Package ‘eatRep’

December 17, 2013

Type Package

Title eatRep

Version 0.4.3

Depends R (>= 2.14.0), eatData, survey, reshape2, stringr, Hmisc, car,boot, combinat, plyr

Imports fmsb

Date 2013-12-08

Author Sebastian Weirich, Martin Hecht

Maintainer Sebastian Weirich <`sebastian.weirich@iqb.hu-berlin.de`>

Description Compute descriptives and regression models for complex survey designs with multiple imputed data

License GPL (>= 2)

LazyLoad yes

LazyData yes

R topics documented:

`eatRep-package` *Statistical analyses in complex survey designs with multiple imputed data.*

Description

Computes some basic statistic operations (means, standard deviations, frequency tables, percentiles and generalized linear models) in complex survey designs comprising multiple imputed variables and/or a clustered sampling structure which both deserve special procedures at least in estimating standard errors.

For example, computing standard errors for the mean of a multiple imputed variable (e.g. plausible values) involves the formulas provided by Rubin (1987). Computing standard errors for the mean of a nested imputed variable involves the formulas provided by Rubin (2003). Both methods are implemented in the package.

Moreover, computing standard errors for the mean of a variable which stems from a clustered design may involve replication methods like balanced repeated replicate (BRR), bootstrap or Jackknife methods. See Weststat (2000), Foy, Galia & Li, 2008, and Wolter, 1985 for details. To date, only the Jackknife-2 procedure (JK2) is supported.

The package `eatRep` is designed to combine both methods which is necessary if (multiple) imputed data are used in clustered designs. Considering the structure is relevant especially for the estimation of standard errors.

Technically, `eatRep` is a wrapper for the `survey` package (Lumley, 2004). Each function in `eatRep` corresponds to a specific function in `survey` which is called repeatedly during the analysis. The process is often similar: Each `eatRep` function first create replicate weights based on JKzone and JKrep variables according to the JK2 procedure. According to multiple imputed data sets, a workbook with several analyses is created. Without multiple imputations, the workbook only contains one analysis. For each entry in the workbook, a design object is created and the appropriate `survey` function is called. If multiple imputed or nested imputed data are analyzed, the results of the workbook analyses are pooled according to Rubin (1987) or Rubin (2003).

Details

Package:	eatRep
Type:	Package
Version:	0.4.3
Date:	2013-12-08
License:	GPL(>=2)

Author(s)

Author/maintainer: Sebastian Weirich <sebastian.weirich@iqb.hu-berlin.de>

References

Foy, P., Galia , J. & Li, I. (2008). Scaling the data from the TIMSS 2007 mathematics and science asssessment. In J. F. Olson, M. O. Martin & I. V. S. Mullis (ed.), *TIMSS 2007 Technical Report*

(S. 225–280). Chestnut Hill, MA: TIMSS & PIRLS International Study Center, Lynch School of Education, Boston College.

Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software* **9(1)**: 1–19

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys.* New York: Wiley.

Rubin, D.B. (2003): Nested multiple imputation of NMES via partially incompatible MCMC. *Statistica Neerlandica* **57, 1**, 3–18.

Satorra, A., & Bentler, P. M. (1994). Corrections to test statistics and standard errors in covariance structure analysis.

Westat (2000). *WesVar.* Rockville, MD: Westat.

Wolter, K. M. (1985). *Introduction to variance estimation.* New York: Springer.

---

dG *Display results of jk2.glm analyses.*

Description

This is an easy function only addressed to display results of jk2.glm analyses in an abbreviated manner.

Usage

```
dG ( object , analyses = NULL )
```

Arguments

`object`	A results object created by jk2.glm().
`analyses`	Optionally: a numeric vector auf analyses which should be displayed.

Value

A simply data frame, not intended for further processing.

Author(s)

Sebastian Weirich

Examples

```
### see examples in the vignette.
```

dM *Display results of jk2.mean analyses.*

Description

This is an easy function only addressed to display results of jk2.mean analyses in an abbreviated manner.

