Package: tidyqpcr (via r-universe)

June 28, 2024

Type Package

Title Quantitative PCR Analysis with the Tidyverse

Version 1.0

Description For reproducible quantitative PCR (qPCR) analysis building on packages from the 'tidyverse', notably 'dplyr' and 'ggplot2'. It normalizes (by ddCq), summarizes, and plots pre-calculated Cq data, and plots raw amplification and melt curves from Roche Lightcycler (tm) machines. It does NOT (yet) calculate Cq data from amplification curves.

Depends R (>= 3.1.0), tibble

Imports rlang, dplyr, ggplot2, scales, readr, forcats, assertthat, tidyr

Suggests knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

License Apache License (>= 2)

LazyData true

RoxygenNote 7.1.2

Encoding UTF-8

URL https://docs.ropensci.org/tidyqpcr, https://github.com/ropensci/tidyqpcr

BugReports https://github.com/ropensci/tidyqpcr/issues

Language en-GB

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

RemoteUrl https://github.com/ropensci/tidyqpcr

RemoteRef main

RemoteSha 0d3f62f16ce3cc216a011ad5c64f270f34d1331d

Contents

`calculate_deltacq_bysampleid`
Calculate delta cq (ΔCq) to normalize quantification cycle (log2-fold) data within sample_id.

Description

This function implements relative quantification by the delta Cq method. For each sample, the Cq values of all targets (e.g. genes, probes, primer sets) are compared to one or more reference target ids specified in ‘ref_target_ids‘.

Usage

```
calculate_deltacq_bysampleid(cq_df, ref_target_ids, norm_function = median)
```

Arguments

`cq_df`	a data frame containing columns ‘sample_id‘, value_name (default ‘cq‘) and tid_name (default ‘target_id‘). Crucially, sample_id should be the same for different technical replicates measuring identical reactions in different wells of the plate, but differ for different biological and experimental replicates. See tidyqpcr vignettes for examples.

`ref_target_ids`	names of targets to normalize by, i.e. reference genes, hydrolysis probes, or primer sets. This can be one reference target id, a selection of multiple target ids, or even all measured target ids. In the case of all of them, the delta Cq value would be calculated relative to the median (or other 'norm_function') of all measured targets.
`norm_function`	Function to use to calculate the value to normalize by on given scale. Default is median, alternatively could use mean.

Value

data frame like cq_df with three additional columns:

ref_cq	summary (median/mean) cq value for reference target ids
delta_cq	normalized value, ΔCq
rel_abund	normalized ratio, $2^(- \Delta Cq)$

Examples

```
# create simple cq dataset with two samples, two targets  and 3 reps
cq_tibble <- tibble(sample_id = rep(c("S_1","S_1","S_1", "S_2", "S_2", "S_2"), 2),
                     target_id = rep(c("T_1",
                                       "T_norm"), each = 6),
                     tech_rep = rep(1:3, 4),
                     well_row = rep(c("A",
                                      "B"), each = 6),
                     well_col = rep(1:6, 2),
                     well = paste0(well_row, well_col),
                     cq = c(10, 10, 10, 12, 12, 11,
                             9,  9,  9,  9,  9,  9))

# calculate deltacq using reference target_id called 'T_norm'

#----- use case 1: median reference target_id value
cq_tibble %>%
    calculate_deltacq_bysampleid(ref_target_ids = "T_norm")

#----- use case 2: mean reference target_id value
cq_tibble %>%
    calculate_deltacq_bysampleid(ref_target_ids = "T_norm",
                                 norm_function = mean)
```

`calculate_deltadeltacq_bytargetid`

Calculate delta delta cq ($\Delta\Delta Cq$) to globally normalize quantification cycle (log2-fold) data across sample_id.

Description

By default, $\Delta\Delta Cq$ is positive if a target is more highly detected in the relevant sample, compared to reference samples. This can be flipped by setting the parameter ‘ddcq_positive‘ to ‘FALSE‘. In either case, The fold change, $2^{\Delta\Delta Cq}$, is also reported.

