Package 'bigMICE'

May 7, 2026

Type Package

Title Multiple Imputation of Big Data

Version 1.0.0

Description
A computational toolbox designed for handling missing values in large datasets with the Multiple Imputation by Chained Equations (MICE) by using 'Apache Spark'. The methodology is described in Morvan et al. (2026) <doi:10.48550/arXiv.2601.21613>.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements Spark: 3.x, or 4.x

Imports dplyr, tidyselect, rlang, sparklyr, data.table, Matrix

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Hugo Morvan [aut],
Oleg Sysoev [aut, cre]

Maintainer Oleg Sysoev <oleg.sysoev@liu.se>

Repository CRAN

Date/Publication 2026-02-25 10:20:14 UTC

Contents

`impute_with_linear_regression` *Linear Regression Imputation function*

Description

This function imputes missing values in a Spark DataFrame using linear regression.

Usage

```
impute_with_linear_regression(
  sc,
  sdf,
  target_col,
  feature_cols,
  elastic_net_param = 0,
  target_col_prev
)
```

Arguments

`sc`	A Spark connection
`sdf`	A Spark DataFrame
`target_col`	The column with missing values to impute
`feature_cols`	The columns to use as features in the linear regression model. These columns should not have missing values.
`elastic_net_param`	The elastic net parameter for the linear regression model. Default is 0 (ridge regression)
`target_col_prev`	the target column at the previous iteration. Used to calculate residuals.

Value

The Spark DataFrame with missing values imputed in the target column

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a simple dataset with missing values
library(bigMICE)
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with some missing values in 'age'
sample_data <- data.frame(
 age = c(25, NA, 35, NA, 45, 30),
 income = c(50000, 60000, 70000, 55000, 80000, 52000),
 education_years = c(16, 18, 20, 17, 22, 16),
 experience = c(3, 8, 12, 5, 18, 7)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Create previous iteration data (for residual calculation)
# In practice, this would be from a previous imputation step
sdf_prev <- sdf %>%
  mutate(age = ifelse(is.na(age), 30, age)) %>%  # Simple initial imputation
  select(age)

# Impute missing age values using income, education_years, and experience
imputed_sdf <- impute_with_linear_regression(
  sc = sc,
  sdf = sdf,
  target_col = "age",
  feature_cols = c("income", "education_years", "experience"),
  elastic_net_param = 0,  # Ridge regression
  target_col_prev = sdf_prev
)

# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`impute_with_logistic_regression`

Logistic Regression Imputation function

Description

This function imputes missing values in a Spark DataFrame using logistic regression. This function is intended for boolean variables only (0/1).

Usage

```
impute_with_logistic_regression(sc, sdf, target_col, feature_cols)
```

Arguments

sc	A Spark connection
sdf	A Spark DataFrame
target_col	The column with missing values to impute
feature_cols	The columns to use as features in the logistic regression model. These columns should not have missing values.

Value

The Spark DataFrame with missing values imputed in the target column

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a dataset with missing boolean values
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values in a boolean column 'has_degree'
sample_data <- data.frame(
  has_degree = c(1, NA, 0, NA, 1, 0),
  age = c(25, 35, 30, 28, 45, 22),
  income = c(50000, 75000, 45000, 52000, 90000, 35000),
  years_experience = c(2, 8, 5, 3, 15, 1)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Impute missing boolean values using age, income, and experience
imputed_sdf <- impute_with_logistic_regression(
  sc = sc,
  sdf = sdf,
  target_col = "has_degree",
  feature_cols = c("age", "income", "years_experience")
)
```

```
# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

---

impute_with_MeMoMe *Mean/Mode/Median Imputation function*

Description

This function imputes missing values in a Spark DataFrame using the mean, mode, or median of the observed values.

Usage

```
impute_with_MeMoMe(sc, sdf, column = NULL, impute_mode, printFlags = TRUE)
```

Arguments

sc	A Spark connection
sdf	A Spark DataFrame
column	The column(s) to impute. If NULL, all columns will be imputed
impute_mode	Which imputation method to use for each column. Options are "mean", "mode", "median", or "none"
printFlags	Wether or not to print the imputation process to the console. Default is TRUE.

