arXiv:2601.21613v1 [stat.CO] 29 Jan 2026

bigMICE：大数据多重插补

Hugo Morvan Jonas Agholme Björn Eliasson KatarinaOlofsson林雪平大学 林雪平大学 哥德堡大学 RegionVästraGötaland

Ludger Grote
哥德堡大学

Fredrik Iredahl[†]
林雪平大学

Oleg Sysoev[†*]
林雪平大学

摘要

缺失数据是许多应用中普遍存在的问题，包括瑞典医疗质量登记系统等大型医疗登记系统。如果处理不当，可能导致分析出现偏差或效率低下。链式方程多重插补（MICE）是一种广受欢迎且用途广泛的多元缺失数据处理方法，但传统实现应用于大数据集时，会面临计算时间和内存限制带来的重大挑战。

为解决这一问题，我们开发了 bigMICE 软件包，利用 Apache Spark MLLib 和 Spark ML 将 MICE 框架扩展至大数据领域。我们的实现支持控制执行期间的最大内存使用量，使得在普通笔记本电脑等内存有限的硬件上也能处理非常大型的数据集。

我们在一个大型瑞典医疗登记数据集上测试了该软件包，以测量内存使用量、运行时间，以及插补质量与样本量和数据缺失比例之间的关系。总之，与常用的 MICE 实现相比，我们的方法处理大型数据集时通常更节省内存、速度也更快。我们还证明，即使某个变量的缺失数据比例很高，处理非常大型的数据集仍能获得高质量的插补结果。本文还提供了有关如何安装和使用这一开源软件包的指南和建议。

关键词：缺失数据、多重插补、MICE、大数据、Apache Spark、R。

1. 引言

缺失数据是许多应用中普遍存在的问题，可能严重阻碍科学与工程领域中的数据分析Yang、Kim 和 Cho （2020）。在研究中，数据集中某些属性缺少值是常见现象Wang、Shakhovska、Sachenko 和 Komar（2020）；Austin、White、Lee 和 van Buuren（2020）；Arefin 和 Masum （2024），其成因多种多样，包括数据录入过程中的文书错误、系统故障、参与者未披露信息、参与者中途退出研究、数据损坏或数据收集程序的差异Austin 等 （2020）；Cesare 和 Were （2022）；Arefin 和 Masum（2024）。缺失数据的存在带来了重大

†：共同未位作者
*：通讯作者，电子邮箱：oleg.sysoev@liu.se

挑战，因为许多算法要求指标矩阵完整且不存在空缺Wang等 （2020），而从不完整数据集中得出的统计结果，尤其是包含非随机缺失值的数据集，可能存在偏差Hasan、Alam、Roy、Dutta、Jawad 和 Das（2021）。

1.1. 问题陈述

社会中的许多大数据集，包括医疗和行政登记库，都是研究和决策的宝贵资源。然而，这些数据集往往含有缺失值。传统的缺失数据处理方法已被证明不适用于大数据情境。处理缺失数据的典型方法包括个案删除（也称为完整个案分析或列表删除），以及简单插补方法，例如用均值、中位数或众数替换缺失值Pampaka、Hutcheson 和 Williams (2016)；Grund、Lüdtke 和 Robitzsch(2016)；Austin 等 (2020）。然而，这些单次插补方法没有考虑缺失数据所带来的不确定性，并且可能会缩小方差和协方差，从而导致置信区间人为变窄、I 类错误率升高 Austin等(2020）。

链式方程多重插补（MICE）是一种广受欢迎且灵活的缺失值插补方法vanBuuren (2007）。该方法通过为每个缺失数据点生成多个合理取值，创建多个用于分析的完整数据集。这样可以确保充分考虑预测值的不确定性，从而使后续结果更加有效、可靠。

尽管各种编程语言和统计软件包中都广泛提供 MICE 实现，但将这些方法应用于大规模数据集时，会出现一个关键限制。由于数据集规模庞大（有时达到数百 GB），且 MICE 具有迭代性质，将 MICE 应用于大数据集可能会导致计算时间极长。然而，主要问题在于，所需内存可能超过特定计算资源的随机存取存储器（RAM）容量，从而导致计算失败。随着现代社会不断收集新数据，数据登记库每年都在扩大，因此，能够为这些大型数据准确地进行插补至关重要。

1.2. 现有软件的局限性

各种编程语言和统计软件包中都有许多 MICE 实现。

R 统计计算环境提供了最全面的 MICE 实现集合之一，其中包括**mice**软件包van Buuren 和 Groothuis-Oudshoorn（2011）的实现，该实现通常被用作 MICE 算法的参考实现。其他值得关注的R 实现包括用于并行计算的**parlMICE**、使用随机森林的**miceRanger** 、使用自助法和 EM 算法的**Amelia** ，以及Pampaka等人（2016）；Khan 和 Hoque （2020）；Little 和 Rubin （2019）；Lo、Siah 和 Wong （2019），以及用于多层数据的专用软件包 **mitml** ，其相关研究见Grund 等人（2016）。
除了 R 之外，Python（**scikit-learn** 的 IterativeImputer、**miceforest**）、SAS（**PROC MI**）、Stata（**ice** 命令）以及其他各种平台上也有相关实现。He、Zhang 和 Hsu(2021)；Kombo、Mwambi 和 Molenberghs (2017)；Pampaka 等 (2016)；van Buuren 和 Groothuis-Oudshoorn(2011)。鉴于 MICE 的计算需求，已有研究致力于通过并行化缩短计算时间。

人们尝试通过并行化来缩短计算时间。由于插补过程本身具有迭代性质，无法对其进行并行化；不过，各次插补彼此独立，因此可以并行处理，这正是 parlMICE 软件包采用的方法。然而，当使用大数据时，这种方法仍无法解决根本性的内存挑战。

目前尚未开发出能够在计算资源有限的情况下处理大型数据集（尤其是大型医疗记录）的 MICE 实现。

1.3. bigMICE 解决方案

为弥补现有方法在大型数据集上计算多重插补方面的不足，我们开发了软件包 **bigMICE**。该软件包提供了分布式缺失数据插补框架，利用 Apache**Spark** 的功能，通过在 RAM 与硬盘之间动态交换信息，使计算能够适应给定的 RAM 空间Apache 软件基金会 (2025）。借助该软件包，即使在普通计算机上，只要具备常见的硬件配置（例如 16 GB RAM 或更少），也能处理包含数百万个观测值的大型数据集。

为处理大型数据集，bigMICE 软件包将 MICE 框架与 Apache Spark 和 MLLib 的功能相结合。现有方法无法在 Spark 的分布式计算环境中原生地对大型数据集执行多重插补。该软件包采用了几项关键设计原则：使用 Spark 数据框进行分布式数据处理，利用 Spark MLLib 优化实现的可扩展回归模型，采用内存高效存储策略，以及适配 Spark 有向无环图（DAG）执行模型的迭代处理。

1.4. 贡献与范围

本文介绍了一种专为大数据集设计的 MICE 框架软件实现。我们的软件包 bigMICE 借助 Apache Spark 环境，对超出传统实现内存容量的数据集进行多重插补，从而填补了统计软件生态系统中的一项重要空白。

本研究的主要贡献包括：

1. **全新的软件架构**：在 Spark 分布式计算框架中重新实现 MICE 算法，在保持统计有效性的同时实现可扩展性。2. **内存管理解决方案**：针对大数据场景下多重插补的内存需求，提出创新方法，包括高效的存储策略和基于参数的合并方法。3. **与现有生态系统集成**：通过 R 的 Spark 接口（sparklyr 软件包），与 R 统计计算环境无缝集成。4. **全面的评估框架**：系统评估不同数据集规模和复杂度下的计算性能、内存效率和统计有效性。

5. **开源实现与发布**：提供一款文档齐全的开源软件包，可针对各种大数据插补场景进行扩展和调整。

1.5. 论文结构

本文其余部分安排如下。第 2 节介绍链式方程多重插补方法和 Spark 分布式计算框架的背景信息。第 3 节说明如何安装和使用该软件包。第 4 节介绍软件架构，并详细说明 bigMICE 软件包的实现。第 5 节展示计算实验的结果；这些实验使用真实的医疗登记数据集，全面评估该实现的性能、可扩展性和统计性质。第 6 节讨论实验结果，并提出该框架未来开发和扩展的方向。

软件实现、文档和可复现示例可通过github.com/bigcausallab/bigMICE获取。

2. 背景

2.1. 链式方程多重插补：理论基础 n

数据缺失机制

要理解缺失数据的性质，关键在于考察导致缺失机制的潜在机制。根据变量缺失这一事实是否与数据集中变量的潜在取值相关，缺失值可分为三种不同类型Little 和 Rubin (2019)；White、Royston 和 Wood(2011)：

- **完全随机缺失（MCAR）**：数据缺失的概率不取决于观测数据或未观测数据。完全随机缺失意味着缺失数据的概率独立于观测变量和未观测变量 Hasan等 (2021)；Wang 等 (2020)；Pampaka 等 (2016)；Grund 等 (2016)；Khan 和 Hoque(2020)；Austin 等 (2020)

- **随机缺失（MAR）**：在给定观测数据的条件下，数据缺失的概率不取决于未观测数据。如果在考虑所有观测变量后，某变量缺失的概率独立于未观测数据，则认为数据是随机缺失的 Hasan 等 (2021)；Wang 等(2020)；Grund等 (2016)；Khan 和 Hoque (2020)；Austin 等 (2020)；van Buuren 和 Groothuis- Oudshoorn (2011）。换言之，可以根据数据集中其他观测变量预测缺失机制 Wang等 (2020)。

- **非随机缺失**：即使在以观测数据为条件之后，缺失数据的概率仍取决于缺失变量本身的取值Hasan 等(2021)；Wang 等(2020)；Grund 等(2016)；Khan和Hoque(2020)；Austin 等(2020)；van Buuren和Groothuis-Oudshoorn(2011). .....的分析

> 非随机缺失条件下的数据尤其具有挑战性，因为它们需要对缺失数据机制本身进行建模，而该机制往往是未知的。 Wang等(2020).

