JAS_DataManagementFor13Tokyo-20201017ver

kotdijian

2020年10月17日

0.はじめに

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。

なお本資料は、東京学芸大学2020年度春学期開講「地域考古学B」課外補講のために作成したものをベースに、「考古学・文化財のためのデータサイエンス・サロン」向けに改変しました。

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("formattable")) install.packages('formattable', 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("formattable")  #表の表示
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

	竪穴住居	土坑	土器
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(20201017現在、島しょ部を除く6280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 | 6280 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6282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] 6291
```

```
# データフレームの構造を確認
str(TokyoTotal)
```

```
## tibble [6,291 x 12] (S3: tbl_df/tbl/data.frame)
##  $ JASID        :integer64 [1:6291] 13101000100 13101000200 13101000300 13101000400 13101000500 13101000600 1
3101000800 13101000900 ...
##  $ 自治体コード : int [1:6291] 13101 13101 13101 13101 13101 13101 13101 13101 13101 13101 ...
##  $ 遺跡番号     : chr [1:6291] "1" "2" "3" "4" ...
##  $ ふりがな     : chr [1:6291] "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" ...
##  $ 遺跡名       : chr [1:6291] "江戸城跡" "旧本丸西貝塚" "千代田区No.3遺跡" "紅葉山遺跡" ...
##  $ 所在地       : chr [1:6291] "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" ...
##  $ 時代         : chr [1:6291] "[近世]" "[縄文時代（前期～晩期）]" "[弥生時代]" "[縄文前期]" ...
##  $ 種別         : chr [1:6291] "城館" "貝塚" "包蔵地" "包蔵地" ...
##  $ 主な遺構/概要: chr [1:6291] "[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" "" "" "" ...
##  $ 主な出土品   : chr [1:6291] "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" ...
##  $ 経度         : num [1:6291] 140 140 140 140 140 ...
##  $ 緯度         : num [1:6291] 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 x 12
##   JASID 自治体コード 遺跡番号 ふりがな 遺跡名 所在地 時代  種別  `主な遺構/概要`
##   <int64>       <int> <chr>    <chr>    <chr> <chr> <chr> <chr> <chr>
## 1 1310~        13101 1        "えどじょうあ~ 江戸城跡~ 千代田区 ~ "[近世~ 城館  "[近世]建物礎石　敷石遺構~
## 2 1310~        13101 2        "きゅうほんま~ 旧本丸西貝~ 千代田区 ~ "[縄文~ 貝塚  ""
## 3 1310~        13101 3        ""       千代田区N~ 千代田区 ~ "[弥生~ 包蔵地~ ""
## 4 1310~        13101 4        "もみじやまい~ 紅葉山遺跡~ 千代田区 ~ "[縄文~ 包蔵地~ ""
## 5 1310~        13101 5        ""       千代田区N~ 千代田区 ~ "[奈良~ 包蔵地~ ""
## 6 1310~        13101 6        ""       千代田区N~ 千代田区 ~ ""    貝塚  ""
## # ... with 3 more variables: 主な出土品 <chr>, 経度 <dbl>, 緯度 <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              自治体コード     遺跡番号           ふりがな
##  Min.    :  1310309502   Min.    :13101   Length:6291        Length:6291
##  1st Qu.: 13112027400   1st Qu.:13112   Class :character   Class :character
##  Median : 13201080200   Median :13201   Mode  :character   Mode  :character
##  Mean    : 13192168045   Mean    :13175
##  3rd Qu.: 13209099100   3rd Qu.:13209
##  Max.    :133080001000   Max.    :13308
##     遺跡名              所在地              時代               種別
##  Length:6291        Length:6291        Length:6291        Length:6291
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  主な遺構/概要       主な出土品            経度            緯度
##  Length:6291        Length:6291        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
```

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。

遺跡地図作成実習 #1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,時代 == "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**時代**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,時代 == "[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。GoogleSpreadsheet上で「"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,時代 == "[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(時代,"旧石器時代"))

nrow(TokyoPal1)
```

```
## [1] 686
```

今度は686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(時代,"[旧石器時代]"))

nrow(TokyoPal2)
```

```
## [1] 5261
```

5261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(時代, "旧石器時代")) %>%
                      mutate(旧石器時代 = "旧石器")

#縄文時代
Tokyo.jomon <- TokyoTotal %>%
                      filter(str_detect(時代, "縄文時代")) %>%
                      mutate(縄文時代 = "縄文")

#弥生時代
Tokyo.yayoi <- TokyoTotal %>%
                      filter(str_detect(時代, "弥生時代")) %>%
                      mutate(弥生時代 = "弥生")

#古墳時代
Tokyo.kofun <- TokyoTotal %>%
                      filter(str_detect(時代, "古墳時代")) %>%
                      mutate(古墳時代 = "古墳")

#奈良時代
Tokyo.nara <- TokyoTotal %>%
                      filter(str_detect(時代, "奈良時代")) %>%
                      mutate(奈良時代 = "奈良")

#平安時代
Tokyo.heian <- TokyoTotal %>%
                      filter(str_detect(時代, "平安時代")) %>%
                      mutate(平安時代 = "平安")

#中世
Tokyo.medieval <- TokyoTotal %>%
                      filter(str_detect(時代, "中世")) %>%
                      mutate(中世 = "中世")

#近世
Tokyo.earlymodern <- TokyoTotal %>%
                      filter(str_detect(時代, "近世")) %>%
                      mutate(近世 = "近世")

#時代不明
Tokyo.unknown <- TokyoTotal %>%
                      filter(str_detect(時代, "不明")) %>%
                      mutate(時代不明 = "不明")
```

