Package ‘OlinkAnalyze’

February 22, 2024

Type Package

Title Facilitate Analysis of Proteomic Data from Olink

Version 3.7.0

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

Depends R (>= 4.1.0)

Imports broom, car, dplyr, emmeans, forcats, generics, ggplot2,
    ggpubr, ggrepel, grid, grid, lme4, magrittr, methods, readxl, rlang, stringr, tidyselect, tools, utils, zip

Suggests arrow, clusterProfiler, extrafont, FSA, ggplotify, here,
    kableExtra, knitr, markdown, msigdbi, ordinal, pheatmap, rmarkdown, scales, systemfonts, testthat (>= 3.0.0),
    umap, vdiffr

VignetteBuilder kableExtra, knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

URL https://olink.com/

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

NeedsCompilation no
Author  Kathleen Nevola [aut, cre] (<https://orcid.org/0000-0002-5183-6444>, kathy-nevola),
Marianne Sandin [aut] (<https://orcid.org/0000-0001-6186-963X>, marisand),
Jamey Guess [aut] (<https://orcid.org/0000-0002-4017-0923>, jrguess),
Simon Forsberg [aut] (<https://orcid.org/0000-0002-7451-9222>, simfor),
Christoffer Cambronero [aut] (Orbmac),
Pascal Pucholt [aut] (<https://orcid.org/0000-0003-3342-1373>, AskPascal),
Boxi Zhang [aut] (<https://orcid.org/0000-0001-7758-6204>, boxizhang),
Masoumeh Sheikhi [aut] (MasoumehSheikhi),
Klev Diamanti [aut] (<https://orcid.org/0000-0002-4922-8415>, klevdiamanti),
Amrita Kar [aut] (amrita-kar),
Lei Conze [aut] (leiimuC),
Kristian Hodén [ctb] (<https://orcid.org/0000-0003-0354-0662>, kristianHoden),
Per Eriksson [ctb] (<https://orcid.org/0000-0001-7633-403X>, b_watcher),
Nicola Moloney [ctb] (<https://orcid.org/0000-0003-4967-3284>),
Britta Löstedt [ctb] (<https://orcid.org/0000-0003-6545-5489>),
Emmett Sprecher [ctb] (<https://orcid.org/0000-0002-7710-695X>),
Jessica Barbagallo [ctb] (jbarbagallo),
Olof Mansson [ctr] (olofmansson),
Ola Caster [ctb] (OlaCaster),
Kristyn Chin [ctb] (kristynchin-olink),
Olink [cph, fnd]

Maintainer  Kathleen Nevola <biostattools@olink.com>

Repository  CRAN

Date/Publication  2024-02-22 08:10:05 UTC

R topics documented:

check_data_completeness ........................................... 3
manifest ................................................................. 4
npx_data1 ............................................................... 5
npx_data2 ............................................................... 5
olink_anova ............................................................. 6
olink_anova_posthoc .................................................. 8
olink_boxplot .......................................................... 11
olink_bridgeselectior .................................................. 12
olink_color_discrete ................................................. 13
olink_color_gradient ................................................ 14
olink_displayPlateDistributions .................................... 14
olink_displayPlateLayout ........................................... 15
olink_dist_plot ....................................................... 16
olink_fill_discrete .................................................. 17
olink_fill_gradient .................................................. 18
check_data_completeness

Check data completeness

Description

Throw informative warnings if a dataset appears to have problems

Usage

check_data_completeness(df)
Arguments
df a NPX dataframe, e.g. from read_NPX()

Value
None. Used for side effects (warnings)

Examples

npx_data1 %>%
dplyr::mutate(NPX = dplyr::if_else(
    SampleID == "A1" & Panel == "Olink Cardiometabolic",
    NA_real_,
    NA_real_,
    NPX)) %>%
OlinkAnalyze:::check_data_completeness()

manifest

Example Sample Manifest

Description
Sample manifest is generated randomly to demonstrate use of functions in this package.

Usage
manifest

Format
This dataset contains columns:

<table>
<thead>
<tr>
<th>SubjectID</th>
<th>Subject Identifier, A-Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit</td>
<td>Visit Number, 1-6</td>
</tr>
<tr>
<td>SampleID</td>
<td>138 unique sample IDs</td>
</tr>
<tr>
<td>Site</td>
<td>Site1 or Site2</td>
</tr>
</tbody>
</table>

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

Data is generated randomly to demonstrate use of functions in 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. Dataset npx_data1 is an Olink NPX data file (tibble) in long format with 158 unique Sample ID’s (including 2 repeats each of control samples: CONTROL_SAMPLE_AS 1 CONTROL_SAMPLE_AS 2). The data also contains 1104 assays (uniquely identified using OlinkID) over 2 Panels.

Description

Data is generated randomly to demonstrate use of functions in this package. The format is very similar to data(npx_data1). Both datasets can be used together 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 ID’s (including 2 repeats each of control samples: CONTROL_SAMPLE_AS1 CONTROL_SAMPLE_AS2). The data also contains 1104 assays (uniquely identified using OlinkID) over 2 Panels. This dataset also contain 16 bridge samples with SampleID’s that are also present in data(npx_data1). These sample ID’s are: A13, A29, A30, A36, A45, A46, A52, A63, A71, A73, B3, B4, B37, B45, B63, 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.

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. 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.
Usage

\[
\text{olink\_anova(}
\text{df,}
\text{variable,}
\text{outcome = "NPX",}
\text{covariates = NULL,}
\text{model\_formula,}
\text{return\_covariates = FALSE,}
\text{verbose = TRUE}
\text{)}
\]

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.

- **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 \text{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.

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

```r
library(dplyr)

npx_df <- npx_data1 %>% filter(!grepl('control',SampleID, ignore.case = TRUE))

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

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

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

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 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).

Usage

```r
olink_anova_posthoc(
  df,
  olinkid_list = NULL,
  variable,
)```
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 ’:’ 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.

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

library(dplyr)

npx_df <- npx_data1 %>% filter(!grepl('"control",SampleID, ignore.case = TRUE))

#Two-way ANOVA, one main effect (Site) covariate.
#Results in model NPX~Treatment*Time+Site.
anova_results <- olink_anova(df = npx_df,
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 %>%
filter(Threshold == 'Significant' & term == 'Treatment:Time') %>%
select(OlinkID) %>%
distinct() %>%
pull()

#Posthoc, all pairwise comparisons
anova_posthoc_results <- olink_anova_posthoc(npx_df,
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 <- olink_anova_posthoc(npx_df,
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 ggplot and ggplot2::geom_boxplot.