Usage

```
calculate_deltadeltacq_bytargetid(
  deltacq_df,
  ref_sample_ids,
  norm_function = median,
  ddcq_positive = TRUE
)
```

Arguments

`deltacq_df`	a data frame containing columns ‘sample_id‘, value_name (default ‘delta_cq‘) and tid_name (default ‘target_id‘). Crucially, sample_id should be the same for different technical replicates measuring identical reactions in different wells of the plate, but differ for different biological and experimental replicates. Usually this will be a data frame that was output by ‘calculate_deltacq_bysampleid‘.
`ref_sample_ids`	reference sample_ids to normalize by
`norm_function`	Function to use to calculate the value to normalize by on given scale. Default is median, alternatively could use mean.
`ddcq_positive`	(default TRUE) output $\Delta\Delta Cq$ as positive if a target is more highly detected in the relevant sample, compared to reference samples.

Details

This function does a global normalization, where all samples are compared to one or more reference samples specified in ‘ref_sample_ids‘. There are other experimental designs that require comparing samples in pairs or small groups, e.g. a time course comparing ‘delta_cq‘ values against a reference strain at each time point. For those situations, instead we recommend adapting code from this function, changing the grouping variables used in to ‘dplyr::group_by‘ to draw the contrasts appropriate for the experiment.

Value

data frame like cq_df with three additional columns:

ref_delta_cq	summary (median/mean) ΔCq value for target_id in reference sample ids
deltadelta_cq	the normalized value, $\Delta\Delta Cq$
fold_change	the normalized fold-change ratio, $2^(- \Delta\Delta Cq)$

Examples

```
# create simple deltacq dataset with two samples, two targets and 3 reps
deltacq_tibble <- tibble(sample_id = rep(c("S_1","S_1","S_1", "S_norm", "S_norm", "S_norm"), 2),
                     target_id = rep(c("T_1",
                                       "T_2"), each = 6),
                     tech_rep = rep(1:3, 4),
                     well_row = rep(c("A",
                                      "B"), each = 6),
                     well_col = rep(1:6, 2),
                     well = paste0(well_row,well_col),
                     delta_cq = c(1, 1, 1, 3, 3, 2,
                                  4, 5, 4, 5, 5, 5))

# calculate deltadeltacq using reference target_id called 'S_norm'

#----- use case 1: median reference sample_id value
deltacq_tibble %>%
    calculate_deltadeltacq_bytargetid(ref_sample_ids = "S_norm")

#----- use case 2: mean reference sample_id value
deltacq_tibble %>%
    calculate_deltadeltacq_bytargetid(ref_sample_ids = "S_norm",
                                       norm_function = mean)
```

`calculate_drdt_plate` *Calculate dR/dT of melt curves for of every well in a plate.*

Description

dR/dT, the derivative of the melt curve (of fluorescence signal R vs temperature T), has a maximum at the melting temperature Tm. A single peak in this suggests a single-length PCR product is present in the well.

Usage

```
calculate_drdt_plate(platemelt, method = "spline", ...)
```

Arguments

`platemelt`	data frame describing melt curves, including variables well, temperature, fluor_raw (raw fluorescence value).
`method`	to use for smoothing: "spline" default, uses smoothing spline stats::smooth.spline. "diff" base::diff for lagged difference
`...`	other arguments to pass to smoothing method.

Details

Note that this function does not group by plate, only by well. The function will give strange results if you pass it data from more than one plate. Avoid this by analysing one plate at a time.

Value

platemelt with additional column dRdT.

See Also

Other melt_curve_functions: `calculate_dydx()`

Examples

```
# create simple curve
# create simple dataset of raw fluorescence with two samples
temp_tibble <- tibble(sample_id = rep(c("S1", "S2"), each = 10),
                          target_id = "T1",
                          well_row = "A",
                          well_col = rep(c(1, 2), each = 10),
                          well = rep(c("A1", "A2"), each = 10),
                          temperature = rep(56:65,2),
                          fluor_raw = c(1:10, 6:15))

# calculate drdt of all melt curves
#----- use case 1 : using splines
temp_tibble %>%
    calculate_drdt_plate()

# optional arguments are passed to smooth.splines function
temp_tibble %>%
    calculate_drdt_plate(spar = 0.5)

#----- use case 2 : using difference between adjacent points
temp_tibble %>%
    calculate_drdt_plate(method = "diff")
```

`calculate_dydx` *Calculate dy/dx vector from vectors y and x*

Description

Used in tidyqpcr to calculate dR/dT for a melt curve of fluorescence signal R vs temperature T.

