JAS\_DataManagementFor13Tokyo-20250619ver

kotdijian

2025年6月19日

## 0.はじめに

### 概要

これは、[考古学・文化財のためのデータサイエンス・サロンオンライン#03オープンリソースによる遺跡地図作成実習part2](https://3dlm.peatix.com/)の教材資料として作成されたRマークダウン・ドキュメントを、2025年度早稲田大学文学学術院「考古学研究9」の授業用に改変したものです。

[Rマークダウンとは?](https://kazutan.github.io/kazutanR/Rmd_intro.html)

[RStudio公式によるRマークダウンチートシート](https://rstudio.com/wp-content/uploads/2016/11/Rmarkdown-cheatsheet-2.0_ja.pdf)

実習では、[東京都遺跡地図情報インターネット提供サービス](https://tokyo-iseki.metro.tokyo.lg.jp/)の収録情報を[リスト化したもの](https://github.com/kotdijian/JASOSR/tree/master/13Tokyo)を整理・加工するためのコードを通じて、Rによるデータ操作を学びます。

## 1.Rとは何か

R言語（あーるげんご）はオープンソース・フリーソフトウェアの統計解析向けのプログラミング言語及びその開発実行環境である。([日本語版Wikipedia「R言語」](https://ja.wikipedia.org/wiki/R%E8%A8%80%E8%AA%9E))

Rは統計計算処理とグラフィックスのためのフリーのソフトウェア環境です。([R公式サイトトップ](https://www.r-project.org/))

Rは統計計算処理とグラフィックスのための言語および環境です。(中略)Rは、線形・非線形モデリング、古典的統計検定、時系列分析、分類、クラスター分析など多様な統計処理とグラフィックスを提供し、非常に拡張性の高いものです。([R公式サイト「Rとは」](https://www.r-project.org/about.html))

R言語（あーるげんご）はオープンソース・フリーソフトウェアの統計解析向けのプログラミング言語及びその開発実行環境である。([日本語版Wikipedia「R言語」](https://ja.wikipedia.org/wiki/R%E8%A8%80%E8%AA%9E))

## 2.R(とRStudio)の基本操作

### 2-1.関数(function)

Rの基本は**関数**(function)の組み合わせで行ないます。ここでの**関数**とは、与えられた入力(input)に対して定められた処理を行ない、結果を出力(output)するものです。

以下、RStudioを使用する前提で操作方法を確認します。

まず最初に、Consoleにprint("こんにちは")と入力（またはコピーペースト）してみましょう。

print()は()内の内容を出力表示する関数です。““(ダブルクオーテーション)で囲んだ文字列を出力するほか、数値などを代入したオブジェクト(object)や、表(table)などを出力することもできます。

実行できたら”“内に任意の文字列を入力して確認して見てください。

### 2-2.数値の計算とオブジェクト

次に、Consoleに1 + 2と入力してみましょう。

計算結果が直接表示されます。

それでは、以下を入力するとどうなるでしょうか?

x <- 1  
y <- 2  
z <- x + y

x, y, zはそれぞれ**オブジェクト**です。**オブジェクト**とは、数値や文字列などのデータや計算結果を格納して保存しておくためのものです。

<-は、左側に示された**オブジェクト**に、右側の入力を格納することを指示する**代入演算子**です。

この3行の**コード**は、xに1、yに2を代入し、zにx + yの結果を代入するものです。

そしてこの3行のコードを実行しても、先ほどの計算(1 + 2)の時のように結果が表示されません。代わりに、右上の**Environment**ウインドウ(pane)に、x, y, zと、それぞれの入力結果が表示されているはずです。

次に、’print(x)と入力して見てください。xの値が表示されます。単にx `と入力するだけでも、同じ結果を得られます。

### 2-3.文字データと行列データ

オブジェクトには、数値だけでなく文字列を格納することができます。以下を入力して見てください。

a <- "私の名前は"  
b <- "まだない"  
c <- paste(a,b)  
c

a、bの内容は数値ではないので、四則演算を適用できません。()内の文字列を結合するpaste()関数を使用します。

オブジェクトには、複数の数値や文字列を格納することもできます(行列またはベクトル)。c(#,#,#)のように記述します。以下のとおりに入力し、出力結果およびEnvironmentウインドウの内容確認しましょう。

s <- c(1, 2, 3)  
t <- 4  
u <- s + t  
s  
s[2]  
u

d <- c("神奈川","千葉","埼玉")  
e <- "県"  
f <- paste(d, e)  
f[3]

**Environment**ウインドウのsおよびuの項に表示されるnumは数値を示します。同じくdおよびfの項に表示されるchrは文字列を示します。

数値行列に四則演算を適用すると、行列内の各項をそれぞれ計算した結果が、行列として返されます。

文字行列にpaste()関数を適用すると、行列内の各項をそれぞれ結合した結果が、行列として返されます。

行列データを格納したオブジェクトのn番目のデータのみを示す場合はオブジェクト名に[n]を添えます。s[2]は、sの2番目のデータ、すなわち2を示します。

### 2-4.オブジェクトの消去

ここまでを理解できたら、いったん、操作練習で作成したオブジェクトを消去しましょう。

オブジェクトの消去にはrm()関数を使用します。カッコ内に消去したいオブジェクト名を入力します。ここでは、すべてのオブジェクトをいったん消去するため、**Console**ウインドウに、rm(list = ls())と入力してください。**Environment**ウインドウのすべてのオブジェクトが消え、Environment is emptyが表示されているのを確認してください。

## 3.パッケージの準備

### 3-1.パッケージとは

Rにはbaseと呼ばれる基本関数があります([BaseRチートシート日本語版](https://rpubs.com/keisato/base-r))

さらに基本関数を組み合わせて、より高度な操作を行なう関数を作成することができます。有用な関数を設定した[*パッケージ*(package)](https://syunsuke.github.io/r_beginners_guide/04_package.html)が多数開発され、提供されていることがRの特徴のひとつです。

これらのパッケージはR同様、オープンソースでありフリーです。CRAN (**C**omprehensive **R** **A**rchive **N**etwork)と呼ばれる公式のアーカイブのほか、**GitHub**等を通じて作成者が公開しているパッケージが多数あります。

今回は、**Tidyverse**、**rio**、**utf8**、**bit64**というパッケージを使用します。

### 3-2.パッケージのインストールとアクティベート

インストールされたパッケージは、Packagesウインドウに表示されます。

インストールしたRパッケージは、library()関数でアクティベートする必要があります。アクティベートされたパッケージは、Packagesウインドウでパッケージ名の前のボックスにチェックが入ります。

以下に、パッケージのチェックとインストール、アクティベートのコードを**コードチャンク**として示します。

#パッケージチェックとインストール  
if(!require("tidyverse")) install.packages('tidyverse', repos='http://cran.us.r-project.org')   
if(!require("rio")) install.packages('rio', repos='http://cran.us.r-project.org')   
if(!require("utf8")) install.packages("utf8", repos='http://cran.us.r-project.org')  
if(!require("bit64")) install.packages("bit64", repos='http://cran.us.r-project.org')  
  
#パッケージのアクティベート  
library("tidyverse") #下に詳述  
library("rio") #データの読み込み  
library("utf8") #UTF-8エンコーディング対応  
library("bit64") #UID13桁対応

