```r
if(!require("lavaan")) install.packages("lavaan"); library(lavaan)
if(!require("MplusAutomation")) install.packages("MplusAutomation"); library(MplusAutomation)
if(!require("semTools")) install.packages("semTools"); library(semTools)
#============================================================ % Parameters
    mplus = TRUE                                            # This is used to determine whether mplus is called or if only the input files created
    alpha = .05                                             # Significance level
    n.teams = 125                                           # Number of teams
    t.id = 1:n.teams                                        # Team IDs
    n.membs = 30L                                           # Number of members within team assuming a balanced design
    N = n.teams * n.membs                                   # Total size of sample
    n.perfs = 20L                                           # Number of peformance tasks
    perf.corrs = .60                                        # Correlation between performance tasks
    td = expand.grid(team.id = t.id, mem.num = 1:n.membs)   # team data structure
#============================================================ % Generate multivariate performance data
    S = matrix(rep(perf.corrs, n.perfs**2), ncol = n.perfs) # Population covariance matrix for performance tasks
    diag(S) = 1                                             # Makes population correlation matrix
    L = chol(S)                                             # Cholesky decomposition to simulate multivariate data
    nvars = dim(L)[1]                                       # Number of performance tasks
    r = t(L) %*% matrix(rnorm(nvars*N), nrow=nvars, ncol=N) # Creates multivariate data frame of performance tasks according to population correlation matrix
    rownames(r)  = paste0("perf.", 1:nvars)                 # Names variables as performance tasks
    pd = as.data.frame(t(r))                                # Individual performance data structure
    d = cbind(td, pd)                                       # Combined data set with individuals randomly assigned to team
# ============================================================ % Function to calculate intraclass correlations
calcICC = function(d){
    # ============================================================ % If multiple performance tasks
    if(n.perfs > 1){
        Ts = do.call(rbind, lapply(1:n.teams, function(x){  # ============= % Sums of Squares
            d.team = d[which(d$team.id == x), grep("perf.", names(d))]          # Get the performance tasks for each group
            x.sums = colSums(d.team)                                                # Calculate the sum of the scores for each group
            return(x.sums)}))                                                   # Return within-team summed performance scores
        Gs = colSums(Ts)                                                        # Grand sum of each of performance task
        SSBs =  colSums(Ts**2 / n.membs) - (Gs**2 / N)                          # Sums of squares between groups
        SSWs = colSums(do.call(rbind, lapply(1:n.teams, function(y){ # ==== % Sums of squares within groups
            d.team = d[which(d$team.id == y), grep("perf.", names(d))]          # Get each team
            ssw = colSums(d.team**2) - (colSums(d.team)**2 / n.membs)           # Calculate normal sums of squares within group
            return(ssw)})))                                                     # Return sums of squares from within teach team
        SSTs = colSums(d[, grep("perf.", names(d))]**2) - (Gs**2 / N)}          # Total sums of squares
    # ============================================================ % If single performance task
    else{
        Ts = do.call(rbind, lapply(1:n.teams, function(x){  # ============= % Sums of Squares
            d.team = d[which(d$team.id == x), grep("perf.", names(d))]          # Get the performance tasks for each group
            x.sums = sum(d.team)                                                # Calculate the sum of the scores for each group
            return(x.sums)}))                                                   # Return within-team summed performance scores
        Gs = sum(Ts)                                                            # Grand sum of each of performance task
        SSBs =  sum(Ts**2 / n.membs) - (Gs**2 / N)                              # Sums of squares between groups
        SSWs = sum(do.call(rbind, lapply(1:n.teams, function(y){     # ========= % Sums of squares within groups
            d.team = d[which(d$team.id == y), grep("perf.", names(d))]          # Get each team
            ssw = sum(d.team**2) - (sum(d.team)**2 / n.membs)                   # Calculate normal sums of squares within group
            return(ssw)})))                                                     # Return sums of squares from within teach team
