Package ‘quark’

March 13, 2016

Type Package

Title Missing data analysis with principal component auxiliary variables

Version 0.6.1

Date 2016-02-25

Author Kyle M. Lang, Steven Chesnut, Todd D. Little

Maintainer Kyle M. Lang <kyle.lang@ttu.edu>

Description Implements the ideas of Howard, Rhemtulla, and Little (2015) to execute a principled missing data analysis that uses principal component scores as the auxiliary variables.

License GPL-3 | file LICENSE

Depends methods, mice, vcd, ICC, rlecuyer, parallel

NeedsCompilation no

R topics documented:

quark-package — *Missing data analysis with principal component auxiliary variables*

Description

Implements the ideas of Howard, Rhemtulla, and Little (2015) to execute a principled missing data analysis that uses principal component scores as the auxiliary variables.

Details

The DESCRIPTION file:

Package:	quark
Type:	Package
Title:	Missing data analysis with principal component auxiliary variables
Version:	0.6.1
Date:	2016-02-25
Author:	Kyle M. Lang, Steven Chesnut, Todd D. Little
Maintainer:	Kyle M. Lang <kyle.lang@ttu.edu>
Description:	Implements the ideas of Howard, Rhemtulla, and Little (2015) to execute a principled missing data analysis tha
License:	GPL-3 | file LICENSE
Depends:	methods, mice, vcd, ICC, rlecuyer, parallel

Index of help topics:

```
createPcAux             Extract Principal Component Auxiliaries for
                        Missing Data Analysis
getImpData              Extract multiply imputed datasets from a
                        RomData object.
inspect                 Access fields of a QuarkData object.
iris2                   A modified version of the Fisher/Anderson iris
                        data.
makePredMatrix          Make a predictor matrix for use with 'mice'.
mergePcAux              Merge Principal Component Auxiliaries with the
                        raw data from which they were constructed.
miWithPcAux             Create multiple imputations using the PcAux
                        produced by 'createPcAux'.
prepData                Prepare Data for Extracting Principal Component
                        Auxiliaries
quark-package           Missing data analysis with principal component
                        auxiliary variables
quarkW                  Print 'quark”s warranty statement.
```

Author(s)

Kyle M. Lang, Steven Chesnut, Todd D. Little

Maintainer: Kyle M. Lang <kyle.lang@ttu.edu>

References

Howard, W. H., Rhemtulla, M., & Little, T. D. (2015). Using principal components as auxiliary variables in missing data estimation. Multivariate Behavioral Research. 50(3). 285-299.

Examples

```
## Load the data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create principal component auxiliary variables:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )

## Conduct MI with the pcAux:
miOut <- miWithPcAux(rawData = iris2,
                     quarkData = pcAuxOut,
                     nImps = 5)

### OR get the constituent parts ###

## Merge the PC auxiliaries with the original data:
outData <- mergePcAux(quarkData = pcAuxOut, rawData = iris2)

### outData can be analyzed via FIML, with the
### pcAux scores used as auxiliary variables.

## Create a predictor matrix:
predMat <- makePredMatrix(mergedData = outData$data)

### You can run mice() manually by supplying
### predMat to the predictorMatrix argument.
```

---

createPcAux *Extract Principal Component Auxiliaries for Missing Data Analysis*

Description

Implement the ideas of Howard, Rhemtulla, and Little (2015).

Usage

```
createPcAux(quarkData,
            nComps = c(10L, 3L),
            useInteract = TRUE,
            usePoly = TRUE,
            maxPower = 3L,
            pcaMemLevel = 0L,
            simMode = FALSE,
            mySeed = 235711L,
            forcePmm = FALSE,
            verbose = !simMode,
            doImputation = TRUE,
            castData = !doImputation,
            control,
            ...)
```