Value

The Spark DataFrame with missing values imputed

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a dataset with various types of missing values
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values in different columns
sample_data <- data.frame(
```

```
  age = c(25, NA, 35, 28, NA, 45),
  salary = c(50000, 60000, NA, 55000, 80000, NA),
  department = c("Sales", NA, "IT", "Sales", "HR", "IT"),
  rating = c(4.2, 3.8, NA, 4.5, 3.9, NA)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Impute different columns using different methods
imputed_sdf <- impute_with_MeMoMe(
  sc = sc,
  sdf = sdf,
  column = c("age", "salary", "department", "rating"),
  impute_mode = c("median", "mean", "mode", "mean"),
  printFlags = TRUE
)

# View results
imputed_sdf %>% collect()

# Example 2: Impute only specific columns
partial_imputed_sdf <- impute_with_MeMoMe(
  sc = sc,
  sdf = sdf,
  column = c("age", "salary"),
  impute_mode = c("mean", "median"),
  printFlags = FALSE
)

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`impute_with_mult_logistic_regression` *Multinomial Logistic Regression Imputation function*

Description

This function imputes missing values in a Spark DataFrame using Multinomial Logistic regression.

Usage

```
impute_with_mult_logistic_regression(sc, sdf, target_col, feature_cols)
```

Arguments

sc	A Spark connection
sdf	A Spark DataFrame
target_col	The column with missing values to impute
feature_cols	The columns to use as features in the multinomial logistic regression model. These columns should not have missing values.

Value

The Spark DataFrame with missing values imputed in the target column

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a dataset with missing categorical values
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values in 'job_category'
sample_data <- data.frame(
 job_category = c("Manager", NA, "Analyst", "Developer", NA, "Manager"),
 years_experience = c(8, 3, 5, 2, 6, 10),
 salary = c(85000, 45000, 55000, 50000, 65000, 95000),
 education_level = c(3, 2, 3, 2, 3, 4),  # 1=High School, 2=Bachelor, 3=Master, 4=PhD
 age = c(35, 28, 30, 25, 32, 42)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Impute missing job categories using experience, salary, education, and age
imputed_sdf <- impute_with_mult_logistic_regression(
 sc = sc,
 sdf = sdf,
 target_col = "job_category",
 feature_cols = c("years_experience", "salary", "education_level", "age")
)

# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

impute_with_random_forest_classifier
Random Forest Classification Imputation function

Description

This function imputes missing values in a Spark DataFrame using Random Forest classification.

Usage

```
impute_with_random_forest_classifier(sc, sdf, target_col, feature_cols)
```

Arguments

sc	A Spark connection
sdf	A Spark DataFrame
target_col	The column with missing values to impute
feature_cols	The columns to use as features in the Random Forest regression model. These columns should not have missing values.

Value

The Spark DataFrame with missing values imputed in the target column

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for Random Forest Classifier
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing categorical values in 'neighborhood'
sample_data2 <- data.frame(
  neighborhood = c("Downtown", NA, "Suburbs", "Rural", NA, "Downtown"),
  price = c(450000, 280000, 320000, 180000, 380000, 420000),
  commute_time = c(10, 25, 35, 60, 15, 12),
  schools_nearby = c(5, 3, 4, 1, 4, 6),
  crime_rate = c(2.1, 1.5, 1.2, 0.8, 1.8, 2.3)
)

# Copy to Spark DataFrame
sdf2 <- copy_to(sc, sample_data2, "sample_data2")
```

```
# Impute missing neighborhood types using Random Forest classification
imputed_sdf2 <- impute_with_random_forest_classifier(
  sc = sc,
  sdf = sdf2,
  target_col = "neighborhood",
  feature_cols = c("price", "commute_time", "schools_nearby", "crime_rate")
)

# View results
imputed_sdf2 %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`impute_with_random_forest_regressor` *Random Forest Regression Imputation function*

Description

This function imputes missing values in a Spark DataFrame using Random Forest regression.

Usage

```
impute_with_random_forest_regressor(
  sc,
  sdf,
  target_col,
  feature_cols,
  target_col_prev,
  max_depth = 15
)
```

Arguments

`sc`	A Spark connection
`sdf`	A Spark DataFrame
`target_col`	The column with missing values to impute
`feature_cols`	The columns to use as features in the Random Forest regression model. These columns should not have missing values.
`target_col_prev`	the target column at the previous iteration. Used to calculate residuals.
`max_depth`	Parameter of ml_random_forest, see its documentation for more details.

