

Национальный исследовательский университет «Высшая школа экономики»

КУРСОВАЯ РАБОТА
Визуализация экономических и статистических данных на
картах с помощью R

Выполнил: А. А. Кондрашов
Факультет: МИЭФ
Курс: 2
Научный руководитель: Б. Б. Демешев

Москва
2016

Содержание

1 Введение	2
2 Подготовка	3
2.1 Среда R	3
2.2 GIS	4
2.3 Дополнительные настройки	7
3 Получение карты	9
3.1 Построение карты с помощью готового шаблона	9
3.1.1 Пример получения карты с GISLab	10
3.1.2 Построение карты в формате geoJSON	17
3.1.3 Конвертация файлов геоданных	23
3.2 Работа в QGIS	24
3.2.1 Получение данных OSM	24
3.2.2 Рисование векторных слоёв	36
3.2.3 Получение административных границ регионов	44
3.2.4 Упрощение карт	49
4 Визуализация экономических данных	54
4.1 Построение карт с градиентом	54
4.2 Построение карт с точечными данными	55
4.3 Построение карт с уровнями	57
4.4 Сравнение двух регионов	57
4.5 Сравнение наборов точечных данных	59
4.6 Отображение изменений на картах	63
4.7 Построение интерактивных карт	66
4.7.1 Создание карт с leaflet	66
4.7.2 Создание карт с plotly	68
5 О пакете rusmaps	72
6 Заключение	75
7 Приложение 1	76
8 Приложение 2	76

1 Введение

Язык программирования R уже давно зарекомендовал себя в качестве инструмента (и даже среды¹) для работы с данными. На сегодняшний день, это один из самых используемых ЯП для этих целей: среди пользователей R есть как крупные компании² (Bank of America, Facebook, Ford, Microsoft, Kickstarter и другие) и университеты, так и обычные люди, у которых есть необходимость провести исследование(я). Большим плюсом использования R в научной деятельности является его интеграция с TeX, что позволяет создавать полноценные научные публикации (в том числе и эту курсовую работу) используя средства как R, так и TeX, без необходимости работать в отдельных программах для верстки текста и написания программ. Также не стоит забывать, что R — свободно распространяемое программное обеспечение, что делает его доступным такому широкому кругу лиц.

Данная работа предназначена помочь начинающим и опытным исследователям в области экономики и статистики, использующим язык R для анализа данных. Возможности этого языка программирования в сфере обработки статистических и экономических данных можно назвать безграничными, поскольку спектр этих возможностей настолько широк, что вряд ли найдётся исследователь, которому понадобятся все эти возможности. Если же какие-то из необходимых инструментов отсутствуют или же, по мнению пользователя среди R, недостаточно тривиальны и нуждаются в упрощении, то он может создать собственные функции и даже пакет, предназначенный для решения какой-либо конкретной задачи, поскольку R - достаточно мощный язык программирования.

Среди ключевых возможностей R можно отметить проведение статистических тестов, работу с матрицами и массивами разной размерности, линейное и нелинейное моделирование, анализ временных рядов и, что немаловажно, построение графиков и диаграмм. Визуализация данных в R занимает особое место: этой теме посвящено множество работ, существует большое количество курсов и пакетов, расширяющих возможности R в этой сфере. В частности, R способен работать и с пространственными данными, а также визуализировать их посредством картограмм. Этой области возможностей R и посвящена данная работа: в ней я собираюсь рассмотреть различные способы получения геоданных (я буду называть их картами) и способы отображения привязанной к ним информации, составив подробные инструкции. В качестве дополнительного (но не менее самостоятельного) материала к этой работе прилагается разработанный мною пакет **rusmaps**, содержащий карты России, на которых можно отображать различные данные. Подробнее с ним можно ознакомиться в секции 5.

¹Сообщество The R Foundation предпочитает называть R средой, а не инструментом <https://www.r-project.org/about.html>.

²Более полный список можно найти по ссылке: <http://www.revolutionanalytics.com/companies-using-r>

2 Подготовка

В данной секции я укажу необходимые элементы для работы, а также дам комментарии касательно их назначения.

2.1 Среда R

В первую очередь, для работы нам понадобится среда R. Последнюю версию можно получить, пройдя по ссылке: <https://cran.r-project.org>. Далее, нам понадобится редактор кода R, и лучше всего для этого подойдёт Rstudio (<https://www.rstudio.com>), прежде всего благодаря своему удобству и интуитивной понятности интерфейса.

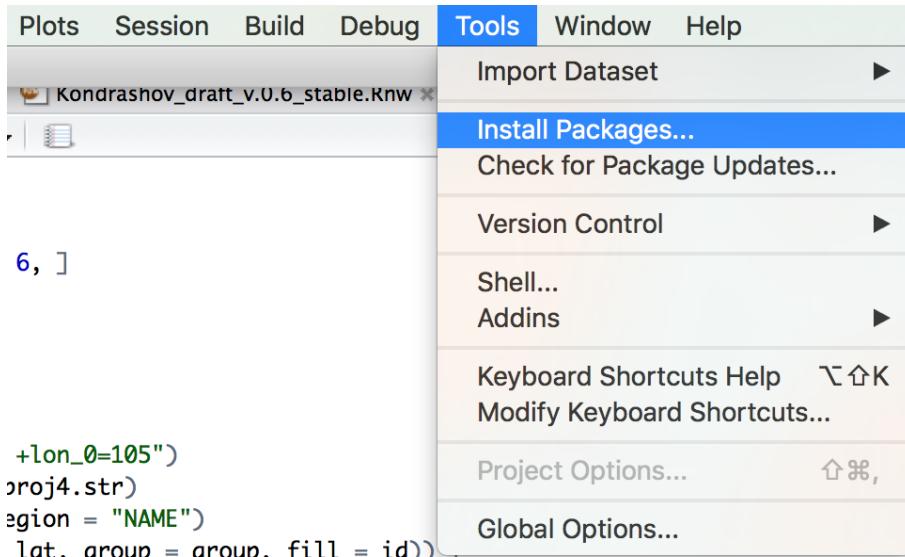
Код лучше писать в окне скриптов (source pane), так как это позволит его сохранить. Также напомню важное правило при работе с любым файлом на компьютере: сохраняйте файл спустя определённые интервалы времени. Это обезопасит часы труда в случае экстренного завершения программы.