Usage

```
dM ( object, omitTerms = c("mean","sd","var", "Ncases","NcasesValid",
     "meanGroupDiff", "se","est") )
```

Arguments

`object`	A results object created by jk2.mean().
`omitTerms`	Optionally: character vector of terms you do not want to see on console.

Value

A simply data frame, not intended for further processing.

Author(s)

Sebastian Weirich

Examples

```
### see examples in the vignette.
```

dQ *Display results of jk2.quantile analyses.*

Description

This is an easy function only addressed to display results of jk2.quantile analyses in an abbreviated manner.

Usage

```
dQ ( object, seOmit = FALSE)
```

Arguments

`object`	A results object created by jk2.quantile().
`seOmit`	Optionally: omit displaying stand errors?

Value

A simply data frame, not intended for further processing.

Author(s)

Sebastian Weirich

Examples

```
### see examples in the vignette.
```

---

dT *Display results of jk2.table analyses.*

Description

This is an easy function only addressed to display results of jk2.table analyses in an abbreviated manner.

Usage

```
dT ( object, reshapeFormula = depVar + group ~ parameter + coefficient, seOmit = FALSE)
```

Arguments

`object`	A results object created by jk2.table().
`reshapeFormula`	Optionally: specify the formula used for reshaping.
`seOmit`	Optionally: omit displaying stand errors?

Value

A simply data frame, not intended for further processing.

Author(s)

Sebastian Weirich

Examples

```
### see examples in the vignette.
```

jk2.glm *JK2 for linear regression models.*

Description

Compute generalized linear models for complex cluster designs with multiple imputed variables based on Jackknife (JK2) procedure. Conceptually, the function combines replication methods and methods for multiple imputed data. Technically, this is a wrapper for the `svyglm()` function of the `survey` package.

Usage

```
jk2.glm(dat, ID, wgt = NULL, JKZone = NULL, JKrep = NULL, groups = list(),
        group.splits = length(groups), group.delimiter = "_", independent = list(),
        reg.statement = NULL, dependent = list(),
        complete.permutation = c("nothing", "groups", "independent", "all") ,
        glm.family, forceSingularityTreatment = FALSE, doCheck = TRUE, na.rm = FALSE)
```

Arguments

`dat`	Data frame containing all variables for analysis.
`ID`	Variable name or column number of ID variable.
`wgt`	Optional: Variable name or column number of weighting variable. If no weighting variable is specified, all cases will be equally weighted.
`JKZone`	Variable name or column number of variable indicating Jackknifing Zone. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`JKrep`	Variable name or column number of variable indicating replicate ID. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`groups`	Optional: List of one or more grouping variables. If grouping variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See example 1, where two group variables are used: One group variable is multiple imputed, the other is not.
`group.splits`	Optional: If groups are defined, `group.splits` optionally specifies whether analysis should be done also in the whole group or overlying groups. See examples for more details.
`group.delimiter`	Character string which separates the group names in the output frame.
`independent`	List of one or more independent variables. If independent variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See details and example 2 for more information.
`reg.statement`	Optional: By default, the regression formula is created automatically by the independent variables connected by a `"+"` symbol. However, when interaction terms should be defined, the regression has to be specified by the user. The right side of the regression formula has to defined in `"reg.statement"` as a string, whereas the names of the independent variables occurring in the `"independent"` argument have to be used. This is due to various names of one variable in multiple imputed data sets. See example 3 for further details.