Value

The Spark DataFrame with missing values imputed in the target column

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for Random Forest Regressor
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing continuous values in 'price'
sample_data <- data.frame(
 price = c(250000, NA, 180000, NA, 320000, 195000),
 bedrooms = c(3, 2, 2, 3, 4, 2),
 bathrooms = c(2, 1, 1, 2, 3, 1),
 sqft = c(1500, 900, 800, 1200, 2000, 850),
 age = c(10, 15, 25, 8, 5, 20)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Create previous iteration data (for residual calculation)
sdf_prev <- sdf %>%
 mutate(price = ifelse(is.na(price), 200000, price)) %>%
 select(price)

# Impute missing house prices using Random Forest regression
imputed_sdf <- impute_with_random_forest_regressor(
 sc = sc,
 sdf = sdf,
 target_col = "price",
 feature_cols = c("bedrooms", "bathrooms", "sqft", "age"),
 target_col_prev = sdf_prev
)

# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

impute_with_random_samples
Random Sample Imputation function

Description

This function imputes missing values in a Spark DataFrame using random samples from the observed values.

Usage

```
impute_with_random_samples(sc, sdf, column = NULL)
```

Arguments

sc	A Spark connection
sdf	A Spark DataFrame
column	The column(s) to impute. If NULL, all columns will be imputed

Value

The Spark DataFrame with missing values imputed

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a dataset with various types of missing values
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values in different columns
sample_data <- data.frame(
  age = c(25, NA, 35, 28, NA, 45),
  salary = c(50000, 60000, NA, 55000, 80000, NA),
  department = c("Sales", NA, "IT", "Sales", "HR", "IT"),
  rating = c(4.2, 3.8, NA, 4.5, 3.9, NA)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Impute different columns using different methods
imputed_sdf <- impute_with_random_samples(
  sc = sc,
```

```
  sdf = sdf,
  column = c("age", "salary", "department", "rating")
)

# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`init_with_random_samples` *Random Sample Imputation function*

Description

This function imputes missing values in a Spark DataFrame using random samples from the observed values and enables checkpointing.

Usage

```
init_with_random_samples(
  sc,
  sdf,
  column = NULL,
  checkpointing = TRUE,
  checkpoint_frequency = 10
)
```

Arguments

`sc`	A Spark connection
`sdf`	A Spark DataFrame
`column`	The column(s) to impute. If NULL, all columns will be imputed
`checkpointing`	Default TRUE. Can be set to FALSE if you are running the package without access to a HDFS directory for checkpointing. It is strongly recommended to keep it to TRUE to avoid Stackoverflow errors.
`checkpoint_frequency`	Advanced parameter, modify with care. If checkpointing = TRUE, how often to checkpoint , default = 10, so after processing every 10 variables, the lineage will be cut and the current state of computation will be save to disk. A low number might slow down computation but enable bigger computation. A number too high (or not checkpoiting) might cause JVM stackOverflowError as the lineage will have grown too big.

Value

The Spark DataFrame with missing values imputed

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Create a dataset with various types of missing values
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values in different columns
sample_data <- data.frame(
  age = c(25, NA, 35, 28, NA, 45),
  salary = c(50000, 60000, NA, 55000, 80000, NA),
  department = c("Sales", NA, "IT", "Sales", "HR", "IT"),
  rating = c(4.2, 3.8, NA, 4.5, 3.9, NA)
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Impute different columns using different methods
imputed_sdf <- init_with_random_samples(
  sc = sc,
  sdf = sdf,
  column = c("age", "salary", "department", "rating"),
  checkpointing = F
)

# View results
imputed_sdf %>% collect()

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`mice.spark` *MICE for Spark DataFrames using Sparklyr and Spark MLlib*

Description

This function imputes missing values in a Spark DataFrame using MICE (Multiple Imputation by Chained Equations) algorithm.

Usage

```
mice.spark(
  data,
  sc,
  variable_types,
  analysis_formula,
  m = 5,
  method = NULL,
  predictorMatrix = NULL,
  formulas = NULL,
  modeltype = NULL,
  maxit = 5,
  printFlag = TRUE,
  seed = NA,
  imp_init = NULL,
  checkpointing = TRUE,
  checkpoint_frequency = 10,
  ...
)
```

