Package 'OlinkAnalyze'

May 7, 2026

Type Package

Title Facilitate Analysis of Proteomic Data from Olink

Version 5.0.0

Description A collection of functions to facilitate analysis of proteomic data from Olink, primarily NPX data that has been exported from Olink Software. The functions also work on QUANT data from Olink by log- transforming the QUANT data. The functions are focused on reading data, facilitating data wrangling and quality control analysis, performing statistical analysis and generating figures to visualize the results of the statistical analysis. The goal of this package is to help users extract biological insights from proteomic data run on the Olink platform.

License AGPL (>= 3)

Contact biostattools@olink.com

URL https://olink.com/

https://github.com/Olink-Proteomics/OlinkRPackage

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first read_npx_l*, read_npx_w*

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports arrow (>= 14.0.0), cli (>= 3.6.2), data.table, dbplyr, dplyr (>= 1.2.0), duckdb, forcats, ggplot2, grDevices, rlang, stringr, tibble, tidyr

Suggests broom, car, clusterProfiler, curl, emmeans, ggplotify, ggpubr, ggrepel, lme4, lmerTest, msigdbr (> 24.1.0), ordinal, readxl, pheatmap, scales, showtext, sysfonts, systemfonts, testthat (>= 3.0.0), vdiffr, withr, writexl, zip, umap, rstatix, FSA, kableExtra, knitr

VignetteBuilder knitr

NeedsCompilation no

Author Kathleen Nevola [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5183-6444>, kathy-nevola),
Marianne Sandin [aut] (ORCID: <https://orcid.org/0000-0001-6186-963X>,
marisand),
Jamey Guess [aut] (ORCID: <https://orcid.org/0000-0002-4017-0923>,
jrguess),
Simon Forsberg [aut] (ORCID: <https://orcid.org/0000-0002-7451-9222>,
simfor),
Christoffer Cambronero [aut] (Orbmac),
Pascal Pucholt [aut] (ORCID: <https://orcid.org/0000-0003-3342-1373>,
AskPascal),
Boxi Zhang [aut] (ORCID: <https://orcid.org/0000-0001-7758-6204>,
boxizhang),
Masoumeh Sheikhi [aut] (MasoumehSheikhi),
Klev Diamanti [aut] (ORCID: <https://orcid.org/0000-0002-4922-8415>,
klevdiamanti),
Amrita Kar [aut] (amrita-kar),
Lei Conze [aut] (leiliuC),
Kristyn Chin [aut] (kristynchin-olink),
Danai Topouza [aut] (dtopouza, ORCID:
<https://orcid.org/0000-0002-6897-9281>),
Stephen Pollo [aut] (spollo-olprot, ORCID:
<https://orcid.org/0000-0001-9252-4976>),
Kang Dong [aut] (KangD-dev, ORCID:
<https://orcid.org/0000-0002-4567-5007>),
Kristian Hodén [ctb] (ORCID: <https://orcid.org/0000-0003-0354-0662>,
kristianHoden),
Per Eriksson [ctb] (ORCID: <https://orcid.org/0000-0001-7633-403X>,
b_watcher),
Nicola Moloney [ctb] (ORCID: <https://orcid.org/0000-0003-4967-3284>),
Britta Lötstedt [ctb] (ORCID: <https://orcid.org/0000-0003-3545-5489>),
Emmett Sprecher [ctb] (ORCID: <https://orcid.org/0000-0002-7710-695X>),
Jessica Barbagallo [ctb] (jbarbagallo),
Olof Mansson [ctr] (olofmansson),
Ola Caster [ctb] (OlaCaster),
Olink [cph, fnd]

Maintainer Kathleen Nevola <biostattools@olink.com>

Repository CRAN

Date/Publication 2026-03-28 12:00:02 UTC

Contents

`assign_subject2plate` *assign subject to a plate for longitudinal randomization*

Description

assign subject to a plate for longitudinal randomization

Usage

```
assign_subject2plate(plate_map, manifest, subject_id)
```

Arguments

`plate_map`	character vector of locations available for samples including plate, row and columns
`manifest`	sample manifest mapping sample IDs and subject IDs
`subject_id`	id of subject

Value

plate_map adding sample IDs to plates keeping samples from the same subject on the same plate

---

check_library_installed *Help function to check if suggested libraries are installed when required.*

Description

Help function to check if suggested libraries are installed when required.

Usage

```
check_library_installed(x, error = FALSE)
```

Arguments

x	A character vector of R libraries.
error	Boolean to return error or a boolean (default).

Value

Boolean if the library is installed or not, and an error if `error = TRUE`.

Author(s)

Klev Diamanti

---

check_npx *Check NPX data format*

Description

This function performs various checks on NPX data, including checking column names, validating Olink identifiers, identifying assays with *NA* values for all samples and detecting duplicate sample identifiers.

Usage

```
check_npx(df, preferred_names = NULL)
```

Arguments

df	A "tibble" or "ArrowObject" from read_npx.
preferred_names	A named character vector where names are internal column names and values are column names to be selected from the input data frame. Read the *description* for further information.

Details

OlinkAnalyze uses pre-defined names of columns of data frames to perform downstream analyses. At the same time, different Olink platforms export data with different column names (e.g. different protein quantification metric). This function aims to instruct each function of OlinkAnalyze on the column it should be using for the downstream analysis. This should be seamless for data exported from Olink Software and imported to R using the read_npx function.

However, in certain cases the columns of interest might be named differently. This function allows assigning custom-named columns of a data frame to internally expected variables that will in turn instruct Olink Analyze functions to use them for downstream analysis. For example, if one wished to use the column *PCNormalizedNPX* for their analysis instead of the column *NPX*, then they can assign this new name to the internal variable *quant* to inform the package that in the downstream analysis *PCNormalizedNPX* should be used. See example 3.

Similarly, in case of multiple matches (e.g. the data frame contains both columns *LOD* and *PlateLOD*) the ties will need to be resolved by the user using the argument *preferred_names* from this function. See example 4.

The argument *preferred_names* is a named character vector with internal column names as names and column names of the current data set as values. Names of the input vector can be one or more of the following: "sample_id", "sample_type", "assay_type", "olink_id", "uniprot", "assay", "panel", "block", "plate_id", "panel_version", "lod", "quant", "ext_npx", "count", "qc_warning", "assay_warn", "normalization", and "qc_version"

Value

A list containing the following elements:

- **col_names** List of column names from the input data frame marking the columns to be used in downstream analyses.
- **oid_invalid** Character vector of invalid *OlinkID*.
- **assay_na** Character vector of assays with all samples having *NA* values.
- **sample_id_dups** Character vector of duplicate *SampleID*.
- **sample_id_na** Character vector containing *SampleID* of samples with quantified values *NA* for all assays.
- **col_class** Data frame with columns of incorrect type including column key *col_key*, column name *col_name*, detected column type *col_class* and expected column type *expected_col_class*.
- **assay_qc** Character vector containing *OlinkID* of assays with at least one assay warning.
- **non_unique_uniprot** Character vector of *OlinkID* mapped to more than one *UniProt* ID.
- **darid_invalid** Character vector containing outdated combinations of *DataAnalysisRefID* and *PanelDataArchiveVersion*.

Author(s)

Masoumeh Sheikhi Klev Diamanti

Examples

```
## Not run:
# Example 0: Use npx_data1 to check that check_npx works
check_npx_result <- OlinkAnalyze::npx_data1 |>
  OlinkAnalyze::check_npx() |>
  suppressWarnings()

# read NPX data
npx_file <- system.file("extdata",
                        "npx_data_ext.parquet",
                        package = "OlinkAnalyze")
npx_df <- OlinkAnalyze::read_npx(filename = npx_file)

# Example 1: run df as is
OlinkAnalyze::check_npx(df = npx_df)

# Example 2: SampleType missing from data frame
npx_df |>
  dplyr::select(
    -dplyr::all_of(
      c("SampleType")
    )
  ) |>
  OlinkAnalyze::check_npx()

# Example 3: Use PCNormalizedNPX instead on NPX
OlinkAnalyze::check_npx(
  df = npx_df,
  preferred_names = c("quant" = "PCNormalizedNPX")
)

# Example 4: Use PCNormalizedNPX instead on NPX, and PlateLOD instead of LOD
npx_df |>
  dplyr::mutate(
    LOD = 1L,
    PlateLOD = 2L
  ) |>
  OlinkAnalyze::check_npx(
    preferred_names = c("quant" = "PCNormalizedNPX",
                        "lod" = "PlateLOD")
  )

## End(Not run)
```

clean_npx *Clean proteomics data quantified with Olink's PEA technology*

Description

This function applies a series of cleaning steps to a data set exported by Olink Software and imported in R by `read_npx()`. Some of the steps of this function rely on results from `check_npx()`.

This function removes samples and assays that are not suitable for downstream statistical analysis. Some of the data records that are removed include duplicate sample identifiers, external controls samples, internal control assays, and samples or assays with quality control flags.

Usage

```
clean_npx(
  df,
  check_log = NULL,
  remove_assay_na = TRUE,
  remove_invalid_oid = TRUE,
  remove_dup_sample_id = TRUE,
  remove_control_assay = TRUE,
  remove_control_sample = TRUE,
  remove_qc_warning = TRUE,
  remove_assay_warning = TRUE,
  control_sample_ids = NULL,
  convert_df_cols = TRUE,
  convert_nonunique_uniprot = TRUE,
  out_df = "tibble",
  verbose = FALSE
)
```

Arguments

df	A "tibble" or "ArrowObject" from `read_npx`.
check_log	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
remove_assay_na	Logical. If `FALSE`, skips filtering assays with all quantified values `NA`. Defaults to `TRUE`.
remove_invalid_oid	Logical. If `FALSE`, skips filtering assays with invalid identifiers. Defaults to `TRUE`.
remove_dup_sample_id	Logical. If `FALSE`, skips filtering samples with duplicate sample identifiers. Defaults to `TRUE`.

`remove_control_assay`
: If `FALSE`, all internal control assays are retained. If `TRUE`, all internal control assays are removed. Alternatively, a character vector with one or more of "assay", "inc", "det", "ext", and "amp" indicating the assay types to remove.

`remove_control_sample`
: If `FALSE`, all control samples are retained. If `TRUE`, all control samples are removed. Alternatively, a character vector with one or more of "sample", "sc", "pc", and "nc" indicating the sample types to remove.

`remove_qc_warning`
: Logical. If `FALSE`, retains samples flagged as `FAIL` in QC warning. Defaults to `TRUE`.

`remove_assay_warning`
: Logical. If `FALSE`, retains assays flagged as `WARN` in assay warning. Defaults to `TRUE`.

`control_sample_ids`
: character vector of sample identifiers of control samples. Default `NULL`, to mark no samples to be removed.

`convert_df_cols`
: Logical. If `FALSE`, retains columns of `df` as are. Defaults to `TRUE`, were columns required for downstream analysis are converted to the expected format.

`convert_nonunique_uniprot`
: Logical. If `FALSE`, retains non-unique OlinkID - UniProt mapping. Defaults to `TRUE`.

`out_df`
: The class of the output dataset. One of "tibble" or "arrow". Defaults to "tibble".

`verbose`
: Logical. If `FALSE` (default), silences step-wise messages.

Details

The pipeline performs the following steps:

1. **Remove assays with invalid identifiers**: assays flagged as having invalid identifiers from `check_npx()`. Occurs when the original data set provided by Olink Software has been modified.
2. **Remove assays with `NA` quantification values**: assays lacking quantification data are reported with `NA` as quantification. These assays are identified in `check_npx()`.
3. **Remove samples with duplicate identifiers**: samples with identical identifiers detected by `check_npx()`. Instances of duplicate sample identifiers cause errors in the downstream analysis of data with, and it is highly discouraged.
4. **Remove external control samples**:
   - Uses column marking sample type (e.g. `SampleType`) to exclude external control samples.
   - Uses column marking sample identifier (e.g. `SampleID`) to remove external control samples, or samples that ones wants to exclude from the downstream analysis.
5. **Remove samples failing quality control**: samples with QC status `FAIL`.
6. **Remove internal control assays**: Uses column marking assay type (e.g. `AssayType`) to exclude internal control assays.

7. **Remove assays with quality controls warnings**: assays with QC status `WARN`.
8. **Correct column data type**: ensure that certain columns have the expected data type (class). These columns are identified in `check_npx()`.
9. **Resolve multiple UniProt mappings per assay**: ensure that each assay identifier (e.g., `OlinkID`) maps uniquely to a single UniProt ID.

Important:

- When data set lacks a column marking sample type (e.g. `SampleType`), one should remove external control samples based on their sample identifiers. This function does not auto-detect external control samples based on their sample identifiers. *Please ensure external control samples have been removed prior to downstream statistical analysis.*
- When data set lacks a column marking assay type (e.g. `AssayType`), one should remove internal control assays manually. This function does not auto-detect internal control assays. *Please ensure internal control assays have been removed prior to downstream statistical analysis.*

Value

Dataset, "tibble" or "ArrowObject", with Olink data in long format.

Author(s)

Kang Dong Klev Diamanti

Examples

```
## Not run:
# run check_npx
check_log <- check_npx(
  df = npx_data1
)

# run clean_npx
clean_npx(
  df = npx_data1,
  check_log = check_log
)

# run clean_npx with messages for all steps
clean_npx(
  df = npx_data1,
  check_log = check_log,
  verbose = TRUE
)

## End(Not run)
```

manifest *Example Sample Manifest*

Description

Synthetic sample manifest to demonstrate use of functions in this package.

Usage

```
manifest
```

Format

This dataset contains columns:

SubjectID Subject Identifier, A-Z

Visit Visit Number, 1-6

SampleID 138 unique sample IDs

Site Site1 or Site2

Details

A tibble with 138 rows and 4 columns. This manifest contains 26 example subjects, with 6 visits and 2 sites.

norm_internal_adjust *Combine reference and non-reference datasets*

Description

The function is used by `norm_internal_subset` and `norm_internal_bridge` to combine the reference dataset that has `Adj_factor = 0` and the non-reference dataset that used the adjustment factors provided in `adj_fct_df`.

Usage

```
norm_internal_adjust(
  ref_df,
  ref_name,
  ref_cols,
  not_ref_df,
  not_ref_name,
  not_ref_cols,
  adj_fct_df
)
```

Arguments

ref_df	The reference dataset to be used in normalization (required).
ref_name	Project name of the reference dataset (required).
ref_cols	Named list of column names in the reference dataset (required).
not_ref_df	The non-reference dataset to be used in normalization (required).
not_ref_name	Project name of the non-reference dataset (required).
not_ref_cols	Named list of column names in the non-reference dataset (required).
adj_fct_df	Dataset containing the adjustment factors to be applied to the non-reference dataset for (required).

Details

The function calls norm_internal_adjust_ref and norm_internal_adjust_not_ref and combines their outputs.

Value

Tibble or ArrowObject with the normalized dataset.

Author(s)

Klev Diamanti

---

norm_internal_adjust_not_ref

Add adjustment factors to a dataset

Description

Add adjustment factors to a dataset

Usage

```
norm_internal_adjust_not_ref(df, name, cols, adj_fct_df, adj_fct_cols)
```

Arguments

df	The dataset to be normalized (required).
name	Project name of the dataset (required).
cols	Named list of column names in the dataset (required).
adj_fct_df	Dataset containing the adjustment factors to be applied to the dataset not_ref_df (required).
adj_fct_cols	Named list of column names in the dataset containing adjustment factors (required).

Value

Tibble or ArrowObject with the normalized dataset with additional columns "Project" and "Adj_factor".

Author(s)

Klev Diamanti

`norm_internal_adjust_ref` *Modify the reference dataset to be combined with the non-reference normalized dataset*

Description

Modify the reference dataset to be combined with the non-reference normalized dataset

Usage

```
norm_internal_adjust_ref(ref_df, ref_name)
```

Arguments

`ref_df`	The reference dataset to be used in normalization (required).
`ref_name`	Project name of the reference dataset (required).

Value

Tibble or ArrowObject with the reference dataset with additional columns "Project" and "Adj_factor".

Author(s)

Klev Diamanti

`norm_internal_assay_median` *Compute median value of the quantification method for each Olink assay*

Description

The function computes the median value of the the quantification method for each Olink assay in the set of samples `samples`, and it adds the column `Project`.

Usage

```
norm_internal_assay_median(df, samples, name, cols)
```

Arguments

df	The dataset to calculate medians from (required).
samples	Character vector of sample identifiers to be used for adjustment factor calculation in the dataset df (required).
name	Project name of the dataset that will be added in the column Project (required).
cols	Named list of column names identified in the dataset df (required).

Details

This function is typically used by internal functions norm_internal_subset and norm_internal_reference_median that compute median quantification value for each assay across multiple samples specified by samples.

Value

Tibble or ArrowObject with one row per Olink assay and the columns OlinkID, Project, and assay_med

Author(s)

Klev Diamanti

---

norm_internal_bridge *Internal bridge normalization function*

Description

Internal bridge normalization function

Usage

```
norm_internal_bridge(
  ref_df,
  ref_samples,
  ref_name,
  ref_cols,
  not_ref_df,
  not_ref_name,
  not_ref_cols
)
```

Arguments

ref_df	The reference dataset to be used in normalization (required).
ref_samples	Character vector of sample identifiers to be used for adjustment factor calculation in the reference dataset (required).
ref_name	Project name of the reference dataset (required).
ref_cols	Named list of column names in the reference dataset (required).
not_ref_df	The non-reference dataset to be used in normalization (required).
not_ref_name	Project name of the non-reference dataset (required).
not_ref_cols	Named list of column names in the non-reference dataset (required).

Value

Tibble or ArrowObject with the normalized dataset.

Author(s)

Klev Diamanti

---

norm_internal_cross_product

Internal function normalizing Olink Explore 3k to Olink Explore 3072

Description

Internal function normalizing Olink Explore 3k to Olink Explore 3072

Usage

```
norm_internal_cross_product(
  ref_df,
  ref_samples,
  ref_name,
  ref_cols,
  prod_uniq,
  not_ref_df,
  not_ref_name,
  not_ref_cols
)
```

Arguments

ref_df	The reference dataset to be used in normalization (required).
ref_samples	Character vector of sample identifiers to be used for adjustment factor calculation in the reference dataset (required).
ref_name	Project name of the reference dataset (required).
ref_cols	Named list of column names in the reference dataset (required).
prod_uniq	Name of products (not_ref, ref)
not_ref_df	The non-reference dataset to be used in normalization (required).
not_ref_name	Project name of the non-reference dataset (required).
not_ref_cols	Named list of column names in the non-reference dataset (required).

Value

Tibble or ArrowObject with a dataset with the following additional columns:

- OlinkID_E3072: Corresponding assay identifier from Olink Explore 3072.
- Project: Project of origin.
- BridgingRecommendation: Recommendation of whether the assay is bridgeable or not. One of "NotBridgeable", "MedianCentering", or "QuantileSmoothing".
- MedianCenteredNPX: NPX values adjusted based on the median of the pair-wise differences of NPX values between bridge samples.
- QSNormalizedNPX: NPX values adjusted based on the quantile smoothing normalization among bridge samples.

Author(s)

Klev Diamanti

---

norm_internal_reference_median

Internal reference median normalization function

Description

Internal reference median normalization function

Usage

```
norm_internal_reference_median(
  ref_df,
  ref_samples,
  ref_name,
  ref_cols,
  reference_medians
)
```

Arguments

ref_df	The reference dataset to be used in normalization (required).
ref_samples	Character vector of sample identifiers to be used for adjustment factor calculation in the reference dataset (required).
ref_name	Project name of the reference dataset (required).
ref_cols	Named list of column names in the reference dataset (required).
reference_medians	Dataset with columns "OlinkID" and "Reference_NPX" (required). Used for reference median normalization.

Value

Tibble or ArrowObject with the normalized dataset.

Author(s)

Klev Diamanti

---

norm_internal_rename_cols

Update column names of non-reference dataset based on those of reference dataset

---

Description

This function handles cases when specific columns referring to the same thing are named differently in df1 and df2 normalization datasets. It only renames columns panel_version, qc_warn, and assay_warn based on their names in the reference dataset.#'

Usage

```
norm_internal_rename_cols(ref_cols, not_ref_cols, not_ref_df)
```

Arguments

ref_cols	Named list of column names identified in the reference dataset.
not_ref_cols	Named list of column names identified in the non-reference dataset.
not_ref_df	Non-reference dataset to be used in normalization.

