("qPCR_data1")
result <- gam_analysis(qPCR_data1,
    smoothed = TRUE, fluo_col = 5,
    selections = c(
        "Well.Position", "Temperature", "Fluorescence", "Normalized"
    )
)
read_tsar(result, output_content = 0)
output_data <- read_tsar(result, output_content = 2)
```

---

remove_raw *Remove selected raw curves*

Description

Removes selected curves with specified wells and range.

Usage

```
remove_raw(raw_data, removerange = NULL, removelist = NULL)
```

Arguments

raw_data	dataframe; to be processed data
removerange	list type input identifying range of wells to select. For example, if removing all 12 wells from row D to H is needed, one can specify the row letters and column numbers like this: removerange = c("D", "H", "1", "12")
removelist	use this parameter to remove selected Wells with full Well names. For example, removelist = c('A01', 'D11')

Value

dataframe; data frame with specified well removed

See Also

Other data_preprocess: model_boltzmann(), model_fit(), model_gam(), normalize(), run_boltzmann(), screen(), view_model(), weed_raw()

Examples

```
data("qPCR_data1")
remove_raw(qPCR_data1, removelist = c("A01", "D11"))
```

`rescale` *Rescale values between minimum and maximum.*

Description

For a vector of numeric values, the minimum and maximum values are determined and each value of the vector is rescaled between 0 and 1. Values near 0 are close to the minimum, values near 1 are close to the max. This function is utilized by other TSAR functions.

Usage

```
rescale(x)
```

Arguments

x	a numeric vector to be rescaled

Value

A numeric vector of rescaled values.

See Also

Other TSAR Formatting: TSA_Tms(), TSA_average(), Tm_difference(), merge_TSA(), merge_norm(), normalize_fluorescence()

Examples

```
x <- c(0, 1, 3)
rescale(x)
```

run_boltzmann *Run Boltzmann Modeling*

Description

Function runs function `model_boltzmann()` and raises warning when modeling generates error or warnings.

Usage

```
run_boltzmann(norm_data)
```

Arguments

`norm_data`	data frame input, preferably normalized using `normalize`.

Value

data frame containing gam model fitted values

See Also

Other data_preprocess: `model_boltzmann()`, `model_fit()`, `model_gam()`, `normalize()`, `remove_raw()`, `screen()`, `view_model()`, `weed_raw()`

Examples

```
data("qPCR_data1")
A01 <- subset(qPCR_data1, Well.Position == "A01")
A01 <- normalize(A01)
run_boltzmann(A01)
```

screen *Screen raw curves*

Description

screens multiple wells of data and prepares to assist identification of corrupted wells and odd out behaviors

Usage

```
screen(raw_data, checkrange = NULL, checklist = NULL)
```

Arguments

`raw_data`	input raw_data
`checkrange`	list type input identifying range of wells to select. For example, if viewing first 8 wells from row A to C is needed, one can specify the row letters and column numbers like this: `checkrange = c("A", "C", "1", "8")`
`checklist`	use this parameter to view selected Wells with full Well names. For example, `checklist = c('A01', 'D11')`

Value

returns a ggplot graph colors by well IDs

See Also

Other data_preprocess: `model_boltzmann()`, `model_fit()`, `model_gam()`, `normalize()`, `remove_raw()`, `run_boltzmann()`, `view_model()`, `weed_raw()`

Examples

```
data("qPCR_data1")
screen(qPCR_data1, checkrange = c("A", "C", "1", "12"))
```

Tm_difference *Calculate Tm difference for all conditions*

Description

From a specified control condition, the change in Tm is calculated for each condition in the tsa_data. Specifically, $Tm = condition - control$. Individual Tm values are averaged by condition, see `TSA_average` for details. To see all conditions use `condition_IDs(tsa_data)`.

Usage