        SSTs = sum(d[, grep("perf.", names(d))]**2) - (Gs**2 / N)}               # Total sums of squares
    if(!all.equal(SSTs, SSBs + SSWs)) {                                         # Confirm that between and within sums of squares sum to the total sums of squares
        warning("Total sums of squares do not equal summed between- and within-group sums of squares. ")}
    # ============================== % Variance / Mean-Squared Error
    df.t = N - 1                        # Total degrees of freedom
    df.b = n.teams - 1                  # Between-group degrees of freedom
    df.w = N - n.teams                  # Within-group degrees of freedom
    MSBs = SSBs / df.b                  # Mean-squared error between (i.e., between-group variance)
    MSWs = SSWs / df.w                  # Mean-squared error within (i.e., within-group variance)
    F.aov = MSBs / MSWs                 # F statistic for Analysis of Variance
    # ============================================================ % ICCs From Bliese (2000)
    ICC.1 = (MSBs - MSWs) / (MSBs + (n.membs - 1) * MSWs)               # ICC(1): expected correlation between any two random performance scores drawn from within the same team
    ICC.2 = n.membs*(ICC.1) / (1 + (n.membs - 1)*ICC.1)                 # ICC(2): reliability of the observed group performande task mean
    return(data.frame(t(rbind(ICC.1, ICC.2, F.aov))))}                  # Return matrix of results from above

# ============================================================================================ % Get ICC and bias results
ICCs = calcICC(d)                                                               # Intraclass correlations for the performance items across groups
print(ICCs)
#============================================================================ % Fit CFA on observed means for each group
team.dat = as.data.frame(do.call(rbind, lapply(1:n.teams, function(x){  # Get team-level data
    td = d[which(d$team.id == x), grep("perf", names(d))]               # Get each team's data independently
    ifelse(n.perfs > 1, return(colMeans(td)), return(mean(td)))})))     # Aggregates all performance tasks to team average
if(n.perfs == 1) names(team.dat) = c("perf")                            # Re-name dataset if only a single performance task
#============================================================================ % Team-level model, 'C' = collective intelligence
    team.mod = c("C =~ NA*perf.1 + ", paste0('perf.', 2:n.perfs, collapse='+'),"C ~~ 1*C ")
    team.cfa.fit = lavaan::cfa(team.mod, team.dat)                          # Team CFA fit on average member task performance
    summary(team.cfa.fit, fit.measures=T)                                   # Summarize team CFA results
    fit.pars = parameterEstimates(team.cfa.fit)                             # Get parameter estimates table
    C.load.pars = which(fit.pars$lhs %in% c("C") & fit.pars$op %in% c("=~"))    # Select the indices for the performance taks loadings on the 'C' factor
    C.loads = fit.pars[C.load.pars, c("est")]                               # Get estimates associated with the loadings
    C.ps = fit.pars[C.load.pars, c("pvalue")]
    err.vars.pars = which(fit.pars$lhs %in% c("C"))                         # Select indices for error variances
    err.vars = fit.pars[-err.vars.pars, c("est")]                           # Get estimates for error variability
    Rxx.C = sum(C.loads)**2 / (sum(C.loads)**2 + sum(err.vars))             # Reliability of team-level 'C' factor
    cat("'C' reliability = ", round(Rxx.C, 3))                              # Print team-level composite reliability
    reliability(team.cfa.fit)                                               # Uncomment to confirm reliability formula above, should equal the 'omega' estiamte
#============================================================================ % Individual-level model, 'G' = general mental ability
    ind.mod = c("G =~ NA*perf.1 + ", paste0('perf.', 2:n.perfs, collapse='+'),"G ~~ 1*G ")
    ind.cfa.fit = lavaan::cfa(ind.mod, d)                                   # Individual CFA fit on raw member task performance
    summary(ind.cfa.fit, fit.measures=T)                                    # Summarize individual-level CFA fit
    fit.pars.ind = parameterEstimates(ind.cfa.fit)                          # Get parameter estimates table
    G.load.pars = which(fit.pars.ind$op %in% c("=~"))                       # Select the indices for the performance taks loadings on the 'G' factor
    G.loads = fit.pars.ind[G.load.pars, c("est")]                           # Get estimates associated with the loadings
    G.ps = fit.pars.ind[G.load.pars, c("pvalue")]                           # Get estimates associated with the loadings
    G.err.vars.pars = which(fit.pars.ind$lhs %in% c("G"))                   # Select indices for error variances
    G.err.vars = fit.pars.ind[-G.err.vars.pars, c("est")]                   # Get estimates for error variability
