# Package 'MetaPipeX'

December 19, 2022

**Type** Package

**Title** Standardizing data structure, analysis code & reporting for experimental data of between groups comparisons in replication projects

**Version** 0.1.0

**Author** Jens Fuenderich

**Maintainer** Jens Fuenderich <Jens.Fuenderich@uni-erfurt.de>

**Description** The MetaPipeX framework is a reaction to the diversity in reporting standards and data formats in multi-lab replication projects. It serves as a proposal to standardize the data structure, analysis code and reporting for experimental data of between groups comparisons in replication projects. MetaPipeX consists of three components: A descriptive pipeline for data transformations and analyses, analysis functions that implement the pipeline and a Shiny App that utilizes the standardized structure to allow various insights into the data produced by the pipeline. While the framework maps most easily to direct replications, its scope may be broadened to conceptual replications and similar projects. Illustrations of the pipeline and the functions are available on github in the Graphics folder: https://github.com/JensFuenderich/MetaPipeX/tree/main/Supplementary_Material The MetaPipeX R-package provides a readily available set of functions to run the standardized part of the pipeline. The package includes three analysis functions that each represent a step in the MetaPipeX pipeline (create_replication_summaries, merge_replication_summaries and meta_analyses) and a fourth that runs the pipeline, starting at person level data (full_pipeline). Further, it includes a function that runs the Shiny App (MetaPipeX::ShinyApp). All meta-analyses use metafor::rma.mv (Viechtbauer, 2010).

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.2

**Imports** dplyr,
DT,
ggplot2,
grDevices,
haven,
janitor,
magrittr,
mathjaxr,

metafor,
puniform,
readr,
shiny,
shinythemes,
shinyWidgets,
stats

**Remotes** RobbievanAert/puniform

**RdMacros** mathjaxr

# R topics documented:



---

`create_replication_summaries`
*Creating Replication Summaries*

---

## Description

\( \let\underscore_ \)

Function to compute replication aggregates from person level data. Components of the standardized mean difference and their standard errors are calculated and reported. This is the first function (and the third computational step) of the MetaPipeX pipeline. For more details on the replication statistics, refer to the Details section. For more details on the pipeline, refer to the documentation of the MetaPipeX-package.

## Usage

```
create_replication_summaries(
  data,
  MultiLab = NULL,
  ReplicationProject = NULL,
  Replication = NULL,
  DV = NULL,
  Group = NULL,
  output_folder = NULL,
  suppress_list_output = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame or list of data frames that contain the individual participant data. The function expects the relevant columns to be named consistently across all list objects. Relevant to this function are columns that represent information on the MultiLab (e.g., Many Labs 2), the ReplicationProject (e.g., Ross1), the Replication (the lab a data point is assigned to), the group (either the treatment or control condition) and the single data point of the dependent variable (DV) per person. A template of this data frame is available on github, as is a codebook for unambiguous identification of the abbreviations. |
| `MultiLab` | Character vector with the name of the columns in the list elements of "data" that contain the multi-lab name(s). If *is.null(MultiLab) == TRUE*, "MultiLab" is chosen as the default. |
| `ReplicationProject` | Character vector with the name of the columns in the list elements of "data" that contain the replication project name(s). If *is.null(ReplicationProject) == TRUE*, "ReplicationProject" is chosen as the default. Each replication project comprises a single target-effect with direct replications across multiple replications (/labs). |
| `Replication` | Character vector with the name of the columns in the list elements of "data" that contain the replication names. If *is.null(Replication) == TRUE*, "Replication" is chosen as the default. The meta-analyses in MetaPipeX::meta_analyses() and MetaPipeX::full_pipeline() are run as random effects models in metafor::rma.mv() with "random = ~ 1 \| Replication". Thus, the pipeline assumes a distribution of true statistics (e.g., treatment means, mean differences, standardized mean differences). |
| `DV` | Character vector with the name of the columns in the list elements of "data" that contain the (aggregated) dependent variable. If *is.null(DV) == TRUE*, "DV" is chosen as the default. |
| `Group` | Character vector with the name of the columns in the list elements of "data" that contain the (treatment/control) group identification. If *is.null(Group) == TRUE*, "Group" is chosen as the default. These should only contain values of 0 (control group), 1 (treatment group) and NA (unidentified). |
| `output_folder` | Specify the output folder for the summaries and the codebook. If no folder is specified, the function will return its output only to the R environment (unless this is suppressed under suppress_list_output). |
| `suppress_list_output` | Logical. FALSE by default. If FALSE, the function will return a list output to the environment, containing the replication summaries and the codebook. If TRUE, these are not returned to the environment. |

## Details

### Replication Statistics

All components of the standardized mean difference and their standard errors are returned by the function. Each standard error is returned to enable a meta-analysis on each component. The components and their standard errors are implemented as follows. Unless other sources are provided, effect size statistics are calculated according to Borenstein et al., 2009. The metafor::escalc function was used for SMD and MD (Viechtbauer, 2010).
## mean (M)

- R-Code
  ## apply the function