`dependent`	List of one or more dependent variables. Each dependent variable will result in a separate analysis. If dependent variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See details and example 1 for more information. If dependent variable is a nested imputed variable, the nesting structure has to be specified in an appropriate listing structure. This structure has to be in agreement with the grouping variable and the independent variable if necessary. If nested imputed variables are included in the analysis, `complete.permutation` is strongly recommended to set to `FALSE`. If you are interested in one single dependent variable which may be labeled as `"reading_competence_plausible_value"` in the data file, type `dependent = list(reading = "reading_competence_plausible_value")`. This statement does not imply any imputation structure, therefore standard errors will not be pooled after the analysis. If your dependent variable is a multiple imputed one whereas the imputations are labeled as `"r_PV1"`, `"r_PV2"`, `"r_PV3"` and so on, type `dependent = list(reading = c("r_PV1", "r_PV2", "r_PV3"))`. This is to define three imputations belonging to one common scale which you can label arbitrarily. The results afterwards will be pooled according to Rubin (1987). If you are interested in one multiple imputed dependent variable with a nesting structure, where you have $N = 2$ nests and $M = 3$ imputations in each nest, your corresponding variables may be labeled as `"PV1_1"`, `"PV1_2"`, `"PV1_3"`, `"PV2_1"`, `"PV2_2"`, `"PV2_3"`. In that case the nesting structure has to be reproduced in an appropriate list structur. Therefore type `dependent = list(reading = list(n` This is to define which variables belong to which nest, whereas all variables belong to one common scale. The results afterwards will be pooled according to Rubin (2003). Example 3 gives a short introduction to this concept.
`complete.permutation`	Argument defines the number of multiple imputed data sets. In general, the number is defined by the number of imputations of a variable. Therefore, this argument only becomes relevant, if the number of imputation of one variable (e.g. the number of plausible values of the dependent variable) differs from the number of imputations used for independent variable(s). If `"all"`, number of datasets are determined through permutation, e.g. 5 plausible values and 3 imputations of the independent variable results in 3x5=15 imputed data sets. If `"nothing"`, only 5 imputed data sets will be used, no matter whether the larger number of imputations is a whole multiple of the smaller number of imputations. If `"groups"`, only the number of imputations of more than one grouping variable will be permutated. If `"independent"`, only the number of imputations of more than one independent variable will be permutated. Note: Within the logic of nested imputations, `complete.permutation` should be set to `FALSE`. See examples for further details.
`glm.family`	Argument of class `"family"`, specifying the link function. See help file of `"glm()"` function for details.
`forceSingularityTreatment`	Logical: Forces the function to use the workaround to handle singularities in regression models.
`doCheck`	Logical: Check the data for consistency before analysis? If `TRUE` groups with insufficient data are excluded from analysis to prevent subsequent functions from crashing.
`na.rm`	Logical: Should cases with missing values be dropped?

Details

Function first creates replicate weights based on JKZone and JKrep variables according to JK2 procedure implemented in WesVar. According to multiple imputed data sets, a workbook with several analyses is created. The function afterwards serves as a wrapper for svyglm() implemented in the survey package. The results of the several analyses are then pooled according to Rubin's rule, which is adapted for nested imputations if the dependent argument implies a nested structure.

Value

A data frame in the long format. For each subpopulation denoted by the groups statement, each dependent variable, each parameter and each coefficient the corresponding value is given.

Author(s)