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(時代, "####")) %>% 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 x 13
##   JASID 自治体コード 遺跡番号 ふりがな 遺跡名 所在地 時代  種別  `主な遺構/概要`
##   <int>       <int> <chr>    <chr>    <chr> <chr> <chr> <chr> <chr>
## 1 1310~       13101 22       "とうきょうこ~ 東京国立近~ 千代田区 ~ [旧石器~ 集落・城~ [旧石器時代]礫群  [縄
文時~
## 2 1310~       13101 42       "きおいちょう~ 紀尾井町遺~ 千代田区 ~ [旧石器~ 包蔵地・~ 土坑  盛土  土手  石組
溝  土~
## 3 1310~       13103 69       ""       港区No.~ 港区 白金~ [旧石器~ 武家屋敷~ [近世]建物  礎石  土坑  地~
## 4 1310~       13103 75       "みただいまち~ 三田台町遺~ 港区 三田~ [旧石器~ 集落  [弥生時代]?  住居  [古墳~
## 5 1310~       13104 2        "おちあいいせ~ 落合遺跡~ 新宿区 中~ [旧石器~ 包蔵地・~ [旧石器時代]礫群  [縄文
時~
## 6 1310~       13104 10       "とやまがはら~ 戸山ヶ原上~ 新宿区 百~ [旧石器~ 包蔵地・~ [旧石器時代]集中地点
[縄~
## # ... with 4 more variables: 主な出土品 <chr>, 経度 <dbl>, 緯度 <dbl>,
## #   旧石器時代 <chr>
```

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(旧石器時代）が加わり**旧石器**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。

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, 旧石器時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.jomon, JASID, 縄文時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.yayoi, JASID, 弥生時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.kofun,JASID,古墳時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.nara,JASID,奈良時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.heian,JASID,平安時代),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.medieval,JASID,中世),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.earlymodern,JASID,近世),by = "JASID") %>%
                      left_join(dplyr::select(Tokyo.unknown,JASID,時代不明),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 x 21
##   JASID 自治体コード 遺跡番号 ふりがな 遺跡名 所在地 時代  種別  `主な遺構/概要`
##   <int>        <int> <chr>    <chr>    <chr>  <chr>  <chr> <chr> <chr>
## 1 1310~        13101 1        "えどじょうあ~ 江戸城跡~ 千代田区 ~ "[近世~ 城館  "[近世]建物礎石  敷石遺構~
## 2 1310~        13101 2        "きゅうほんま~ 旧本丸西貝~ 千代田区 ~ "[縄文~ 貝塚  ""
## 3 1310~        13101 3        ""       千代田区N~ 千代田区 ~ "[弥生~ 包蔵地~ ""
## 4 1310~        13101 4        "もみじやまい~ 紅葉山遺跡~ 千代田区 ~ "[縄文~ 包蔵地~ ""
## 5 1310~        13101 5        ""       千代田区N~ 千代田区 ~ "[奈良~ 包蔵地~ ""
## 6 1310~        13101 6        ""       千代田区N~ 千代田区 ~ ""    貝塚  ""
## # ... with 12 more variables: 主な出土品 <chr>, 経度 <dbl>, 緯度 <dbl>,
## #   旧石器時代 <chr>, 縄文時代 <chr>, 弥生時代 <chr>, 古墳時代 <chr>,
## #   奈良時代 <chr>, 平安時代 <chr>, 中世 <chr>, 近世 <chr>, 時代不明 <chr>
```

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 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 0 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(旧石器時代 =0, 縄文時代 =0, 弥生時代 = 0, 古墳時代 = 0, 奈良時代 = 0, 平安時代 = 0, 中世 = 0, 近世 = 0, 時代不明 = 0))

head(TokyoTotal.age)
```

```
## # A tibble: 6 x 21
##   JASID 自治体コード 遺跡番号 ふりがな 遺跡名 所在地 時代  種別  `主な遺構/概要`
##   <int>        <int> <chr>    <chr>    <chr>  <chr>  <chr> <chr> <chr>
## 1 1310~        13101 1        "えどじょうあ~ 江戸城跡~ 千代田区 ~ "[近世~ 城館  "[近世]建物礎石  敷石遺構~
## 2 1310~        13101 2        "きゅうほんま~ 旧本丸西貝~ 千代田区 ~ "[縄文~ 貝塚  ""
## 3 1310~        13101 3        ""       千代田区N~ 千代田区 ~ "[弥生~ 包蔵地~ ""
## 4 1310~        13101 4        "もみじやまい~ 紅葉山遺跡~ 千代田区 ~ "[縄文~ 包蔵地~ ""
## 5 1310~        13101 5        ""       千代田区N~ 千代田区 ~ "[奈良~ 包蔵地~ ""
## 6 1310~        13101 6        ""       千代田区N~ 千代田区 ~ ""    貝塚  ""
## # ... with 12 more variables: 主な出土品 <chr>, 経度 <dbl>, 緯度 <dbl>,
## #   旧石器時代 <chr>, 縄文時代 <chr>, 弥生時代 <chr>, 古墳時代 <chr>,
## #   奈良時代 <chr>, 平安時代 <chr>, 中世 <chr>, 近世 <chr>, 時代不明 <chr>
```

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, 旧石器時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.jomon, JASID, 縄文時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.yayoi, JASID, 弥生時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.kofun, JASID, 古墳時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.nara, JASID, 奈良時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.heian, JASID, 平安時代),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.medieval, JASID, 中世),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.earlymodern, JASID, 近世),by = "JASID") %>%
                        left_join(dplyr::select(Tokyo.unknown, JASID, 時代不明),by = "JASID") %>%
                        replace_na(list(旧石器時代 =0, 縄文時代 =0, 弥生時代 = 0, 古墳時代 = 0, 奈良時代 = 0,
 平安時代 = 0, 中世 = 0, 近世 = 0, 時代不明 = 0))
```