```
# treatment group mean (T_M):
mean(treatment_group$DV)
# control group mean (C_M):
mean(control_group$DV)
```

- Model

  treatment group mean (T_M):

$$\bar{x}_T = \frac{1}{n} \sum_{i \in T} x$$

  control group mean (C_M):

$$\bar{x}_C = \frac{1}{n} \sum_{i \in C} x$$

## standard error of the mean (SE_T_M, SE_C_M)

- R-Code

```
## define the function
SE_of_mean_fct <- function(x){
estimated_sd <- sqrt(sum((x-mean(x))^2)/(length(x)-1))
SE_of_mean <- sd(x) / sqrt(length(x))
return(SE_of_mean)}

## apply the function
# standard error of treatment group mean (SE_T_M):
SE_of_mean_fct(treatment_group$DV)
# standard error of control group mean (SE_C_M):
SE_of_mean_fct(control_group$DV)
```

- Model

$$\hat{\sigma}_{\bar{x}} = \frac{\hat{\sigma}_x}{\sqrt{n}} = \sqrt{\frac{\frac{1}{n-1} \sum_{i=1}^{n} (x - \bar{x})^2}{n}}$$

## standard deviation (T_SD, C_SD)

- R-Code

```
## apply the function
# treatment group standard deviation (T_SD):
sd(treatment_group$DV)
# control group standard deviation (C_SD):
sd(control_group$DV)
```

- Model

$$\hat{\sigma} = \sqrt{\frac{\sum (x - \bar{x}^2)}{n - 1}}$$

## standard error of the standard deviation (SE_T_SD, SE_C_SD)

- R-Code

```
## define the function
SE_SD_fct <- function(x){
SE_SD <- sd(x) / sqrt(2*(length(x)-1)) # for large n
return(SE_SD) }

## apply the function
# standard error of the treatment group standard deviation (SE_T_SD):
SE_SD_fct(treatment_group$DV)
# standard error of the control group standard deviation (SE_C_SD):
SE_SD_fct(control_group$DV)
```

- Model

$$\hat{\sigma}_{\hat{\sigma}} = \frac{\hat{\sigma}_x}{\sqrt{2(n-1)}} = \sqrt{\frac{\frac{1}{n-1}\sum_{i=1}^{n}(x-\bar{x})^2}{2(n-1)}}$$

$\hat{\sigma}_{\hat{\sigma}}$ is a simplified version of $\sigma_{K_n S}$ in Ahn & Fessler (2003). The authors demonstrate that for n > 10 it is reasonable to use Kn = 1. As for the overwhelming majority of samples n > k may be assumed, we excluded the term $K_n$.

## mean difference (MD)

- R-Code

```
## apply the function
metafor::escalc(
measure = "MD",
m1i = mean(treatment_group$DV),
m2i = mean(control_group$DV),
sd1i = sd(treatment_group$DV),
sd2i = sd(control_group$DV),
n1i = length(treatment_group$DV),
n2i = length(control_group$DV),
vtype = "HO" # assuming homoscedasticity
)$yi
```

- Model

$$D = \bar{x}_T - \bar{x}_C$$

## standard error of mean difference (SE_MD)

- R-Code