仅凭观测数据无法区分随机缺失和非随机缺失，尽管通过收集其他变量并将其纳入分析，可以使随机缺失假设更具合理性White 等 (2011)。

多重插补框架

多重插补（MI）是一种用于分析不完整数据集的通用统计方法 van Buuren (2007)；Pampaka 等 (2016)；Kombo 等 (2017)；Austin 等(2020)。MI 提供了一种策略，可针对缺失数据条目中的统计估计进行有效推断 van Buuren (2007)。MI 的主要原则由 Rubin 提出 Rubin(2004)，使其成为处理缺失数据的主要方法。MI 的核心思想是利用观测数据的分布，为缺失数据估计一组合理值，同时纳入随机成分以反映其不确定性，并创建多个（m）完整数据集。随后分别对这些数据集进行分析，并使用 Rubin 法则合并结果，以获得总体估计和方差-协方差矩阵White等 (2011)。

使用 MI 进行统计分析通常包含三个主要步骤van Buuren (2007)；Pam-paka等(2016)：

- **插补**：第一步是指定插补模型，并为数据中的缺失值生成称为插补的合理合成数据值 vanBuuren (2007)；Pampaka 等(2016)；van Buuren 和 Groothuis-Oudshoorn (2011）。这一过程会生成若干 m 完整数据集，其中缺失数据由从合理值分布中随机抽取的值替代 vanBuuren (2007)；Pampaka等(2016）。插补次数 m 通常为 3 到 10 次 vanBuuren(2007)，不过某些分析可能需要更多次数Grund 等(2016)。

- **分析**：第二步是对每个 m 插补数据集进行分析，采用统计方法估计科学研究关注的量 van Buuren(2007)；Pampaka 等(2016)，例如回归模型的系数。由此得到 m 多次分析结果，它们之间的差异仅源于插补结果的变化 vanBuuren(2007）。MI 使分析者能够使用为完整数据设计的方法Pampaka等(2016)。

- **合并**：第三步是将 m各项估计合并为一个单一估计值vanBuuren (2007)。这一过程结合了 m插补数据集van Buuren (2007)内部和之间的变异；Kombo等人(2017)。在相当宽松的条件下，这一步会得出统计上有效的估计值，并通过置信区间的宽度反映缺失数据所造成的不确定性van Buuren (2007)；Pampaka 等人(2016)。

合并步骤遵循 Rubin 法则，该法则允许将 m 插补中的估计参数合并为总体估计和方差-协方差矩阵，其中同时纳入插补内变异（对单个数据集结果的不确定性）和插补间变异（反映缺失信息所导致的不确定性）White等人(2011)。

链式方程多重插补

在多重插补框架下，针对多变量数据插补，主要形成了两种方法：联合建模（JM）和链式方程多重插补（MICE）van Buuren (2007)；van Buuren 和 Groothuis-Oudshoorn(2011)。联合建模通过为缺失数据van Buuren 和 Groothuis-Oudshoorn (2011)指定多变量分布，而 MICE（也称为完全条件设定（FCS）或序贯回归多变量插补）van Buuren (2007)；van Buuren 和 Groothuis-Oudshoorn(2011)；Yang 等人(2020)；Austin 等人(2020)，则采用了不同的方法。MICE 算法旨在通过使用一组条件模型（每个含有缺失数据的变量对应一个模型），克服单变量插补的局限性。

首先介绍一些符号。令 Y_j 为具有 p 个变量的数据集 Y 的第 j^{th} 列（变量），Y_j^{mis} 为 Y_j 中含有缺失值的部分，Y_j^{obs} 为 Y_j 中包含观测值（非缺失值）的部分。记作 $Y^{obs} = \{Y_1^{obs}, \ldots, Y_p^{obs}\}$ 和 $Y^{mis} = \{Y_j^{mis}, \ldots Y_p^{mis}\}$。MICE 算法通过对每个变量的缺失值进行多次插补（多次迭代）来运行。将第 t 次迭代中插补的变量记为 Y_j^t，并记为 $Y_{-j}^t = (Y_1^t, \ldots, Y_{j-1}^t, Y_{j+1}^{t-1}, \ldots, Y_p^{t-1})$。

最初，缺失值通过对观测值进行有放回简单随机抽样来填补。随着 MICE 程序的推进，这些初始随机抽样值会被迭代替换。在第一次迭代中，第一个含有缺失值的变量 Y_1 使用观测到 Y_1 的个案进行回归，回归自变量为所有其他变量 $Y_2, \ldots, Y_k$；随后，从 Y_1 的后验预测分布中模拟抽样，以替换缺失值。对于每个含有缺失数据的变量，该过程依次继续进行，例如，变量 Y_2 使用插补后的 Y_1 对 $Y_1, Y_3, \ldots, Y_k$ 进行回归，依此类推。重复该过程若干次，理想情况下直到收敛，从而生成一个完整的插补数据集；然后将整个程序重复 m 次，以生成 m 个多重插补数据集。

更一般地，对于每个不完整变量，都需要指定一个条件模型：

$$P(Y_j \mid Y_j^{obs}, Y_{-j}, \theta_j),$$

其中 Y_j 是要进行插补的变量，Y_{-j} 表示数据集中其余的变量，θ_j 是条件模型的参数。van Buuren（2007）。
从初始猜测值开始，MICE 插补通过迭代所有指定的条件模型来进行van Buuren （2007）。如果这些条件分布所隐含的联合分布存在，该算法便作为吉布斯采样器运行van Buuren 和 Groothuis- Oudshoorn（2011）；van Buuren （2007）。

MICE 在构建多变量模型方面具有很大的灵活性，而且通常比联合建模（JM）更易于实现，尤其是在存在复杂数据结构时，例如跳答模式、限制条件和边界；这些情况在大规模调查数据中很常见 van Buuren（2007）；van Buuren 和 Groothuis-Oudshoorn（2011）；He 等人（2021）。它支持采用变量特异性插补方法，例如对连续数据使用线性回归、对二元数据使用逻辑回归，或对有序数据使用比例优势回归 van Buuren（2007）；Resche-Rigon 和 White（2018）；Kombo等人（2017）；He 等人（2021），并支持纳入约束条件，以确保插补值在逻辑上保持一致 van Buuren（2007）；van Buuren、Brand、Groothuis-Oudshoorn 和 Rubin（2006）。算法1介绍了使用 MICE 方法生成一次插补的过程。

MICE 中的“链式方程”是指用于对每个变量中的缺失值进行迭代插补的回归模型。使用多个回归模型可以

算法1： 链式方程多重插补

输入： 数据集 Y，每个不完整变量的后验模型和后验预测模型：
变量 Y_j： $P(\theta_j|Y_j^{obs}, Y_{-j})$ 和 $P(Y_j|Y_{-j}, \theta_j)$
输出： 第 T 次迭代时的插补数据集 $Y^T = (Y^{obs}, Y^{mis,T})$

初始化循环：
对于 j 从 1 到 p **执行**

从 Y_j^{obs} 的经验分布中抽取初始插补 $Y^{mis,j}{}^0$

结束循环
插补循环：
对于 t 从 1 到 T **执行**

对于 j 从 1 到 p**执行**

$\theta_j^t =$ 从后验分布 $P(\theta_j|Y_j^{obs}, Y_{-j}^t)$ 中抽取
$Y^{mis,j}{}^t =$ 从后验预测分布 $P(Y_j^{mis}|Y_{-j}^t, \theta_j^t)$ 中抽取

结束循环

结束循环
返回 Y^T

图 1：链式方程多重插补（MICE）

用于处理多种变量类型。条件模型的常见设定包括针对二元变量的逻辑回归、针对连续变量的线性回归或随机森林回归，以及针对分类变量的多项逻辑回归或随机森林分类。

2.2. Apache Spark与分布式计算解决方案

海量数据的生成催生了专为管理和分析这些信息而设计的大数据平台。大数据平台是全面的框架，借助分布式计算、并行处理和高级分析，使组织能够存储、处理和分析结构化与非结构化数据。大数据平台提供多种功能，包括强大的存储解决方案和分布式处理能力，以便在多个节点上处理大量数据。它们确保容错能力，从而保证数据持续可用、处理不中断，即使发生故障也不例外。

Apache Spark 框架

Spark Apache 软件基金会(2025) 是一个功能强大的开源平台，专为大规模数据处理而设计，尤其适合迭代式统计和机器学习任务Xiangrui Meng、Yavuz、Sparks、Venkataraman、Liu、Freeman、Tsai、Amde、Owen、

Xin、Xin、Franklin、Zadeh、Zaharia 和 Talwalkar(2016); Zaharia、Xin、Wendell、Das、Armbrust、Dave、Meng、Rosen、Venkataraman、Franklin、Ghodsi、Gonzalez、Shenker 和 Stoica(2016). **Spark** 改进了传统的 MapReduce 范式 Dean 和 Ghemawat (2008)，通过提供更好的数据管理，从而显著提升了性能Salloum、Dautov、Chen、Peng 和 Huang (2016)。其核心在于，**Spark**将弹性分布式数据集（RDD）用作计算之间高效数据共享的基本抽象Zaharia 等人(2016);Salloum 等人 (2016)。借助 **Spark**，数据可通过由算子构成的更通用的有向无环图（DAG），利用丰富的转换和动作集合进行处理。它会自动将数据分布到整个集群，并将所需操作并行化。它支持多种转换，使数据预处理更加容易，尤其是在检视大数据集变得愈发困难时Salloum 等人(2016)。

在 Spark 中，缓存、持久化和检查点保存等机制支持高效的数据复用和容错。缓存用于将数据临时存储在内存中，尤其适用于同一数据集在多项操作中被反复访问的情况，并能显著加快处理速度。持久化的作用类似，但灵活性更高，用户可以根据资源约束，决定将数据存储在内存、磁盘上，或同时存储在两者中。另一方面，检查点保存是一种更稳健的流程，专为长而复杂的计算链设计。它会将数据保存到可靠的存储系统（如 Hadoop 分布式文件系统 (HDFS)），从而有助于防止数据丢失，并通过截断转换的血缘关系来减少计算开销。

检查点是 Spark 的一项重要优势，使计算机在 RAM 容量较小的情况下也能对大型数据集执行大规模运算。bigMICE 广泛使用检查点来避免内存问题，例如堆栈溢出错误。

用于机器学习的 *Spark MLLib*

Spark 的一个重要组成部分是**Spark** 是 **MLLib**，这是一个开源分布式机器学习库。**MLLib** 为各种学习场景提供高效功能，并包含多种底层统计、优化和线性代数原语Xiangrui Meng 等人(2016年）。使用**MLLib** 的一项主要优势是，它包含许多回归和分类模型，以及经过优化、能够在**Spark** 平台上高效运行的算法，包括线性回归、逻辑回归、决策树和随机森林；这些都是 MICE 框架中常用的插补模型。

