R によるデータ解析のためのデータ可視化

第 3 部: 地図を描画する

瓜生真也（国立環境研究所）

2021/9/5

第 3 部では、地理空間データを利用した地図作成を行います。教科書においては、アメリカ合衆国の行政単位である州や郡の可視化例が示されます。このチュートリアルでは、身近なデータとして日本の行政区域をもとに地図を作る方法を紹介します。

行政区域の地図は、地図作成の作業において対象となることが多いです。この地図は都道府県や市町村などの行政区域、あるいはより小さな地域を対象に行われた観測データと紐付けた階級区分図（コロプレス図）に代表されます。

階級区分図を作る作業において、地図と観測データの結合作業を伴うことがあります。また自在に両データを操作できるようになることで、例えば市町村単位で記録された値をより大きな都道府県単位に集計しての可視化もできます。本チュートリアルでは、こうした地図データの操作についても紹介します。さらに教科書の内容に従い、地理空間関係を考慮した地図表現方法についても学びます。

本チュートリアルで利用するデータ並びにコードを GitHub にて公開しています。必要に応じて参照してください。

まずは本チュートリアル全体で用いる R パッケージを読み込みます。このほかにいくつかのパッケージが登場しますが、それらは都度必要な時に読み込みます。

```
library(dplyr) # データ操作を容易に行うパッケージ
library(ggplot2) # 可視化のためのパッケージ
library(sf) # 地理空間データを扱うパッケージ
library(rnaturalearth) # パブリックドメインで利用可能な行政地図データを提供するパッケージ
```

行政区域の地図

sf オブジェクトと geom_sf()による地図描画

地図描画のために最低限必要なのは、地図に示す対象のデータです。今回は行政上の境界線を示す行政区域が該当します。このデータは通常、shp ファイルなど

の形式によって提供されます。ですがここではまず、R から直接利用可能なデータを使う例を見ていきます。これはパブリックドメインで利用可能な Natural Earth のデータを R で呼び出す rnaturalearth パッケージを利用します。次のコードチャンクを実行すると日本の都道府県の形状を示すポリゴンデータが手に入ります。

```
# Natural Earth から日本の都道府県ポリゴンを取得
ne_jpn <-
  ne_states(country = "Japan",
                          returnclass = "sf") %>%
  # 使わない列を除外し、必要な列だけを選ぶようにします
  select(iso_3166_2, gn_name) %>%
  tibble::new_tibble(nrow = nrow(.), class = "sf")
```

ポリゴンを取得する ne_states()では返り値のクラスを指定する引数 returnclass があります。デフォルトでは“sp”ですが、ここでは“sf”を指定しました。

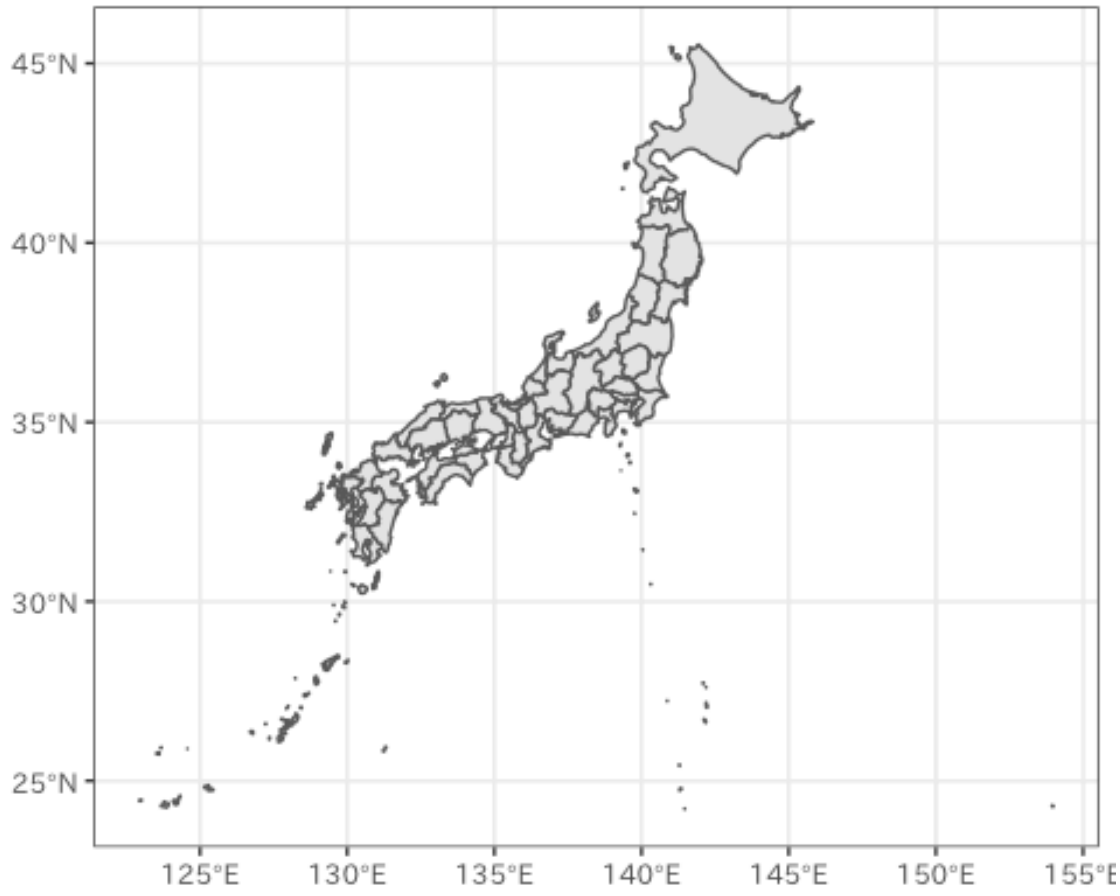
```
class(ne_jpn)

## [1] "sf"         "tbl_df"     "tbl"        "data.frame"

glimpse(ne_jpn)

## Rows: 47
## Columns: 3
## $ iso_3166_2 <chr> "JP-46", "JP-44", "JP-40", "JP-41", "JP-42", "JP-43
", "JP-4…
## $ gn_name    <chr> "Kagoshima-ken", "Oita-ken", "Fukuoka-ken", "Saga-k
en", "Na…
## $ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((129.7832 31..., MULTIP
OLYGON (…
```

オブジェクトのクラスを確認すると、sf クラスであることが確認できます。データの内容を見ると、日本の都道府県に関する列（iso_3166_2、gn_name）のほかに geometry 列があります。この列は都道府県の形状を示すポリゴンの情報が格納された sfc (simple-feature column)です（詳細は割愛します）。

sf は近年の R での地理空間データ、特にベクトル形式のデータを扱うために広く利用されるオブジェクトクラスです。同名の sf パッケージにより提供されます。

ggplot2 にもこの sf クラスのオブジェクトを描画する関数 geom_sf() が用意されています。教科書では地図の描画に geom_polygon()を用いました。しかし sf オブジェクトに対しては geom_sf()を適用します。この関数を使った以下のコードチャンクが ggplot2 を用いた地図描画の基礎となります。

```
ggplot(ne_jpn) +
  geom_sf()
```



geom_sf()の使用例

`geom_sf()`は ggplot2 の他の `geom_*()`と同様に操作します。これは散布図や棒グラフなどの geom と同じように描画するデータを指定して `aes()`を介したマッピングが可能なことを意味します。この例を見るために続いて県ごとの塗り分け図を描画してみます。