В работе нам понадобятся следующие пакеты (некоторые пакеты будут подключены позже):

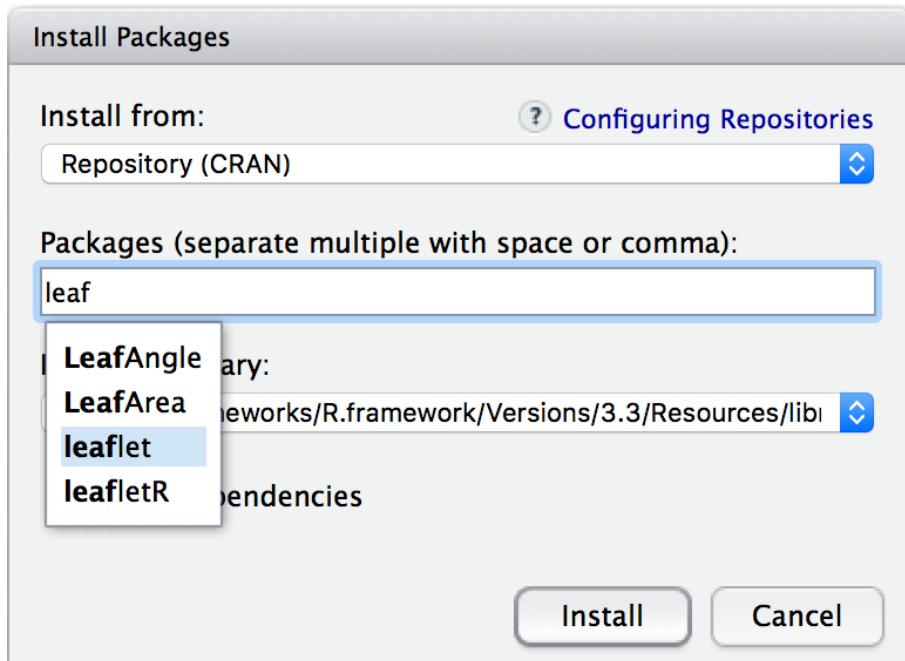
```
#пакеты для создания карт:  
library("ggplot2")  
library("ggmap")  
library("maptools")  
library("tmap")  
library("gridExtra")  
#пакеты для создания интерактивных карт  
library("leaflet")  
library("plotly")  
#пакеты для обработки данных  
library("rgdal")  
library("rgeos")  
library("dplyr")  
library("tidyverse")  
#пакет для вывода графиков и карт, содержащих кириллицу, в pdf  
library("Cairo")  
#пакет, увеличивающий разнообразие цветовых палитр  
library("RColorBrewer")
```

Если какой-либо из пакетов отсутствует, то его необходимо загрузить с помощью команды `install.packages("имя пакета")`, либо же, пройдя по вкладке:

Tools -> Install packages...



выбрать и установить пакеты самостоятельно.



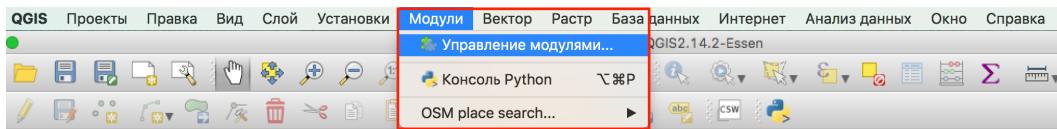
2.2 GIS

Для создания собственных shape-файлов, а также конвертации файлов геоданных разных форматов нам также понадобится QGIS (<http://www.qgis.org/ru/site/forusers/download.html>) или любая другая геоинформационная система. В данной работе, я буду использовать QGIS. Разумеется, перед тем, как начинать с ним работать, нужно его настроить под свои нужды.

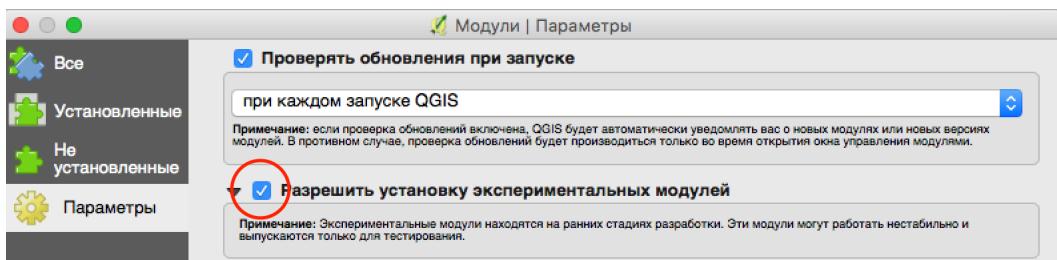
Нам понадобятся два модуля (плагина), которые можно загрузить, зайдя по вклад-

кам меню:

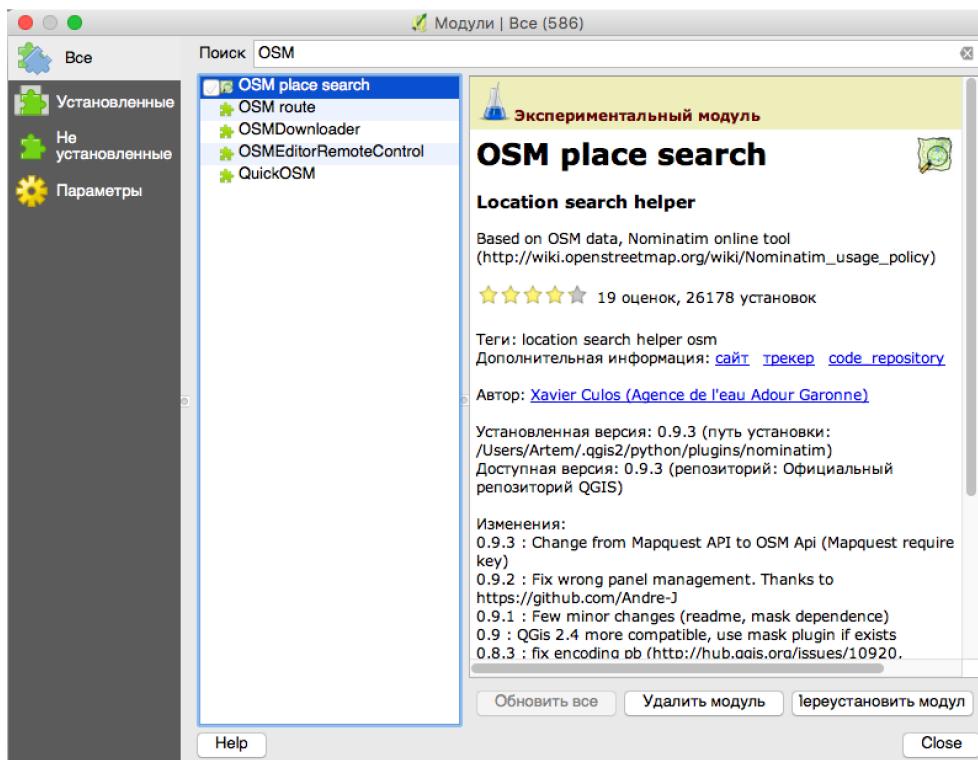
Модули -> Управление модулями...



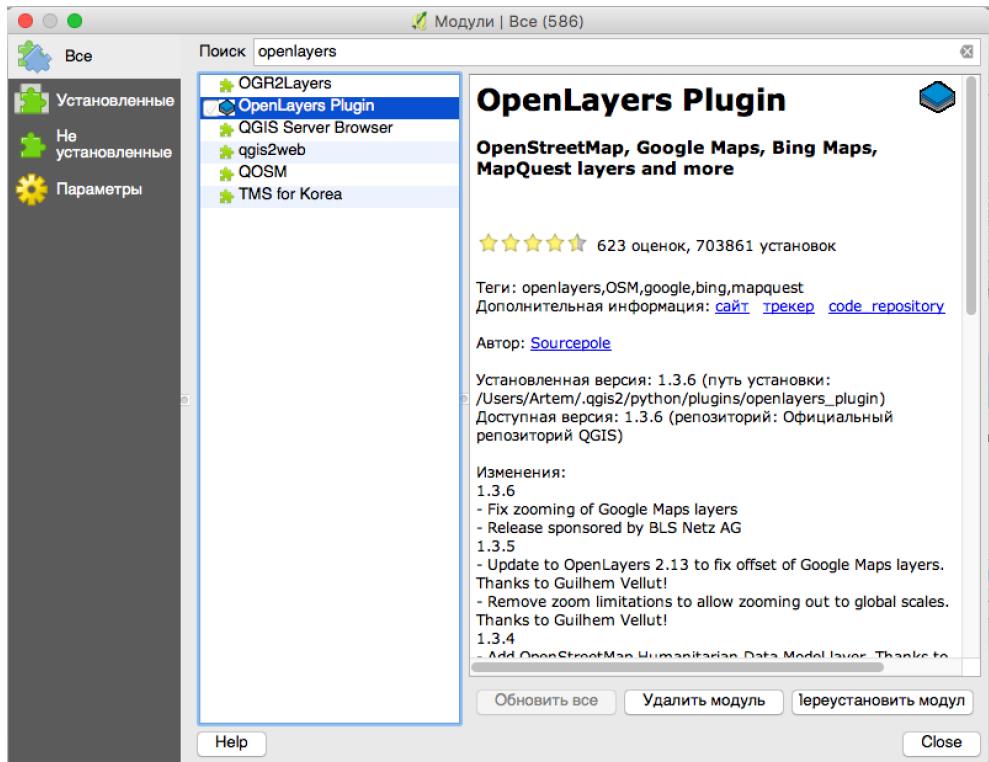
В открывшемся окне переходим на вкладку «Параметры» и ставим флажок у «Разрешить установку экспериментальных модулей», так как один из необходимых модулей входит в категорию экспериментальных.



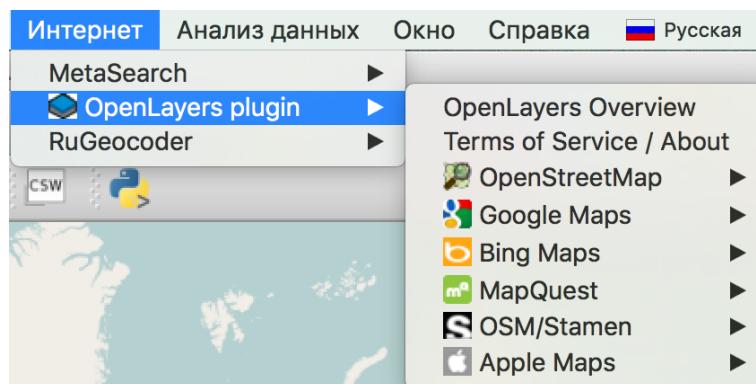
Наконец, перейдя на вкладку «Все», выбираем необходимые модули: **OSM Place Search** (экспериментальный)...



... и **OpenLayers**, введя первые несколько букв названия в поле поиска, и загружаем их. Первый нужен для того, чтобы находить интересующие нас районы (места) в системе OSM, в то время как второй загружает слои (что очевидно из названий плагинов).



Модуль **OpenLayers** можно найти в меню «Интернет»,



а **OSM Place Search** появится как панель. В случае, если Вы случайно её закроете, её (как и остальные панели) можно вернуть по вкладкам меню:

Вид -> Панели -> OSM Place Search...

На этом настройка QGIS завершена.