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,経度) #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,経度)
write.csv(Tokyo.palaeolithic, "13Tokyo_palaeolithicSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.jomon <- drop_na(Tokyo.jomon,経度)
write.csv(Tokyo.jomon, "13Tokyo_jomonSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.yayoi<- drop_na(Tokyo.yayoi,経度)
write.csv(Tokyo.yayoi, "13Tokyo_yayoiSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.kofun <- drop_na(Tokyo.kofun,経度)
write.csv(Tokyo.kofun, "13Tokyo_kofunSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.nara <- drop_na(Tokyo.nara,経度)
write.csv(Tokyo.nara, "13Tokyo_naraSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.heian <- drop_na(Tokyo.heian,経度)
write.csv(Tokyo.heian, "13Tokyo_heianSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.medieval <- drop_na(Tokyo.medieval,経度)
write.csv(Tokyo.medieval, "13Tokyo_medievalSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.earlymodern <- drop_na(Tokyo.earlymodern,経度)
write.csv(Tokyo.earlymodern, "13Tokyo_earlymodernSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.unknown <- drop_na(Tokyo.unknown,経度)
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(自治体コード) %>%  #自治体コードで集約
  count %>%                   #レコード数=遺跡数をカウント
  rename(遺跡数合計 = n) %>%  #集計結果が格納される列の名前をn→遺跡数合計に変更
  arrange(自治体コード)       #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**自治体コード**、2列目に**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", en
coding ="UTF-8" ) #LGC_13Tokyo.csv=東京都自治体コードリスト
  LGC <- rename(LGC,区市町村名 = 名称)

  #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,自治体コード,区市町村名), by = "自治体コード") %>%
                          dplyr::select(自治体コード, 区市町村名, 遺跡数合計) #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0-1行列でグループ化し、該当する時代が存在する=各時代列の値が0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、集計結果を格納する**n**列のラベルを時代名に変更(`rename()` 関数)、 `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(自治体コード, 旧石器時代)  %>%
    filter(旧石器時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("旧石器" = "n") %>%
    dplyr::select(-旧石器時代)

  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 縄文
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 縄文時代)  %>%
    filter(縄文時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("縄文" = "n") %>%
    dplyr::select(-縄文時代)
  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 弥生
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 弥生時代)  %>%
    filter(弥生時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("弥生" = "n") %>%
    dplyr::select(-弥生時代)
  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 古墳
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 古墳時代)  %>%
    filter(古墳時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("古墳" = "n") %>%
    dplyr::select(-古墳時代)
  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 奈良
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 奈良時代)  %>%
    filter(奈良時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("奈良" = "n") %>%
    dplyr::select(-奈良時代)
  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 平安
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 平安時代)  %>%
    filter(平安時代 != 0) %>%
    count %>%
    ungroup %>%
    rename("平安" = "n") %>%
    dplyr::select(-平安時代)
  Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

  # 中世
  Tokyo.summary2 <- TokyoTotal.age %>%
    group_by(自治体コード, 中世)  %>%
    filter(中世 != 0) %>%
    count %>%
```

```
  ungroup %>%
  dplyr::select(-中世) %>%
  rename("中世" = "n")
Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

# 近世
Tokyo.summary2 <- TokyoTotal.age %>%
  group_by(自治体コード, 近世)  %>%
  filter(近世 != 0) %>%
  count %>%
  ungroup %>%
  dplyr::select(-近世) %>%
  rename("近世" = "n")
Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

# 時代不明
Tokyo.summary2 <- TokyoTotal.age %>%
  group_by(自治体コード, 時代不明)  %>%
  filter(時代不明 != 0) %>%
  count %>%
  ungroup %>%
  dplyr::select(-時代不明) %>%
  rename("不明" = "n")
Tokyo.summary1 <- left_join(Tokyo.summary1, Tokyo.summary2, by = "自治体コード")

# 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。 次のチャンクで.csv形式で保存するとともに、**formattable**パッケージを用いて表を表示します。

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

# formattableによる表の表示
formattable(Tokyo.summary1)
```

自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
13101	千代田区	89	2	19	12	5	3	6	7	77	0
13102	中央区	152	0	2	1	1	1	1	3	150	0
13103	港区	243	2	37	14	30	7	7	8	185	0
13104	新宿区	151	28	66	23	23	18	16	13	136	0
13105	文京区	131	19	72	33	22	17	17	14	115	0
13106	台東区	154	4	27	11	31	45	44	22	146	0
13107	墨田区	79	0	1	0	1	0	0	0	78	0
13108	江東区	6	0	0	0	0	0	0	0	6	0
13109	品川区	30	0	16	3	8	4	2	4	7	0
13110	目黒区	58	13	47	10	5	1	6	12	18	1
13111	大田区	235	3	32	22	120	78	26	39	6	16
13112	世田谷区	289	43	134	29	161	36	34	65	83	2
13113	渋谷区	78	6	41	7	34	17	4	2	9	4
13114	中野区	94	6	67	15	17	11	6	3	7	4

自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
13115	杉並区	164	34	135	25	58	20	16	12	35	0
13116	豊島区	16	1	12	4	5	1	4	4	12	0
13117	北区	54	5	21	23	37	22	21	16	16	0
13118	荒川区	7	1	3	2	0	2	2	3	5	0
13119	板橋区	173	27	112	88	84	28	35	21	29	31
13120	練馬区	106	52	93	27	15	12	15	15	21	0
13121	足立区	27	0	2	2	19	8	7	8	8	0
13122	葛飾区	27	0	1	4	15	13	12	15	11	1
13123	江戸川区	13	0	1	3	2	2	2	6	2	4
13201	八王子市	996	100	760	88	192	358	634	235	448	7
13202	立川市	20	8	13	0	3	6	7	4	4	0
13203	武蔵野市	5	3	3	0	0	2	3	0	0	0
13204	三鷹市	55	27	48	3	17	21	15	8	36	0
13205	青梅市	181	8	138	8	21	55	66	28	14	5
13206	府中市	58	11	19	2	30	13	13	10	8	4
13207	昭島市	46	4	24	0	6	11	20	4	3	1
13208	調布市	90	19	46	7	60	37	27	22	24	0
13209	町田市	912	56	738	102	178	511	565	135	102	14
13210	小金井市	24	17	22	0	4	0	1	5	5	1
13211	小平市	4	3	1	0	0	1	1	0	1	0
13212	日野市	52	5	24	3	24	20	24	23	6	1
13213	東村山市	148	22	120	13	49	49	52	54	4	3
13214	国分寺市	46	19	36	1	0	25	23	6	1	0
13215	国立市	54	5	14	0	34	8	18	9	10	0
13218	福生市	19	0	16	0	1	0	4	2	1	1
13219	狛江市	72	1	22	15	64	32	25	10	31	1
13220	東大和市	76	18	68	0	0	5	39	2	12	0
13221	清瀬市	77	7	47	3	3	3	57	5	2	0
13222	東久留米市	138	40	131	4	0	1	9	0	2	0
13223	武蔵村山市	40	6	33	4	15	1	9	1	3	0
13224	多摩市	311	29	228	8	35	110	156	78	112	3
13225	稲城市	146	15	112	7	23	55	60	38	35	6
13227	羽村市	10	0	8	0	2	1	1	1	1	0
13228	あきる野市	183	3	89	6	107	15	27	23	15	0
13229	西東京市	14	7	12	2	1	0	3	1	6	1
13303	瑞穂町	22	5	16	2	2	3	2	5	5	0
13305	日の出町	25	1	14	2	9	5	6	8	8	0
13307	檜原村	42	1	38	0	0	0	0	2	0	3

自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
13308	奥多摩町	49	0	47	0	0	1	12	1	0	0

formattableパッケージではオプションで表にバーグラフを追加することができます。ただしバーの幅は列ごとの相対表示であり、列間の比較はできません。