都道府県別の選挙結果の地図

Natural Earth のデータには塗り分けに使える数値がないため、別途用意することになります。教科書ではアメリカ合衆国における大統領選挙のデータが使われます。しかしここでは 2017 年（平成 29 年）に行われた第 48 回衆議院議員総選挙のデータを用います。このデータの作成手順については後述します。

```
# natural earth の都道府県ポリゴンと選挙データの結合
# 選挙データ: 第48回衆議院議員総選挙小選挙区での選挙結果
# データを用意する手順は後述します

# 格納されたデータを確認
glimpse(ne_jpn_shugiin48)
```

```
## Rows: 47
## Columns: 8
## $ iso_3166_2 <chr> "JP-46", "JP-44", "JP-40", "JP-41", "JP-42", "JP-43
", "JP-4…
## $ prefecture <chr> "鹿児島県", "大分県", "福岡県", "佐賀県", "長崎県", "
熊本県…
## $ gn_name    <chr> "Kagoshima-ken", "Oita-ken", "Fukuoka-ken", "Saga-k
en", "Na…
## $ party      <chr> "自由民主党", "自由民主党", "自由民主党", "自由民主党",
 "自…
## $ votes      <dbl> 403187.0, 279778.0, 1190045.0, 178075.0, 335814.0,
489223.0…
## $ is_ruling  <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRU
E, TRUE,…
## $ prop       <dbl> 53.15506, 51.12192, 54.56482, 44.24345, 52.18960, 5
9.92354,…
## $ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((129.7832 31..., MULTIP
OLYGON (…
```

```
class(ne_jpn_shugiin48) # sf オブジェクト
```

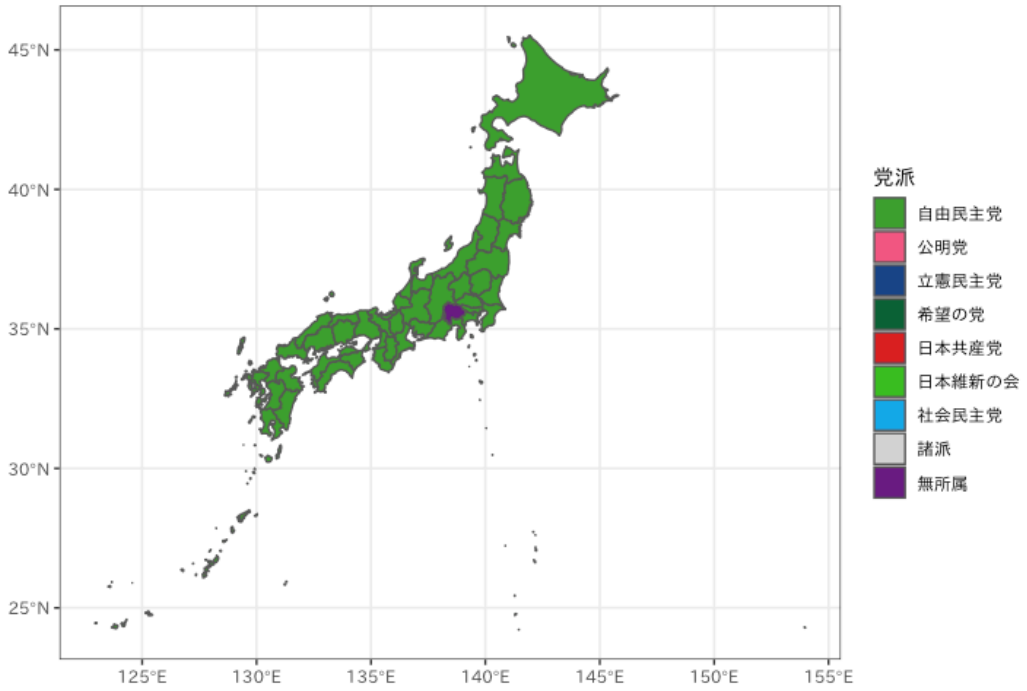
```
## [1] "sf"         "tbl_df"     "tbl"        "data.frame"
```

選挙結果のデータには各都道府県でもっとも得票率の高かった党派(party 列)の得票数(votes 列)と得票率(prop 列)が記録されています。オブジェクトのクラスは sf です。これは先述の通り、R での地理空間データを扱う形式の一つです。このクラスのオブジェクトは日本の白地図の例で見たように geom_sf() によるマッピングが可能です。

ggplot2 では aes()を使った要素の塗り分けに fill を指定することを思い出してください。これは sf オブジェクトをマッピングするための関数 geom_sf() において

も同様に機能します。データに付与された変数を用いて、塗り分けを行う例を次に示します。ここではあらかじめ党派の違いを示すカラーコードを定義して地図の塗り分けに適用します。

```
# 党派を示すカラーコードを定義する
party_colors <- c(`自由民主党` = "#41A12E",
                  `公明党` = "#F35A82",
                  `立憲民主党` = "#1B4787",
                  `希望の党` = "#136437",
                  `日本共産党` = "#D90A26",
                  `日本維新の会` = "#3EC021",
                  `社会民主党` = "#1CA9E9",
                  `諸派` = "#D3D3D3",
                  `無所属` = "#691D82")

p <-
  ggplot(data = ne_jpn_shugiin48,
         aes(fill = party))
```

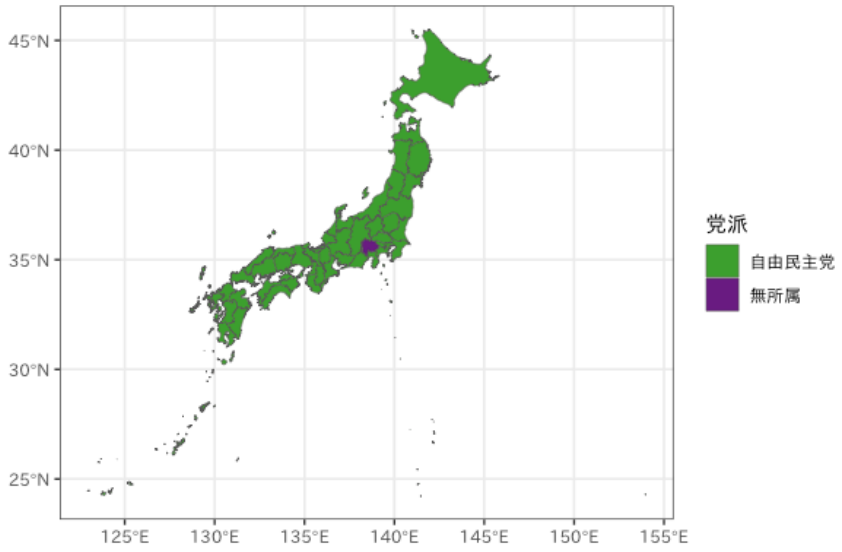


都道府県ごとの第一党

図を見ると全国で自由民主党が優勢だったことがわかりますが、山梨県のみ無所属の党派が第一党となったことも示されています。また、自由民主党、無所属以外の党派はいずれの県においても第一党となっていない点も明らかになりました。

この結果を反映して、使用されない凡例を減らした図を作り直してみましょう。さらに追加の処理として県境を明確にするために線の太さを変更します。コードを見る前に、どのような変更を加えることになるか考えてみてください。

```
p2 <-
  p +
  # 県境を明確にするために太さを調節する
  geom_sf(size = 0.2) +
  # 使われない凡例を削除する
  scale_fill_manual(values = party_colors[c(1, 9)],
                    guide = guide_legend(title = "党派"))

p2
```