2.3 Дополнительные настройки

Поскольку данная работа включает в себя построение интерактивных карт, я решил добавить инструкцию для вывода карт, созданных с помощью **leaflet** в формат pdf. Если это Вам не нужно, можете смело пропустить эту подсекцию.

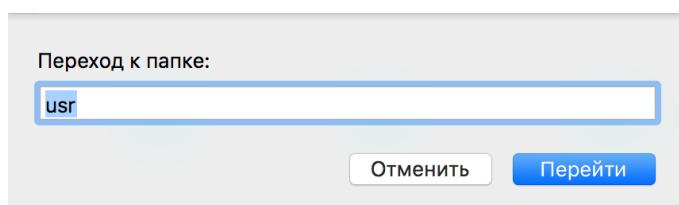
Для создания интерактивных карт с **leaflet**³, нам понадобится одноимённый пакет (его мы загрузили и установили в подсекции «Среда R»). Чтобы сохранить полученную карту в pdf (также можно выбрать png и jpeg), нам понадобится пакет **webshot**. Данный пакет требует ссылки на страницу, на которой нужно сделать скриншот, поэтому нам необходимо подключить пакет **htmlwidgets**, чтобы создать документ html, содержащий параметры объекта leaflet. На этот файл и будет ссылаться команда **webshot()**.

```
#пакеты для web-скриншотов (для работы требуется PhantomJS)
library("webshot")
library("htmlwidgets")
```

Для работы пакета **webshot**⁴ нам понадобится **PhantomJS** - консольный WebKit с интерфейсом **javascript**. Скачать его можно с официального сайта: <http://phantomjs.org/download.html>, а пользователи Windows могут воспользоваться встроенной в пакет **webshot** функцией `install_phantomjs(version = "2.1.1")`, которая установит PhantomJS версии 2.1.1 (самая свежая на момент написания данной работы). Если же Вы пользуетесь другой ОС, то придётся выполнить сборку (build). Инструкцию можно найти по ссылке <http://phantomjs.org/build.html>. Я же расскажу как установить PhantomJS для OSX (El Capitan на момент написания).

В первую очередь, необходимо скачать PhantomJS для OSX (ссылка выше). Распакуем полученный архив и в папке **bin** найдём искомый файл. Теперь нам нужно поместить его в системную папку, путь к которой `/usr/local/bin`. Так как эти папки системные, то они скрыты, и чтобы их проявить, нужно воспользоваться Терминатором. Но я предлагаю другой, более простой метод:

1. Перейдём на директорию жесткого диска (или SSD диска, в зависимости от конфигурации). Доступ к нему можно подключить через настройки Finder.
2. Введём комбинацию клавиш cmd+shift+G. Откроется окно перехода к другой папке, находящейся в данной директории, поиск «видит» скрытые папки (очень полезно когда в каталоге слишком много объектов). Введём «usr».

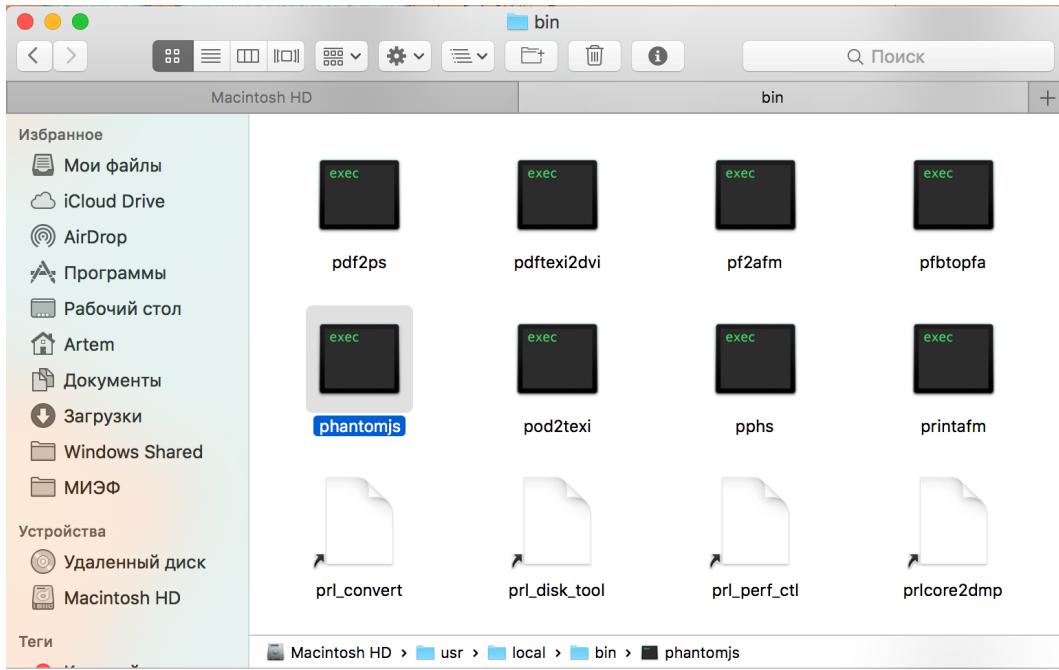


³Подробнее на <https://cran.r-project.org/web/packages/leaflet/leaflet.pdf>

⁴Подробнее на <https://cran.r-project.org/web/packages/webshot/webshot.pdf>

3. Далее проследуем по следующему пути:

usr/local/bin



4. Здесь и должен находиться файл phantomjs. Копируем (или переносим) его в эту папку (потребуется подтверждение паролем администратора). Готово!

Теперь можно экспортовать скриншоты leaflet-карт в pdf.