Arguments

quarkData	An object of class QuarkData returned from prepData.
nComps	A two-element integer vector giving the number of linear and nonlinear, respectively, component scores to extract. Defaults to nComps = c(10L, 3L).
useInteract	A logical flag indicating whether or not to include the two-way interactions in the PCA. Defaults to useInteract = TRUE.
usePoly	A logical flag indicating whether or not to include polynomial terms in the PCA. Defaults to usePoly = TRUE.
maxPower	An integer giving the maximum power used when constructing the polynomial terms. Defaults to maxPower = 3L.
pcaMemLevel	An integer code representing a tradeoff between memory usage and numerical accuracy in the algorithm used to extract the principal component scores. A value of '0L' (the default) will extract the PC scores with the stats::prcomp() package for maximal accuracy. A value of '1L' will use the quark:::simplePca() subroutine to extract the PC scores with considerably lower memory usage but, possibly, less numerical accuracy than the prcomp() approach. Leaving this option at the default value should be sufficient for most applications.
simMode	A logical switch turning 'Simulation Mode' on or off. In Simulation Mode all of the automatic data checks will be suppressed. This mode is intended for use when quark is being called as part of a Monte Carlo simulation study in which the data properties are well-known by the user. This mode should not be used for 'real-world' data analysis. Defaults to simMode = FALSE.
mySeed	An integer used to seed the random number generator used by the imputation algorithm. Defaults to mySeed = 235711L.
forcePmm	A logical flag indicating whether or not the initial single imputation should use predictive mean matching as the elementary imputation method for all variables. If forcePmm == FALSE, the elementary imputation methods are chosen to match each variable's declared type. Defaults to forcePmm = FALSE.

`verbose` A logical switch indicating whether output should be printed to the screen. Warnings are always printed, regardless of the value assigned to `verbose`. Defaults to `verbose = !simMode`.

`doImputation` A logical switch indicating whether the data should be imputed before extracting the principal component scores. Set to `FALSE` if the data element in `quarkData` has no missing values (e.g., the imputation was done elsewhere). Defaults to `doImputation = TRUE`.

`castData` A logical switch indicating whether the data element in `quarkData` should have its variables re-typed. Keep as `FALSE` unless the data have been manipulated after running `prepData`. Defaults to `castData = FALSE`.

`control` An optional list of control parameters (see 'Details').

`...` Used to pass undocumented debugging arguments to the internal methods. These arguments are primarily intended for developer use. Refer to the source code to understand how these arguments are employed.

Details

The `control` argument is a key-paired list with the following possible entries:

- miceIters: Number of EM iterations supplied to the `maxit` argument of mice() during the initial single imputation. Defaults to `miceIters = 10L`.
- miceRidge: Value of the ridge penalty parameter used to stabalize the imputation models used by mice(). Defaults to `miceRidge = 1e-5`.
- collinThresh: The strength of linear association used to flag collinear variable for removal. Defaults to `collinThresh = 0.95`.
- minRespCount: The minimum number of observations allowed on each variable without triggering a warning. Defaults to `floor(0.05 * nrow(rawData))`.
- minPredCor: The minimum magnitude of correlation supplied to the `mincor` argument of mice::quickpred() when constructing the predictor matrix used by mice() during the initial single imputation. Defaults to `minPredCor = 0.1`.
- maxNetWts: The maximum number of network weights used by nnet() to fit the polytimous regression models used to impute nominal variables with mice(). Defaults to `maxNetWts = 10000L`.
- nomMaxLev: The maximum number of response levels for nominal variables that won't trigger a warning. Defaults to `nomMaxLev = 10L`.
- ordMaxLev: The maximum number of response levels for ordinal variables that won't trigger a warning. Defaults to `ordMaxLev = 10L`.
- conMinLev: The minimum number of unique responses for continuous variables that won't trigger a warning. Defaults to `minConLev = 10L`.
- nGVarCats: The number of categories into which continuous grouping variables will be split, if applicable. Defaults to `nGVarCats = 3L`.