都道府県ごとの第一党。修正版

`geom_sf()` を使った ggplot2 による地図描画はこのように sf オブジェクトを対象とし、データフレーム中の変数をマッピングに指定可能です。次は、この地図をより地図らしく仕上げていく方法を紹介します。

地図を磨き上げる

グラフの見た目やデータを整える作業は地図作成においても同様です。地理空間データでは、特に 3 次元の地球上の位置を 2 次元に投影するためにさまざまな表現（投影法）が用いられます。また、Natural Earth の日本ポリゴンと選挙データで示したように、興味のある地理空間データと説明のためのデータが揃っている場合は多くはありません。そのため、必要に応じてデータの結合を行う作業が伴

います。加えて行政単位の地図では地図上の単位と集計単位が異なることもあります。こうした地図表現で求められる処理方法について、下記で解説します。

投影法の変換

地図の見た目は対象物の位置を示す座標の表現方法によって変わります。3 次元である地球上の位置を表現するために主に地理座標系、投影座標系が使われます。先ほどの図では緯度と経度で表現する地理座標系が用いられています。この座標系の特徴は広域の表示に適していますが、一方で地球が完全な球ではなく楕円形であることに起因した面積や角度の歪みが生じます。これに対して投影座標系では 2 次元の XY 座標で表現します。用いる投影法によって、面積・距離・角度のいずれかに対する歪みを補正できます。

代表的な座標系として下記のものがあります。

- 地理座標系... 緯線と経線で構成される。東西方向を経度、南北方向を緯度で表現する。多くの場合、北半球は正の緯度を持ち、南半球は負の緯度となる。経度の原点として本初子午線が使用される。
- 投影座標系
  - モルワイデ図法 ... 正積図法（面積が正しく表現される）の一つ。世界地図の表現に使われる。
  - メルカトル図法... - 正角図法（角が正しく表現される）の一つ。子午線は平行、等間隔で与えられる。緯線も平行だが極に近づくにつれて間隔が広くなる。また極地域は表示されず、極地域に近づくほど面積に歪みが生じる。
  - ユニバーサル横メルカトル図法（UTM 座標系）... 円筒図法の一種。地球を 60 のゾーンに分割し、領域内での歪みを少なくする。日本ではゾーン 51 から 54 が割り当てられ、国土地理院発行の縮尺 1:10,000 から 1:200,000 の地形図で使用される。
  - 平面直角座標系... 日本国内を測量するために策定された平面直交座標系であり、公共測量の場において使われる。全国を 19 の座標系に区分する

特に日本周辺を表示する座標系としてメルカトル図法、UTM 座標系、平面直角座標系が使われます。

ggplot2 を介した地図描画では 2 つの方法によって地図データの座標系を変換して表現可能です。一つは `coord_sf()`を用いて任意の座標系を指定する方法、もう一

つは対象のオブジェクトの座標参照系 (Coordinate Reference System: CRS) を事前に変更しておく方法です。

広範囲を表示する地図では座標系の違いが明確になります。それぞれの例を見るために、改めて Natural Earth から全球のポリゴンデータをダウンロードします。

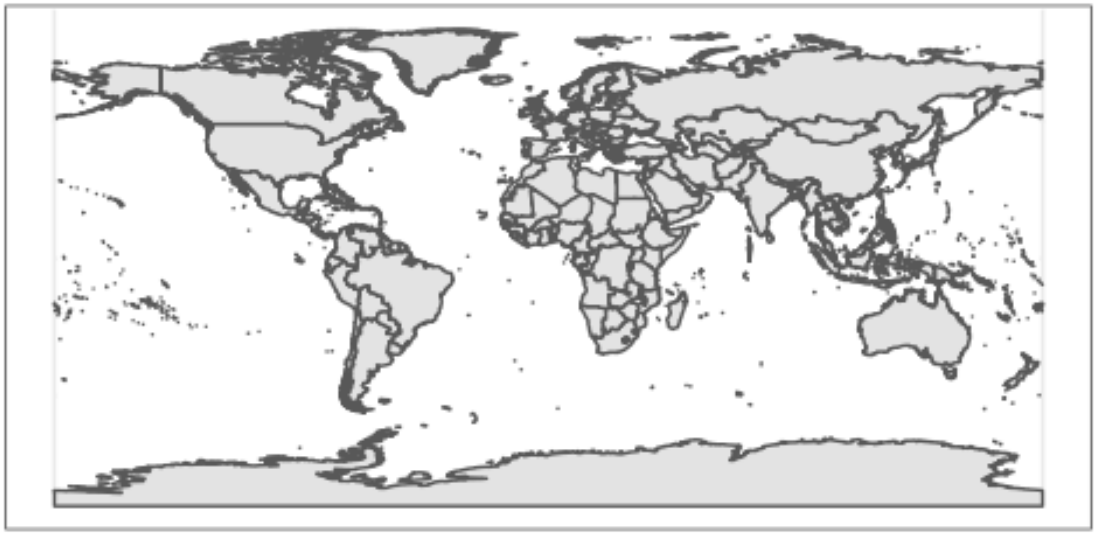
```
# Natural Earth から全球のポリゴンデータをダウンロード
ne_world <-
  ne_countries(scale = 10,
               returnclass = "sf")

x <-
  st_crs(ne_world)
x$input
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

sf オブジェクトは座標参照系に関する情報を所有しています。これは sf オブジェクトをコンソールに出力した際にも表示されますが、`st_crs()`を使った確認もできます。`x$input` で表示される値が座標参照系の要素となります。これを見ると `ne_asia` の座標系は地理座標系であることがわかります（`+proj=longlat`）。

地理座標系の sf オブジェクトを `geom_sf()`で出力すると、デフォルトで緯度と経度を X 軸と Y 軸の平面上に投影した結果となります。

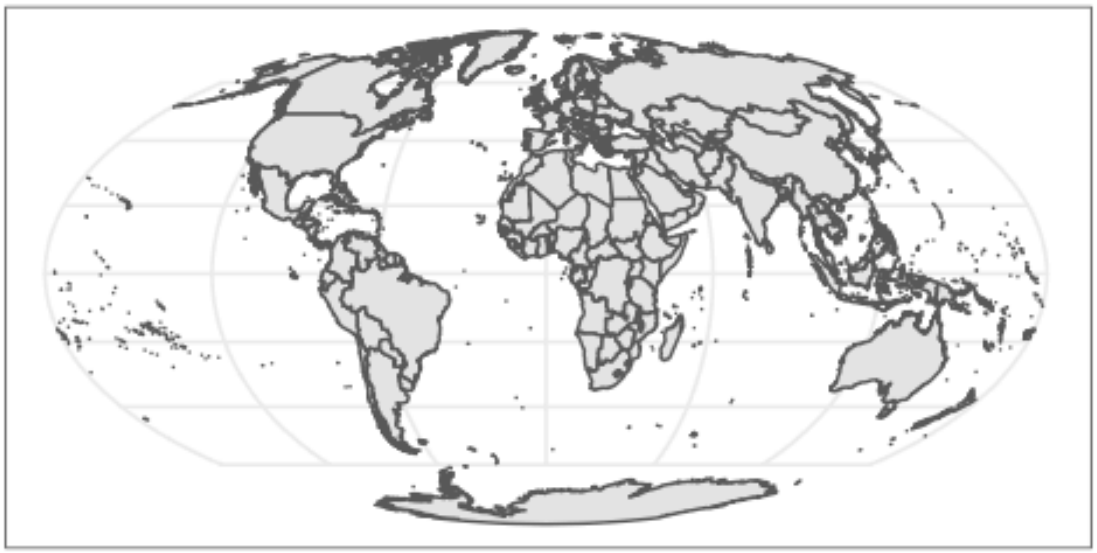
```
p <-
  ggplot(data = ne_world) +
  geom_sf()

p
```



世界地図の描画

それではこの図の投影法を変更してみます。まずはあらかじめデータをマッピングした後での coord_sf()を利用した変更方法です。引数 crs に任意の座標参照系を指定して実行します。次のチャンクコードではモルワイデ図法による投影法で地図を描画させます。