3 Получение карты

Для визуализации пространственных данных необходима картограмма, которую можно построить с помощью shape-файлов. Shapefile⁵ — векторный формат для хранения объектов, содержащих географические данные, которые представлены в виде точек, линий или полигонов, а также соответствующие каждому объекту атрибуты (название, температура, площадь и т.д.). Каждый файл хранит объекты только одного типа, то есть в одном файле не могут храниться точки и линии одновременно. Необходимо отметить, что для работы с объектом(ами) необходимы по крайней мере три файла (.shp, .shx, .dbf), поскольку они содержат разные данные об объекте. Основным файлом является .shp — в нем хранится информация о геометрических объектах, далее следуют .dbf (файл базы данных — атрибутов) и .shx — индексный файл. Таким образом, shape-файл является готовым шаблоном для картограммы с привязанной таблицей с данными для анализа по регионам (точкам).

В данной работе мы рассмотрим два способа получения shape-файла: использование различных интернет-ресурсов и создание собственного shape-файла в QGIS.

3.1 Построение карты с помощью готового шаблона

В Интернете можно найти большое количество уже готовых shape-файлов, предназначенных для исследований разного рода. Как правило, нас будут интересовать файлы, содержащие полигоны, то есть описывающие территории (регионы) и как по ним распределены те или иные объекты изучения.

В качестве ресурсов, на которых можно найти готовые shape-файлы (а также файлы геоданных в других форматах, например, GeoJSON, речь о котором пойдет в другой подсекции), а также помочь для их редактирования, можно привести следующие сайты:

- GISLAB

На данном сайте в разделе «Геоданные» можно найти несколько разделов, содержащих карты OpenStreetMap в формате shape-файлов. Есть уже готовые данные и несколько слоев.

Ссылка: <http://gis-lab.info/qa/data.html>

- GISGeo

Очень полезный ресурс поскольку содержит не только ссылки на шаблоны карт, но и на данные, которые можно к ним привязать.

Ссылка: <http://gisgeo.org/index.php?id=7#GovOpen>

- OpenStreetMap wiki

Собственно, центр знаний по проекту OSM. На странице «Shapefiles» можно найти ряд полезных ссылок.

Ссылка: https://wiki.openstreetmap.org/wiki/Main_Page

⁵ С техническим описанием можно ознакомиться на <https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>.

- Highmaps

Доступно множество карт по разным странам и материкам, в форматах SVG, GeoJSON и Javascript.

Ссылка: <http://code.highcharts.com/mapdata/>

- Global Administrative Areas (GADM)

На данном источнике можно найти shape-файлы по разным странам (для некоммерческого использования, разрешены научные публикации).

Ссылка: <http://gadm.org>

Стоит отметить, что использование готовых shape-файлов имеет свои недостатки, среди которых можно выделить:

1. Зависимость от обновлений.
2. Отсутствие части данных (например, невозможность выделить регион по административному уровню, поскольку его нет в таблице).

3.1.1 Пример получения карты с GISLab

В данной секции я разберу пример получения карты с GISLab. Файлы в формате .shp можно найти по ссылке: <http://beryllium.gis-lab.info/project/osmshp/>. Выбираем необходимый регион и жмем «Скачать». В качестве примера я выбрал Хабаровск (<http://beryllium.gis-lab.info/project/osmshp/region/RU-KHA>).

Для начала установим рабочую директорию с помощью команды `setwd()`, либо по вкладкам меню:

Session -> Set Working Directory -> Choose Directory...

Будет проще, если рабочая директория не будет находиться в папке со слоями shape-файла (в данном случае это папка **data**).

Далее мы прочитаем содержимое shape-файла, параллельно занеся его содержимое в объект (назовем его **khab1**). Для этого используем команду `readOGR(dsn, layer)`, где `dsn` - путь к директории, в которой находится shape-файл, а `layer` - имя слоя, который нас интересует. Нам нужен слой **boundary-polygon**: он содержит информацию об административных границах регионов.

```
khab1 <- readOGR(dsn = "Карты/RU-KHA/data", "boundary-polygon")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/RU-KHA/data", layer: "boundary-polygon"
## with 255 features
## It has 3 fields
```

Полученный объект **khab1** содержит информацию о 24 полигонах, являющимися границами разного административного уровня⁶, и относится к классу

⁶ С полным списком административных уровней можно ознакомиться в Приложении 1

SpatialPolygonsDataFrame что наделяет его несколько иными от **dataframe** свойствами. Команды, работающие с **dataframe** применимы и к нему, однако требуют небольшой коррекции. В связи с тем, что объект Spatial класса состоит из слотов (ячеек), то нужно не только указать его имя, но и имя слота. То есть, чтобы получить доступ к данным атрибутов shape-файла, нужно указать слот `@data`. Чтобы получить первые 20 значений атрибутов, введём следующую команду:

```
head(khab1@data, 20) #По умолчанию head() выводит 6 значений
```

##	OSM_ID	NAME	ADMIN_LVL
## 0	-1651688	Тугуро-Чумиканский район	6
## 1	-1651685	Верхнебуреинский район	6
## 2	-3967161	Чергалинский сельсовет	8
## 3	-1650458	Аяно-Майский район	6
## 4	-2373069	Наслег Анамы	8
## 5	-2372156	Чагдинский наслег	8
## 6	-3971392	городское поселение Февральск	8
## 7	-1744412	Селемджинский район	6
## 8	-3971390	Исинский сельсовет	8
## 9	-4127674	сельское поселение Этыркэн	8
## 10	-4127773	Тырминское сельское поселение	8
## 11	-4127669	сельское поселение Алонка	8
## 12	-4839857	сельское поселение Село Красицкое	8
## 13	-4127772	Аланапское сельское поселение	8
## 14	-4127647	сельское поселение Село Усть-Ургал	8
## 15	-4109857	сельское поселение Село Покровка	8
## 16	-4839759	Глебовское сельское поселение	8
## 17	-4127740	Согдинское сельское поселение	8
## 18	-4127668	Чекундинское сельское поселение	8
## 19	-4127646	Новоургальское городское поселение	8