许多机器学习算法都具有迭代特性，例如逻辑回归会采用梯度下降等过程来优化目标函数，因此**Spark**能够对分布式数据集执行内存计算，这对提升效率至关重要Zaharia等人 (2016年）；Zaharia、Chowdhury、Das、Dave、Ma、McCauley、Franklin、Shenker 和 Stoica(2012年）。由于**Spark** 可以在这些迭代过程中将数据缓存在 RAM 中，因此无需在每一步都从磁盘重新加载数据，从而与 Hadoop的MapReduce 等系统相比显著提升速度，尤其是在处理迭代算法时Zaharia 等人(2016，2012年）。**Spark**的**MLLib**库提供了这些算法的优化实现，进一步简化了它们在大数据机器学习流水线中的应用。

*sparklyr*接口

该 pac kage sparklyr 是 Apache Spark 的 R接口，使数据科学家和分析师能够 s

直接在 R 环境中利用 Spark 的分布式计算能力。它提供了连接 Spark 集群、使用 dplyr 风格的语法处理大规模数据集、通过 Spark MLLib 执行机器学习，以及运行自定义 R 或 SQL 代码的工具。sparklyr 与 R 和 Spark 无缝集成，支持可扩展的数据处理与建模，非常适合涉及大数据分析的研究和应用，而且研究人员无需离开熟悉的 R 环境。

2.3. bigMICE 的方法学调整

bigMICE 的实现对传统的 MICE 工作流程进行了几项关键改进：

- **初始化方法**：为初始化缺失值开发了两种方法：使用汇总统计量（均值、众数、中位数），以及从观测值中随机抽样；两种方法均通过 Spark 操作实现，以支持可扩展计算。

- **分类与回归模型集成**：bigMICE 软件包采用专门函数，利用 Spark MLLib 中提供的各种回归模型，包括用于连续变量的线性回归、用于二元变量的逻辑回归、用于分类变量的多项逻辑回归，以及用于回归和分类任务的随机森林方法。

- **Rubin 法则的实现**：为了控制内存需求，该软件包应用 Rubin 法则，仅存储用于合并结果的模型参数，而不在整个过程中将完整的插补数据集保留在内存中。

- **检查点**：为防止偶发的内存瓶颈导致执行失败，我们采用 Spark 的检查点机制。

- **运行环境的自定义**：通过配置 Spark 连接，bigMICE 用户可以指定分配给计算任务的资源（例如 CPU 核心数和内存限制）。这使软件包能够灵活适用于不同的执行环境，从分布式集群到小型笔记本电脑皆可使用；只要硬盘空间足够大，无论可用 RAM 或 CPU 如何，都能使用 bigMICE 软件包。

3. 设置与指南

使用 bigMICE 前，必须先安装其他几个软件。以下部分将详细介绍所需步骤，并展示一个简单的软件包使用示例。

3.1. 设置

sparklyr 软件包

bigMICE 软件包与 sparklyr 框架集成，使用户能够直接从 R 管理 Spark 集群或配备多个核心的独立机器。该软件包已

使用 Spark 4.0.0 版本和 sparklyr 1.9.1 版本进行测试，不过为了与早期版本的 sparklyr 兼容，可能需要使用更早版本的 Spark（$<$ 4.0.0）。
首先安装 sparklyr，即可直接从 R 安装 Spark，方法如下：

```
R> install.packages("sparklyr") # version 1.9.1
R> options(timeout = 6000) # for slow internet connections
R> library("sparklyr")
R> spark_install(version="4.0.0") # compatible with sparklyr 1.9.1
```

安装好 sparklyr 和 Spark 后，用户可以连接到 Spark，同时指定所需使用的资源：

```
R> conf <- spark_config()
R> conf$`sparklyr.shell.driver-memory` <- "10G" # How much memory
R> conf$`sparklyr.cores.local` <- 4 # How many CPU cores
R> sc = spark_connect(master = "local", config = conf)
```

Hadoop 分布式文件系统

除了 Spark，还建议安装 Hadoop，以便使用 Hadoop 分布式文件系统（HDFS）。集成 HDFS 后即可使用检查点功能，从而提高计算的稳定性和容错能力。如果无法使用 HDFS，用户仍可通过使用参数 `checkpointing = FALSE`禁用检查点功能来使用该软件包。

给定一个 HDFS 目录后，连接到 Spark 会话即可按如下方式设置检查点目录：

```
R> spark_set_checkpoint_dir(sc, "/path/to/your/HDFSdirectory/" )
```

bigMICE 的开发版本（*GitHub*）

bigMICE软件包托管在 GitHub 上，可通过 devtools 软件包安装：

```
R> install.packages("devtools")
R> library("devtools")
R> devtools::install_github("bigcausallab/bigMICE")
```

3.2. 使用示例

以下示例展示了在安装了 Spark 的单台机器上运行 bigMICE 软件包时的典型工作流程。流程包括设置环境、加载数据、定义变量类型和分析模型，以及执行多重插补。

加载所需的库：在此示例中，我们加载 bigMICE、sparklyr 和 dplyr，以便进行数据操作。

```
R> library("bigMICE")
R> library("sparklyr")
R> library("dplyr")
```

初始化本地 Spark 会话：我们为本地会话指定 Spark 配置。我们将 Spark 内存限制设为 10 Gb，并配置 4 个 CPU 核心：

```
R> conf <- spark_config()
R> conf$`sparklyr.shell.driver-memory` <- "10G"
R> conf$spark.memory.fraction <- 0.8
R> conf$`sparklyr.cores.local` <- 4
R> sc <- spark_connect(master = "local", config = conf)
```

加载并准备数据集：本例使用 **mice** 软件包中的“boys”数据集，可从以下位置下载 https://github.com/amices/mice/blob/master/data/boys.rda。我们首先将此数据集转换为 `csv` 格式，并将其保存到工作目录中。

```
R> data <- load("boys.rda")
R> write.csv(boys, "data.csv", row.names = FALSE)
```

假设数据集现已保存到工作目录中，文件名为 `data.csv`，接下来将其加载到 Spark 中，并从分析中排除某些变量。

```
R> sdf <- spark_read_csv(sc, "data", "data.csv",+     header = TRUE,
 infer_schema = TRUE,  null_value = "NA") %>%
+     select(-all_of(c("hgt", "wgt", "bmi", "hc"))) # removed variables
```

定义变量类型和分析公式： 必须明确指定变量类型，并为下游分析定义公式。

```
R> variable_types <- c(+     age = "Continuous_float",
+     gen = "Nominal",+     phb = "Nominal",
+     tv  = "Continuous_int",+     reg = "Nominal"+ )
R> analysis_formula <- as.formula("phb ~ age + gen + tv + reg")
```

执行多重插补：使用 `mice.spark` 进行插补，每次插补生成三个插补数据集（`m = 3`），并进行 2 次迭代（`maxit = 2`）。此处已禁用检查点：

```
R> imputation_results <- bigMICE::mice.spark(
+     data = sdf,+     sc = sc,
+     variable_types = variable_types,
+     analysis_formula = analysis_formula,
+     m = 3,+     maxit = 2,
+     checkpointing = FALSE+ )
```

提取 **结果** 以下元素由 mice.spark函数返回 .
分析的合并结果:

```
R> print(imputation_results)

Multiple Imputation Results
===========================

Number of imputations: 3
Total imputation time: 299.94 seconds
Average time per imputation: 99.98 seconds

Pooled Parameter Estimates (Rubin's Rules)
==========================================
            Estimate Within_Var Between_Var Total_Var      SE t_stat
(Intercept)        0    97.2486   2625.7130 3598.1993 59.9850      0
age                0     0.3800     10.2599   14.0599  3.7497      0
tv                 0     0.0003      0.0073    0.0100  0.1001      0

Diagnostic Information:
-----------------------
   (Intercept)    : r=36.000, lambda=0.973, df=2.1
age            : r=36.000, lambda=0.973, df=2.1
tv             : r=36.000, lambda=0.973, df=2.1

Notes:
------
- SE: Standard Error (sqrt of Total_Var)
- t_stat: t-statistic for testing parameter = 0
- r: Relative increase in variance due to nonresponse
- lambda: Fraction of missing information
- df: Degrees of freedom for t-distribution
- Use show_individual=TRUE to see results from each imputation
```

为每个插补数据集估算的模型参数:

```
imputation_results$model_params
```

以及插补过程的一些统计信息，例如插补时间：

```
imputation_results$imputation_stats
```

此外，也可以使用 `mice.spark.plus` 函数而非 `mice.spark` 函数返回完整的插补数据集（内存开销更大），随后：

```
imputation_results$imputations
```

4. 软件包描述

尽管已有多种 MICE 实现，但仍缺乏能够高效处理 Spark 数据框所管理数据的规模及其分布式特性的专用工具。为满足这一需求，bigMICE软件包应运而生并得以开发。该软件包提供了一个分布式缺失数据插补框架，充分利用 Spark 的能力。为使 bigMICE 能够与 Spark 数据框及 Spark MLLib 库无缝集成，对标准 MICE 框架进行了多项修改。这些修改将在本节中加以说明。

4.1. 初始化方法

原始 MICE 框架支持多种方法来初始插补缺失值，包括对观测值进行随机抽样，以及使用均值、众数或中位数等汇总统计量来插补缺失值。因此，bigMICE 开发了两种方法，分别使用汇总统计量或对观测值进行随机抽样来初始插补缺失值。

`impute_with_MeMoMe` 函数可用于使用汇总统计量初始插补缺失值。此函数接受一个 Spark 连接、一个 Spark 数据框，以及一个命名向量；该向量为每一列指定插补方法（均值、众数、中位数或无）。用户提供的变量类型决定了每个变量适用的初始插补方法。`mice.spark` 函数中的预定义字典会将不同的变量类型映射到相应的初始插补方法。例如，二元、名义型和有序变量使用众数进行初始化，整数和计数变量使用中位数，而浮点变量则使用均值进行初始化。类型指定为代码、日期/时间和字符串的变量会设为“无”，表示不对这些变量进行初始插补，因为无法为这些变量类型计算汇总统计量。随后，系统会利用此映射创建初始化方法向量，用于指定输入 Spark 数据框中每一列的初始插补方法。接着，`impute_with_MeMoMe` 函数会根据每列指定的方法，计算其观测值的相应均值、众数或中位数。然后，使用这些计算结果替换 Spark 数据框相应列中的缺失值，从而得到经过初始插补的 DataFrame。