Value

An Reference Class object of class QuarkData with fields for each of the `createPcAux` function's arguments (except for the raw data which are removed to save resources) and the following modified or additional fields:

- call: A list containing the matched function call to `quark`.
- pcAux: A list of length 2. The first element contains the linear principal component auxiliary scores. The second element contains the non-linear principal component auxiliary scores.
- rSquared: A list of length 2. The first element contains the cumulative proportion of variance explained by the linear principal component auxiliary scores. The second element containst the cumulative proportion of variance explained by the non-linear principal component auxiliary scores.
- typeVec: A character vector giving the types assigned to each variable in `rawData`.
- methVec: A character vector giving the elementary imputation methods used by **mice**.
- respCounts: An integer vector giving the variable-wise counts of any missing data in `rawData` that remain after the initial single imputation. Any variables with non-zero entries in respCounts are dropped from the data before extracting the principal component scores to keep the PCA from using listwise-deletion.
- initialPm: A numeric vector giving the initial, variable-wise percents missing for `rawData` before any treatment.
- dropVars: A two-column character matrix. The first column contains the names of all variables dropped from the analysis. The second column contains the reason that the corresponding variable was dropped.
- dummyVars: A character vector containing the names of the dummy-coded representations of the nominal variables.
- probNoms: A character vector giving the variable names for any nominal variables with more levels than `control$nomMaxLev`.
- probOrds: A character vector giving the variable names for any ordinal variables with more levels than `control$ordMaxLev`.
- probCons: A character vector giving the variable names for any continuous variables with fewer levels than `control$conMinLev`.
- levelVec: An integer vector giving the number of unique, non-missing, levels for each variable in `rawData`.
- highPmVars: A character vector containing the names of variables with fewer observed responses than `control$minRespCount`.
- emptyVars: A character vector giving the names of empty columns in `rawData`.
- constants: A character vector giving the names of constant columns in `rawData`.
- collinVars: A three-column character matrix. The first two columns contain the names of pairs of approximately collinear variables. The third column contains their observed linear association.
- impFails: A named list of length 4 with elements: 'firstPass', 'pmm', 'groupMean', and 'grandMean' containing the names of any variables that were not successfully imputed via the named imputation strategy. 'First Pass' imputation refers to the ideal approach that assigns the elementary imputation methods according to each variables declared type. The remaining three methods are less-optimal fall-back approaches.
- patterns: If the imputation process falls back to group mean substitution, this field contains a list of the concatonated grouping patterns used to define the strata within which the group means were computed. This list will have length equal to `length(groupVars)`.

- frozenGVars: If group mean substitution is attempted and some grouping variables are continuous, this field contains the binned versions of the continous grouping variables that were used for the group mean substitution.
- idFills: A list containing the values used to deterministically fill any missing data that occured on the ID variables. The length of this argument will equal the number of incomplete ID variables in `rawData`.

Author(s)

Kyle M. Lang & Steven Chesnut

References

Howard, W. H., Rhemtulla, M., & Little, T. D. (2015). Using principal components as auxiliary variables in missing data estimation. Multivariate Behavioral Research. 50(3). 285-299.

See Also

`prepData`, `miWithPcAux`

Examples

```
## Load data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create the principal component auxiliaries:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )
```

---

getImpData — *Extract multiply imputed datasets from a RomData object.*

Description

This is a simple wrapper function that extracts the completed, multiply imputed data sets from a fitted RomData object produced by running the `rom` function.

Usage

```
getImpData(quarkData)
```

Arguments

quarkData A fitted object of class QuarkData produced as output of the miWithPcAux function.

Value

A set of multiply imputed data sets. The format of these data sets is defined by the completeFormat value in quarkData. See miWithPcAux for more information.

Author(s)

Kyle M. Lang

See Also

miWithPcAux, createPcAux

Examples

```
## Load the data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create principal component auxiliary variables:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )

## Conduct MI with the pcAux:
miOut <- miWithPcAux(rawData = iris2,
                     quarkData = pcAuxOut,
                     nImps = 5)

## Extract a list of imputed data sets:
impList <- getImpData(quarkData = miOut)
```

---

inspect *Access fields of a QuarkData object.*

Description

Provide S3/S4-like access to fields of a QuarkData Reference Class object.