Также мы можем узнать, границы объектов какого административного уровня хранятся в файле. Для этого воспользуемся функцией `unique()`, которая выводит уникальные значения из вектора значений (в нашем случае, это будет столбец `ADMIN_LVL`).

```
unique(khab1@data$ADMIN_LVL)

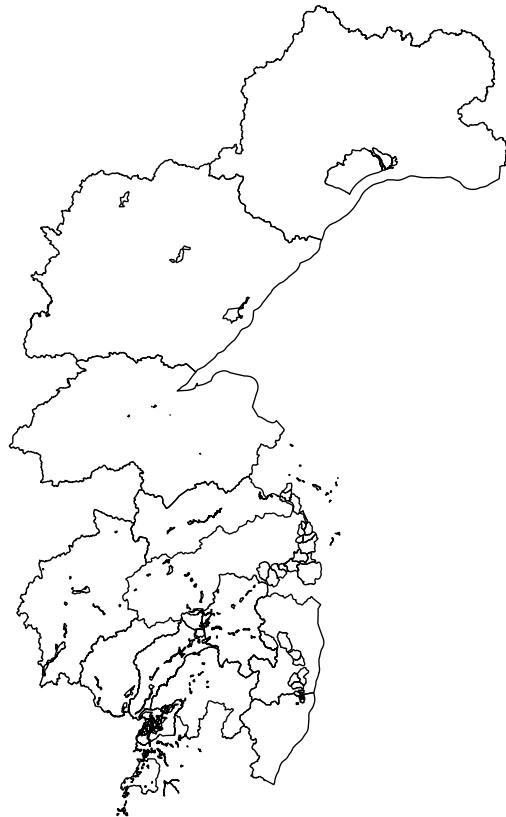
## [1] 6   8   9   4   <NA> 10
## Levels: 10 4 6 8 9
```

Как мы видим, в файле содержатся границы муниципальных районов и административные районы городских округов, которые нам и нужны.

Допустим, у меня есть некоторые данные по административным округам Хабаровска. Отобразим карту командой `plot(khab1)`, добавив `main =` для заголовка:

```
#Зададим размеры полей  
par(mar = c(0, 0, 1, 0))  
plot(khab1, main = "Хабаровский край")
```

Хабаровский край



Очевидно, что данная проекция содержит лишние полигоны, представляя карту в неудобном масштабе, а также увеличивает время прорисовки. Чтобы отобразить только нужные регионы, необходимо выбрать их, указав их административный уровень, предварительно избавившись от значений <NA>, чтобы избежать ошибки.

```
khab1@data <- na.omit(khab1@data)
```

Функция `na.omit` принимает слот `@data` из объекта `khab1`, и удаляет⁷ из него все строки, содержащие отсутствующие данные.

Теперь нужно выбрать строки, содержащие данные по регионам нужного административного уровня. Для административных округов города нам нужен девятый (9) уровень. Создадим новый объект `khab2`, в который попадут только нужные нам полигоны и прорисуем его:

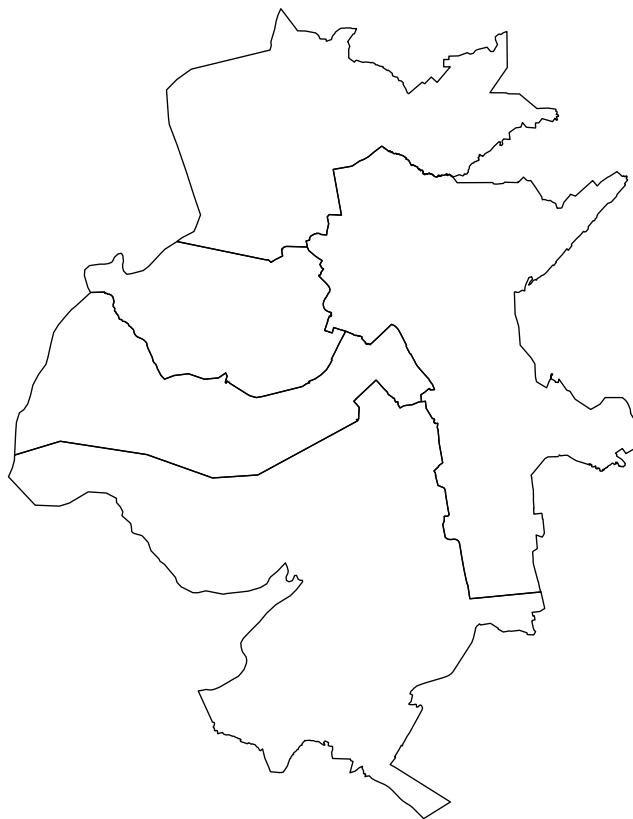
⁷Функция `na.omit()` может удалить строки с необходимыми полигонами, которые по ошибке есть недостающие значения. В таком случае лучше присвоить этим значениям имя или номер с помощью функции `is.na()`.

```

khab2 <- khab1[khab1$ADMIN_LVL == 9, ]
par(mar = c(0, 0, 1, 0))
plot(khab2, main = "Хабаровск")

```

Хабаровск



Таким образом, мы получили карту административных округов Хабаровска.

В полученном объекте содержится всего 5 полигонов. Переименуем ID каждого из них и выведем содержимое слота следующими командами:

```

khab2$OSM_ID <- c(1, 2, 3, 4, 5)
khab2@data

##      OSM_ID           NAME ADMIN_LVL
## 71      1   Центральный район      9
## 72      2 Индустриальный район      9
## 78      3   Кировский район      9
## 80      4 Краснофлотский район      9
## 85      5 Железнодорожный район      9