```
Tm_difference(tsa_data, control_condition)
```

Arguments

`tsa_data`	a data frame that is merged and generated by TSAR::merge_TSA(). If `y = 'RFU'`, tsa_data must also be generated by TSAR::normalize_fluorescence. The Temperature column will be rounded and the average & sd of each rounded temperature is calculated.
`control_condition`	character string matching a Condition ID. Must be equal to a value within tsa_data$condition_ID. See unique condition IDs with `condition_IDs`.

Value

a data frame of reformatted data with the `TSA_average` data and the Tm.

See Also

`merge_TSA` for preparing data. `TSA_average` for more information on the output data. `condition_IDs` to get unique Condition IDs within the input. `TSA_boxplot` for application.

Other TSAR Formatting: `TSA_Tms()`, `TSA_average()`, `merge_TSA()`, `merge_norm()`, `normalize_fluorescence()`, `rescale()`

Examples

```
data("example_tsar_data")
control <- condition_IDs(example_tsar_data)[1]
Tm_difference(example_tsar_data, control_condition = control)
```

`Tm_est` *Find inflection point function*

Description

Looks for Tm temperature values by finding the inflection point in the fluorescence data. The inflection point is approximated by locating the maximum first derivative stored in "norm_deriv" column.

Usage

```
Tm_est(norm_data, min, max)
```

Arguments

`norm_data`	data frame; data frame input containing derivative values can only be data frames for one well; finding inflections points across multiple wells require iteration through individual wells
`min`	restricts finding to be above the given minimum temperature
`max`	restricts finding to be below the given maximum temperature parameter min and max can be used to remove messy or undesired data for better accuracy in tm estimation; removing data is before fitting the model is more recommended than removing here

Value

integer; tm estimation

See Also

Other tsa_analysis: `gam_analysis()`

Examples

```
data("qPCR_data1")
test <- subset(qPCR_data1, Well.Position == "A01")
test <- normalize(test, fluo = 5, selected = c(
    "Well.Position", "Temperature",
    "Fluorescence", "Normalized"
))
gammodel <- model_gam(test, x = test$Temperature, y = test$Normalized)
fit <- model_fit(test, model = gammodel)
Tm_est(fit)
```

TSA_average *Average TSA Curves*

Description

This function will take either Fluorescence or Normalized Fluorescence curves from the submitted data frame and find the average (mean) and standard deviation (sd) for each temperature measured in the TSA curve. Mean and sd are smoothened by default to generate cleaner curves. The function gam from the mgcv package is used for regression to smoothen lines. Smoothing can be turned off and the true average for each point can be given, however, plots will look messier. The qPCR machine may return temperatures with many decimal places, and TSAR only merges identical values, therefore rounding is necessary. Data is rounded to one decimal place to improve regression smoothing.

Note: All submitted data is averaged, regardless of condition or well ID. If you wish to average by condition, you will need to sort the data frame and run this function on subsets.

Usage

```
TSA_average(
  tsa_data,
  y = "Fluorescence",
  digits = 1,
  avg_smooth = TRUE,
  sd_smooth = TRUE
)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(). If y = 'RFU', tsa_data must also be generated by TSAR::normalize_fluorescence. The Temperature column will be rounded and the average & sd of each rounded temperature is calculated.
y	character string; c('Fluorescence', 'RFU'). When y = 'Fluorescence', the original Fluorescence data from TSAR::read_raw_data() is averaged. When y = 'RFU', the average is calculated by the rescaled fluorescence.
digits	an integer; digits = 1 by default. The number of decimal places to round temperature to for averaging.
avg_smooth, sd_smooth	logical; TRUE by default. Decides if the average (avg_smooth) or standard deviation (sd_smooth) will be smoothened by regression via mgcv::gam()

Value

a data frame of each temperature measured with the average, sd, and n(# of averaged values) calculated. Depending on avg_smooth and sd_smooth, the smoothened lines for the maximum and mimimum sd and the average will also be returned.

See Also

merge_TSA and merge_TSA for preparing data.

Other TSAR Formatting: TSA_Tms(), Tm_difference(), merge_TSA(), merge_norm(), normalize_fluorescence(), rescale()

Examples

```
data("example_tsar_data")
TSA_average(example_tsar_data,
    y = "Fluorescence", digits = 1,
    avg_smooth = TRUE, sd_smooth = TRUE
)
```

TSA_boxplot *TSA Box Plot*

Description

Generates a box and whiskers plot for each condition specified. This is used to compare Tm values between the data set. See Tm_difference for details.