Sebastian Weirich

Examples

```
data(reading_writing)
### Example 1: Computes linear regression from reading score on gender separately for each
### country and each hisei. This serves as an example for two group variables, where one
### group variable is a multiple imputed one.
mod1  <- jk2.glm(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD", JKZone = "JKZone",
         JKrep = "JKrep", groups = list(country = "country", hisei = paste("zehisei", 1:5, sep = "")),
         dependent = list(reading = paste("reading_score", 1:3, sep = ""),
                          writing = paste("writing_score", 1:3, sep = "")),
         independent = list(gender = "sex"),
         complete.permutation = "no", glm.family = gaussian(link="identity") )
### Example 2: Computes log linear regression from pass/fail on hisei and gender
### separately for each country
mod2  <- jk2.glm(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD", JKZone = "JKZone",
         JKrep = "JKrep", groups = list(country = "country"),
         dependent = list(reading.pass = paste("passed_reading", 1:3, sep = ""),
                          writing.pass = paste("passed_writing", 1:3, sep = "")),
         independent = list(hisei = paste("zehisei", 1:5, sep = ""), gender = "sex"),
         complete.permutation = "no", glm.family = binomial(link="logit") )
### Example 3: Computes linear regression from pass/fail on hisei and gender (and its interaction)
### separately for each country. The regression statement has to be defined manually. Note that,
### despite "hisei" occurs as a multiple imputed variable, only one statement has to be defined.
### Specifically, write "hisei*gender" instead of "zehisei1*sex" in the reg.statement argument.
### Moreover, the dependent variable herien is defined as a nested imputed variable with two nests
### and 3 imputations in each nest. The group and independent variable(s) have to correspond to
### the nested structure, i.e. they have to be specified as a single variable (i.e. without any
### imputations), or they also may be imputed in two nests with 3 imputations in each nest, or
### they may be imputed with two imputations (according to the number of nests in the dependent
### variable.
### In example 3, group variable has no imputations, and the first independent variable has 2
### imputations (according to the two nests), whereas the second independent variable also has
### no imputations.
mod3  <- jk2.glm(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD", JKZone = "JKZone",
         JKrep = "JKrep", groups = list(country = "country"),
         dependent = list(ability = list ( nest1 = paste("passed_reading", 1:3, sep = ""),
                                           nest2 = paste("passed_writing", 1:3, sep = ""))),
         independent = list(hisei = paste("zehisei", 1:2, sep = ""), gender = "sex"),
         reg.statement = "hisei*gender",
         complete.permutation = "no", glm.family = binomial(link="logit") )
```

jk2.mean *JK2 for mean estimates.*

Description

Compute totals, means, variances and standard deviations with standard errors for complex cluster designs with multiple imputed variables (e.g. plausible values) based on Jackknife (JK2) procedure. Conceptually, the function combines replication methods and methods for multiple imputed data. Nested imputations of the dependent variable(s) are supported as well. Technically, this is a wrapper for the `svymean()` and `svyvar()` functions of the `survey` package.

Usage

```
jk2.mean(dat, ID, wgt = NULL, JKZone = NULL, JKrep = NULL, groups = list(),
         group.splits = length(groups), group.differences.by = NULL,
         group.delimiter = "_", dependent = list(), na.rm = FALSE,
         complete.permutation = c("nothing", "groups", "all"),
         forcePooling = TRUE, boundary = 3, doCheck = TRUE)
```

Arguments

`dat`	Data frame containing all variables for analysis.
`ID`	Variable name or column number of ID variable.
`wgt`	Optional: Variable name or column number of weighting variable. If no weighting variable is specified, all cases will be equally weighted.
`JKZone`	Optional: Variable name or column number of variable indicating Jackknifing Zone. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`JKrep`	Optional: Variable name or column number of variable indicating replicate ID. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`groups`	Optional: List of one or more grouping variables. If grouping variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. If a nesting structure in the dependent variable is implemented, all grouping variables are expected to have no imputations (i.e. only one variable in the data frame) or exactly as many imputations as nests are specified in the dependent variable. For example, if two nests are specified in the dependent variable, a statement like this may be appropriate: `group = list(gender = "sex", hisei = c("hise` The function will be assume that no imputation takes place for variable `gender` and two imputations for variable `hisei`, corresponding to the two nests. Conversely, a statement like `group = list(gender = "sex", hisei = c("hisei1","hisei2","his` would be inconsistent. Albeit the function will not crash, the corresponding standard errors may not be trustworthy.
`group.splits`	Optional: If groups are defined, `group.splits` optionally specifies whether analysis should be done also in the whole group or overlying groups. See examples for more details.
`group.differences.by`	Optional: Specifies variable group differences should be computed for.