Usage

```
inspect(object, what)
```

Arguments

`object`	An initialized RC object of class QuarkData.
`what`	A character string naming the field to access in `object`.

Value

The current value stored in the `what` field of `object`.

Author(s)

Kyle M. Lang

Examples

```
## Load data:
data(iris2)

## Prepare the data:
newData <- prepData(rawData = iris2,
                    nomVars = "Species",
                    ordVars = "Petal.Width",
                    idVars = "ID",
                    dropVars = "Junk",
                    groupVars = "Species")

## Pull the 'data' field from 'newData':
inspect(object = newData, what = "data")
```

`iris2` *A modified version of the Fisher/Anderson iris data.*

Description

This is a slight modification of the famous Fisher/Anderson iris data. I've binned petal width and added an ID and junk variable to demonstate the usage of **package:quark** more effectively.

Usage

```
data("iris2")
```

Format

A data frame with 150 observations on the following 7 variables describing the characteristics of a sample of three species of iris.

`ID` A numeric vector of IDs

`Sepal.Length` A numeric vector of sepal lengths

`Sepal.Width` A numeric vector of sepal widths

`Petal.Length` A numeric vector of petal lengths

`Petal.Width` An ordered factor with levels 1 < 2 < 3 < 4 < 5 giving a categorized measure of petal width

`Species` A factor with levels `setosa` `versicolor` `virginica` giving the iris' species

`Junk` A constant nusiance factor with levels `badVar`

Source

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7, 179-188.

The data were collected by: Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59, 2-5.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The new s language. Wadsworth & Brooks/Cole.

Examples

```
data(iris2)
```

`makePredMatrix` *Make a predictor matrix for use with* **mice**.

Description

Make a predictor matrix for use with **mice** that correctly specifies the auxiliary principal component scores produced by `createPcAux` as the sole predictors in the imputation model.

Usage

```
makePredMatrix(mergedData, nLinear, nNonLinear)
```

Arguments

- `mergedData` A data frame, such as one returned by `quark::mergePcAux`, containing the incomplete variables to be imputed and the principal component auxiliary variable scores.
- `nLinear` The number of linear principal component auxiliaries to use as predictors in the imputation model. If not specified, all linear PcAux scores contained in `mergedData` will be used.
- `nNonLinear` The number of non-linear principal component auxiliaries to use as predictors in the imputation model. If not specified, all non-linear PcAux scores contained in `mergedData` will be used.

Value

A pattern matrix with dimensions: `c(ncol(mergedData), ncol(mergedData))` that can be supplied to the `predictorMatrix` argument of **mice**.

Author(s)

Kyle M. Lang

See Also

`miWithPcAux`

Examples

```
## Load the data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create principal component auxiliary variables:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )

## Merge the PC auxiliaries with the original data:
outData <- mergePcAux(quarkData = pcAuxOut, rawData = iris2)

## Create a predictor matrix:
predMat <- makePredMatrix(mergedData = outData$data)
```

`mergePcAux` *Merge Principal Component Auxiliaries with the raw data from which they were constructed.*

Description

Merge Auxiliary Principal Component scores produced by `createPcAux` with a data frame containing the raw data from which the component scores were constructed.

Usage

```
mergePcAux(quarkData,
           rawData,
           nLinear = NULL,
           nNonLinear = NULL,
           varExpLin = NULL,
           varExpNonLin = NULL,
           verbose = TRUE)
```