Usage

```
TSA_boxplot(
  tsa_data,
  control_condition = NA,
  color_by = "Protein",
  label_by = "Ligand",
  separate_legend = TRUE
)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(). If y = 'RFU', tsa_data must also be generated by TSAR::normalize_fluorescence. The Temperature column will be rounded and the average & sd of each rounded temperature is calculated.
control_condition	Either a condition_ID or NA; NA by default. When a valid Condition ID is provided, a vertical line appears at the average Tm for the specified condition. When NA, this is skipped.
color_by	character string, either c("Ligand", "Protein"). The condition category to color the boxes within the box plot for comparison. This is represented in the legend. Set to NA to skip.
label_by	character string, either c("Ligand", "Protein"). The condition category to group the boxes within the box plot. This is represented in the axis. Set to NA to skip.
separate_legend	logical; separate_legend = TRUE by default. When TRUE, the ggplot2 legend is separated from the TSA curve. This is to help with readability. One ggplot is returned when FALSE.

Value

by default, two ggplots are returned: one TSA curve and one key. When `separate_legend = FALSE` one ggplot is returned.

See Also

`merge_TSA` for preparing data. See `Tm_difference` and `get_legend` for details on function parameters.

Other TSA Plots: `TSA_compare_plot()`, `TSA_wells_plot()`, `get_legend()`, `graph_tsar()`, `view_deriv()`

Examples

```
data("example_tsar_data")
TSA_boxplot(example_tsar_data,
    color_by = "Protein",
    label_by = "Ligand", separate_legend = FALSE
)
```

TSA_compare_plot *Compare TSA curves to control*

Description

Generate a number of plots based on the input data to compare the average and standard deviation (sd) of each unique condition to a specified control condition. To see all conditions use `condition_IDs(tsa_data)`.

Usage

```
TSA_compare_plot(
  tsa_data,
  control_condition,
  y = "Fluorescence",
  show_Tm = FALSE,
  title_by = "both",
  digits = 1
)
```

Arguments

`tsa_data`	a data frame that is merged and generated by TSAR::merge_TSA(). If `y = 'RFU'`, tsa_data must also be generated by TSAR::normalize_fluorescence. The Temperature column will be rounded and the average & sd of each rounded temperature is calculated.
`control_condition`	character string matching a Condition ID. Must be equal to a value within tsa_data$condition_ID. See unique condition IDs with `condition_IDs`.
`y`	character string; c('Fluorescence', 'RFU'). When `y = 'Fluorescence'`, the original Fluorescence data from TSAR::read_raw_data() is averaged. When `y = 'RFU'`, the average is calculated by the rescaled fluorescence.

show_Tm	logical; show_Tm = FALSE by default. When TRUE, the Tm is displayed on the plot. When FALSE, the Tm is not added to the plot.
title_by	character string; c("ligand", "protein", "both"). Automatically names the plots by the specified condition category.
digits	integer; the number of decimal places to round for change in Tm calculations displayed in the subtitle of each plot..

Value

Generates a number of ggplot objects equal to the number of unique Condition IDs present in the input data.

See Also

merge_TSA and normalize_fluorescence for preparing data. See TSA_average and get_legend for details on function parameters. See TSA_wells_plot for individual curves of the averaged conditions shown.

Other TSA Plots: TSA_boxplot(), TSA_wells_plot(), get_legend(), graph_tsar(), view_deriv()

Examples

```
data("example_tsar_data")
TSA_compare_plot(example_tsar_data,
    y = "RFU",
    control_condition = "CA FL_DMSO"
)
```

TSA_ligands *TSA Ligands*

Description

This function is used to extract information from a data frame of TSA data. The Ligand values should be assigned in the TSA software.

Usage

```
TSA_ligands(tsa_data, n = FALSE)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(), or an unmerged data frame read by TSAR::read_analysis() or TSAR::read_raw_data(). The data frame must have a column named 'Ligand'.
n	logical value; n = FALSE by default. When TRUE, a numeric value describing the number of unique ligand names is returned. When FALSE, a character vector of unique IDs are returned.