```
## apply the function
metafor::escalc(
measure = "MD",
m1i = mean(treatment_group$DV),
m2i = mean(control_group$DV),
sd1i = sd(treatment_group$DV),
sd2i = sd(control_group$DV),
n1i = length(treatment_group$DV),
n2i = length(control_group$DV),
```

```
vtype = "HO" # assuming homoscedasticity
)$vi
```

- Model

$$\hat{\sigma}_{\bar{x}_T-\bar{x}_C} = \sqrt{\frac{n_T+n_C}{n_T n_C}\sigma^2_{TC}} = \sqrt{\frac{n_T+n_C}{n_T n_C}\frac{\sum_{i=1}^{n}(x_T-\bar{x}_T)^2+\sum_{i=1}^{n}(x_C-\bar{x}_C)^2}{n_T+n_C-2}}$$

## pooled standard deviation (pooled_SD)

- R-Code

```
## define the function
pooled_SD_fct <- function(t,c){
pooled_SD <- sqrt((
(sum((t-mean(t))^2))+ # sample sum of squares sums treatment group
(sum((c-mean(c))^2)) # sample sum of squares control group)/
(length(t) + length(c) -2) # n+n-2
) # end of sqrt
return(pooled_SD)}
## apply the function
pooled_SD_fct(treatment_group$DV, control_group$DV)
```

- Model

$$\hat{\sigma}_{TC} = \sqrt{\frac{\sum_{i=1}^{n}(x_T-\bar{x}_T)^2+\sum_{i=1}^{n}(x_C-\bar{x}_C)^2}{n_T+n_C-2}}$$

## standard error of pooled standard deviation (SE_pooled_SD)

- R-Code

```
## define the function
SE_pooled_SD_fct <- function(t,c){
pooled_SD <- sqrt((
(sum((t-mean(t))^2))+ # sample sum of squares sums treatment group
(sum((c-mean(c))^2)) # sample sum of squares control group)/
(length(t) + length(c) -2) # n+n-2
) # end of sqrt
SE_pooled_SD <- pooled_SD/sqrt(2*(length(t)+length(c)-1))
return(SE_pooled_SD)}
## apply the function
SE_pooled_SD_fct(treatment_group$DV, control_group$DV)
```

- Model

$$\hat{\sigma}_{\hat{\sigma}_{TC}} = \frac{\hat{\sigma}_{TC}}{\sqrt{2(n_T+n_C-1)}}$$

The standard error is equivalent to that of the standard deviation. For further information, refer to the "standard error of the standard deviation" section.

## standardized mean difference (SMD)

- R-Code

```
## apply the function
metafor::escalc(
measure = "SMD",
m1i = mean(treatment_group$DV),
m2i = mean(control_group$DV),
sd1i = sd(treatment_group$DV),
sd2i = sd(control_group$DV),
n1i = length(treatment_group$DV),
n2i = length(control_group$DV),
vtype = "LS2" # Borenstein variance
)$yi
## apply the function
```

- Model

$$g = d\left(1 - \frac{3}{4(n_T + n_C - 2) - 1}\right)$$

with

$$d = \frac{\bar{x}_T - \bar{x}_C}{\sqrt{\frac{\sum_{i=1}^{n}(x_T - \bar{x}_T)^2 + \sum_{i=1}^{n}(x_C - \bar{x}_C)^2}{n_T + n_C - 2}}}$$

## standard error of standardized mean difference (SE_SMD)

- R-Code