```
# モルワイデ図法による世界地図の描画
p +
  coord_sf(crs = "+proj=moll")
```



モルワイデ図法による世界地図の描画

もう一つ、座標参照系を事前に変更する方法では、sf パッケージの`st_transform()`を使います。この関数は座標系を変更したいオブジェクトを対象に、新たに指定する座標系を入力して実行します。次のチャンクコードでこの処理を実行し、`geom_sf()`によるマッピングを行います。

```
# st_transform()による座標参照系の変更
ne_world_moll <-
  st_transform(ne_world, crs = "+proj=moll")

# 結果は省略
```

```
ggplot(data = ne_world_moll) +
  geom_sf()
```

地理空間データの結合と集約

地理空間データと属性データを紐づける

ここでは再び衆議院議員総選挙のデータを扱います。先述の通り、このデータは Natural Earth が提供する日本の都道府県ポリゴンと選挙結果が結合されたものとなっています。これによって都道府県の地図の上に選挙結果をマッピング可能になりました。では、このデータがどのようにして作成されたものなのか、現実の結合作業で生じる問題にも触れながら解説します。

選挙データは GitHub リポジトリの data フォルダに保存されています。このリポジトリを git クローンし、次のチャンクコードを実行するとデータが読み込まれます。

```
df_shugiin48_party_votes <-
  readr::read_rds(here::here("data/shugiin48_prefecture_party_votes.rds
"))

glimpse(df_shugiin48_party_votes)
```

```
## Rows: 48
## Columns: 31
## $ 区分             <chr> "北海道", "青森県", "岩手県", "宮城県", "秋田県",
 "山形…
## $ 自由民主党_男    <dbl> 1107667, 365462, 226455, 552240, 261709, 214176,
 425155, 477…
## $ 自由民主党_女    <dbl> 82096.0, NA, 57381.0, NA, NA, 103973.1, NA, 165
437.0, NA, 20…
## $ 自由民主党_計    <dbl> 1189763.0, 365462.0, 283836.0, 552240.0, 261709.
0, 318149.1,…
## $ 立憲民主党_男    <dbl> 604003, NA, NA, NA, NA, NA, NA, NA, NA, 67456,
119091, 34343…
## $ 立憲民主党_女    <dbl> 234162, NA, NA, 78704, NA, NA, NA, NA, NA, NA,
73250, NA, 17…
## $ 立憲民主党_計    <dbl> 838165, NA, NA, 78704, NA, NA, NA, NA, NA, 6745
6, 192341, 34…
## $ 希望の党_男      <dbl> 162745.0, 171280.0, 186376.0, 99141.0, 220759.
6, 236150.0, …
## $ 希望の党_女      <dbl> 90363, NA, NA, NA, NA, NA, NA, 51060, 42820, 3
0127, 108069,…
## $ 希望の党_計      <dbl> 253108.0, 171280.0, 186376.0, 99141.0, 220759.
6, 236150.0, …
```

```
## $ 公明党_男        <dbl> 96795, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
 NA, 112597,…
## $ 公明党_女        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
 NA, NA, NA…
## $ 公明党_計        <dbl> 96795, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
 NA, 112597,…
## $ 日本共産党_男    <dbl> 124196.00, 37007.00, NA, 58795.00, 25858.39, 37
518.87, 55121…
## $ 日本共産党_女    <dbl> 35543, 19004, 21549, NA, 13642, NA, NA, 66173,
NA, 25818, 18…
## $ 日本共産党_計    <dbl> 159739.00, 56011.00, 21549.00, 58795.00, 39500.
39, 37518.87,…
## $ 日本維新の会_男 <dbl> 21643, NA, NA, 10001, NA, NA, NA, NA, NA, NA, 7
4743, 52148, N…
## $ 日本維新の会_女 <dbl> NA, NA, NA, NA, NA, NA, 9685, NA, NA, NA, NA, N
A, NA, NA, NA,…
## $ 日本維新の会_計 <dbl> 21643, NA, NA, 10001, NA, NA, 9685, NA, NA, NA,
 74743, 52148,…
## $ 社会民主党_男    <dbl> NA, NA, NA, NA, NA, NA, 8063, NA, NA, 14008, NA,
 10713, 3819…
## $ 社会民主党_女    <dbl> NA, NA, NA, NA, NA, NA, 7186, NA, NA, NA, 36755,
 NA, NA, NA,…
## $ 社会民主党_計    <dbl> NA, NA, NA, NA, NA, NA, 15249, NA, NA, 14008, 3
6755, 10713, …
## $ 諸派_男          <dbl> 7632, 4115, NA, 2413, NA, 2329, NA, 2564, 1561,
 NA, 2009,…
## $ 諸派_女          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
 13965.08,…
## $ 諸派_計          <dbl> 7632.00, 4115.00, NA, 2413.00, NA, 2329.00, NA,
 2564.00, …
## $ 無所属_男        <dbl> 125771, NA, 130229, 90970, NA, NA, 92930, 7771
9, 108712, N…
## $ 無所属_女        <dbl> NA, NA, NA, 110243, NA, NA, 126664, NA, NA, NA,
 21614, NA,…
## $ 無所属_計        <dbl> 125771, NA, 130229, 201213, NA, NA, 219594, 77
719, 108712,…
## $ 合計_男          <dbl> 2250452.0, 577864.0, 543060.0, 813560.0, 50832
7.0, 490173…
## $ 合計_女          <dbl> 442164.0, 19004.0, 78930.0, 188947.0, 13642.0,
 103973.1, …
## $ 合計_計          <dbl> 2692616, 596868, 621990, 1002507, 521969, 5941
47, 902732,…
```

df_shugiin48_party_votes は第 48 回衆議院議員総選挙小選挙区における都道府県・届出政党等別の得票数のデータセットです。候補者の性別ごとに与党、野党ならびに無所属での得票数が記録されています。こうした属性データを地図データと結合し、地図描画を行うためのデータセットを構築する方法を下記で説明していきます。なおこのデータセットの作り方は GitHub リポジトリ中のコード（data-raw/shugiin48.R）を参照してください。

```
# 選挙データの加工 1/5
df_shugiin48_party_votes_mod <-
  df_shugiin48_party_votes %>%
  filter(区分 != "計") %>%
  select(prefecture = 区分, ends_with("計")) %>%
  select(!starts_with("合計"))
```

上記のチャンクコードの説明をします。まず filter()で全体の集計結果の行を除外します。続いて select()によって列選択を行います。その際、一部の列名を変更しています。

ここで示したデータ操作の関数 filter()や select()は第 2 部で登場した dplyr パッケージによるものです。dplyr パッケージには、このあと扱う mutate()やデータ結合の関数も登場しますが、他にもデータ操作のための関数が豊富に提供されています。

作成したオブジェクトを確認すると、集計対象の都道府県(prefecture)と各党の得票数の合計列(党名_計の形式)が残っていることがわかります。このようにデータ加工の作業でも、グラフ作成の時同様に中間オブジェクトを作って処理結果を確認しておくと良いでしょう。