Usage

```
calculate_dydx(x, y, method = "spline", ...)
```

Arguments

`x`	input variable, numeric vector, assumed to be temperature
`y`	output variable, numeric vector of same length as x, assumed to be fluorescence signal.
`method`	to use for smoothing: "spline" default, uses smoothing spline stats::smooth.spline. "diff" base::diff for lagged difference
`...`	other arguments to pass to smoothing method.

Value

estimated first derivative of y with respect to x, numeric vector of same length as y.

See Also

Other melt_curve_functions: `calculate_drdt_plate()`

Examples

```
# create simple curve
x = 1:5
y = x^2

# calculate gradient of curve
#----- use case 1 : using splines
calculate_dydx(x, y)

# optional arguments are passed to smooth.splines function
calculate_dydx(x, y, spar = 0.5)

#----- use case 2 : using difference between adjacent points
calculate_dydx(x, y, method = "diff")
```

`calculate_efficiency` *Calibrate primer sets / probes by calculating detection efficiency and R squared*

Description

Note efficiency is given in ratio, not per cent; multiply by 100 for that.

Usage

```
calculate_efficiency(cq_df_1, formula = cq ~ log2(dilution) + biol_rep)
```

Arguments

`cq_df_1`	data frame with cq (quantification cycle) data, 1 row per well. Must have columns cq, dilution. Assumes data are only for 1 probe/primer set/target_id, i.e. all values in cq_df_1 are fit with the same slope.
`formula`	formula to use for log-log regression fit. Default value assumes multiple biological replicates, cq ~ log2(dilution) + biol_rep. If only a single Biological Replicate, change to cq ~ log2(dilution).

Value

data frame with 1 single row, and columns: efficiency, efficiency.sd, r.squared.

See Also

calculate_efficiency_bytargetid

Examples

```
# create simple dilution dataset
dilution_tibble <- tibble(dilution = rep(c(1, 0.1, 0.001, 0.0001), 2),
                     cq = c(1, 3, 4, 6,
                            4, 5, 6, 7),
                     biol_rep = rep(c(1,2), each = 4),
                     target_id = "T1")

# calculate primer efficiency

#----- use case 1: include difference across replicates in model
dilution_tibble %>%
    calculate_efficiency()

#----- use case 2: ignore difference across replicates
dilution_tibble %>%
    calculate_efficiency(formula = cq ~ log2(dilution))
```

`calculate_efficiency_bytargetid` *Calibrate multiple probes by calculating detection efficiency and R squared*

Description

See calibration vignette for example of usage.

Usage

```
calculate_efficiency_bytargetid(
  cq_df,
  formula = cq ~ log2(dilution) + biol_rep,
  use_prep_types = "+RT"
)
```

Arguments

`cq_df`	a data frame with cq (quantification cycle) data, 1 row per well Must have columns prep_type, target_id, cq, dilution. Only prep_type=="+RT" columns are used.
`formula`	formula to use for log-log regression fit. Default value assumes multiple biological replicates, cq ~ log2(dilution) + biol_rep. If only a single Biological Replicate, change to cq ~ log2(dilution). If multiple sample_ids, change to cq ~ log2(dilution) + sample_id. See ?formula for background and help.
`use_prep_types`	prep_type column values to use, default "+RT" for RT-qPCR. By default, this includes only reverse-transcribed values in the efficiency estimation, so excludes negative controls such as no-template and no-RT. To skip this filtering step, set use_prep_types=NA.

Details

Note efficiency is given in ratio, not per cent; multiply by 100 for that.

Value

data frame with columns: target_id, efficiency, efficiency.sd, r.squared.