**Usage**

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

- `...`  
  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

```r
library(dplyr)

anova_results <- olink_anova(npx_data1, variable = "Site")
significant_assays <- anova_results %>%
  filter(Threshold == 'Significant') %>%
pull(OlinkID)
olink_boxplot(npx_data1,
  variable = "Site",
  olinkid_list = significant_assays,
  verbose = TRUE,
  number_of_proteins_per_plot = 3)
```

**olink_bridgeselector**  
*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, which passes 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 sampleMissingFreq = 0.10 which represents a maximum of 10% data below LOD per sample. If there are not enough samples output, increase to 20%.

The function accepts NPX Excel files with data < LOD replaced.

**Usage**

```r
olink_bridgeselector(df, sampleMissingFreq, n)
```

**Arguments**

- `df`  
  Tibble/data frame in long format such as produced by the Olink Analyze read_NPX function.
- `sampleMissingFreq`  
  The threshold for sample wise missingness.
- `n`  
  Number of bridge samples to be selected.

**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

    bridge_samples <- olink_bridgeselector(npx_data1, sampleMissingFreq = 0.1, n = 20)

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
coloroption  string, one or more of the following: c('red', 'orange', 'yellow', 'green', 'teal',
             'turquoise', 'lightblue', 'darkblue', 'purple', 'pink')

Value

No return value, called for side effects

Examples

library(ggplot2)

    ggplot(mtcars, aes(x=wt, y=mpg, color=as.factor(cyl))) +
    geom_point(size = 4) +
    olink_color_discrete() +
    theme_bw()

    ggplot(mtcars, aes(x=wt, y=mpg, color=as.factor(cyl))) +
    geom_point(size = 4) +
    olink_color_discrete(coloroption = c('lightblue', 'red', 'green')) +
    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 scale_color_gradientn()
alpha  transparency (optional)
coloroption  string, one or more of the following: c('red', 'orange', 'yellow', 'green', 'teal', 'turquoise', 'lightblue', 'darkblue', 'purple', 'pink')

Value

No return value, called for side effects

Examples

library(ggplot2)

dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
dsub$diff <- with(dsub, sqrt(abs(x-y))* sign(x-y))

ggplot(dsub, aes(x, y, colour=diff)) +
geom_point() +
theme_bw() +
olink_color_gradient()

olink_displayPlateDistributions

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

```r
olink_displayPlateDistributions(data, fill.color)
```

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)

See Also

- `olink_plate_randomizer()` for generating a plating scheme
- `olink_displayPlateLayout()` for visualizing the generated plate layouts

Examples

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

---

### olink_displayPlateLayout

*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

```r
olink_displayPlateLayout(
  data,
  fill.color,
  PlateSize = 96,
  num_ctrl = 8,
  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.

See Also

- olink_plate_randomizer() for generating a plating scheme
- olink_displayPlateDistributions() for validating that sites are properly randomized

Examples

```r
randomized.manifest <- olink_plate_randomizer(manifest)
olink_displayPlateLayout(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

```r
olink_dist_plot(df, color_g = "QC_Warning", ...)
```

Arguments

- **df**: NPX data frame in long format. Must have columns SampleID, NPX and Panel
- **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

```r
olink_dist_plot(npx_data1, color_g = "QC_Warning")
```

Description

Olink fill scale for discrete ggplots

Usage

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

Arguments

- `...` Optional. Additional arguments to pass to `ggplot2::discrete_scale()`
- `alpha` transparency (optional)

Value

No return value, called for side effects

Examples

```r
library(ggplot2)

dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
dsub$diff <- with(dsub, sqrt(abs(x-y))* sign(x-y))

ggplot(dsub, aes(x, y, colour=diff)) +
geom_point() +
theme_bw() +
olink_fill_discrete()
```
olink_fill_gradient  

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', 'turquoise', 'lightblue', 'darkblue', 'purple', 'pink')

Value

No return value, called for side effects

Examples

library(ggplot2)

dsub <- subset(diamonds, x > 5 & x < 6 & y > 5 & y < 6)
dsub$diff <- with(dsub, sqrt(abs(x-y))* sign(x-y))
ggplot(dsub, aes(x, y, colour=diff)) + 
  geom_point() +
  theme_bw() +
  olink_fill_gradient()

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.
olink_heatmap_plot

Usage

olink_heatmap_plot(
  df,
  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.

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

```r
library(dplyr)
npx_data <- npx_data1 %>%
    filter(!stringr::str_detect(SampleID,'CONT'))
try({ # This will fail if ggplotify is not installed
  #Heatmap
  olink_heatmap_plot(df=npx_data)

  #Heatmap with annotation
  olink_heatmap_plot(df=npx_data, variable_row_list = c('Time', 'Site'))

  #Heatmap with calls from pheatmap
  olink_heatmap_plot(df=npx_data, cutree_rows = 3)
})
```

olink_lmer  
Function which 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/or covariates.

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. 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 random variable only takes main effect(s).

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.
olink_lmer

Usage

olink_lmer(
  df,
  variable,
  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. 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.
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.

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" Name of Olink Panel
- sumsq: "numeric" sum of square
- meansq: "numeric" mean of square
- NumDF: "integer" numerator of degrees of freedom
• DenDF: "numeric" denominator of degrees 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

# Results in model NPX~Time*Treatment+(1|Subject)+(1|Site)
lmer_results <- olink_lmer(df = npx_data1,
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,
  variable,
  outcome = "NPX",
  random,
  olinkid_list = NULL,
  covariates = NULL,
  x_axis_variable,
  col_variable = NULL,
  number_of_proteins_per_plot = 6,
  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.
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.

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

```r
library(dplyr)

lmer_results <- olink_lmer(df = npx_data1, 
                          variable=c("Time", 'Treatment'),
                          random = c('Subject'))

assay_list <- lmer_results %>%
  filter(Threshold == 'Significant' & term == 'Time:Treatment') %>%
  select(OlinkID) %>%
  distinct() %>%
  pull()

list_of_pointrange_plots <- olink_lmer_plot(df = npx_data1, 
                                          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 = 10)
```
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.

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.