Arguments

`quarkData`	An object of class *QuarkData* produced by a call to `createPcAux`.
`rawData`	A data frame containing the raw data used to run `createPcAux`.
`nLinear`	An optional integer giving the number of linear principal component auxiliary scores to merge onto `rawData`. If unspecified, all linear principal component scores that exist in `quarkData` are used.
`nNonLinear`	An optional integer giving the number of non-linear principal component auxiliary scores to merge onto `rawData`. If unspecified, all non-linear principal component scores that exist in `quarkData` are used.
`varExpLin`	An optional real number giving the proportion of variance in `rawData` to be explained by the set of linear principal component auxiliary scores merged onto `rawData` (see 'Details' for a description of the implementation). If unspecified, the `nLinear` argument defines the number of linear component scores used.
`varExpNonLin`	An optional real number giving the proportion of variance in `rawData` to be explained by the set of non-linear principal component auxiliary scores merged onto `rawData` (see 'Details' for a description of the implementation). If unspecified, the `nNonLinear` argument defines the number of non-linear component scores used.
`verbose`	A logical flag indicating whether verbose output should be printed to stdout. Defaults to `verbose = TRUE`.

Details

This function will attempt to use the ID variables defined in Quark's `idVars` argument to align rows for merging. If these ID variables are not suitable (i.e., because they don't exist in the raw data or they're not unique row-identifiers), the merging will be accomplished via naive column-binding.

When non-zero values are provided for varExpLin and/or varExpNonLin, the number of linear and/or non-linear principal component auxiliary scores that are merged onto rawData is taken to be the minimum number needed to explain *at least* as much variance in rawData as the proportions given by varExpLin and/or varExpNonLin. This means that the actual proportions of variance explain by the selected number of components will usually be slightly higher than the threshold requested, because the existence of a set of principal component scores that explain exactly the requested proportion of variance is unlikely.

Value

A data frame with (a subset of) the principal component auxiliary scores from quarkData$pcAux merged onto the end of the raw data.

Author(s)

Kyle M. Lang

Examples

```
## Load the data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create principal component auxiliary variables:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )

## Merge the PC auxiliaries with the original data:
outData <- mergePcAux(quarkData = pcAuxOut, rawData = iris2)
```

---

miWithPcAux — *Create multiple imputations using the PcAux produced by* createPcAux.

Description

Create multiple imputations with the **mice** package using the principal component auxiliary variable scores produced by createPcAux as the predictors in the imputation model.

Usage

```
miWithPcAux(rawData,
            quarkData,
            nImps = 100L,
            nomVars = NULL,
            ordVars = NULL,
            idVars = NULL,
            dropVars = "useExtant",
            nLinear = NULL,
            nNonLinear = NULL,
            varExpLin = NULL,
            varExpNonLin = NULL,
            completeFormat = "list",
            mySeed = 235711L,
            simMode = FALSE,
            forcePmm = FALSE,
            useParallel = FALSE,
            nProcess = 1L,
            verbose = !simMode,
            control)
```

Arguments

`rawData` A data frame containing the incomplete data for which to create the multiple imputations.

`quarkData` An object of class `QuarkData` produced by a run of `createPcAux`.

`nImps` An integer giving the number of imputations to create. Defaults to `nImps = 100L`.

`nomVars` An optional character vector containing names of any nominal variables (i.e., unordered factors) that exist in `rawData`. If unspecified, any `nomVars` defined in `quarkData` will be used.

`ordVars` An optional character vector containing names of any ordinal variables (i.e., ordered factors) that exist in `rawData`. If unspecified, any `ordVars` defined in `quarkData` will be used.

`idVars` An optional character vector containing names of any ID variables that exist in `rawData`. Any columns flagged as ID variables should not be represented in `nomVars`, `ordVars`, `dropVars`, or `groupVars`. If unspecified, any `idVars` defined in `quarkData` will be used.

`dropVars` An optional character vector containing names of any nuisance variables that should be excluded from the imputaiton process. If unspecified, the default value of `dropVars = "useExtant"` causes any user-defined `dropVars` defined in `quarkData` to be used.

`nLinear` An integer giving the number of linear principal component auxiliary scores to use as predictors in the imputation model. If unspecified, all linear principal component auxiliary scores contained in `quarkData` will be used.