```
# formattableによる表の表示の拡張
formattable(Tokyo.summary1,list(合計=color_bar("tomato"),
                                旧石器=color_bar("blue"),
                                縄文=color_bar("brown"),
                                弥生=color_bar("orange"),
                                古墳=color_bar("green"),
                                奈良=color_bar("purple"),
                                平安=color_bar("yellow"),
                                中世=color_bar("pink"),
                                近世=color_bar("skyblue"),
                                時代不明=color_bar("lightgrey")))
```

自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
13101	千代田区	89	2	19	12	5	3	6	7	77	0
13102	中央区	152	0	2	1	1	1	1	3	150	0
13103	港区	243	2	37	14	30	7	7	8	185	0
13104	新宿区	151	28	66	23	23	18	16	13	136	0
13105	文京区	131	19	72	33	22	17	17	14	115	0
13106	台東区	154	4	27	11	31	45	44	22	146	0
13107	墨田区	79	0	1	0	1	0	0	0	78	0
13108	江東区	6	0	0	0	0	0	0	0	6	0
13109	品川区	30	0	16	3	8	4	2	4	7	0
13110	目黒区	58	13	47	10	5	1	6	12	18	1
13111	大田区	235	3	32	22	120	78	26	39	6	16
13112	世田谷区	289	43	134	29	161	36	34	65	83	2
13113	渋谷区	78	6	41	7	34	17	4	2	9	4
13114	中野区	94	6	67	15	17	11	6	3	7	4
13115	杉並区	164	34	135	25	58	20	16	12	35	0
13116	豊島区	16	1	12	4	5	1	4	4	12	0
13117	北区	54	5	21	23	37	22	21	16	16	0
13118	荒川区	7	1	3	2	0	2	2	3	5	0
13119	板橋区	173	27	112	88	84	28	35	21	29	31
13120	練馬区	106	52	93	27	15	12	15	15	21	0
13121	足立区	27	0	2	2	19	8	7	8	8	0
13122	葛飾区	27	0	1	4	15	13	12	15	11	1
13123	江戸川区	13	0	1	3	2	2	2	6	2	4
13201	八王子市	996	100	760	88	192	358	634	235	448	7
13202	立川市	20	8	13	0	3	6	7	4	4	0
13203	武蔵野市	5	3	3	0	0	2	3	0	0	0

自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
13204	三鷹市	55	27	48	3	17	21	15	8	36	0
13205	青梅市	181	8	138	8	21	55	66	28	14	5
13206	府中市	58	11	19	2	30	13	13	10	8	4
13207	昭島市	46	4	24	0	6	11	20	4	3	1
13208	調布市	90	19	46	7	60	37	27	22	24	0
13209	町田市	912	56	738	102	178	511	565	135	102	14
13210	小金井市	24	17	22	0	4	0	1	5	5	1
13211	小平市	4	3	1	0	0	1	1	0	1	0
13212	日野市	52	5	24	3	24	20	24	23	6	1
13213	東村山市	148	22	120	13	49	49	52	54	4	3
13214	国分寺市	46	19	36	1	0	25	23	6	1	0
13215	国立市	54	5	14	0	34	8	18	9	10	0
13218	福生市	19	0	16	0	1	0	4	2	1	1
13219	狛江市	72	1	22	15	64	32	25	10	31	1
13220	東大和市	76	18	68	0	0	5	39	2	12	0
13221	清瀬市	77	7	47	3	3	3	57	5	2	0
13222	東久留米市	138	40	131	4	0	1	9	0	2	0
13223	武蔵村山市	40	6	33	4	15	1	9	1	3	0
13224	多摩市	311	29	228	8	35	110	156	78	112	3
13225	稲城市	146	15	112	7	23	55	60	38	35	6
13227	羽村市	10	0	8	0	2	1	1	1	1	0
13228	あきる野市	183	3	89	6	107	15	27	23	15	0
13229	西東京市	14	7	12	2	1	0	3	1	6	1
13303	瑞穂町	22	5	16	2	2	3	2	5	5	0
13305	日の出町	25	1	14	2	9	5	6	8	8	0
13307	檜原村	42	1	38	0	0	0	0	2	0	3
13308	奥多摩町	49	0	47	0	0	1	12	1	0	0

7-3.河川水系別に並べ替え

自治体コードによる順列では地理的な感覚が捉えづらいので、河川水系別のデータを追加して順列を変更します。ただし各自治体は一つの水系にだけ含まれるわけではなく、ひとつの水系で代表させています。あくまで便宜的な区分ということで理解してください。なお同一水系内では、下流→上流と自治体の順列を与えています。

以下のチャンクを実行すると、まずGitHubリポジトリから `13Tokyo_river.csv` を読み込みます。続いて**自治体コード**でつないで**水系**データを `Tokyo.summary1` に追加しますが、一対一対応による左側結合ではなく、一対多対応による `inner_join()` 関数を用います。