`impute_with_random_samples` 函数可用于通过有放回随机抽样来初始化缺失值。该函数以 f（即 Spark 连接）和 Spark 数据框作为参数。该函数利用 sparklyr 的 `sdf_sample` 函数对观测值执行有放回抽样。我们采取一些额外步骤来

规避该函数的缺陷。 `sdf_sample` 并不接受精确的所需样本数量作为输入，而是接受输入向量的一个比例，因此有时精度不足，返回的样本数少于指定比例所对应的数量。因此，要使用该函数，所需比例通过缺失值数量除以观测值数量来计算。然后调用该函数，从观测值中进行有放回抽样，以获得用于替换缺失值的所需数量。如果抽到的数值数量正确，则用它们替换数据中的缺失值。否则，将重新调用抽样函数，直到抽到正确数量为止。尽管可能需要多次抽样尝试才能获得所需的样本数量，但在实践中，它仍然比使用其他（更精确但）未针对 Spark 数据优化、因而慢得多的抽样函数更高效。

这一初始插补步骤既可以由 `mice.spark`函数本身执行，也可以由用户作为输入提供（使用 `imp_init` 可选参数），从而提供更大的灵活性。默认情况下， `mice.spark` 使用 `impute_with_random_samples` 来初始化缺失值。这一已初始化的 DataFrame 将作为 MICE 算法中后续迭代插补过程的起点。

4.2. 插补方法

所开发的软件包采用一套专用函数，使用 Spark MLLib 中提供的各种回归与分类模型来执行各个插补步骤。选择基于可通过 sparklyr接口访问的 MLLib 现有模型进行构建，在大数据处理场景中尤为有利，因为其底层框架专门设计为通过将执行分布到多个核心/机器上，并在 RAM 与硬盘之间无缝交换信息，以将执行控制在给定的内存限制之内，从而高效处理大型数据集。这意味着 MLLib 提供的模型能够以可扩展且计算高效的方式处理大型数据集，而使用传统的单机插补方法则难以甚至无法做到。借助这些经过优化的实现，bigMICE 可以对海量数据集执行插补任务，从而能够分析以往对于标准统计软件而言规模过大、或需要专门计算设施的数据集。

所有这些插补函数都遵循一致的工作流程：

- 在输入 Spark 数据框中添加一个临时的顺序 ID，以保留原始行顺序。保留数据的顺序十分重要，因为缺失机制的模式会影响插补质量。

- 将 DataFrame 分为两个子集：一个包含目标变量的完整观测值，另一个包含缺失值。观测数据将用于拟合回归或分类模型，然后使用该模型预测缺失值。

- 生成模型公式，指定由特征变量预测目标变量。这是 sparklyr 中回归模型的一个重要参数。

- 使用目标变量中具有观测值的个案来训练回归模型。

- 然后使用训练好的模型预测目标变量中的缺失值。
- 目标变量中的缺失观测值由根据边际分布生成的数据替代。
- 最后，将完整子集与插补子集合并，根据临时 ID 重新排序，并移除 ID 列，然后返回生成的 DataFrame。

bigMICE 中实现的**bigMICE** 插补函数见表1。

表1：bigMIC 中的插补函数 **E**

函数名称	**描述**	**底层 sparklyr 函数**	**目标变量 Type**
`impute_with_linear_regression`	线性回归	`ml_linear_regression`	连续型
`impute_with_logistic_regression`	逻辑回归	`ml_logistic_regression`	二元分类变量
`impute_with_mult_logistic_regression`	多项逻辑回归	`ml_logistic_regression`	分类变量（>2 个水平）
`impute_with_random_forest_regressor`	随机森林回归器	`ml_random_forest_regressor`	连续型
`impute_with_random_forest_classifier`	随机森林分类器	`ml_random_forest_classifier`	分类变量

4.3. 从边际分布生成

连续变量

首先根据预测值与观测值之间的模型残差计算均方根误差（RMSE）：

$$\hat{\sigma}_{res} = \sqrt{\frac{1}{n_{obs}} \sum_{i=1}^{n_{obs}} (y_i^{obs} - \hat{y}_i^{obs})^2},$$

其中 n_{obs} 是观测值的数量，y_i^{obs}表示观测值，而$\hat{}y_i^{obs}$表示当前模型对应的预测均值。为生成与连续变量相对应的边际分布，从而向每个插补预测值添加随机噪声 $\epsilon_i \sim \mathcal{N}(0, \hat{}\sigma_{res}^2)$：$y_{imp,\ i} = \hat{}y_i + \epsilon_i$。这些值用于插补缺失值。此方法遵循 Efron (1981) 提出的参数自助法思想。对于随机森林等非贝叶斯模型，这可以视为从后验预测分布中抽样的一种适当替代方法。

分类变量

使用概率抽样方法插补缺失的分类变量，同时保留分类不确定性。使用完整个案训练适当的分类算法，以预测类别归属的概率 $P(Y = c_k|X)$ ，针对每个类别 c_k， $k = 1,\ldots,K$。我们没有确定性地指定概率最高的类别，而是按照逆累积分布函数法进行概率插补 Devroye(1986)。对于每个存在缺失值的观测 i ，模型会生成一个概率向量 $\mathbf{p}_i = [p_{i1}, p_{i2}, \ldots, p_{iK}]$ ，其中 $\sum_{k=1}^{K} p_{ik} = 1$。计算累积概率得到 $F_{ik} = \sum_{j=1}^{k} p_{ij}$，从而为每个观测构造累积分布函数。随后生成一个均匀随机变量 $U_i \sim \mathcal{U}(0,\ 1)$，并通过逆变换抽样确定插补类别： $Y_i = c_k$ ，如果 $F_{i,\ k-1} < U_i \leq F_{ik}$，根据惯例，$F_{i,\ 0} = 0$ 。

4.4. bigMICE采样器

`sampler.spark`函数是 bigMICE 中执行迭代式多重插补的核心组件，它整合了上述所有函数。该函数接收一个初始插补后的 Spark 数据框，并在指定的周期数内，针对每个变量迭代插补缺失值。对于每个变量，它会确定相关的预测变量（默认使用所有其他变量，也可使用用户提供的预测变量矩阵），并根据用户提供的变量类型选择适当的插补方法。随后，该函数会调用相应的插补函数（`impute_with_...`），在当前迭代中对该特定变量执行插补。此过程遵循 MICE 算法，在所有变量上重复执行，并贯穿设定的迭代次数，最终返回一个完全插补后的 Spark 数据框。

4.5. 主函数与下游分析

`mice.spark()`函数通过链式方程迭代插补缺失值，生成 m 个完整数据集；其中，每个存在缺失数据的变量都以所有其他变量为条件建立模型。对于每次插补 $i = 1,\ldots,m$，该函数使用 `init_with_random_samples()` 初始化缺失值，通过`sampler.spark()` 执行迭代插补（最多进行 `maxit` 次迭代），并拟合用户指定的分析模型ˆ，以获得参数估计值 θ_i。

为避免将每个插补数据集保存在内存中，该函数会在每次插补循环后直接执行给定的下游分析，并仅保存感兴趣的参数，以便后续合并。目前，下游分析仅限于使用

`sparklyr::ml_logistic_regression` 函数并结合用户提供的公式进行逻辑回归建模。这种高效利用内存的方法避免了保存完整插补数据集所需的大量存储空间，尤其适用于大型数据集和插补次数较多的情况。

根据 Rubin 的多重插补推断法则，该函数会合并 m 参数估计值，以计算总体估计。假设 ˆ Q_l 是第 l^{th} 次重复插补的估计值，则合并后的估计值等于 $\bar{Q} = \frac{1}{m}\sum_{l=1}^{m} \hat{Q}_l$。插补内方差定义为 $\bar{U} = \frac{1}{m}\sum_{l=1}^{m} \bar{U}_l$ ，其中， $\bar{U}_l$ 项是通过第 l^{th} 次插补得到的 Q 的方差-协方差矩阵。插补间方差 $B = \frac{1}{m-1}\sum_{l=1}^{m}(\hat{Q}_l - \bar{Q})(\hat{Q}_l - \bar{Q})^T$，总方差 $T = \bar{U} + B + B/m$。这种方法能够妥善考虑

每次插补内部的不确定性，以及插补过程本身引入的额外不确定性，从而在存在缺失数据时实现有效的统计推断，同时利用 **Spark** 的分布式计算能力，扩展至大型数据集。该函数会返回一个全面的列表，其中包含 Rubin统计量 vanBuuren (2018)、每次插补的统计量，以及保存的模型参数。

4.6. 访问插补数据

在大数据背景下，处理和存储多个插补数据集会带来显著的内存挑战。使用 `mice.spark` 函数时，首选的工作流程是仅返回最终分析结果，从而避免在内存中管理大型插补数据集所产生的开销。不过，在某些情况下，访问插补数据本身可能也很有用，例如评估插补质量或开展自定义的下游分析。

为支持此类用例，该软件包提供了 `mice.spark.plus` 函数，可用于获取插补数据集。插补数据集以 Spark DataFrame 的形式存储在内存中，并作为函数输出的一部分返回，使用户能够直接检查和分析插补结果。对于大型数据集，这种方法可能会占用大量内存，从而限制可扩展性。

4.7. 其他参数

该 `mice.spark`函数提供了多个可选参数，可用于更好地控制插补过程和内存使用。附录A中完整介绍了所有参数，包括其用途和用法。

5. 实验

5.1. 数据：瑞典国家糖尿病登记处

国家糖尿病登记处（NDR）由瑞典糖尿病学会于 1996 年创立。NDR 的一项目标是，理想情况下，瑞典所有糖尿病患者都应每年上报一次；上报依据是患者实际前往初级卫生保健机构或内科医院门诊就诊时登记的年度数据。上报内容包括人口统计学数据、糖尿病病程、治疗方式，以及各种风险因素和糖尿病并发症。Gudbjörnsdottir、Cederholm、Nilsson、Eliasson，以及瑞典国家糖尿病登记处指导委员会 (2003)。NDR 可能是全球最大的国家级糖尿病登记库之一 Gudbjörnsdottir 等 (2003).该数据集涵盖了瑞典大多数 2 型糖尿病患者，并包含每位患者随时间记录的多个数据点 Gudbjörnsdottir等(2003).该数据集约有 1460 万行和 58 个不同类型的变量，每一行代表某一特定时间点的一条患者记录。该数据集中各变量的缺失机制比例从 0% 到 99.96% 不等，详见补充表8。