`nNonLinear` An integer giving the number of non-linear principal component auxiliary scores to use as predictors in the imputation model. If unspecified, all non-linear principal component auxiliary scores contained in `quarkData` will be used.

varExpLin An optional real number giving the proportion of variance to be explained by the linear principal component auxiliary scores (see 'Details' for a description of the implementation). This argument overrides nLinear. If unspecified, the nLinear argument will define the number of linear principal component scores used.

varExpNonLin An optional real number giving the proportion of variance to be explained by the non-linear principal component auxiliary scores (see'Details' for a description of the implementation). This argument overrides nNonLinear. If unspecified, the nNonLinear argument will define the number of non-linear principal component scores used.

completeFormat The format in which the multiply-imputed data sets are returned. Valid arguments are "list", which returns a list of length nImps with each entry containing one imputed data set, "long", "broad", and "repeated". The latter three options are passed directly to the action argument of the mice::complete function. See the documentation for mice::complete for more details on the behavior of the "long", "broad", and "repeated" options. Defaults to completeFormat = "list".

mySeed An integer used to seed the random number generator used by the imputation algorithm. Defaults to mySeed = 235711L.

simMode A logical switch turning 'Simulation Mode' on or off. In Simulation Mode all of the automatic data checks will be suppressed. This mode is intended for use when miWithPcAux is being called as part of a Monte Carlo simulation study in which the data properties are well-known by the user. This mode should not be used for 'real-world' data analysis. Defaults to simMode = FALSE.

forcePmm A logical flag indicating whether or not the imputation should use predictive mean matching as the elementary imputation method for all variables. If forcePmm == FALSE, the elementary imputation methods are chosen to match each variable's declared type. Defaults to forcePmm = FALSE.

useParallel A logical switch indicating if the multiple imputation should be executed using parallel processing. When set to TRUE, the imputaiton job will be executed in nProcess parallel jobs using the parLapply function from the **parallel** package. Defaults to useParallel = FALSE.

nProcess An integer that gives the number of parallel processes to use when useParallel == TRUE. Must be less than or equal to the number of available logical processor cores. Defaults to nProcess = 1L.

verbose A logical switch indicating whether output should be printed to the screen. Warnings are always printed, regardless of the value assigned to verbose. Defaults to verbose = !simMode.

control An optional list of control parameters (see 'Details').

Details

When non-zero values are provided for varExpLin and/or varExpNonLin, the number of linear and/or non-linear principal component auxiliary scores used as predictors in the imputation model is taken to be the minimum number needed to explain *at least* as much variance in rawData as the proportions given by varExpLin and/or varExpNonLin. This means that the actual proportions of variance explain by the selected number of components will usually be slightly higher than the threshold requested, because the existence of a set of principal component scores that explain exactly the requested proportion of variance is unlikely.

The `control` argument is a key-paired list with the following possible entries:

- miceRidge: Value of the ridge penalty parameter used to stabalize the imputation models used by mice(). Defaults to `miceRidge = 1e-5`.
- minRespCount: The minimum number of observations allowed on each variable without triggering a warning. Defaults to `floor(0.05 * nrow(rawData))`.
- maxNetWts: The maximum number of network weights used by nnet() to fit the polytimous regression models used to impute nominal variables with mice(). Defaults to `maxNetWts = 10000L`.
- nomMaxLev: The maximum number of response levels for nominal variables that won't trigger a warning. Defaults to `nomMaxLev = 10L`.
- ordMaxLev: The maximum number of response levels for ordinal variables that won't trigger a warning. Defaults to `ordMaxLev = 10L`.
- conMinLev: The minimum number of unique responses for continuous variables that won't trigger a warning. Defaults to `minConLev = 10L`.