```

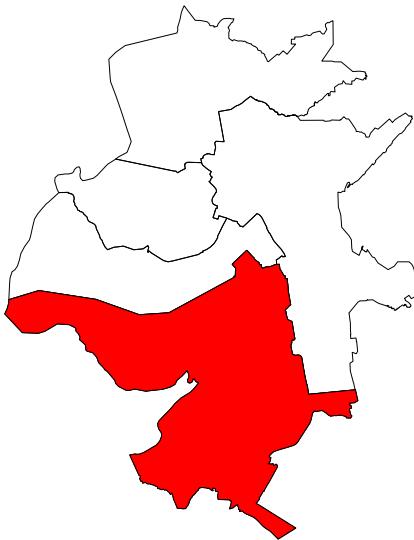
Теперь к каждому региону можно обратиться по его номеру, но карта все ещё не информативна: невозможно определить, где какой район. Закрасим Индустриальный район (под номером 2) красным цветом:

```

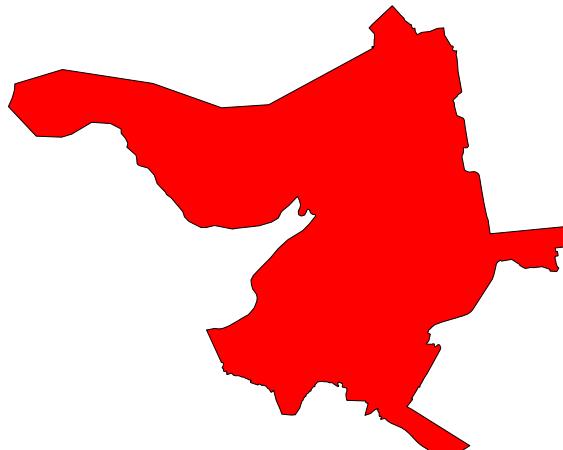
par(mar = c(0, 0, 1, 0))
#Получим карту слева
plot(khab2, main = "Правильная карта")
plot(khab2[khab2$OSM_ID == 2, ], col = "red", add = TRUE)
par(mar = c(0, 0, 1, 0))
#Получим карту справа
plot(khab2, main = "Правильная карта")
plot(khab2[khab2$OSM_ID == 2, ], col = "red",
      add = FALSE, main = "Неправильная карта")

```

Правильная карта



Неправильная карта



Заметим, что параметр `add = TRUE` нужен для того, чтобы последующее изображение наслойлось на предыдущее (также нам потребуется 2 команды `plot()`). В противном случае мы получим изображение отдельного района, как на рисунке справа.

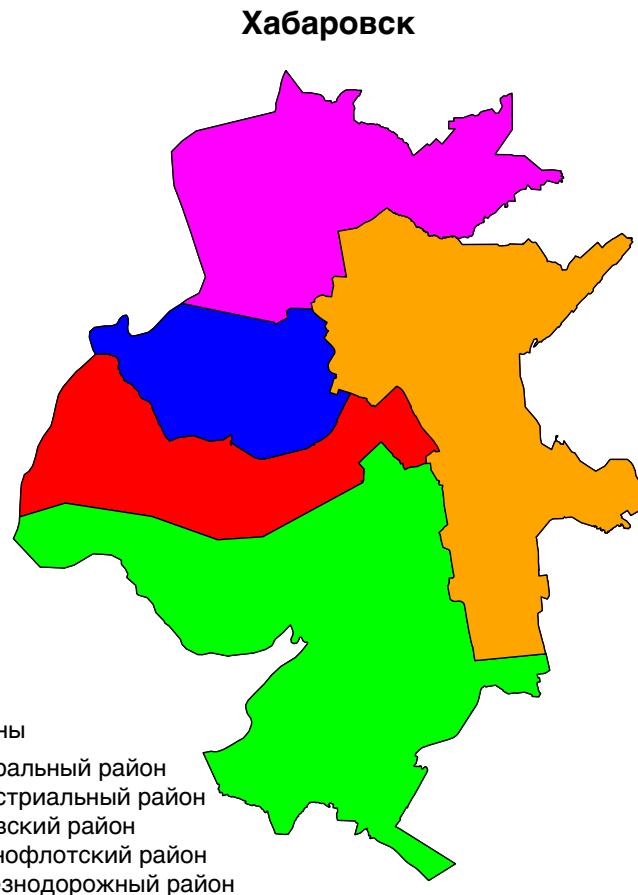
Наконец, сделаем легенду для нашей карты. Функция `plot()` в данном случае потребует громоздкой конструкции (для компактности используем цикл `for` и создадим вектора цветов и имен районов):

```

leg <- as.vector(khab2@data$NAME)
col.vec <- c("red", "green", "blue", "magenta", "orange")
par(mar = c(0, 0, 1, 0))
plot(khab2, main = "Хабаровск")
#С помощью цикла мы закрасим 5 областей
for (i in 1:length(khab1@data$OSM_ID)) {
  plot(khab2[khab2$OSM_ID == i, ], col = col.vec[i], add = TRUE)
}
legend("bottomleft", leg, bg = "transparent",
       box.lty = 0, fill = col.vec,

```

```
title = "Районы", cex = 0.9,
title.adj = 0.22)
```



поэтому разумнее воспользоваться пакетом `ggplot2`. Карта, а также все визуальные усовершенствования (к примеру, легенда и подписи осей координат) будут храниться в отдельном объекте, который мы назовём `khab.map`. Команды выглядят следующим образом (замечу, что система координат⁸ для всех прилагающихся в `rusmaps` карт — WGS 84 (код: EPSG 4326) и не нуждается в коррекции):