`group.delimiter`	Character string which separates the group names in the output frame.
`dependent`	List of one or more dependent variables. Each dependent variable will result in a separate analysis. If dependent variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. If dependent variable is a nested imputed variable, the nesting structure has to be specified in an appropriate listing structure. This structure has to be in agreement with the grouping variable if necessary. If nested imputed variables are included in the analysis, `complete.permutation` is strongly recommended to set to `FALSE`. To give a short example: If you are interested in one single dependent variable which may be labeled as `"reading_competence_plausible_value"` in the data file, type `dependent = list(reading = "reading_competence_plausible_value")`. This statement does not imply any imputation structure, therefore standard errors will not be pooled after the analysis. If your dependent variable is a multiple imputed one whereas the imputations are labeled as `"r_PV1"`, `"r_PV2"`, `"r_PV3"` and so on, type `dependent = list(reading = c("r_PV1", "r_PV2", "r_PV3"))`. This is to define three imputations belonging to one common scale which you can label arbitrarily. The results afterwards will be pooled according to Rubin (1987). If you are interested in one multiple imputed dependent variable with a nesting structure, where you have *N* = 2 nests and *M* = 3 imputations in each nest, your corresponding variables may be labeled as `"PV1_1"`, `"PV1_2"`, `"PV1_3"`, `"PV2_1"`, `"PV2_2"`, `"PV2_3"`. In that case the nesting structure has to be reproduced in an appropriate list structur. Therefore type `dependent = list(reading = list(n` This is to define which variables belong to which nest, whereas all variables belong to one common scale. The results afterwards will be pooled according to Rubin (2003). See details and examples for more information.
`na.rm`	Logical: Should cases with missing values be dropped?
`complete.permutation`	Argument defines the number of multiple imputed data sets. In general, the number is defined by the number of imputations of a variable. Therefore, this argument only becomes relevant, if the number of imputations of one variable (e.g. plausible values of the dependent variable) differs from the number of imputations used for group variable(s). If `"all"`, number of datasets are determined through permutation, e.g. 5 plausible values and 3 imputations of one group variable results in 3 x 5 = 15 imputed data sets. If `"nothing"`, only 5 imputed data sets will be used, no matter whether the larger number of imputations is a whole multiple of the smaller number of imputations. If `"groups"`, only the number of imputations of more than one grouping variable will be permutated. Note: Within the logic of nested imputations, `complete.permutation` should be set to `FALSE`. See examples for further details.
`forcePooling`	Logical: If variables in groups or subgroups does not have a positive variance, standard errors cannot be computed or pooled. Function will abort with an error message consequently. However, `forcePooling = TRUE` forces the algorithm to continue until the number of missing values does not exceed the criteria defined in `boundary`. Invalid variance estimators and standard errors will set to zero to allow pooling. A warning is printed to console. However, it is strongly advised not to trust in starndard errors then.
`boundary`	Numerical: Defines the maximum number of missing values in standard errors to continue pooling when `forcePooling == TRUE`.
`doCheck`	Logical: Check the data for consistency before analysis? If `TRUE` groups with insufficient data are excluded from analysis to prevent subsequent functions from crashing.

Details

Function first creates replicate weights based on JKZone and JKrep variables according to JK2 procedure implemented in WesVar. According to multiple imputed data sets, a workbook with several analyses is created. The function afterwards serves as a wrapper for `svymean()` called by `svyby()` implemented in the 'survey' package. The results of the several analyses are then pooled according to Rubin's rule.

Value

A data frame in the long format. For each subpopulation denoted by the `groups` statement, each dependent variable, each parameter (i.e., mean, variance, or group differences) and each coefficient (i.e., the estimate and the corresponding standard error) the corresponding value is given.