Value

A Reference Class object of class QuarkData with all of the fields from the object provided to the `quarkData` argument preserved, new fields for each of the `miWithPcAux` function's arguments and the following modified or additional fields:

- call: A list containing the matched function call to `miWithPcAux`.
- miDatasets: The completed, multiply imputed data sets. The structure of this field's contents is dictated by the `compFormat` argument to `miWithPcAux`.
- miceObject: The `mids` object returned by **mice** in the process of creating the multiple imputations of `rawData`.
- nComps: An integer vector of length 2 that contains the number of linear and non-linear, respectively, principal component auxiliary variable scores used as predictors in the multiple imputation models.
- typeVec: A character vector giving the types assigned to each variable in `rawData`.
- methVec: A character vector giving the elementary imputation methods used by **mice**.
- respCounts: An integer vector giving the variable-wise response counts for `rawData`.
- initialPm: A numeric vector giving the initial, variable-wise percents missing for `rawData`, before any treatment.
- dropVars: A two-column character matrix. The first column contains the names of all variables that were excluded from the imputation process (these variables appear in their original, incomplete, form in the multiply imputed data sets). The second column contains the reason that the corresponding variable was excluded.
- probNoms: A character vector giving the variable names for any nominal variables with more levels than `control$nomMaxLev`.
- probOrds: A character vector giving the variable names for any ordinal variables with more levels than `control$ordMaxLev`.

- probCons: A character vector giving the variable names for any continuous variables with fewer levels than `control$conMinLev`.
- levelVec: An integer vector giving the number of unique, non-missing, levels for each column of `rawData`.
- highPmVars: A character vector containing the names of variables with fewer observed responses than `control$minRespCount`.
- emptyVars: A character vector giving the names of empty columns in `rawData`.
- constants: A character vector giving the names of constant columns in `rawData`.

Author(s)

Kyle M. Lang & Stephen Chesnut

See Also

createPcAux

Examples

```
## Load the data:
data(iris2)

## Prepare the data:
cleanData <- prepData(rawData = iris2,
                      nomVars = "Species",
                      ordVars = "Petal.Width",
                      idVars = "ID",
                      dropVars = "Junk",
                      groupVars = "Species")

## Create principal component auxiliary variables:
pcAuxOut <- createPcAux(quarkData = cleanData,
                        nComps = c(3, 2)
                        )

## Conduct MI with the pcAux:
miOut <- miWithPcAux(rawData = iris2,
                     quarkData = pcAuxOut,
                     nImps = 5)
```

`prepData` *Prepare Data for Extracting Principal Component Auxiliaries*

Description

Data cleaning to facilitate implementation the ideas of Howard, Rhemtulla, and Little (2015).

Usage

```
prepData(rawData,
         nomVars = NULL,
         ordVars = NULL,
         idVars = NULL,
         dropVars = NULL,
         groupVars = NULL,
         simMode = FALSE,
         mySeed = 235711L,
         verbose = !simMode,
         control,
         ...)
```

Arguments

rawData	A data frame from which to extract the auxiliary pricipal components.
nomVars	An optional character vector containing names of any nominal variables (i.e., unordered factors) that exist in `rawData`.
ordVars	An optional character vector containing names of any ordinal variables (i.e., ordered factors) that exist in `rawData`.
idVars	An optional character vector containing names of any ID variables that exist in `rawData`. Any columns flagged as ID variables should not be represented in `nomVars`, `ordVars`, `dropVars`, or `groupVars`
dropVars	An optional character vector containing names of any nuisance variables that should be dropped before extracting the auxiliary principal component scores.
groupVars	An optional character vector containing names of any grouping variables that can be used to create the strata that define the groups used by the fall-back group-mean substitution. If continuous grouping variables are specified, they are binned via the `cut()` function with `breaks = control$nGVarCuts`.
simMode	A logical switch turning 'Simulation Mode' on or off. In Simulation Mode all of the automatic data checks will be suppressed. This mode is intended for use when `prepData` is being called as part of a Monte Carlo simulation study in which the data properties are well-known by the user. This mode should not be used for 'real-world' data analysis. Defaults to `simMode = FALSE`.
mySeed	An integer used to seed the random number generator used by the imputation algorithm. Defaults to `mySeed = 235711L`.
verbose	A logical switch indicating whether output should be printed to the screen. Warnings are always printed, regardless of the value assigned to `verbose`. Defaults to `verbose = !simMode`.
control	An optional list of control parameters (see 'Details').
...	Not currently used.