Usage

```r
olink_lmer_posthoc(
  df, 
  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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>df</td>
<td>NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt, 1-2 variables with at least 2 levels and subject ID.</td>
</tr>
<tr>
<td>olinkid_list</td>
<td>Character vector of OlinkID’s on which to perform post hoc analysis. If not specified, all assays in df are used.</td>
</tr>
<tr>
<td>variable</td>
<td>Single character value or character array. Variable(s) to test. If length &gt; 1, the included variable names will be used in crossed analyses.</td>
</tr>
<tr>
<td>outcome</td>
<td>Character. The dependent variable. Default: NPX.</td>
</tr>
<tr>
<td>random</td>
<td>Single character value or character array.</td>
</tr>
</tbody>
</table>
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.

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

library(dplyr)

lmer_results <- olink_lmer(df = npx_data1,
variable=c("Time", 'Treatment'),
...
### olink_normalization

**Normalization of all proteins (by OlinkID).**

#### Description

Normalizes NPX data frames to another data frame or to reference medians. If two dataframes are normalized to one another, Olink’s default is using the older dataframe as reference. The function handles three different types of normalization:

- **Bridging normalization:** One of the dataframes is adjusted to another using overlapping samples (bridge samples). The overlapping samples need to be named the same between the dataframes and adjustment is made using the median of the paired differences between the bridge samples in the two data frames. The two dataframes are inputs df1 and df2, the one being adjusted to is specified in the input `reference_project` and the overlapping samples are specified in `overlapping_samples_df1`. Only `overlapping_samples_df1` should be input, no matter which dataframe is used as `reference_project`.

- **Subset normalization:** One of the dataframes is adjusted to another dataframe using a sample subset. Adjustment is made using the differences in median between the subsets from the two dataframes. Both `overlapping_samples_df1` and `overlapping_samples_df2` need to be input. The samples do not need to be named the same.

A special case of subset normalization are to use all samples (except control samples and samples with QC warning) from df1 as input in `overlapping_samples_df1` and all samples from df2 as input in `overlapping_samples_df2`.

```r
random = c('Subject'))

assay_list <- lmer_results %>%
  filter(Threshold == 'Significant' & term == 'Time:Treatment') %>%
  select(OlinkID) %>%
  distinct() %>%
  pull()

results_lmer_posthoc <- olink_lmer_posthoc(df = npx_data1,
  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 <- olink_lmer_posthoc(df = npx_data1,
  olinkid_list = assay_list,
  model_formula = "NPX~Time*Treatment+(1|Subject)",
  effect_formula = "pairwise-Treatment|Time",
  verbose = TRUE)
```

---

```r
olink_normalization

Normalization of all proteins (by OlinkID).

Description

Normalizes NPX data frames to another data frame or to reference medians. If two dataframes are normalized to one another, Olink’s default is using the older dataframe as reference. The function handles three different types of normalization:

- **Bridging normalization:** One of the dataframes is adjusted to another using overlapping samples (bridge samples). The overlapping samples need to be named the same between the dataframes and adjustment is made using the median of the paired differences between the bridge samples in the two data frames. The two dataframes are inputs df1 and df2, the one being adjusted to is specified in the input `reference_project` and the overlapping samples are specified in `overlapping_samples_df1`. Only `overlapping_samples_df1` should be input, no matter which dataframe is used as `reference_project`.

- **Subset normalization:** One of the dataframes is adjusted to another dataframe using a sample subset. Adjustment is made using the differences in median between the subsets from the two dataframes. Both `overlapping_samples_df1` and `overlapping_samples_df2` need to be input. The samples do not need to be named the same.

A special case of subset normalization are to use all samples (except control samples and samples with QC warning) from df1 as input in `overlapping_samples_df1` and all samples from df2 as input in `overlapping_samples_df2`.

```
in overlapping_samples_df2.

Reference median normalization: Working only on one dataframe. This is effectively subset normalization, but using difference of medians to pre-recorded median values. df1, overlapping_samples_df1 and reference_medians need to be specified. Adjustment of df1 is made using the differences in median between the overlapping samples and the reference medians.

Usage

```r
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
)
```

Arguments

- `df1`: First dataframe to be used in normalization (required).
- `df2`: Second dataframe 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 df1.
- `df1_project_nr`: Project name of first dataset.
- `df2_project_nr`: Project name of second dataset.
- `reference_project`: Project name of reference_project. Needs to be the same as either df1_project_nr or df2_project_nr. The project to which the second project is adjusted to.
- `reference_medians`: Dataframe which needs to contain columns "OlinkID", and "Reference_NPX". Used for reference median normalization.

Value

A "tibble" of NPX data in long format containing normalized NPX values, including adjustment factors. Columns include same as df1/df2 with additional column Adj_factor which includes the adjustment factor in the normalization.

Examples

```r
library(dplyr)
```
npx_df1 <- npx_data1 %>% dplyr::mutate(Project = 'P1')
npx_df2 <- npx_data2 %>% dplyr::mutate(Project = 'P2')

# Bridging normalization:
# Find overlapping samples, but exclude Olink control
overlap_samples <- intersect((npx_df1 %>%
  dplyr::filter(!grepl("control", SampleID,
                  ignore.case=TRUE)))$SampleID,
  (npx_df2 %>%
  dplyr::filter(!grepl("control", SampleID,
                  ignore.case=TRUE)))$SampleID)

# 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')

# Subset normalization:
# Find a suitable subset of samples from both projects, but exclude Olink controls
# and samples which do not pass QC.
df1_sampleIDs <- npx_df1 %>%
  dplyr::group_by(SampleID) %>%
  dplyr::filter(all(QC_Warning == 'Pass')) %>%
  dplyr::filter(!stringr::str_detect(SampleID, 'CONTROL_SAMPLE')) %>%
  dplyr::select(SampleID) %>%
  unique() %>%
  dplyr::pull(SampleID)
df2_sampleIDs <- npx_df2 %>%
  dplyr::group_by(SampleID) %>%
  dplyr::filter(all(QC_Warning == 'Pass')) %>%
  dplyr::filter(!stringr::str_detect(SampleID, 'CONTROL_SAMPLE')) %>%
  dplyr::select(SampleID) %>%
  unique() %>%
  dplyr::pull(SampleID)
some_samples_df1 <- sample(df1_sampleIDs, 16)
some_samples_df2 <- sample(df2_sampleIDs, 16)

olink_normalization(df1 = npx_df1,
  df2 = npx_df2, overlapping_samples_df1 = some_samples_df1,
  overlapping_samples_df2 = some_samples_df2)

## Special case of subset normalization when using all samples.
olink_normalization(df1 = npx_df1,
  df2 = npx_df2, overlapping_samples_df1 = df1_sampleIDs,
  overlapping_samples_df2 = df2_sampleIDs)
#Reference median normalization:
# For the sake of this example, set the reference median to 1
ref_median_df <- npx_df1 %>%
  dplyr::select(OlinkID) %>%
  dplyr::distinct() %>%
  dplyr::mutate(Reference_NPX = 1)

# Normalize
olink_normalization(df1 = npx_df1,
  overlapping_samples_df1 = some_samples_df1,
  reference_medians = ref_median_df)

---

olink_normalization_bridge

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

### Description

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

### Usage