```
## apply the function
sqrt(metafor::escalc(
measure = "SMD",
m1i = mean(treatment_group$DV),
m2i = mean(control_group$DV),
sd1i = sd(treatment_group$DV),
sd2i = sd(control_group$DV),
n1i = length(treatment_group$DV),
n2i = length(control_group$DV),
vtype = "LS2" # Borenstein variance
)$vi)
## apply the function
```

- Model

$$\hat{\sigma}_g = \sqrt{\hat{\sigma}_d^2 \left(1 - \frac{3}{4(n_T + n_C - 2) - 1}\right)^2}$$

with

$$\hat{\sigma}_d^2 = \frac{n_T + n_C}{n_T n_C} + \frac{d^2}{2(n_T + n_C)}$$

## Value

The function create_replication_summaries returns a list consisting of two elements: A codebook and a list of data frames. Each data frame contains all replication summary statistics for the according replication (/effect). The summary statistics returned (including their standard error) are

the means and standard deviations for control and experimental groups, pooled standard deviations, raw mean differences and standardized mean differences (Hedge's g according to Borenstein et al., 2009).

## References

Ahn, S., & Fessler, J. A. (2003). Standard errors of mean, variance, and standard deviation estimators. EECS Department, The University of Michigan, 1(2).

Borenstein, M., Hedges, L. V., Higgins, J. P. T., & Rothstein, H. R. (2009). Introduction to Meta-Analysis John Wiley & Sons. Ltd, Chichester, UK. 10.1002/9780470743386

Viechtbauer, W. (2010). Conducting meta-analyses in R with the metafor package. Journal of Statistical Software, 36(3), 1-48. doi: 10.18637/jss.v036.i03

## Examples

```
# create vectors with names
# two multi-labs
MultiLab_names <- c("Multi_Lab_1", "Multi_Lab_2") # two projects
# two replication projects per multi lab
ReplicationProject_names <- c("Effect_A", "Effect_B", "Effect_C", "Effect_D")
# five replications per replication project
Replication_names <- c("Lab_A", "Lab_B", "Lab_C", "Lab_D", "Lab_E",
                       "Lab_A", "Lab_B", "Lab_C", "Lab_D", "Lab_E",
                       "Lab_F", "Lab_G", "Lab_H", "Lab_I", "Lab_J",
                       "Lab_F", "Lab_G", "Lab_H", "Lab_I", "Lab_J")

# create df with all example data
set.seed(1973)
example_data_df <- data.frame(MultiLab = rep(MultiLab_names, each = 100),
                          ReplicationProject = rep(ReplicationProject_names, each = 50),
                              # n = 10 (5 in control, 5 in treatment group)
                              Replication = rep(Replication_names, each = 10),
                              # random sampling for simulated data
                              DV = round(stats::rnorm(n = 2e2, mean = 0, sd = 5), 0),
                              Treatment = rep(c(1,0), times = 100))

# split the data per replication project to prepare for use in MetaPipeX::full_pipeline()
example_data_list <- split(example_data_df,
                           example_data_df$ReplicationProject)

## applying the input to the MetaPipeX function

# run create_replication_summaries
example_MetaPipeX_output <- MetaPipeX::create_replication_summaries(
data = example_data_list,
MultiLab = "MultiLab",
ReplicationProject = "ReplicationProject",
Replication = "Replication",
DV = "DV",
Group = "Treatment"
)

## Not run:
All examples with additional comments are available on github:
https://github.com/JensFuenderich/MetaPipeX/tree/main/Supplementary_Material/Code_Examples
```

```
## End(Not run)
```

---

`full_pipeline` *Full Pipeline Function*

---

## Description

\( \let\underscore_ \)

This function is built on three MetaPipeX functions (create_replication_summaries, merge_replication_summaries, meta_analyses), but also combines the meta-analytical data with the lab data in order to achieve the MetaPipeX data format. As input it expects the same specifications as the create_replication_summaries function. This function performs all standardized computational steps (3-6) of the MetaPipeX pipeline. For more details on the pipeline, refer to the documentation of the MetaPipeX-package.

## Usage

