

# 計量経済学

## 15. 再現可能な研究

矢内 勇生

経済・マネジメント学群



KOCHI UNIVERSITY OF TECHNOLOGY

2018 年 11 月 27 日

# 今日の内容



## 1 再現可能な研究

- 再現可能性
- 再現可能な研究をしよう！

## 2 再現可能な研究を R と RStudio で実施する

- 記録、記録、記録！！！
- コードの可読性を高める
- 文芸的プログラミング入門

# 再現可能な研究とは何か



## 再現可能な研究 (reproducible research)

- 他の研究者が実施しても同じ結果が得られる
- 研究に用いたデータやコードを結果とともに公表する
- 研究の透明性
- 研究のより深い理解

# Replication と Reproduction



- ① 同じデータと方法を用いれば、誰が実行しても同じ研究結果が得られる： **reproduction**
  - ② 同じ（類似の）方法を異なるデータに適用しても、研究の実質的な結果が同じ： **replication**
- 
- 科学研究は、1 と 2 の両者を確保する必要がある
  - 2 を欠く研究は、誤っているかもしれない（通常の科学）
  - 1 を欠く研究は、科学ではない（擬似科学、非科学）
  - すべての科学者が同意しているわけではない

## 何をすべきか



- 研究過程を細かく記録する
- 研究結果がだけでなく、研究のプロセス全体をすべて書き留める
- 研究成果を公表するときは、データも公表する
- データ分析を行ったときは、分析に使ったコンピュータコードも公表する
  - 他人が読んでも理解できる読み易いコードを書かなければならぬ
  - 他人が読んでもわかるようにたくさん（コメント等）書かなければならぬ

再現可能な研究をしよう！

## 再現可能な研究の利点



- 研究手続きの洗練化
- 共同研究の促進
- 被引用数の増加（特に、データを公表した場合）
- 知の蓄積を促進する
- コミュニティに対する貢献（→評判を上げる）

再現可能な研究は得である！

# 研究の記録法



- データをどうやって集めたか
- データセットやそれに含まれる変数をどのように作ったか
- どのようにデータを分析したか
- データ分析の結果をどのように解釈したか、またその理由
- それぞれにタイムスタンプ（日時）をつける

## データ分析プロジェクトを RStudio で管理する



- 1つの研究（1つの論文）に関連するファイルは、すべて1つの場所（フォルダ）にまとめておく
- RStudio を使うと、管理が楽になる（git 等のバージョン管理を併用するとさらに良い）
- RStudio で新規プロジェクトを作る：[File] → [New Project]
  - 既に存在するフォルダを使うときは [Existing Directory] を選ぶ
  - ゼロからプロジェクトを立ち上げるときは [New Directory] を選ぶ
- プロジェクトには容易に判別可能な名前をつける：プロジェクトの拡張子は “.Rproj”
- RStudio でプロジェクトを開くと、プロジェクトのフォルダが自動的に作業ディレクトリに選ばれる：`setwd()` を実行する必要がない

## RStudio で R コードを書く



- プロジェクトを開く： [File] → [Open Project]
- 新しい R スクリプトを開く： [File] → [New File] → [R Script]
- スクリプトを名前をつけて保存する： ファイルの拡張子は “.R”
- R スクリプトはテキストファイルなので、テキストエディタなら（R がインストールされていなくても）開ける
- “#” を使ってコメントをつける
- スクリプトの冒頭にファイルの説明をつける
- “ctrl (or cmd) + enter” で現在の行を評価（実行）する

## R スクリプトに何を書くべきか



- ファイル名
- スクリプトを書いた目的
- スクリプトを実行するために必要なファイル (e.g., データファイル) とスクリプトを実行した結果できるファイル (e.g., 整形されたデータファイル)
- ファイルの作成日と作成者
- ファイルの修正日と修正者
- コードに対するコメント：何をしたかよりも、なぜそれをしたか

## スクリプトの例



```
#####
## example.R
## Purpose: R コードの書き方を説明する
## 使用するデータファイル
##     data/fake-data-01.csv
##     data/fake-data-02.dta
## Created: 2018-10-01 Yuki Yanai
## Last Modified: 2019-11-26 YY
#####

## tidyverse パッケージを読み込む
library("tidyverse")
```

## コードを書くときに考慮すべきこと



- 可読性：適切なスペーシング、改行、字下げ（ブロック化）
- オブジェクト名の一貫性： e.g, linear\_model or linearModel
- 来週, 来月, 来年, 再来年, 5 年後, 10 年後 … に読んでも内容を理解できるか
- 他人が読んでも理解できるか
- コメントが少な過ぎないか（多すぎることは決してない！）  
大まかな目安：スクリプトの文字のうち 30–70% はコメント
- 以上のこととは、R 以外の言語（Python スクリプトなど）にも当てはまる