```
head(df_shugiin48_party_votes_mod)
```

```
## # A tibble: 6 × 10
##   prefecture 自由民主党_計 立憲民主党_計 希望の党_計 公明党_計 日本共産党_計
##   <chr>              <dbl>         <dbl>       <dbl>     <dbl>          <dbl>
## 1 北海道           1189763        838165      253108      96795         159739
## 2 青森県            365462            NA      171280         NA          56011
## 3 岩手県            283836            NA      186376         NA          21549
## 4 宮城県            552240         78704       99141         NA          58795
## 5 秋田県            261709            NA      220760.        NA
```

```
39500.
## 6 山形県            318149.            NA     236150         NA
37519.
## # … with 4 more variables: 日本維新の会_計 <dbl>, 社会民主党_計 <dbl>,
## #   諸派_計 <dbl>, 無所属_計 <dbl>
```

続いて、このデータを党別に集計等ができるよう、縦長の形式、すなわち党名の列と得票数の列に整形してみましょう。これには tidyr パッケージの関数 pivot_longer()を使います。

```
# 選挙データの加工 2/5
df_shugiin48_party_votes_long <-
  df_shugiin48_party_votes_mod %>%
  tidyr::pivot_longer(cols = ends_with("計"),
                      names_to = "party",
                      values_to = "votes") %>%
  mutate(party = stringr::str_remove(party, "_計"),
         is_ruling = if_else(party %in% c("自由民主党", "公明党"),
                             TRUE,
                             FALSE))
```

pivot_longer()の処理では引数 cols に整形対象の列（ここでは党名の列が党名_計の形式であることを利用して ends_with()で複数列を対象にする）を選択し、列名が格納される列を names_to、各列に含まれる対応する値を格納する列を values_to 引数にそれぞれ指定します。最後に、mutate()で列の値に変更を加えます。ここでは 2 つの処理が行われ、一つは新たに党名が格納された party 列から元の列名に含まれる「_計」という文字列を削除し、党名のみが記録されるようにします。もう一つは党名が「自由民主党」および「公明党」、党名が与党である時に *TRUE* が記録される is_ruling 列を新たに作る処理です。ここでも処理結果を中間オブジェクトとして残しておきます。

```
head(df_shugiin48_party_votes_long)
```

```
## # A tibble: 6 × 4
##   prefecture party          votes is_ruling
##   <chr>      <chr>          <dbl> <lgl>
## 1 北海道      自由民主党   1189763 TRUE
## 2 北海道      立憲民主党    838165 FALSE
## 3 北海道      希望の党      253108 FALSE
## 4 北海道      公明党         96795 TRUE
## 5 北海道      日本共産党    159739 FALSE
## 6 北海道      日本維新の会   21643 FALSE
```

ここまでできたら、地図データと結合させるために最後の一手を加えます。それは以下のチャンクコードで示すように、都道府県ごとに党派ごとの得票率を求め、各都道府県でもっとも得票率が高い党派を抽出する処理です。

```
# 選挙データの加工 3/5
df_shugiin48_party_votes_tops <-
  df_shugiin48_party_votes_long %>%
  group_by(prefecture) %>%
  mutate(prop = votes / sum(votes, na.rm = TRUE) * 100) %>%
  top_n(n = 1, wt = prop) %>%
  ungroup()

head(df_shugiin48_party_votes_tops)
```

```
## # A tibble: 6 × 5
##   prefecture party         votes is_ruling  prop
##   <chr>      <chr>         <dbl> <lgl>     <dbl>
## 1 北海道     自由民主党 1189763  TRUE       44.2
## 2 青森県     自由民主党  365462  TRUE       61.2
## 3 岩手県     自由民主党  283836  TRUE       45.6
## 4 宮城県     自由民主党  552240  TRUE       55.1
## 5 秋田県     自由民主党  261709  TRUE       50.1
## 6 山形県     自由民主党  318149. TRUE       53.5
```

地図データと紐付けるデータが用意できたら、いよいよ結合処理を行います。ここで改めて日本の都道府県ポリゴンの確認をします。このデータでは都道府県の名前がローマ字表記で記録されています。一方の選挙データは漢字です。そのため、この状態では結合処理が失敗します。

```
# 2 つのデータセットでの都道府県名の表記
ne_jpn$gn_name
```

```
##  [1] "Kagoshima-ken" "Oita-ken"      "Fukuoka-ken"   "Saga-ken"
##  [5] "Nagasaki-ken"  "Kumamoto-ken"  "Miyazaki-ken"  "Tokushima-ken"
##  [9] "Kagawa-ken"    "Ehime-ken"     "Kochi-ken"     "Shimane-ken"
## [13] "Yamaguchi-ken" "Tottori-ken"   "Hyogo-ken"     "Kyoto-fu"
## [17] "Fukui-ken"     "Ishikawa-ken"  "Toyama-ken"    "Niigata-ken"
## [21] "Yamagata-ken"  "Akita-ken"     "Aomori-ken"    "Iwate-ken"
## [25] "Miyagi Ken"    "Fukushima-ken" "Ibaraki-ken"   "Chiba-ken"
## [29] "Tokyo-to"      "Kanagawa-ken"  "Shizuoka-ken"  "Aichi-ken"
## [33] "Mie-ken"       "Wakayama-ken"  "Osaka-fu"      "Okayama-ken"
## [37] "Hiroshima-ken" "Hokkaido"      "Okinawa-ken"   "Gunma-ken"
## [41] "Nagano-ken"    "Tochigi-ken"   "Gifu-ken"      "Shiga-ken"
## [45] "Saitama-ken"   "Yamanashi-ken" "Nara-ken"
```

```
df_shugiin48_party_votes_tops$prefecture
```

```
##  [1] "北海道"    "青森県"    "岩手県"    "宮城県"    "秋田県"    "山形県"
##  [7] "福島県"    "茨城県"    "栃木県"    "群馬県"    "埼玉県"    "千葉県"
## [13] "東京都"    "神奈川県" "新潟県"    "富山県"    "石川県"    "福井県"
## [19] "山梨県"    "長野県"    "岐阜県"    "静岡県"    "愛知県"    "三重県"
## [25] "滋賀県"    "京都府"    "大阪府"    "兵庫県"    "奈良県"    "和歌山県"
## [31] "鳥取県"    "島根県"    "岡山県"    "広島県"    "山口県"    "徳島県"
## [37] "香川県"    "愛媛県"    "高知県"    "福岡県"    "佐賀県"    "長崎県"
## [43] "熊本県"    "大分県"    "宮崎県"    "鹿児島県" "沖縄県"
```

データフレーム間の結合処理では共通の値が記録された列が必要になります。そこで今度は都道府県ポリゴンデータに処理を加えます。ベクトルでローマ字表記に対応する都道府県の漢字を与えても良いですが、ここでは新たなデータセットを利用して結合に用いる列を用意する方法を示します。

zipangu パッケージのデータセットには都道府県の漢字、ローマ字両方の表記が記録されます。この情報を都道府県のポリゴンデータと紐づけてみましょう。結合のための関数は `inner_join()`を使います。この関数は結合対象の 2 つのデータセット間で共通の値をもつ行だけを結合、返却します。