```r
olink_normalization_bridge(
  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 = "P1",  # Name of the first project (default: P1).
  project_2_name = "P2",  # Name of the second project (default: P2).
  project_ref_name = "P1"  # 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 ad-
                           # justed to (default: P1).
)
```
Details

This function is a wrapper of olink_normalization.

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

```r
npx_df1 <- npx_data1 |> dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> dplyr::select(-Project) |> dplyr::mutate(Normalization = "Intensity")
npx_df2 <- npx_data2 |> dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> dplyr::select(-Project) |> dplyr::mutate(Normalization = "Intensity")

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

# Normalize
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")
```
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.

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

This function is a wrapper of olink_normalization_bridge and olink_normalization_subset.

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

```r
# prepare datasets
npx_df1 <- npx_data1 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
  dplyr::select(-Project) |>
  dplyr::mutate(Normalization = "Intensity")

npx_df2 <- npx_data2 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
  dplyr::select(-Project) |>
  dplyr::mutate(Normalization = "Intensity")

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

# create tibble for input
norm_schema_bridge <- dplyr::tibble(
  order = c(1, 2),
  name = c("NPX_DF1", "NPX_DF2"),
  data = list("NPX_DF1" = npx_df1,
               "NPX_DF2" = npx_df2)
)
```
"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
olink_normalization_n(norm_schema = norm_schema_bridge)

### Subset normalization of two projects

# datasets
npx_df1 <- npx_data1 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::select(-Project) |>
dplyr::mutate(Normalization = "Intensity")
npx_df2 <- npx_data2 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::select(-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(SampleID, "CONTROL_")) |>
dplyr::group_by(SampleID) |>
dplyr::filter(all(QC_Warning == "Pass")) |>
dplyr::pull(SampleID) |>
unique() |>
sample(size = 16, replace = FALSE)
df2_samples <- npx_df2 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::group_by(SampleID) |>
dplyr::filter(all(QC_Warning == "Pass")) |>
dplyr::pull(SampleID) |>
unique() |>
sample(size = 16, 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(1, 2),
  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

```r
olink_normalization_n(norm_schema = norm_schema_subset)
```

### Subset normalization of two projects using all samples

# datasets

```r
npx_df1 <- npx_data1 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::select(-Project) |>
dplyr::mutate(Normalization = "Intensity")
npx_df2 <- npx_data2 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::select(-Project) |>
dplyr::mutate(Normalization = "Intensity")
```

# Find a suitable subset of samples from both projects, but exclude Olink
# controls and samples that fail QC.

```r
df1_samples_all <- npx_df1 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::group_by(SampleID) |>
dplyr::filter(all(QC_Warning == "Pass")) |>
dplyr::pull(SampleID) |>
unique()
df2_samples_all <- npx_df2 |>
dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |>
dplyr::group_by(SampleID) |>
dplyr::filter(all(QC_Warning == "Pass")) |>
dplyr::pull(SampleID) |>
unique()
```

# create named list

```r
subset_samples_all_list <- list("DF1" = df1_samples_all,
                                "DF2" = df2_samples_all)
```

# create tibble for input

```r
norm_schema_subset_all <- dplyr::tibble(
    order = c(1, 2),
    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

```r
olink_normalization_n(norm_schema = norm_schema_subset_all)
```

### Multi-project normalization using bridge and subset samples
## NPX data frames to bridge

```r
npx_df1 <- npx_data1 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-Project) |> 
  dplyr::mutate(Normalization = "Intensity")

npx_df2 <- npx_data2 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-Project) |> 
  dplyr::mutate(Normalization = "Intensity")
```

# manipulating the sample NPX datasets to create another two random ones

```r
npx_df3 <- npx_data2 |> 
  dplyr::mutate(SampleID = paste(SampleID, "_mod", sep = ""), 
    PlateID = paste(PlateID, "_mod", sep = ""), 
    NPX = sample(x = NPX, size = dplyr::n(), replace = FALSE)) |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-Project) |> 
  dplyr::mutate(Normalization = "Intensity")

npx_df4 <- npx_data1 |> 
  dplyr::mutate(SampleID = paste(SampleID, "_mod2", sep = ""), 
    PlateID = paste(PlateID, "_mod2", sep = ""), 
    NPX = sample(x = NPX, size = dplyr::n(), replace = FALSE)) |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-Project) |> 
  dplyr::mutate(Normalization = "Intensity")
```

## samples to use for normalization

# Bridge samples with same identifiers between npx_df1 and npx_df2

```r
overlap_samples <- dplyr::intersect(unique(npx_df1$SampleID), 
  unique(npx_df2$SampleID))
overlap_samples_df1_df2 <- list("DF1" = overlap_samples, 
  "DF2" = overlap_samples)
```

# samples to use for normalization

# Bridge samples with different identifiers between npx_df2 and npx_df3

```r
overlap_samples_df2_df3 <- list("DF1" = sample(x = unique(npx_df2$SampleID), 
  size = 10, 
  replace = FALSE), 
  "DF2" = sample(x = unique(npx_df3$SampleID), 
  size = 10, 
  replace = FALSE))
```

# Samples to use for intensity normalization between npx_df4 and the
# normalized dataset of npx_df1 and npx_df2

```r
overlap_samples_df12_df4 <- list("DF1" = sample(x = c(unique(npx_df1$SampleID), 
  unique(npx_df2$SampleID)), 
  size = 100, 
  replace = FALSE), 
  "DF2" = sample(x = unique(npx_df4$SampleID), 
  size = 40, 
  replace = FALSE))
```
# create tibble for input
norm_schema_n <- dplyr::tibble(
    order = c(1, 2, 3, 4),
    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")
)

olink_normalization_n(norm_schema = norm_schema_n)

olink_normalization_n_check

An internal function to perform checks on the input of the function olink_normalization_n.