See Also

calculate_efficiency

Examples

```
# create simple dilution dataset for two target_ids with two biological reps
dilution_tibble <- tibble(target_id = rep(c("T_1",
                                            "T_2"), each = 8),
                          well_row = rep(c("A",
                                           "B"), each = 8),
                          well_col = rep(1:8, 2),
                          well = paste0(well_row, well_col),
                          dilution = rep(c(1, 0.1, 0.001, 0.0001), 4),
                          cq = c(1, 3, 4, 6, 1, 3, 5, 7,
                                 4, 5, 6, 7, 3, 7, 8, 9),
                          biol_rep = rep(c(1, 1, 1, 1, 2, 2, 2, 2), 2),
                          prep_type = "+RT")
```

```
# calculate primer efficiency for multiple targets

#----- use case 1: include difference across replicates in model
dilution_tibble %>%
    calculate_efficiency_bytargetid()

#----- use case 2: ignore difference across replicates
dilution_tibble %>%
    calculate_efficiency_bytargetid(formula = cq ~ log2(dilution))
```

calculate_normvalue *Calculate a normalized value for a subset of reference ids*

Description

This is used to calculate the normalized ‘cq‘ values for reference ‘target_ids‘ (e.g. genes), to use in ‘delta_cq‘ calculation for each ‘sample_id‘.

Usage

```
calculate_normvalue(
  value_df,
  ref_ids,
  value_name = "value",
  id_name = "id",
  norm_function = median
)
```

Arguments

value_df	data frame containing relevant columns, those named in ‘value_name‘ and ‘id_name‘ parameters.
ref_ids	values of reference ids, that are used to calculate normalized reference value.
value_name	name of column containing values. This column should be numeric.
id_name	name of column containing ids.
norm_function	Function to use to calculate the value to normalize by. Default function is median, alternatively could use mean, geometric mean, etc.

Details

Also used to calculate the normalized ‘delta_cq‘ values for reference ‘sample_ids‘, to use in ‘deltadelta_cq‘ calculation for each ‘target_id‘.

Examples

```
# create simple cq dataset with one sample, two targets  and 3 reps
cq_tibble <- tibble(sample_id = "S_1",
                     target_id = rep(c("T_1",
                                       "T_norm"), each = 3),
                     tech_rep = rep(1:3, 2),
                     well_row = rep(c("A",
                                      "B"), each = 3),
                     well_col = 1,
                     well = paste0(well_row, well_col),
                     cq = c(10, 10, 10,
                            12, 12, 11))

# normalise cq to reference target_id called 'T_norm'

#----- use case 1: median reference target_id value
cq_tibble %>%
    calculate_normvalue(ref_ids = "T_norm",
                        value_name = "cq",
                        id_name = "target_id")

#----- use case 2: mean reference target_id value
cq_tibble %>%
    calculate_normvalue(ref_ids = "T_norm",
                        value_name = "cq",
                        id_name = "target_id",
                        norm_function = mean)
```

`create_blank_plate` *Create a blank plate template as a tibble (with helper functions for common plate sizes)*

Description

For more help, examples and explanations, see the plate setup vignette: `vignette("platesetup_vignette", package = "tidyqpcr")`

Usage

```
create_blank_plate(well_row = LETTERS[1:16], well_col = 1:24)

create_blank_plate_96well()

create_blank_plate_1536well(
  well_row = make_row_names_lc1536(),
  well_col = 1:48
)
```

Arguments

`well_row`	Vector of Row labels, usually LETTERS
`well_col`	Vector of Column labels, usually numbers

Value

tibble (data frame) with columns well_row, well_col, well. This contains all pairwise combinations of well_row and well_col, as well as individual well names. Both well_row and well_col are coerced to factors (even if well_col is supplied as numbers), to ensure order is consistent.

However, well is a character vector as that is the default behaviour of "unite", and display order doesn't matter.

Default value describes a full 384-well plate.

