Основы программирования в R

Работа с картами из shape-файлов в R

Алла Тамбовцева, НИУ ВШЭ

Содержание

Shape-файлы

На этом занятии мы обсудим, как работать со статическими картами — картами, которые загружаются из файлов (без подключения к сервисам типа Яндекс-карты или Google Maps) и не поддерживают интерактив (нельзя уменьшать/увеличивать определенные области, двигать карту и прочее).

Один из широко распространенных форматов хранения географической информации – shape-файл. В этом файле хранятся географические (а точнее, геометрические) объекты разных типов: точки, линии, многоугольники и другие. Несмотря на то, что shape-файл является основным файлом, нужным для построения карты, одного его недостаточно: рядом с ним всегда есть необходимые вспомогательные файлы. В shp-файле хранится информация о геометрических объектах, в dbf-файле – об их атрибутах, а shx-файл случит для связи между первыми двумя. Наверное, эта информация выглядит сейчас немного туманно, но главное следующее:

- файлы с расширениями `.shp`, `.dbf` и `.spx` должны храниться в одной папке, не нужно их разделять;
- хотя для работы с картами мы будем загружать в R shp-файл, остальные файлы выбрасывать не нужно.

Чтобы понять, как выглядят нужные для построения карт файлы, скачаем набор файлов для России с сайта *Global Administrative Areas*. (Еще ресурсы, где можно найти файлы для карт России бесплатно: GIS-Lab, GisGeo). Так как уровень детализации может быть разный (вся страна, регионы или районы), файлы в папке тоже разные (`_0`, `_1`, `_2`, `_3`). Нам нужен файл `gadm36_RUS_2.shp` с детализацией по районам.

Загрузка shape-файла

Перед работой нам необходимо установить библиотеку `rgdal` для работы с shape-файлами и библиотеки `classInt` и `RColorBrewer` для разбиения выборки на группы и выбора палитры.

```
install.packages("rgdal")
install.packages("classInt")
install.packages("RColorBrewer")
```

Подгрузим эти библиотеки, а также `tidyverse`:

```
library(rgdal)
library(classInt)
library(RColorBrewer)
library(tidyverse)
```

Для удобства сделаем папку со всеми файлами с картами рабочей через `setwd()` или через меню *Session — Set working directory — Choose directory*, а потом загрузим shape-файл через функцию `readOGR()`:

```
setwd('/Users/allat/Desktop/gadm36_RUS_shp')
ru <- readOGR('gadm36_RUS_2.shp')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/allat/Desktop/gadm36_RUS_shp/gadm36_RUS_2.shp", layer: "gadm36_RUS_2"
## with 2445 features
## It has 13 fields
```

Если текст на кириллице не считался корректно, попробуйте добавить отдельные опции с кодировкой:

```
ru <- readOGR('gadm36_RUS_2.shp', use_iconv = TRUE, encoding = "UTF-8")
```

Если посмотрим на объект `ru`, заметим, что он похож одновременно и на список, и на датафрейм, на самом деле это *SpatialPolygonsDataFrame*, датафрейм с отдельным «слоем» пространственных данных:

```
View(ru)
```

Но отдельные столбцы из такого датафрейма можно выбирать по-прежнему через `$`. Например, посмотрим на названия регионов на латинице и на кириллице:

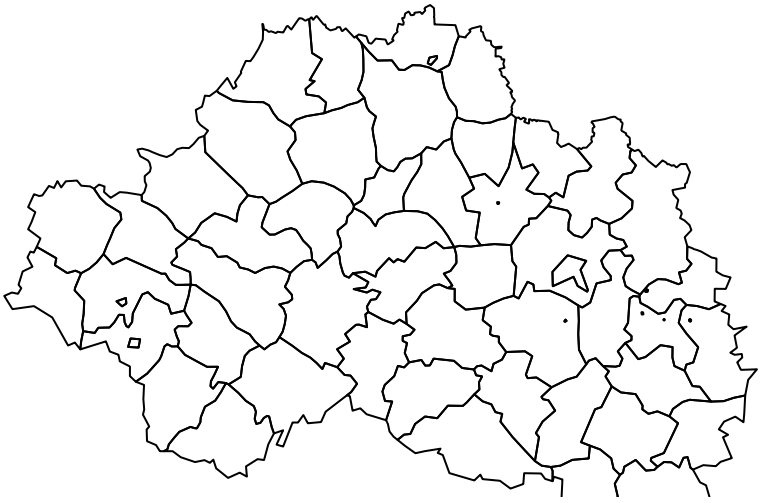
```
unique(ru$NAME_1)
unique(ru$NL_NAME_1)
```

Выберем Калужскую область и будем работать только с ней. Сохраним выбранные строки в `kaluga_map` и построим карту:

```
kaluga_map <- ru[ru$NAME_1 == 'Kaluga', ]
plot(kaluga_map)
```

Если бы мы хотели построить карту по нескольким соседним областям, можно было бы отфильтровать все нужные строки и построить графики друг за другом, добавив опцию `add = TRUE` в функции `plot()`:

```
tula_map <- ru[ru$NAME_1 == 'Tula', ]
plot(kaluga_map)
plot(tula_map, add = TRUE)  # если хотим более 1 региона
```



Примечание: можно было бы просто сформулировать сложное условие при выборе строк датафрейма (выбрать несколько регионов через `|`) и использовать `plot()` один раз.