Value

not_ref_df with updated column names.

Author(s)

Klev Diamanti

`norm_internal_subset` *Internal subset normalization function*

Description

This function performs subset normalization using a subset of the samples from either or both reference and non-reference datasets. When all samples from each dataset are used, the function performs intensity normalization.

Usage

```
norm_internal_subset(
  ref_df,
  ref_samples,
  ref_name,
  ref_cols,
  not_ref_df,
  not_ref_samples,
  not_ref_name,
  not_ref_cols
)
```

Arguments

`ref_df`	The reference dataset to be used in normalization (required).
`ref_samples`	Character vector of sample identifiers to be used for adjustment factor calculation in the reference dataset (required).
`ref_name`	Project name of the reference dataset (required).
`ref_cols`	Named list of column names in the reference dataset (required).
`not_ref_df`	The non-reference dataset to be used in normalization (required).
`not_ref_samples`	Character vector of sample identifiers to be used for adjustment factor calculation in the non-reference dataset (required).
`not_ref_name`	Project name of the non-reference dataset (required).
`not_ref_cols`	Named list of column names in the non-reference dataset (required).

Value

Tibble or ArrowObject with the normalized dataset.

Author(s)

Klev Diamanti

norm_internal_update_maxlod *Update MaxLOD to the maximum MaxLOD across normalized datasets.*

Description

Update MaxLOD to the maximum MaxLOD across normalized datasets.

Usage

```
norm_internal_update_maxlod(df, cols)
```

Arguments

df	Normalized Olink dataset (required).
cols	Named list of column names in the dataset (required).

Value

The same dataset as the input *df* with the column reflecting MaxLOD updated.

npx_data1 *NPX Data in Long format.*

Description

This is a synthetic dataset aiming to use-cases of functions from this package.

Usage

```
npx_data1
```

Format

In addition to standard read_npx() columns, this dataset also contains columns:

Subject Subject Identifier

Treatment Treated or Untreated

Site Site indicator, 5 unique values

Time Baseline, Week.6 and Week.12

Project Project ID number

Details

A tibble with 29,440 rows and 17 columns.

npx_data1 is an Olink NPX data file (tibble) in long format with 158 unique Sample identifiers (including 2 repeats each of control samples: CONTROL_SAMPLE_AS 1 and CONTROL_SAMPLE_AS 2). The data also contains 1104 assays uniquely identified using OlinkID over 2 Olink Panels.

---

npx_data2 *NPX Data in Long format, a follow-up.*

Description

This is a synthetic dataset aiming to use-cases of functions from this package. The format is very similar to *npx_data1*. Both datasets can be used to demonstrate the use of normalization functionality.

Usage

```
npx_data2
```

Format

In addition to standard read_npx() columns, this dataset also contains columns:

Subject Subject Identifier

Treatment Treated or Untreated

Site Site indicator, 5 unique values

Time Baseline, Week.6 and Week.12

Project Project ID number

Details

A tibble with 32,384 rows and 17 columns.

npx_data2 is an Olink NPX data file (tibble) in long format with 174 unique Sample identifiers (including 2 repeats each of control samples: *CONTROL_SAMPLE_AS 1* and *CONTROL_SAMPLE_AS 2*). The data also contains 1,104 assays uniquely identified using OlinkID over 2 Panels. This dataset also contains 16 bridge samples with SampleIDs that are also present in data *npx_data1*. These samples are: A13, A29, A30, A36, A45, A46, A52, A63, A71, A73, B3, B4, B37, B45, B63 and B75.

olink_anova *Function which performs an ANOVA per protein.*

Description

Performs an ANOVA F-test for each assay (by OlinkID) in every panel using car::Anova and Type III sum of squares. The function handles both factor and numerical variables and/or covariates.

Usage

```
olink_anova(
  df,
  variable,
  check_log = NULL,
  outcome = "NPX",
  covariates = NULL,
  model_formula,
  return.covariates = FALSE,
  verbose = TRUE
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
variable	Single character value or character array. Variable(s) to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' or '*' notation.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
outcome	Character. The dependent variable. Default: NPX.
covariates	Single character value or character array. Default: NULL. Covariates to include. Takes ':' or '*' notation. Crossed analysis will not be inferred from main effects.
model_formula	(optional) Symbolic description of the model to be fitted in standard formula notation (e.g. "NPX~A*B"). If provided, this will override the outcome, variable and covariates arguments. Can be a string or of class stats::formula().
return.covariates	Boolean. Default: False. Returns F-test results for the covariates. Note: Adjusted p-values will be NA for the covariates.
verbose	Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Details

Samples that have no variable information or missing factor levels are automatically removed from the analysis (specified in a message if verbose = TRUE). Character columns in the input dataframe are automatically converted to factors (specified in a message if verbose = TRUE). Numerical variables are not converted to factors. Control samples should be removed before using this function. Control assays (AssayType is not "assay", or Assay contains "control" or "ctrl") should be removed before using this function. If a numerical variable is to be used as a factor, this conversion needs to be done on the dataframe before the function call.

Crossed analysis, i.e. A*B formula notation, is inferred from the variable argument in the following cases:

- c('A','B')
- c('A: B')
- c('A: B', 'B') or c('A: B', 'A')

Inference is specified in a message if verbose = TRUE.

For covariates, crossed analyses need to be specified explicitly, i.e. two main effects will not be expanded with a c('A','B') notation. Main effects present in the variable takes precedence. The formula notation of the final model is specified in a message if verbose = TRUE.

Adjusted p-values are calculated by stats::p.adjust according to the Benjamini & Hochberg (1995) method ("fdr"). The threshold is determined by logic evaluation of Adjusted_pval < 0.05. Covariates are not included in the p-value adjustment.

Value

A "tibble" containing the ANOVA results for every protein. The tibble is arranged by ascending p-values. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- df: "numeric" degrees of freedom
- sumsq: "numeric" sum of square
- meansq: "numeric" mean of square
- statistic: "numeric" value of the statistic
- p.value: "numeric" nominal p-value
- Adjusted_pval: "numeric" adjusted p-value for the test (Benjamini&Hochberg)
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("broom", "car"))) {
  #data
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control|ctrl",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  # check data
  npx_df_check_log <- OlinkAnalyze::check_npx(
    df = npx_df
  )

  # One-way ANOVA, no covariates.
  # Results in a model NPX~Time
  anova_results <- OlinkAnalyze::olink_anova(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = "Time"
  )

  # Two-way ANOVA, one main effect covariate.
  # Results in model NPX~Treatment*Time+Site.
  anova_results <- OlinkAnalyze::olink_anova(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Treatment:Time"),
    covariates = "Site"
  )

  # One-way ANOVA, interaction effect covariate.
  # Results in model NPX~Treatment+Site:Time+Site+Time.
  anova_results <- OlinkAnalyze::olink_anova(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = "Treatment",
    covariates = "Site:Time"
  )
}
```

---

olink_anova_posthoc *Function which performs an ANOVA posthoc test per protein.*

Description

Performs a post hoc ANOVA test using emmeans::emmeans with Tukey p-value adjustment per assay (by OlinkID) for each panel at confidence level 0.95. See `olink_anova` for details of input notation.

Usage

```
olink_anova_posthoc(
  df,
  check_log = NULL,
  olinkid_list = NULL,
  variable,
  covariates = NULL,
  outcome = "NPX",
  model_formula,
  effect,
  effect_formula,
  mean_return = FALSE,
  post_hoc_padjust_method = "tukey",
  verbose = TRUE
)
```

Arguments

`df`	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
`olinkid_list`	Character vector of OlinkID's on which to perform post hoc analysis. If not specified, all assays in df are used.
`variable`	Single character value or character array. Variable(s) to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' notation.
`covariates`	Single character value or character array. Default: NULL. Covariates to include. Takes ':' or '*' notation. Crossed analysis will not be inferred from main effects.
`outcome`	Character. The dependent variable. Default: NPX.
`model_formula`	(optional) Symbolic description of the model to be fitted in standard formula notation (e.g. "NPX~A*B"). If provided, this will override the `outcome`, `variable` and `covariates` arguments. Can be a string or of class `stats::formula()`.
`effect`	Term on which to perform post-hoc. Character vector. Must be subset of or identical to variable.
`effect_formula`	(optional) A character vector specifying the names of the predictors over which estimated marginal means are desired as defined in the `emmeans` package. May also be a formula. If provided, this will override the `effect` argument. See `?emmeans::emmeans()` for more information.

`mean_return` Boolean. If true, returns the mean of each factor level rather than the difference in means (default). Note that no p-value is returned for mean_return = TRUE and no adjustment is performed.

`post_hoc_padjust_method` P-value adjustment method to use for post-hoc comparisons within an assay. Options include `tukey`, `sidak`, `bonferroni` and `none`.

`verbose` Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Details

The function handles both factor and numerical variables and/or covariates. Control samples should be removed before using this function. Control assays (AssayType is not "assay", or Assay contains "control" or "ctrl") should be removed before using this function. The posthoc test for a numerical variable compares the difference in means of the outcome variable (default: NPX) for 1 standard deviation difference in the numerical variable, e.g. mean NPX at mean(numerical variable) versus mean NPX at mean(numerical variable) + 1*SD(numerical variable).

Value

A "tibble" of posthoc tests for specified effect, arranged by ascending adjusted p-values. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- contrast: "character" the groups that were compared
- estimate: "numeric" difference in mean NPX between groups
- conf.low: "numeric" confidence interval for the mean (lower end)
- conf.high: "numeric" confidence interval for the mean (upper end)
- Adjusted_pval: "numeric" adjusted p-value for the test
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("car", "emmeans"))) {
  # data
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control|ctrl",
        x = .data[["SampleID"]],
        ignore.case = TRUE
       )
    )
```

```
  # check data
  npx_df_check_log <- OlinkAnalyze::check_npx(
    df = npx_df
  )

  # Two-way ANOVA, one main effect (Site) covariate.
  # Results in model NPX~Treatment*Time+Site.
  anova_results <- OlinkAnalyze::olink_anova(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Treatment:Time"),
    covariates = "Site"
  )

  # Posthoc test for the model NPX~Treatment*Time+Site,
  # on the interaction effect Treatment:Time with covariate Site.

  # Filtering out significant and relevant results.
  significant_assays <- anova_results |>
    dplyr::filter(
      .data[["Threshold"]] == "Significant"
      & .data[["term"]] == "Treatment:Time"
    ) |>
    dplyr::select(
      dplyr::all_of("OlinkID")
    ) |>
    dplyr::distinct() |>
    dplyr::pull()

  # Posthoc, all pairwise comparisons
  anova_posthoc_results <- OlinkAnalyze::olink_anova_posthoc(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Treatment:Time"),
    covariates = "Site",
    olinkid_list = significant_assays,
    effect = "Treatment:Time"
  )

  # Posthoc, treated vs untreated at each timepoint, adjusted for Site effect
  anova_posthoc_results <- OlinkAnalyze::olink_anova_posthoc(
    df = npx_df,
    check_log = npx_df_check_log,
    model_formula = "NPX~Treatment*Time+Site",
    olinkid_list = significant_assays,
    effect_formula = "pairwise~Treatment|Time"
  )
}
```

olink_boxplot *Function which plots boxplots of selected variables*

Description

Generates faceted boxplots of NPX vs. grouping variable(s) for a given list of proteins (OlinkIDs) using ggplot2::ggplot and ggplot2::geom_boxplot.

Usage

```
olink_boxplot(
  df,
  variable,
  olinkid_list,
  verbose = FALSE,
  number_of_proteins_per_plot = 6,
  posthoc_results = NULL,
  ttest_results = NULL,
  check_log = NULL,
  ...
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID (unique), UniProt and at least one grouping variable.
variable	A character vector or character value indicating which column to use as the x-axis and fill grouping variable. The first or single value is used as x-axis, the second as fill. Further values in a vector are not plotted.
olinkid_list	Character vector indicating which proteins (OlinkIDs) to plot.
verbose	Boolean. If the plots are shown as well as returned in the list (default is false).
number_of_proteins_per_plot	Number of boxplots to include in the facet plot (default 6).
posthoc_results	Data frame from ANOVA posthoc analysis using olink_anova_posthoc() function.
ttest_results	Data frame from ttest analysis using olink_ttest() function.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df...
...	coloroption passed to specify color order.

Value

A list of objects of class "ggplot" (the actual ggplot object is entry 1 in the list). Box and whisker plot of NPX (y-axis) by variable (x-axis) for each Assay.

Examples

```
if (rlang::is_installed(pkg = c("broom", "car"))) {
  npx_df <- npx_data1 |>
    dplyr::filter(
      !grepl(pattern = "control|ctrl",
             x = .data[["SampleID"]],
             ignore.case = TRUE)
    )
  anova_results <- OlinkAnalyze::olink_anova(
    df = npx_df,
    variable = "Site"
  )
  significant_assays <- anova_results |>
    dplyr::filter(
      .data[["Threshold"]] == "Significant"
    ) |>
    dplyr::pull(
      .data[["OlinkID"]]
    )
  OlinkAnalyze::olink_boxplot(
    df = npx_df,
    variable = "Site",
    olinkid_list = significant_assays,
    verbose = TRUE,
    number_of_proteins_per_plot = 3L
  )
}
```

---

olink_bridgeability_plot

Plots for each bridgeable assays between two products.

Description

Plots for each bridgeable assays between two products.

Usage

```
olink_bridgeability_plot(
  df,
  check_log = NULL,
  olink_id = NULL,
  median_counts_threshold = 150L,
  min_count = 10L
)
```

Arguments

df	A tibble containing the cross-product bridge normalized dataset generated by olink_normalization.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
olink_id	Character vector of Olink assay identifiers *OlinkID* for which bridgeability plots will be created. If null, plots for all assays in *data* will be created. (default = NULL)
median_counts_threshold	Threshold indicating the minimum median counts for each product (default = 150).
min_count	Threshold indicating the minimum number of counts per data point (default = 10). Data below *min_count* are excluded.

Value

An object of class "ggplot" containing 4 plots for each assay.

Author(s)

Amrita Kar Klev Diamanti

Generates a combined plot per assay containing a violin and boxplot for IQR ranges; correlation plot of NPX values; a median count bar plot and KS plots from the 2 products.

Examples

```
if (rlang::is_installed(pkg = c("ggpubr"))) {
  npx_ht <- OlinkAnalyze:::data_ht_small |>
    dplyr::filter(
      .data[["SampleType"]] == "SAMPLE"
    )

  npx_3072 <- OlinkAnalyze:::data_3k_small |>
    dplyr::filter(
      .data[["SampleType"]] == "SAMPLE"
    )

  overlapping_samples <- intersect(
    x = npx_ht$SampleID,
    y = npx_3072$SampleID
  )

  data_norm <- OlinkAnalyze::olink_normalization(
    df1 = npx_ht,
    df2 = npx_3072,
    overlapping_samples_df1 = overlapping_samples,
    df1_project_nr = "Explore HT",
    df2_project_nr = "Explore 3072",
    reference_project = "Explore HT",
```

```
    df1_check_log = check_npx(df = npx_ht) |>
      suppressMessages() |>
      suppressWarnings(),
    df2_check_log = check_npx(df = npx_3072) |>
      suppressMessages() |>
      suppressWarnings()
  )

  data_norm_bridge_p <- OlinkAnalyze::olink_bridgeability_plot(
    df = data_norm,
    check_log = check_npx(df = data_norm) |>
      suppressMessages() |>
      suppressWarnings(),
    olink_id = c("OID40770", "OID40835"),
    median_counts_threshold = 150L,
    min_count = 10L
  )
}
```

`olink_bridge_selector` *Bridge selection function*

Description

The bridge selection function will select a number of bridge samples based on the input data. It selects samples with good detection that pass QC and cover a good range of the data. If possible, Olink recommends 8-16 bridge samples. When running the selector, Olink recommends starting at sample_missing_freq = 0.10 which represents a maximum of 10\ per sample. If there are not enough samples output, increase to 20\ The function accepts NPX Excel files with data < LOD replaced.

Usage

```
olink_bridge_selector(df, sample_missing_freq, n, check_log = NULL)

olink_bridgeselector(df, ..., n, check_log = NULL)
```

Arguments

`df`	Tibble/data frame in long format such as produced by the Olink Analyze read_npx function.
`sample_missing_freq`	The threshold for sample wise missingness.
`n`	Number of bridge samples to be selected.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.

...	Additional arguments. Currently only accepts sampleMissingFreq for backward compatibility. Please use sample_missing_freq instead of sampleMissingFreq in the future.

Details

olink_bridgeselector() is a synonym of olink_bridge_selector() .

Value

A "tibble" with sample IDs and mean NPX for a defined number of bridging samples. Columns include:

- SampleID: Sample ID
- PercAssaysBelowLOD: Percent of Assays that are below LOD for the sample
- MeanNPX: Mean NPX for the sample

Examples

```
check_log <- OlinkAnalyze::check_npx(df = npx_data1)

bridge_samples <- OlinkAnalyze::olink_bridge_selector(
  df = npx_data1,
  sample_missing_freq = 0.1,
  n = 20L,
  check_log = check_log
)
```

---

olink_color_discrete *Olink color scale for discrete ggplots*

Description

Olink color scale for discrete ggplots

Usage

```
olink_color_discrete(..., alpha = 1, coloroption = NULL)
```

Arguments

...	Optional. Additional arguments to pass to ggplot2::discrete_scale()
alpha	transparency (optional)
coloroption	string, one or more of the following: c("red", "orange", "yellow", "green", "teal", "turqoise", "lightblue", "darkblue", "purple", "pink")

Value

No return value, called for side effects

Examples

```
ggplot2::ggplot(
  data = datasets::mtcars,
  mapping = ggplot2::aes(
    x = .data[["wt"]],
    y = .data[["mpg"]],
    color = as.factor(x = .data[["cyl"]])
  )
) +
  ggplot2::geom_point(
    size = 4L
  ) +
  OlinkAnalyze::olink_color_discrete() +
  ggplot2::theme_bw()

ggplot2::ggplot(
  data = datasets::mtcars,
  mapping = ggplot2::aes(
    x = .data[["wt"]],
    y = .data[["mpg"]],
    color = as.factor(x = .data[["cyl"]])
  )
) +
  ggplot2::geom_point(
    size = 4L
  ) +
  OlinkAnalyze::olink_color_discrete(
    coloroption = c("lightblue", "red", "green")
  ) +
  ggplot2::theme_bw()
```

---

olink_color_gradient *Olink color scale for continuous ggplots*

Description

Olink color scale for continuous ggplots

Usage

```
olink_color_gradient(..., alpha = 1, coloroption = NULL)
```

Arguments

...	Optional. Additional arguments to pass to ggplot2::scale_color_gradientn()
alpha	transparency (optional)
coloroption	string, one or more of the following: c("red", "orange", "yellow", "green", "teal", "turqoise", "lightblue", "darkblue", "purple", "pink")

Value

No return value, called for side effects

Examples

```
ggplot2::diamonds |>
  dplyr::filter(
    .data[["x"]] > 5
    & .data[["x"]] < 6
    & .data[["y"]] > 5
    & .data[["y"]] < 6
  ) |>
  dplyr::mutate(
    diff = sqrt(
      x = abs(
        x = .data[["x"]] - .data[["y"]]
      )
    ) * sign(
      x = .data[["x"]] - .data[["y"]]
    )
  ) |>
  ggplot2::ggplot(
    mapping = ggplot2::aes(
      x = .data[["x"]],
      y = .data[["y"]],
      colour = .data[["diff"]]
    )
  ) +
  ggplot2::geom_point() +
  ggplot2::theme_bw() +
  OlinkAnalyze::olink_color_gradient()
```

---

olink_display_plate_dist

Plot distributions of a given variable for all plates

Description

Displays a bar chart for each plate representing the distribution of the given grouping variable on each plate using ggplot2::ggplot and ggplot2::geom_bar.

Usage

```
olink_display_plate_dist(data, fill.color = "plate")

olink_displayPlateDistributions(data, fill.color = "plate")
```

Arguments

data	tibble/data frame in long format returned from the olink_plate_randomizer function.
fill.color	Column name to be used as coloring variable for wells.

Value

An object of class "ggplot" showing the percent distribution of fill.color in each plate (x-axis)

Examples

```
randomized.manifest <- olink_plate_randomizer(manifest)
olink_display_plate_dist(data=randomized.manifest,
fill.color="Site")
```

---

olink_display_plate_layout

Plot all plates colored by a variable

Description

Displays each plate in a facet with cells colored by the given variable using ggplot and ggplot2::geom_tile.