Value

Either a character vector of unique well_IDs or a numeric value.

See Also

merge_TSA, read_raw_data, and read_analysis for preparing input.

Other TSA Summary Functions: TSA_proteins(), condition_IDs(), well_IDs()

Examples

```
data("example_tsar_data")
TSA_ligands(example_tsar_data)
```

TSA_proteins *TSA Proteins*

Description

This function is used to extract information from a data frame of TSA data. The Protein values should be assigned in the TSA software.

Usage

```
TSA_proteins(tsa_data, n = FALSE)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(), or an unmerged data frame read by TSAR::read_analysis() or TSAR::read_raw_data(). The data frame must have a column named 'Protein'.
n	logical value; n = FALSE by default. When TRUE, a numeric value describing the number of unique protein names is returned. When FALSE, a character vector of unique IDs are returned.

Value

Either a character vector of unique well_IDs or a numeric value.

See Also

merge_TSA, read_raw_data, and read_analysis for preparing input.

Other TSA Summary Functions: TSA_ligands(), condition_IDs(), well_IDs()

Examples

```
data("example_tsar_data")
TSA_proteins(example_tsar_data)
```

TSA_Tms *Reformat TSA data into TSA Tms*

Description

This function is used to output calculated Tm data from TSA analysis. The input data frame will be transformed into a new format that is helpful for user reading and automated analysis. The Tm values can be listed as a data frame of individual wells or the Tms from identical conditions can be averaged. When condition_average is TRUE (the default), samples with identical condition IDs will be aggregated and the average / standard deviation will be calculated where appropriate. To analyze multiple TSA experiments, use merge_TSA() to make a single data frame for analysis.

Usage

```
TSA_Tms(analysis_data, condition_average = TRUE)
```

Arguments

analysis_data	a data frame that is unmerged and generated by TSAR::read_analysis() or a merged TSA data frame generated by TSAR::merge_TSA(). Data frames require a column named 'condition_ID' for averaging.
condition_average	logical value; n = TRUE by default. When TRUE, the average Tm is calculated by matched condition IDs within the data frame. When FALSE, each well is reported as a unique value with the corresponding Tm.

Value

A data frame of Tm values.

See Also

merge_TSA, read_raw_data, and read_analysis for preparing input.

Other TSAR Formatting: TSA_average(), Tm_difference(), merge_TSA(), merge_norm(), normalize_fluorescenc
rescale()

Examples

```
data("example_tsar_data")
TSA_Tms(example_tsar_data)
```

TSA_wells_plot *TSA Well Curves Plot*

Description

Generates the individual curves for each well in the merged tsa data input. Options to create an average and standard deviation sd of the plot in addition to the individual curves. The average and sd will be smoothened by linear regression; see TSA_average for details.

Usage

```
TSA_wells_plot(
  tsa_data,
  y = "RFU",
  show_Tm = TRUE,
  Tm_label_nudge = 7.5,
  show_average = TRUE,
  plot_title = NA,
  plot_subtitle = NA,
  smooth = TRUE,
  separate_legend = TRUE
)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(). If y = 'RFU', tsa_data must also be generated by TSAR::normalize_fluorescence. The Temperature column will be rounded and the average & sd of each rounded temperature is calculated.
y	character string; c('Fluorescence', 'RFU'). When y = 'Fluorescence', the original Fluorescence data from TSAR::read_raw_data() is averaged. When y = 'RFU', the average is calculated by the rescaled fluorescence.
show_Tm	logical; show_Tm = TRUE by default. When TRUE, the Tm is displayed on the plot. When FALSE, the Tm is not added to the plot.
Tm_label_nudge	numeric; Tm_label_nudge = 7.5 the direction in the x direction to move the Tm label. This is used prevent the label from covering data. Ignored if show_Tm = FALSE.
show_average	logical; show_average = TRUE by default. When TRUE, the average is and sd is plotted as generated by merge_TSA.
plot_title, plot_subtitle	characer string, NA by default. User-specified plots to overright automatic naming.
smooth	logical; smooth = TRUE by default. When TRUE, linear regression by gam is used to make clean lines on the plot. See TSA_average for more details. When FALSE, individual points are plotted (slows down rendering).
separate_legend	logical; separate_legend = TRUE by default. When TRUE, the ggplot2 legend is separated from the TSA curve. This is to help with readability. One ggplot is returned when FALSE.