```
full_pipeline(
  data,
  MultiLab = NULL,
  ReplicationProject = NULL,
  Replication = NULL,
  DV = NULL,
  Group = NULL,
  output_path = NULL,
  folder_name = NULL,
  suppress_list_output = FALSE,
  method = "REML",
  sparse = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame or list of data frames that contain the individual participant data. The function expects the relevant columns to be named consistently across all list objects. Relevant to this function are columns that represent information on the MultiLab (e.g., Many Labs 2), the ReplicationProject (e.g., Ross1), the Replication (the lab a data point is assigned to), the group (either the treatment or control condition) and the single data point of the dependent variable (DV) per person. A template of this data frame is available on github, as is a codebook for unambiguous identification of the abbreviations. |
| `MultiLab` | Character vector with the name of the columns in the list elements of "data" that contain the project name(s). If *is.null(Project)* == *TRUE*, "Project" is chosen as the default. |
| `ReplicationProject` | Character vector with the name of the columns in the list elements of "data" that contain the replication projects name(s). If *is.null(Replication)* == *TRUE*, "Replication_Project" is chosen as the default. Each replication project comprises a single target effect with direct replications across multiple replications (/labs). |

`Replication` Character vector with the name of the columns in the list elements of "data" that contain the replication names (usually the name of the lab). If *is.null(Replication) == TRUE*, "Replication" is chosen as the default. The meta-analyses in MetaPipeX::meta_analyses() and MetaPipeX::full_pipeline() are run as random effects models in metafor::rma.mv() with "random = ~ 1 | Replication". Thus, the pipeline assumes a distribution of true statistics (e.g., treatment means, mean differences, standardized mean differences).

`DV` Character vector with the name of the columns in the list elements of "data" that contain the (aggregated) dependent variable. If *is.null(DV) == TRUE*, "DV" is chosen as the default.

`Group` Character vector with the name of the columns in the list elements of "data" that contain the (treatment/control) group identification. If *is.null(Group) == TRUE*, "Group" is chosen as the default. These should only contain values of 0 (control group), 1 (treatment group) and NA (unidentified).

`output_path` Specify the output path for the full documentation of the MetaPipeX pipeline. For an example of the exported structure please refer to the github repository. If no folder is specified, the function will return its output only to the R environment (unless this is suppressed under suppress_list_output).

`folder_name` Optional character string to assign a custom name to the output folder. When folder_name is not specified, the folder name is set to "MetaPipeX_Output".

`suppress_list_output` Logical. FALSE by default. If FALSE, the function will return a list output to the environment, containing the replication summaries and the codebook. If TRUE, these are not returned to the environment.

`method` Optional argument to specify the estimation method of the meta-analyses (the default is "REML"). For more information, please refer to the documentation of the metafor package.

`sparse` A logical indicating whether sparse matrices should be used.

## Details

## General notes on the pipeline

The MetaPipeX pipeline is a tool to provide structure to the meta-analytical-analyses of multi-lab replication projects. A flowchart that depicts the whole process is available on github The yellow blocks with rounded corners are .csv files. The purple/white rectangles each refer to a step in the pipeline that is performed by a MetaPipeX function. MetaPipeX::full_pipeline() performs the steps 3-6 and returns "MetaPipeX_Data.csv" which may be provided to the MetaPipeX App for handy data selection and basic plotting of the analysis results. Pleasre refer to github for an example of the MetaPipeX Output structure.

## full_pipeline

This function executes the pipeline as follows:

- MetaPipeX::create_replication_summaries()
- MetaPipeX::merge_replication_summaries
- MetaPipeX::meta_analyses()
- merging replication- and meta-level data to achieve MetaPipeX data format

## Value

The output is a nested list object that represents the folder structure that is available under LINK EINFUEGEN