Usage

```
olink_display_plate_layout(
  data,
  fill.color,
  PlateSize = 96L,
  num_ctrl = 8L,
  rand_ctrl = FALSE,
  Product,
  include.label = FALSE
)

olink_displayPlateLayout(
  data,
  fill.color,
  PlateSize = 96L,
  num_ctrl = 8L,
  rand_ctrl = FALSE,
```

```
  Product,
  include.label = FALSE
)
```

Arguments

data	tibble/data frame in long format returned from the olink_plate_randomizer function.
fill.color	Column name to be used as coloring variable for wells.
PlateSize	Integer. Either 96 or 48. 96 is default.
num_ctrl	Numeric. Number of controls on each plate (default = 8)
rand_ctrl	Logical. Whether controls are added to be randomized across the plate (default = FALSE)
Product	String. Name of Olink product used to set PlateSize if not provided. Optional.
include.label	Should the variable group be shown in the plot.

Value

An object of class "ggplot" showing each plate in a facet with the cells colored by values in column fill.color in input data.

Examples

```
randomized_manifest <- OlinkAnalyze::olink_plate_randomizer(
  Manifest = manifest
)
OlinkAnalyze::olink_display_plate_layout(
  data = randomized_manifest,
  fill.color = "Site"
)
```

---

olink_dist_plot *Function to plot the NPX distribution by panel*

Description

Generates boxplots of NPX vs. SampleID colored by QC_Warning (default) or any other grouping variable and faceted by Panel using ggplot and ggplot2::geom_boxplot.

Usage

```
olink_dist_plot(df, check_log = NULL, color_g = "QC_Warning", ...)
```

Arguments

df	NPX data frame in long format. Must have columns SampleID, NPX and Panel
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
color_g	Character value indicating which column to use as fill color. (default: QC_Warning).
...	Color option passed to specify color order.

Value

An object of class "ggplot" which displays NPX distribution for each sample per panel

Examples

```
# Optional: check and clean dataset
check_log <- OlinkAnalyze::check_npx(
  df = npx_data1
)

cleaned_data <- OlinkAnalyze::clean_npx(
  df = npx_data1,
  check_log = check_log
)

OlinkAnalyze::olink_dist_plot(
  df = npx_data1,
  check_log = check_log,
  color_g = "QC_Warning"
)

OlinkAnalyze::olink_dist_plot(
  df = cleaned_data,
  check_log = check_log,
  color_g = "QC_Warning"
)
```

---

olink_fill_discrete *Olink fill scale for discrete ggplots*

Description

Olink fill scale for discrete ggplots

Usage

```
olink_fill_discrete(..., alpha = 1, coloroption = NULL)
```

Arguments

...	Optional. Additional arguments to pass to ggplot2::discrete_scale()
alpha	transparency (optional)
coloroption	string, one or more of the following: c("red", "orange", "yellow", "green", "teal", "turqoise", "lightblue", "darkblue", "purple", "pink")

Value

No return value, called for side effects

Examples

```
ggplot2::diamonds |>
  dplyr::filter(
    .data[["x"]] > 5
    & .data[["x"]] < 6
    & .data[["y"]] > 5
    & .data[["y"]] < 6
  ) |>
  dplyr::mutate(
    diff = sqrt(
      x = abs(
        x = .data[["x"]] - .data[["y"]]
      )
    ) * sign(
      x = .data[["x"]] - .data[["y"]]
    )
  ) |>
  ggplot2::ggplot(
    mapping = ggplot2::aes(
      x = .data[["x"]],
      y = .data[["y"]],
      colour = .data[["diff"]]
    )
  ) +
  ggplot2::geom_point() +
  ggplot2::theme_bw() +
  OlinkAnalyze::olink_fill_discrete()
```

---

olink_fill_gradient *Olink fill scale for continuous ggplots*

Description

Olink fill scale for continuous ggplots

Usage

```
olink_fill_gradient(..., alpha = 1, coloroption = NULL)
```

Arguments

...	Optional. Additional arguments to pass to ggplot2::scale_fill_gradientn()
alpha	transparency (optional)
coloroption	string, one or more of the following: c("red", "orange", "yellow", "green", "teal", "turqoise", "lightblue", "darkblue", "purple", "pink")

Value

No return value, called for side effects

Examples

```
ggplot2::diamonds |>
  dplyr::filter(
    .data[["x"]] > 5
    & .data[["x"]] < 6
    & .data[["y"]] > 5
    & .data[["y"]] < 6
  ) |>
  dplyr::mutate(
    diff = sqrt(
      x = abs(
        x = .data[["x"]] - .data[["y"]]
      )
    ) * sign(
      x = .data[["x"]] - .data[["y"]]
    )
  ) |>
  ggplot2::ggplot(
    mapping = ggplot2::aes(
      x = .data[["x"]],
      y = .data[["y"]],
      colour = .data[["diff"]]
    )
  ) +
  ggplot2::geom_point() +
  ggplot2::theme_bw() +
  OlinkAnalyze::olink_fill_gradient()
```

`olink_format_oid_no_overlap` *Retrieve non-overlapping assays between two NPX datasets*

Description

For use in `olink_normalization_format` function. Generates a message stating how many assays were not overlapping. Appends additional columns depending on the normalization type to match normalized data output. For cross-product normalization, splits any concatenated OlinkIDs.

Usage

```
olink_format_oid_no_overlap(lst_check)
```

Arguments

`lst_check`	Normalization input list checks generated by `olink_norm_input_check`.

Value

A combined "tibble" of Olink data in long format containing only the non-overlapping assays from each input dataset.

Author(s)

Danai Topouza Klev Diamanti

`olink_format_rm_ext_ctrl` *Remove negative controls and plate controls from dataset. For use in olink_normalization_format function. Generates a message stating which control samples were removed.*

Description

Remove negative controls and plate controls from dataset. For use in olink_normalization_format function. Generates a message stating which control samples were removed.

Usage

```
olink_format_rm_ext_ctrl(df, lst_check)
```

Arguments

`df`	NPX dataset to be processed.
`lst_check`	Normalization input list checks generated by `olink_norm_input_check`.

Value

A "tibble" of Olink data in long format containing the input dataset with negative controls and plate controls removed.

Author(s)

Danai G. Topouza Klev Diamanti

`olink_heatmap_plot` *Function to plot a heatmap of the NPX data*

Description

Generates a heatmap using `pheatmap::pheatmap` of all samples from NPX data.

Usage

```
olink_heatmap_plot(
  df,
  check_log = NULL,
  variable_row_list = NULL,
  variable_col_list = NULL,
  center_scale = TRUE,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  show_rownames = TRUE,
  show_colnames = TRUE,
  colnames = "both",
  annotation_legend = TRUE,
  fontsize = 10,
  na_col = "black",
  ...
)
```

Arguments

`df`	Data frame in long format with SampleID, NPX, OlinkID, Assay and columns of choice for annotations.
`check_log`	output from check_npx on `df`
`variable_row_list`	Columns in `df` to be annotated for rows in the heatmap.
`variable_col_list`	Columns in `df` to be annotated for columns in the heatmap.
`center_scale`	Logical. If data should be centered and scaled across assays (default `TRUE`).
`cluster_rows`	Logical. Determining if rows should be clustered (default `TRUE`).

cluster_cols	Logical. Determining if columns should be clustered (default TRUE).
show_rownames	Logical. Determining if row names are shown (default TRUE).
show_colnames	Logical. Determining if column names are shown (default TRUE).
colnames	Character. Determines how to label the columns. Must be 'assay', 'oid', or 'both' (default 'both').
annotation_legend	Logical. Determining if legend for annotations should be shown (default TRUE).
fontsize	Fontsize (default 10)
na_col	Color of cells with NA (default black)
...	Additional arguments used in pheatmap::pheatmap

Details

The values are by default scaled across and centered in the heatmap. Columns and rows are by default sorted by by dendrogram. Unique sample names are required.

Value

An object of class ggplot, generated from the gtable returned by pheatmap::pheatmap.

Examples

```
if (rlang::is_installed(pkg = c("ggplotify", "pheatmap"))) {
  npx_data <- npx_data1 |>
    dplyr::filter(
      !stringr::str_detect(
        string = .data[["SampleID"]],
        pattern = "CONT"
      )
    )
  check_log <- OlinkAnalyze::check_npx(
    df = npx_data
  )

  # Heatmap
  OlinkAnalyze::olink_heatmap_plot(
    df = npx_data,
    check_log = check_log
  )

  # Heatmap with annotation
  OlinkAnalyze::olink_heatmap_plot(
    df = npx_data,
    check_log = check_log,
    variable_row_list = c("Time", "Site")
  )

  # Heatmap with calls from pheatmap
  OlinkAnalyze::olink_heatmap_plot(
```

```
    df = npx_data,
    check_log = check_log,
    cutree_rows = 3L
  )
}
```

`olink_iqr` *Compute inter-quartile range (IQR) of multiplied by a fixed value*

Description

Compute inter-quartile range (IQR) of multiplied by a fixed value

Usage

```
olink_iqr(df, quant_col, iqr_group, iqr_sd)
```

Arguments

`df`	Olink dataset
`quant_col`	Character vector of name of quantification column
`iqr_group`	Grouping for which to compute IQR for
`iqr_sd`	Fixed value to multiply IQR with

Value

Input dataset with two additional columns, iqr and iqr_sd

`olink_lmer` *Function that performs a linear mixed model per protein.*

Description

Fits a linear mixed effects model for every protein (by OlinkID) in every panel, using `lmerTest::lmer` and `stats::anova`. The function handles both factor and numerical variables and potential covariates.

Usage

```
olink_lmer(
  df,
  variable,
  check_log = NULL,
  outcome = "NPX",
  random,
  covariates = NULL,
  model_formula,
  return.covariates = FALSE,
  verbose = TRUE
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, 1-2 variables with at least 2 levels.
variable	Single character value or character array. Variables to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' or '*' notation.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
outcome	Character. The dependent variable. Default: NPX.
random	Single character value or character array.
covariates	Single character value or character array. Default: NULL. Covariates to include. Takes ':' or '*' notation. Crossed analysis will not be inferred from main effects.
model_formula	(optional) Symbolic description of the model to be fitted in standard formula notation (e.g. NPX~A*B + (1|ID)). If provided, this will override the outcome, variable and covariates. arguments. Can be a string or of class stats::formula().
return.covariates	Boolean. Default: FALSE. Returns results for the covariates. Note: Adjusted p-values will be NA for the covariates.
verbose	Boolean. Default: TRUE. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Details

Samples that have no variable information or missing factor levels are automatically removed from the analysis (specified in a message if verbose = TRUE). Character columns in the input dataset are automatically converted to factors (specified in a message if verbose = TRUE). Numerical variables are not converted to factors. If a numerical variable is to be used as a factor, this conversion needs to be done on the dataset before the function call.

Crossed analysis, i.e. A*B formula notation, is inferred from the variable argument in the following cases:

- c('A','B')

- c('A:B')
- c('A:B', 'B') or c('A:B', 'A')

Inference is specified in a message if `verbose = TRUE`.

For covariates, crossed analyses need to be specified explicitly, i.e. two main effects will not be expanded with a `c('A','B')` notation. Main effects present in the variable takes precedence. The random variable only takes main effects. The formula notation of the final model is specified in a message if `verbose = TRUE`.

Output p-values are adjusted by `stats::p.adjust` according to the Benjamini-Hochberg method ("fdr"). Adjusted p-values are logically evaluated towards adjusted `p-value<0.05`.

Value

A "tibble" containing the results of fitting the linear mixed effects model to every protein by OlinkID, ordered by ascending p-value. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- sumsq: "numeric" sum of square
- meansq: "numeric" mean of square
- NumDF: "integer" numerator of degrees of freedom
- DenDF: "numeric" denominator of decrees of freedom
- statistic: "numeric" value of the statistic
- p.value: "numeric" nominal p-value
- Adjusted_pval: "numeric" adjusted p-value for the test (Benjamini&Hochberg)
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("lme4", "lmerTest", "broom"))) {
  #data
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control|ctrl",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  # check data
  npx_df_check_log <- OlinkAnalyze::check_npx(
    df = npx_df
```

```
  )

  # Results in model NPX ~ Time * Treatment + (1 | Subject) + (1 | Site)
  lmer_results <- OlinkAnalyze::olink_lmer(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Time", "Treatment"),
    random = c("Subject", "Site")
  )
}
```

`olink_lmer_plot` *Function which performs a point-range plot per protein on a linear mixed model*

Description

Generates a point-range plot faceted by Assay using `ggplot` and `ggplot2::geom_pointrange` based on a linear mixed effects model using `lmerTest:lmer` and `emmeans::emmeans`. See `olink_lmer` for details of input notation.

Usage

```
olink_lmer_plot(
  df,
  check_log = NULL,
  variable,
  outcome = "NPX",
  random,
  olinkid_list = NULL,
  covariates = NULL,
  x_axis_variable,
  col_variable = NULL,
  number_of_proteins_per_plot = 6L,
  verbose = FALSE,
  ...
)
```

Arguments

`df`	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, 1-2 variables with at least 2 levels.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.

variable	Single character value or character array. Variable(s) to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' or '*' notation.
outcome	Character. The dependent variable. Default: NPX.
random	Single character value or character array.
olinkid_list	Character vector indicating which proteins (by OlinkID) for which to create figures.
covariates	Single character value or character array. Default: NULL. Covariates to include. Takes ':' or '*' notation. Crossed analysis will not be inferred from main effects.
x_axis_variable	Character. Which main effect to use as x-axis in the plot.
col_variable	Character. If provided, the interaction effect col_variable:x_axis_variable will be plotted with x_axis_variable on the x-axis and col_variable as color.
number_of_proteins_per_plot	Number plots to include in the list of point-range plots. Defaults to 6 plots per figure
verbose	Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.
...	coloroption for color ordering

Value

A list of objects of class "ggplot" showing point-range plot of NPX (y-axis) over x_axis_variable for each assay (facet), colored by col_variable if provided.

Examples

```
if (rlang::is_installed(pkg = c("lme4", "lmerTest", "broom", "emmeans"))) {
  #data
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control|ctrl",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  # check data
  npx_df_check_log <- OlinkAnalyze::check_npx(
    df = npx_df
  )

  # Results in model NPX ~ Time * Treatment + (1 | Subject) + (1 | Site)
  lmer_results <- OlinkAnalyze::olink_lmer(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Time", "Treatment"),
```

```
    random = c("Subject")
  )

  # List of significant proteins for the interaction effect Time:Treatment
  assay_list <- lmer_results |>
    dplyr::filter(
    .data[["Threshold"]] == "Significant"
    & .data[["term"]] == "Time:Treatment"
  ) |>
    dplyr::distinct(.data[["OlinkID"]]) |>
    dplyr::pull()

  lst_pointrange_plots <- OlinkAnalyze::olink_lmer_plot(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Time", "Treatment"),
    random = c("Subject"),
    x_axis_variable = "Time",
    col_variable = "Treatment",
    verbose = TRUE,
    olinkid_list = assay_list,
    number_of_proteins_per_plot = 10L
  )
}
```

`olink_lmer_posthoc` *Function which performs a linear mixed model posthoc per protein.*

Description

Similar to `olink_lmer` but performs a post-hoc analysis based on a linear mixed model effects model using `lmerTest::lmer` and `emmeans::emmeans` on proteins. See `olink_lmer` for details of input notation.

Usage

```
olink_lmer_posthoc(
  df,
  check_log = NULL,
  olinkid_list = NULL,
  variable,
  outcome = "NPX",
  random,
  model_formula,
  effect,
  effect_formula,
  covariates = NULL,
```

```
  mean_return = FALSE,
  post_hoc_padjust_method = "tukey",
  verbose = TRUE
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, 1-2 variables with at least 2 levels and subject identifier.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
olinkid_list	Character vector of OlinkID's on which to perform post hoc analysis. If not specified, all assays in df are used.
variable	Single character value or character array. Variables to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' or '*' notation.
outcome	Character. The dependent variable. Default: NPX.
random	Single character value or character array.
model_formula	(optional) Symbolic description of the model to be fitted in standard formula notation (e.g. NPX~A*B + (1|ID)). If provided, this will override the outcome, variable and covariates. arguments. Can be a string or of class stats::formula().
effect	Term on which to perform post-hoc. Character vector. Must be subset of or identical to variable.
effect_formula	(optional) A character vector specifying the names of the predictors over which estimated marginal means are desired as defined in the emmeans package. May also be a formula. If provided, this will override the effect argument. See ?emmeans::emmeans() for more information.
covariates	Single character value or character array. Default: NULL. Covariates to include. Takes ':' or '*' notation. Crossed analysis will not be inferred from main effects.
mean_return	Boolean. If true, returns the mean of each factor level rather than the difference in means (default). Note that no p-value is returned for mean_return = TRUE and no adjustment is performed.
post_hoc_padjust_method	P-value adjustment method to use for post-hoc comparisons within an assay. Options include tukey, sidak, bonferroni and none.
verbose	Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Details

The function handles both factor and numerical variables and/or covariates. Differences in estimated marginal means are calculated for all pairwise levels of a given variable. Degrees of freedom are estimated using Satterthwaite's approximation. The posthoc test for a numerical variable compares the difference in means of the outcome variable (default: NPX) for 1 standard deviation difference in the numerical variable, e.g. mean NPX at mean(numerical variable) versus mean NPX at mean(numerical variable) + 1*SD(numerical variable). The output tibble is arranged by ascending Tukey adjusted p-values.

Value

A "tibble" containing the results of the pairwise comparisons between given variable levels for proteins specified in olinkid_list (or full df). Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- contrast: "character" the groups that were compared
- estimate: "numeric" difference in mean NPX between groups
- conf.low: "numeric" confidence interval for the mean (lower end)
- conf.high: "numeric" confidence interval for the mean (upper end)
- Adjusted_pval: "numeric" adjusted p-value for the test
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("lme4", "lmerTest", "emmeans", "broom"))) {
  #data
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control|ctrl",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  # check data
  npx_df_check_log <- OlinkAnalyze::check_npx(
    df = npx_df
  )

  # Results in model NPX ~ Time * Treatment + (1 | Subject)
  lmer_results <- OlinkAnalyze::olink_lmer(
    df = npx_df,
    check_log = npx_df_check_log,
    variable = c("Time", "Treatment"),
    random = c("Subject")
  )

  # List of significant proteins for the interaction effect Time:Treatment
  assay_list <- lmer_results |>
    dplyr::filter(
    .data[["Threshold"]] == "Significant"
    & .data[["term"]] == "Time:Treatment"
  ) |>
```

```
    dplyr::distinct(.data[["OlinkID"]]) |>
    dplyr::pull()

  # Run lmer posthoc on significant proteins
  results_lmer_posthoc <- OlinkAnalyze::olink_lmer_posthoc(
    df = npx_df,
    check_log = npx_df_check_log,
    olinkid_list = assay_list,
    variable = c("Time", "Treatment"),
    effect = "Time:Treatment",
    random = "Subject",
    verbose = TRUE
  )

  # Estimate treated vs untreated at each timepoint
  results_lmer_posthoc <- OlinkAnalyze::olink_lmer_posthoc(
    df = npx_df,
    check_log = npx_df_check_log,
    olinkid_list = assay_list,
    model_formula = "NPX~Time*Treatment+(1|Subject)",
    effect_formula = "pairwise~Treatment|Time",
    verbose = TRUE
  )
}
```

`olink_lod` *Calculate LOD using Negative Controls or Fixed LOD*

Description

Calculate LOD using Negative Controls or Fixed LOD

Usage

```
olink_lod(data, check_log = NULL, lod_file_path = NULL, lod_method = "NCLOD")
```

Arguments

`data`	npx data file
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
`lod_file_path`	location of lod file from Olink. Only needed if lod_method = "FixedLOD" or "Both". Default `NULL`.
`lod_method`	method for calculating LOD using either "FixedLOD" or negative controls ("NCLOD"), or both ("Both"). Default `NCLOD`.

Value

A dataframe with 2 additional columns, LOD and PCNormalizedLOD if `lod_method` is FixedLOD or NCLOD. When `Normalization = "Plate Control"`, LOD and PCNormalizedLOD are identical.