    Rxx.G = sum(G.loads)**2 / (sum(G.loads)**2 + sum(G.err.vars))           # Reliability of individual-level 'G' factor
    cat("'G' reliability = ", round(Rxx.G, 3))                              # Print indivdiual-level composite reliability
    reliability(ind.cfa.fit)                                                # Uncomment to confirm reliability formula above, should equal the 'omega' estiamte
# Print the team- and individual-level loadings with p-values smaller than the alpha specified above
cat("The significant lavaan team-level loadings are:\n"); with(fit.pars, fit.pars[which(lhs == c("C") & pvalue < alpha), ])
cat("The significant lavaan individual-level loadings are:\n"); with(fit.pars.ind, fit.pars.ind[which(lhs == c("G") & pvalue < alpha), ])
```

```r
#============================================================================== % Run multilevel CFA in Mplus
ml.cfa.obj = mplusObject(VARIABLE = paste0( "\n\tUSEVARIABLES = team_id\n",                 # Mplus USEVARIABLES command: team_id
                                 paste0( "\t\t\tperf_", 1:n.perfs, collapse="\n"),   # Mplus USEVARIABLES command: performance tasks
                                         ";\n\tCLUSTER = team_id;\n"),                # Mplus CLUSTER command: team_id, identifies multilevel structure
                         ANALYSIS =  paste0( "\n\tTYPE = TWOLEVEL;\n",                # Mplus TWOLEVEL command: specificies robust maximum likelihood estimation
                                         "\tH1ITERATIONS = 5000;\n"),                 # Mplus H1ITERATIONS command: default is 2000, might need more to converge, e.g., 5000
                         MODEL =     paste0( "\n\t%WITHIN%\n\n",                      # Mplus %WITHIN% command: specifies the within-team performance factor
                                         "\t\tG BY \t*perf_1\n",                     # *perf_1 frees first loading to be estimated
                                         paste0( "\t\t\tperf_", 2:n.perfs, collapse="\n"),   # Performance tasks 2 through the number of tasks
                                         ";\n\t\tG@1.0;\n\n",                        # Standardizes within loadings by fixing the G factor variance to be 1.0
                                         "\n\t%BETWEEN%\n\n",                        # Mplus %BETWEEN% command: specifies the between-team performance factor
                                         "\t\tC BY \t*perf_1\n",                     # *perf_1 frees first loading to be estimated
                                         paste0( "\t\t\tperf_", 2:n.perfs, collapse="\n"),   # Performance tasks 2 through the number of tasks
                                         ";\n\t\tC@1.0;"),                          # Standardizes between loadings by fixing the C factor variance to be 1.0
                         OUTPUT =    c("RESIDUAL"),                                 # Get the model implied covariance matrix to calculate average variance extracted
                         rdata=d,                                                   # Use the data simulated above to run Mplus analyses
                         usevariables=names(d))
res = mplusModeler( ml.cfa.obj,                                                     # Fit the multilevel CFA in Mplus, creates MPlus input file ml_cfa.inp with full syntax
                    dataout=paste0("d_T", n.teams, "_I", n.membs, ".dat"),          # _T123, e.g., indicates team-level sample size; _I12 indicates number of team members
                    modelout=paste0("ml_cfa_T", n.teams, "_I", n.membs, ".inp"),    # _T123, e.g., indicates team-level sample size; _I12 indicates number of team members
                    run=as.integer(mplus))                                          # If mplus = TRUE above, sruns in Mplus; if mplus = FALSE only creates input files
out = readModels(paste0("ml_cfa_T", n.teams, "_I", n.membs, ".out"))                # Read in Mplus output file
pars = out$parameters$unstandardized                                                # Get parameter estimates from Mplus fit to calculate reliability
win.sigma = out$residuals$WITHIN$covarianceEst                                      # Model-implied covariance matrix for %WITHIN% level
btn.sigma = out$residuals$BETWEEN$covarianceEst                                     # Model-implied covariance matrix for %BETWEEN% level
#============================================================== % Two-level CFA Results % ====================== Uses equations found in Geldhoff et al. (2013)
#============================================================================================================== %WITHIN%
    win.loads = with(pars, pars[which(paramHeader == c("G.BY") & BetweenWithin == c("Within")), ])                  # Get within-team loadings
    win.sigma.var = diag(win.sigma)                                                                                 # Get within-team item variances