**コードチャンク**はRマークダウンの特徴的な機能のひとつで、文書の中にコードを実行可能なかたちで埋め込んだものです。

マークダウン文書中に、灰色の背景で区画された範囲が**コードチャンク**です。右上に右向きの緑三角の実行ボタン(Run current chunk)があり、これをクリックするとチャンク内のコードが全て実行されます。

11行のコードは、if(!require("パッケージ名")) install.packages("パッケージ名", repos = "リポジトリURL")として、必要とするパッケージがすでにインストールされているかどうかをチェックし、まだの場合はインストールを実行した後、library()でアクティベートするというコードです。

なお**コードチャンク**中に#を付されているのはコメントまたは注釈と呼ばれるもので、プログラムの実行に関係のないメモとして記述されるものです。

パッケージのインストールには時間がかかる場合があります。すでにインストール済みのパッケージは右下の**Packages**ウインドウに表示されるので、検索・確認して見てください。

すでにtidyverse、rio、utf8、formattable、bit64がインストール済みの場合は、157～160行目を選択し、Ctrl+Enterで実行してください。この方法で、チャンク全体ではなく選択範囲のコードのみを実行することができます。

アクティベートが完了したパッケージは、**Package**ウインドウのパッケージ名の左の四角にチェックマークが入ります。これにより、各パッケージに含まれる関数・機能が使用可能になります。

### 3-3. Tidyverseについて

[Tidyverse](https://www.tidyverse.org/))は、おそらく、データサイエンスまわりではもっとも多用されている、ある意味では必須のRパッケージでしょう。

**Tidyverse**は、データを取り込み、整然データ(Tidy data)に加工して、分析や可視化に提供することを目的とするもので、以下のパッケージ群を含みます。  
- [tibble](https://tibble.tidyverse.org/): Tidyverse用のデータフレーム  
- [dplyr](https://dplyr.tidyverse.org/): データフレーム(後述)操作のパッケージ  
- [tidyr](https://tidyr.tidyverse.org/): 整然データを作るためのパッケージ  
- [readr](https://readr.tidyverse.org/): .csvなどの表データファイル読込のためのパッケージ  
- [stringr](https://stringr.tidyverse.org/): 文字列を操作するためのパッケージ  
- [purrr](https://purrr.tidyverse.org/): 繰り返し計算のためのパッケージ  
- [forcats](https://forcats.tidyverse.org/): 因子型(factor)データを扱うためのパッケージ  
- [ggplot2](https://ggplot2.tidyverse.org/): グラフ描画パッケージ

ここでは各パッケージの詳細には触れませんが、インターネット上で日本語のものを含む多数の解説が提供されています。検索して見てください。

#### 参考

* [「R自学自習の基礎知識」HeavyWatal](https://heavywatal.github.io/rstats/intro.html)
* [「Tidyverse」Nishii’s Notebook](http://bcl.sci.yamaguchi-u.ac.jp/~jun/notebook/r/tidyverse/)

体系的に学習したい方のためには、**Tidyverse**の作者による著書 “*R for Data Science*” の日本語版が刊行されています。 - [『Rではじめるデータサイエンス』H.Wickham & G.Grolemund,2017,オライリージャパン](https://amzn.to/3k2me28)

### 3-4.整然データ(tidy data)

**Tidyverse**の根幹は、**整然データ(tidy data)**です。**整然データ**とは、構造と意味が合致するものであり、Rをはじめとするデータの分析にもっとも適したかたちにデータを整えておくことが重要であるという考え方になります（参考：[「整然データとは何か」Colorless Green Ideas](https://id.fnshr.info/2017/01/09/tidy-data-intro/))。

**整然データ**の要件は以下の通りです。  
- 一連の値(変数: variables、フィールドとも)が一つの列に収められている  
- 一組のデータセット(レコード)が一つの行に収められている  
- 一つの項目(行・列を構成するセル)には一つの値(文字でも数値でもよい)が収められている  
- 一つの列のデータ型は同一である

これは、内容が正規化され、全体が構造化された表と見なすことができます。

たとえば、以下のようなデータは整然データではありません。

#### 例1

| 遺跡名 | 報告書 | 時代 | 種別 |
| --- | --- | --- | --- |
| A遺跡 | 報告書B | 縄文時代 | 集落 |
| B遺跡 | 報告書C 報告書D | 旧石器時代 古墳時代 | 集落 墳墓 |

ここでは、報告書、時代、種別に複数の値が入っているセルがあります。これは例えば、以下のように整形されるべきです。

| 遺跡名 | 報告書 | 時代 | 種別 |
| --- | --- | --- | --- |
| A遺跡 | 報告書B | 縄文時代 | 集落 |
| B遺跡 | 報告書C | 旧石器時代 | 集落 |
| B遺跡 | 報告書D | 古墳時代 | 墳墓 |

データの2行目と3行目が同一の遺跡を示しているからと言って、値(報告書名、時代、種別)を1つのセルにまとめてはいけません。見やすいからとセル内で改行したり、セルを結合するのもいけません。

#### 例2

|  | 竪穴住居 | 土坑 | 土器 |
| --- | --- | --- | --- |
| A遺跡 | 8 | 12 | 38 |
| B遺跡 |  | 3 | 16 |
| C遺跡 | 25 | 108 | 127 |

クロス集計表はデータ全体の様相を把握するために便利です。しかし整然データの視点から見ると、各遺跡の遺構・遺物の構成を分析するためには以下のように整形すべきです。

| 遺跡名 | 遺構・遺物種別 | 検出数 |
| --- | --- | --- |
| A遺跡 | 竪穴住居 | 8 |
| A遺跡 | 土坑 | 12 |
| A遺跡 | 土器 | 38 |
| B遺跡 | 土坑 | 3 |
| B遺跡 | 土器 | 16 |
| C遺跡 | 竪穴住居 | 25 |
| C遺跡 | 土坑 | 108 |
| C遺跡 | 土器 | 127 |

遺跡名-遺構・遺物種別-検出数で一組のデータセットであり、一行で記述されます。「竪穴住居」「土坑」「土器」などは、遺構・遺物種別を示す**変数**であり分析の対象として扱われます。

他にも、整然データでないものを整然データに整形する手順・方法はいくつもありますが、以下の実習を通して学んでいきたいと思います。

#### 参考文献

* 西原史暁(2017)「整然データとは何か」『情報の科学と技術』67-9: 448-455. ([JSTAGE](https://www.jstage.jst.go.jp/article/jkg/67/9/67_448/_pdf))

## 4.データの取得と確認

リポジトリから東京都遺跡地図全データ(202505現在、島しょ部を除く6456件)のCSVファイルを取得します。

これは[遺跡地図作成実習part1](https://github.com/kotdijian/JASOSR/tree/master/MappingWokrshop)でも取り扱ったものです。

TokyoTotal <- import("https://github.com/kotdijian/JASOSR/raw/master/13Tokyo/13Tokyo\_total.csv", setclass= "tbl\_df", encoding = "UTF-8") # TokyoTotalに原データcsvを読み込み、エンコードの指定に注意

ここではrioパッケージのimport関数を使用しています。他のパッケージの関数と区別するためrio::import()と記述することもできます。

関数名の後ろの()内には、実行条件を示す**引数**(argument)が記述されます。ここではまず”“内にデータ取得元のURLが指定され、次にsetclass="tbl\_df"、すなわち**データフレーム**として読み込むことが宣言され、またencoding = "UTF-8"としてエンコーディングを指定しています。