Description

An internal function to perform checks on the input of the function olink_normalization_n.

Usage

olink_normalization_n_check(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 above for
details of the structure of the data frame. See details in help for olink_normalization_n.
(required)

Value

a character message. If the message is "TRUE" then all checks passed, otherwise an error message
will be printed.
olink_normalization_project_name_check

An internal function to perform checks on the input project names in the functions olink_normalization_bridge and olink_normalization_subset. The function is expected to run all checks on project names to make sure that normalization can be performed smoothly. It should work independently of the function calling it.

Description

An internal function to perform checks on the input project names in the functions olink_normalization_bridge and olink_normalization_subset. The function is expected to run all checks on project names to make sure that normalization can be performed smoothly. It should work independently of the function calling it.

Usage

olink_normalization_project_name_check(
    project_1_name,
    project_2_name,
    project_ref_name
)

Arguments

project_1_name Name of project 1 (required)
project_2_name Name of project 2 (required)
project_ref_name Name of reference project (required)

Value

a character message. If the message is "TRUE" then all checks passed, otherwise an error message will be printed.

olink_normalization_sample_check

An internal function to perform checks on the input samples in the functions olink_normalization_bridge and olink_normalization_subset. The function is expected to run all checks on SampleID to make sure that normalization can be performed smoothly. It should work independently of the function calling it.
olink_normalization_subset

Description

An internal function to perform checks on the input samples in the functions olink_normalization_bridge and olink_normalization_subset. The function is expected to run all checks on SampleID to make sure that normalization can be performed smoothly. It should work independently of the function calling it.

Usage

olink_normalization_sample_check(
  list_samples,
  check_mode,
  project_1_all_samples,
  project_2_all_samples
)

Arguments

list_samples Named list of 2 arrays containing SampleID of the subset or bridge samples to be used for normalization. The names of the two arrays should be DF1 and DF2 corresponding to projects 1 and 2, respectively. (required)
check_mode Flag "bridge" or "subset" indicating the type of normalization the check should be tailored to (required)
project_1_all_samples Array of all samples from project 1 (required)
project_2_all_samples Array of all samples from project 2 (required)

Value

a character message. If the message is "TRUE" then all checks passed, otherwise an error message will be printed.

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.
Usage

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

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).

Details

This function is a wrapper of `olink_normalization`.

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

```r
# datasets
npx_df1 <- npx_data1 |> dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> dplyr::select(-Project) |> dplyr::mutate(Normalization = "Intensity")
npx_df2 <- npx_data2 |> dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> dplyr::select(-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(SampleID, "CONTROL_")) |> dplyr::group_by(SampleID) |> dplyr::filter(all(QC_Warning == "Pass")) |> dplyr::pull(SampleID) |> unique() |> sample(size = 16, replace = FALSE)
df2_samples <- npx_df2 |> dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> dplyr::group_by(SampleID) |> dplyr::filter(all(QC_Warning == "Pass")) |> dplyr::pull(SampleID) |> unique() |> sample(size = 16, replace = FALSE)

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

# Normalize
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")

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

# datasets
npx_df1 <- npx_data1 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-Project) |> 
  dplyr::mutate(Normalization = "Intensity")

npx_df2 <- npx_data2 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::select(-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(SampleID, "CONTROL_")) |> 
  dplyr::group_by(SampleID) |> 
  dplyr::filter(all(QC_Warning == 'Pass')) |> 
  dplyr::pull(SampleID) |> 
  unique()

df2_samples_all <- npx_df2 |> 
  dplyr::filter(!stringr::str_detect(SampleID, "CONTROL_")) |> 
  dplyr::group_by(SampleID) |> 
  dplyr::filter(all(QC_Warning == 'Pass')) |> 
  dplyr::pull(SampleID) |> 
  unique()

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

# Normalize
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")

---

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,
  variable,
  dependence = FALSE,
  subject = 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.
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

library(dplyr)

# One-way Kruskal-Wallis Test
try({ # May fail if dependencies are not installed
  kruskal_results <- olink_one_non_parametric(df = npx_data1,
                                             variable = "Site")
})

# Friedman Test
friedman_results <- olink_one_non_parametric(df = npx_data1,
                                           variable = "Time前",
                                           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,
  olinkid_list = NULL,
  variable,
  test = "kruskal",
  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.
- **test**: Single character value indicates running the post hoc test for friedman or kruskal.
- **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

```r
library(dplyr)
try({
  # May fail if dependencies are not installed
  # One-way Kruskal-Wallis Test
  kruskal_results <- olink_one_non_parametric(df = npx_data1,
                                            variable = "Site")
})

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

#Posthoc test for the results from Friedman Test
friedman_posthoc_results <- olink_one_non_parametric_posthoc(npx_data1,
                                                              variable = "Time",
                                                              test = "friedman",
                                                              verbose = TRUE)
```
Function which 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.

Description

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

Usage

olinkOrdinalRegression(
  df,
  variable,
  covariates = NULL,
  return.covariates = F,
  verbose = T
)
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.

- **verbose**
  - Logical. 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 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

```
library(dplyr)

try({
  # May fail if dependencies are not installed.
  # Two-way Ordinal Regression with CLM.
  # Results in model NPX~Treatment+Time+Treatment:Time.
  ordinalRegression_results <- olink_ordinalRegression(df = npx_data1,
                                                       variable="Treatment:Time")
})
```
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

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

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.

- **posthoc_adjust_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.

### 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

```r
library(dplyr)
#Two-way Ordinal Regression.
#Results in model NPX~Treatment*Time.
try({ # May not work if dependencies are not installed.
  ordinalRegression_results <- olink_ordinalRegression(df = npx_data1,
 variable="Treatment:Time")

  #Filtering out significant and relevant results.
  significant_assays <- ordinalRegression_results %>%
    filter(Threshold == 'Significant' & term == 'Time') %>%
    select(OlinkID) %>%
    distinct() %>%
    pull()

  #Posthoc test for the model NPX~Treatment*Time,
  #on the effect Time.
  #Posthoc
})
```
ordinalRegression_results_posthoc_results <- olink_ordinalRegression_posthoc(npx_data1,  
  variable=c("Treatment:Time"),  
  olinkid_list = significant_assays,  
  effect = "Time")

### olink_pal

**Olink color panel for plotting**

**Description**

Olink color panel for plotting

**Usage**

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

**Arguments**

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

**Value**

A character vector of palette hex codes for colors

**Examples**

```r
library(scales)

#Color matrices
show_col(olink_pal()(10), labels = FALSE)
show_col(olink_pal(coloroption = c('lightblue', 'green'))(2), labels = FALSE)

#Contour plot
filled.contour(volcano, color.palette = olink_pal(), asp = 1)
filled.contour(volcano, color.palette = 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 gsea and enrich functions for MSigDB.