```
# zipangu パッケージからデータセットを利用する
jpnprefs <-
  zipangu::jpnprefs %>%
  select(prefecture_kanji, gn_name = prefecture)
glimpse(jpnprefs)
```

```
## Rows: 47
## Columns: 2
## $ prefecture_kanji <chr> "北海道", "青森県", "岩手県", "宮城県", "秋田県
", "山…
## $ gn_name          <chr> "Hokkaido", "Aomori-ken", "Iwate-ken", "Miyag
i-ken", …
```

```
ne_jpn %>%
  inner_join(jpnprefs,
             by = "gn_name")
```

```
## Simple feature collection with 46 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 122.9382 ymin: 24.2121 xmax: 153.9856 ymax: 45.52
041
## CRS:           +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs
84=0,0,0
## # A tibble: 46 × 4
##    iso_3166_2 gn_name                                  geometry prefec
```

```
ture_kanji
##    <chr>      <chr>                          <MULTIPOLYGON [°]> <chr>

##  1 JP-46      Kagoshima-ken (((129.7832 31.79963, 129.7909 31.… 鹿児島
県
##  2 JP-44      Oita-ken      (((131.2009 33.61271, 131.2199 33.… 大分県

##  3 JP-40      Fukuoka-ken   (((130.0363 33.45759, 130.0402 33.… 福岡県

##  4 JP-41      Saga-ken      (((129.8145 33.33125, 129.8147 33.… 佐賀県

##  5 JP-42      Nagasaki-ken  (((130.2041 32.94357, 130.2038 32.… 長崎県

##  6 JP-43      Kumamoto-ken  (((130.3446 32.16178, 130.345 32.1… 熊本県

##  7 JP-45      Miyazaki-ken  (((131.8723 32.73163, 131.8711 32.… 宮崎県

##  8 JP-36      Tokushima-ken (((134.4424 34.20827, 134.4686 34.… 徳島県

##  9 JP-37      Kagawa-ken    (((133.5919 34.02381, 133.6209 34.… 香川県

## 10 JP-38      Ehime-ken     (((132.6399 32.90892, 132.6272 32.… 愛媛県

## # … with 36 more rows
```

どうしたことでしょう。結果のデータフレームには 46 行しかありません。元の都道府県データには 47 県分あったことから、1 県の結合が失敗したことが推測されます。これをコードで確認してみましょう。これは先ほどとは別の結合関数 anti_join()を使って調べられます。この関数は他の結合関数とは異なり、データセット間で一致しない行を返却します。つまりここで出力される都道府県名に問題があることを示します。

```
ne_jpn %>%
    anti_join(jpnprefs,
              by = "gn_name")

## Simple feature collection with 1 feature and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 140.2696 ymin: 37.7702 xmax: 141.6736 ymax: 38.99
015
## CRS:           +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs
84=0,0,0
## # A tibble: 1 × 3
```

```
##   iso_3166_2 gn_name
  geometry
##   <chr>      <chr>                                                  <MULTIPO
LYGON [°]>
## 1 JP-04      Miyagi Ken (((141.6403 38.9675, 141.6406 38.9652, 141.645
3 38.923,…
```

```
ne_jpn$gn_name[25]
```

```
## [1] "Miyagi Ken"
```

```
ne_jpn$gn_name[1]
```

```
## [1] "Kagoshima-ken"
```

```
ne_jpn$gn_name[47]
```

```
## [1] "Nara-ken"
```

結合に失敗した原因はなんでしょう。また、結合を正しく行うにはどうすれば良いでしょうか。文字列の一致を見る結合処理では、こうした結合処理の不備は現実の作業でも頻繁に生じます。原因の多くは今回のような表記揺れ、誤字脱字です。

結合に失敗した「Miyagi Ken」が他の県と 2 点異なります。まず県名と県名の末尾をつなぐハイフン（-）がないこと、そして他の県が先頭文字以外は小文字であるのに対して、大文字が 2 箇所出現している点です。

上記の問題に対して、mutate()で値の書き換えを行います。次のチャンクコードの結果では正しく 47 県の情報が失われることなく結合処理が完了します。

```
# 選挙データの加工4/5
ne_jpn_kanji <-
  ne_jpn %>%
  mutate(gn_name = recode(gn_name,
                          # ローマ字表記の規則を他県と合わせる
                          `Miyagi Ken` = "Miyagi-ken")) %>%
  inner_join(jpnprefs,
             by = "gn_name") %>%
  select(iso_3166_2, prefecture = prefecture_kanji, gn_name)

# 行数を確認
nrow(ne_jpn_kanji)
```

```
## [1] 47
```

いろいろな苦労がありましたが、ようやく地図データと選挙データの紐付けが行える状態となりました。このデータを使って都道府県別の選挙結果の地図が描画できるようになります。

```
# 選挙データの加工 5/5
ne_jpn_shugiin48 <-
  ne_jpn_kanji %>%
  left_join(df_shugiin48_party_votes_tops, by = "prefecture") %>%
  relocate(geometry, .after = last_col())
```

地理空間データの集約

選挙データ並びに日本地図ポリゴンはいずれも都道府県単位ですが、例えば関東、東北といった地方別に集約する機会を想定します。こうした地理空間データの規模の変更、データの集約方法を説明していきます。

ここでは先ほど中間オブジェクトとして作成した df_shugiin48_party_votes_long を起点にしてデータを作っていきます。まずは都道府県ごとに集計されたこのデータから、党派ごとに地方単位での集計データを作ります。

```
df_shugiin48_party_votes_region <-
  df_shugiin48_party_votes_long %>%
  left_join(zipangu::jpnprefs %>%
              select(prefecture = prefecture_kanji, region),
    by = "prefecture") %>%
  group_by(region, party) %>%
  summarise(votes = sum(votes),
            .groups = "drop") %>%
  filter(!is.na(votes))

head(df_shugiin48_party_votes_region)
```

```
## # A tibble: 6 × 3
##   region   party        votes
##   <chr>    <chr>        <dbl>
## 1 Chubu    自由民主党 4694971
## 2 Chubu    日本共産党  683237
## 3 Chugoku  自由民主党 1830445
## 4 Chugoku  日本共産党  287845
## 5 Hokkaido 希望の党    253108
## 6 Hokkaido 公明党       96795
```

ここでも都道府県とつながる地方名の情報を zipangu::jpnprefs から得ることにました。漢字表記の都道府県名をキーとして 2 つのデータを結合。その後の処理が集約のための手続きとなります。具体的には group_by()と summarise()の 2 つ

の関数を組み合わせて処理します。group_by()は、後に続く処理（ここではsummarise()）の適用対象となるグループを指定します。コードではregion、partyを与えて、次の集約関数summarise()に地方、党派ごとに処理する命令を宣言しています。処理内容は得票数（votes）を合算する、です。これにより党派ごとに地方単位での集計データが出来上がります。

地方別の集約データは地図データの方でも用意する必要があります。幸いなことにsfとdplyrの相性は良く、前述のgroup_by()やsummarise()はsfオブジェクトにおいても同等に機能します。ただしsfオブジェクトの場合、summarise()での集約処理とは別に地理空間データの集約化も行われる点で異なります。

次のチャンクコードを実行すると、都道府県のポリゴンを集約した地方別のポリゴンデータが生成されます。

```
ne_jpn_region <-
  ne_jpn_kanji %>%
  left_join(zipangu::jpnprefs %>%
              select(prefecture = prefecture_kanji, region),
            by = "prefecture") %>%
  group_by(region) %>%
  summarise(.groups = "drop")