## Rスクリプトの長所と短所



### 長所

- スクリプト全体を一度に実行できる

```
# run the script  
source("example.R")
```

- 書くのが簡単

### 短所

- コードを文章で説明するのに向いていない
- コードの説明と、結果（出力）と一緒に読めない
- テキストエディタによるキーワードのハイライトを除いては、単なるテキストファイルである

## 人間に優しいコードを！ コードの可読性を高める



- 「正しい」 コードは優れたコードの必要条件だが、十分条件ではない
- 優れたコードは可読性が高いコードである
  - 管理、変更、再利用が容易
  - 共同研究が行いやすい
  - 透明性が高い

## 可読性 (1) : コメント



コメントをたくさん書け！

- 他人のコードを読んでいるつもりで、気になる点をコメントする
- 例：算術平均を計算する関数に対するコメント

```
get_mean <- function(x) {  
  ## Function to calculate the arithmetic mean  
  ## Argument: x = a numerical vector  
  ## Return: mean_x = the arithmetic mean of x  
  
  n <- length(x)    # n is the length of x  
  sum_x <- sum(x)   # get the sum of all x  
  mean_x <- sum_x / n  
  
  return(mean_x)    # return the mean  
}
```

## 可読性 (2) : 字下げによるコードのブロック化



半角 2 文字または 4 文字の字下げで、コードブロックを作る

### ● 悪い例

```
for(i in 1:n){  
  for(j in 1:k){  
    x[i, j] <- mean(rnorm(10))  
  } }
```

### ● 良い例

```
for (i in 1:n) { # loop for the rows of x  
  for (j in 1:k) { # loop for the columns of x  
    x[i, j] <- mean(rnorm(10))  
  }  
}
```

## 可読性 (3) : スペースと改行の適切な利用



コードがより美しく見えるように、スペースと改行を利用する

- 悪い例

```
a<- (1+2) *4+5-8  
plot(x,y,xlim=c(1,10),ylim=c(-5,5),xlab="x-la  
bel",ylab="y-label",main="Title_of_fig")
```

- 良い例

```
a <- (1 + 2) *4 + 5 - 8  
plot(x, y, xlim = c(1, 10), ylim = c(-5, 5),  
      xlab = "x-label", ylab = "y-label",  
      main = "Title_of_fig")
```

# 文芸的プログラミングとは何か



## 文芸的プログラミング (literate programming)

コンピュータプログラム（コード）を自然言語（日本語、英語など）によるプログラムの説明や解釈とともに書く方法

- Donald Knuth ( $\text{\TeX}$  の開発者) が提唱
- 1つのファイルを書くだけで、データ分析の結果とその結果をまとめたレポートをまとめて作れる

# RStudio を用いた文芸的プログラミング



## ① プロジェクトを開く

**File** → [New File] → [R Markdown] と選択する

## ② 名前をつけてファイルを保存する：ファイルの拡張子は “.Rmd”

## ③ ファイルのヘッダ情報を入力する

## ④ コードの説明や結果の解釈などを通常の文で書く： Markdown を使う

## ⑤ R コードを「コードチャンク」に書く

## ⑥ コード編集画面の上にある [Knit HTML/PDF] をクリックし て HTML/PDF ファイルを作る

# R Markdown 入門



ヘッダを書く： ヘッダは開始と終了に 3 つのハイフンをもつ

---

```
title: "RStudio を用いた文芸的プログラミング入門"
```

```
author: "矢内 勇生"
```

```
date: "2018-11-26"
```

```
output:
```

```
  html_document:
```

```
    theme: united
```

```
    highlight: tango
```

```
    toc: true
```

---

## R Markdown : 文章の書き方



通常の文は、ワープロソフトやテキストエディタと同様に書く

- ヘディングは “#” で始める：“#” の数が少ないほど上位
- “\*\*” または “\_” で挟まれた部分はイタリック
- “\*\*” または “\_\_” で挟まれた部分は太字
- “\*\*\*\*” または “\_\_\_” で挟まれた部分は太字のイタリック
- 箇条書き
  - 順序のない箇条書きは “-” (ハイフン) で始める
  - 順序のある箇条書きは 数字で始める (数字はなんでもよい。全部 1 でもよい)
  - Tab で字下げすると箇条書きを入れ子にできる
- URL リンク：[リンク先の説明](URL)
- 画像：![画像が表示できないときに表示する説明](画像ファイルへのパス)

## R Markdown : コード



- R コードはコードチャunkに書く
- コードチャunkの始点: '```{r chunk-name,  
chunk-options}
- 終点: '``'
- 始点と終点の中に R コードを書く
- 各チャunkにユニークな名前をつける
- 必要なら、チャunkのオプションを指定する

## R Markdown : 文章の中にコードを入れる



- コード自体を見せるときは、逆引用符で挟む：例  
`'sessionInfo()'`
- コードを評価した結果を見せたいとき
  - (入力)  $x$  の分散は `'r var(x)'` です
  - (出力)  $x$  の分散は 30.8 です