Functions

- `create_blank_plate_96well`: create blank 96-well plate
- `create_blank_plate_1536well`: create blank 1536-well plate

See Also

Other plate creation functions: `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_blank_plate(well_row=LETTERS[1:2],well_col=1:3)

create_blank_plate_96well()

create_blank_plate_1536well()

# create blank 96-well plate with empty edge wells

create_blank_plate(well_row=LETTERS[2:7], well_col=2:11)

# create blank 1536-well plate with empty edge wells

full_plate_row_names <- make_row_names_lc1536()

create_blank_plate(well_row=full_plate_row_names[2:31], well_col=2:47)
```

`create_colkey_4diln_2ctrl_in_24` *Create a 4-dilution column key for primer calibration*

Description

Creates a 24-column key for primer calibration, with 2x biol_reps and 2x tech_reps, and 5-fold dilution until 5^4 of +RT; then -RT (no reverse transcriptase), NT (no template) negative controls. That is a total of 6 versions of each sample replicate.

Usage

```
create_colkey_4diln_2ctrl_in_24(
  dilution = c(5^(0:-3), 1, 1),
  dilution_nice = c("1x", "5x", "25x", "125x", "-RT", "NT"),
  prep_type = c(rep("+RT", 4), "-RT", "NT"),
  biol_rep = rep(c("A", "B"), each = 12, length.out = 24),
  tech_rep = rep(1:2, each = 6, length.out = 24)
)
```

Arguments

`dilution`	Numeric vector of length 6 describing sample dilutions
`dilution_nice`	Character vector of length 6 with nice labels for sample dilutions
`prep_type`	Character vector of length 6 describing type of sample (+RT, -RT, NT)
`biol_rep`	Character vector of length 6 describing biological replicates
`tech_rep`	Character vector of length 6 describing technical replicates

Value

tibble (data frame) with 24 rows, and columns well_col, dilution, dilution_nice, prep_type, biol_rep, tech_rep.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_2`
`create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_colkey_4diln_2ctrl_in_24()
```

create_colkey_6diln_2ctrl_in_24

Create a 6-dilution column key for primer calibration

Description

Creates a 24-column key for primer calibration, with 1x biol_reps and 3x tech_reps, and 5-fold dilution until 5^6 of +RT; then -RT (no reverse transcriptase), NT (no template) negative controls. That is a total of 8 versions of each replicate.

Usage

```
create_colkey_6diln_2ctrl_in_24(
  dilution = c(5^(0:-5), 1, 1),
  dilution_nice = c("1x", "5x", "25x", "125x", "625x", "3125x", "-RT", "NT"),
  prep_type = c(rep("+RT", 6), "-RT", "NT"),
  tech_rep = rep(1:3, each = 8, length.out = 24)
)
```

Arguments

`dilution`	Numeric vector of length 8 describing sample dilutions
`dilution_nice`	Character vector of length 8 with nice labels for sample dilutions
`prep_type`	Character vector of length 8 describing type of sample (+RT, -RT, NT)
`tech_rep`	Character vector of length 8 describing technical replicates

Value

tibble (data frame) with 24 rows, and variables well_col, dilution, dilution_nice, prep_type, biol_rep, tech_rep.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_colkey_6diln_2ctrl_in_24()
```

`create_colkey_6_in_24` *Create a 6-value, 24-column key for plates*

Description

Create a 24-column key with 6 values repeated over 24 plate columns. Each of the 6 values is repeated over 3x +RT Techreps and 1x -RT.

Usage

```
create_colkey_6_in_24(...)
```

Arguments

`...`	Vectors of length 6 describing well contents, e.g. sample_id or target_id

Details

This helps to create plate layouts with standard designs.

Value

tibble (data frame) with 24 rows, and columns well_col, prep_type, tech_rep, and supplied values.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_colkey_6_in_24(sample_id=LETTERS[1:6])
```

`create_rowkey_4_in_16` *Create a 4-value, 16-row key for plates*

Description

Create a 16-row key with 4 values repeated over 16 plate rows. Each of the 4 values is repeated over 3x +RT Techreps and 1x -RT.

Usage

```
create_rowkey_4_in_16(...)
```

Arguments

`...`	Vectors of length 4 describing well contents, e.g. sample_id or target_id

Details

This helps to create plate layouts with standard designs.