```
  # 水系リストを読み込み・結合
  river <- import("https://github.com/kotdijian/JASOSR/raw/master/13Tokyo/13Tokyo_river.csv", setclass = "tbl_df", encoding="UTF-8")
  Tokyo.sumRiver <- Tokyo.summary1 %>%
    inner_join(river, by = "自治体コード") %>%
    dplyr::select("水系", "配列", "自治体コード", "区市町村名", "遺跡数合計", "旧石器", "縄文", "弥生", "古墳", "奈良", "平安", "中世", "近世", "不明")


# 表形式で表示
  #河川水系でソート
  Tokyo.sumRiver <- arrange(Tokyo.sumRiver, 配列)

  #formattableで表出力
  formattable(Tokyo.sumRiver,list(合計=color_bar("tomato"),
                            旧石器=color_bar("blue"),
                            縄文=color_bar("brown"),
                            弥生=color_bar("orange"),
                            古墳=color_bar("green"),
                            奈良=color_bar("purple"),
                            平安=color_bar("yellow"),
                            中世=color_bar("pink"),
                            近世=color_bar("skyblue"),
                            時代不明=color_bar("lightgrey")))
```

水系	配列	自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
多摩川	101	13111	大田区	235	3	32	22	120	78	26	39	6	16
多摩川	102	13112	世田谷区	289	43	134	29	161	36	34	65	83	2
多摩川	103	13219	狛江市	72	1	22	15	64	32	25	10	31	1
多摩川	104	13208	調布市	90	19	46	7	60	37	27	22	24	0
多摩川	105	13206	府中市	58	11	19	2	30	13	13	10	8	4
多摩川	106	13215	国立市	54	5	14	0	34	8	18	9	10	0
多摩川	107	13202	立川市	20	8	13	0	3	6	7	4	4	0
多摩川	108	13207	昭島市	46	4	24	0	6	11	20	4	3	1
多摩川	109	13218	福生市	19	0	16	0	1	0	4	2	1	1
多摩川	110	13227	羽村市	10	0	8	0	2	1	1	1	1	0
多摩川	111	13205	青梅市	181	8	138	8	21	55	66	28	14	5
多摩川	112	13308	奥多摩町	49	0	47	0	0	1	12	1	0	0
野川	121	13204	三鷹市	55	27	48	3	17	21	15	8	36	0
野川	122	13210	小金井市	24	17	22	0	4	0	1	5	5	1
野川	123	13214	国分寺市	46	19	36	1	0	25	23	6	1	0
三沢川	131	13225	稲城市	146	15	112	7	23	55	60	38	35	6
乞田川	141	13224	多摩市	311	29	228	8	35	110	156	78	112	3
残堀川	151	13303	瑞穂町	22	5	16	2	2	3	2	5	5	0
浅川	161	13212	日野市	52	5	24	3	24	20	24	23	6	1
浅川	162	13201	八王子市	996	100	760	88	192	358	634	235	448	7
秋川	171	13228	あきる野市	183	3	89	6	107	15	27	23	15	0
秋川	172	13307	檜原村	42	1	38	0	0	0	0	2	0	3
平井川	181	13305	日の出町	25	1	14	2	9	5	6	8	8	0

水系	配列	自治体コード	区市町村名	遺跡数合計	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
目黒川	201	13109	品川区	30	0	16	3	8	4	2	4	7	0
目黒川	202	13110	目黒区	58	13	47	10	5	1	6	12	18	1
渋谷川	211	13103	港区	243	2	37	14	30	7	7	8	185	0
渋谷川	212	13113	渋谷区	78	6	41	7	34	17	4	2	9	4
神田川	301	13101	千代田区	89	2	19	12	5	3	6	7	77	0
神田川	302	13104	新宿区	151	28	66	23	23	18	16	13	136	0
神田川	303	13115	杉並区	164	34	135	25	58	20	16	12	35	0
神田川	304	13203	武蔵野市	5	3	3	0	0	2	3	0	0	0
小石川	311	13105	文京区	131	19	72	33	22	17	17	14	115	0
小石川	312	13116	豊島区	16	1	12	4	5	1	4	4	12	0
妙正寺川	321	13114	中野区	94	6	67	15	17	11	6	3	7	4
石神井川	401	13120	練馬区	106	52	93	27	15	12	15	15	21	0
石神井川	402	13229	西東京市	14	7	12	2	1	0	3	1	6	1
石神井川	403	13211	小平市	4	3	1	0	0	1	1	0	1	0
隅田川	501	13102	中央区	152	0	2	1	1	1	1	3	150	0
隅田川	502	13108	江東区	6	0	0	0	0	0	0	0	6	0
隅田川	503	13107	墨田区	79	0	1	0	1	0	0	0	78	0
隅田川	504	13106	台東区	154	4	27	11	31	45	44	22	146	0
隅田川	505	13118	荒川区	7	1	3	2	0	2	2	3	5	0
隅田川	506	13117	北区	54	5	21	23	37	22	21	16	16	0
荒川	511	13123	江戸川区	13	0	1	3	2	2	2	6	2	4
荒川	512	13122	葛飾区	27	0	1	4	15	13	12	15	11	1
荒川	513	13121	足立区	27	0	2	2	19	8	7	8	8	0
荒川	514	13119	板橋区	173	27	112	88	84	28	35	21	29	31
黒目川	601	13222	東久留米市	138	40	131	4	0	1	9	0	2	0
柳瀬川	701	13221	清瀬市	77	7	47	3	3	3	57	5	2	0
柳瀬川	702	13213	東村山市	148	22	120	13	49	49	52	54	4	3
空堀川	712	13220	東大和市	76	18	68	0	0	5	39	2	12	0
空堀川	712	13223	武蔵村山市	40	6	33	4	15	1	9	1	3	0
鶴見川	901	13209	町田市	912	56	738	102	178	511	565	135	102	14

前述の通り、ここで与えた**水系**は便宜的なもので正確なデータではありませんが、練習のために水系別-時代別遺跡数も集計してみます。