Details

The `control` argument is a key-paired list with the following possible entries:

- miceIters: Number of EM iterations supplied to the `maxit` argument of mice() during the initial single imputation. Defaults to `miceIters = 10L`.
- miceRidge: Value of the ridge penalty parameter used to stabalize the imputation models used by mice(). Defaults to `miceRidge = 1e-5`.
- collinThresh: The strength of linear association used to flag collinear variable for removal. Defaults to `collinThresh = 0.95`.
- minRespCount: The minimum number of observations allowed on each variable without triggering a warning. Defaults to `floor(0.05 * nrow(rawData))`.
- minPredCor: The minimum magnitude of correlation supplied to the `mincor` argument of mice::quickpred() when constructing the predictor matrix used by mice() during the initial single imputation. Defaults to `minPredCor = 0.1`.
- maxNetWts: The maximum number of network weights used by nnet() to fit the polytimous regression models used to impute nominal variables with mice(). Defaults to `maxNetWts = 10000L`.
- nomMaxLev: The maximum number of response levels for nominal variables that won't trigger a warning. Defaults to `nomMaxLev = 10L`.
- ordMaxLev: The maximum number of response levels for ordinal variables that won't trigger a warning. Defaults to `ordMaxLev = 10L`.
- conMinLev: The minimum number of unique responses for continuous variables that won't trigger a warning. Defaults to `minConLev = 10L`.
- nGVarCats: The number of categories into which continuous grouping variables will be split, if applicable. Defaults to `nGVarCats = 3L`.

Value

An Reference Class object of class QuarkData with fields for each of the `prepData` function's arguments and the following additional, non-trivial fields:

- call: A list containing the matched function call to `prepData`.
- typeVec: A character vector giving the types assigned to each variable in `rawData`.
- initialPm: A numeric vector giving the initial, variable-wise percents missing for `rawData` before any treatment.
- dropVars: A two-column character matrix. The first column contains the names of all variables dropped from the analysis. The second column contains the reason that the corresponding variable was dropped.
- probNoms: A character vector giving the variable names for any nominal variables with more levels than `control$nomMaxLev`.
- probOrds: A character vector giving the variable names for any ordinal variables with more levels than `control$ordMaxLev`.
- probCons: A character vector giving the variable names for any continuous variables with fewer levels than `control$conMinLev`.
- levelVec: An integer vector giving the number of unique, non-missing, levels for each variable in `rawData`.
- highPmVars: A character vector containing the names of variables with fewer observed responses than `control$minRespCount`.

- emptyVars: A character vector giving the names of empty columns in `rawData`.
- constants: A character vector giving the names of constant columns in `rawData`.
- collinVars: A three-column character matrix. The first two columns contain the names of pairs of approximately collinear variables. The third column contains their observed linear association.
- idFills: A list containing the values used to deterministically fill any missing data that occured on the ID variables. The length of this argument will equal the number of incomplete ID variables in `rawData`.

Author(s)

Kyle M. Lang & Steven Chesnut

References

Howard, W. H., Rhemtulla, M., & Little, T. D. (2015). Using principal components as auxiliary variables in missing data estimation. Multivariate Behavioral Research. 50(3). 285-299.

See Also

`createPcAux`, `miWithPcAux`

Examples

```
## Load data:
data(iris2)

## Prepare the data:
newData <- prepData(rawData = iris2,
                    nomVars = "Species",
                    ordVars = "Petal.Width",
                    idVars = "ID",
                    dropVars = "Junk",
                    groupVars = "Species")
```

---

quarkW — *Print **quark**'s warranty statement.*

Description

Print the sections of the GPL-3 that describe the warranty (or complete lack thereof) for **quark**.

Usage

```
quarkW()
```

Value

Text giving the warranty-specific sections of the GPL-3.

Author(s)

Kyle M. Lang

Examples

```
## Check quark's warranty:
quarkW()
```

Index