Value

tibble (data frame) with 16 rows, and variables well_row, prep_type, tech_rep, and supplied values.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_rowkey_4_in_16(sample_id=c("sheep","goat","cow","chicken"))
```

`create_rowkey_8_in_16_plain`

Create a plain 8-value, 16-row key for plates

Description

Create a 16-row key with 8 values repeated over 16 plate rows. No other information is included by default, hence "plain".

Usage

```
create_rowkey_8_in_16_plain(...)
```

Arguments

`...`	Vectors of length 8 describing well contents, e.g. sample or probe.

Details

This helps to create plate layouts with standard designs.

Value

tibble (data frame) with 16 rows, and variables well_col, and supplied values.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
create_rowkey_8_in_16_plain(sample_id=c("me","you","them","him",
                                        "her","dog","cat","monkey"))
```

`debaseline` *Remove baseline from amplification curves (BETA)*

Description

Remove baseline from qPCR amplification curves by subtracting median of initial cycles.

Usage

```
debaseline(plateamp, maxcycle = 10)
```

Arguments

`plateamp`	data frame with plate amplification data, including variables well, cycle, fluor_raw (raw fluorescence value), and program_no. Assume program 2 for amplification curves from Roche Lightcycler format data.
`maxcycle`	maximum cycle value to use for baseline, before amplification.

Details

BETA function version because:

- assumes Roche Lightcycler format, we should ideally replace "program_no == 2" by something more generic?

- the rule-of thumb "baseline is median of initial 10 cycles" has not been tested robustly

Value

platemap with additional columns per well:

fluor_base	baseline /background value
fluor_signal	normalized fluorescence signal, i.e. fluor_raw - fluor_base

Examples

```
# create simple dataset of raw fluorescence
# with two samples over 15 cycles
raw_fluor_tibble <- tibble(sample_id = rep(c("S1", "S2"), each = 15),
                          target_id = "T1",
                          well_row = "A",
                          well_col = rep(c(1, 2), each = 15),
                          well = rep(c("A1", "A2"), each = 15),
                          cycle = rep(1:15,2),
                          fluor_raw = c(1:15, 6:20),
                          program_no = 2)

# remove base fluorescence from dataset
raw_fluor_tibble %>%
    debaseline()
```

`display_plate` *Display an empty plate plan which can be populated with ggplot2 geom elements.*

Description

Display an empty plate plan which can be populated with ggplot2 geom elements.

Usage

```
display_plate(plate)
```

Arguments

`plate`	tibble with variables well_col, well_row.

Value

ggplot object; major output is to plot it

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
library(ggplot2)

# display empty plot of empty plate
display_plate(create_blank_plate_96well())

# display wells of empty plate filled by column
display_plate(create_blank_plate_96well()) +
  geom_tile(aes(fill = well_col), colour = "black")

# display wells of empty 1536-well plate filled by row
display_plate(create_blank_plate_1536well()) +
  geom_tile(aes(fill = well_row), colour = "black")
```

`display_plate_qpcr` *Display qPCR plate plan with sample_id, target_id, prep_type per well*

Description

Display qPCR plate plan with sample_id, target_id, prep_type per well

Usage

```
display_plate_qpcr(plate)
```

Arguments

`plate`	tibble with variables well_col, well_row, sample_id, target_id, prep_type. Output from label_plate_rowcol.

Value

ggplot object; major output is to plot it

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
# create basic 6-well plate
basic_plate <-
    label_plate_rowcol(plate = create_blank_plate(well_row = LETTERS[1:2],
                                                   well_col = 1:3),
                       rowkey = tibble(well_row = factor(LETTERS[1:2]),
                                       target_id = c("T_A","T_B")),
                       colkey = tibble(well_col = factor(1:3),
                                       sample_id = c("S_1","S_2", "S_3"),
                                       prep_type = "+RT"))

# display basic plate
display_plate_qpcr(basic_plate)

# create full 384 well plate
full_plate <- label_plate_rowcol(create_blank_plate(),
                                 create_rowkey_8_in_16_plain(target_id = c("T_1", "T_2",
                                                                           "T_3", "T_4",
                                                                           "T_5", "T_6",
                                                                         "T_7", "T_8")),
                                  create_colkey_6diln_2ctrl_in_24() %>%
                                      dplyr::mutate(sample_id = paste0(dilution_nice,
                                                                       "_",
                                                                       tech_rep)))

# display full plate
display_plate_qpcr(full_plate)
```

`display_plate_value` *Display the value of each well across the plate.*

Description

Plots the plate with each well coloured by its value. Example values are Cq, Delta Cq or Delta Delta Cq.