## Examples

```
# create vectors with names
# two projects
MultiLab_names <- c("Multi_Lab_1", "Multi_Lab_2")
# two replications per project
ReplicationProject_names <- c("Effect_A", "Effect_B", "Effect_C", "Effect_D")
Multi_Lab_1 <- rep(c("Lab_A", "Lab_B", "Lab_C", "Lab_D", "Lab_E"), times = 2)
Multi_Lab_2 <- rep(c("Lab_F", "Lab_G", "Lab_H", "Lab_I", "Lab_J"), times = 2)
Replication_names <- c(Multi_Lab_1, Multi_Lab_2) # k = 5 per replication

# create df with all example data
set.seed(1973)
example_data_df <- data.frame(MultiLab = rep(MultiLab_names, each = 100),
ReplicationProject = rep(ReplicationProject_names, each = 50),
Replication = rep(Replication_names, each = 10), # n = 10 (5 in control, 5 in treatment group)
DV = round(stats::rnorm(n = 2e2, mean = 0, sd = 5), 0), # random sampling for simulated data
Treatment = rep(c(1,0), times = 100)
)

# split the data per replication project to prepare for use in MetaPipeX::full_pipeline()
example_data_list <- split(example_data_df,
                           example_data_df$ReplicationProject)

## applying the input to the MetaPipeX function \cr

# run full_pipeline
example_MetaPipeX_output <- MetaPipeX::full_pipeline(
data = example_data_list,
MultiLab = "MultiLab", # column name needs no change
ReplicationProject = "ReplicationProject",
Replication = "Replication",
DV = "DV",
Group = "Treatment" # column name needs changing
)

## Not run:
All examples with additional comments are available on github:
https://github.com/JensFuenderich/MetaPipeX/tree/main/Supplementary_Material/Code_Examples

## End(Not run)
```

---

merge_replication_summaries

*Merging Replication Summaries*

---

## Description

\( \let\underscore_ \) Function to merge the replication statistics returned by MetaPipeX::create_replication_summaries() into a single data frame. This is the second function (and the fourth computational step) of the MetaPipeX pipeline. For more details on the pipeline, refer to the documentation of the MetaPipeX-package.

## Usage

```
merge_replication_summaries(
  data,
  output_folder = NULL,
  suppress_list_output = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The function expects the input to be a list of data frames or a path to a folder containing the replication summaries as .csv files. The input may either be produced by the MetaPipeX::create_replication_summaries() function, or any inputs that use the data template. A template of this data frame is available on github, as is a codebook for unambiguous identification of the abbreviations. |
| output_folder | Define a path to which the merged replication summaries and the codebook are exported. If no path is specified, results are returned only in R. |
| suppress_list_output | A logical indicating whether results should be returned in R. If TRUE, no output is returned in R. |

## Details

No transformations are performed on the data in this step of the MetaPipeX pipeline.

## Value

A list object containing the following components:
## merged_replication_summaries A data frame with all replications from the input.

## codebook A codebook that applies to the data frame (merged_replication_summaries).
In order to export the data structure as .csv files in a folder, output_folder has to be specified.

## Examples

```
# import the according table template
Replication_Summaries_template <- readr::read_csv(url(
paste("https://raw.githubusercontent.com/JensFuenderich/MetaPipeX/main/Supplementary_Material/",
"Table_Templates/2_Replication_Summaries/Replication_Summaries_template.csv",
sep = ""
)))

# set seed for drawing data
set.seed(1973)

# create vectors with names
MultiLab_names <- c("MultiLab_1", "MultiLab_1", "MultiLab_2",  "MultiLab_2")
ReplicationProject_names <- c("Effect_A", "Effect_B", "Effect_C", "Effect_D")
```

```
Replication_names <- c("Lab_A", "Lab_B", "Lab_C", "Lab_D", "Lab_E", "Lab_F", "Lab_G", "Lab_H")