**データフレーム**とは、数値や文字列などを含む表(行列)です([参照](http://cse.naro.affrc.go.jp/takezawa/r-tips/r/39.html))。1行目は**ラベル**(項目名)です。データの型(数値、文字列など)は各列ごとに異なっていても構いませんが、同一列の中では同じ型でなければなりません。

読み込みが完了するとEnvironmentウインドウに、Data: TokyoTotal | 6456 obs. of 12 variablesと表示されます。これは12項目(列または変数)からなる6456件のデータセットがTokyoTotalに格納されたことを示します。

データフレーム・オブジェクトは、データ量が多いため**Enviroment**ウインドウには内容が表示されません。オブジェクト名の左の青丸+白三角をクリックすると、各列のラベルとデータ型、項目数、先頭数行分のデータが表示されます。さらにオブジェクト名をダブルクリックすると**Source**ウインドウに内容が表示されます。ここでデータが**文字化け**していないかどうか確認しましょう。

データフレーム・オブジェクトの確認は、以下の通り、nrow()関数、str()関数、summary()関数、によっても実行できます。このうちnrow()関数がもっともシンプルなもので、データフレームの行数=レコード数をカウントします。

以下のチャンクをそれぞれ実行してください。

#データフレームのレコード数(行数)を確認  
nrow(TokyoTotal)

## [1] 6456

# データフレームの構造を確認  
str(TokyoTotal)

## tibble [6,456 × 12] (S3: tbl\_df/tbl/data.frame)  
## $ JASID :integer64 [1:6456] 13101000100 13101000200 13101000300 13101000400 13101000500 13101000600 13101000800 13101000900 ...   
## $ LGC : int [1:6456] 13101 13101 13101 13101 13101 13101 13101 13101 13101 13101 ...  
## $ Site\_no : chr [1:6456] "1" "2" "3" "4" ...  
## $ SiteName\_furigana : chr [1:6456] "えどじょうあと" "きゅうほんまるにしかいづか" "" "もみじやまいせき" ...  
## $ SiteName\_original : chr [1:6456] "江戸城跡" "旧本丸西貝塚" "千代田区No.3遺跡" "紅葉山遺跡" ...  
## $ Address : chr [1:6456] "千代田区 千代田　 皇居外苑　 北の丸公園" "千代田区 千代田" "千代田区 千代田" "千代田区 千代田" ...  
## $ Chronology : chr [1:6456] "[近世]" "[縄文時代（前期〜晩期）]" "[弥生時代]" "[縄文前期]" ...  
## $ SiteType : chr [1:6456] "城館" "貝塚" "包蔵地" "包蔵地" ...  
## $ ArchaeologicalFeatures: chr [1:6456] "[近世]建物礎石　敷石遺構　囲状遺構　土留遺構　石組溝　暗渠　城門　城橋　天守台　櫓堀　石塁　石垣　地下室　井戸　上下水道" "" "" "" ...  
## $ MajorArtefacts : chr [1:6456] "縄文土器　弥生土器　土師器　陶磁器　瓦　金属製品　木製品　鉄釘　火箸　簪　笄　チャコ　銭貨" "縄文土器　打斧　石鏃　有溝土錘　骨角器" "弥生土器" "縄文土器" ...  
## $ Lon : num [1:6456] 140 140 140 140 140 ...  
## $ Lat : num [1:6456] 35.7 35.7 35.7 35.7 35.7 ...

#データフレームの先頭数行の内容を表示  
head(TokyoTotal)

## # A tibble: 6 × 12  
## JASID LGC Site\_no SiteName\_furigana SiteName\_original Address Chronology  
## <int64> <int> <chr> <chr> <chr> <chr> <chr>   
## 1 1.e10 13101 1 "えどじょうあと" 江戸城跡 千代田区 千… "[近世]"   
## 2 1.e10 13101 2 "きゅうほんまるにしかいづか"…… 旧本丸西貝塚 千代田区 千… "[縄文時代（前期…  
## 3 1.e10 13101 3 "" 千代田区No.3遺跡 千代田区 千… "[弥生時代]"……  
## 4 1.e10 13101 4 "もみじやまいせき"…… 紅葉山遺跡 千代田区 千… "[縄文前期]"……  
## 5 1.e10 13101 5 "" 千代田区No.5遺跡 千代田区 千… "[奈良時代][平…  
## 6 1.e10 13101 6 "" 千代田区No.6遺跡 千代田区 千… ""   
## # ℹ 5 more variables: SiteType <chr>, ArchaeologicalFeatures <chr>,  
## # MajorArtefacts <chr>, Lon <dbl>, Lat <dbl>

#データフレームの概要・基礎統計量を表示  
summary(TokyoTotal) #数値データ列は、最大最小値、第1・第3四分位値、中央・平均値が計算される

## JASID LGC Site\_no SiteName\_furigana   
## Min. : 1310309502 Min. :13101 Length:6456 Length:6456   
## 1st Qu.:13112015200 1st Qu.:13112 Class :character Class :character   
## Median :13201073600 Median :13201 Mode :character Mode :character   
## Mean :13171712097 Mean :13174   
## 3rd Qu.:13209095200 3rd Qu.:13209   
## Max. :13308005100 Max. :13308   
##   
## SiteName\_original Address Chronology SiteType   
## Length:6456 Length:6456 Length:6456 Length:6456   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
##   
## ArchaeologicalFeatures MajorArtefacts Lon Lat   
## Length:6456 Length:6456 Min. :139.0 Min. :35.50   
## Class :character Class :character 1st Qu.:139.4 1st Qu.:35.62   
## Mode :character Mode :character Median :139.5 Median :35.66   
## Mean :139.5 Mean :35.67   
## 3rd Qu.:139.7 3rd Qu.:35.72   
## Max. :139.9 Max. :35.85   
## NA's :1 NA's :1

## 5.時代別データの抽出

### 5-1.データセットと方針の確認

「東京都遺跡地図」から取得した原データセットでは、一つの「時代」列に、各遺跡の属性項目が列記されています。単一の時代だけの遺跡は一セルに一つのデータですが、多くの遺跡では一セルに複数のデータが含まれています。

以前の授業では、*GoogleSpreadsheet*や*Excel*のフィルター機能を利用して、部分一致で時代名を検索・抽出し、あらたなシートを作成して.csvで保存するという手順で作業をしました。

しかしデータ量が多く、作業の都度、条件指定をしてフィルタリング、コピー・ペーストをするのは手間がかかる上に、手作業ゆえのミスもおきやすいです。そこで、検索や集計を容易にするために、各時代ごとに01ベクトル（その時代の記載があれば1、なければ0）を作成して、TokyoTotalに追加します。

### 5-2. filter関数による抽出

ここでは**dplyr**パッケージのfilter()関数を使用します。**dplyr**は、最初にインストールした**Tidyverse**パッケージに含まれるため、あらたにインストールする必要はありません。なお、filter()関数は同名の関数がほかのパッケージにも含まれているため、環境によっては競合して動作しない場合があります。その際は、dplyr::filter()と記述することで、パッケージを特定して実行することができます。

filter()関数の構文は以下の通りです。

`filter(.data, ...)`  
  
- `data`は、対象とするデータセットをオブジェクト名で指定します。今回は`TokyoTotal`です。  
- `...`はフィルターの条件を指定します。

フィルターを実行した結果を、あらたなオブジェクトに格納して保管することで、原データとフィルター後のデータセットを保持します。以下のチャンクを実行してみましょう。

TokyoPal1 <- filter(TokyoTotal,Chronology == "旧石器時代")  
  
nrow(TokyoPal1)

## [1] 0

結果は0件でした。フィルターの条件設定に使用した==は一致を意味します。TokyoTotalデータセットの**Chronology**列には”旧石器時代”に一致するデータは1件も含まれていないのです。原データセットを見ると、時代はで括られて表記されています。そこで次のチャンクを実行しましょう。

TokyoPal1 <- filter(TokyoTotal,Chronology == "[旧石器時代]")  
  
nrow(TokyoPal1)

## [1] 41

今度は41件が抽出されました。少し少なすぎる気がします。Microsoft ExcelやGoogle Spreadsheetで「“旧石器時代”を含む」でフィルターをかけると683件が抽出されます。この違いは何でしょうか?