Arguments

data	A Spark DataFrame
sc	A Spark connection
variable_types	A named character vector, the variable types of the columns in the data.
analysis_formula	A formula, the formula to use for the analysis
m	The number of imputations to perform
method	A character vector, the imputation method to use for each variable. If NULL, the function will infer the method based on the variable types.
predictorMatrix	A matrix, the predictor matrix to use for the imputation.
formulas	A list, the formulas to use for the imputation. If NULL, the function will infer the formulas based on the other variables present in the data.
modeltype	A character vector, the model type to use for the imputation. If NULL, the function will infer the model type based on the variable types.
maxit	The maximum number of iterations to perform
printFlag	A boolean, whether to print debug information
seed	An integer, the seed to use for reproducibility
imp_init	A Spark DataFrame, the original data with missing values, but with initial imputation (by random sampling or mean/median/mode imputation). Can be set to avoid re-running the initialisation step. Otherwise, the function will perform the initialisation step using the MeMoMe function.

checkpointing Default TRUE. Can be set to FALSE if you are running the package without access to a HDFS directory for checkpointing. It is strongly recommended to keep it to TRUE to avoid Stackoverflow errors.

checkpoint_frequency
Advanced parameter, modify with care. If checkpointing = TRUE, how often to checkpoint , default = 10, so after processing every 10 variables, the lineage will be cut and the current state of computation will be save to disk. A low number might slow down computation but enable bigger computation. A number too high (or not checkpoiting) might cause JVM stackOverflowError as the lineage will have grown too big.

... Additional arguments to be passed to the function.

Value

A list containing the Rubin's statistics for the model parameters, the per-imputation statistics, the imputation statistics, and the model parameters.

References

Morvan H, Agholme J, Eliasson B, Olofsson K, Grote L, Iredahl F, Sysoev O. bigMICE: Multiple Imputation of Big Data. arXiv:2601.21613 [stat.CO]. 2026. doi:10.48550/arXiv.2601.21613

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for mice.spark function
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values
sample_data <- data.frame(
  outcome = c(1, 0, NA, 1, NA, 0),
  age = c(25, NA, 35, 28, 45, NA),
  income = c(50000, 60000, NA, 55000, 80000, 52000),
  education = c("High", "Medium", "High", NA, "Medium", "Medium")
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Define variable types
variable_types <- c(
  outcome = "Binary",
  age = "Continuous_int",
  income = "Continuous_int",
  education = "Nominal"
```

```
)

# Define analysis formula
analysis_formula <- outcome ~ age + income + education

# Run MICE imputation
mice_result <- mice.spark(
  data = sdf,
  sc = sc,
  variable_types = variable_types,
  analysis_formula = analysis_formula,
  m = 3,  # Number of imputations
  maxit = 2,  # Number of iterations
  printFlag = TRUE,
  seed = 123,
  checkpointing = FALSE  # Set to TRUE if HDFS is available
)

# See results
 print(mice_result)
# Clean up
 spark_disconnect(sc)

## End(Not run)
```

`mice.spark.plus` *MICE+ for Spark DataFrames using Sparklyr and Spark MLlib*

Description

This function imputes missing values in a Spark DataFrame using MICE (Multiple Imputation by Chained Equations) algorithm. Additionally, it allows to look at the imputed values to see if they are reasonable and measure the uncertainty of the imputation.

Usage

```
mice.spark.plus(
  data,
  sc,
  variable_types,
  analysis_formula,
  where_missing,
  m = 5,
  method = NULL,
  predictorMatrix = NULL,
  formulas = NULL,
  modeltype = NULL,
  maxit = 5,
  printFlag = TRUE,
```