    win.sigma.covs = win.sigma[lower.tri(win.sigma, diag=FALSE)]                                                    # Get within-team item covariances
    win.alpha = (n.perfs**2*mean(win.sigma.covs)) / (sum(2*win.sigma.covs) + sum(win.sigma.var))                # Calculate within team coefficient alpha
    win.ave = sum(win.loads[, c("est")]**2) / sum(win.sigma.var)                                                    # Calculate within average variance extracted
    win.evars = with(pars, pars[which(paramHeader == c("Residual.Variances") & BetweenWithin == c("Within")), ])   # Get item residual variances for %WITHIN%
    win.omega = sum(win.loads[, c("est")])**2 / (sum(win.loads[, c("est")])**2 + sum(win.evars[, c("est")]))       # Calculate within-team latent composite reliability
    #========= % Print Results for Within-level Model
    cat("The significant Mplus loadings for the within-level are:\n");with(win.loads, win.loads[which(pval < alpha), ]) # Print significant within-team loadings
    cat("The Average Variance Extracted for the within-level is: ", win.ave)                                        # Print individual-level composite reliability
    cat("The latent composite reliability for the within-level is: ", win.omega)                                    # Print individual-level composite reliability
    cat("The coefficient alpha for the within-level is: ", win.alpha)                                               # Print individual-level composite reliability
#============================================================================================================== %BETWEEN%
    btn.loads = with(pars, pars[which(paramHeader == c("C.BY") & BetweenWithin == c("Between")), ])                 # Get between-team loadings
    btn.sigma.var = diag(btn.sigma)                                                                                 # Get between-team item variances
    btn.sigma.covs = btn.sigma[lower.tri(btn.sigma, diag=FALSE)]                                                    # Get between-team item covariances
    btn.alpha = (n.perfs**2*mean(btn.sigma.covs)) / (sum(2*btn.sigma.covs) + sum(btn.sigma.var))                # Calculate between-team coefficient alpha
    btn.ave = sum(btn.loads[, c("est")]**2) / sum(btn.sigma.var)                                                    # Calculate between average variance extracted
    btn.evars = with(pars, pars[which(paramHeader == c("Residual.Variances") & BetweenWithin == c("Between")), ])  # Get item residual variances for %BETWEEN%
    btn.omega = sum(btn.loads[, c("est")])**2 / (sum(btn.loads[, c("est")])**2 + sum(btn.evars[, c("est")]))       # Calculate between-team latent composite reliability
    #========= % Print Results for Between-Level Model
    cat("The significant Mplus team-level loadings are:\n");with(btn.loads, btn.loads[which(pval < alpha), ])      # Print significant team-level loadings
    cat("The Average Variance Extracted for the between-level is: ", btn.ave)                                       # Print team-level composite reliability
    cat("The latent composite reliability for the between-level is: ", btn.omega)                                   # Print team-level composite reliability
    cat("The coefficient alpha for the begween-level is: ", btn.alpha)                                              # Print individual-level composite reliability
#============================================================================================== Save all objects above using unique name based on n.teams/n.membs combination
save(d,                     # Individual-level data
     team.dat,              # Aggregated team-level data
     team.cfa.fit,          # CFA fit on aggregated team-level data
     ind.cfa.fit,           # CFA fit on individual-level data
     ml.cfa.obj,            # Mplus object specifying the multilevel CFA
     res,                   # Mplus results
     pars,                  # Parameter estimates from Mplus
     win.loads,             # Within-team loadings
     win.sigma.var,             # Within-team implied item variances
     win.evars,             # Within-team item residual variances
     win.ave,               # Within-team average variance extracted
     win.omega,             # Within-team latent composite reliability
     win.alpha,             # Within-team coefficient alpha
     btn.loads,             # Between-team loadings
     btn.sigma.var,             # Between-team implied item variances
     btn.evars,             # Between-team item residual variances
     btn.omega,             # Between-team latent composite reliaiblity
     btn.alpha,             # Between-team coefficient alpha
     btn.ave,               # Between-team average variance extracted
     file=paste0("ML_Spurious_CFA_T", n.teams, "_I", n.membs, ".RData"))    # Saves all data files and results from above
```