```
#旧石器
Tokyo.sumRiver2 <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,旧石器時代) %>%
                          filter(旧石器時代 != 0) %>%
                          count %>%
                          rename(旧石器 = n) %>%
                          ungroup %>%
                          dplyr::select(-旧石器時代)

#縄文
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,縄文時代) %>%
                          filter(縄文時代 != 0) %>%
                          count %>%
                          rename(縄文 = n) %>%
                          ungroup %>%
                          dplyr::select(-縄文時代)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#弥生
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,弥生時代) %>%
                          filter(弥生時代 != 0) %>%
                          count %>%
                          rename(弥生 = n) %>%
                          ungroup %>%
                          dplyr::select(-弥生時代)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#古墳
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,古墳時代) %>%
                          filter(古墳時代 != 0) %>%
                          count %>%
                          rename(古墳 = n) %>%
                          ungroup %>%
                          dplyr::select(-古墳時代)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#奈良
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,奈良時代) %>%
                          filter(奈良時代 != 0) %>%
                          count %>%
                          rename(奈良 = n) %>%
                          ungroup %>%
                          dplyr::select(-奈良時代)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#平安
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,平安時代) %>%
                          filter(平安時代 != 0) %>%
                          count %>%
                          rename(平安 = n) %>%
                          ungroup %>%
                          dplyr::select(-平安時代)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#中世
Tokyo.sumRiverX <- TokyoTotal.age %>%
```

```
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,中世) %>%
                          filter(中世 != 0) %>%
                          count %>%
                          ungroup %>%
                          dplyr::select(-中世) %>%
                          rename(中世 = n)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#近世
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,近世) %>%
                          filter(近世 != 0) %>%
                          count %>%
                          ungroup %>%
                          dplyr::select(-近世) %>%
                          rename(近世 = n)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#不明
Tokyo.sumRiverX <- TokyoTotal.age %>%
                          inner_join(river, by = "自治体コード") %>%
                          group_by(水系,時代不明) %>%
                          filter(時代不明 != 0) %>%
                          count %>%
                          rename(不明 = n) %>%
                          ungroup %>%
                          dplyr::select(-時代不明)
Tokyo.sumRiver2 <- left_join(Tokyo.sumRiver2,Tokyo.sumRiverX, by = "水系")

#NAを0に置換
Tokyo.sumRiver2[is.na(Tokyo.sumRiver2)] <- 0

# 表形式で表示
  #formattableで表出力
  formattable(Tokyo.sumRiver2,list(合計=color_bar("tomato"),
                           旧石器=color_bar("blue"),
                           縄文=color_bar("brown"),
                           弥生=color_bar("orange"),
                           古墳=color_bar("green"),
                           奈良=color_bar("purple"),
                           平安=color_bar("yellow"),
                           中世=color_bar("pink"),
                           近世=color_bar("skyblue"),
                           時代不明=color_bar("lightgrey")))
```

水系	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
空堀川	24	101	4	15	6	48	3	15	0
隅田川	10	54	37	70	70	68	44	401	0
乞田川	29	228	8	35	110	156	78	112	3
荒川	27	116	97	120	51	56	50	50	36
黒目川	40	131	4	0	1	9	0	2	0
三沢川	15	112	7	23	55	60	38	35	6
残堀川	5	16	2	2	3	2	5	5	0
秋川	4	127	6	107	15	27	25	15	3
渋谷川	8	78	21	64	24	11	10	194	4
小石川	20	84	37	27	18	21	18	127	0

水系	旧石器	縄文	弥生	古墳	奈良	平安	中世	近世	不明
神田川	67	223	60	86	43	41	32	248	0
石神井川	62	106	29	16	13	19	16	28	1
浅川	105	784	91	216	378	658	258	454	8
多摩川	102	513	83	502	278	253	195	185	30
鶴見川	56	738	102	178	511	565	135	102	14
平井川	1	14	2	9	5	6	8	8	0
妙正寺川	6	67	15	17	11	6	3	7	4
目黒川	13	63	13	13	5	8	16	25	1
野川	63	106	4	21	46	39	19	42	1
柳瀬川	29	167	16	52	52	109	59	6	3

```
#一時保管オブジェクトを削除
rm(Tokyo.sumRiverX)
```

7-4.表の行列の入れ替え

チャンク22で作成したデータフレーム(Tokyo.sumRiver2)は、行(縦軸)が水系、列(横軸)が時代となっています。**formattable**パッケージによるカラー・バーグラフは、同一列内での相対的な数値の大小を示すので、同じ時代における水系ごとの遺跡数の多寡が表現されています。

ここでデータフレームの行列を入れ替えることができると、今度は同一の水系における時代ごとの遺跡数の多寡を視覚化することができます。

まず次のチャンクを実行すると、データフレームが縦長に変換されます。

```
# Tokyo.sumRiver3 <- rename(Tokyo.sumRiver2, "0旧石器" = 旧石器, "1縄文" = 縄文, "2弥生" = 弥生, "3古墳" = 古墳, "4奈良" = 奈良, "5平安" = 平安, "6中世" = 中世, "7近世" = 近世, "8不明" = 不明) #昇順に並び替えるために時代列名を変更→因子型に変更

Tokyo.sumRiver3 <- gather(Tokyo.sumRiver2, key = 時代, value = 遺跡数, -水系)

head(Tokyo.sumRiver3)
```

```
## # A tibble: 6 x 3
##   水系    時代    遺跡数
##   <chr>  <chr>   <int>
## 1 空堀川 旧石器     24
## 2 隅田川 旧石器     10
## 3 乞田川 旧石器     29
## 4 荒川   旧石器     27
## 5 黒目川 旧石器     40
## 6 三沢川 旧石器     15
```

次に、時代を縦軸(行)、水系を横軸(列)に入れ替えます。水系ごとの時代別の遺跡数の推移が可視化されます。

※2020/10/31追加：なおこれまで「時代」列は文字列(chr)型として扱ってきましたが、ここでは表の順序を「時代」列で指定するために**因子(factor)型**に変更します。因子型は数値や文字(通常は文字コード順)と異なり指定した順列として扱うことが可能になります。1038行目で factor() 関数数の因数 levels() により順列を指定しています。ここでは文字の行列のため levels = c("###", "###") と指定しています。