Usage

olink_pathway_enrichment(
  data,  # NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt,SampleID, QC_Warning, NPX, and LOD
  test_results,  # a dataframe of statistical test results including Adjusted_pval and estimate columns.
  method = "GSEA",  # Either "GSEA" (default) or "ORA"
  ontology = "MSigDb",  # MSigDb contains C2 and C5 genesets. C2 and C5 encompass KEGG, GO, and Reactome.
  organism = "human",  # Either "human" (default) or "mouse"
  pvalue_cutoff = 0.05,  # (numeric) maximum Adjusted p-value cutoff for ORA filtering of foreground set (default = 0.05). This argument is not used for GSEA.
  estimate_cutoff = 0  # (numeric) minimum estimate cutoff for ORA filtering of foreground set (default = 0) This argument is not used for GSEA.
)

Arguments

data  # NPX data frame in long format with at least protein name (Assay), OlinkID, UniProt,SampleID, QC_Warning, NPX, and LOD

test_results  # a dataframe of statistical test results including Adjusted_pval and estimate columns.

method  # Either "GSEA" (default) or "ORA"

ontology  # Supports "MSigDb" (default), "KEGG", "GO", and "Reactome" as arguments. MSigDB contains C2 and C5 genesets. C2 and C5 encompass KEGG, GO, and Reactome.

organism  # Either "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. test_results must contain estimates for all assays. Posthoc results can be used but should be filtered for one contrast to improve interpretability. 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 needed for ORA. Any statistical results that contains one estimate per protein will work as long as the estimates are comparable to each other.
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.

A few notes on 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. Pathways can only be 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 more well suited for data on the Explore platform or on multiple Target 96 panels. For smaller lists of proteins it may be more informative to use biological annotation in directed research, to discover which significant assay are related to keywords of interest.

Value

A data frame of enrichment results. Columns for ORA include:

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

Columns for GSEA:

- ID: "character" Pathway ID from MSigDB
- Description: "character" Description of Pathway from MSigDB
- setSize: "integer" ratio of input proteins that are annotated in a term
- enrichmentScore: "numeric" Enrichment score, degree to which a gene set is over-represented at the top or bottom of the ranked list of genes
- NES: "numeric" Normalized Enrichment Score, 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: "numeric" p-value of enrichment
• p.adjust: "numeric" Adjusted p-value (Benjamini-Hochberg)
• qvalue: "numeric" false discovery rate, the estimated probability that the normalized enrichment score represents a false positive finding
• rank: "numeric" the position in the ranked list where the maximum enrichment score occurred
• leading_edge: "character" contains tags, list, and signal. Tags gives an indication of the percentage of genes contributing to the enrichment score. List gives an indication of where in the list the enrichment score is obtained. Signal represents the enrichment signal strength and combines the tag and list.
• core_enrichment: "character" list of input proteins (Gene Symbols) annotated in a term delimited by "/"

See Also

• olink_pathway_heatmap for generating a heat map of results
• olink_pathway_visualization for generating a bar graph of results

Examples

```r
library(dplyr)
npx_df <- npx_data1 %>% filter(!grepl("control", SampleID, ignore.case = TRUE))
ttest_results <- olink_ttest(
  df = npx_df,
  variable = "Treatment",
  alternative = "two.sided"
)
try({ # This expression might fail if dependencies are not installed
gsea_results <- olink_pathway_enrichment(data = npx_data1, test_results = ttest_results)
ora_results <- olink_pathway_enrichment(
  data = npx_data1,
  test_results = ttest_results, method = "ORA"
)
}, silent = TRUE)
```

olink_pathway_heatmap  Creates a heatmap of selected pathways and proteins

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 = 20
)

Arguments

enrich_results data frame of enrichment results from olink_pathway_enrichment()

test_results filtered results from statistical test with Assay, OlinkID, and estimate columns

method method used in olink_pathway_enrichment ("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

See Also

• olink_pathway_enrichment for generating enrichment results

• olink_pathway_visualization for generating a bar graph of results

Examples

library(dplyr)
# Run t-test results (see olink_ttest documentation)
npx_df <- npx_data1 %>% filter(!grepl('control',SampleID, ignore.case = TRUE))
ttest_results <- olink_ttest(df=npx_df,
  variable = 'Treatment',
  alternative = 'two.sided')

try({
  # This expression might fail if dependencies are not installed
  # Run olink_pathway_enrichment (see documentation)
  gsea_results <- olink_pathway_enrichment(data = npx_data1, test_results = ttest_results)
  ora_results <- olink_pathway_enrichment(data = npx_data1,
    test_results = ttest_results, method = "ORA")
  olink_pathway_heatmap(enrich_results = gsea_results, test_results = ttest_results)
  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 = 20
)

Arguments

enrich_results  data frame of enrichment results from olink_pathway_enrichment()
method               method used in olink_pathway_enrichment ("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

See Also

• olink_pathway_enrichment for generating enrichment results
• olink_pathway_heatmap for generating a heat map of results

Examples

library(dplyr)
# Run olink_ttest or other stats test (see documentation)
npx_df <- npx_data1 %>% filter(!grepl('control',SampleID, ignore.case = TRUE))
ttest_results <- olink_ttest(df=npx_df,
  variable = 'Treatment',
try({
  # This expression might fail if dependencies are not installed
  # Run olink_pathway_enrichment (see documentation)
  gsea_results <- olink_pathway_enrichment(data = npx_data1, test_results = ttest_results)
  ora_results <- olink_pathway_enrichment(data = npx_data1,
                                          test_results = ttest_results, method = "ORA")
  olink_pathway_visualization(enrich_results = gsea_results)
  olink_pathway_visualization(enrich_results = gsea_results, keyword = "immune")
  olink_pathway_visualization(enrich_results = ora_results, method = "ORA", number_of_terms = 15)
})