Value

by default, two ggplots are returned: one TSA curve and one key. When `separate_legend = FALSE` one ggplot is returned.

See Also

`merge_TSA` and `normalize_fluorescence` for preparing data. See `TSA_average` and `get_legend` for details on function parameters.

Other TSA Plots: `TSA_boxplot()`, `TSA_compare_plot()`, `get_legend()`, `graph_tsar()`, `view_deriv()`

Examples

```
data("example_tsar_data")
check <- subset(example_tsar_data, condition_ID == "CA FL_PyxINE HCl")
TSA_wells_plot(check, separate_legend = FALSE)
```

---

view_deriv *View Derivative Curves*

Description

Function reviews data by well and output a graph of the all derivatives wanted. Function called within graph_tsar function but also runnable outside.

Usage

```
view_deriv(tsar_data, frame_by = "Well")
```

Arguments

`tsar_data`	dataset input, analyzed must have norm_deriv as a variable; dataset qualifying norm_data or tsar_data both fulfills this parameter, although tsar_data is more recommended given more data options.
`frame_by`	builds plotly by specified frame variable. To graph by a concentration gradient, well position, or other specified variable, simple specify `frame_by = "condition_ID"`. To view all derivative curves without frames, set to `frame_by = FALSE`, else it is defaulted to frame by well labels.

Value

plotly object of derivative curves

See Also

Other TSA Plots: `TSA_boxplot()`, `TSA_compare_plot()`, `TSA_wells_plot()`, `get_legend()`, `graph_tsar()`

Examples

```
data("example_tsar_data")
view_deriv(example_tsar_data, frame_by = "condition_ID")
```

view_model *View Model*

Description

Function reviews data by well and output a graph of the fit and a graph of derivative. Function called within analyze_norm function.

Usage

```
view_model(raw_data)
```

Arguments

raw_data	dataset input, not processing needed

Value

list of two ggplot graphs

See Also

Other data_preprocess: model_boltzmann(), model_fit(), model_gam(), normalize(), remove_raw(), run_boltzmann(), screen(), weed_raw()

Examples

```
data("qPCR_data1")
test <- subset(qPCR_data1, Well.Position == "A01")
test <- normalize(test)
gammodel <- model_gam(test, x = test$Temperature, y = test$Normalized)
test <- model_fit(test, model = gammodel)
view_model(test)
```

weed_raw *Weed raw data for corrupt curves*

Description

The weed_raw function allows users to interact with a screening graph and select curves to weed out before entering analysis. Function wraps together screen and remove_raw.

Usage

```
weed_raw(raw_data, checkrange = NULL, checklist = NULL)
```

Arguments

raw_data	The raw data for screening.
checkrange	list type input identifying range of wells to select. For example, if viewing first 8 wells from row A to C is needed, one can specify the row letters and column numbers like this: checkrange = c("A", "C", "1", "8")
checklist	use this parameter to view selected Wells with full Well names. For example, checklist = c('A01', 'D11')

Value

prompts separate app window for user interaction, does not return specific value

See Also

screen and remove_raw

Other data_preprocess: model_boltzmann(), model_fit(), model_gam(), normalize(), remove_raw(), run_boltzmann(), screen(), view_model()

Examples

```
data("qPCR_data1")
if (interactive()) {
    runApp(weed_raw(qPCR_data1, checkrange = c("A", "B", "1", "12")))
}
```

---

well_IDs *TSAR Well IDs*

Description

This function is used to extract information of the well IDs from a merged TSA data frame. Well IDs are automatically generated by the read_analysis and read_raw_data functions in the automated workflow. This function returns either a character vector of unique IDs present or a numeric value of the number of unique IDs.