If `lod_method` is "Both", 4 additional columns will be added:

- NCLOD - LOD calculated from negative controls and normalized based on normalization column
- NCPCNormalizedLOD - PC Normalized LOD calculated from negative controls
- FixedLOD - LOD calculated from fixed LOD file and normalized based on normalization column
- FixedPCNormalizedLOD - PC Normalized LOD calculated from fixed LOD file

Examples

```
## Not run:
  \donttest{
  try(
    {
      # This will fail if the files do not exist.

      # Import NPX data
      npx_data <- read_npx(filename = "path/to/npx_file")

      # Check NPX data
      check_log <- check_npx(df = npx_data)

      # Clean NPX data
      npx_data_clean <- clean_npx(
        df = npx_data,
        check_log = check_log
      )

      # Re-check NPX data
      check_log_clean <- check_npx(df = npx_data_clean)

      # Estimate LOD from negative controls
      npx_data_lod_nc <- olink_lod(
        data = npx_data_clean,
        check_log = check_log_clean,
        lod_method = "NCLOD"
      )

      # Estimate LOD from fixed LOD
      ## Locate the fixed LOD file
      lod_file_path <- "path/to/lod_file"

      npx_data_lod_Fixed <- olink_lod(
        data = npx_data,
        check_log = check_log_clean,
```

```
      lod_file_path = lod_file_path,
      lod_method = "FixedLOD"
    )

    # Estimate LOD from both negative controls and fixed LOD
    npx_data_lod_both <- olink_lod(
      data = npx_data,
      check_log = check_log_clean,
      lod_file_path = lod_file_path,
      lod_method = "Both"
    )
  }
)
}

## End(Not run)
```

`olink_median` *Compute median of quantified value*

Description

Compute median of quantified value

Usage

```
olink_median(df, quant_col, median_group)
```

Arguments

`df`	Olink dataset
`quant_col`	Character vector of name of quantification column
`median_group`	Grouping for which to compute median for

Value

Input dataset with one additional columns, median

olink_median_iqr_outlier *Compute outliers based on median +/- iqr_sd * IQR*

Description

Compute outliers based on median +/- iqr_sd * IQR

Usage

```
olink_median_iqr_outlier(df, quant_col, group, iqr_sd)
```

Arguments

df	Olink dataset
quant_col	Character vector of name of quantification column
group	Grouping for which to compute median for
iqr_sd	Fixed value to multiply IQR with

Value

Boolean vector with length equal to the number of input rows indicating outlier.

olink_normalization *Normalize two Olink datasets*

Description

Normalizes two Olink datasets to each other, or one Olink dataset to a reference set of medians values.

Usage

```
olink_normalization(
  df1,
  df2 = NULL,
  overlapping_samples_df1,
  overlapping_samples_df2 = NULL,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P1",
  reference_medians = NULL,
  format = FALSE,
  df1_check_log = NULL,
  df2_check_log = NULL
)
```

Arguments

df1	First dataset to be used for normalization (required).
df2	Second dataset to be used for normalization. Required for bridge and subset normalization.
overlapping_samples_df1	Character vector of samples to be used for the calculation of adjustment factors in df1 (required).
overlapping_samples_df2	Character vector of samples to be used for the calculation of adjustment factors in df2. Required for subset normalization.
df1_project_nr	Project name of first dataset (required).
df2_project_nr	Project name of second dataset. Required for bridge and subset normalization.
reference_project	Project to be used as reference project. Should be one of df1_project_nr and df2_project_nr. Required for bridge and subset normalization.
reference_medians	Dataset with columns "OlinkID" and "Reference_NPX". Required for reference median normalization.
format	Boolean that controls whether the normalized dataset will be formatted for input to downstream analysis.
df1_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df1.
df2_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df2.

Details

The function handles four different types of normalization:

- **Bridge normalization**: One of the datasets is adjusted to another using overlapping samples (bridge samples). Overlapping samples need to have the same identifiers in both datasets. Normalization is performed using the median of the pair-wise differences between the bridge samples in the two datasets. The two datasets are provided as df1 and df2, and the one being adjusted to is specified in the input reference_project; overlapping samples are specified in overlapping_samples_df1. Only overlapping_samples_df1 should be provided regardless of the dataset used as reference_project.
- **Subset normalization**: One of the datasets is adjusted to another using a subset of samples from each. Normalization is performed using the differences of the medians between the subsets from the two datasets. Both overlapping_samples_df1 and overlapping_samples_df2 need to be provided, and sample identifiers do not need to be the same.
  - A special case of subset normalization occurs when all samples (except control samples and samples with QC warnings) from each dataset are used for normalization; this special case is called intensity normalization. In intensity normalization all unique sample identifiers from df1 are provided as input in overlapping_samples_df1 and all unique sample identifiers from df2 are provided as input in overlapping_samples_df2.

- **Reference median normalization**: One of the datasets (`df1`) is adjusted to a predefined set of adjustment factors. This is effectively subset normalization, but using differences of medians to pre-recorded median values. `df1`, `overlapping_samples_df1`, `df1_project_nr` and `reference_medians` need to be specified. Dataset `df1` is normalized using the differences in median between the overlapping samples and the reference medians.
- **Cross-product normalization**: One of the datasets is adjusted to another using the median of pair-wise differences of overlapping samples (bridge samples) or quantile smoothing using overlapping samples as reference to adjust the distributions. Overlapping samples need to have the same identifiers in both datasets. The two datasets are provided as `df1` and `df2`, and the one being adjusted to is specified in the input `reference_project`; **Note that** in cross-product normalization the reference project is predefined, and in case the argument `reference_project` does not match the expected reference project an error will be returned. Overlapping samples are specified in `overlapping_samples_df1`. Only `overlapping_samples_df1` should be provided regardless of the dataset used as `reference_project`. This functionality **does not** modify the column with original quantification values (e.g. NPX), instead it normalizes it with 2 different approaches in columns "MedianCenteredNPX" and "QSNormalizedNPX", and provides a recommendation in "BridgingRecommendation" about which of the two columns is to be used.

The output dataset is `df1` if reference median normalization, or `df2` appended to `df1` if bridge, subset or cross-product normalization. The output dataset contains all original columns from the original dataset(s), and the columns:

- "Project" and "Adj_factor" in case of reference median, bridge and subset normalization. The former marks the project of origin based on `df1_project_nr` and `df2_project_nr`, and the latter the adjustment factor that was applied to the non-reference dataset.
- "Project", "OlinkID_E3072", "MedianCenteredNPX", "QSNormalizedNPX", "BridgingRecommendation" in case of cross-product normalization. The columns correspond to the project of origin based on `df1_project_nr` and `df2_project_nr`, the assay identifier in the non-reference project, the bridge-normalized quantification value, the quantile smoothing-normalized quantification value, and the recommendation about which of the two normalized values is more suitable for downstream analysis.

Value

Tibble or ArrowObject with the normalized dataset.

Examples

```
# prepare datasets
npx_df1 <- npx_data1 |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# check datasets
```

```
npx_df1_check <- check_npx(df = npx_df1)
npx_df2_check <- check_npx(df = npx_df2)

# bridge normalization

# overlapping samples - exclude control samples
overlap_samples <- intersect(x = npx_df1$SampleID,
                             y = npx_df2$SampleID) |>
  (\(x) x[!grepl("^CONTROL_SAMPLE", x)])()

# normalize
olink_normalization(
  df1 = npx_df1,
  df2 = npx_df2,
  overlapping_samples_df1 = overlap_samples,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P1",
  df1_check_log = npx_df1_check,
  df2_check_log = npx_df2_check
)

# subset normalization

# find a suitable subset of samples from each dataset:
# exclude control samples
# exclude samples that do not pass QC
df1_samples <- npx_df1 |>
  dplyr::group_by(
    dplyr::pick(
      dplyr::all_of("SampleID")
    )
  )|>
  dplyr::filter(
    all(.data[["QC_Warning"]] == 'Pass')
  ) |>
  dplyr::ungroup() |>
  dplyr::filter(
    !grepl(pattern = "^CONTROL_SAMPLE", x = .data[["SampleID"]])
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()
df2_samples <- npx_df2 |>
  dplyr::group_by(
    dplyr::pick(
      dplyr::all_of("SampleID")
    )
  )|>
  dplyr::filter(
    all(.data[["QC_Warning"]] == 'Pass')
```

```
  ) |>
  dplyr::ungroup() |>
  dplyr::filter(
    !grepl(pattern = "^CONTROL_SAMPLE", x = .data[["SampleID"]])
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()

# select a subset of samples from each set from above
df1_subset <- sample(x = df1_samples, size = 16L)
df2_subset <- sample(x = df2_samples, size = 20L)

# normalize
olink_normalization(
  df1 = npx_df1,
  df2 = npx_df2,
  overlapping_samples_df1 = df1_subset,
  overlapping_samples_df2 = df2_subset,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P1",
  df1_check_log = npx_df1_check,
  df2_check_log = npx_df2_check
)

# special case of subset normalization using all samples
olink_normalization(
  df1 = npx_df1,
  df2 = npx_df2,
  overlapping_samples_df1 = df1_samples,
  overlapping_samples_df2 = df2_samples,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P1",
  df1_check_log = npx_df1_check,
  df2_check_log = npx_df2_check
)

# reference median normalization

# For the sake of this example, set the reference median to 1
ref_med_df <- npx_data1 |>
  dplyr::select(
    dplyr::all_of(
      c("OlinkID")
    )
  ) |>
  dplyr::distinct() |>
  dplyr::mutate(
    Reference_NPX = runif(n = dplyr::n(),
                          min = -1,
```

```
                          max = 1)
  )

# normalize
olink_normalization(
  df1 = npx_df1,
  overlapping_samples_df1 = df1_subset,
  reference_medians = ref_med_df,
  df1_check_log = npx_df1_check
)

# cross-product normalization

# get reference samples
overlap_samples_product <- intersect(
  x = unique(OlinkAnalyze:::data_ht_small$SampleID),
  y = unique(OlinkAnalyze:::data_3k_small$SampleID)
) |>
  (\(.) .[!grepl("CONTROL", .)])()

# check datasets

npx_ht_check <- check_npx(df = OlinkAnalyze:::data_ht_small)
npx_3k_check <- check_npx(df = OlinkAnalyze:::data_3k_small)

# normalize
olink_normalization(
  df1 = OlinkAnalyze:::data_ht_small,
  df2 = OlinkAnalyze:::data_3k_small,
  overlapping_samples_df1 = overlap_samples_product,
  df1_project_nr = "proj_ht",
  df2_project_nr = "proj_3k",
  reference_project = "proj_ht",
  format = FALSE,
  df1_check_log = npx_ht_check,
  df2_check_log = npx_3k_check
)
```

`olink_normalization_bridge` *Bridge normalization of all proteins between two NPX projects.*

Description

Normalizes two NPX projects (data frames) using shared samples.

This function is a wrapper of olink_normalization.

Usage

```
olink_normalization_bridge(
  project_1_df,
  project_2_df,
  bridge_samples,
  project_1_name = "P1",
  project_2_name = "P2",
  project_ref_name = "P1",
  format = FALSE,
  project_1_check_log = NULL,
  project_2_check_log = NULL
)
```

Arguments

project_1_df	Data frame of the first project (required).
project_2_df	Data frame of the second project (required).
bridge_samples	Named list of 2 arrays containing SampleID of shared samples to be used for the calculation of adjustment factor. The names of the two arrays should be DF1 and DF2 corresponding to projects 1 and 2, respectively. Arrays should be of equal length and index of each entry should correspond to the same sample. (required)
project_1_name	Name of the first project (default: P1).
project_2_name	Name of the second project (default: P2).
project_ref_name	Name of the project to be used as reference set. Needs to be one of the project_1_name or project_2_name. It marks the project to which the other project will be adjusted to (default: P1).
format	Boolean that controls whether the normalized dataset will be formatted for input to downstream analysis.
project_1_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using project_1_df. (default: NULL)
project_2_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using project_2_df. (default: NULL)

Details

In bridging normalization one of the projects is adjusted to another using shared samples (bridge samples). It is not necessary for the shared samples to be named the same in each project. Adjustment between the two projects is made using the median of the paired differences between the shared samples. The two data frames are inputs project_1_df and project_2_df, the one being adjusted to is specified in the input project_ref_name and the shared samples are specified in bridge_samples.

Value

A "tibble" of NPX data in long format containing normalized NPX values, including adjustment factors and name of project.

Examples

```
# prepare datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find overlapping samples, but exclude Olink control
overlap_samples <- dplyr::intersect(x = unique(npx_df1[["SampleID"]]),
                                    y = unique(npx_df2[["SampleID"]]))
overlap_samples_list <- list("DF1" = overlap_samples,
                             "DF2" = overlap_samples)

# check npx
df1_check_log <- OlinkAnalyze::check_npx(df = npx_df1)
df2_check_log <- OlinkAnalyze::check_npx(df = npx_df2)

# Normalize
OlinkAnalyze::olink_normalization_bridge(
  project_1_df = npx_df1,
  project_2_df = npx_df2,
  bridge_samples = overlap_samples_list,
  project_1_name = "P1",
  project_2_name = "P2",
  project_ref_name = "P1",
  project_1_check_log = df1_check_log,
  project_2_check_log = df2_check_log
)
```

olink_normalization_bridgeable *Identify if assays shared between Olink Explore 3072 and Olink Explore HT can be bridged*

Description

The function uses a dataset from Olink Explore 3072 and a dataset from Olink Explore HT, and examines if the matched assays between the two products can be normalized to each other. The input datasets should be exported from Olink software and should not be altered prior to importing them to this function.

Usage

```
olink_normalization_bridgeable(lst_df, ref_cols, not_ref_cols, seed = 1)
```

Arguments

lst_df	A named list of the 2 input datasets. First element should be the reference dataset from Olink Explore HT and the second element should originate from Olink Explore 3072.
ref_cols	A named list with the column names to use. Exported from olink_norm_input_check.
not_ref_cols	A named list with the column names from the non-reference dataset. Exported from olink_norm_input_check.
seed	Integer random seed (Default: seek = 1).

Details

All processes below assume that the first element from *lst_df* is the reference dataset (e.g. Olink Explore HT), and the other element of the list is the non-reference dataset (e.g. Olink Explore 3072). The input datasets **have to be pre-processed** by olink_norm_input_check which will take care of mapping of assay identifiers and various checks. Also, the input datasets should exclusively contain datapoints from bridge samples. When this function is called from the function olink_normalization, then the list is created seamlessly in the background, and the datasets have been already processed by olink_norm_input_check.

The input *ref_cols* is a named list masking column names of the reference dataset. This list is generated automatically from olink_norm_input_check when it is called from olink_normalization. In addition, olink_normalization has also utilized norm_internal_rename_cols to rename the columns of the non-reference dataset according to the ones of the reference dataset, hence all column names should match.

Value

A "tibble" in long format with the following columns:

- OlinkID: Underscore-separated Olink identifiers of matching assays between Olink Explore HT and Olink Explore 3072.
- BridgingRecommendation: A character vector indicating whether the matching assays are considered as bridgeable or not, and the recommended type of normalization to perform.

Author(s)

Amrita Kar Marianne Sandin Danai G. Topouza Klev Diamanti

Examples

```
# check_npx
data_ht_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_ht_small
)

data_3k_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_3k_small
)

# check input datasets
data_explore_check <- OlinkAnalyze:::olink_norm_input_check(
  df1 = OlinkAnalyze:::data_3k_small,
  df1_check_log = data_3k_small_check,
  df2 = OlinkAnalyze:::data_ht_small,
  df2_check_log = data_ht_small_check,
  overlapping_samples_df1 = intersect(
    x = unique(OlinkAnalyze:::data_3k_small$SampleID),
    y = unique(OlinkAnalyze:::data_ht_small$SampleID)
  ) |>
    (\(x) x[!grepl("CONTROL", x)])() |>
    head(20L),
  overlapping_samples_df2 = NULL,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P2",
  reference_medians = NULL
)

# create lst_df
lst_df <- list(
  data_explore_check$ref_df,
  data_explore_check$not_ref_df
)
names(lst_df) <- c(data_explore_check$ref_name,
                   data_explore_check$not_ref_name)

# create ref_cols
ref_cols <- data_explore_check$ref_check_log$col_names
```

```
not_ref_cols <- data_explore_check$not_ref_check_log$col_names

# run olink_normalization_bridgeable
is_bridgeable_result <- OlinkAnalyze:::olink_normalization_bridgeable(
  lst_df = lst_df,
  ref_cols = ref_cols,
  not_ref_cols = not_ref_cols,
  seed = 1
)
```

`olink_normalization_format` *Format the output of olink_normalization for seamless use with downstream analysis functions.*

Description

For within-product bridging and subset normalization:

- Adds non-overlapping assays between projects to the bridged file without adjustment.
- Removes external controls, except sample controls.

For cross-product bridging:

- Adds non-overlapping assays between projects and not bridgeable assays to the bridged file without adjustment.
- Removes external controls, except sample controls.
- Replaces the NPX values of the non-reference project by the Median Centered or QS Normalized NPX, according to the Bridging Recommendation.
- Edits the BridgingRecommendation column to indicate whether an assay is NotBridgeable, NotOverlapping, MedianCentering, or QuantileSmoothing bridged.
- Replaces OlinkID by the concatenation of each product's OlinkIDs to record the OlinkIDs from both projects for bridgeable assays. Assays that are NotBridgeable or NotOverlapping retain their original OlinkIDs and NPX values.
- Replaces Panel by the concatenation of each product panel per assay. Assays that are NotBridgeable or NotOverlapping retain their original Panel value.
- Removes MedianCenteredNPX, QSNormalizedNPX, OlinkID_E3072 columns.

#' For reference median normalization:

- Adds non-overlapping assays from the dataset, but not from the reference medians, to the bridged file without adjustment.
- Removes external controls, except sample controls.

In all cases, normalization and formatting changes are applied to the NPX column. The contents of the Count and PCNormalizedNPX columns remain unchanged.

Usage

```
olink_normalization_format(df_norm, lst_check)
```

Arguments

df_norm	A "tibble" of Olink data in long format resulting from the olink_normalization function.
lst_check	Normalization input list checks generated by olink_norm_input_check.

Value

A "tibble" of Olink data in long format containing both input datasets with the bridged NPX quantifications, with the above modifications.

Author(s)

Danai G. Topouza Klev Diamanti

Examples

```
# bridge samples
bridge_samples <- intersect(
  x = unique(OlinkAnalyze:::data_ht_small$SampleID),
  y = unique(OlinkAnalyze:::data_3k_small$SampleID)
) |>
  (\(x) x[!grepl("CONTROL", x)])()

# check_npx
data_ht_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_ht_small
)

data_3k_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_3k_small
)

# run olink_normalization
df_norm <- OlinkAnalyze::olink_normalization(
  df1 = OlinkAnalyze:::data_ht_small,
  df2 = OlinkAnalyze:::data_3k_small,
  overlapping_samples_df1 = bridge_samples,
  df1_project_nr = "Explore HT",
  df2_project_nr = "Explore 3072",
  reference_project = "Explore HT",
  format = FALSE,
  df1_check_log = data_ht_small_check,
  df2_check_log = data_3k_small_check
)

# generate lst_check
lst_check <- OlinkAnalyze:::olink_norm_input_check(
```

```
  df1 = OlinkAnalyze:::data_ht_small,
  df1_check_log = data_ht_small_check,
  df2 = OlinkAnalyze:::data_3k_small,
  df2_check_log = data_3k_small_check,
  overlapping_samples_df1 = bridge_samples,
  overlapping_samples_df2 = NULL,
  df1_project_nr = "Explore HT",
  df2_project_nr = "Explore 3072",
  reference_project = "Explore HT",
  reference_medians = NULL
)

# format output
OlinkAnalyze:::olink_normalization_format(
  df_norm = df_norm,
  lst_check = lst_check
)
```

`olink_normalization_n` *Bridge and/or subset normalization of all proteins among multiple NPX projects.*

Description

This function normalizes pairs of NPX projects (data frames) using shared samples or subsets of samples.

This function is a wrapper of olink_normalization_bridge and olink_normalization_subset.

Usage

```
olink_normalization_n(norm_schema)
```

Arguments

`norm_schema`	A tibble with more than 1 rows and (strictly) the following columns: "order", "name", "data", "samples", "normalization_type", "normalize_to". See "Details" for the structure of the data frame (required)

Details

The input of this function is a tibble that contains all the necessary information to normalize multiple NPX projects. This tibble is called the normalization schema. The basic idea is that every row of the data frame is a separate project to be normalized. We assume that there is always one baseline project that does not normalize to any other. All other project normalize to one or more projects. The function handles projects that are normalized in a chain, for example:

- 1. project 2 normalizes to project 1, and project 3 normalizes to project 2.