### 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.

**Usage**

```r
olink_pca_plot(
  df,
  color_g = "QC_Warning",
  x_val = 1,
  y_val = 2,
  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

color_g  Character value indicating which column to use for colors (default QC_Warning)

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 +/-outlierDef[X,Y] 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 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% for single
plate projects. 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 +/-outlierDef[X,Y] standard deviations from the mean of the plotted PC will be labelled.
Both arguments have to be specified.
Value

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

Examples

```r
library(dplyr)
npx_data <- npx_data1 %>%
  filter(!grepl('CONTROL', SampleID))

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

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

#Label outliers
olink_pca_plot(df=npx_data, color_g = "QC_Warning",
  outlierDefX = 2, outlierDefY = 4) #All data
olink_pca_plot(df=npx_data, color_g = "QC_Warning",
  outlierDefX = 2.5, outlierDefY = 4, byPanel = TRUE) #Per panel

#Retrieve the outliers
outliers <- lapply(g, function(x){x$data}) %>%
  bind_rows() %>%
  filter(Outlier == 1)
```

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.

Usage

```r
olink_plate_randomizer(
  Manifest,
  PlateSize = 96,
  Product,
  SubjectColumn,
  iterations = 500,
  available.spots,
  num_ctrl = 8,
```
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.
um_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.

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)

# 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")

# 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 +/- IQR_outlierDef standard deviations from the mean IQR (default 3). Vertical dashed lines indicate +/- median_outlierDef standard deviations from the mean sample median (default 3).

Usage

```r
olink_qc_plot(
  df,
  color_g = "QC_Warning",
  plot_index = FALSE,
  label_outliers = TRUE,
  IQR_outlierDef = 3,
  median_outlierDef = 3,
  outlierLines = TRUE,
  facetNrow = NULL,
  facetNcol = NULL,
  ...
)
```
Arguments

- **df**: NPX data frame in long format. Must have columns SampleID, NPX and Panel
- **color_g**: Character value indicating which column to use as fill color (default QC_Warning)
- **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.
- **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 +/-IQR_outlierDef and +/-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

```r
library(dplyr)

olink_qc_plot(npx_data1, color_g = "QC_Warning")

# Change the outlier threshold to +/-4SD
olink_qc_plot(npx_data1, color_g = "QC_Warning", IQR_outlierDef = 4, median_outlierDef = 4)

# Identify the outliers
qc <- olink_qc_plot(npx_data1, color_g = "QC_Warning", IQR_outlierDef = 4, median_outlierDef = 4)
outliers <- qc$data %>% filter(Outlier == 1)
```

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

```r
olink_ttest(df, variable, pair_id, ...)
```

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.

...  
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

```r
library(dplyr)

npx_df <- npx_data1 %>% filter(!grepl('control', SampleID, ignore.case = TRUE))

ttest_results <- olink_ttest(df=npx_df,
```
# Paired t-test

```r
npx_df %>%
  filter(Time %in% c("Baseline","Week.6")) %>%
  olink_ttest(variable = "Time", pair_id = "Subject")
```

### 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% for multi-plate projects and <5% for single plate projects.

#### Usage

```r
olink_umap_plot(
  df,
  color_g = "QC_Warning",
  x_val = 1,
  y_val = 2,
  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)
- **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)
- **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 +/-outlierDef[X,Y] 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 +/-outlierDef[X,Y] 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

library(dplyr)
npx_data <- npx_data1 %>%
  mutate(SampleID = paste(SampleID, "_", Index, sep = ""))
try({ # Requires umap package dependency
  #UMAP using all the data
  olink_umap_plot(df=npx_data, color_g = "QC_Warning")
  #UMAP per panel
})
g <- olink_umap_plot(df=npx_data, color_g = "QC_Warning", byPanel = TRUE)
g$Inflammation #Plot only the Inflammation panel

#Label outliers
olink_umap_plot(df=npx_data, color_g = "QC_Warning",
    outlierDefX = 2, outlierDefY = 4) #All data
olink_umap_plot(df=npx_data, color_g = "QC_Warning",
    outlierDefX = 3, outlierDefY = 2, byPanel = TRUE) #Per panel

#Retrieve the outliers
outliers <- lapply(g, function(x){x$data}) %>%
    bind_rows() %>%
    filter(Outlier == 1)
}

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

```r
library(dplyr)

npx_df <- npx_data1 %>% filter(!grepl('control',SampleID, ignore.case = TRUE))
ttest_results <- olink_ttest(df=npx_df,
    variable = 'Treatment',
    alternative = 'two.sided')
olink_volcano_plot(ttest_results)
```

```r
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

```r
olink_wilcox(df, variable, pair_id, ...)
```

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.
- `...` : 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

library(dplyr)
npx_df <- npx_data1 %>% filter(!grepl('/quotesingle.Varcontrol/SampleID, ignore.case = TRUE))
wilcox_results <- olink_wilcox(df = npx_df,
variable = 'Treatment',
alternative = 'two.sided')

#Paired Mann-Whitney U Test
npx_df %>%
  filter(Time %in% c("Baseline","Week.6")) %>%
  olink_wilcox(variable = "Time", pair_id = "Subject")

print_and_capture

Capture the output of printing an object

Description

Capture the output of printing an object

Usage

print_and_capture(x)

Arguments

x printable object

Value

string representation of the provided object
### read_flex

**Read in flex data**

**Examples**

```r
OlinkAnalyze:::print_and_capture(npx_data1)
```

**Description**

Called by read_NPX

**Usage**

```r
read_flex(filename)
```

**Arguments**

- `filename` where the file is located

**Value**

tibble of data

### read_NPX

**Function to read NPX data into long format**

**Description**

Imports an NPX or QUANT file exported from Olink Software. No alterations to the output format is allowed.

**Usage**

```r
read_NPX(filename)
```

**Arguments**

- `filename` Path to Olink Software output file.
A "tibble" in long format. Columns include:

- SampleID: Sample ID
- Index: Index
- OlinkID: Olink ID
- UniProt: UniProt ID
- Assay: Protein symbol
- MissingFreq: Proportion of sample below LOD
- Panel_Version: Panel Version
- PlateID: Plate ID
- QC_Warning: QC Warning Status
- LOD: Limit of detection
- NPX: Normalized Protein Expression

Additional columns may be present or missing depending on the platform

Examples