繰り返しになりますが、filter(TokyoTotal,Chronology == "[旧石器時代]")では、抽出条件[旧石器時代]を==で指定しているので、完全一致のレコード、つまり旧石器時代単独のだけが抽出されています。41件という数値は、旧石器時代単独遺跡の数を指します。

### 5-3. str\_detect()関数を組み合わせて部分一致検索

そこで、抽出条件にstr\_detect()関数を用います。str\_detect()は**Stringr**パッケージの関数ですで、**Tidyverse**パッケージに含まれています。

次のチャンクを実行してみましょう。

TokyoPal1 <- filter(TokyoTotal, str\_detect(Chronology,"旧石器時代"))  
  
nrow(TokyoPal1)

## [1] 693

今度は693件が抽出され、Spreadsheetによるフィルターと同じ結果を得ることができました。なお、なぜ検索条件が[旧石器時代]ではなく、旧石器時代なのかについては次項で説明します。

### 5-4. 正規表現について

str\_detect()関数では、部分一致、前方一致、後方一致など検索条件を簡略に記号で記述することができます。このような記述方法を**正規表現**と呼びます。**R**以外でも、様々なプログラミング言語で使用可能ですが、言語ごとのローカルルールがある場合もありますので、注意してください。

#### 参考

* [「基本的な正規表現一覧」murashun.jp](https://murashun.jp/blog/20190215-01.html)

先のチャンク8で、検索条件を[旧石器時代]ではなく旧石器時代としたのも、この**正規表現**が関係します。[]は、**含まれる文字列のどれかに一致する**を指定する正規表現なのです。したがって、チャンク8の検索条件を[旧石器時代]にすると次のような結果になってしまいます。

TokyoPal2 <- filter(TokyoTotal, str\_detect(Chronology,"[旧石器時代]"))  
  
nrow(TokyoPal2)

## [1] 5294

5294件という数字は、実は旧石器時代のどれか一文字でも含まれているものを抽出したものです。**東京都遺跡地図**で用いられている時代区分は以下の通りなので、中世、近世を含まない遺跡が抽出されたことになります。

#### 東京都遺跡地図の時代区分一覧

- 旧石器時代   
- 縄文時代   
- 弥生時代   
- 古墳時代   
- 奈良時代   
- 平安時代   
- 中世   
- 近世   
- 時代不明

## 6.時代区分データの作成

### 6-1.時代区分データ作成の意味

チャンク8では、TokyoPal1という新規のオブジェクト(データフレーム)に[旧石器時代]を含むレコードを抽出して格納しました。それでは6280件のレコード全体から、各時代の遺跡数を集計したり、特定の時代(ひとつ、または複数)の遺跡を抽出するにはどうしたらよいでしょうか?

ここまで見てきたやり方を踏襲すると、各時代別の新規のオブジェクトを作成し、それぞれの行数を集計したり、地図作成データにする方法を思いつくかもしれません。

しかし全遺跡が一つのデータセット(データフレーム)となっていることの利便性も捨て難いものがあります。時代別にオブジェクトを分けてしまうと、複数の時代にまたがる操作、分析が煩雑になります。

そこで以下に、TokyoTotalに、旧石器時代～時代不明の9つの**新規の列を追加**します。

#旧石器時代  
Tokyo.palaeolithic <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "旧石器時代")) %>%   
 mutate(P = TRUE)  
  
#縄文時代  
Tokyo.jomon <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "縄文時代")) %>%   
 mutate(J = TRUE)  
  
#弥生時代  
Tokyo.yayoi <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "弥生時代")) %>%   
 mutate(Y = TRUE)  
  
#古墳時代  
Tokyo.kofun <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "古墳時代")) %>%   
 mutate(K = TRUE)  
  
#奈良時代  
Tokyo.nara <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "奈良時代")) %>%   
 mutate(N = TRUE)  
  
#平安時代  
Tokyo.heian <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "平安時代")) %>%   
 mutate(H = TRUE)  
  
#中世  
Tokyo.medieval <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "中世")) %>%   
 mutate(M = TRUE)  
  
#近世  
Tokyo.earlymodern <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "近世")) %>%   
 mutate(E = TRUE)  
  
#時代不明  
Tokyo.unknown <- TokyoTotal %>%   
 filter(str\_detect(Chronology, "不明")) %>%   
 mutate(U = TRUE)

基本はチャンク8と同じですが、新しい関数や表記法が出てきました。以下に解説します。

### 6-2.パイプ演算子

%>%は**パイプ演算子**または単に**パイプ**と呼ばれるものです。繰り返し使用するオブジェクトを何度も入力・記載しなくて済むので便利です。

チャンク10の例について解説します。

TokyoTotal %>% filter(str\_detect(Chronology, "####")) %>% mutate(旧石器時代 = 1)

このコードでは、まず最初にTokyoTotalというオブジェクトを扱うことを宣言しています。次に**パイプ**%>%を用いて宣言を次のfilter()関数に受け渡します。

チャンク8～9のコードは、filter(TokyoTotal, 時代, str\_detect("####")でした。これはTokyoTotalオブジェクトの時代列でfilter()関数を実行するという指定です。チャンク6～7でも同様に、filter()関数のカッコ内の引数の先頭はオブジェクト名です。

一方チャンク10では、filter(の直後に直接str\_detect((時代, "####"))と記述されていて、オブジェクト名が指定されていません。これでも実行できるのは、最初のオブジェクト名の宣言が**パイプ**によって次に受け渡されているからです。同様に、filter()関数の後にも**パイプ**があり、次のmutate()関数にも受け渡されています。

このように**パイプ**は、つながれている関数等の最初の**引数**に、最初の指定を次々に受け渡すという機能を持っています。チャンクの例では、1つの実行あたり2回しか同じオブジェクトを指定していませんが、これが5回、10回と多くなると、繰り返し入力するのは煩雑になりますし、入力ミスなどのチェックも大変です。そこで作業の効率化とコードの簡略化のために**パイプ**を使用するのです。

なお参考までに、チャンク10と同じ内容を**パイプ**なしで記述すると以下の通りとなります。

filter(TokyoTotal, str\_detect(時代, "####"))  
mutate(TokyoTotal, 旧石器時代 = "旧石器")

### 6-3. mutate()関数

チャンク10で新しく出てきた関数mutate()は、データフレームに新規列を作成しデータを追加するものです。

mutate(オブジェクト名, 追加する列名 = 追加するデータ)という構文になります。ただしチャンク10では**パイプ**を用いているのでオブジェクト名が省略されています。##時代 = "##"として、TokyoTotalのデータフレームに##時代という列を追加し、時代ごとに抽出されたレコードに"##の値を与えます。