- 2. project 2 normalizes to project 1, and project 3 normalizes to the combined data frame of projects 1 and 2 (that is already normalized).

The function can also handle a mixed schema of bridge and subset normalization.

Specifications of the normalization schema data frame:

- order: should strictly be a numeric or integer array with unique identifiers for each project. It is necessary that this array starts from 1 and that it contains no NAs.
- name: should strictly be a character array with unique identifiers for each project. Each entry should represent the name of the project located in the same row. No NAs are allowed.
- data: a named list of NPX data frames representing the projects to be normalized. Names of the items of the list should be identical to "names". No NAs are allowed.
- samples: a two-level nested named list of sample identifiers from each NPX project from "data". Names of the first level of the nested list should be identical to "names" and to the names of the list from "data". Projects that will be used only as reference should have their corresponding element in the list as NA, while all other projects should contain a named list of 2 arrays containing identifiers of samples to be used for the calculation of adjustment factor. The names of the two arrays should be DF1 and DF2 corresponding to the reference project and the project in the current row, respectively. For bridge normalization arrays should be of equal length and the index of each entry should correspond to the same sample. For subset normalization arrays do not need to be of equal length and the order the samples appear in does not matter. DF1 might contain sample identifiers from more than one project as long as the project in the current row is to be normalized to multiple other projects.
- normalization_type: a character array containing the flags "Bridge" or "Subset". Projects that will be used only as reference should have their corresponding element in the array as NA, while all other projects should contain a flag. For the time being the flag "Median" is not supported.
- normalize_to: a character array pointing to the project this project is to be normalized to. Elements of the array should be exclusively from the "order" column. Elements of the array may be comma-separated if the project is to be normalized to multiple projects.

Value

A "tibble" of NPX data in long format containing normalized NPX values, including adjustment factors and name of project.

Examples

```
#### Bridge normalization of two projects

# prepare datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
```

```
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find overlapping samples, but exclude Olink control
overlap_samples <- dplyr::intersect(x = unique(npx_df1[["SampleID"]]),
                                    y = unique(npx_df2[["SampleID"]]))
overlap_samples_list <- list("DF1" = overlap_samples,
                             "DF2" = overlap_samples)

# create tibble for input
norm_schema_bridge <- dplyr::tibble(
  order              = c(1L, 2L),
  name               = c("NPX_DF1", "NPX_DF2"),
  data               = list("NPX_DF1" = npx_df1,
                            "NPX_DF2" = npx_df2),
  samples            = list("NPX_DF1" = NA_character_,
                            "NPX_DF2" = overlap_samples_list),
  normalization_type = c(NA_character_, "Bridge"),
  normalize_to       = c(NA_character_, "1")
)

# normalize
OlinkAnalyze::olink_normalization_n(
  norm_schema = norm_schema_bridge
)

#### Subset normalization of two projects

# datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
```

```
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find a suitable subset of samples from both projects, but exclude Olink
# controls and samples that fail QC.
df1_samples <- npx_df1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique() |>
  sample(
    size = 16L,
    replace = FALSE
  )
df2_samples <- npx_df2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique() |>
  sample(
    size = 16L,
```

```
    replace = FALSE
  )

# create named list
subset_samples_list <- list("DF1" = df1_samples,
                            "DF2" = df2_samples)

# create tibble for input
norm_schema_subset <- dplyr::tibble(
  order              = c(1L, 2L),
  name               = c("NPX_DF1", "NPX_DF2"),
  data               = list("NPX_DF1" = npx_df1,
                            "NPX_DF2" = npx_df2),
  samples            = list("NPX_DF1" = NA_character_,
                            "NPX_DF2" = subset_samples_list),
  normalization_type = c(NA_character_, "Subset"),
  normalize_to       = c(NA_character_, "1")
)

# Normalize
OlinkAnalyze::olink_normalization_n(
  norm_schema = norm_schema_subset
)

#### Subset normalization  of two projects using all samples

# datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find a suitable subset of samples from both projects, but exclude Olink
# controls and samples that fail QC.
df1_samples_all <- npx_df1 |>
```

```
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()
df2_samples_all <- npx_df2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()

# create named list
subset_samples_all_list <- list("DF1" = df1_samples_all,
                                "DF2" = df2_samples_all)

# create tibble for input
norm_schema_subset_all <- dplyr::tibble(
  order              = c(1L, 2L),
  name               = c("NPX_DF1", "NPX_DF2"),
  data               = list("NPX_DF1" = npx_df1,
                            "NPX_DF2" = npx_df2),
  samples            = list("NPX_DF1" = NA_character_,
                            "NPX_DF2" = subset_samples_all_list),
 normalization_type = c(NA_character_, "Subset"),
 normalize_to       = c(NA_character_, "1")
)

# Normalize
OlinkAnalyze::olink_normalization_n(
  norm_schema = norm_schema_subset_all
```

```
)

#### Multi-project normalization using bridge and subset samples

## NPX data frames to bridge
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# manipulating the sample NPX datasets to create another two random ones
npx_df3 <- npx_data2 |>
  dplyr::mutate(
    SampleID = paste(.data[["SampleID"]], "_mod", sep = ""),
    PlateID = paste(.data[["PlateID"]], "_mod", sep = ""),
    NPX = sample(x = .data[["NPX"]], size = dplyr::n(), replace = FALSE)
  ) |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

npx_df4 <- npx_data1 |>
  dplyr::mutate(
    SampleID = paste(.data[["SampleID"]], "_mod2", sep = ""),
    PlateID = paste(.data[["PlateID"]], "_mod2", sep = ""),
    NPX = sample(x = .data[["NPX"]], size = dplyr::n(), replace = FALSE)
  ) |>
  dplyr::filter(
```

```
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

## samples to use for normalization
# Bridge samples with same identifiers between npx_df1 and npx_df2
overlap_samples <- dplyr::intersect(unique(npx_df1[["SampleID"]]),
                                    unique(npx_df2[["SampleID"]]))
overlap_samples_df1_df2 <- list("DF1" = overlap_samples,
                                "DF2" = overlap_samples)

# Bridge samples with different identifiers between npx_df2 and npx_df3
overlap_samples_df2_df3 <- list(
  "DF1" = sample(x = setdiff(x = unique(npx_df2[["SampleID"]]),
                             y = overlap_samples),
                 size = 10L,
                 replace = FALSE),
  "DF2" = sample(x = setdiff(x = unique(npx_df3[["SampleID"]]),
                             y = overlap_samples),
                 size = 10L,
                 replace = FALSE)
)

# Samples to use for intensity normalization between npx_df4 and the
# normalized dataset of npx_df1 and npx_df2
overlap_samples_df12_df4 <- list(
  "DF1" = sample(
    x = c(unique(npx_df1[["SampleID"]]), unique(npx_df2[["SampleID"]])),
    size = 100L,
    replace = FALSE
  ) |>
    unique(),
  "DF2" = sample(
    x = unique(npx_df4[["SampleID"]]),
    size = 40L,
    replace = FALSE
  )
)

# create tibble for input
norm_schema_n <- dplyr::tibble(
  order              = c(1L, 2L, 3L, 4L),
  name               = c("NPX_DF1", "NPX_DF2", "NPX_DF3", "NPX_DF4"),
  data               = list("NPX_DF1" = npx_df1,
                            "NPX_DF2" = npx_df2,
                            "NPX_DF3" = npx_df3,
                            "NPX_DF4" = npx_df4),
```

```
  samples            = list("NPX_DF1" = NA_character_,
                            "NPX_DF2" = overlap_samples_df1_df2,
                            "NPX_DF3" = overlap_samples_df2_df3,
                            "NPX_DF4" = overlap_samples_df12_df4),
  normalization_type = c(NA_character_, "Bridge", "Bridge", "Subset"),
  normalize_to       = c(NA_character_, "1", "2", "1,2")
)

OlinkAnalyze::olink_normalization_n(
  norm_schema = norm_schema_n
)
```

`olink_normalization_qs` *Quantile smoothing normalization of all proteins between two NPX projects.*

Description

This function uses bridge samples to map quantiles of the non-reference dataset to the ones of the reference dataset. Mapped quantiles are used to transform the quantifications of the the non-reference dataset to the reference.

Usage

```
olink_normalization_qs(
  lst_df,
  ref_cols,
  not_ref_cols,
  bridge_samples,
  prod_uniq
)
```

Arguments

`lst_df`	A named list of the 2 input datasets. First element should be the reference dataset from Olink Explore HT and the second element should originate from Olink Explore 3072. (required)
`ref_cols`	A named list with the column names to use. Exported from olink_norm_input_check. (required)
`not_ref_cols`	A named list with the column names from the non-reference dataset. Exported from olink_norm_input_check. (required)
`bridge_samples`	Character vector of samples to be used for the quantile mapping. (required)
`prod_uniq`	Name of products (not_ref, ref)

Details

In the case when a study is separated into multiple projects, an additional normalization step is needed to allow the data to be comparable across projects. Across different Olink products, some of the assays exist in corresponding but distinct NPX spaces. For those assays, the median of paired differences is insufficient for bridging as it only considers one anchor point (the median/50% quantile). Instead, quantile smoothing (QS) using multiple anchor points (5%, 10%, 25%, 50%, 75%, 90% and 95% quantiles) is favored to map the Explore 3072 data to the Explore HT distribution. The `olink_normalization_qs()` performs quantile smoothing bridging normalization between datasets from two Olink products (for example Olink Explore 3072 and Olink Explore HT) by performing the following steps:

- An empirical cumulative distribution function is used to map datapoints for the bridging samples from one product to the equivalent space in the other product.
- A spline regression model is constructed using unmapped and mapped data from one product, using anchor points from the quantiles defined above.
- The spline regression model is used to predict the normalized NPX values for all datapoints

More information on quantile smoothing and between product normalization can be found in the Bridging Olink Explore 3072 to Olink Explore HT tutorial.

Value

A "tibble" of Olink data in long format containing both input datasets with the quantile normalized quantifications.

Author(s)

Amrita Kar Marianne Sandin Masoumeh Sheikhi Klev Diamanti

Examples

```
# Bridge samples
bridge_samples <- intersect(
  x = unique(OlinkAnalyze:::data_ht_small$SampleID),
  y = unique(OlinkAnalyze:::data_3k_small$SampleID)
) |>
  (\(x) x[!grepl("CONTROL", x)])()

# check_npx
data_ht_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_ht_small
)

data_3k_small_check <- OlinkAnalyze::check_npx(
  df = OlinkAnalyze:::data_3k_small
)

# Run the internal function olink_norm_input_check
check_norm <- OlinkAnalyze:::olink_norm_input_check(
```

```
  df1 = OlinkAnalyze:::data_ht_small,
  df1_check_log = data_ht_small_check,
  df2 = OlinkAnalyze:::data_3k_small,
  df2_check_log = data_3k_small_check,
  overlapping_samples_df1 = bridge_samples,
  overlapping_samples_df2 = NULL,
  df1_project_nr = "P1",
  df2_project_nr = "P2",
  reference_project = "P1",
  reference_medians = NULL
)

# Named list of input datasets
lst_df <- list(
  check_norm$ref_df,
  check_norm$not_ref_df
)
names(lst_df) <- c(check_norm$ref_name, check_norm$not_ref_name)

ref_cols <- check_norm$ref_check_log$col_names
not_ref_cols <- check_norm$not_ref_check_log$col_names

qs_result <- OlinkAnalyze:::olink_normalization_qs(
  lst_df = lst_df,
  ref_cols = ref_cols,
  not_ref_cols = not_ref_cols,
  bridge_samples = bridge_samples,
  prod_uniq = c("E3072", "HT")
)
```

`olink_normalization_subset` *Subset normalization of all proteins between two NPX projects.*

Description

Normalizes two NPX projects (data frames) using all or a subset of samples.

This function is a wrapper of olink_normalization.

Usage

```
olink_normalization_subset(
  project_1_df,
  project_2_df,
  reference_samples,
  project_1_name = "P1",
  project_2_name = "P2",
```

```
  project_ref_name = "P1",
  format = FALSE,
  project_1_check_log = NULL,
  project_2_check_log = NULL
)
```

Arguments

project_1_df	Data frame of the first project (required).
project_2_df	Data frame of the second project (required).
reference_samples	Named list of 2 arrays containing SampleID of the subset of samples to be used for the calculation of median NPX within each project. The names of the two arrays should be DF1 and DF2 corresponding to projects 1 and 2, respectively. Arrays do not need to be of equal length and the order the samples appear in does not play any role. (required)
project_1_name	Name of the first project (default: P1).
project_2_name	Name of the second project (default: P2).
project_ref_name	Name of the project to be used as reference set. Needs to be one of the project_1_name or project_2_name. It marks the project to which the other project will be adjusted to (default: P1).
format	Boolean that controls whether the normalized dataset will be formatted for input to downstream analysis.
project_1_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using project_1_df. (default: NULL)
project_2_check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using project_2_df. (default: NULL)

Details

In subset normalization one of the projects is adjusted to another using a subset of all samples from each. Please note that the subsets of samples are not expected to be replicates of each other or to have the SampleID. Adjustment between the two projects is made using the assay-specific differences in median between the subsets of samples from the two projects. The two data frames are inputs project_1_df and project_2_df, the one being adjusted to is specified in the input project_ref_name and the shared samples are specified in reference_samples.

A special case of subset normalization is to use all samples (except control samples) from each project as a subset.

Value

A "tibble" of NPX data in long format containing normalized NPX values, including adjustment factors and name of project.

Examples

```
#### Subset normalization

# datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find a suitable subset of samples from both projects, but exclude Olink
# controls and samples that fail QC.
df1_samples <- npx_df1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique() |>
  sample(
    size = 16L,
    replace = FALSE
  )
df2_samples <- npx_df2 |>
  dplyr::filter(
```

```
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique() |>
  sample(
    size = 16L,
    replace = FALSE
  )

# create named list
subset_samples_list <- list("DF1" = df1_samples,
                            "DF2" = df2_samples)

# check npx
df1_check_log <- OlinkAnalyze::check_npx(df = npx_df1)
df2_check_log <- OlinkAnalyze::check_npx(df = npx_df2)

# Normalize
OlinkAnalyze::olink_normalization_subset(
  project_1_df = npx_df1,
  project_2_df = npx_df2,
  reference_samples = subset_samples_list,
  project_1_name = "P1",
  project_2_name = "P2",
  project_ref_name = "P1",
  project_1_check_log = df1_check_log,
  project_2_check_log = df2_check_log
)


#### Special case of subset normalization using all samples

# datasets
npx_df1 <- npx_data1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
```

```
    Normalization = "Intensity"
  )
npx_df2 <- npx_data2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::select(
    -dplyr::all_of("Project")
  ) |>
  dplyr::mutate(
    Normalization = "Intensity"
  )

# Find a suitable subset of samples from both projects, but exclude Olink
# controls and samples that fail QC.
df1_samples_all <- npx_df1 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()
df2_samples_all <- npx_df2 |>
  dplyr::filter(
    !stringr::str_detect(string = .data[["SampleID"]],
                         pattern = "CONTROL_")
  ) |>
  dplyr::group_by(
    dplyr::across(
      dplyr::all_of("SampleID")
    )
  ) |>
  dplyr::filter(
    all(.data[["QC_Warning"]] == "Pass")
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()

# create named list
subset_samples_all_list <- list("DF1" = df1_samples_all,
```

```
                                "DF2" = df2_samples_all)

# check npx
df1_check_log <- OlinkAnalyze::check_npx(df = npx_df1)
df2_check_log <- OlinkAnalyze::check_npx(df = npx_df2)

# Normalize
OlinkAnalyze::olink_normalization_subset(
  project_1_df = npx_df1,
  project_2_df = npx_df2,
  reference_samples = subset_samples_all_list,
  project_1_name = "P1",
  project_2_name = "P2",
  project_ref_name = "P1",
  project_1_check_log = df1_check_log,
  project_2_check_log = df2_check_log
)
```

`olink_norm_input_assay_overlap` *Check datasets and reference_medians for Olink identifiers not shared across datasets.*

Description

Check *datasets* and *reference_medians* for Olink identifiers not shared across datasets.

Usage

```
olink_norm_input_assay_overlap(lst_df, reference_medians, lst_cols, norm_mode)
```

Arguments

`lst_df`	Named list of datasets to be normalized.
`reference_medians`	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.
`lst_cols`	Named list of vectors with the required column names for each dataset in *lst_df*.
`norm_mode`	Character string indicating the type of normalization to be performed. Expecting one of bridge, subset, ref_median or norm_cross_product. # nolint: line_length_linter

Value

A named list containing *lst_df* and *reference_medians* with assays shared across all datasets.

Author(s)

Klev Diamanti

`olink_norm_input_check` *Check inputs of* `olink_normalization` *function.*

Description

This function is a wrapper of multiple help functions which check the inputs of the `olink_normalization` function.

Usage

```
olink_norm_input_check(
  df1,
  df1_check_log = NULL,
  df2,
  df2_check_log = NULL,
  overlapping_samples_df1,
  overlapping_samples_df2,
  df1_project_nr,
  df2_project_nr,
  reference_project,
  reference_medians
)
```

Arguments

`df1`	First dataset to be used in normalization (required).
`df1_check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df1`.
`df2`	Second dataset to be used in normalization.
`df2_check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df2`.
`overlapping_samples_df1`	Samples to be used for adjustment factor calculation in df1 (required).
`overlapping_samples_df2`	Samples to be used for adjustment factor calculation in df2.
`df1_project_nr`	Project name of first dataset (df1).
`df2_project_nr`	Project name of first dataset (df2).
`reference_project`	Project name of reference_project. Should be one of *df1_project_nr* or *df2_project_nr*. Indicates the project to which the other project is adjusted to.
`reference_medians`	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.

Details

The following checks are performed:

- olink_norm_input_validate:
  - Determines the normalization to be performed by intersecting inputs with internal global variable olink_norm_mode_combos.
  - Returns the type of normalization to be performed from olink_norm_modes.
  - Message with the normalization type.
  - Error message if input is invalid.
- olink_norm_input_class:
  - Checks if all inputs are of the expected class:
    - df1, df2 and reference_medians: tibble or R6 ArrowObject
    - overlapping_samples_df1, overlapping_samples_df2, df1_project_nr, df2_project_nr and reference_project: Character vector
  - Also checks the validity of names of project and reference project.
  - Error if invalid input classes are detected.
- olink_norm_input_check_df_cols:
  - Detects the column names of input datasets df1 and df2 to allow for alternative names.
  - Returns named list of column names to use downstream.
  - Warning if Normalization column missing from all datasets.
  - Warning if LOD is missing or if there are multiple LOD columns.
  - Error if required columns are missing.
  - Error if not all input datasets have or lack Normalization column.
  - Error if input datasets have been quantified with different methods.
- olink_norm_input_ref_medians:
  - Checks validity of dataset containing reference_medians.
  - Error if required columns are missing based on olink_norm_ref_median_cols.
  - Error if columns are not of the correct class bases on olink_norm_ref_median_cols.
  - Error if there duplicate assay identifiers.
- olink_norm_input_check_samples:
  - Check character vectors of reference sample identifiers for:
    - Being present in df1 and/or df2.
    - Duplicate identifiers.
- olink_norm_input_clean_assays:
  - Returns a named list with the updated df1, df2 and/or reference_medians.
  - Removes assays that are not of the format OID followed by 5 digits.
  - Removes assays that are marked with Normalization = EXCLUDED.
- olink_norm_input_assay_overlap:
  - Returns a named list with the updated df1, df2 and/or reference_medians.
  - Remove assays not shared between df1 and df2, or between df1 and reference_medians.
- olink_norm_input_norm_method:
  - Check if all assays in df1 and df2 have been originally normalized with the same method "Intensity" or "Plate control".
  - Warning is thrown if not.

Value

Named list of updated inputs to use for normalization:

- `df1`: dataset df1.
- `df2`: `NULL` if reference median normalization, or dataset df2.
- `overlapping_samples_df1`: character vector of reference samples from df1.
- `overlapping_samples_df2`: `NULL` if reference median normalization, or character vector of reference samples from df1.
- `df1_project_nr`: name of df1 project.
- `df2_project_nr`: `NULL` if reference median normalization, or name of df2 project.
- `reference_project`: `NULL` if reference median normalization, or name of reference project.
- `reference_medians`: `NULL` if bridge or subset normalization, or dataset with reference_medians.
- `df1_cols`: column names of df1 to use downstream.
- `df2_cols`: `NULL` if reference median normalization, or column names of df2 to use downstream.
- `norm_mode`: one of bridge, subset, ref_median, and norm_cross_product indicating the normalization to be performed.