```
#時代列を因子(factor)型に変更
Tokyo.sumRiver3$時代 <- as.factor(Tokyo.sumRiver3$時代)

#因子型のデータに順列を与える (factor(object, levels = c())
Tokyo.sumRiver3$時代 <- factor(Tokyo.sumRiver3$時代, levels = c("旧石器", "縄文", "弥生", "古墳", "奈良", "平安", "中世", "近世", "不明"))

#縦長のデータを横長に変換(spread())
Tokyo.sumRiver3 <- spread(Tokyo.sumRiver3, key = 水系, value = 遺跡数)

#並べ替え
Tokyo.sumRiver3 <- dplyr::select(Tokyo.sumRiver3, 時代, 多摩川, 野川, 三沢川, 乞田川, 残堀川, 浅川, 秋川, 平井川, 目黒川, 渋谷川, 神田川, 小石川, 妙正寺川, 石神井川, 隅田川, 荒川, 黒目川, 柳瀬川, 空堀川, 鶴見川)

#formattableで表出力
formattable(Tokyo.sumRiver3,list(
                            多摩川=color_bar("blue"),
                            野川=color_bar("skyblue"),
                            三沢川=color_bar("green"),
                            乞田川=color_bar("lightgreen"),
                            残堀川=color_bar("darkblue"),
                            浅川=color_bar("lightblue"),
                            秋川=color_bar("blue"),
                            平井川=color_bar("lightgreen"),
                            目黒川=color_bar("pink"),
                            渋谷川=color_bar("orange"),
                            神田川=color_bar("red"),
                            小石川=color_bar("pink"),
                            妙正寺川=color_bar("orange"),
                            石神井川=color_bar("yellow"),
                            隅田川=color_bar("grey"),
                            荒川=color_bar("black"),
                            黒目川=color_bar("lightgreen"),
                            柳瀬川=color_bar("green"),
                            空堀川=color_bar("darkgreen"),
                            鶴見川=color_bar("tomato")
                            )
            )
```

時代	多摩川	野川	三沢川	乞田川	残堀川	浅川	秋川	平井川	目黒川	渋谷川	神田川	小石川	妙正寺川	石神井川	隅田川	荒川	黒目川	柳瀬川	空堀川	鶴見川
旧石器	102	63	15	29	5	105	4	1	13	8	67	20	6	62	10	27	40	29	24	56
縄文	513	106	112	228	16	784	127	14	63	78	223	84	67	106	54	116	131	167	101	738
弥生	83	4	7	8	2	91	6	2	13	21	60	37	15	29	37	97	4	16	4	102
古墳	502	21	23	35	2	216	107	9	13	64	86	27	17	16	70	120	0	52	15	178
奈良	278	46	55	110	3	378	15	5	5	24	43	18	11	13	70	51	1	52	6	511
平安	253	39	60	156	2	658	27	6	8	11	41	21	6	19	68	56	9	109	48	565
中世	195	19	38	78	5	258	25	8	16	10	32	18	3	16	44	50	0	59	3	135
近世	185	42	35	112	5	454	15	8	25	194	248	127	7	28	401	50	2	6	15	102
不明	30	1	6	3	0	8	3	0	1	4	0	0	4	1	0	36	0	3	0	14

7-5.時代別-遺跡種別の集計

自治体別-または河川水系別-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
  #列名変更
  TokyoTotal.age <- rename(TokyoTotal.age, "遺構" = "主な遺構/概要")
  #ベースリスト
    Tokyo.sp <- TokyoTotal.age %>%
      group_by(自治体コード) %>%
      count %>%
      rename("遺跡数合計" = "n")

  # 縄文貝塚
    #リスト
      Tokyo.Jkaizuka <- TokyoTotal.age %>%
        filter(縄文時代 != 0) %>%
        filter(str_detect(種別, "貝塚"))
    #集計
      Tokyo.st2 <- Tokyo.Jkaizuka %>%
        group_by(自治体コード)  %>%
        count %>%
        rename("貝塚" = "n")
      Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

  # 方形周溝墓
    #リスト
      Tokyo.Yshukoubo <- TokyoTotal.age %>%
        filter(弥生時代 != 0) %>%
        filter(str_detect(遺構, "方形周溝墓"))
    #集計
      Tokyo.st2 <- Tokyo.Yshukoubo %>%
        group_by(自治体コード) %>%
        count %>%
        rename("方形周溝墓" = "n")
      Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

  # 古墳
    #リスト
      Tokyo.Kkofun <- TokyoTotal.age %>%
        filter(古墳時代 != 0) %>%
        filter(str_detect(種別, "古墳"))
    #集計
      Tokyo.st2 <- Tokyo.Kkofun %>%
        group_by(自治体コード) %>%
        count() %>%
        rename("古墳" = "n")
      Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

  # 横穴墓
    #リスト
      Tokyo.Kyokoana <- TokyoTotal.age %>%
        filter(古墳時代 != 0) %>%
        filter(str_detect(種別, "横穴墓"))
    #集計
      Tokyo.st2 <- Tokyo.Kyokoana %>%
        group_by(自治体コード) %>%
        count %>%
        rename("横穴墓" = "n")
      Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

  # 中世城館
    #リスト
      Tokyo.Mjokan <- TokyoTotal.age %>%
        filter(中世 != 0) %>%
        filter(str_detect(種別, "城館"))
    #集計
      Tokyo.st2 <- Tokyo.Mjokan %>%
        group_by(自治体コード) %>%
        count %>%
        rename("城館" = "n")
```

```
    Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

# 塚
  #リスト
    Tokyo.Mtsuka <- TokyoTotal.age %>%
      filter(str_detect(種別, "塚"))
  #集計
    Tokyo.st2 <- Tokyo.Mtsuka %>%
      group_by(自治体コード) %>%
      count %>%
      rename("塚" = "n")
    Tokyo.sp <- left_join(Tokyo.sp, Tokyo.st2, by = "自治体コード")

# NAを0に置換
Tokyo.sp[is.na(Tokyo.sp)] <- 0

# 一時データの削除
rm(Tokyo.st2)

# 水系リストを読み込み・結合
Tokyo.sp <- Tokyo.sp %>%
  inner_join(river, by = "自治体コード") %>%
  dplyr::select("水系", "配列", "自治体コード", "名称", "貝塚", "方形周溝墓", "古墳", "横穴墓", "城館", "塚")

#河川水系でソート
Tokyo.sp <- arrange(Tokyo.sp, 配列)

#formattableで表出力
formattable(Tokyo.sp,list(貝塚=color_bar("blue"),
                          方形周溝墓=color_bar("orange"),
                          古墳=color_bar("green"),
                          横穴墓=color_bar("brown"),
                          城館=color_bar("red"),
                          塚=color_bar("purple")
                          )
          )
```