結果を見てみましょう。なおチャンク10では、原データであるTokyoTotalにダイレクトに時代区分の列を追加せず、いったん、時代別のオブジェクトを作成してそこに格納しています。これは後で時代別データセットとして書き出し、分布図作成などに利用するためです。

例として旧石器時代のデータセットTokyo.Palaeolithicの内容を見てみます。

head(Tokyo.palaeolithic)

## # A tibble: 6 × 13  
## JASID LGC Site\_no SiteName\_furigana SiteName\_original Address Chronology  
## <int64> <int> <chr> <chr> <chr> <chr> <chr>   
## 1 1.e10 13101 22 とうきょうこくりつきんだいびじゅ… 東京国立近代美術館遺跡…… 千代田区 北… [旧石器時代][縄…  
## 2 1.e10 13101 42 きおいちょういせき…… 紀尾井町遺跡 千代田区 紀… [旧石器時代][縄…  
## 3 1.e10 13103 9 ながとはぎはんもうりけやしきあと… 長門萩藩毛利家屋敷跡遺跡…… 港区 赤坂九… [旧石器時代][縄…  
## 4 1.e10 13103 69 きゅうしろがねごりょうちないぶけ… 旧白金御料地内武家屋敷跡遺跡…… 港区 白金台… [旧石器時代][縄…  
## 5 1.e10 13103 75 みただいまちいせき…… 三田台町遺跡 港区 三田四… [旧石器時代][縄…  
## 6 1.e10 13104 2 おちあいいせき 落合遺跡 新宿区 中落… [旧石器時代][縄…  
## # ℹ 6 more variables: SiteType <chr>, ArchaeologicalFeatures <chr>,  
## # MajorArtefacts <chr>, Lon <dbl>, Lat <dbl>, P <lgl>

原データのTokyoTotalは12列(項目)×6282行(レコード)のデータフレームでしたが、新規作成したTokyo.palaeolithicは13列×683行のデータフレームとなっています。一番右に新しい列(P)が加わり**TRUE**が入力されています。他の時代別のデータフレームも同様です。 この新しい列は、**TRUE/FALSE**、つまり真または偽かの2値で示される**論理型(logical)**のデータとなっています。参考までに、QGISでは**Boulean**型と表示されます。

### 6-4.オブジェクトの連結・統合

続いて、当初の方針通りに、全体を統合して一つのデータフレームに格納します。TokyoTotalに書き加えても良いのですが、ここでは原データセットをそのまま保持することとして、新たなTokyoTotal.ageというオブジェクトを作成することとします。

原データを保持しておくことは、ミスが発覚した時に検証したり、作業を巻き戻す上で重要です。一方でオブジェクトが増えすぎると煩雑となり管理が難しくなる側面もあります。このあたりは、状況を検討して最適化を図ってください。

まず新規オブジェクトに加える項目を、select()関数で指定します。次にjoin()関数を用いて、各時代データセットを統合していきます。2つの関数はともに**dplyr**パッケージの関数です。当然**Tidyverse**パッケージに含まれています。

それでは次のチャンクを実行してみましょう。

#統合:TokyoTotal.ageは遺跡名+位置座標+時期区分のみ  
TokyoTotal.age <- TokyoTotal %>%  
 left\_join(dplyr::select(Tokyo.palaeolithic, JASID, P),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.jomon, JASID, J),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.yayoi, JASID, Y),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.kofun,JASID,K),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.nara,JASID,N),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.heian,JASID,H),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.medieval,JASID,M),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.earlymodern,JASID,E),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.unknown,JASID,U),by = "JASID")

オブジェクトTokyoTotalを計10回指定しているので、**パイプ**の有効性が分かりますね。

ここでは、join()関数の一用法である、left\_join()を用いています。以下、チャンク内のコードを解説します。

まず全体に共通するオブジェクトとしてTokyoTotalを指定し、**パイプ**で受け渡します。次にselect()関数で’TokyoTotal’の12項目のうち5項目を選択します。

left\_join()関数は、()内で初め(左側)に指定されたオブジェクトに、次に指定したオブジェクトを結合するものです。ここでは、一連のコードの冒頭で宣言して以下、**パイプ**でつないでいるTokyoTotalが結合先、チャンク10で作成した各時代の一覧が結合データセットになります。ただし両者の項目(列)は大部分重複しているので、ここでもselect()関数で選択したもののみを結合するように指示しています。具体的には、照合のための**JASID**と、各時代の0-1行列が入力される追加された列です。

()内の最後の引数by =は、結合先と結合データをつなげるための**キー**を指定します。ここでは、遺跡のUIDとして設定した**JASID**をキーとしています。すなわち、結合データのうち結合先と**JASID**が一致するものだけが付け加えられていくのです。

この処理を、**パイプ**でつないで、時代ごとに計9回実施します。これにより、時代別に作成された0-1行列、すなわち各遺跡における指定の時代の有無が、全遺跡に対して入力されました。

処理内容の複雑さに対する処理速度の速さを実感できたでしょうか。この一連の処理を、SpreadsheetやExcelで手作業で行なった場合、どれだけの手順を繰り返す必要があるでしょうか?

こうした複雑・大量データ処理は、**R**などによるプログラム処理が効果を発揮する分野です。

なお**filter()はレコード(行)に対して選択・抽出を行なう**のに対して、**select()はラベルで指定した列に対して選択・抽出を行なう**ものです。select()はラベルだけでなく列の順番でも指定可能です。

### 6-5.空白(NA)セルを0に置換する

ところで各時代データセットは、それぞれが帰属する時代の列にのみ1が入力されており、それ以外は**空白**です。

次のチャンクを実行して確認してみましょう。

head(TokyoTotal.age)

## # A tibble: 6 × 21  
## JASID LGC Site\_no SiteName\_furigana SiteName\_original Address Chronology  
## <int64> <int> <chr> <chr> <chr> <chr> <chr>   
## 1 1.e10 13101 1 "えどじょうあと" 江戸城跡 千代田区 千… "[近世]"   
## 2 1.e10 13101 2 "きゅうほんまるにしかいづか"…… 旧本丸西貝塚 千代田区 千… "[縄文時代（前期…  
## 3 1.e10 13101 3 "" 千代田区No.3遺跡 千代田区 千… "[弥生時代]"……  
## 4 1.e10 13101 4 "もみじやまいせき"…… 紅葉山遺跡 千代田区 千… "[縄文前期]"……  
## 5 1.e10 13101 5 "" 千代田区No.5遺跡 千代田区 千… "[奈良時代][平…  
## 6 1.e10 13101 6 "" 千代田区No.6遺跡 千代田区 千… ""   
## # ℹ 14 more variables: SiteType <chr>, ArchaeologicalFeatures <chr>,  
## # MajorArtefacts <chr>, Lon <dbl>, Lat <dbl>, P <lgl>, J <lgl>, Y <lgl>,  
## # K <lgl>, N <lgl>, H <lgl>, M <lgl>, E <lgl>, U <lgl>

多くの場合、私たちはデータの0と**空白**(データ処理上はNA)を区別せず、**空白**のセルは0と扱っているでしょう。しかし前者は、観察・計測等を行なった結果が0であったことを示すのに対し、後者は何らかの理由で観察・計測が行なわれていないか、行なわれたけれどデータが失われていることを示します。