Author(s)

Sebastian Weirich

Examples

```
data(reading_writing)
### First example: only means, SD and variances for each country
means  <- jk2.mean(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD", JKZone = "JKZone",
          JKrep = "JKrep", groups = list(country = "country"),
          dependent = list(reading = paste("reading_score",1:3,sep=""),
                           writing = paste("writing_score",1:3,sep="") ),
          complete.permutation = "all" )

### Second example: Sex differences by country. Assume equally weighted cases by omitting
### wgt argument.
means  <- jk2.mean(dat = reading_writing, ID = "idstud", JKZone = "JKZone",JKrep = "JKrep",
          groups = list(country = "country", GENDER = "sex"), group.differences.by = "GENDER",
          dependent = list(reading = paste("reading_score",1:3,sep=""),
                           writing = paste("writing_score",1:3,sep="") ),
          complete.permutation = "all" )

### Third example: Nested imputations of dependent variable
### First split the income in above and below 2000
reading_writing[,"incomeS1"] <- ifelse(reading_writing[,"income1"]>2000,1,0)
reading_writing[,"incomeS2"] <- ifelse(reading_writing[,"income2"]>2000,1,0)
### Assuming 2 nests (i.e. variable "income" with 2 imputations), and one common dependent variable
### of reading and writing. Note that complete.permutation should be set to FALSE
means  <- jk2.mean(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD", JKZone = "JKZone",
          JKrep = "JKrep", groups = list(country = "country", INCOME = c("incomeS1","incomeS2")),
          dependent = list(ability = list(nest1 = paste("reading_score",1:3,sep=""),
                                          nest2 = paste("writing_score",1:3,sep="") )),
          complete.permutation = "no" )

### Fourth example: Assume a completely random sample (e.g. no cluster structure and no weights),
### and give descriptives for subpopulations and the overlying populations
means  <- jk2.mean(dat = reading_writing, ID = "idstud",
          groups = list(country = "country", GENDER = "sex"),
          group.splits = c(0:2),
          dependent = list(reading = paste("reading_score",1:3,sep=""),
                           writing = paste("writing_score",1:3,sep="") ),
          complete.permutation = "all" )
```

`jk2.quantile` *JK2 method for quantiles.*

Description

Compute quantiles with standard errors for complex cluster designs with multiple imputed variables (e.g. plausible values) based on Jackknife (JK2) procedure. Conceptually, the function combines replication methods and methods for multiple imputed data. Technically, this is a wrapper for the `svyquantile()` function of the survey package.

Usage

```
jk2.quantile(dat, ID, wgt = NULL, JKZone = NULL, JKrep = NULL, groups = list(),
            group.splits = length(groups), group.delimiter = "_", dependent = list(),
            probs = seq(0, 1, 0.25),  na.rm = FALSE,
            complete.permutation = c("nothing", "groups", "all"), nBoot = NULL,
            bootMethod = c("wSampling","wQuantiles"), doCheck = TRUE)
```

Arguments

`dat`	Data frame containing all variables for analysis.
`ID`	Variable name or column number of ID variable.
`wgt`	Optional: Variable name or column number of weighting variable. If no weighting variable is specified, all cases will be equally weighted.
`JKZone`	Variable name or column number of variable indicating Jackknifing Zone. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`JKrep`	Variable name or column number of variable indicating replicate ID. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`groups`	Optional: List of one or more grouping variables. If grouping variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See details for more information.
`group.splits`	Optional: If groups are defined, `group.splits` optionally specifies whether analysis should be done also in the whole group or overlying groups. See examples for more details.
`group.delimiter`	Character string which separates the group names in the output frame.
`dependent`	List of one or more grouping variables. Each dependent variable will result in a separate analysis. If grouping variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See details for more information.
`probs`	Numeric vector with probabilities for which to compute quantiles.
`na.rm`	Logical: Should cases with missing values be dropped?
`complete.permutation`	Argument defines the number of multiple imputed data sets. In general, the number is defined by the number of imputations of a variable. Therefore, this argument only becomes relevant, if the number of imputations of one variable