# random sampling for simulated data & building identifier variables
list_of_replication_summaries <- lapply(1:4, function(x){
  # sampling
  data_example <- as.data.frame(matrix(
 data = stats::rnorm(n = 200*(ncol(Replication_Summaries_template)-3), mean = 5, sd = 0.5),
  nrow = 200,
  ncol = ncol(Replication_Summaries_template)-3)
  )
  # rename columns according to template
  names(data_example) <- names(
  Replication_Summaries_template
  )[4:length(names(Replication_Summaries_template))]
  data_example$T_N <- round(data_example$T_N, 0)
  data_example$T_N <- round(data_example$C_N, 0)
  # building identifier variables
  MultiLab <- rep(MultiLab_names[x], times = nrow(data_example))
  ReplicationProject <- rep(ReplicationProject_names[x], times = nrow(data_example))
  Replication <- rep(if (x == 1 | x == 2) {
  Replication_names[1:4]
  } else if (x == 3 | x == 4) {
  Replication_names[5:8]
  }, each = nrow(data_example)/4)
  # combine data & identifiers
  cbind(MultiLab, ReplicationProject, Replication, data_example)
})
# rename list objects
names(list_of_replication_summaries) <- c("MultiLab_1_ReplicationProject_A_Replication_summaries",
                                "MultiLab_1_ReplicationProject_B_Replication_summaries",
                                "MultiLab_2_ReplicationProject_C_Replication_summaries",
                                "MultiLab_2_ReplicationProject_D_Replication_summaries")

## applying the input to the MetaPipeX function

# run merge_replication_summaries
example_MetaPipeX_output <- MetaPipeX::merge_replication_summaries(
data = list_of_replication_summaries
)

## Not run:
All examples with additional comments are available on github:
https://github.com/JensFuenderich/MetaPipeX/tree/main/Supplementary_Material/Code_Examples

## End(Not run)
```

---

`meta_analyses` *Meta Analyses*

---

## Description

\( \let\underscore_ \) Function to run meta-analyses on the mean difference (MD) and the standardized mean difference (SMD). The meta-analyses are run with the metafor::rma.mv function

(Viechtbauer, 2010). For more details on the meta-analyses, refer to the Details and Return section. This function is the third (and fifth computational step) of the MetaPipeX pipeline. For more details on the pipeline, refer to the documentation of the MetaPipeX-package.

## Usage

```
meta_analyses(
  data,
  output_folder = NULL,
  suppress_list_output = FALSE,
  method = "REML",
  sparse = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The function expects the input to be a data frame. The input may either be the data frame produced by the MetaPipeX::merge_replication_summaries() function, or one with the same columns names. A template of this data frame is available on github, as is a codebook for unambiguous identification of the abbreviations. Further, it is possible to use a reduced version of the codebook, as meta-analyses are applied to MD and SMD only. |
| output_folder | Specify the output folder for the replication summaries and the codebook. If no folder is specified, the function will return its output only to the R environment (unless this is suppressed under suppress_list_output). |
| suppress_list_output | A logical indicating whether results should be returned in R. If TRUE, no output is returned in R. |
| method | A character string to specify the type of model to be fitted. Default is "REML". For more details, refer to the metafor documentation. |
| sparse | A logical indicating whether sparse matrices should be used. |

## Details

The meta-analyses within the function are written with metafor::rma.mv (Viechtbauer, 2010). The multivariate version of the rma function is deployed to allow for the use of sparse matrices ("sparse = TRUE") for optimal performance in meta-analyses with thousands of replications. They are fitted as a random-effects model with "random = ~ 1 | Replication" and a restricted maximum likelihood estimation ("REML"). The function runs two meta-analyses per replication project:

- mean difference (yi = MD, sei = SE_MD)
- standardized mean difference (yi = SMD, sei = SE_SMD)

## Value

The output is a list with two objects: A data frame with the meta-analytical results and a codebook for unambiguous identification of its columns.

## meta analyses
The data frame contains information to identify each analysis (MultiLab, ReplicationProject) and statistical output from the two meta-analyses per replication project. The statistical output for each meta-analysis includes:

- A model estimate for the y of interest (Est__).
- The number of replications included in the analysis (Result__K).
- The estimated $\tau^2$ (sigma2 from the rma.mv object) value (Tau2__).
- The estimated $\tau$ (the square root of the sigma2 from the rma.mv object) value (Tau2__).
- The estimated $I^2$ value. $I^2$ is not part of the rma.mv output object and has to be calculated from $\tau$.

