mfd : S3 class

format: < +/- attribute [= default_value] : type >

- data_out = NULL : tibble # data merged with outputs of stage 1 and 2 (input of stage 3 functions)

- A = NULL : matrix
- Z = NULL : matrix
- S = NULL : matrix
- fac_dim=NULL : numeric
- ppca_object: ppca type object = NULL # TODO might be confusing, 2 versions of the loadings

- time : character
- subject : character

format: < +/- function(parameter [= default_value] : type, ...): return_parameter : type, ... >

1) - remove_obs_and_meas_NA_threshold(data_in : tibble, columns : list of strings, min_nonNA_fraction_per_row = 0.0 : double, min_nonNA_fraction_per_col = 0.0 : double) -> data_in : tibble
# normalization must be done before sampling (done over all possible samples), also requires meta information (e.g. which subject)
2) - as_ranks(data_in : tibble, columns : string or list of strings) -> data_in : tibble
3) - normalize(data_in : tibble, columns : string or list of strings, type : string, series_group = NULL : string)
3.1) - normalize_grouped(data_in : tibble, columns : string or list of strings, type = "standardize", group = NULL) -> data_in : tibble
# sampling requires meta information (e.g. which subject) -> must be done before selecting columns
4) - sampling(data_in : tibble, series_group : string, subgroup : string, total_sample_size : int, type = "equal_n_series_from_indications") -> data_in: tibble
4.1) - stratified_sampling(data_in : tibble, group : string, total_sample_size : int, sample_unit = NULL : string, subgroup_sample_strat = "proportionate" : string, with_replace = FALSE : bool) -> data_in : tibble

5) - select_columns(data_in : tibble, selected_columns : list of strings) -> data_in : tibble
6) - remove_all_NA_rows(Y_loadings : marix, Y_scores : matrix, data_loadings : tibble, data_scores : tibble) -> list(Y_loadings : matrix, Y_scores : matrix, data_loadings : tibble, data_scores : tibble)

7) + estimate_loadings(Y_in : **matrix**, type : string, var_explained = NULL : double, n_PCs = NULL : int or string, rotation = "promax", conv_thres = 1e-4, sparse_abs_thres = 0, ppca_seed = 1, normalize_loadings_type = "largest_abs" : string) -> list(A : matrix, ppca_object : ppca object type)
8.1) - sparse_rotation(mat : matrix, type : string) -> mat : matrix
8.2 - normalize_loadings(mat: matrix, type = "largest_abs" : string) -> mat : matrix
8.3) - sparse_heur_thres(mat : matrix, thres = 0.0 : double) -> mat : matrix

9) + estimate_scores(Y_in: matrix, A_in : **matrix**) -> list(Z : matrix, S : matrix, type = "lin_reg")
9.1) - find_unique_missing_patterns(Y_in : matrix) -> unique_missing_patterns : list, indices_Y_in_list : list

10) - prepare_data_long(mfd_in: mfd object) -> mfd_in : mfd object

11) - normalize_stage_1_2_results(A_in : matrix , Z_in : matrix, S_in : matrix, type = "scaling" : string) --> list[A_sc : matrix, Z_sc : matrix, S_sc : matrix]

"main interface function" for user aka. "mfd constructor" -> calls 1)-9)
# Primary (necessary) inputs
12) + mfd(data_in : tibble, vars : list of strings, time : string, subject : string,
# Secondary inputs
loadings.type = "n_PCs" : string, loadings.param = "max" : int or double or string,
rank_transform = FALSE, boolean,
normalize_data_loadings.type = "per_meas_standard" : string,
normalize_data_loadings.group = NULL : string,
normalize_A_Z_S.type = "normalize_custom" : string,
loadings.sparse_abs_thres = 0.0 : double,
# Tertiary inputs (almost never requiring user interference)
seed = NULL : int,   # NULL means random seed
loadings.rotation = "promax" : string,
keep_vars = NULL : vector of strings
) -> mfd_object : mfd object type

13) plot.mfd(mfd_object : mfd object, save_dir = NULL : string, colour_by = NULL : string)
13.1) plot_mfd_ppca(mfd_object mfd object, save_path = NULL : string)
13.2) plot_loadings_heatmap(mfd_object : mfd object, save_path = NULL : string, axis_y_text_size = 6 : int)
13.3) plot_z_trajectory(mfd_object : mfd object, save_path = NULL : string, factor : int, time : string, subject : string, colour_by = NULL : string, selected_subjects = NULL : vector of strings)

14) summary.mfd(mfd_object : mfd object)

mlfa : S3 class

format: < +/- attribute [= default_value] : type >

- mfd = NULL : mfd object
- data = NULL : tibble
- gamm_objects = NULL : list of gamm objects
- fac_dim = NULL ; int
- type = 1 : int

- time = mfd$time : string
- subject = mfd$subject : string

type 1 specific parameters
- type1.strata = strata
- type1.group = group

format: < +/- function(parameter [= default_value] : type, ...): return_parameter : type, ... >

1.1) mlfa_type1(data_in : tibble, df : int, factor = 1 : int, strata : string, group : string)

"main interface function" for user aka. "mlfa constructor" -> calls 1)
# Primary (necessary) inputs
2) mlfa(mfd : mfd object, type = 1 : int, df : int,
subject_data = NULL : tibble,
# Type 1
type1.strata = NULL : string, type1.group : string
) -> mlfa_object : mlfa object type

3) plot.mlfa(mlfa : mlfa object, save_dir = NULL : string)
3.1) predict_traj(gamm_object : gamm object, data : tibble, conf_SDs = 2 : int, step = 0.1 : float)
3.2.1) plot_traj_type1(gamm_object : gamm object, data : tibble, fill = True : boolean, save_path = NULL : string)

4) summary.mlfa(mlfa_object : mlfa object)