# geometry に格納されるのは地方のポリゴンデータとなります
glimpse(ne_jpn_region)
```

```
## Rows: 8
## Columns: 2
## $ region   <chr> "Chubu", "Chugoku", "Hokkaido", "Kansai", "Kanto", "K
yushu", …
## $ geometry <GEOMETRY [°]> MULTIPOLYGON (((138.7157 35..., MULTIPOLYGON
 (((133.…
```

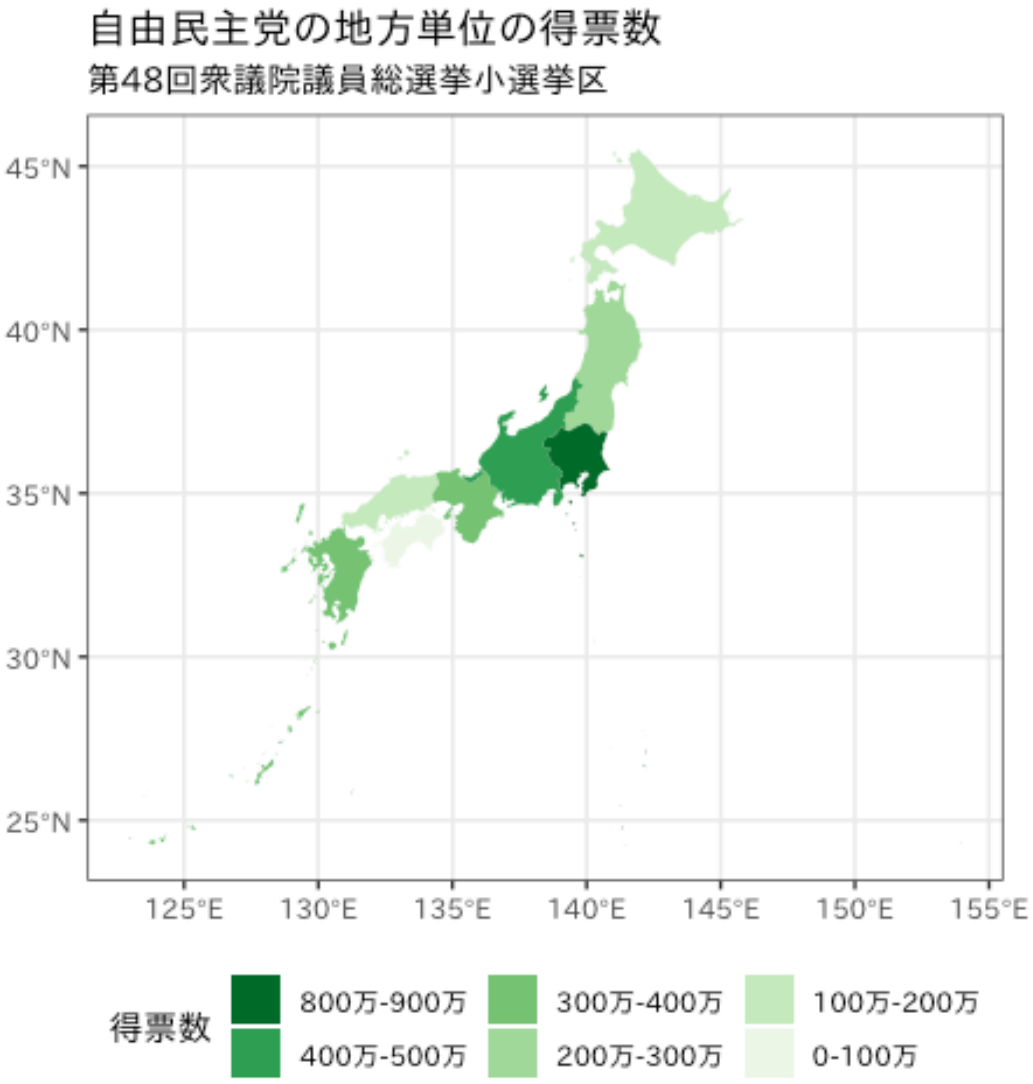
地図データと属性データの結合方法は先ほどと同じです。結合する2つのデータセットに共通の列regionがあるため、特別な処理は不要です。

```
ne_jpn_shugiin48_region <-
  ne_jpn_region %>%
  left_join(df_shugiin48_party_votes_region,
            by = "region")
```

結果を確認してみます。以下のチャンクコードは党派の中から自由民主党のデータを抽出し、地方単位での階級区分図を作成します。