Загрузка данных для раскраски

Карту Калужской области мы построили не просто так, а для того, чтобы визуализировать распределение какого-нибудь показателя по районам. Загрузим csv-файл с данными по численности населения, выгруженными с сайта Росстат:

```
kaluga_dat <- read.csv("https://allatambov.github.io/rprog/data/Report.csv",
                       sep = ";",
                       encoding="UTF-8")
```

Удалим строки с пропущенными значениями (из-за особенностей представления данных на сайте Росстата названия районов дублируются):

```
kaluga_dat <- na.omit(kaluga_dat)
```

Посмотрим на уникальные значения названий районов:

```
unique(kaluga_map$NL_NAME_2)
```

Теперь нужно как-то объединить карту и данные. Для того, чтобы это стало возможным, нужно, чтобы районы в двух датафреймах имели одинаковые названия. Кроме того, так как склеивать «географический» и «демографический» датафреймы мы будем по столбцу с названиями районов, нужно, чтобы этот столбец тоже был назван одинаково в двух датафреймах.

Для начала сравним наборы районов в карте и данных с Росстата:

```
length(kaluga_map$NL_NAME_2)
```

```
## [1] 28
```

```
length(kaluga_dat$district)
```

```
## [1] 26
```

В данных Росстата на два района меньше: статистика приведена по муниципальным районам, в отличие от карты не вынесены отдельно Киров и Людиново. Теперь сравним названия:

```
unique(kaluga_map$NL_NAME_2)
unique(kaluga_dat$district)
```

Чтобы получить одинаковые названия районов (в качестве ориентира берем названия из датафрейма с геоданными), нам нужно выполнить следующее:

- убрать слово «муниципальный» (учесть написание как с маленькой буквы, так и с большой);
- заменить `Городской округ "город Обнинск"` на `Обнинск` и `Городской округ "город Калуга"` на `Калуга (горсовет)`;
- заменить `Малоярославецкий район` на `Малоярославецкий`;
- после удаления слова «муниципальный» убедиться, что в названиях нет лишних пробелов — заменить последовательности из двух и более пробелов на один пробел.

Для замены одних строк на другие воспользуемся функцией `str_replace()` из `tidyverse`, плюс, при необходимости задействуем регулярные выражения. В `str_replace()` на первом месте указываем, где производим замену, на втором — что заменяем, на третьем — на что заменяем.

```
kaluga_dat <- kaluga_dat %>%
  mutate(district = str_replace(district, "[Мм]униципальный", ""),
         district = str_replace(district, "\\s{2,}", " ")) %>%
  mutate(district = str_replace(district,
                                'Городской округ "город Калуга"', 'Калуга (горсовет)'),
         district = str_replace(district, 'Городской округ "город Обнинск"', 'Обнинск'),
         district = str_replace(district, 'Малоярославецкий район', 'Малоярославецкий'))
```

Пояснения к регулярным выражениям:

- `[Мм]униципальный`: в `[]` указываем или один символ (`М`), который надо найти, или другой (`м`).
- `\\s{2,}`: здесь `\\s` — это символ пробела (*s* — от *space*), далее в фигурных скобках указано число раз, минимум два раза, максимум любой.

Теперь осталось добавить две строки к датафрейму `kaluga_dat`, одну строку для города Киров, одну строку для города Людиново. Так как данных по этим городам по отдельности у нас нет, припишем им значения численности населения в Кировском и Людиновском районе соответственно. Воспользуемся функцией `add_row()`:

```
kaluga_dat <- kaluga_dat %>%
  add_row(district = "Киров", X2019 = 40307) %>%
  add_row(district = "Людиново", X2019 = 41784)
View(kaluga_dat)
```

Остался последний штрих — переименовать первый столбец в `kaluga_dat` в `NL_NAME_2`, чтобы названия столбцов с районами в датафрейме с геоданными и в датафрейме с данными Росстата были одинаковыми:

```
colnames(kaluga_dat)[1] <- "NL_NAME_2"
```

Теперь все готово к склеиванию двух датафреймов. Преобразуем `kaluga_map` в обычный датафрейм (пока это *SpatialPolygonsDataFrame*) и применим функцию `merge()`:

```
kaluga_map_df <- as.data.frame(kaluga_map)

# по какому столбцу склеиваем - в by
full <- merge(kaluga_map_df, kaluga_dat, by = "NL_NAME_2")
```

Готово! Теперь перейдем к выбору палитры для раскрашивания карты.