水系	配列	自治体コード	名称	貝塚	方形周溝墓	古墳	横穴墓	城館	塚
多摩川	101	13111	大田区	16	3	55	57	4	46
多摩川	102	13112	世田谷区	2	3	79	26	7	18
多摩川	103	13219	狛江市	0	1	24	0	0	1
多摩川	104	13208	調布市	0	0	33	3	4	1
多摩川	105	13206	府中市	0	0	24	1	0	9
多摩川	106	13215	国立市	0	0	31	0	1	0
多摩川	107	13202	立川市	0	0	3	0	1	0
多摩川	108	13207	昭島市	0	0	4	0	1	2
多摩川	109	13218	福生市	0	0	0	0	0	0
多摩川	110	13227	羽村市	0	0	0	0	0	0
多摩川	111	13205	青梅市	0	1	0	0	12	8
多摩川	112	13308	奥多摩町	0	0	0	0	0	0
野川	121	13204	三鷹市	0	0	1	6	0	0
野川	122	13210	小金井市	0	0	0	1	0	1
野川	123	13214	国分寺市	0	0	0	0	0	1
三沢川	131	13225	稲城市	0	1	0	2	4	5

水系	配列	自治体コード	名称	貝塚	方形周溝墓	古墳	横穴墓	城館	塚
乞田川	141	13224	多摩市	0	0	13	3	2	6
残堀川	151	13303	瑞穂町	0	0	0	0	1	0
浅川	161	13212	日野市	0	0	7	3	4	5
浅川	162	13201	八王子市	0	7	5	2	26	18
秋川	171	13228	あきる野市	0	0	75	0	9	2
秋川	172	13307	檜原村	0	0	0	0	1	0
平井川	181	13305	日の出町	0	0	1	0	2	1
目黒川	201	13109	品川区	6	0	3	1	3	6
目黒川	202	13110	目黒区	1	0	2	0	2	2
渋谷川	211	13103	港区	13	1	19	2	4	15
渋谷川	212	13113	渋谷区	4	0	16	14	1	4
神田川	301	13101	千代田区	6	1	0	0	1	7
神田川	302	13104	新宿区	1	5	0	0	3	2
神田川	303	13115	杉並区	0	3	1	1	0	0
神田川	304	13203	武蔵野市	0	0	0	0	0	1
小石川	311	13105	文京区	15	0	6	0	0	15
小石川	312	13116	豊島区	3	1	1	0	0	3
妙正寺川	321	13114	中野区	0	0	2	1	1	0
石神井川	401	13120	練馬区	0	3	0	0	3	6
石神井川	402	13229	西東京市	0	0	0	0	0	1
石神井川	403	13211	小平市	0	0	0	0	0	0
隅田川	501	13102	中央区	0	0	0	0	0	1
隅田川	502	13108	江東区	0	0	0	0	0	0
隅田川	503	13107	墨田区	0	0	0	0	0	1
隅田川	504	13106	台東区	5	0	5	0	0	9
隅田川	505	13118	荒川区	1	0	0	0	1	1
隅田川	506	13117	北区	7	7	16	1	4	7
荒川	511	13123	江戸川区	0	0	0	0	1	6
荒川	512	13122	葛飾区	0	0	4	0	2	3
荒川	513	13121	足立区	0	0	10	0	1	3
荒川	514	13119	板橋区	9	10	4	0	3	35
黒目川	601	13222	東久留米市	0	1	0	0	0	0
柳瀬川	701	13221	清瀬市	0	0	0	0	2	1
柳瀬川	702	13213	東村山市	0	0	0	0	0	4
空堀川	712	13220	東大和市	0	0	0	0	0	0
空堀川	712	13223	武蔵村山市	0	0	0	0	0	1
鶴見川	901	13209	町田市	0	3	6	19	10	15

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(経度) %>%
                          mutate(色 = "blue")
write.csv(Tokyo.Jkaizuka, "13Tokyo_kaizukaSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.Yshukoubo <- Tokyo.Yshukoubo %>%
                          drop_na(経度) %>%
                          mutate(色 = "orange")
write.csv(Tokyo.Yshukoubo, "13Tokyo_shukoboSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.Kkofun<- Tokyo.Kkofun %>%
                          drop_na(経度) %>%
                          mutate(色 = "green")
write.csv(Tokyo.Kkofun, "13Tokyo_kofunSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.Kyokoana <- Tokyo.Kyokoana %>%
                          drop_na(経度) %>%
                          mutate(色 = "brown")
write.csv(Tokyo.Kyokoana, "13Tokyo_yokoanaSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.Mjokan <- Tokyo.Mjokan %>%
                          drop_na(経度) %>%
                          mutate(色 = "red")
write.csv(Tokyo.Mjokan, "13Tokyo_jokanSJS.csv", row.names=FALSE, fileEncoding = "CP932")

Tokyo.Mtsuka <- Tokyo.Mtsuka %>%
                          drop_na(経度) %>%
                          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。

また**R**上で分布図を作成することも可能です。関心のある方はこちら(https://github.com/LiYingWang/Japan_GIS_workshop_202006_LW)をご参照ください。リクエストが多ければ、ワークショップの開催も企画します。