出土遺物全点を検索・検討した結果、石器が1点も含まれていないことを確認した場合、石器の点数は0です。一方、全点の検索・検討が行なわれていない状態で石器の点数が**空白**の場合、石器が**ない(=0)**とは言い切れません。

このように0（または論理型における**FALSE**）と**空白**(NA)は明確に区別する必要があります。

今回の場合、各時代データセットは、他の時代についてのデータを持たないため、そこに含まれないレコードの値はNAになります。しかし実際には、全遺跡について各時代の有無をすべて検索しているので、NAではなくてFALSEとすべきです。これを一括置換するのが、**tidyr**パッケージのreplace\_na()関数です。

# naを一括置換する  
TokyoTotal.age <- replace\_na(TokyoTotal.age, list(P = FALSE, J = FALSE, Y = FALSE, K = FALSE, N = FALSE, H = FALSE, M = FALSE, E = FALSE, U = FALSE))  
  
head(TokyoTotal.age)

## # A tibble: 6 × 21  
## JASID LGC Site\_no SiteName\_furigana SiteName\_original Address Chronology  
## <int64> <int> <chr> <chr> <chr> <chr> <chr>   
## 1 1.e10 13101 1 "えどじょうあと" 江戸城跡 千代田区 千… "[近世]"   
## 2 1.e10 13101 2 "きゅうほんまるにしかいづか"…… 旧本丸西貝塚 千代田区 千… "[縄文時代（前期…  
## 3 1.e10 13101 3 "" 千代田区No.3遺跡 千代田区 千… "[弥生時代]"……  
## 4 1.e10 13101 4 "もみじやまいせき"…… 紅葉山遺跡 千代田区 千… "[縄文前期]"……  
## 5 1.e10 13101 5 "" 千代田区No.5遺跡 千代田区 千… "[奈良時代][平…  
## 6 1.e10 13101 6 "" 千代田区No.6遺跡 千代田区 千… ""   
## # ℹ 14 more variables: SiteType <chr>, ArchaeologicalFeatures <chr>,  
## # MajorArtefacts <chr>, Lon <dbl>, Lat <dbl>, P <lgl>, J <lgl>, Y <lgl>,  
## # K <lgl>, N <lgl>, H <lgl>, M <lgl>, E <lgl>, U <lgl>

ここでは、解説のためselect()、left\_join()の手順とreplace\_na()の手順を別のチャンクに分けていますが、これも**パイプ**でつなぐことが可能です。その場合のコードは次の通りとなります。

TokyoTotal.age <- TokyoTotal %>%  
 left\_join(dplyr::select(Tokyo.palaeolithic, JASID, P),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.jomon, JASID, J),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.yayoi, JASID, Y),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.kofun, JASID, K),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.nara, JASID, N),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.heian, JASID, H),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.medieval, JASID, M),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.earlymodern, JASID, E),by = "JASID") %>%  
 left\_join(dplyr::select(Tokyo.unknown, JASID, U),by = "JASID") %>%  
 replace\_na(list(P = FALSE, J = FALSE, Y = FALSE, K = FALSE, N = FALSE, H = 0, M = FALSE, E = FALSE, U = FALSE))

処理が終わったら、新しく作られたTokyoTotal.ageを開いて内容を確認しましょう。

またTokyo.palaeolithic～Tokyo.unknownには各時代ごとの遺跡一覧が収納されています。こちらも確認してみましょう。

### 6-6. .csvファイルの保存

#### write \_csv()関数による保存

このセクションの最後として、作成したデータを.csv形式で保存し、ウェブGISに読み込めるようにしておきましょう。.csvファイルの書き出しは、**readr**パッケージのwrite\_csv()関数を用います。一般的な(今後の事実上の標準となる)**UTF-8**だけでなく、**ひなたGIS**で読み込める**Shift-JIS**でエンコードした.csvファイルも作成しましょう。

.csvのファイル保存の基本は以下のチャンクの通りです。これを実行すると、現在、この.Rmdファイルを保存・実行しているフォルダに、13Tokyo.csv(原データ)と13TokyoAge.csv(時代0-1行列データ)が保存されます。

#全データ書き出し（UTF-8)  
write.csv(TokyoTotal, "13Tokyo.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(TokyoTotal.age, "13TokyoAge.csv", row.names=FALSE, fileEncoding = "UTF-8")

保存先が分からなくなってしまった場合は、**Console**にgetwd()と入力してください。現在のワーキングディレクトリのパスが表示されます。

保存先のフォルダを指定したい場合は、たとえばwrite.csv(Tokyo.total, "C:/Users/Atsushi Noguchi/Documents/13Tokyo.csv, raw.names = FALSE, fileEncoding = "UTF-8")のように、ファイル名の前にフォルダのパスを入力してください。ここではC:Users/Atsushi Noguchi/Documents/がフォルダのパスで、私のマイクロソフトアカウントの**ドキュメント**フォルダを指定しています。

なおフォルダのパスが分からない時は、Windowsの場合、保存先としたいフォルダを**Shift+マウス右クリック**で表示されるメニューから**パスをコピー**で取得できます。ただしC: Users Atsushi Noguchi Documentsというかたちでコピーされるので、(円マーク)を/(スラッシュ)に変換してください。

ちなみにチャンク2では原データをGitHubリポジトリから読み込んでいますが、ローカルに保存したファイルを読み込ませたい場合は、import("ローカルファイルのパス+ファイル名", setclass = "tbl\_df", encoding = "UTF-8")と書き換えます。encoding =""の指定は、対象により適宜変更してください。自分で作成したファイルをローカルに保存・管理する場合は、この方法で対応できます。

#### 時代別データの保存

時代別データの保存は次のチャンクの通りです。

#時代別データ書き出し  
write.csv(Tokyo.palaeolithic, "13Tokyo\_palaeolithic.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.jomon, "13Tokyo\_jomon.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.yayoi, "13Tokyo\_yayoi.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.kofun, "13Tokyo\_kofun.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.nara, "13Tokyo\_nara.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.heian, "13Tokyo\_heian.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.medieval, "13Tokyo\_medieval.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.earlymodern, "13Tokyo\_earlymodern.csv", row.names=FALSE, fileEncoding = "UTF-8")  
write.csv(Tokyo.unknown, "13Tokyo\_unknown.csv", row.names=FALSE, fileEncoding = "UTF-8")

#### ウェブGIS用データの保存

**ひなたGIS**では座標値のないデータはエラーとなり読み込みをストップします。**Googleマイマップ**などでは[ヌル島](https://ja.wikipedia.org/wiki/%E3%83%8C%E3%83%AB%E5%B3%B6)が発生するので、座標値のないデータを削除して保存する必要があります。

次のチャンクでは、**tidyr**パッケージのdrop\_na()関数を用いて、**経度**が空白のデータを削除したものを保存します。**ひなたGIS**用に**Shift-JIS**で保存しますが、Rでの**Shift-JIS**のエンコーディングの指定はCP932になりますので注意してください。

TokyoTotal.coord <- drop\_na(TokyoTotal,Lon) #経度NAのレコードを除外して新しいオブジェクトに格納  
write.csv(TokyoTotal.coord, "13Tokyo\_totalCoordSJS.csv", row.names = FALSE, fileEncoding = "CP932")　#Shift-JIS書き出し, ファイル名を適宜指定(全レコード)  
  