```
  seed = NA,
  imp_init = NULL,
  checkpointing = TRUE,
  checkpoint_frequency = 10,
  ...
)
```

Arguments

data	A Spark DataFrame, the original data with extra missing values
sc	A Spark connection
variable_types	A named character vector, the variable types of the columns in the data.
analysis_formula	A formula, the formula to use for the analysis
where_missing	A logical vector, the locations of the missing values in the data
m	The number of imputations to perform
method	A character vector, the imputation method to use for each variable. If NULL, the function will infer the method based on the variable types.
predictorMatrix	A matrix, the predictor matrix to use for the imputation.
formulas	A list, the formulas to use for the imputation. If NULL, the function will infer the formulas based on the other variables present in the data.
modeltype	A character vector, the model type to use for the imputation. If NULL, the function will infer the model type based on the variable types. The methods specified must match the order of the variables and must be one of "Logistic","Mult_Logistic","Linear","RandomForestClassifier","RandomForestRegressor" or "none".
maxit	The maximum number of iterations to perform
printFlag	A boolean, whether to print debug information
seed	An integer, the seed to use for reproducibility
imp_init	A Spark DataFrame, the original data with missing values, but with initial imputation (by random sampling or mean/median/mode imputation). Can be set to avoid re-running the initialisation step. Otherwise, the function will perform the initialisation step using the MeMoMe function.
checkpointing	Default TRUE. Can be set to FALSE if you are running the package without access to a HDFS directory for checkpointing. It is strongly recommended to keep it to TRUE to avoid Stackoverflow errors.
checkpoint_frequency	Advanced parameter, modify with care. If checkpointing = TRUE, how often to checkpoint , default = 10, so after processing every 10 variables, the lineage will be cut and the current state of computation will be save to disk. A low number might slow down computation but enable bigger computation. A number too high (or not checkpoiting) might cause JVM stackOverflowError as the lineage will have grown too big.
...	Additional arguments to be passed to the function.

Value

A list containing the Rubin's statistics for the model parameters, the per-imputation statistics, the imputation statistics, and the model parameters.

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for mice.spark.plus function
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values
sample_data <- data.frame(
  outcome = c(1, 0, NA, 1, NA, 0),
  age = c(25, NA, 35, 28, 45, NA),
  income = c(50000, 60000, NA, 55000, 80000, 52000),
  education = c("High", "Medium", "High", NA, "Medium", "Medium")
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Define variable types
variable_types <- c(
  outcome = "Binary",
  age = "Continuous_int",
  income = "Continuous_int",
  education = "Nominal"
)

# Define analysis formula
analysis_formula <- outcome ~ age + income + education

# Create complete data (without extra missing values)
complete_data <- data.frame(
  outcome = c(1, 0, 1, 1, 0, 0),
  age = c(25, 30, 35, 28, 45, 32),
  income = c(50000, 60000, 70000, 55000, 80000, 52000),
  education = c("High", "Medium", "High", "Low", "Low", "Medium")
)

# Copy complete data to Spark
sdf_complete <- copy_to(sc, complete_data, "complete_data")

# Create where_missing indicator (logical vector)
where_missing <- c(FALSE, TRUE, TRUE, FALSE, TRUE, TRUE)  # Indicates artificially missing
```

```
# Run MICE+
mice_plus_result <- mice.spark.plus(
  data = sdf,  # Data with missing values
  data_true = sdf_complete,  # Complete data
  sc = sc,
  variable_types = variable_types,
  analysis_formula = analysis_formula,
  where_missing = where_missing,
  m = 3,
  maxit = 2,
  printFlag = TRUE,
  seed = 123,
  checkpointing = FALSE
)

# View results
mice_plus_result$rubin_stats

# Clean up
spark_disconnect(sc)

## End(Not run)
```

`print.mi_results` *Print Method for Multiple Imputation Results*

Description

Print Method for Multiple Imputation Results

Usage

```
## S3 method for class 'mi_results'
print(x, digits = 4, show_individual = FALSE, ...)
```

Arguments

`x`	A list containing multiple imputation results with Rubin's statistics
`digits`	Number of digits to display for numeric values (default: 4)
`show_individual`	Logical, whether to show individual imputation results (default: FALSE)
`...`	Additional arguments (not currently used)

Value

No return value, called for side effects

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for mice.spark function
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values
sample_data <- data.frame(
  outcome = c(1, 0, NA, 1, NA, 0),
  age = c(25, NA, 35, 28, 45, NA),
  income = c(50000, 60000, NA, 55000, 80000, 52000),
  education = c("High", "Medium", "High", NA, "Medium", "Medium")
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Define variable types
variable_types <- c(
  outcome = "Binary",
  age = "Continuous_int",
  income = "Continuous_int",
  education = "Nominal"
)

# Define analysis formula
analysis_formula <- outcome ~ age + income + education

# Run MICE imputation
mice_result <- mice.spark(
  data = sdf,
  sc = sc,
  variable_types = variable_types,
  analysis_formula = analysis_formula,
  m = 3,  # Number of imputations
  maxit = 2,  # Number of iterations
  printFlag = TRUE,
  seed = 123,
  checkpointing = FALSE  # Set to TRUE if HDFS is available
)

# See results
 print(mice_result)
# Clean up
 spark_disconnect(sc)

## End(Not run)
```

`sampler.spark` *MICE sampler function*

Description

This function is the core of the MICE algorithm. It iteratively imputes missing values in a Spark DataFrame using a set of imputation methods based on the variable types.

Usage