Группировка данных и выбор палитры

Мы хотим нанести на карту данные по численности населения в районах: чем темнее цвет района на карте, тем больше численность населения в нем. Вопрос: каким образом мы будем разбивать значения численности населения, чтобы получать районы разных оттенков? Один из вариантов такой — разбивать

по процентилям. Упорядочить все значения по возрастанию и объединить их в группы: те, что входят в первые 20% — это первая группа (районы закрашиваются самым светлым цветом), во вторые 20% — это вторая группа (районы закрашиваются цветом потемнее), и так далее.

Разобьем значения численности населения на 5 групп. Для этого нам понадобится функция `classIntervals()` из библиотеки `classInt`. В аргументе `style` можно выставить что-то более интересное, например, кластеризацию методом k-средних, но мы пока оставновимся на квантилях.

```
brks <- classIntervals(full$X2019, n = 5, style = "quantile")
```

Посмотрим на `brks`:

```
brks
```

```
## style: quantile
##   one of 12,650 possible partitions of this variable into 5 classes
##    [5946,9190.4)   [9190.4,13198)    [13198,25621)  [25621,46472.4)
##                6               5                6               5
## [46472.4,353540]
##                6
```

```
str(brks)
```

```
## List of 2
##  $ var : num [1:28] 18306 5946 62711 52799 13881 ...
##  $ brks: num [1:6] 5946 9190 13198 25621 46472 ...
##  - attr(*, "style")= chr "quantile"
##  - attr(*, "nobs")= int 26
##  - attr(*, "call")= language classIntervals(var = full$X2019, n = 5, style = "quantile")
##  - attr(*, "intervalClosure")= chr "left"
##  - attr(*, "class")= chr "classIntervals"
```

Объект `brks` хранит границы интервалов разбиения на группы. Выберем пороговые значения для выделения групп и сохраним их в `breaks`:

```
breaks <- brks$brks
length(brks) # их 6, то есть на 1 больше, чем самих групп
```

```
## [1] 2
```

Теперь воспользуемся функцией `brewer.pal()` из библиотеки `RColorBrewer` и выберем палитру — пять оттенков красного цвета от более светлого к более темному.

```
colors <- brewer.pal(5, "Reds")
colors
```

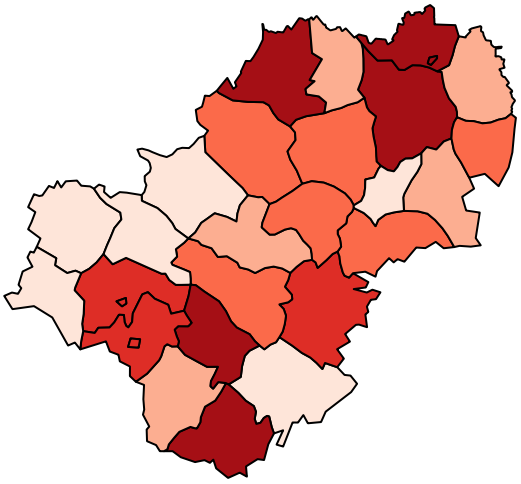
```
## [1] "#FEE5D9" "#FCAE91" "#FB6A4A" "#DE2D26" "#A50F15"
```

Добавим в датафрейм с геоданными столбец `X2019` с численностью населения.

```
kaluga_map$X2019 <- full$X2019
```

Теперь построим карту с учетом выбранных цветов:

```
plot(kaluga_map,
     col = colors[findInterval(full$X2019, breaks,
                               all.inside = TRUE)],
     axes = FALSE)  # убираем оси, на карте ни к чему
```



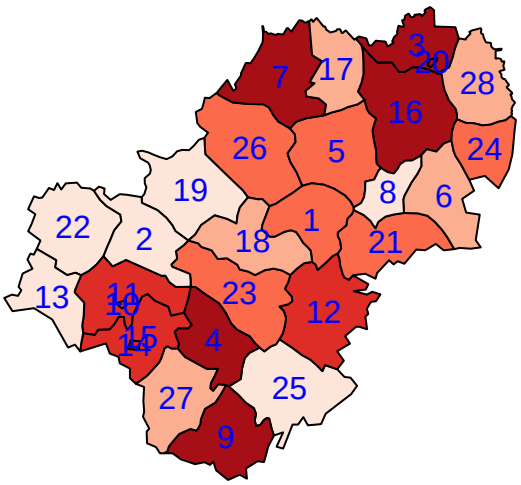
Пояснения к коду выше:

- Функция `findInterval()` определяет, к какой группе из 5 относится тот или иной район.
- Когда мы указываем набор номеров групп из `findInterval()` в квадратных скобках после `colors`, за каждой группой закрепляется определенный оттенок цвета.