	(e.g. number of plausible values of the dependent variable) differs from the number of imputations used for group variable(s). If `"all"`, number of datasets are determined through permutation, e.g. 5 plausible values and 3 imputations of one group variable results in 3x5=15 imputed data sets. If `"nothing"`, only 5 imputed data sets will be used, no matter whether the larger number of imputations is a whole multiple of the smaller number of imputations. If `"groups"`, only the number of imputations of more than one grouping variable will be permutated.
`nBoot`	Optional: Without replicates, standard error cannot be computed in a weighted sample. Alternatively, standard errors may be computed using the `boot` package. `nBoot` therefore specifies the number of bootstrap samples. If not specified, no standard errors will be given. In analyses containing replicates or samples without specifying person weights, `nBoot` will be ignored.
`bootMethod`	Optional: If standard error are computed in a bootstrap, two possible methods may be applied. `wSampling` requests the function to draw `nBoot` weighted bootstrap samples for which unweighted quantiles are computed. `wQuantiles` requests the function to draw `nBoot` unweighted bootstrap samples for which weighted quantiles are computed.
`doCheck`	Logical: Check the data for consistency before analysis? If `TRUE` groups with insufficient data are excluded from analysis to prevent subsequent functions from crashing.

Details

Function first creates replicate weights based on JKZone and JKrep variables according to JK2 procedure implemented in WesVar. According to multiple imputed data sets, a workbook with several analyses is created. The function afterwards serves as a wrapper for `svyquantile()` called by `svyby()` implemented in the 'survey' package. The results of the several analyses are then pooled according to Rubins rule, which is adapted for nested imputations if the `dependent` argument implies a nested structure.

Value

A list of data frames, one for each dependent variable. Each data frame contains percentile values and their standard errors.

Author(s)

Sebastian Weirich

Examples

```
data(reading_writing)
### First example: Computes percentile for reading and writing scores conditionally on country
perzent   <- jk2.quantile(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD",
             JKZone = "JKZone", JKrep = "JKrep", groups = list(country = "country"),
             dependent = list(reading = paste("reading_score", 1:3, sep = ""),
                              writing = paste("writing_score", 1:3, sep = "")  ) ,
             probs = seq(0.1,0.9,0.2), complete.permutation = "no" )

### Second example: Computes percentile for reading and writing scores conditionally on country,
### use 100 bootstrap samples
perzent   <- jk2.quantile(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD",
             groups = list(country = "country"),
             dependent = list(reading = paste("reading_score", 1:3, sep = ""),
                              writing = paste("writing_score", 1:3, sep = "")  ) ,
             probs = seq(0.1,0.9,0.2), complete.permutation = "no", nBoot = 100 )
```

`jk2.table` *JK2 for frequency tables.*

Description

Compute frequency tables for categorical variables (e.g. factors: dichotomous or polytomous) in complex cluster designs. Estimation of standard errors optionally takes the clustered structure and multiple imputed variables into account. To date, only the Jackknife-2 procedure (JK2) is implemented to account for clustered designs. Procedures of Rubin (1987) and Rubin (2003) are implemented to account for multiple imputed data and nested imputed data. Conceptually, the function combines replication and imputation methods. Technically, this is a wrapper for the svymean() function of the survey package.

Usage

```
jk2.table(dat, ID, wgt = NULL, JKZone = NULL, JKrep = NULL, groups = list(),
          group.splits = length(groups), group.delimiter = "_", dependent = list(),
          separate.missing.indikator = FALSE, expected.values = list(),
          complete.permutation = c("nothing", "groups", "all"), doCheck = TRUE )
```

Arguments

`dat`	Data frame containing all variables for analysis.
`ID`	Variable name or column number of ID variable.
`wgt`	Optional: Variable name or column number of weighting variable. If no weighting variable is specified, all cases will be equally weighted.
`JKZone`	Optional: Variable name or column number of variable indicating Jackknifing Zone. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`JKrep`	Optional: Variable name or column number of variable indicating replicate ID. If `NULL`, no cluster structure is assumed and standard errors are computed according to a random sample.
`groups`	Optional: List of one or more grouping variables. If grouping variable is a multiple imputed variable, all names concerning one variable are interpreted as its imputations. See details for more information.
`group.splits`	Optional: If groups are defined, `group.splits` optionally specifies whether analysis should be done also in the whole group or overlying groups. See examples for more details.
`group.delimiter`	Character string which separates the group names in the output frame.
`dependent`	List of one or more dependent variables, which have to be dichotomous or polytomous. (In R, variables of this kind are often represented as factors. However, dependent variable may also be of class numeric or character.) Each dependent variable will result in a separate analysis. If dependent variable is a multiple