Author(s)

Klev Diamanti

`olink_norm_input_check_df_cols` *Check columns of a list of datasets to be normalized.*

Description

This function takes as input a named list of datasets and checks if their columns allow the normalization to be performed. The input may contain "tibble", "ArrowTable" or a mixture of them.

Usage

```
olink_norm_input_check_df_cols(lst_df, lst_cols)
```

Arguments

`lst_df`	Named list of datasets to be normalized.
`lst_cols`	Named list of check logs returned by `check_npx`.

Value

`NULL` unless there is an error.

Author(s)

Klev Diamanti

Examples

```
# One dataset
lst_df_v1 <- list(
  "p1" = npx_data1
) |>
  lapply(function(l_df) {
    l_df |>
      dplyr::select(
        -dplyr::any_of(c("Normalization"))
      )
  })

lst_df_v1_check <- lst_df_v1 |>
  lapply(function(.x) {
    check_npx(df = .x) |>
      suppressWarnings() |>
      suppressMessages() |>
      (\(.) .$col_names)()
  })

OlinkAnalyze:::olink_norm_input_check_df_cols(
  lst_df = lst_df_v1,
  lst_cols = lst_df_v1_check
)

# Two datasets
lst_df_v2 <- list(
  "p1" = npx_data1,
  "p2" = npx_data2
) |>
  lapply(function(l_df) {
    l_df |>
      dplyr::select(
        -dplyr::any_of(c("Normalization"))
      )
  })

lst_df_v2_check <- lst_df_v2 |>
  lapply(function(.x) {
    check_npx(df = .x) |>
      suppressWarnings() |>
      suppressMessages() |>
      (\(.) .$col_names)()
  })

OlinkAnalyze:::olink_norm_input_check_df_cols(
  lst_df = lst_df_v2,
  lst_cols = lst_df_v2_check
```

```
)

# Multiple datasets
lst_df_v3 <- list(
  "p1" = npx_data1,
  "p2" = npx_data2,
  "p3" = npx_data1,
  "p4" = npx_data2
) |>
  lapply(function(l_df) {
    l_df |>
      dplyr::select(
        -dplyr::any_of(c("Normalization"))
      )
  })

lst_df_v3_check <- lst_df_v3 |>
  lapply(function(.x) {
    check_npx(df = .x) |>
      suppressWarnings() |>
      suppressMessages() |>
      (\(.) .$col_names)()
  })

OlinkAnalyze:::olink_norm_input_check_df_cols(
  lst_df = lst_df_v3,
  lst_cols = lst_df_v3_check
)
```

`olink_norm_input_check_samples` *Check reference samples to be used for normalization*

Description

This function takes as input a two named lists of character vectors with matching names and checks the validity of the reference samples. In case of 1 set of df samples, then all checks are skipped as reference median normalization is to be performed.

Usage

```
olink_norm_input_check_samples(
  lst_df_samples,
  lst_ref_samples,
  lst_dup_samples,
  norm_mode
)
```

Arguments

lst_df_samples	Named list of all sample identifiers from datasets to be normalized.
lst_ref_samples	Named list of reference sample identifiers to be used for normalization.
lst_dup_samples	Named list of duplicate sample identifiers identified by check_npx.
norm_mode	Character string indicating the type of normalization to be performed. Expecting one of bridge, subset, ref_median or norm_cross_product. # nolint: line_length_linter

Value

NULL if no warning or error.

Author(s)

Klev Diamanti

Examples

```
# Reference median normalization
OlinkAnalyze:::olink_norm_input_check_samples(
  lst_df_samples = list(
    "p1" = unique(npx_data1$SampleID)
  ),
  lst_ref_samples = list(
    "p1" = npx_data1 |>
      dplyr::filter(
        !grepl(pattern = "CONTROL_SAMPLE",
        x = .data[["SampleID"]],
        fixed = TRUE)
      ) |>
      dplyr::pull(.data[["SampleID"]]) |>
      unique() |>
      sort() |>
      head(n = 6L)
  ),
  lst_dup_samples = list(
    "p1" = character(0L)
  ),
  norm_mode = "ref_median"
)

# Bridge normalization
ref_samples_bridge <- intersect(x = npx_data1$SampleID,
                                y = npx_data2$SampleID) |>
  (\(x) x[!grepl(pattern = "CONTROL_SAMPLE", x = x, fixed = TRUE)])()

OlinkAnalyze:::olink_norm_input_check_samples(
  lst_df_samples = list(
```

```
    "p1" = unique(npx_data1$SampleID),
    "p2" = unique(npx_data2$SampleID)
  ),
  lst_ref_samples = list(
    "p1" = ref_samples_bridge,
    "p2" = ref_samples_bridge
  ),
  lst_dup_samples = list(
    "p1" = character(0L),
    "p2" = character(0L)
  ),
  norm_mode = "bridge"
)

# Subset normalization
ref_samples_subset_1 <- npx_data1 |>
  dplyr::filter(
    !grepl(pattern = "CONTROL_SAMPLE",
           x = .data[["SampleID"]],
           fixed = TRUE)
    & .data[["QC_Warning"]] == "Pass"
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()
ref_samples_subset_2 <- npx_data2 |>
  dplyr::filter(
    !grepl(pattern = "CONTROL_SAMPLE",
           x = .data[["SampleID"]],
           fixed = TRUE)
    & .data[["QC_Warning"]] == "Pass"
  ) |>
  dplyr::pull(
    .data[["SampleID"]]
  ) |>
  unique()

OlinkAnalyze:::olink_norm_input_check_samples(
  lst_df_samples = list(
    "p1" = unique(npx_data1$SampleID),
    "p2" = unique(npx_data2$SampleID)
  ),
  lst_ref_samples = list(
    "p1" = ref_samples_subset_1,
    "p2" = ref_samples_subset_2
  ),
  lst_dup_samples = list(
    "p1" = character(0L),
    "p2" = character(0L)
  ),
  norm_mode = "subset"
)
```

olink_norm_input_class *Check classes of input in olink_normalization function*

Description

Check if *df1* , *df2* and/or *reference_medians* are tibble or ArrowDataset datasets; if *overlapping_samples_df1* and/or *overlapping_samples_df2* are character vectors; and if *df1_project_nr*, *df2_project_nr* and/or *reference_project* are scalar character vectors.

Usage

```
olink_norm_input_class(
  df1,
  df2,
  overlapping_samples_df1,
  overlapping_samples_df2,
  df1_project_nr,
  df2_project_nr,
  reference_project,
  reference_medians,
  norm_mode
)
```

Arguments

df1	First dataset to be used in normalization (required).
df2	Second dataset to be used in normalization.
overlapping_samples_df1	Samples to be used for adjustment factor calculation in df1 (required).
overlapping_samples_df2	Samples to be used for adjustment factor calculation in df2.
df1_project_nr	Project name of first dataset (df1).
df2_project_nr	Project name of first dataset (df2).
reference_project	Project name of reference_project. Should be one of *df1_project_nr* or *df2_project_nr*. Indicates the project to which the other project is adjusted to.
reference_medians	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.
norm_mode	Scalar character from *olink_norm_modes* with the normalization to be performed. Output from olink_norm_input_validate.

Value

NULL unless there is an error.

Author(s)

Klev Diamanti

---

olink_norm_input_clean_assays *Check datasets and reference_medians for unexpected Olink identifiers or excluded assays*

Description

Check *datasets* and *reference_medians* for unexpected Olink identifiers or excluded assays

Usage

```
olink_norm_input_clean_assays(lst_df, reference_medians, lst_cols, norm_mode)
```

Arguments

lst_df	Named list of datasets to be normalized.
reference_medians	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.
lst_cols	Named list of vectors with the required column names for each dataset in *lst_df*.
norm_mode	Character string indicating the type of normalization to be performed. Expecting one of bridge, subset, ref_median or norm_cross_product. # nolint: line_length_linter

Value

A named list containing *lst_df* and *reference_medians* stripped from unexpected Olink identifiers or excluded assays

Author(s)

Klev Diamanti

`olink_norm_input_cross_product` *Check if bridge or cross-platform normalization*

Description

A function to check whether we are to perform simple bridge normalization, or cross-platform (Olink Explore 3072 - Olink Explore HT/Olink Reveal) normalization.

The function uses the internal dataset *eHT_e3072_mapping* to determine the product source of each dataset. If both datasets originate from the same Olink product, then it will return bridge. If the datasets to be normalized originate from Olink Explore HT and Olink Explore 3072 or Olink Reveal and Olink Explore 3072, it will return norm_cross_product. In any other case an error is thrown.

Usage

```
olink_norm_input_cross_product(
  lst_df,
  lst_cols,
  reference_project,
  product_ids,
  ref_ids
)
```

Arguments

`lst_df`	Named list of datasets to be normalized.
`lst_cols`	Named list of vectors with the required column names for each dataset in *lst_df*.
`reference_project`	Project name of reference_project. Should be one of *df1_project_nr* or *df2_project_nr*. Indicates the project to which the other project is adjusted to.
`product_ids`	Named character vector with the Olink product name that each input dataset matches to.
`ref_ids`	Named character vector with *df1_project_nr* and *df2_project_nr* marked as "ref" and "not_ref".

Value

Character string indicating the type of normalization to be performed. One of bridge, subset, ref_median or norm_cross_product. # nolint: line_length_linter And the updated list of datasets in case of cross-platform normalization.

Author(s)

Klev Diamanti

olink_norm_input_norm_method

Check datasets and reference_medians for Olink identifiers not shared across datasets.

Description

Check *datasets* and *reference_medians* for Olink identifiers not shared across datasets.

Usage

```
olink_norm_input_norm_method(lst_df, lst_cols)
```

Arguments

lst_df	Named list of datasets to be normalized.
lst_cols	Named list of vectors with the required column names for each dataset in *lst_df*.

Value

NULL if all assays are normalized with the same approach.

Author(s)

Klev Diamanti Kathleen Nevola

olink_norm_input_ref_medians

Check datasets of reference_medians

Description

Check datasets of *reference_medians*

Usage

```
olink_norm_input_ref_medians(reference_medians)
```

Arguments

reference_medians	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.

Value

NULL otherwise error.

Author(s)

Klev Diamanti

`olink_norm_input_validate` *Validate inputs of normalization function*

Description

This function takes as input some of the inputs of the Olink normalization function and checks the validity of the input.

Usage

```
olink_norm_input_validate(
  df1,
  df2,
  overlapping_samples_df1,
  overlapping_samples_df2,
  reference_medians
)
```

Arguments

`df1`	First dataset to be used in normalization (required).
`df2`	Second dataset to be used in normalization.
`overlapping_samples_df1`	Samples to be used for adjustment factor calculation in df1 (required).
`overlapping_samples_df2`	Samples to be used for adjustment factor calculation in df2.
`reference_medians`	Dataset with columns "OlinkID" and "Reference_NPX". Used for reference median normalization.

Details

Depending on the input the function will return:

- **Error**: if the required components are lacking from the input or if the normalization cannot be performed.
- **Warning**: if the normalization can be determined but extra inputs are provided. This will be followed by a message and the type of normalization to be performed.
- **Message**: Information about the type of normalization to be performed.

Note that input are passed directly from the main `olink_normalization` function.

Value

Scalar character from *olink_norm_modes* if normalization can be determined from the input, otherwise see details.

Author(s)

Klev Diamanti

`olink_norm_product_id` *Identify names of product for each project*

Description

Identify names of product for each project

Usage

```
olink_norm_product_id(lst_df, lst_cols)
```

Arguments

`lst_df`	Named list of datasets to be normalized.
`lst_cols`	Named list of vectors with the required column names for each dataset in *lst_df*.

Value

Named character vector with the Olink product name that each input dataset matches to.

Author(s)

Kathy Nevola Klev Diamanti

`olink_norm_reference_id`
Identify reference project.

Description

Identify reference project.

Usage

```
olink_norm_reference_id(lst_product, reference_project)
```

Arguments

`lst_product`	Named character vector with the Olink product name that each input dataset matches to.
`reference_project`	Project name of reference_project. Should be one of *df1_project_nr* or *df2_project_nr*. Indicates the project to which the other project is adjusted to.

Value

Named character vector with *df1_project_nr* and *df2_project_nr* marked as "ref" and "not_ref".

Author(s)

Kathy Nevola Klev Diamanti

`olink_one_non_parametric` *Function which performs a Kruskal-Wallis Test or Friedman Test per protein*

Description

Performs an Kruskal-Wallis Test for each assay (by OlinkID) in every panel using stats::kruskal.test. Performs an Friedman Test for each assay (by OlinkID) in every panel using rstatix::friedman_test. The function handles factor variable.

Samples that have no variable information or missing factor levels are automatically removed from the analysis (specified in a message if verbose = TRUE). Character columns in the input dataframe are automatically converted to factors (specified in a message if verbose = T). Numerical variables are not converted to factors. If a numerical variable is to be used as a factor, this conversion needs to be done on the dataframe before the function call.

Inference is specified in a message if verbose = TRUE.
The formula notation of the final model is specified in a message if verbose = TRUE.

Adjusted p-values are calculated by stats::p.adjust according to the Benjamini & Hochberg (1995) method (“fdr”). The threshold is determined by logic evaluation of Adjusted_pval < 0.05.

Usage

```
olink_one_non_parametric(
  df,
  check_log = NULL,
  variable,
  dependence = FALSE,
  subject = NULL,
  verbose = TRUE
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
variable	Single character value.
dependence	Boolean. Default: FALSE. When the groups are independent, the kruskal-Wallis will run, when the groups are dependent, the Friedman test will run.
subject	Group information for the repeated measurement. If (dependence = TRUE), this parameter need to be specified.
verbose	Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Value

A tibble containing the Kruskal-Wallis Test or Friedman Test results for every protein.

Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- df: "numeric" degrees of freedom
- method: "character" which method was used
- statistic: "named numeric" the value of the test statistic with a name describing it
- p.value: "numeric" p-value for the test
- Adjusted_pval: "numeric" adjusted p-value for the test (Benjamini&Hochberg)
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("broom", "rstatix"))) {

 check_log <- check_npx(df = npx_data1)

 # One-way Kruskal-Wallis Test
 kruskal_results <- OlinkAnalyze::olink_one_non_parametric(
    df = npx_data1,
    check_log = check_log,
    variable = "Site"
  )

  # Friedman Test
  friedman_results <- OlinkAnalyze::olink_one_non_parametric(
```

```
    df = npx_data1,
    check_log = check_log,
    variable = "Time",
    subject = "Subject",
    dependence = TRUE
  )
}
```

`olink_one_non_parametric_posthoc` *Function which performs posthoc test per protein for the results from Friedman or Kruskal-Wallis Test.*

Description

Performs a posthoc test using rstatix::wilcox_test or FSA::dunnTest with Benjamini-Hochberg p-value adjustment per assay (by OlinkID) for each panel at confidence level 0.95. See `olink_one_non_parametric` for details of input notation.

The function handles both factor and numerical variables.

Usage

```
olink_one_non_parametric_posthoc(
  df,
  check_log = NULL,
  olinkid_list = NULL,
  variable,
  test = "kruskal",
  subject = "Subject",
  verbose = TRUE
)
```

Arguments

`df`	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
`olinkid_list`	Character vector of OlinkID's on which to perform post hoc analysis. If not specified, all assays in df are used.
`variable`	Single character value or character array.
`test`	Single character value indicates running the post hoc test for friedman or kruskal.
`subject`	Group information for the repeated measurement. If (dependence = TRUE), this parameter need to be specified.

verbose — Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Value

Tibble of posthoc tests for specified effect, arranged by ascending adjusted p-values.

Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- contrast: "character" the groups that were compared
- estimate: "numeric" the value of the test statistic with a name describing it
- Adjusted_pval: "numeric" adjusted p-value for the test
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("FSA", "broom", "rstatix"))) {

  check_log <- check_npx(df = npx_data1)

  # One-way Kruskal-Wallis Test
  kruskal_results <- OlinkAnalyze::olink_one_non_parametric(
    df = npx_data1,
    check_log = check_log,
    variable = "Site"
  )

  # Friedman Test
  friedman_results <- OlinkAnalyze::olink_one_non_parametric(
    df = npx_data1,
    check_log = check_log,
    variable = "Time",
    subject = "Subject",
    dependence = TRUE
  )

  # Posthoc test for the results from Friedman Test
  friedman_posthoc_results <- OlinkAnalyze::olink_one_non_parametric_posthoc(
    df = npx_data1,
    check_log = check_log,
    variable = "Time",
    test = "friedman",
    olinkid_list = friedman_results |>
      dplyr::filter(.data[["Threshold"]] == "Significant") |>
      dplyr::select(
```

```
        dplyr::all_of("OlinkID")
      ) |>
      dplyr::distinct() |>
      dplyr::pull()
  )
}
```

---

olink_ordinal_regression

Function that performs a two-way ordinal analysis.

Description

Function that performs a two-way ordinal analysis of variance can address an experimental design with two independent variables, each of which is a factor variable. The main effect of each independent variable can be tested, as well as the effect of the interaction of the two factors.

Usage

```
olink_ordinal_regression(
  df,
  variable,
  covariates = NULL,
  return.covariates = FALSE,
  check_log = NULL,
  verbose = TRUE
)

olink_ordinalRegression(
  df,
  variable,
  covariates = NULL,
  return.covariates = FALSE,
  check_log = NULL,
  verbose = TRUE
)
```

Arguments

df	NPX or Quantified_value data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
variable	Single character value or character array. Variable(s) to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':'/'*' notation.

`covariates` Single character value or character array. Default: NULL. Covariates to include. Takes ':'/'*' notation. Crossed analysis will not be inferred from main effects.

`return.covariates` Logical. Default: False. Returns F-test results for the covariates. Note: Adjusted p-values will be NA for the covariates.

`check_log` A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.

`verbose` Logical. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Details

Performs an ANOVA F-test for each assay (by OlinkID) in every panel using stats::Anova and Type III sum of squares. Dependent variable will be treated as ordered factor. The function handles only factor and/or covariates.

Samples that have no variable information or missing factor levels are automatically removed from the analysis (specified in a message if verbose = T). Character columns in the input dataframe are automatically converted to factors (specified in a message if verbose = T). Crossed analysis, i.e. A*B formula notation, is inferred from the variable argument in the following cases:

- c('A','B')
- c('A: B')
- c('A: B', 'B') or c('A: B', 'A')

Inference and the formula notation of the final model are specified in a message if verbose = T.

Adjusted p-values are calculated by stats::p.adjust according to the Benjamini & Hochberg (1995) method ("fdr"). The threshold is determined by logic evaluation of Adjusted_pval < 0.05. Covariates are not included in the p-value adjustment.

Value

A tibble containing the ANOVA results for every protein. The tibble is arranged by ascending p-values. Columns include:#'

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- statistic: "numeric" value of the statistic
- p.value: "numeric" nominal p-value
- Adjusted_pval: "numeric" adjusted p-value for the test
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("ordinal", "broom"))) {
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
    !grepl(
      pattern = "control",
      x = .data[["SampleID"]],
      ignore.case = TRUE
    )
  )
  check_log <- OlinkAnalyze::check_npx(df = npx_df)

  # Two-way Ordinal Regression with CLM.
  # Results in model NPX~Treatment+Time+Treatment:Time.
  ordinalRegression_results <- OlinkAnalyze::olink_ordinal_regression(
    df = npx_df,
    variable = "Treatment:Time"
  )
}
```

`olink_ordinal_regression_posthoc`
Function which performs an posthoc test per protein.

Description

Performs a post hoc ANOVA test using emmeans::emmeans with Tukey p-value adjustment per assay (by OlinkID) for each panel at confidence level 0.95. See `olink_anova` for details of input notation.

The function handles both factor and numerical variables and/or covariates. The posthoc test for a numerical variable compares the difference in means of the ordinal outcome variable (default: NPX) for 1 standard deviation difference in the numerical variable, e.g. mean ordinal NPX at mean(numerical variable) versus mean NPX at mean(numerical variable) + 1*SD(numerical variable).