#時代別データ書き出し  
Tokyo.palaeolithic <- drop\_na(Tokyo.palaeolithic,Lon)  
write.csv(Tokyo.palaeolithic, "13Tokyo\_palaeolithicSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.jomon <- drop\_na(Tokyo.jomon,Lon)  
write.csv(Tokyo.jomon, "13Tokyo\_jomonSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.yayoi<- drop\_na(Tokyo.yayoi,Lon)  
write.csv(Tokyo.yayoi, "13Tokyo\_yayoiSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.kofun <- drop\_na(Tokyo.kofun,Lon)  
write.csv(Tokyo.kofun, "13Tokyo\_kofunSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.nara <- drop\_na(Tokyo.nara,Lon)  
write.csv(Tokyo.nara, "13Tokyo\_naraSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.heian <- drop\_na(Tokyo.heian,Lon)  
write.csv(Tokyo.heian, "13Tokyo\_heianSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.medieval <- drop\_na(Tokyo.medieval,Lon)  
write.csv(Tokyo.medieval, "13Tokyo\_medievalSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.earlymodern <- drop\_na(Tokyo.earlymodern,Lon)  
write.csv(Tokyo.earlymodern, "13Tokyo\_earlymodernSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.unknown <- drop\_na(Tokyo.unknown,Lon)  
write.csv(Tokyo.unknown, "13Tokyo\_unknownSJS.csv", row.names=FALSE, fileEncoding = "CP932")

## 7.集計

### 7-1.自治体別遺跡数の集計

時代別データが整備できたので、遺跡数の集計を行ないます。集計は、**dplyr**パッケージの関数を用いて実行します。

基本的な手順は、group\_by()関数を用いてデータを集約し、tally()関数で合計を計算し、あるいはcount()関数で集計します。計算結果は**n**という新しい列に格納されるので、rename()関数を用いて列名を変更します。この一連の手順を**パイプ**でつなげて一気に実行します。

以下のチャンクを実行すると、区市町村ごとの遺跡数の合計を取得します。

Tokyo.summary1 <- TokyoTotal %>%   
 group\_by(LGC) %>% #自治体コードで集約  
 count %>% #レコード数=遺跡数をカウント  
 arrange(LGC) #自治体コードで昇順に並び替え

**Environment**ウインドウに、Tokyo.summary1が表示されるので、開いて内容を確認してみましょう。

1列目に**LGC(自治体コード)**、2列目に**n(遺跡数合計)**が格納されたデータフレームとなっています。しかし自治体コードだけだとぱっと見、把握しづらいですね。そこで自治体コードと自治体名の対応リストを読み込んで、自治体名を付け加えます。

#区市町村名追加  
 #自治体名の読み込み(more human readable)  
 LGC <- import("https://github.com/kotdijian/JASOSR/raw/master/13Tokyo/LGC\_13Tokyo.csv", setclass= "tbl\_df", encoding ="UTF-8" ) #LGC\_13Tokyo.csv=東京都自治体コードリスト  
  
 #自治体コードでつないで区市町村名を追加  
 Tokyo.summary1 <- Tokyo.summary1 %>%  
 left\_join(dplyr::select(LGC,LGC, Name), by = "LGC") %>%  
 rename(SUM = n) %>%  
 dplyr::select(LGC, Name, SUM) #列名の並べ替え

まず**rio**パッケージのimport()関数で自治体コードと自治体名の対応リスト(LGC\_13Tokyo.csv)を読み込みます。リストの**名称**列を**区市町村名**に変更します(rename()関数)。続いてleft\_join()関数で**区市町村名**を集計リスト(Tokyo.summary1)に追加します。新規の列は常に右に追加されるので、select()関数で並び替えを行ないます。このようにselect()関数では列の抽出だけでなく、引数の配列として列の順序を並び替えることもできます。

###7-2.自治体別-時代別集計

続いて自治体別-時代別の遺跡数を集計します。**自治体コード**と各時代のTRUE-FALSEデータでグループ化し、該当する時代が存在する=各時代列の値がFALSE以外(!=は不一致を示す)でフィルター、集計(count()関数)します。グループ化を解除後、各時代列を削除、left\_join()関数で自治体別集計リスト(Tokyo.summary1)に追加していきます。最後に値がNAのセルを一括で0に置換します。

#集計2:時代別  
　 # 旧石器  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, P)　%>%  
 filter(P != FALSE) %>%  
 count %>%  
 ungroup %>%  
 dplyr::select(-P) %>%  
 rename("P" = "n")  
   
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 縄文  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, J)　%>%  
 filter(J != FALSE) %>%  
 count %>%   
 ungroup %>%  
 dplyr::select(-J) %>%  
 rename("J" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 弥生  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, Y)　%>%  
 filter(Y != FALSE) %>%  
 count %>%   
 ungroup %>%  
 dplyr::select(-Y) %>%  
 rename("Y" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 古墳  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, K)　%>%  
 filter(K != FALSE) %>%  
 count %>%   
 ungroup %>%  
 dplyr::select(-K) %>%  
 rename("K" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 奈良  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, N)　%>%  
 filter(N != FALSE) %>%   
 count %>%  
 ungroup %>%  
 dplyr::select(-N) %>%  
 rename("N" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 平安  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, H)　%>%  
 filter(H != FALSE) %>%  
 count %>%   
 ungroup %>%  
 dplyr::select(-H) %>%  
 rename("H" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 中世  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, M)　%>%  
 filter(M != FALSE) %>%  
 count %>%   
 ungroup %>%   
 dplyr::select(-M) %>%  
 rename("M" = "n")   
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 近世  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, E)　%>%  
 filter(E != FALSE) %>%  
 count %>%   
 ungroup %>%   
 dplyr::select(-E) %>%  
 rename("E" = "n")   
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
  
 # 時代不明  
 Tokyo.summary2 <- TokyoTotal.age %>%   
 group\_by(LGC, U)　%>%  
 filter(U != FALSE) %>%  
 count %>%   
 ungroup %>%  
 dplyr::select(-U) %>%  
 rename("U" = "n")  
 Tokyo.summary1 <- left\_join(Tokyo.summary1, Tokyo.summary2, by = "LGC")  
   
 # NAを0に置換  
 Tokyo.summary1[is.na(Tokyo.summary1)] <- 0  
   
 # 一時保管オブジェクトを削除  
 rm(Tokyo.summary2)

これでTokyo.summary1に自治体別-時代別遺跡数の集計結果が格納されました。 続く2つのチャンクで全域の時代別集計を行い自治体別の集計に結合、また作成したデータをCSVで保存します。

# 時代別集計  
 # 横-縦変換  
 Tokyo.summary2 <- Tokyo.summary1 %>%  
 dplyr::select(-Name, -SUM) %>%  
 pivot\_longer(  
 cols=c(P, J, Y, K, N, H, M, E, U),  
 names\_to = "S",  
 values\_to = "ss"  
 )  
  
 # 集計  
 Tokyo.summary2 <- Tokyo.summary2 %>%  
 group\_by(S) %>%  
 summarise(Sum = sum(ss))  
   
# 縦-横変換  
 Tokyo.summary2 <- Tokyo.summary2 %>%  
 pivot\_wider(  
 names\_from = S,  
 values\_from = Sum  
 ) %>%  
 mutate(LGC = 13000, Name ="東京都", count(TokyoTotal)) %>% #合計行のコードと名称追加  
 rename(SUM = n) %>% #全集計列の名称変更  
 dplyr::select(LGC, Name, SUM, P, J, Y, K, N, H, M, E, U) #整列