5.2. 运行时间和内存使用量与样本量的关系

为评估 bigMICE 的计算运行时间和内存使用量如何随数据集规模增大而变化，我们使用国家糖尿病登记处数据集开展了一系列实验。我们选取了 10 个变量作为子集，涵盖不同类型的变量（二元、整数、浮点数、有序和名义型），且这些变量的缺失值比例各不相同，范围为 0% 至 65.8%。使用线性回归插补整数和浮点变量，使用逻辑回归插补二元变量，使用多项逻辑回归插补名义变量，并使用随机森林分类器方法插补有序变量。对于每个变量的条件模型，均使用其余所有变量作为预测变量，并采用相应 MLLib 函数的默认参数。

通过对原始数据集进行无放回随机抽样，生成了一系列数据集，其大小按对数递增，从 1000 行逐步增加至包含 1460 万行的完整数据集。为运行实验，我们使用了一台配备 512 Gb RAM 和 32 核处理器（共 64 个线程）的计算服务器。对于 bigMICE，通过 Spark 会话参数将执行限制为 8 个线程和 16 Gb RAM；而作为基线实现的 mice 则可使用全部线程和全部内存。对于每种样本量，均使用 bigMICE 实现和 mice 实现运行 MICE 插补程序，以进行基线比较。

插补次数设为 m=1，每次插补执行 5 次迭代（默认设置），并针对每种样本量重复该过程五次，以考虑系统性能和运行时间稳定性的变异。除有序变量外，**bigMICE** 方法与基线使用相同的插补模型和参数；对于有序变量，我们在**bigMICE**中使用随机森林，而在**mice**中使用随机森林或多项回归，原因将在本节后文解释。有关用于插补的变量类型和模型的详细信息，请参见表2。

基线和 **bigMICE** 方法的内存消耗采用不同的测量方式，因为它们的计算过程有所不同。对于基线，在执行产生插补的 R 进程期间，测量运行代码的进程的内存使用量 `mice()` 函数。对于 **bigMICE** 插补过程，由于计算由 Java 虚拟机（JVM）执行，且 **Spark** 会话运行在其中，因此无法通过监控 R 进程来捕获内存消耗。我们改为在 **Spark**连接中启用指标收集，并在过程执行后从 **Spark** UI 中获取指标。具体而言，我们使用 JVM 堆使用量峰值（`peakMemoryMetrics.JVMHeapMemory`）来监控过程执行期间的最大内存消耗。用于监控内存的代码的详细实现见附录C.1。

运行时间测量仅统计插补过程的计算时间。所有其他计算（例如加载库、将数据加载到 R 或 Spark 中，或启动 Spark 连接）均不计入运行时间测量。

本实验的结果展示于图2 的内存图和图3 的运行时间图中。此外，相同结果也列于表3中。内存图中的红线表示 2025 年典型笔记本电脑的最大可用内存（16Gb）。

基线测量采用略有不同的配置进行了两次。其中一种配置使用随机森林对有序值进行插补，但运行时间迅速增长，在可能耗费过多

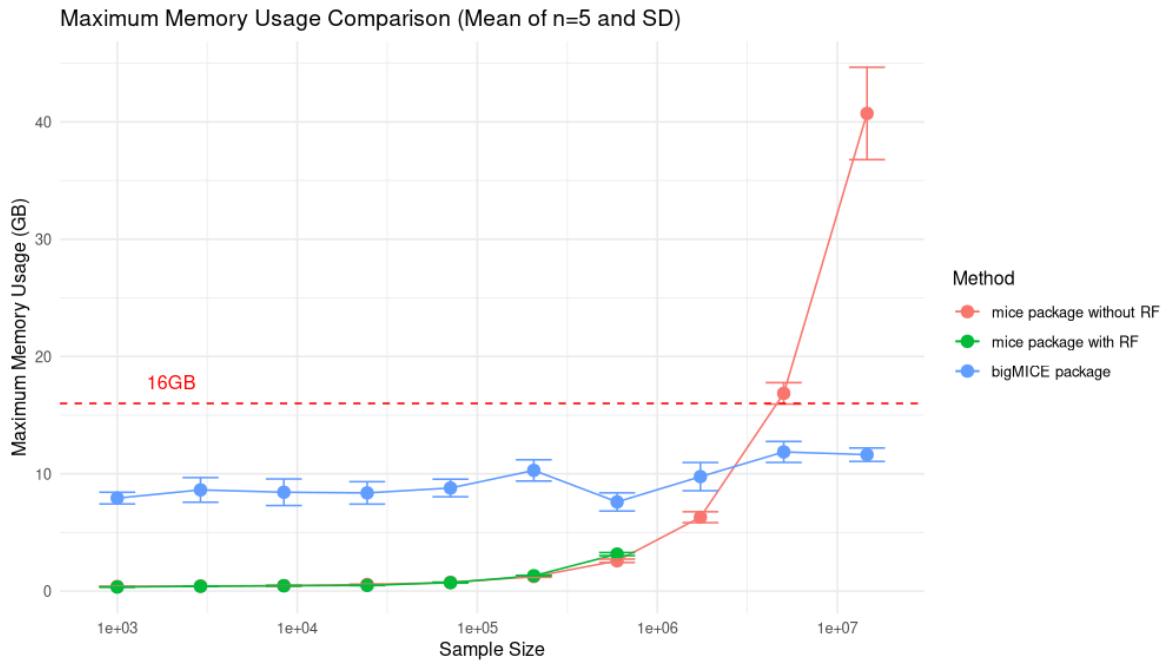

图 2：基准软件包 mice 与软件包 bigMIC 的内存使用量对比 **E**

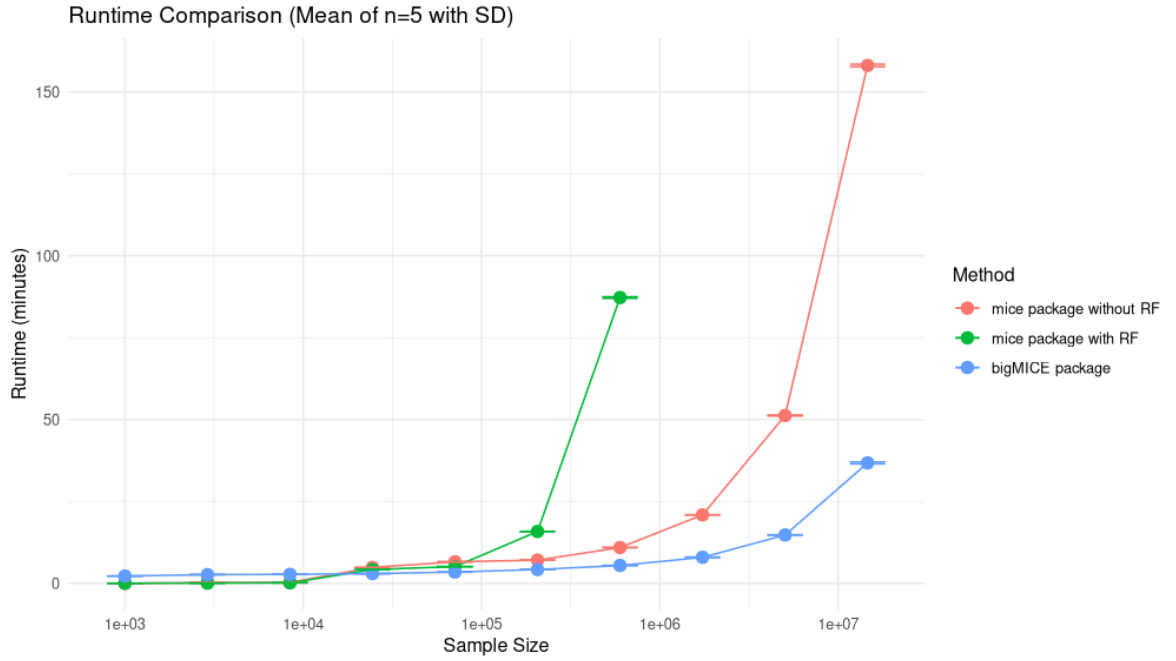

图3：基准软件包 mice 与软件包 bigMICE 的运行时间对比

变量名称	数据类型	基线模型 (function)	bigMICE 模型 (function)
alder	整数	贝叶斯线性回归-回归（norm）	线性回归（ml_linear_regression）
bmi	浮点数	贝叶斯线性回归-归（norm）	线性回归（ml_linear_regression）
胆固醇	浮点数	贝叶斯线性回归 (norm)	线性回归 (ml_linear_regression)
sex	名义型	多项逻辑回归归（polyreg）	多项 Logistic 回归（ml_logistic_regression）
临床_糖尿病_类型	名义型	多项逻辑回归回归（polyreg）	多项 Logistic 回归 (ml_logistic_regression)
糖化血红蛋白	整数	贝叶斯线性回归-归 (norm)	线性回归 (ml_logistic_regression)
中风	二元	逻辑回归 (logreg)	逻辑回归 (ml_logistic_regression)
体力_活动	有序	随机森林（rf）或多项回归（polr）	随机森林 (ml_random_forest)
rokare	名义型	多项逻辑回归（polyreg）	多项式 Logistic 逻辑回归回归 (ml_logistic_regression)
低血糖	有序	随机森林（rf）或多项回归 (polr)	随机森林 (ml_random_forest)

表 2：内存使用量和运行时间测量实验中使用的变量，以及基线方法和 Spark 方法所使用的相关模型。

内存，因此我们不得不停止实验。于是，我们运行了另一种基线配置，将随机森林模型替换为多项回归，从而证明计算所需的内存会随着样本量增加而迅速增长。

图2 显示，所开发的方法在所有样本量下均表现出恒定的内存使用量，且未超过分配的 16Gb 内存；在较大的样本量下，其运行时间优于基线方法。这表明 **bigMICE** 能够对非常大型的数据集中的缺失值进行插补，同时控制内存使用量。