Usage

```
olink_ordinal_regression_posthoc(
  df,
  olinkid_list = NULL,
  variable,
  covariates = NULL,
  effect,
  effect_formula,
  mean_return = FALSE,
  post_hoc_padjust_method = "tukey",
```

```
  check_log = NULL,
  verbose = TRUE
)

olink_ordinalRegression_posthoc(
  df,
  olinkid_list = NULL,
  variable,
  covariates = NULL,
  effect,
  effect_formula,
  mean_return = FALSE,
  post_hoc_padjust_method = "tukey",
  check_log = NULL,
  verbose = TRUE
)
```

Arguments

df	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, Panel and a factor with at least 3 levels.
olinkid_list	Character vector of OlinkID's on which to perform post hoc analysis. If not specified, all assays in df are used.
variable	Single character value or character array. Variable(s) to test. If length > 1, the included variable names will be used in crossed analyses. Also takes ':' notation.
covariates	Single character value or character array. Default: NULL. Covariates to include. Takes ':'/'*' notation. Crossed analysis will not be inferred from main effects.
effect	Term on which to perform post-hoc. Character vector. Must be subset of or identical to variable.
effect_formula	(optional) A character vector specifying the names of the predictors over which estimated marginal means are desired as defined in the emmeans package. May also be a formula. If provided, this will override the effect argument. See ?emmeans::emmeans() for more information.
mean_return	Boolean. If true, returns the mean of each factor level rather than the difference in means (default). Note that no p-value is returned for mean_return = TRUE and no adjustment is performed.
post_hoc_padjust_method	P-value adjustment method to use for post-hoc comparisons within an assay. Options include tukey, sidak, bonferroni and none.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
verbose	Boolean. Default: True. If information about removed samples, factor conversion and final model formula is to be printed to the console.

Value

Tibble of posthoc tests for specified effect, arranged by ascending adjusted p-values. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- term: "character" term in model
- contrast: "character" the groups that were compared
- estimate: "numeric" difference in mean of the ordinal NPX between groups
- Adjusted_pval: "numeric" adjusted p-value for the test
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("ordinal", "emmeans"))) {
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
    !grepl(
      pattern = "control",
      x = .data[["SampleID"]],
      ignore.case = TRUE
    )
  )
  check_log <- OlinkAnalyze::check_npx(df = npx_df)

  # Two-way Ordinal Regression with CLM.
  # Results in model NPX~Treatment+Time+Treatment:Time.
  ordinalRegression_results <- OlinkAnalyze::olink_ordinal_regression(
    df = npx_df,
    variable = "Treatment:Time"
  )

  significant_assays <- ordinalRegression_results |>
    dplyr::filter(
      .data[["Threshold"]] == "Significant"
      & .data[["term"]] == "Time"
    ) |>
    dplyr::pull(
      .data[["OlinkID"]]
    ) |>
    unique()

  # Posthoc test
  ordRegr_results_posthoc <- OlinkAnalyze::olink_ordinal_regression_posthoc(
    df = npx_df,
    variable = c("Treatment:Time"),
    olinkid_list = significant_assays,
```

```
    effect = "Time",
    check_log = check_log
  )
}
```

`olink_osi_dist_plot` *OSI distribution plot*

Description

Generates a density plot showing the distribution of the selected OSI score among dataset samples using ggplot2. OSI score can be one of "OSITimeToCentrifugation", "OSIPreparationTemperature", or "OSISummary". Olink external controls are excluded from this visualization.

Usage

```
olink_osi_dist_plot(df, check_log = NULL, osi_score = NULL)
```

Arguments

df	data frame with OSI data present
check_log	check log from check NPX
osi_score	OSI column to graph, one of OSISummary, OSITimeToCentrifugation, or OSIPreparationTemperature

Value

distribution plot (histogram overlayed with density plot) of osi values for corresponding osi_score column

Examples

```
# Creating fake OSI data from Site data
npx_df <- OlinkAnalyze::npx_data1 |>
  dplyr::filter(
    !grepl(pattern = "control",
    x = .data[["SampleID"]],
    ignore.case = TRUE)
  ) |>
  dplyr::mutate(
    OSISummary = as.numeric(as.factor(.data[["Site"]])),
    OSISummary = .data[["OSISummary"]] - min(.data[["OSISummary"]],
                                             na.rm = TRUE),
    OSISummary = .data[["OSISummary"]] / max(.data[["OSISummary"]],
                                             na.rm = TRUE)
  )
```

```
check_log <- OlinkAnalyze::check_npx(
  df = npx_df
)

# Generate figure
OlinkAnalyze::olink_osi_dist_plot(
  df = npx_df,
  check_log = check_log,
  osi_score = "OSISummary"
)
```

`olink_pal` *Olink color panel for plotting*

Description

Olink color panel for plotting

Usage

```
olink_pal(alpha = 1, coloroption = NULL)
```

Arguments

`alpha`	transparency (optional)
`coloroption`	string, one or more of the following: c("red", "orange", "yellow", "green", "teal", "turqoise", "lightblue", "darkblue", "purple", "pink")

Value

A character vector of palette hex codes for colors.

Examples

```
if (rlang::is_installed(pkg = c("scales"))) {
  # Color matrices
  scales::show_col(
    colours = OlinkAnalyze::olink_pal()(10L),
    labels = FALSE
  )
  scales::show_col(
    colours = OlinkAnalyze::olink_pal(
      coloroption = c("lightblue", "green")
    )(2L),
    labels = FALSE
  )
```

```
  # Contour plot
  filled.contour(
    x = datasets::volcano,
    color.palette = OlinkAnalyze::olink_pal(),
    asp = 1
  )
  filled.contour(
    x = datasets::volcano,
    color.palette = scales::hue_pal(),
    asp = 1
  )
}
```

`olink_pathway_enrichment` *Performs pathway enrichment using over-representation analysis (ORA) or gene set enrichment analysis (GSEA)*

Description

This function performs enrichment analysis based on statistical test results and full data using `clusterProfiler`'s functions `gsea` and `enrich` for MSigDB.

Usage

```
olink_pathway_enrichment(
  df,
  test_results,
  check_log = NULL,
  method = "GSEA",
  ontology = "MSigDb",
  organism = "human",
  pvalue_cutoff = 0.05,
  estimate_cutoff = 0
)
```

Arguments

`df`	NPX data frame in long format with at least protein name ("Assay"), "OlinkID", "UniProt", "SampleID", QC warning ("QC_Warning" or "SampleQC"), quantification column ("NPX", "Ct" or "Quantified_value"), and one or more columns representing limit of detection ("LOD", "PlateLOD" or "MaxLOD").
`test_results`	a data frame of statistical test results including the columns "Adjusted_pval" and "estimate".
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.

method	One of "GSEA" (default) or "ORA".
ontology	One of "MSigDb" (default), "MSigDb_com", "KEGG", "GO", and "Reactome". "MSigDb" contains "C2" and "C5" gene sets which encompass "KEGG", "GO", and "Reactome". "MSigDb_com" consists of "C2" and "C5" gene sets without "KEGG", as the latter not permitted for commercial use.
organism	One of "human" (default) or "mouse".
pvalue_cutoff	(numeric) maximum adjusted p-value cutoff for ORA filtering of foreground set (default = 0.05). This argument is not used for GSEA.
estimate_cutoff	(numeric) minimum estimate cutoff for ORA filtering of foreground set (default = 0). This argument is not used for GSEA.

Details

MSigDB is subset if the ontology argument is "KEGG", "GO", or "Reactome". The argument `test_results` must contain estimates for all assays, otherwise an error will be thrown. Results from a post-hoc statistical test can be used as argument for `test_results`, but the user needs to select and filter one contrast to improve interpretability of the results. Alternative statistical results can be used as input as long as they include the columns "OlinkID", "Assay", and "estimate". A column named "Adjusted_pval" is also required for ORA. Any statistical result that contains exactly one estimate per protein will work as long as the estimates are comparable to each other.

The R library `clusterProfiler` is originally developed by Guangchuang Yu at the School of Basic Medical Sciences at Southern Medical University.

NB: We strongly recommend to set a seed prior to running this function to ensure reproducibility of the results.

An important note regarding Pathway Enrichment with Olink Data

It is important to note that sometimes the proteins that are assayed in Olink Panels are related to specific biological areas and therefore do not represent an unbiased overview of the proteome as a whole, which is an assumption for pathway enrichment. Pathways can only interpreted based on the background/context they came from. For this reason, an estimate for all assays measured must be provided. Furthermore, certain pathways cannot come up based on Olink's coverage in this area. Additionally, if only the Inflammation panel was run, then the available pathways would be given based on a background of proteins related to inflammation. Both ORA and GSEA can provide mechanistic and disease related insight and are best to use when trying to uncover pathways/annotations of interest. It is recommended to only use pathway enrichment for hypothesis generating data, which is better suited for data originating from Olink's NGS platforms "Explore 3072", "Explore HT", and "Reveal" or on multiple Target 96 panels from Olink's qPCR platform. For smaller lists of proteins it may be more informative to use biological annotation in directed research, to discover which significant assays are related to keywords of interest.

Value

A data frame of enrichment results.

Columns for ORA include:

- ID: Pathway ID from MSigDB.

- Description: Description of Pathway from MSigDB.
- GeneRatio: Ratio of input proteins that are annotated in a term.
- BgRatio: Ratio of all genes that are annotated in this term.
- pvalue: P-value of enrichment.
- p.adjust: Benjamini-Hochberg adjusted p-value.
- qvalue: False discovery rate (FDR), the estimated probability that the normalized enrichment score represents a false positive finding.
- geneID: List of input proteins (Gene Symbols) annotated in a term, delimited by "/".
- Count: Number of input proteins that are annotated in a term.

Columns for GSEA:

- ID: Pathway ID from MSigDB.
- Description: Description of Pathway from MSigDB.
- setSize: Ratio of input proteins that are annotated in a term.
- enrichmentScore: Enrichment score (ES), degree to which a gene set is over-represented at the top or bottom of the ranked list of genes.
- NES: Normalized Enrichment Score (NES), normalized to account for differences in gene set size and in correlations between gene sets and expression data sets. NES can be used to compare analysis results across gene sets.
- pvalue: P-value of enrichment.
- p.adjust: Benjamini-Hochberg adjusted p-value.
- qvalue: False discovery rate (FDR), the estimated probability that the normalized enrichment score represents a false positive finding.
- rank: The position in the ranked list where the maximum enrichment score occurred.
- leading_edge: Contains tags, list, and signal. Tags provide an indication of the percentage of genes contributing to the ES. List gives an indication of where in the list the ES is obtained. Signal represents the enrichment signal strength and combines the tag and list.
- core_enrichment: List of input proteins (Gene Symbols) annotated in a term, delimited by "/".

Author(s)

Kathleen Nevola Klev Diamanti

References

Wu, T. et al. (2021). *clusterProfiler 4.0: A universal enrichment tool for interpreting omics data*. The Innovation, 2(3):100141. doi: 10.1016/j.xinn.2021.100141.

Examples

```
if (rlang::is_installed(pkg = c("msigdbr", "clusterProfiler"))) {
  npx_df <- npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  check_log <- check_npx(df = npx_df)

  ttest_results <- OlinkAnalyze::olink_ttest(
    df = npx_df,
    variable = "Treatment",
    alternative = "two.sided",
    check_log = check_log
  )

  # GSEA
  gsea_results <- OlinkAnalyze::olink_pathway_enrichment(
    df = npx_df,
    test_results = ttest_results,
    check_log = check_log
  )

  # ORA
  ora_results <- OlinkAnalyze::olink_pathway_enrichment(
    df = npx_df,
    test_results = ttest_results,
    check_log = check_log,
    method = "ORA"
  )
}
```

---

olink_pathway_heatmap *Creates a heatmap of proteins related to pathways using enrichment results from* olink_pathway_enrichment.

Description

Creates a heatmap of proteins related to pathways using enrichment results from olink_pathway_enrichment.

Usage

```
olink_pathway_heatmap(
```

```
  enrich_results,
  test_results,
  method = "GSEA",
  keyword = NULL,
  number_of_terms = 20L
)
```

Arguments

enrich_results	data frame of enrichment results from olink_pathway_enrichment
test_results	a data frame of statistical test results including the columns "Adjusted_pval" and "estimate".
method	One of "GSEA" (default) or "ORA".
keyword	(optional) keyword to filter enrichment results on. If not specified, displays top terms.
number_of_terms	number of terms to display (default is 20).

Value

A heatmap as a ggplot object.

Examples

```
if (rlang::is_installed(pkg = c("msigdbr", "clusterProfiler"))) {

  # Run olink_ttest or other stats test (see documentation )
  npx_df <- npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  check_log <- check_npx(df = npx_df)

  ttest_results <- OlinkAnalyze::olink_ttest(
    df = npx_df,
    variable = "Treatment",
    alternative = "two.sided",
    check_log = check_log
  )

  # Run olink_pathway_enrichment (see documentation)

  # GSEA
  gsea_results <- OlinkAnalyze::olink_pathway_enrichment(
    df = npx_df,
```

```
  test_results = ttest_results,
  check_log = check_log
)

# ORA
ora_results <- OlinkAnalyze::olink_pathway_enrichment(
  df = npx_df,
  test_results = ttest_results,
  check_log = check_log,
  method = "ORA"
)

# Plot

OlinkAnalyze::olink_pathway_heatmap(
  enrich_results = gsea_results,
  test_results = ttest_results
)

OlinkAnalyze::olink_pathway_heatmap(
  enrich_results = ora_results,
  test_results = ttest_results,
  method = "ORA",
  keyword = "cell"
)
}
```

`olink_pathway_visualization` *Creates bargraph of top/selected enrichment terms from GSEA or ORA results from* `olink_pathway_enrichment`

Description

Pathways are ordered by increasing p-value (unadjusted)

Usage

```
olink_pathway_visualization(
  enrich_results,
  method = "GSEA",
  keyword = NULL,
  number_of_terms = 20L
)
```

Arguments

enrich_results	data frame of enrichment results from olink_pathway_enrichment
method	One of "GSEA" (default) or "ORA".
keyword	(optional) keyword to filter enrichment results on. If not specified, displays top terms.
number_of_terms	number of terms to display (default is 20).

Value

A bargraph as a ggplot object.

Examples

```
if (rlang::is_installed(pkg = c("msigdbr", "clusterProfiler"))) {

  # Run olink_ttest or other stats test (see documentation )
  npx_df <- npx_data1 |>
    dplyr::filter(
      !grepl(
        pattern = "control",
        x = .data[["SampleID"]],
        ignore.case = TRUE
      )
    )

  check_log <- check_npx(df = npx_df)

  ttest_results <- OlinkAnalyze::olink_ttest(
    df = npx_df,
    variable = "Treatment",
    alternative = "two.sided",
    check_log = check_log
  )

  # Run olink_pathway_enrichment (see documentation)

  # GSEA
  gsea_results <- OlinkAnalyze::olink_pathway_enrichment(
    df = npx_df,
    test_results = ttest_results,
    check_log = check_log
  )

  # ORA
  ora_results <- OlinkAnalyze::olink_pathway_enrichment(
    df = npx_df,
    test_results = ttest_results,
    check_log = check_log,
    method = "ORA"
  )
```

```
  # Plot

  OlinkAnalyze::olink_pathway_visualization(
    enrich_results = gsea_results
  )

  OlinkAnalyze::olink_pathway_visualization(
    enrich_results = gsea_results,
    keyword = "immune"
  )

  OlinkAnalyze::olink_pathway_visualization(
    enrich_results = ora_results,
    method = "ORA",
    number_of_terms = 15L
  )
}
```

`olink_pca_plot` *Function to plot a PCA of the data*

Description

Generates a PCA projection of all samples from NPX data along two principal components (default PC2 vs. PC1) including the explained variance and dots colored by QC_Warning using `stats::prcomp` and `ggplot2::ggplot`.

The values are by default scaled and centered in the PCA and proteins with missing NPX values are by default removed from the corresponding assay.

Unique sample names are required.

Imputation by the median is done for assays with missingness <10\ for multi-plate projects and <5\

The plot is printed, and a list of ggplot objects is returned. If byPanel = TRUE, the data processing (imputation of missing values etc) and subsequent PCA is performed separately per panel. A faceted plot is printed, while the individual ggplot objects are returned.

The arguments outlierDefX and outlierDefY can be used to identify outliers in the PCA. Samples more than +/- outlierDefX and outlierDefY standard deviations from the mean of the plotted PC will be labelled. Both arguments have to be specified.

Usage

```
olink_pca_plot(
  df,
  check_log = NULL,
  color_g = "QC_Warning",
  x_val = 1L,
```

```
  y_val = 2L,
  label_samples = FALSE,
  drop_assays = FALSE,
  drop_samples = FALSE,
  n_loadings = 0,
  loadings_list = NULL,
  byPanel = FALSE,
  outlierDefX = NA,
  outlierDefY = NA,
  outlierLines = FALSE,
  label_outliers = TRUE,
  quiet = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

df	data frame in long format with Sample Id, NPX and column of choice for colors.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
color_g	Character value indicating which column to use for colors (default QC_Warning). Continuous color scale for Olink(R) Sample Index (OSI) columns OSITimeToCentrifugation, OSIPreparationTemperature and OSISummary is also supported.
x_val	Integer indicating which principal component to plot along the x-axis (default 1)
y_val	Integer indicating which principal component to plot along the y-axis (default 2)
label_samples	Logical. If TRUE, points are replaced with SampleID (default FALSE)
drop_assays	Logical. All assays with any missing values will be dropped. Takes precedence over sample drop.
drop_samples	Logical. All samples with any missing values will be dropped.
n_loadings	Integer. Will plot the top n_loadings based on size.
loadings_list	Character vector indicating for which OlinkID's to plot as loadings. It is possible to use n_loadings and loadings_list simultaneously.
byPanel	Perform the PCA per panel (default FALSE)
outlierDefX	The number standard deviations along the PC plotted on the x-axis that defines an outlier. See also 'Details"
outlierDefY	The number standard deviations along the PC plotted on the y-axis that defines an outlier. See also 'Details"
outlierLines	Draw dashed lines at +/- outlierDefX and outlierDefY standard deviations from the mean of the plotted PCs (default FALSE)
label_outliers	Use ggrepel to label samples lying outside the limits set by the outlierLines (default TRUE)

quiet	Logical. If TRUE, the resulting plot is not printed
verbose	Logical. Whether warnings about the number of samples and/or assays dropped or imputed should be printed to the console.
...	coloroption passed to specify color order.

Value

A list of objects of class "ggplot", each plot contains scatter plot of PCs

Examples

```
if (rlang::is_installed(pkg = c("ggrepel", "ggpubr"))) {
  npx_data <- npx_data1 |>
    dplyr::filter(
      !grepl(pattern = "CONTROL",
             x = .data[["SampleID"]],
             ignore.case = TRUE)
    )

  check_log <- check_npx(npx_data)

  # PCA using all the data
  OlinkAnalyze::olink_pca_plot(
    df = npx_data,
    check_log = check_log,
    color_g = "QC_Warning"
  )

  # PCA per panel
  g <- OlinkAnalyze::olink_pca_plot(
    df = npx_data,
    check_log = check_log,
    color_g = "QC_Warning",
    byPanel = TRUE
  )
  g[[2L]] # Plot only the second panel

  # Label outliers
  OlinkAnalyze::olink_pca_plot(
    df = npx_data,
    check_log = check_log,
    color_g = "QC_Warning",
    outlierDefX = 2L,
    outlierDefY = 4L
   ) # All data

  OlinkAnalyze::olink_pca_plot(
    df = npx_data,
    check_log = check_log,
    color_g = "QC_Warning",
    outlierDefX = 2.5,
    outlierDefY = 4L,
```

```
    byPanel = TRUE
  ) # Per panel

  # Retrieve the outliers
  g <- OlinkAnalyze::olink_pca_plot(
    df = npx_data,
    check_log = check_log,
    color_g = "QC_Warning",
    outlierDefX = 2.5,
    outlierDefY = 4L,
    byPanel = TRUE
  )
  outliers <- lapply(g, function(x) {
    return(x$data)
  }) |>
  dplyr::bind_rows() |>
  dplyr::filter(
    .data[["Outlier"]] == 1L
  )
}
```

`olink_plate_randomizer` *Randomly assign samples to plates*

Description

Generates a scheme for how to plate samples with an option to keep subjects on the same plate and/or to keep studies together.