imputed variable, all names concerning one variable are interpreted as its imputations. If dependent variable is a nested imputed variable, the nesting structure has to be specified in an appropriate listing structure. This structure has to be in agreement with the grouping variable if necessary. If nested imputed variables are included in the analysis, `complete.permutation` is strongly recommended to set to `FALSE`.

`separate.missing.indikator`
: Logical. Should frequencies of missings in dependent variable be integrated? Note: That is only useful if missing occur as `NA`. If the dependent variable is coded as character, for example `male, female, missing`, separate missing indicator is not necessary.

`expected.values`
: Optional. A vector auf values expected in dependent variable. Recommend to left this argument empty.

`complete.permutation`
: Argument defines the number of multiple imputed data sets. In general, the number is defined by the number of imputations of a variable. Therefore, this argument only becomes relevant, if the number of imputations of one variable (e.g. dependent variable) differs from the number of imputations used for group variable(s). If `"all"`, number of datasets are determined through permutation, e.g. 5 plausible values and 3 imputations of one group variable results in 3x5=15 imputed data sets. If `"nothing"`, only 5 imputed data sets will be used, no matter whether the larger number of imputations is a whole multiple of the smaller number of imputations. If `"groups"`, only the number of imputations of more than one grouping variable will be permutated.

`doCheck`
: Logical: Check the data for consistency before analysis? If `TRUE` groups with insufficient data are excluded from analysis to prevent subsequent functions from crashing.

Details

Function first creates replicate weights based on JKZone and JKrep variables according to JK2 procedure implemented in WesVar. According to multiple imputed data sets, a workbook with several analyses is created. The function afterwards serves as a wrapper for `svymean()` called by `svyby()` implemented in the `survey` package. Relative frequencies of the categories of the dependent variable are computed by the means of the dichotomous indicators (e.g. dummy variables) of each category. The results of the several analyses are then pooled according to Rubin's rule, which is adapted for nested imputations if the `dependent` argument implies a nested structure.

Value

A data frame in the long format. For each subpopulation denoted by the `groups` statement, each dependent variable, each parameter (i.e., the values of the corresponding categories of the dependent variable) and each coefficient (i.e., the estimate and the corresponding standard error) the corresponding value is given.

Author(s)

Sebastian Weirich

References

Rubin, D.B. (2003): Nested multiple imputation of NMES via partially incompatible MCMC. *Statistica Neerlandica* **57, 1**, 3–18.

Examples

```
data(reading_writing)
### First example: Computes frequencies of the hisei group conditionally on the
### groups of passed and failed with respect to reading score
freq.tab  <- jk2.table(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD",
             JKZone = "JKZone", JKrep = "JKrep",
             groups = list(country = "country",
                          passed.reading = paste("passed_reading",1:3,sep="" )),
             dependent = list(hisei = paste("zehisei", 1:5, sep = "") ) ,
             complete.permutation = "no" )

### Second example: Computes frequencies of the hisei group conditionally on the
### groups of passed and failed with respect to reading score. Assuming no cluster
### but a nested imputation structure. complete.permutation is recommended to set
### to nothing.
freq.tab  <- jk2.table(dat = reading_writing, ID = "idstud", wgt = "wgtSTUD",
             groups = list(country = "country",
                          passed.reading = paste("passed_reading",1:2,sep="" )),
             dependent = list(hisei = list(nest1 = paste("zehisei", 1:2, sep = "") ,
                                           nest2 = paste("zehisei", 3:4, sep = "") )),
             complete.permutation = "no" )
```

Index