```
ne_jpn_shugiin48_region %>%
  filter(party == "自由民主党") %>%
```

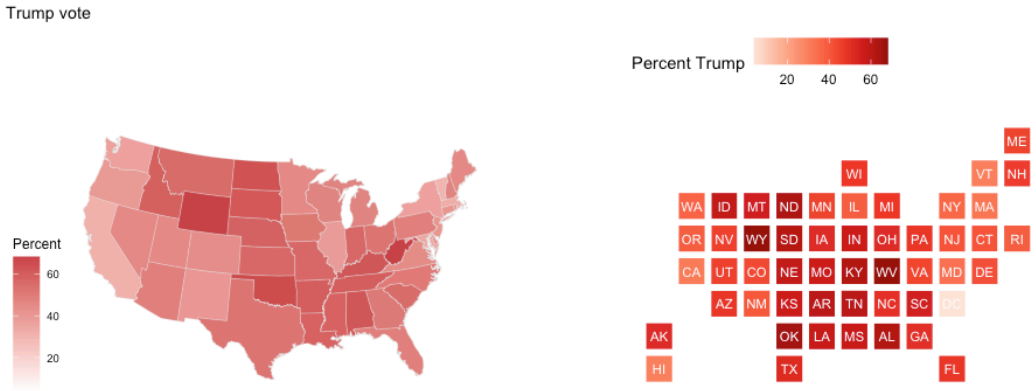
```
  mutate(votes = case_when(
    between(votes, 0, 1000000) ~ "0-100 万",
    between(votes, 1000001, 2000000) ~ "100 万-200 万",
    between(votes, 2000001, 3000000) ~ "200 万-300 万",
    between(votes, 3000001, 4000000) ~ "300 万-400 万",
    between(votes, 4000001, 5000000) ~ "400 万-500 万",
    between(votes, 5000001, 6000000) ~ "500 万-600 万",
    between(votes, 6000001, 7000000) ~ "600 万-700 万",
    between(votes, 7000001, 8000000) ~ "700 万-800 万",
    between(votes, 8000001, 9000000) ~ "800 万-900 万",
    between(votes, 9000001, Inf) ~ "900 万以上",
  )) %>%
  ggplot() +
  geom_sf(aes(fill = votes), color = "transparent") +
  scale_fill_brewer(palette = "Greens",
                    guide = guide_legend(title = "得票数",
                                         reverse = TRUE)) +
  labs(title = "自由民主党の地方単位の得票数",
       subtitle = "第 48 回衆議院議員総選挙小選挙区") +
  theme(legend.position = "bottom")
```



自由民主党の地方単位の得票数

空間配置を考慮したグラフ

教科書では典型的な地図表現とは別に地理的な空間配置を考慮したグラフが紹介されます。アメリカ合衆国の州を一定の大きさのグリッドまたはポイント形式で表現する statebins パッケージは州の面積、位置関係の正しさを許容し、各州を均等な表示となるようにします。これにより、面積の小さな州のデータを見落とす心配がなくなる利点があります。次の図は、アメリカ合衆国で行われた大統領選挙のデータを用いたものです。どちらの図もドナルド・トランプへの得票率を州ごとに塗り分けた図となっています。

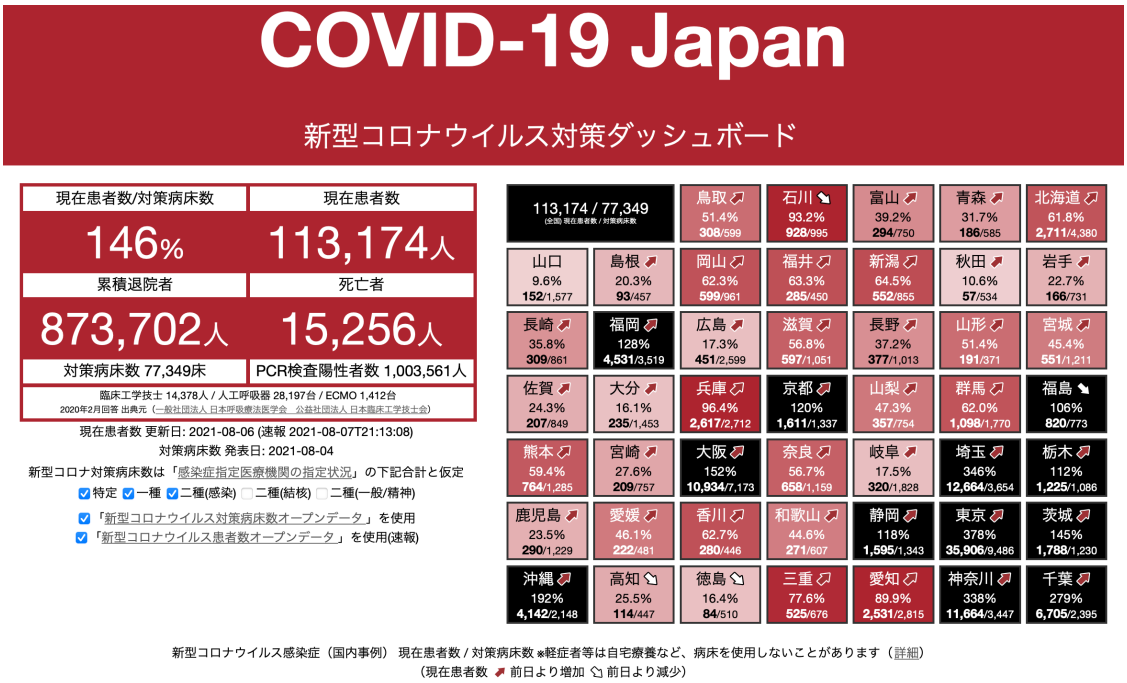


アメリカ合衆国大統領選挙の結果

tabularmaps

アメリカ合衆国の地図では statebins が使えますが、日本で同様の表現を行うには tabularmaps パッケージが役立ちます。tabularmaps は「カラム地図」と呼ばれる行政区域を表状に配置する表現を可能にするパッケージです。この表現を用いれば、日本の都道府県や東京 23 区をはじめ、任意の行政区域の空間的な配置を考慮した可視化ができます。

カラム地図の利用例として、新型コロナウイルス対策ダッシュボードがあります。都道府県の面積によらず、各県の様子を一望でき、単純な地図表現よりも比較に優れます。



カラム地図の利用例 - 新型コロナウイルス対策ダッシュボード
https://www.stopcovid19.jp

tabularmaps は ggplot2 をベースとしたパッケージですが statebins と同じく geom_*()ではない、専用の描画関数を提供します。tabularmap()に表示するデータ、配置する位置を指定する列を指定して可視化が行われます。

```
library(tabularmaps)

jpn77 %>%
  select(jis_code,
         prefecture = prefecture_kanji,
         x,
         y) %>%
  left_join(df_shugiin48_party_votes_tops,
            by = "prefecture") %>%
  tabularmap(x = x,
             y = y,
             group = jis_code,
             label = prefecture,
             fill = prop,
             size = 2,
             family = "IPAexGothic") +
  scale_fill_viridis_b() +
  theme_tabularmap(base_family = "IPAexGothic")
```

参考資料

- Robin Lovelace, Jakub Nowosad, Jannes Muenchow (2019). Geocomputation with R. CRC Press.
- 松村 優哉,湯谷 啓明,紀ノ定 保礼,前田 和寛 (2021). 改訂 2 版 R ユーザのための RStudio 実践入門のおすすめポイント. 技術評論社.

発表時の動作環境

R version 4.0.5 (2021-03-31)

Platform: x86_64-apple-darwin17.0 (64-bit)

locale: ja_JP.UTF-8||ja_JP.UTF-8||ja_JP.UTF-8||C||ja_JP.UTF-8||ja_JP.UTF-8

attached base packages:

- grid
- stats
- graphics
- grDevices
- utils
- datasets
- methods
- base

other attached packages:

- statebins(v.1.4.0)
- socviz(v.1.2)
- patchwork(v.1.1.1)
- rnaturalearth(v.0.1.0)
- sf(v.1.0-2)
- ggplot2(v.3.3.5)
- dplyr(v.1.0.7)

loaded via a namespace (and not attached):

- Rcpp(v.1.0.7)
- here(v.1.0.1)
- lattice(v.0.20-44)
- tidyr(v.1.1.3)
- class(v.7.3-19)

- assertthat(v.0.2.1)
- rprojroot(v.2.0.2)
- digest(v.0.6.27)
- prompt(v.1.0.1)
- utf8(v.1.2.2)
- R6(v.2.5.1)
- evaluate(v.0.14)
- e1071(v.1.7-8)
- highr(v.0.9)
- pillar(v.1.6.2)
- rlang(v.0.4.11)
- rstudioapi(v.0.13)
- rnaturalearthhires(v.0.2.0)
- rmarkdown(v.2.10)
- labeling(v.0.4.2)
- pander(v.0.6.4)
- readr(v.2.0.1)
- stringr(v.1.4.0)
- munsell(v.0.5.0)
- proxy(v.0.4-26)
- compiler(v.4.0.5)
- xfun(v.0.25)
- pkgconfig(v.2.0.3)
- rgeos(v.0.5-5)
- htmltools(v.0.5.2)
- tidyselect(v.1.1.1)
- tibble(v.3.1.4)
- fansi(v.0.5.0)
- crayon(v.1.4.1)
- tzdb(v.0.1.2)
- withr(v.2.4.2)
- wk(v.0.5.0)
- gtable(v.0.3.0)
- lifecycle(v.1.0.0)
- DBI(v.1.1.1)
- magrittr(v.2.0.1)
- units(v.0.7-2)
- scales(v.1.1.1)

- KernSmooth(v.2.23-20)
- cli(v.3.0.1)
- stringi(v.1.7.4)
- mapproj(v.1.2.7)
- farver(v.2.1.0)
- sp(v.1.4-5)
- ellipsis(v.0.3.2)
- generics(v.0.1.0)
- vctrs(v.0.3.8)
- zipangu(v.0.2.3)
- RColorBrewer(v.1.1-2)
- s2(v.1.0.6)
- tools(v.4.0.5)
- glue(v.1.4.2)
- purrr(v.0.3.4)
- maps(v.3.3.0)
- hms(v.1.1.0)
- fastmap(v.1.1.0)
- yaml(v.2.2.1)
- colorspace(v.2.0-2)
- classInt(v.0.4-3)
- knitr(v.1.33)