```
sampler.spark(
  sc,
  data,
  imp_init,
  fromto,
  var_types,
  ud_methods = NULL,
  predictorMatrix = NULL,
  checkpointing,
  checkpoint_frequency = 10,
  printFlag
)
```

Arguments

`sc`	A Spark connection
`data`	A Spark DataFrame, the original data with missing values
`imp_init`	A Spark DataFrame, the original data with missing values, but with initial imputation (by random sampling or mean/median/mode imputation)
`fromto`	A vector of length 2, the range of iterations to perform (from, to)
`var_types`	A named character vector, the variable types of the columns in the data.
`ud_methods`	The user-defined methods for imputing each variables. Beta
`predictorMatrix`	A matrix, the predictor matrix to use for the imputation. Beta
`checkpointing`	Default TRUE. Can be set to FALSE if you are running the package without access to a HDFS directory for checkpointing. It is strongly recommended to keep it to TRUE to avoid Stackoverflow errors.
`checkpoint_frequency`	Advanced parameter, modify with care. If checkpointing = TRUE, how often to checkpoint , default = 10, so after processing every 10 variables, the lineage will be cut and the current state of computation will be save to disk. A low number might slow down computation but enable bigger computation. A number too high (or not checkpoiting) might cause JVM stackOverflowError as the lineage will have grown too big.
`printFlag`	A boolean, whether to print debug information.

Value

The Spark DataFrame with missing values imputed for all variables

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for sampler.spark function
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values
sample_data <- data.frame(
  age = c(25, NA, 35, 28, 45, NA),
  income = c(50000, 60000, NA, 55000, 80000, 52000),
  education = c("High", "Medium", "High", NA, "Medium", "Medium")
)

# Copy to Spark DataFrame
sdf <- copy_to(sc, sample_data, "sample_data")

# Define variable types for sampler
var_types <- c(
  age = "Continuous_int",
  income = "Continuous_int",
  education = "Nominal"
)

# Create initial imputation (simple mean/mode)
imp_init <- sdf %>%
  mutate(
  age = ifelse(is.na(age), 35, age),
    income = ifelse(is.na(income), 60000, income),
    education = ifelse(is.na(education), "Medium", education)
  )

# Run sampler
sampled_data <- sampler.spark(
  sc = sc,
  data = sdf,
  imp_init = imp_init,
  fromto = c(1, 2),
  var_types = var_types,
  printFlag = TRUE,
  checkpointing = FALSE
)

# View results
```

```
sampled_data %>% collect()

 # Clean up
 spark_disconnect(sc)

## End(Not run)
```

---

summary.mi_results *Summary Method for Multiple Imputation Results*

Description

Summary Method for Multiple Imputation Results

Usage

```
## S3 method for class 'mi_results'
summary(object, ...)
```

Arguments

object	A list containing multiple imputation results
...	Additional arguments passed to print method

Value

No return value, called for side effects

Examples

```
# This example is not executed since it needs additional software (Apache Spark)
## Not run:
# Example for mice.spark function
library(sparklyr)
library(dplyr)

# Connect to Spark
# Assumes that you have already installed Spark with sparklyr::spark_install()
sc <- spark_connect(master = "local")

# Create sample data with missing values
sample_data <- data.frame(
  outcome = c(1, 0, NA, 1, NA, 0),
  age = c(25, NA, 35, 28, 45, NA),
  income = c(50000, 60000, NA, 55000, 80000, 52000),
  education = c("High", "Medium", "High", NA, "Medium", "Medium")
)

# Copy to Spark DataFrame
```

```
sdf <- copy_to(sc, sample_data, "sample_data")

# Define variable types
variable_types <- c(
  outcome = "Binary",
  age = "Continuous_int",
  income = "Continuous_int",
  education = "Nominal"
)

# Define analysis formula
analysis_formula <- outcome ~ age + income + education

# Run MICE imputation
mice_result <- mice.spark(
  data = sdf,
  sc = sc,
  variable_types = variable_types,
  analysis_formula = analysis_formula,
  m = 3,  # Number of imputations
  maxit = 2,  # Number of iterations
  printFlag = TRUE,
  seed = 123,
  checkpointing = FALSE  # Set to TRUE if HDFS is available
)

# See results
summary(mice_result)
# Clean up
spark_disconnect(sc)

## End(Not run)
```

Index