5.3. 运行时间与纳入变量数量的关系

为评估纳入变量数量增加时 bigMICE 方法的运行时间如何变化，我们使用国家糖尿病登记处数据集开展了一系列实验。

随机　选取了一个包含一千行的子集。使用线性回归对　e

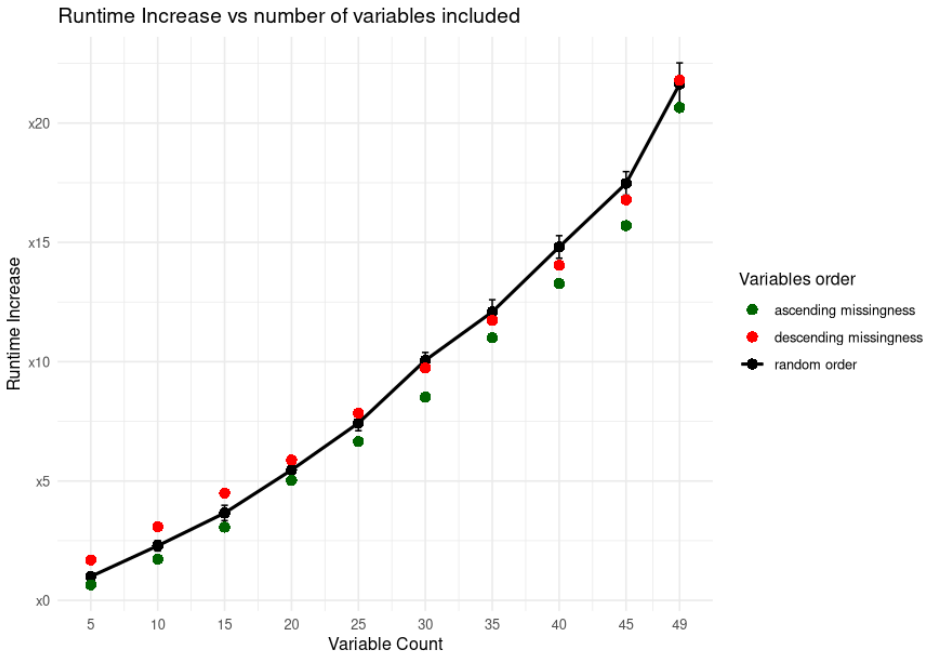

图4：不同变量数量子集的运行时间比较。

整数和浮点数变量进行插补；使用逻辑回归对二元变量进行插补；使用多项逻辑回归对名义变量进行插补；使用随机森林分类器方法对有序变量进行插补。我们进行了七次测量，变量数量从 5 到 49，每次递增 5，以评估运行时间的变化。为减轻变量缺失比例对运行时间的影响，每次测量时都以不同顺序添加变量。其中两次运行采用确定的顺序：按缺失比例递增和按缺失比例递减；其余五次则通过随机打乱变量顺序进行。

结果见图4 和表4。从图4 可以看出，运行时间的增长速度略高于线性。此外，待插补变量的缺失比例似乎对运行时间有轻微但明显的影响。这在变量较少时的测量结果中尤为明显：按缺失比例降序排列（优先插补缺失比例最高的变量）会使运行时间略高于平均水平。

5.4. 插补质量与缺失比例及样本量的关系

为评估 bigMICE 方法生成的插补结果的质量，我们设计了两项互补实验：（1）在保持缺失比例大致恒定的同时改变样本量；（2）在样本量恒定的情况下改变缺失比例。

从国家糖尿病登记处数据集中选取连续型变量 `GFR` 作为这些实验的目标变量。该变量原本的缺失值比例较低，因此可以对条目进行可控的人为遮蔽。通过将插补值与已知真实值进行比较，这种设置能够精确评估插补质量。

通过计算插补值与真实值之间的均方根误差（RMSE），量化变量 GFR 的插补质量。随后按以下方式生成各实验条件下的相关子集。在样本量变化实验中，生成了六种不同样本量的数据集，分别包含 1,000、5,000、9,503、90,303、858,129 和 8,154,612 行。这些样本量旨在涵盖广泛的实际应用场景，从小规模分析到超大型数据集。选择中间值是为了按大致对数递增的方式覆盖中等规模和较大规模；最大样本量则对应数据集中所有观测到的 GFR 行。所有样本量下的缺失值比例均保持为 ∼50%。在缺失比例变化实验中，固定使用 100 万行的样本量。人为移除条目的比例设为 1%、10%、30%、50%、70%、90%、99% 和 99.9%，以涵盖从极低到极端水平的各种缺失情形。由于数据集规模很大，即使移除相当大比例的值，仍可进行有意义的评估，因此实验也纳入了极端缺失比例。

在两项实验中，均通过随机掩盖目标变量中必要数量的值来引入缺失值。这模拟了该变量完全随机缺失（MCAR）的情形。对于每种条件，生成了 $m = 5$ 次插补，每次插补执行5次迭代。对每次插补计算均方根误差。

我们使用随机森林对 GFR 变量进行插补。所选变量所用的其余插补模型见补充表5。

样本量不同的实验结果见补充表6。缺失比例不同的实验结果见补充表7。补充表6表明，插补误差随样本量增加总体呈下降趋势，这是预期的模型行为。补充表7 表明，对于这种包含100万行的大型数据集，插补准确度仅在缺失比例达到99.9%时才出现明显下降，而在所考虑的其他所有缺失比例下，插补准确度相当接近。

6. 讨论

6.1. 结果讨论

The **bigMICE** 方法在所有测试样本量下进行插补时，内存使用量都相近。相比之下，基线内存使用量会随样本量增加而迅速上升。这种差异源于**Spark**能够在 RAM 和硬盘之间动态交换数据，并在 RAM 容量达到上限时将数据卸载到硬盘。具体来说，测得的 RAM 使用量并未达到分配的完整 16 GB，这是因为我们只跟踪堆内存峰值，即专用于计算的空间，而**Spark** 会将其余分配的内存留作辅助任务（例如元数据存储）或其他系统操作之用 Karau 和 Warren (2017)。

除了内存效率之外，在对大型数据集进行插补时，bigMICE 的运行时间也优于基线方法。这一改进归因于在 Spark 中分布式执行基学习器，以及采用计算复杂度较低的机器学习模型实现。此外，基线方法依赖单线程计算，因此其运行时间会随着样本量增加而增长得更快。

这一局限性在使用随机森林的实验中尤为明显：运行时间变得长到难以接受，迫使测试提前终止。

运行时间还取决于待插补变量的数量。随着变量数量增加，运行时间也会增加；但当变量超过 30 个时，增幅会略微更加明显。这种影响可能源于在高维特征空间中拟合模型的复杂性增加。此外，硬盘读写速度和网络延迟也可能影响运行性能，尤其是在数据存储和计算分别在不同机器上进行时。在此情境下，检查点操作至关重要。检查点机制使该方法能够将大型计算的执行拆分为易于管理的多个部分，并通过定期将中间结果保存到磁盘来缓解内存限制。对于内存有限的大型数据集处理而言，这一机制必不可少，因为它可以防止内存不足错误并确保稳定性，但代价是引入会直接影响运行时间的输入输出开销。

样本量变化实验表明，尽管缺失值比例高达 50%，增加样本仍能提高插补质量。这是因为即使缺失比例高达 50%，大型数据集仍包含许多观测案例。样本量为 100 万行时，针对缺失比例变化开展的实验进一步印证了这一发现。结果表明，均方根误差均值变化不大，直到缺失比例达到极端水平，即 99.9%。

这展示了 bigMICE 对超大型数据集执行多重插补的巨大潜力。丰富的观测值使其能够实现良好的插补质量，即使缺失比例非常高也是如此。然而，需要注意的是，我们的实验假设数据属于完全随机缺失（MCAR），这可能并不总能反映现实场景。例如，在医疗记录等数据集中，PumpPagaendeModell 等变量可能并非随机缺失，而是因为患者不使用泵而缺失。若未对此加以考虑，这类非随机缺失（MNAR）机制可能会引入偏差。尽管如此，bigMICE 在高缺失比例下对大型数据集进行插补时表现出的稳健性，为此前因缺失比例过高而被排除的某些变量开辟了新的研究机会。

6.2. 未来工作

尽管当前实现对于大数据而言效率较高，但仍需进一步开发，以实现与原始软件包 mice 类似的易用性。mice 是一种广泛采用的多重插补工具，但其处理超大型数据集的能力有限。未来工作的主要方向包括以下几个方面。

- 扩展插补模型支持：可以拓宽支持的插补模型范围，以适应更多使用场景和数据类型。例如，sparklyr 提供的其他 MLLib模型尚未集成到 bigMICE 中。

- 增强功能与用户控制：为匹配 mice 的灵活性，该软件包应纳入自定义插补序列、收敛诊断和用户定义模型参数等功能。用户反馈有助于确定应优先实现哪些关键功能。

- 优化基于 Spark 的流程：可对 Spark 工作流进行优化，以减少计算开销并提高运行时间效率。例如，通过更少或更高效的 Spark 操作获得相同结果，或许能够降低计算成本。

- 评估集群计算性能：使用 Spark 实现 MICE 的一项主要优势，是有望进行分布式计算。我们的研究使用了一台独立的多核计算机，但评估集群环境中的性能也很有价值。不过，节点间数据传输量可能会带来额外开销。与独立计算机上的性能进行基准测试，有助于厘清分布式执行的权衡与优势。要进一步探索这一问题，需要使用集群。

7. 结论

软件包 bigMICE 是将 MICE 框架扩展到大数据场景的成功第一步。它能够明确控制插补过程中的 RAM 消耗，同时与一种广受欢迎的 MICE 实现相比，在超大型数据集上实现更快的计算速度。该方法在内存使用量和运行时间方面都能随着样本量的增加而高效扩展，使大数据多重插补能够在合理的时间内完成，即使使用普通笔记本电脑等计算资源有限的设备也不例外。

数据可用性

如需访问 NDR 数据集，可向登记处组织提出申请。对糖尿病登记处的数据开展研究的前提是，必须获得伦理审查委员会的批准、数据披露许可，并由西约塔兰地区中央数据控制员进行保密性评估 Eeg-Olofsson(2025)。

致谢

F.I 是瓦伦堡分子医学中心（WCMM）的副临床研究员，并获得克努特和爱丽丝·瓦伦堡基金会的经费支持。

参考文献

Apache 软件基金会（2025）。“Apache SparkTM - 大规模数据分析统一引擎。”访问日期：2025-03-10，网址 https://spark.apache.org/。