Дополнительные настройки

Построенная нами карта выглядит красиво. Но есть две проблемы: нет названий районов и легенды, поясняющей, чему соответствуют цвета на карте. Исправим: подпишем номера районов с помощью `text()` .

```
plot(kaluga_map,
     col = colors[findInterval(full$X2019, breaks,
                               all.inside = TRUE)],
     axes = FALSE)
text(coordinates(kaluga_map), labels = 1:28, col = "blue")
```



Пояснения к коду выше:

Функция `text()` добавляет любые подписи внутри графика. В начале указываются координаты, куда эти подписи поместить (здесь геометрические центры районов на карте, получены с помощью функции `coordinates()`). В `labels` перечисляются сами подписи.

Теперь вроде бы все хорошо. Но не очень: номера районов плохо видны на темно-красном цвете. Можно, конечно, просто поиграть с цветами и подобрать какой-то более удачный цвет, но можно сделать все более концептуально: выделять текст на темном фоне светлым цветом, а на более бледном – темным. Предлагаю подумать об этом сейчас немного и написать код, который бы создавал вектор цветов, в котором столько элементов, сколько районов элементы могут принимать только два значения `"white"` (для темных районов) и `"black"` (для светлых районов).

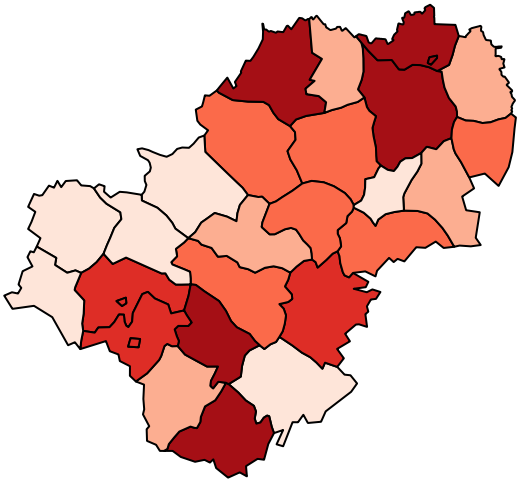
Сохраним в `coldrops` числовые метки групп (от 1 до 5), которые нам возвращает функция `findInterval`. Мы знаем, что чем больше число, тем темнее цвет на карте. Сохраним две самые «темные» метки в вектор `dark`, а потом с помощью `ifelse()` будем записывать значение `"white"`, если числовая метка группы есть в векторе `dark`, и `"black"`, если ее там нет.

```
coldrops <- findInterval(full$X2019, breaks,
             all.inside = TRUE)
dark <- c(4, 5)
coltext <- ifelse(coldrops %in% dark, "white", "black")
```

Построим карту с обновленными названиями районов. А заодно посмотрим, как можно сохранить карту в файл (вообще это касается любых графиков, не только карт). Понятно, что можно просто сделать это вручную: во вкладке *Plots* выбрать *Export* и экспортировать картинку в файл нужного формата. Но иногда этот способ не подходит: бывает, что картинка при ручном экспорте деформируется, особенно, если она большая. Поэтому посмотрим, как сохранять картинку с помощью кода.

```
# строим карту - основной слой

plot(kaluga_map,
     col = colors[findInterval(full$X2019, breaks,
                               all.inside = TRUE)],
     axes = FALSE)
```



```
# открываем файл map.pdf
# фиксируем ширину и высоту (R работает с дюймами)

dev.copy(pdf, "map.pdf", width = 15, height = 10)
```

```
## pdf
##   3
```

```
# достраиваем остальные слои граифка

text(coordinates(kaluga_map), labels = 1:28, col = coltext, cex = 0.7)
```

```
# обязательно закрываем файл pdf
# иначе он будет пустым, изменения не сохранятся

dev.off()
```

```
## pdf
##   2
```

Осталось добавить легенду графика. Давайте добавим две легенды: одна будет пояснять, каким значениям показателя соответствует тот или иной цвет, а другая — какой район скрывается за тем или иным кодом от 1 до 28.

```
plot(kaluga_map,
     col = colors[findInterval(full$X2019, breaks,
                               all.inside = TRUE)],
     axes = FALSE)

text(coordinates(kaluga_map), labels = 1:28, col = coltext, cex = 0.7)

# легенда с цветами в нижнем левом углу

legend(x = 'bottomleft',
       legend = c('5946-8852',
                  '8852-13040',
                  '13040-21113',
                  '21113-41784',
                  '41784-353540'),
       fill = colors,
       title = 'Population')

# легенда с районами в нижнем правом углу

legend(x = 'bottomright',
       legend = paste(1:28, full$NAME_2),
       title = 'District')
```