  $$I^2 = 100\frac{\hat{\tau}^2}{\hat{\tau}^2 + \tilde{v}}$$

  with

  $$\tilde{v} = \frac{(k-1)\sum w_i}{(\sum w_i) - \sum w_i^2}$$

  Transformation according to: https://wviechtb.github.io/metafor/reference/print.rma.html
- The estimated $H^2$ value. $H^2$ is not part of the rma.mv output object and has to be calculated from $\tau$.

  $$H^2 = 100\frac{\hat{\tau}^2 + \tilde{v}}{\tilde{v}}$$

  with

  $$\tilde{v} = \frac{(k-1)\sum w_i}{(\sum w_i) - \sum w_i^2}$$
- The Q statistic (QE__).
- The p-value from the test on the Q statistic (QEp__).

## codebook
A codebook that applies to the data frame (meta_analyses).

## References

Viechtbauer, W. (2010). Conducting meta-analyses in R with the metafor package. Journal of Statistical Software, 36(3), 1-48. doi: 10.18637/jss.v036.i03

## Examples

```
# import the according table template
Merged_Replication_Summaries_template <- readr::read_csv(url(
paste("https://raw.githubusercontent.com/JensFuenderich/MetaPipeX/main/Supplementary_Material/",
"Table_Templates/3_Merged_Replication_Summaries/Merged_Replication_Summaries_template.csv",
sep = "")
))

# set seed for drawing data
set.seed(1973)

# create vectors with names
MultiLab_names <- c("MultiLab_1", "MultiLab_1", "MultiLab_2",  "MultiLab_2")
ReplicationProject_names <- c("Effect_A", "Effect_B", "Effect_C", "Effect_D")
Replication_names <- c("Lab_A", "Lab_B", "Lab_C", "Lab_D", "Lab_E", "Lab_F", "Lab_G", "Lab_H")


# random sampling for simulated data & building identifier variables
list_of_replication_summaries <- lapply(1:4, function(x){
  # sampling
  data_example <- as.data.frame(matrix(
```

```
 data = stats::rnorm(n = 200*(ncol(Merged_Replication_Summaries_template)-3), mean = 5, sd = 0.5),
 nrow = 200,
 ncol = ncol(Merged_Replication_Summaries_template)-3)
 )
 # rename columns according to template
 names(data_example) <- names(
 Merged_Replication_Summaries_template
 )[4:length(names(Merged_Replication_Summaries_template))]
 data_example$T_N <- round(data_example$T_N, 0)
 data_example$T_N <- round(data_example$C_N, 0)
 # building identifier variables
 MultiLab <- rep(MultiLab_names[x], times = nrow(data_example))
 ReplicationProject <- rep(ReplicationProject_names[x], times = nrow(data_example))
 Replication <- rep(if (x == 1 | x == 2) {
 Replication_names[1:4]
 } else if (x == 3 | x == 4) {
 Replication_names[5:8]
 }, each = nrow(data_example)/4)
 # combine data & identifiers
 cbind(MultiLab, ReplicationProject, Replication, data_example)
})

# merge list object
merged_replication_summaries <- rbind(list_of_replication_summaries[[1]],
                                      list_of_replication_summaries[[2]],
                                      list_of_replication_summaries[[3]],
                                      list_of_replication_summaries[[4]])


## applying the input to the MetaPipeX function

# run merge_replication_summaries
example_MetaPipeX_output <- MetaPipeX::meta_analyses(data = merged_replication_summaries)

## Not run:
All examples with additional comments are available on github:
https://github.com/JensFuenderich/MetaPipeX/tree/main/Supplementary_Material/Code_Examples

## End(Not run)
```

---

ShinyApp *MetaPipeX Shiny App*

---

## Description

The MetaPipeX app is a GUI to provide insight into data that is in the MetaPipeX format. Make sure you are connected to the internet before running the app. For more details, please refer to the MetaPipeX tutorial. A web version of the app is available on a server of the LMU Munich.

## Usage

```
ShinyApp()
```

## Value

If executed this function will start a local instance of the MetaPipeX app.

# Index