Arefin M, Masum AKM（2024）。“一种用于缺失数据插补的概率方法。”复杂。doi:10.1155/2024/4737963。

Austin PC, White IR, Lee DS, van Buuren S（2020）。“临床研究中的缺失数据：多重插补教程。”加拿大心脏病学杂志。 doi:10.1016/j.cjca.2020.11.010。Cesare N, Were LPO（2022）。“管理时间和患者变异电子健康记录中缺失数据的多步骤方法。”BMC Research Notes。doi:10.1186/s13104-022-05911-w。Dean J, Ghemawat S（2008）。“MapReduce：大型集群上的简化数据处理。”ACM通讯，51(1)，107–113。Devroye L（1986）。《非均匀随机变数生成》。Springer。ISBN 978- 1-4613-8645-2。doi:10.1007/978-1-4613-8643-8。URL https://doi.org/10.1007/978-1-4613-8643-8。Eeg-Olofsson K（2025）。“国家糖尿病登记处。” https://ndr.registercentrum.se/。访问日期：2025-08-18。Efron B（1981）。“标准误的非参数估计：刀切法、自助法及其他方法。”Biometrika，68(3)，589–599。ISSN 00063444，14643510。URL http://www.jstor.org/stable/2335441。Grund S, Lüdtke O, Robitzsch A（2016）。“多层缺失数据的多重插补：R 软件包 pan 简介。” SAGE Open。 doi:10.1177/2158244016668220。Gudbjörnsdottir S, Cederholm J, Nilsson PM, Eliasson B, 代表瑞典国家糖尿病登记处指导委员会（2003）。“瑞典国家糖尿病登记：圣文森特宣言在糖尿病质量改进中的实施。”Diabetes Care，26(4)，1270–1276。ISSN 0149-5992。 doi:10.2337/diacare.26.4.1270。 https://diabetesjournals.org/care/article-pdf/26/4/1270/659430/dc0403001270.pdf，URL https://doi.org/10.2337/diacare.26.4.1270。Hasan MK, Alam MA, Roy S, Dutta A, Jawad MT, Das S（2021）。“缺失值插补会影响机器学习的性能：文献综述与分析（2010–2021）。”Informatics in Medicine Unlocked。 doi:10.1016/j.imu.2021.100799。He Y, Zhang G, Hsu CH（2021）。“多元缺失数据的多重插补：完全条件设定方法。”《实践中的缺失数据多重插补》。 doi:10.1201/9780429156397-7。Karau H, Warren R（2017）。《高性能 Spark：扩展和优化 Apache Spark 的最佳实践》。O’Reilly Media, Inc. Khan SI, Hoque ASML（2020）。“SICE：一种改进的缺失数据插补技术。”Journal of Big Data。 doi:10.1186/s40537-020-00313-w。Kombo AY, Mwambi H, Molenberghs G（2017）。“具有单调缺失数据模式的有序纵向数据多重插补。”应用统计学杂志。 doi:10.1080/02664763.2016.1168370。Little RJA, Rubin DB（2019）。“《缺失数据的统计分析》第二版。”Wiley概率与统计丛书。 doi:10.1002/9781119482260。

Lo AW, Siah KW, Wong CH (2019). “用于预测药物审批的机器学习与统计插补。”第 1 期。doi:10.2139/ssrn.2973611。Pampaka M, Hutcheson G, Williams J (2016). “处理缺失数据：使用多重插补分析一个具有挑战性的数据集。”《国际教育研究与方法杂志》. doi:10.1080/1743727x.2014.979146。Resche-Rigon M, White IR (2018). “使用链式方程多重插补处理系统性和零星缺失的多层数据。”《医学研究统计方法》. doi:10.1177/0962280216666564。Rubin DB (2004). “调查无应答的多重插补 / Don-ald B. Rubin。” URLhttps://research.ebsco.com/linkprocessor/plink?id=66d3bd8f-da75-32b7-ad66-2995a0e9d45a。Salloum S, Dautov R, Chen X, Peng PX, Huang JZ (2016). “基于 Apache Spark 的大数据分析。”《国际数据科学与分析杂志》, 1(3)，145–164。 doi:10.1007/s41060-016-0027-9。URL https://doi.org/10.1007/s41060-016-0027-9。van Buuren S (2007). “通过完全条件设定对离散数据和连续数据进行多重插补。”《医学研究统计方法》.doi:10.1177/0962280206074463。van Buuren S (2018). “《缺失数据的灵活插补》第二版。”null。 doi:10.1201/9780429492259。van Buuren S, Brand J, Groothuis-Oudshoorn CGM, Rubin DB (2006). “多变量插补中的完全条件设定。”《统计计算与模拟杂志》. doi:10.1080/10629360600810434。van Buuren S, Groothuis-Oudshoorn K (2011). “mice：R 中的链式方程多变量插补。”《统计软件杂志》，45(3)，1–67。 doi:10.18637/jss.v045.i03。URL https://www.jstatsoft.org/index.php/jss/article/view/v045i03。Wang C, Shakhovska N, Sachenko A, Komar M (2020). “大数据接口中缺失数据插补的新方法。”《信息技术与控制》. doi:10.5755/j01.itc.49.4.27386。White IR, Royston P, Wood AM (2011). “使用链式方程进行多重插补：实践中的问题与指导。”《医学统计》. doi:10.1002/sim.4067。Xiangrui Meng JB, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A (2016). “MLlib：Apache Spark 中的机器学习。”《机器学习研究杂志》, 17(34)，1–7。URL http://jmlr.org/papers/v17/15-237.html。Yang Y, Kim JK, Cho IH (2020). “用于大规模不完整数据处理的并行分数热卡插补与方差估计。”《IEEE知识与数据工程汇刊》. doi:10.1109/tkde.2020.3029146。

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012). “弹性分布式数据集：一种用于内存集群计算的容错抽象。”载于第九届 *USENIX* 网络系统设计与实现会议论文集，NSDI’12，第 2 页。USENIX 协会，美国。

Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I（2016）。“Apache Spark：大数据处理的统一引擎。”*Commun. ACM*，**59**(11), 56–65. ISSN 0001-0782。 doi:10.1145/2934664。网址 https://doi.org/10.1145/2934664。

A. mice.spark() 参数参考

本附录详细概述了 `mice.spark()` 函数可用的参数。

sc. 使用 `sparklyr::spark_connect()` 创建的 Spark 连接。该连接提供了上下文，所有数据处理和建模都将在其中执行。

data. 包含缺失值数据集的 Spark DataFrame。该函数直接对这个分布式数据对象进行操作，无需将其转换为内存中的 `R`数据框。

变量_类型。 一个命名字符向量，用于指定数据集中每一列的类型（例如， `"Continuous_int"`、 `"Binary"`、 `"Nominal"`）。这些类型将指导初始化过程和插补方法的选择。

分析_公式。 描述感兴趣分析模型的公式，例如 `outcomeage + income + education`。每次插补后，都会将此公式传递给 Spark MLLib，以拟合分析模型。

m. 要生成的多重插补次数。默认值为 5。较大的值可减少蒙特卡洛误差，但会增加计算成本。

method. 指定用于每个变量的插补策略。如果为 `NULL`，则根据 `variable_types` 推断默认值。用户可提供字符向量进行覆盖，例如指定基于回归、逻辑回归或多项式插补。

predictorMatrix. 可选的二元矩阵，用于定义哪些变量用作其他变量插补时的预测变量。行对应待插补的变量，列对应潜在的预测变量。默认情况下，除变量自身外，所有其他变量均会使用。

formulas. 由 Spark 兼容公式组成的列表，用于为各个变量指定自定义插补模型。此设置会覆盖 `predictorMatrix` 所隐含的预测变量结构。

modeltype. 指定用于插补的统计模型类型，例如线性回归、逻辑回归或多项式回归。如果为 `NULL`，则根据 `variable_types` 推断。

maxit. 每次插补中链式方程要执行的最大迭代次数。默认值为 5。较大的值有助于更充分地收敛，但需要更多计算。

printFlag。 控制执行期间是否打印诊断消息和进度消息的逻辑标志。适用于调试或监控长时间计算。

seed。 用于确保插补结果可复现的整数随机种子。如果为 `NA`，则会选择一个随机种子。

插补_初始化。一个可选的 Spark 数据框，包含经过初始轮插补后的数据（例如使用均值、中位数或随机抽样）。如果未提供，则由系统在内部处理初始化。

检查点。用于指示是否应启用 Spark 检查点的逻辑标志。默认值为 `TRUE`。建议启用，以防止血缘增长以及 Spark 中可能出现的堆栈溢出错误。

检查点_频率。控制在 `checkpointing= TRUE` 时触发检查点的频率。默认值为 10，即每处理 10 个变量后执行一次检查点。较低的值可减少血缘增长，但会增加磁盘输入 / 输出。

B. 附加表格

表8：NDR 注册表变量元数据摘要。

变量名称	缺失（%）	数据类型
alder	0.00	整数
LopNr	0.00	数值型（代码）
登记日期	0.00	小日期时间
SenPNr	0.00	数值型（代码）
sex	0.00	名义型
临床_糖尿病_类型	0.09	名义型
糖尿病_行为 / 治疗	3.19	名义型
降压药	3.57	二元
发病年份	3.68	整数
降脂药	3.95	二元
asa	5.18	二元
临床_糖尿病_型_登记	9.41	数值型（代码）
身高	9.63	整数
缺血性_心脏病	11.87	二元
中风	12.97	二元
足部检查	18.43	数值型（代码）
足部风险	24.01	名义型
ogon_日期	25.27	小日期时间
fot_日期	25.67	小日期时间
视网膜病变	31.20	二元
糖化血红蛋白	31.62	整数
收缩压	34.64	整数
舒张压	34.78	整数

续见下页

表8 – 接上一页

变量名称	缺失（%）	数据类型
发病年份_reg	38.90	数值型（代码）
kreatinin	44.25	整数
GFR	44.27	浮点数
rokare	45.05	名义型
vikt	45.27	浮点数
bmi	47.21	浮点数
吸烟习惯	54.94	有序
胆固醇	55.63	浮点数
ldl	56.25	浮点数
身体_活动	56.58	有序
白蛋白尿	57.89	名义型
hdl	58.11	浮点数
甘油三酯	62.31	浮点数
低血糖	65.77	有序
ins_方法	66.98	名义型
视力障碍	72.92	二元
死亡日期	78.91	小日期时间
吸鼻烟习惯	81.63	有序
CGM	87.84	二元
尿白蛋白肌酐	89.72	浮点数
眼部治疗	90.38	二元
视网膜病变诊断	90.72	名义型
远程	93.07	二元
治疗年度	95.01	整数
胰岛素	95.14	二元
CGM类型	95.92	名义型
过去两周葡萄糖占比	98.13	整数
葡萄糖范围占比	98.15	整数
碳水化合物	98.44	二元
PumpPagaendeModell	98.55	名义型
近两周平均血糖	98.67	浮点数
近两周连续血糖监测标准差	99.00	浮点数
泵治疗适应证	99.34	名义型
停用年份	99.56	整数
泵治疗终止原因	99.96	名义型