Usage

```
display_plate_value(plate, value = "cq")
```

Arguments

`plate`	tibble with variables well_col, well_row, and the variable to be plotted.
`value`	character vector selecting the variable in plate to plot as the well value

Details

For a specific example see the calibration vignette: `vignette("calibration_vignette", package = "tidyqpcr")`

Value

ggplot object; major output is to plot it

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
library(dplyr)
library(ggplot2)

# create 96 well plate with random values
plate_randomcq <- create_blank_plate_96well() %>%
    mutate(cq = runif(96) * 10,
           deltacq = runif(96) * 2)


# display well Cq value across plate
display_plate_value(plate_randomcq)

# display well Delta Cq value across plate with red colour pallette
display_plate_value(plate_randomcq, value = "deltacq") +   # uses ggplot syntax
    scale_fill_gradient(high = "#FF0000")
```

`label_plate_rowcol` *Label a plate with sample and probe information*

Description

For more help, examples and explanations, see the plate setup vignette: `vignette("platesetup_vignette", package = "tidyqpcr")`

Usage

```
label_plate_rowcol(plate, rowkey = NULL, colkey = NULL, coercefactors = TRUE)
```

Arguments

`plate`	tibble (data frame) with variables well_row, well_col, well. This would usually be produced by create_blank_plate(). It is possible to include other information in additional variables.
`rowkey`	tibble (data frame) describing plate rows, with variables well_row and others.
`colkey`	tibble (data frame) describing plate columns, with variables well_col and others.
`coercefactors`	if TRUE, coerce well_row in rowkey and well_col in colkey to factors

Details

For worked examples of tidyqpcr analysis with 384-well plates, see: `vignette("calibration_vignette", package = "tidyqpcr")`

Value

tibble (data frame) with variables well_row, well_col, well, and others.

This tibble contains all combinations of well_row and well_col found in the input plate, and all information supplied in rowkey and colkey distributed across every well of the plate. Return plate is ordered by row well_row then column well_col.

Note this ordering may cause a problem if well_col is supplied as a character (1,10,11,...), instead of a factor or integer (1,2,3,...). For this reason, the function by default converts well_row in ‘rowkey‘, and well_col in ‘colkey‘, to factors, taking factor levels from ‘plate‘, and messages the user.

If ‘plate$well_col‘ or ‘plate$well_row‘ are not factors and coercefactors = TRUE label_plate_rowcol will automatically convert them to factors, but will output a warning telling users this may lead to unexpected behaviour.

Other tidyqpcr functions require plate plans to contain variables sample_id, target_id, and prep_type, so ‘label_plate_rowcol‘ will message if any of these are missing. This is a message, not an error, because these variables can be added by users later.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `make_row_names_echo1536()`, `make_row_names_lc1536()`

Examples

```
label_plate_rowcol(plate = create_blank_plate()) # returns blank plate

# label blank 96-well plate with empty edge wells

label_plate_rowcol(plate = create_blank_plate(well_row = LETTERS[2:7],
                                              well_col = 2:11))

# label 96-well plate with sample id in rows
```

```
label_plate_rowcol(plate = create_blank_plate(well_row = LETTERS[1:8],
                                              well_col = 1:12),
                   rowkey = tibble(well_row = LETTERS[1:8],
                                   sample_id = paste0("S_",1:8)))

# label fraction of 96-well plate with target id in columns

label_plate_rowcol(plate = create_blank_plate(well_row = LETTERS[1:8],
                                              well_col = 1:4),
                   colkey = tibble(well_col = 1:4,
                                   target_id = paste0("T_",1:4)))
```

---

make_row_names_echo1536 *Generates row names for the Labcyte Echo 1536-well plates*

Description

Creates a vector containing 36 row names according to the labelling system used by the Labcyte Echo

Usage