Usage

```
well_IDs(tsa_data, n = FALSE)
```

Arguments

tsa_data	a data frame that is merged and generated by TSAR::merge_TSA(), or an unmerged data frame read by TSAR::read_analysis() or TSAR::read_raw_data(). Data frames require a column named 'well_ID'.
n	logical value; n = FALSE by default. When TRUE, a numeric value of unique IDs is returned. When FALSE, a character vector of unique IDs are returned.

Value

Either a character vector of unique well IDs or a numeric value.

See Also

merge_TSA, read_raw_data, and read_analysis for preparing input.

Other TSA Summary Functions: TSA_ligands(), TSA_proteins(), condition_IDs()

Examples

```
data("example_tsar_data")
well_IDs(example_tsar_data)
```

---

well_information *example well information Data*

Description

Dataset Description: This file is a readin using well_information_template. File contains the conditions of well, specifying protein and ligand content in well. All experimental setup and relevant information are determined and manually put in by the author of this package.

Usage

```
data(well_information)
```

Format

A data frame with the following columns:

...1 n/a

Protein...2 Protein in Well 1

Ligand...3 Ligand in Well 1

Protein...4 Protein in Well 2

Ligand...5 Ligand in Well 2

Protein...6 Protein in Well 3

Ligand...7 Ligand in Well 3

Protein...8 Protein in Well 4

Ligand...9 Ligand in Well 4

Protein...10 Protein in Well 5

Ligand...11 Ligand in Well 5

Protein...12 Protein in Well 6

Ligand...13 Ligand in Well 6

Protein...14 Protein in Well 7

Ligand...15 Ligand in Well 7

Protein...16 Protein in Well 8

Ligand...17 Ligand in Well 8

Protein...18 Protein in Well 9

Ligand...19 Ligand in Well 9

Protein...20 Protein in Well 10

Ligand...21 Ligand in Well 10

Protein...22 Protein in Well 11

Ligand...23 Ligand in Well 11

Protein...24 Protein in Well 12

Ligand...25 Ligand in Well 12

Value

well information data frame

`well_information_template`

Well Information Template

Description

Dataset Description: Template specifies the way condition information will be read in as, specifying protein and ligand content in well.

Usage

```
data(well_information_template)
```

Format

A data frame with the following columns:

...1 n/a

Protein...2 Protein in Well 1

Ligand...3 Ligand in Well 1

Protein...4 Protein in Well 2

Ligand...5 Ligand in Well 2

Protein...6 Protein in Well 3

Ligand...7 Ligand in Well 3

Protein...8 Protein in Well 4

Ligand...9 Ligand in Well 4

Protein...10 Protein in Well 5

Ligand...11 Ligand in Well 5

Protein...12 Protein in Well 6

Ligand...13 Ligand in Well 6

Protein...14 Protein in Well 7

Ligand...15 Ligand in Well 7

Protein...16 Protein in Well 8

Ligand...17 Ligand in Well 8

Protein...18 Protein in Well 9
Ligand...19 Ligand in Well 9
Protein...20 Protein in Well 10
Ligand...21 Ligand in Well 10
Protein...22 Protein in Well 11
Ligand...23 Ligand in Well 11
Protein...24 Protein in Well 12
Ligand...25 Ligand in Well 12

Value

well information template in data frame

`write_tsar` *write output files*

Description

writes output into csv or txt files

Usage

```
write_tsar(data, name, file = "txt")
```

Arguments

`data`	input data frame
`name`	string, name file to be saved as. Final name will be appended "tsar_output"
`file`	`file = "txt"` writes txt output files; `file = "csv"` writes csv output files; default set to `file = "txt"`

Value

file output on the working directory where data was read in

See Also

Other read_write_analysis: `join_well_info()`, `read_tsar()`

Examples

```
data("qPCR_data1")
result <- gam_analysis(qPCR_data1,
    smoothed = TRUE, fluo_col = 5,
    selections = c(
        "Well.Position", "Temperature", "Fluorescence", "Normalized"
    )
)
output_data <- read_tsar(result, output_content = 2)
# example does not run, will build excessive file in package
# write_tsar(output_data, name = "2022_03_18_test", file = "txt")
```

Index