C. 附加代码示例

C.1. 内存监控

监控 *R* 函数执行的内存

通过监控父进程的内存使用量，来监控 R 函数的内存使用情况。函数 `monitor_memory` 会创建一个并行进程，在执行期间跟踪其父进程的内存，从而监控 R 脚本的内存使用情况。以下示例展示了如何使用 `monitor_memory` 函数监控一个简单过程的内存使用量和运行时间。

```
R> mice_procedure <- function(){R>   library(mice)
R>   sim_data <- data.frame(R>     ID = 1:10000,
R>     Age = sample(18:80, 10000, replace = TRUE),
R>     Income = round(rnorm(10000, mean = 50000, sd = 15000), 0),
R>     Score = runif(10000, min = 0, max = 100),
R>     Category = sample(c("A", "B", "C"), 10000, replace = TRUE)R>   )
R>   amputed_data <- ampute(sim_data, prop = 0.2)$amp
R>   imputed_data <- mice(amputed_data, m = 5, method = 'pmm', seed = 123)
R>R>   return(imputed_data)R> }
```

```
R> res <- monitor_memory(
R>   func = mice_procedure
R> )

R> max_mem_during <- max(res$memory_data$memory_r_mb[res$memory_data$phase=="during"])

R> print(max_mem_during)
[1] 486.5859

R> print(res$run_time)
[1] 1.765408
```

监控 *Spark* 应用程序的内存

监控 Spark 应用程序的内存使用量相对更为直接，因为 Spark 用户界面在运行时间执行期间已经提供了许多指标测量结果。要收集这些指标，我们只需向 Spark 用户界面发出 GET 请求，并从返回的 JSON 文件中提取感兴趣的指标。以下代码展示了如何在 Spark 会话中启用指标收集，然后在 Spark 作业完成后提取这些指标。

```
R> get_spark_ui_port <- function(sc) {
R>   sc_context <- spark_context(sc)
R>   ui_url_option <- invoke(sc_context, "uiWebUrl")
```

```
R>   if (is.null(ui_url_option)) {
R>     warning("Spark UI URL not available.")
R>     return(NA_integer_)
R>   }
R>   ui_url <- invoke(ui_url_option, "get")
R>   port_str <- str_extract(ui_url, "(?<=:)\\d+")
R>   return(as.integer(port_str)) # Convert to integer and return
R> }

R> monitor_spark_memory <- function(sc, physical_limit_gb = 16) {
R>   app_id <- spark_context_config(sc)$spark.app.id
R>   ui_port <- get_spark_ui_port(sc) # usually 4040, but can be 4041, 4042,...
R>   if(is.null(ui_port)) message("Could not find ui port")R>   # Try to connect to Spark UI
R>   endpoint <- paste0("http://localhost:", ui_port, "/api/v1/applications/", app_id
                                                                                    ,
   "/executors")R>   response <- GET(endpoint, timeout(5))
R>   if (status_code(response) == 200) {
R>     data <- content(response, "text") %>% fromJSON(flatten = TRUE)
R>     summary_tbl <- data.frame(R>       executor_id = data$id,
R>       total_cores = data$totalCores,
R>       max_memory_gb = data$maxMemory / (1024^3),
R>       memory_used_gb = data$memoryUsed / (1024^3),
R>       peak_jvm_heap_memory = data$peakMemoryMetrics.JVMHeapMemory / (1024^3),
R>       total_gc_time_seconds = data$peakMemoryMetrics.TotalGCTime / 1000,
R>       disk_used_gb = data$diskUsed / (1024^3),
R>       stringsAsFactors = FALSER>     )
R>     print("Memory Statistics Summary:")R>     print(summary_tbl)
R>     return(list(R>       port_used = ui_port,R>       app_id = app_id,
R>       raw_data = data,R>       summary = summary_tblR>     ))R>   }
R>   message("Unable to connect to Spark UI and retrieve metrics.")
R>   return(NULL)R> }
```

然后，我们可以在 Spark 作业结束时使用上述函数获取感兴趣的指标：

```
R> conf <- spark_config()
R> # Basic parameters
```

```
R> conf$`sparklyr.shell.driver-memory` <- "16g"
R> conf$`sparklyr.shell.executor-memory` <- "16g"
R> conf$spark.memory.fraction <- 0.9
R> conf$`sparklyr.cores.local` <- 8
R> conf$`spark.local.dir` <- "/local/data/spark_tmp/"

R> # Enable executor metrics
R> conf$spark.metrics.executorMetricsSource.enabled <- "true"
R> conf$spark.eventLog.enabled <- "true"
R> conf$spark.eventLog.dir <- "./"
R> conf$spark.eventLog.logStageExecutorMetrics <- "true"

R> sc <- spark_connect(master = "local",R>                      config = conf,
R>                     spark_home = "/home/user/spark/spark-4.0.0-bin-hadoop3")


R> spark_set_checkpoint_dir(sc, "/local/data/spark_tmp/" )

R> sdf <- spark_read_csv(sc, "path/to/data", infer_schema = TRUE, null_value = "NA")

R> variable_types <- c(alder = "Continuous_int",
R>                     bmi = "Continuous_float",
R>                     stroke = "Binary")

R> start_time <- Sys.time()

R> data <- sdf,R> mice.spark(                sc = sc,
R>                              variable_types = variable_types,
R>                              analysis_formula = as.formula("stroke ~ alder + bmi"),
R>                              predictorMatrix = NULL,
R>                              m = 2,R>                              maxit = 5)


R> end_time <- Sys.time()
R> # Collect memory data and close connection
R> spark_mem <- monitor_spark_memory(sc = sc)

R> spark_disconnect(sc)

R> runtime <- as.numeric(difftime(end_time, start_time, units="secs"))
R> max_spark_memory <- spark_mem$summary$peak_jvm_heap_memory * 1000 # Gb -> Mb
```

所属机构：
名 姓 单位 地址，国家 电子邮箱：
name@address网址：
http://link/to/webpage/

样本量	方法	内存（Gb）（均值 ± sd）	运行时间（分钟）（均值 ± sd）
1000	mice	0.391 ± 0.000	0.012 ± 0.001
	带随机森林的 MICE	0.348 ± 0.015	0.020 ± 0.001
	bigMICE	7.929 ± 0.500	2.219 ± 0.025
2902	mice	0.430 ± 0.000	0.260 ± 0.001
	带随机森林的 MICE	0.410 ± 0.012	0.061 ± 0.002
	bigMICE	8.627 ± 1.050	2.659 ± 0.030
8426	mice	0.462 ± 0.008	0.313 ± 0.002
	带随机森林的 MICE	0.455 ± 0.009	0.156 ± 0.002
	bigMICE	8.429 ± 1.133	2.736 ± 0.018
24459	mice	0.563 ± 0.007	4.861 ± 0.002
	带随机森林的 MICE	0.483 ± 0.007	4.213 ± 0.004
	bigMICE	8.371 ± 0.954	2.957 ± 0.017
70999	mice	0.732 ± 0.011	6.569 ± 0.007
	带随机森林的 MICE	0.723 ± 0.036	5.105 ± 0.022
	bigMICE	8.789 ± 0.751	3.438 ± 0.017
206096	mice	1.242 ± 0.065	7.084 ± 0.019
	带随机森林的 MICE	1.307 ± 0.017	15.829 ± 0.033
	bigMICE	10.290 ± 0.908	4.216 ± 0.043
598253	mice	2.579 ± 0.145	10.930 ± 0.025
	带随机森林的 MICE	3.154 ± 0.130	87.238 ± 0.219
	bigMICE	7.601 ± 0.775	5.446 ± 0.036
1736597	mice	6.299 ± 0.465	20.897 ± 0.043
	带随机森林的 MICE	NaN ± NA	NaN ± NA
	bigMICE	9.758 ± 1.202	7.943 ± 0.085
5040959	mice	16.856 ± 0.914	51.226 ± 0.152
	带随机森林的 MICE	NaN ± NA	NaN ± NA
	bigMICE	11.863 ± 0.893	14.754 ± 0.065
14632799	mice	40.723 ± 3.939	158.088 ± 0.391
	带随机森林的 MICE	NaN ±NA	NaN ± NA
	bigMICE	11.629 ± 0.570	36.754 ± 0.285

表3：性能比较

变量数量	平均运行时间（秒）	标准差（秒）	运行时间增幅
5	32.98	5.00	1.00x
10	75.43	7.20	2.29倍
15	120.72	10.63	3.66倍
20	180.15	9.91	5.46倍
25	245.06	10.62	7.43倍
30	331.78	10.82	10.06倍
35	398.64	16.78	12.09倍
40	488.36	15.62	14.81x
45	576.16	16.30	17.47x
49	713.34	29.35	21.63x

表 4：运行时间汇总：均值、标准差，以及相对于使用 5 个变量时运行时间的相对增幅

表 5：变量、类型和所用插补模型概览

变量	**Type**	**插补模型**
alder	整数	线性回归
sex	名义型	多项逻辑回归
糖化血红蛋白	整数	线性回归
klin_糖尿病_typ	名义型	随机森林分类器
发病年份	整数	线性回归
diab_beh	名义型	随机森林分类器
降压药	二元	逻辑回归
降脂药	二元	逻辑回归
asa	二元	逻辑回归
身高	整数	线性回归
收缩压	整数	线性回归
舒张压	整数	线性回归
kreatinin	整数	线性回归
GFR	**浮点数**	**随机森林回归**
视网膜病变	二元	逻辑回归
bmi	浮点数	线性回归
胆固醇	浮点数	线性回归
ldl	浮点数	线性回归
hdl	浮点数	线性回归
白蛋白尿	名义型	随机森林分类器

样本量	均方根误差均值	标准差
1000	1.70038	0.003850
5000	1.68944	0.002221
9503	1.62626	0.001940
90303	1.63172	0.001524
858129	1.61430	0.002354
8154612	1.62250	0.002431

表6：当缺失比例为50%时，样本量对GFR变量插补质量的影响（m=5次插补）

缺失比例（%）	均方根误差均值	标准差
10	1.61979	0.000610
30	1.61666	0.000276
50	1.61509	0.000254
70	1.61634	0.002322
90	1.61088	0.000393
99	1.62880	0.001507
99.9	1.70810	0.003224

表 7：缺失比例对 100 万个样本（m=5 次插补）中 GFR 变量插补质量的影响