Usage

```
olink_plate_randomizer(
  Manifest,
  PlateSize = 96,
  Product,
  SubjectColumn,
  iterations = 500,
  available.spots,
  num_ctrl = 8L,
  rand_ctrl = FALSE,
  seed,
  study = NULL
)
```

Arguments

`Manifest`	tibble/data frame in long format containing all sample ID's. Sample ID column must be named SampleID.
`PlateSize`	Integer. Either 96 or 48. 96 is default.
`Product`	String. Name of Olink product used to set PlateSize if not provided. Optional.
`SubjectColumn`	(Optional) Column name of the subject ID column. Cannot contain missing values. If provided, subjects are kept on the same plate. This argument is used for longitudinal studies and must be a separate column from the SampleID column.
`iterations`	Number of iterations for fitting subjects on the same plate.
`available.spots`	Numeric. Number of wells available on each plate. Maximum 40 for T48 and 88 for T96. Takes a vector equal to the number of plates to be used indicating the number of wells available on each plate.
`num_ctrl`	Numeric. Number of controls on each plate (default = 8)
`rand_ctrl`	Logical. Whether controls are added to be randomized across the plate (default = FALSE)
`seed`	Seed to set. Highly recommend setting this for reproducibility.
`study`	String. Optional. Name of column that includes study information. For when multiple studies are being plated and randomizing within studies. If `study` column is present in manifest, within study randomization will be performed.

Details

Variables of interest should if possible be randomized across plates to avoid confounding with potential plate effects. In the case of multiple samples per subject (e.g. in longitudinal studies), Olink recommends keeping each subject on the same plate. This can be achieved using the SubjectColumn argument.

Value

A "tibble" including SampleID, SubjectID etc. assigned to well positions. Columns include same columns as Manifest with additional columns:

- plate: Plate number
- column: Column on the plate
- row: Row on the plate
- well: Well location on the plate

See Also

- `olink_displayPlateLayout()` for visualizing the generated plate layouts
- `olink_displayPlateDistributions()` for validating that sites are properly randomized

Examples

```
#Generate randomization scheme using complete randomization
randomized.manifest_a <- olink_plate_randomizer(manifest, seed=12345)

# Generate randomization scheme that keeps subjects on the same plate
# (for longitudinal studies)
randomized.manifest_b <- olink_plate_randomizer(manifest,
                                                SubjectColumn="SubjectID",
                                                available.spots=c(88,88),
                                                seed=12345)

# Generate randomization scheme that keeps samples from the same
# study together
randomized.manifest_c <- olink_plate_randomizer(manifest, study = "Site")

# Visualize the generated plate layouts
olink_displayPlateLayout(randomized.manifest_a, fill.color = 'Site')
olink_displayPlateLayout(randomized.manifest_a, fill.color = 'SubjectID')
olink_displayPlateLayout(randomized.manifest_b, fill.color = 'Site')
olink_displayPlateLayout(randomized.manifest_b, fill.color = 'SubjectID')
olink_displayPlateLayout(randomized.manifest_c, fill.color = 'Site')

# Validate that sites are properly randomized
olink_displayPlateDistributions(randomized.manifest_a, fill.color = 'Site')
olink_displayPlateDistributions(randomized.manifest_b, fill.color = 'Site')
```

---

olink_qc_plot *Function to plot an overview of a sample cohort per Panel.*

Description

Generates a facet plot per Panel using ggplot2::ggplot and ggplot2::geom_point and stats::IQR plotting IQR vs. median for all samples. Horizontal dashed lines indicate $\pm$*IQR_outlierDef* standard deviations from the mean IQR (default 3). Vertical dashed lines indicate $\pm$*median_outlierDef* standard deviations from the mean sample median (default 3).

Usage

```
olink_qc_plot(
  df,
  check_log = NULL,
  color_g = "QC_Warning",
  plot_index = FALSE,
  label_outliers = FALSE,
  IQR_outlierDef = 3L,
  median_outlierDef = 3L,
  outlierLines = TRUE,
```

```
  facetNrow = NULL,
  facetNcol = NULL,
  ...
)
```

Arguments

df	NPX data frame in long format. Must have columns SampleID, NPX and Panel.
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
color_g	Character value indicating which column to use as fill color (default QC_Warning). Continuous color scale for Olink(R) Sample Index (OSI) columns OSITimeToCentrifugation, OSIPreparationTemperature and OSISummary is also supported.
plot_index	Boolean. If FALSE (default), a point will be plotted for a sample. If TRUE, a sample's unique index number is displayed.
label_outliers	Boolean. If TRUE, an outlier sample will be labelled with its SampleID. (default FALSE)
IQR_outlierDef	The number of standard deviations from the mean IQR that defines an outlier. (default 3)
median_outlierDef	The number of standard deviations from the mean sample median that defines an outlier. (default 3)
outlierLines	Draw dashed lines at $\pm$*IQR_outlierDef* and $\pm$*median_outlierDef* standard deviations from the mean IQR and sample median respectively. (default TRUE)
facetNrow	The number of rows that the panels are arranged on.
facetNcol	The number of columns that the panels are arranged on.
...	coloroption passed to specify color order.

Value

An object of class "ggplot". Scatterplot shows IQR vs median for all samples per panel

Examples

```
if (rlang::is_installed(pkg = c("ggrepel"))) {

  # standard plot
  OlinkAnalyze::olink_qc_plot(
    df = npx_data1,
    color_g = "QC_Warning",
    label_outliers = TRUE
  )

  # Change the outlier threshold to +/-4SD
  OlinkAnalyze::olink_qc_plot(
    df = npx_data1,
```

```
    color_g = "QC_Warning",
    IQR_outlierDef = 4L,
    median_outlierDef = 4L,
    label_outliers = TRUE
  )

  # Identify the outliers
  qc <- OlinkAnalyze::olink_qc_plot(
    df = npx_data1,
    color_g = "QC_Warning",
    IQR_outlierDef = 4L,
    median_outlierDef = 4L,
    label_outliers = TRUE
  )

  outliers <- qc$data |>
    dplyr::filter(
      .data[["Outlier"]] == 1L
    )
}
```

`olink_ttest` *Function which performs a t-test per protein*

Description

Performs a Welch 2-sample t-test or paired t-test at confidence level 0.95 for every protein (by OlinkID) for a given grouping variable using stats::t.test and corrects for multiple testing by the Benjamini-Hochberg method ("fdr") using stats::p.adjust. Adjusted p-values are logically evaluated towards adjusted p-value<0.05. The resulting t-test table is arranged by ascending p-values.

Usage

```
olink_ttest(df, variable, pair_id, check_log = NULL, ...)
```

Arguments

`df`	NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt and a factor with 2 levels.
`variable`	Character value indicating which column should be used as the grouping variable. Needs to have exactly 2 levels.
`pair_id`	Character value indicating which column indicates the paired sample identifier.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
`...`	Options to be passed to t.test. See `?t.test` for more information.

Value

A "tibble" containing the t-test results for every protein. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID
- Panel: "character" Name of Olink Panel
- estimate: "numeric" difference in mean NPX between groups
- Group 1: "numeric" Column is named first level of variable when converted to factor, contains mean NPX for that group
- Group 2: "numeric" Column is named second level of variable when converted to factor, contains mean NPX for that group
- statistic: "named numeric" value of the t-statistic
- p.value: "numeric" p-value for the test
- parameter: "named numeric" degrees of freedom for the t-statistic
- conf.low: "numeric" confidence interval for the mean (lower end)
- conf.high: "numeric" confidence interval for the mean (upper end)
- method: "character" which t-test method was used
- alternative: "character" describes the alternative hypothesis
- Adjusted_pval: "numeric" adjusted p-value for the test (Benjamini & Hochberg)
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("broom"))) {
  npx_df <- OlinkAnalyze::npx_data1 |>
    dplyr::filter(
    !grepl(
      pattern = "control",
      x = .data[["SampleID"]],
      ignore.case = TRUE
    )
  )
  check_log <- OlinkAnalyze::check_npx(df = npx_df)

  ttest_results <- OlinkAnalyze::olink_ttest(
    df = npx_df,
    variable = "Treatment",
    alternative = "two.sided",
    check_log = check_log
  )

  # Paired t-test
  ttest_paired_results <- npx_df |>
    dplyr::filter(
      .data[["Time"]] %in% c("Baseline", "Week.6")
```

```
    ) |>
    OlinkAnalyze::olink_ttest(
      variable = "Time",
      pair_id = "Subject",
      check_log = check_log
    )
}
```

---

olink_umap_plot *Function to make a UMAP plot from the data*

Description

Computes a manifold approximation and projection using umap::umap and plots the two specified components. Unique sample names are required and imputation by the median is done for assays with missingness <10\ projects and <5\

Usage

```
olink_umap_plot(
  df,
  color_g = "QC_Warning",
  x_val = 1L,
  y_val = 2L,
  check_log = NULL,
  config = NULL,
  label_samples = FALSE,
  drop_assays = FALSE,
  drop_samples = FALSE,
  byPanel = FALSE,
  outlierDefX = NA,
  outlierDefY = NA,
  outlierLines = FALSE,
  label_outliers = TRUE,
  quiet = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

df	data frame in long format with Sample Id, NPX and column of choice for colors.
color_g	Character value indicating which column to use for colors (default QC_Warning). Continuous color scale for Olink(R) Sample Index (OSI) columns OSITimeToCentrifugation, OSIPreparationTemperature and OSISummary is also supported.

x_val	Integer indicating which UMAP component to plot along the x-axis (default 1)
y_val	Integer indicating which UMAP component to plot along the y-axis (default 2)
check_log	A named list returned by check_npx(). If NULL, check_npx() will be run internally using df.
config	object of class umap.config, specifying the parameters for the UMAP algorithm (default umap::umap.defaults)
label_samples	Logical. If TRUE, points are replaced with SampleID (default FALSE)
drop_assays	Logical. All assays with any missing values will be dropped. Takes precedence over sample drop.
drop_samples	Logical. All samples with any missing values will be dropped.
byPanel	Perform the UMAP per panel (default FALSE)
outlierDefX	The number standard deviations along the UMAP dimension plotted on the x-axis that defines an outlier. See also 'Details"
outlierDefY	The number standard deviations along the UMAP dimension plotted on the y-axis that defines an outlier. See also 'Details"
outlierLines	Draw dashed lines at +/- outlierDefX and outlierDefY standard deviations from the mean of the plotted PCs (default FALSE)
label_outliers	Use ggrepel to label samples lying outside the limits set by the outlierLines (default TRUE)
quiet	Logical. If TRUE, the resulting plot is not printed
verbose	Logical. Whether warnings about the number of samples and/or assays dropped or imputed should be printed to the console.
...	coloroption passed to specify color order.

Details

The plot is printed, and a list of ggplot objects is returned.

If byPanel = TRUE, the data processing (imputation of missing values etc) and subsequent UMAP is performed separately per panel. A faceted plot is printed, while the individual ggplot objects are returned. The arguments outlierDefX and outlierDefY can be used to identify outliers in the UMAP results. Samples more than +/- outlierDefX and outlierDefY standard deviations from the mean of the plotted UMAP component will be labelled. Both arguments have to be specified.

NOTE: UMAP is a non-linear data transformation that might not accurately preserve the properties of the data. Distances in the UMAP plane should therefore be interpreted with caution.

Value

A list of objects of class "ggplot", each plot contains scatter plot of UMAPs

Examples

```
if (rlang::is_installed(pkg = c("umap", "ggrepel", "ggpubr"))) {

  npx_data <- npx_data1 |>
```

```
  dplyr::mutate(
    SampleID = paste(.data[["SampleID"]], "_", .data[["Index"]], sep = "")
  )

check_log <- check_npx(df = npx_data)

# UMAP using all the data
OlinkAnalyze::olink_umap_plot(
  df = npx_data,
  color_g = "QC_Warning",
  check_log = check_log
)

# UMAP per panel
g <- OlinkAnalyze::olink_umap_plot(
  df = npx_data,
  color_g = "QC_Warning",
  byPanel = TRUE,
  check_log = check_log
)
# Plot only the Inflammation panel
g$Inflammation

# Label outliers
OlinkAnalyze::olink_umap_plot(
  df = npx_data,
  color_g = "QC_Warning",
  outlierDefX = 2L,
  outlierDefY = 4L,
  check_log = check_log
)

OlinkAnalyze::olink_umap_plot(
  df = npx_data,
  color_g = "QC_Warning",
  outlierDefX = 3L,
  outlierDefY = 2L,
  byPanel = TRUE,
  check_log = check_log
)

# Retrieve outliers
p <- OlinkAnalyze::olink_umap_plot(
  df = npx_data,
  color_g = "QC_Warning",
  outlierDefX = 3L,
  outlierDefY = 2L,
  byPanel = TRUE,
  check_log = check_log
)
outliers <- lapply(p, function(x) x$data) |>
  dplyr::bind_rows() |>
  dplyr::filter(
```

```
      .data[["Outlier"]] == 1L
    )
}
```

`olink_volcano_plot` *Easy volcano plot with Olink theme*

Description

Generates a volcano plot using the results of the olink_ttest function using ggplot and ggplot2::geom_point. The estimated difference is plotted on the x-axis and the negative 10-log p-value on the y-axis. The horizontal dotted line indicates p-value=0.05. Dots are colored based on the Benjamini-Hochberg adjusted p-value cutoff 0.05 and can optionally be annotated by OlinkID.

Usage

```
olink_volcano_plot(p.val_tbl, x_lab = "Estimate", olinkid_list = NULL, ...)
```

Arguments

`p.val_tbl`	a data frame of results generated by olink_ttest()
`x_lab`	Optional. Character value to use as the X-axis label
`olinkid_list`	Optional. Character vector of proteins (by OlinkID) to label in the plot. If not provided, default is to label all significant proteins.
`...`	Optional. Additional arguments for olink_color_discrete()

Value

An object of class "ggplot", plotting significance (y-axis) by estimated difference between groups (x-axis) for each protein.

Examples

```
if (rlang::is_installed(pkg = c("broom", "ggrepel"))) {
  npx_df <- npx_data1 |>
  dplyr::filter(
    !grepl(pattern = "control",
           x = .data[["SampleID"]],
           ignore.case = TRUE

     )
  )

  check_log <- check_npx(df = npx_df)

  ttest_results <- OlinkAnalyze::olink_ttest(
```

```
    df = npx_df,
    check_log = check_log,
    variable = "Treatment",
    alternative = "two.sided"
  )

  OlinkAnalyze::olink_volcano_plot(
    p.val_tbl = ttest_results
  )
}
```

`olink_wilcox` *Function which performs a Mann-Whitney U Test per protein*

Description

Performs a Welch 2-sample Mann-Whitney U Test at confidence level 0.95 for every protein (by OlinkID) for a given grouping variable using stats::wilcox.test and corrects for multiple testing by the Benjamini-Hochberg method ("fdr") using stats::p.adjust. Adjusted p-values are logically evaluated towards adjusted p-value<0.05. The resulting Mann-Whitney U Test table is arranged by ascending p-values.

Usage

```
olink_wilcox(df, variable, pair_id, check_log = NULL, ...)
```

Arguments

`df`	NPX or Quantified_value data frame in long format with at least protein name (Assay), OlinkID, UniProt and a factor with 2 levels.
`variable`	Character value indicating which column should be used as the grouping variable. Needs to have exactly 2 levels.
`pair_id`	Character value indicating which column indicates the paired sample identifier.
`check_log`	A named list returned by `check_npx()`. If `NULL`, `check_npx()` will be run internally using `df`.
`...`	Options to be passed to wilcox.test. See `?wilcox_test` for more information.

Value

A data frame containing the Mann-Whitney U Test results for every protein. Columns include:

- Assay: "character" Protein symbol
- OlinkID: "character" Olink specific ID
- UniProt: "character" UniProt ID

- Panel: "character" Name of Olink Panel
- estimate: "numeric" median of NPX differences between groups
- statistic: "named numeric" the value of the test statistic with a name describing it
- p.value: "numeric" p-value for the test
- conf.low: "numeric" confidence interval for the median of differences (lower end)
- conf.high: "numeric" confidence interval for the median of differences (upper end)
- method: "character" which wilcoxon method was used
- alternative: "character" describes the alternative hypothesis
- Adjusted_pval: "numeric" adjusted p-value for the test (Benjamini&Hochberg)
- Threshold: "character" if adjusted p-value is significant or not (< 0.05)

Examples

```
if (rlang::is_installed(pkg = c("broom"))) {
  npx_df <- npx_data1 |>
    dplyr::filter(
    !grepl(
      pattern = "control",
      x = .data[["SampleID"]],
      ignore.case = TRUE
    )
  )
  check_log <- OlinkAnalyze::check_npx(df = npx_df)

  # Mann-Whitney U Test
  wilcox_results <- OlinkAnalyze::olink_wilcox(
    df = npx_df,
    variable = "Treatment",
    alternative = "two.sided",
    check_log = check_log
  )

  # Paired Mann-Whitney U Test
  wilcox_paired_results <- npx_df |>
    dplyr::filter(
      .data[["Time"]] %in% c("Baseline", "Week.6")
    ) |>
    OlinkAnalyze::olink_wilcox(
      variable = "Time",
      pair_id = "Subject",
      check_log = check_log
    )
}
```

read_npx *Read Olink data in R.*

Description

Imports a file exported from Olink software that quantifies protein levels in NPX, Ct or absolute quantification.

Note: Do not modify the Olink software output file prior to importing it with `read_npx` as it might fail.

Usage

```
read_npx(
  filename,
  out_df = "tibble",
  long_format = NULL,
  olink_platform = NULL,
  data_type = NULL,
  .ignore_files = c("README.txt"),
  quiet = TRUE,
  legacy = FALSE
)

read_NPX(
  filename,
  out_df = "tibble",
  long_format = NULL,
  olink_platform = NULL,
  data_type = NULL,
  .ignore_files = c("README.txt"),
  quiet = TRUE,
  legacy = FALSE
)
```

Arguments

`filename`	Path to Olink software output file in wide or long format. Expecting extensions "xls" or "xlsx" for excel files, "csv" or "txt" for delim files, "parquet" for parquet files, and "zip" for compressed files.
`out_df`	The class of the output dataset. One of "tibble" or "arrow". Defaults to "tibble".
`long_format`	Boolean marking format of input file. One of `TRUE` for long format and `FALSE` for wide format files. Defaults to `NULL` for auto-detection.
`olink_platform`	Olink platform used to generate the input file. One of "Target 48", "Flex", "Target 96", "Explore 3072", "Explore HT", "Focus", or "Reveal". Defaults to `NULL` for auto-detection.

data_type	Quantification method of the input data. One of "Ct", "NPX", or "Quantified". Defaults to NULL for auto-detection.
.ignore_files	Character vector of files included in the zip-compressed Olink software output files that should be ignored. Used only for zip-compressed input files (default = *c("README.txt")*).
quiet	Boolean to print a confirmation message when reading the input file. Applies to excel or delimited input only. TRUE skips printing the message, and FALSE otherwise.
legacy	Boolean to run the legacy version of the read_npx function. **Important: should be used only to wide format files from Target 96 or Target 48 with NPX Software version earlier than 1.8!** (default FALSE).

Value

Dataset, "tibble" or "ArrowObject", with Olink data in long format.

Author(s)

Klev Diamanti Kathleen Nevola Pascal Pucholt Christoffer Cambronero Boxi Zhang Olof Mansson Marianne Sandin

Examples

```
file <- system.file("extdata",
                    "npx_data_ext.parquet",
                    package = "OlinkAnalyze")
read_npx(filename = file)
```

---

set_plot_theme *Function to set plot theme*

Description

This function sets a coherent plot theme for functions.

Usage

```
set_plot_theme(font = "Arial")
```

Arguments

font	Font family to use for text elements. Default: "Arial".

Value

No return value, used as theme for ggplots

Examples

```
if (rlang::is_installed(pkg = c("showtext", "systemfonts",
                                "sysfonts", "curl"))) {
  ggplot2::ggplot(
    data = datasets::mtcars,
    mapping = ggplot2::aes(
      x = .data[["wt"]],
      y = .data[["mpg"]],
      color = as.factor(x = .data[["cyl"]])
    )
  ) +
    ggplot2::geom_point(
      size = 4L
    ) +
    OlinkAnalyze::set_plot_theme()

  ggplot2::ggplot(
    data = datasets::mtcars,
    mapping = ggplot2::aes(
      x = .data[["wt"]],
      y = .data[["mpg"]],
     color = as.factor(x = .data[["cyl"]])
    )
  ) +
    ggplot2::geom_point(
      size = 4L
    ) +
    OlinkAnalyze::set_plot_theme(
      font = ""
    )
}
```

Index