```
make_row_names_echo1536()
```

Value

Vector of row names: A,B,...,Z,AA,AB,...,AF.

See Also

Other plate creation functions: create_blank_plate(), create_colkey_4diln_2ctrl_in_24(), create_colkey_6_in_24(), create_colkey_6diln_2ctrl_in_24(), create_rowkey_4_in_16(), create_rowkey_8_in_16_plain(), display_plate_qpcr(), display_plate_value(), display_plate(), label_plate_rowcol(), make_row_names_lc1536()

Examples

```
make_row_names_echo1536()
```

`make_row_names_lc1536` *Generates row names for the Roche Lightcycler (tm) 1536-well plates*

Description

Creates a vector containing 36 row names according to the labelling system used by the Roche Lightcycler (tm)

Usage

```
make_row_names_lc1536()
```

Value

Vector of row names: Aa,Ab,Ac,Ad,Ba,...,Hd.

See Also

Other plate creation functions: `create_blank_plate()`, `create_colkey_4diln_2ctrl_in_24()`, `create_colkey_6_in_24()`, `create_colkey_6diln_2ctrl_in_24()`, `create_rowkey_4_in_16()`, `create_rowkey_8_in_16_plain()`, `display_plate_qpcr()`, `display_plate_value()`, `display_plate()`, `label_plate_rowcol()`, `make_row_names_echo1536()`

Examples

```
make_row_names_lc1536()
```

`read_lightcycler_1colour_cq` *Reads quantification cycle (cq) data in 1 colour from Roche Lightcyclers*

Description

This is the data from "export in text format" from the analysis tab in the Lightcycler software. That software calls cq, "Cp".

Usage

```
read_lightcycler_1colour_cq(
  filename,
  skip = 2,
  col_names = c("include", "color", "well", "sample_info", "cq", "concentration",
    "standard", "status"),
  col_types = "liccddil",
  ...
)
```

Arguments

`filename`	file name
`skip`	number of lines to skip, defaults to 2
`col_names`	names to give to columns
`col_types`	data types of columns
`...`	other arguments to pass to read_tsv, if needed

Details

This function is a thin wrapper around readr::read_tsv.

Value

tibble containing cq data

See Also

read_lightcycler_1colour_raw

Examples

```
read_lightcycler_1colour_cq(system.file("extdata/Edward_qPCR_Nrd1_calibration_2019-02-02_Cq.txt.gz",
                                        package = "tidyqpcr"))
```

`read_lightcycler_1colour_raw`
Reads raw text-format fluorescence data in 1 colour from Roche Lightcyclers

---

Description

This is the data from "export in text format" from the Lightcycler software. The other data format, .ixo, can be converted to .txt format by the Lightcycler software.

Usage

```
read_lightcycler_1colour_raw(
  filename,
  skip = 2,
 col_names = c("well", "sample_info", "program_no", "segment_no", "cycle", "time",
    "temperature", "fluor_raw"),
  col_types = "ccffinnn",
  ...
)
```

Arguments

`filename`	file name
`skip`	number of lines to skip, defaults to 2
`col_names`	names to give to columns
`col_types`	data types of columns
`...`	other arguments to pass to read_tsv, if needed

Details

This function is a thin wrapper around readr::read_tsv.

Value

tibble containing raw data, with default column names:

well: the well of the plate, e.g. A1

sample_info: this is the "Sample" field entered in lightcycler software, defaults to "Sample X"

program_no: the number of the cycler program, for 2-step PCR defaults to 1 = melt, 2 = amplify, 3 = melt.

segment_no: the number of the segment of the cycler program, e.g. hold/raise/lower temperature

cycle: the cycle number, for programs with repeated cycles (i.e. amplification)

time: the time of fluorescence reading acquisition (in what units???)

temperature: the temperature of the block at fluorescence acquisition

fluor_raw: the raw fluorescence reading in "arbitrary units". For SYBR safe, this would be 483nm excitation, 533nm emission.

See Also

read_lightcycler_1colour_cq

Examples

```
read_lightcycler_1colour_raw(system.file("extdata/Edward_qPCR_Nrd1_calibration_2019-02-02.txt.gz",
                                         package = "tidyqpcr"))
```

Index