```r
file <- system.file("extdata", "Example_NPX_Data.csv", package = "OlinkAnalyze")
read_NPX(file)
```

**Description**

Helper function to read in Olink Explore csv or txt files

**Usage**

```r
read_npx_csv(filename)
```

**Arguments**

- `filename` Path to Olink Software output txt of csv file.
Value

A "tibble" in long format. Some of the columns are:

- SampleID: Sample ID
- Index: Index
- OlinkID: Olink ID
- UniProt: UniProt ID
- Assay: Protein symbol
- MissingFreq: Proportion of sample below LOD
- Panel_Version: Panel Version
- PlateID: Plate ID
- QC_Warning: QC Warning Status
- LOD: Limit of detection
- NPX: Normalized Protein Expression

Additional columns may be present or missing depending on the platform

Examples

```r
file <- system.file("extdata", "Example_NPX_Data.csv", package = "OlinkAnalyze")
read_NPX(file)
```

Description

Helper function to read in Olink Explore parquet output files

Usage

```r
read_npx_parquet(filename)
```

Arguments

- `filename` : Path to Olink Software parquet output file.
Value

A "tibble" in long format. Some of the columns are:

• SampleID: Sample ID
• OlinkID: Olink ID
• UniProt: UniProt ID
• Assay: Protein symbol
• PlateID: Plate ID
• Count: Counts from sequences
• ExtNPX: External control normalized counts
• NPX: Normalized Protein Expression

Additional columns may be present or missing depending on the platform

Examples

```r
file <- system.file("extdata", "Example_NPX_Data.csv", package = "OlinkAnalyze")
read_NPX(file)
```

---

**read_npx_zip**

*Helper function to read in Olink Explore zip csv files*

**Description**

Helper function to read in Olink Explore zip csv files

**Usage**

```r
read_npx_zip(filename)
```

**Arguments**

- `filename` Path to Olink Software output zip file.

**Value**

A "tibble" in long format. Some of the columns are:

• SampleID: Sample ID
• Index: Index
• OlinkID: Olink ID
• UniProt: UniProt ID
set_plot_theme

- Assay: Protein symbol
- MissingFreq: Proportion of sample below LOD
- Panel_Version: Panel Version
- PlateID: Plate ID
- QC.Warning: QC Warning Status
- LOD: Limit of detection
- NPX: Normalized Protein Expression

Additional columns may be present or missing depending on the platform

Examples

```r
try({
  # May fail if dependencies are not installed
  file <- system.file("extdata", "Example_NPX_Data.csv", package = "OlinkAnalyze")
  read_NPX(file)
})
```

---

| set_plot_theme | Function to set plot theme |

Description

This function sets a coherent plot theme for functions.

Usage

```r
set_plot_theme(font = "Swedish Gothic Thin")
```

Arguments

- **font**: Font family to use for text elements. Depends on extrafont package.

Value

No return value, used as theme for ggplots
Examples

```r
library(ggplot2)

ggplot(mtcars, aes(x = wt, y = mpg, color = as.factor(cyl))) +
  geom_point(size = 4) +
  set_plot_theme()

ggplot(mtcars, aes(x = wt, y = mpg, color = as.factor(cyl))) +
  geom_point(size = 4) +
  set_plot_theme(font = "")
```
Index

* Bridge
  olink_normalization_bridge, 29
  olink_normalization_n, 31
* Heatmap
  olink_heatmap_plot, 18
* NPX
  olink_dist_plot, 16
  olink_heatmap_plot, 18
  olink_pca_plot, 55
  olink_qc_plot, 59
  olink_umap_plot, 62
  read_NPX, 67
  read_npx_csv, 68
  read_npx_parquet, 69
  read_npx_zip, 70
* Normalization;
  olink_normalization_bridge, 29
  olink_normalization_n, 31
  olink_normalization_subset, 38
* Normalization
  olink_normalization, 26
* Olink
  olink_pal, 49
* PCA
  olink_pca_plot, 55
* Subset
  olink_normalization_n, 31
  olink_normalization_subset, 38
* UMAP
  olink_umap_plot, 62
* color
  olink_pal, 49
* datasets
  manifest, 4
  npx_data1, 5
  npx_data2, 5
* ggplot
  olink_displayPlateDistributions, 14
  olink_displayPlateLayout, 15
* normalization;
  olink_normalization_n, 31
* normalization
  olink_normalization_bridge, 29
  olink_normalization_subset, 38
* palette
  olink_pal, 49
* plates
  olink_displayPlateDistributions, 14
  olink_displayPlateLayout, 15
  olink_plate_randomizer, 57
* randomized
  olink_displayPlateDistributions, 14
  olink_displayPlateLayout, 15
  olink_plate_randomizer, 57
check_data_completeness, 3
manifest, 4
npx_data1, 5
npx_data2, 5
olink_anova, 6
olink_anova_posthoc, 8
olink_boxplot, 11
olink_bridgeselector, 12
olink_color_discrete, 13
olink_color_gradient, 14
olink_displayPlateDistributions, 14
olink_displayPlateDistributions(), 16, 58
olink_displayPlateLayout, 15
olink_displayPlateLayout(), 15, 58
olink_dist_plot, 16
olink_fill_discrete, 17
olink_fill_gradient, 18
olink_heatmap_plot, 18
olink_lmer, 20
olink_lmer_plot, 22
olink_lmer_posthoc, 24
olink_normalization, 26
olink_normalization_bridge, 29
olink_normalization_n, 31
olink_normalization_n_check, 36
olink_normalization_project_name_check, 37
olink_normalization_sample_check, 37
olink_normalization_subset, 38
olink_one_non_parametric, 41
olink_one_non_parametric_posthoc, 43
olink_ordinalRegression, 45
olink_ordinalRegression_posthoc, 47
olink_pal, 49
olink_pathway_enrichment, 50, 53, 54
olink_pathway_heatmap, 52, 52, 54
olink_pathway_visualization, 52, 53, 54
olink_pca_plot, 55
olink_plate_randomizer, 57
olink_plate_randomizer(), 15, 16
olink_qc_plot, 59
olink_ttest, 60
olink_umap_plot, 62
olink_volcano_plot, 64
olink_wilcox, 65
print_and_capture, 66

read_flex, 67
read_NPX, 67
read_npx_csv, 68
read_npx_parquet, 69
read_npx_zip, 70

set_plot_theme, 71