# 合計行データの結合  
Tokyo.summary1 <- bind\_rows(Tokyo.summary1, Tokyo.summary2)  
  
# 一時保管オブジェクトを削除  
rm(Tokyo.summary2)

# .csv保存  
write.csv(Tokyo.summary1, "13TokyoSummary1.csv", row.names = FALSE, fileEncoding = "UTF-8")

### 7-3.時代別-遺跡種別の集計

自治体別-時代別の集計と同じ手順で、遺跡種別データを集計します。集計の際に作成するリストは、遺跡分布図作成データとしても利用できるので保管し、後ほど.csvファイルとして書き出します。

なお**主な遺構/概要**という列名には/(半角スラッシュ)が入っているためデータ操作時に問題が生じます。最初に列名を**遺構**に変更しておきます。

#集計2:遺跡種別  
 #ベースリスト  
 Tokyo.sp <- TokyoTotal.age %>%  
 group\_by(LGC) %>%  
 count %>%  
 rename("SUM" = "n")  
   
 # 縄文貝塚  
 #リスト  
 Tokyo.Jkaizuka <- TokyoTotal.age %>%   
 filter(J != FALSE) %>%   
 filter(str\_detect(SiteType, "貝塚"))  
 #集計  
 Tokyo.st2 <- Tokyo.Jkaizuka %>%  
 group\_by(LGC)　%>%  
 count %>%   
 rename("ShellMidden" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
   
 # 方形周溝墓  
 #リスト  
 Tokyo.Yshukoubo <- TokyoTotal.age %>%   
 filter(Y != FALSE) %>%   
 filter(str\_detect(SiteType, "方形周溝墓"))  
 #集計  
 Tokyo.st2 <- Tokyo.Yshukoubo %>%   
 group\_by(LGC) %>%   
 count %>%   
 rename("HShukobo" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
   
 # 古墳  
 #リスト  
 Tokyo.Kkofun <- TokyoTotal.age %>%   
 filter(K != FALSE) %>%   
 filter(str\_detect(SiteType, "古墳"))  
 #集計  
 Tokyo.st2 <- Tokyo.Kkofun %>%   
 group\_by(LGC) %>%   
 count() %>%   
 rename("Kofun" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
   
 # 横穴墓  
 #リスト  
 Tokyo.Kyokoana <- TokyoTotal.age %>%   
 filter(K != FALSE) %>%   
 filter(str\_detect(SiteType, "横穴墓"))  
 #集計  
 Tokyo.st2 <- Tokyo.Kyokoana %>%   
 group\_by(LGC) %>%   
 count %>%   
 rename("Yokoanabo" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
   
 # 中世城館  
 #リスト  
 Tokyo.Mjokan <- TokyoTotal.age %>%   
 filter(M != FALSE) %>%   
 filter(str\_detect(SiteType, "城館"))  
 #集計  
 Tokyo.st2 <- Tokyo.Mjokan %>%   
 group\_by(LGC) %>%   
 count %>%   
 rename("Jokan" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
   
 # 塚  
 #リスト  
 Tokyo.Mtsuka <- TokyoTotal.age %>%   
 filter(str\_detect(SiteType, "M"))  
 #集計  
 Tokyo.st2 <- Tokyo.Mtsuka %>%   
 group\_by(LGC) %>%   
 count %>%   
 rename("Mound" = "n")  
 Tokyo.sp <- left\_join(Tokyo.sp, Tokyo.st2, by = "LGC")  
  
 # NAを0に置換  
 Tokyo.sp[is.na(Tokyo.sp)] <- 0  
   
 #自治体コードでつないで区市町村名を追加  
 Tokyo.sp <- Tokyo.sp %>%  
 left\_join(dplyr::select(LGC,LGC, Name), by = "LGC") %>%  
 dplyr::select(LGC, Name, SUM, ShellMidden, HShukobo, Kofun, Yokoanabo, Jokan, Mound) #列名の並べ替え  
   
 # 一時データの削除  
 rm(Tokyo.st2)

なお時代別データと同様、left\_join()関数で抽出した遺跡・遺構種別データを原データTokyoTotalに追加しておくと、後からの検索や集計が容易になる。その手順・方法はチャンク12と基本的に同じです。

## 8.ウェブGISデータの書き出し

先にチャンク15では、時代別リストをそのままShift-JISの.csvファイルに書き出しました。

ここでは遺跡・遺構種別のデータを**ひなたGIS**に読み込み表示させる際のマーカーの色を指定してから保存します。色を指定する**色**列とデータの追加は、mutate()関数を用います。

#遺跡・遺構種別データ書き出し  
Tokyo.Jkaizuka <- Tokyo.Jkaizuka %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "blue")  
write.csv(Tokyo.Jkaizuka, "13Tokyo\_kaizukaSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.Yshukoubo <- Tokyo.Yshukoubo %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "orange")  
write.csv(Tokyo.Yshukoubo, "13Tokyo\_shukoboSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.Kkofun<- Tokyo.Kkofun %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "green")  
write.csv(Tokyo.Kkofun, "13Tokyo\_kofunSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.Kyokoana <- Tokyo.Kyokoana %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "brown")  
write.csv(Tokyo.Kyokoana, "13Tokyo\_yokoanaSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.Mjokan <- Tokyo.Mjokan %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "red")  
write.csv(Tokyo.Mjokan, "13Tokyo\_jokanSJS.csv", row.names=FALSE, fileEncoding = "CP932")  
  
Tokyo.Mtsuka <- Tokyo.Mtsuka %>%  
 drop\_na(Lon) %>%  
 mutate(色 = "purple")  
write.csv(Tokyo.Mtsuka, "13Tokyo\_tsukaSJS.csv", row.names=FALSE, fileEncoding = "CP932")

## おわりに

東京都遺跡地図データを用いた実習は以上です。今回は大規模なデータセットを整形・編集するために有効な**dplyr**パッケージの主要な関数について実習を通して学びました。ここで取り上げなかった関数や機能もあります。参考文献やインターネット上の情報を活用してみてください。

**R**を使ったデータ編集の最大の利点は、自動化、効率化と同時に、手順をコードとして記録しておくことで、過程を検証したり、再現することが可能になることにもあります。SpreadsheetやExcelでの作業時に、以前の手順が分からなくなったり、そのため何故そのようなデータが生じているのかが分からなくなったりした経験はないでしょうか?

このRマークダウン文書にあるようなプログラムの実行コード(ソースコード)を書くことは、知識・経験が必要であり、実際にはSpreadsheetやExcel上で手作業を行なうのと手間は大きく変わらないかもしれません。しかしコードを書き、記録しておくことで検証・再現が可能になると同時に、同じような作業はコードを再利用することで効率化できる場合もあります。作業が完了したデータリストのエンコーディングを指定・変換して保存することを繰り返すような処理は、それこそ手作業ではなく機械化(プログラム化)することで、間違いを減らせるとともに、他の作業に当てられる時間を確保できるようにもなるでしょう。

さらにコードを公開・共有することで、自分ひとりで作業するよりも効率化を推進することが可能になるでしょう。

そのような観点から、これからも積極的に**R**を利用してみてください。