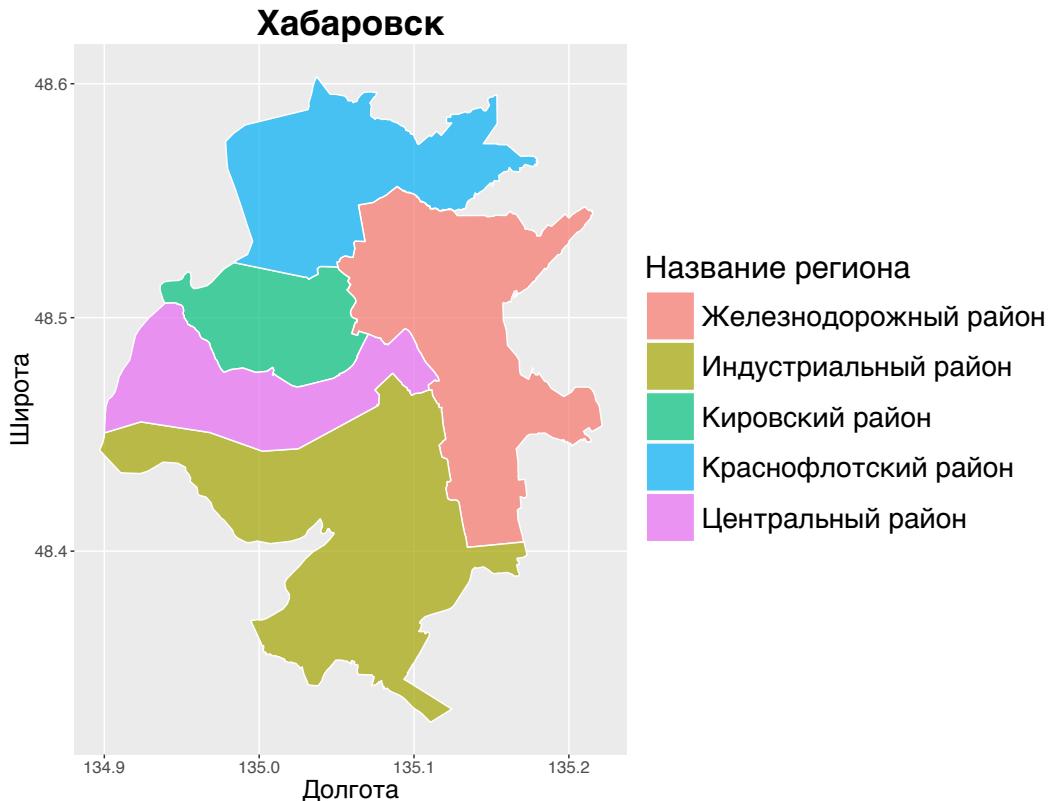
```
#Перевод проекции карты в dataframe:
map.df <- fortify(khab2, region = "NAME")
#Создание окружения для карты:
plot5theme <- theme(axis.text = element_text(size = 12),
                      axis.title = element_text(size = 18),
                      legend.text = element_text(size = 20),
                      legend.title = element_text(size = 22),
                      plot.title = element_text(size = 25, face = "bold"))
#Создание объекта карты:
khab.map <- ggplot(map.df, aes(long, lat, group = group, fill = id)) +
```

⁸Система координат задается кодом EPSG, подробнее на <https://goo.gl/DcEVwT>

```

#Добавим alpha = 0.7, чтобы полигоны слегка просвечивали
geom_polygon(color = "white", alpha = 0.7) +
coord_map() +
labs(x = "Долгота", y = "Широта", fill = "Название региона") +
ggtitle("Хабаровск")
#Выведем полученный объект:
khab.map + plot5theme

```



Теперь нам вполне понятно расположение регионов на карте⁹. С помощью `ggplot2` стало возможным наслаждение карты на систему координат в градусах, а также полу-прозрачности и других эффектов. Если же значения широты и долготы не нужны, то их можно удалить добавив к объекту `khab.map` команды `scale_x_discrete` и `scale_y_discrete` для осей x и y соответственно. Касательно окружения (оформления) карты, как и любого другого графика, я рекомендую самостоятельно поэкспериментировать с элементами функции `theme()`. К примеру, если я хочу получить карту без легенды и значений широты и долготы, то моя команда будет выглядеть следующим образом:

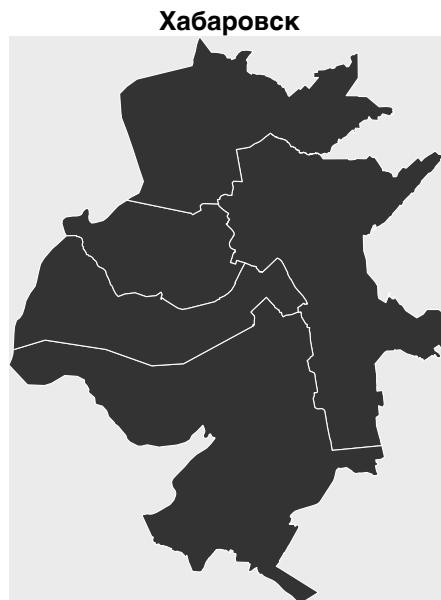
⁹Несмотря на наглядность, эту карту разумнее назвать схемой (картограммой), так как ориентироваться по ней почти невозможно. Однако, этого может быть вполне достаточно для решения многих задач.

```

#Удалим fill = id, чтобы убрать легенду
#Также удалим labs, поскольку подписи осей теперь тоже не нужны
khab.map.noax <- ggplot(map.df, aes(long, lat, group = group)) +
  geom_polygon(color = "white") +
  coord_map() +
  ggtile("Хабаровск") +
  scale_x_discrete(breaks = NULL) +
  scale_y_discrete(breaks = NULL)

#Отобразим карту
khab.map.noax + plot5theme + theme(axis.title = element_blank())

```



Таким образом, мы рассмотрели пример получения карты Хабаровска, используя готовый шаблон с GISLAB.

3.1.2 Построение карты в формате geoJSON

GeoJSON¹⁰, как и shapefile, является популярным форматом для хранения геоданных, обладающим своими преимуществами и недостатками. К его преимуществам можно отнести то, что, во-первых, geoJSON существует в виде одного файла, в то время как shapefile для работы требует три. Во-вторых, данные в geoJSON хранятся в виде программного кода (javascript), который понятен человеку и который можно редактировать средствами, отличными от GIS. Это делает его даже более популярным среди программистов, особенно среди тех, кто работает с интерактивными картами и пространственными данными в целом. К недостаткам же можно отнести меньшую, в отличие от shape-файлов компактность: размер файлов geoJSON, как правило, в два раза больше shapefile (и, возможно, требует упрощения геометрии границ¹¹).

¹⁰С более подробной спецификацией можно ознакомиться на http://gis-lab.info/docs/geojson_ru.html

¹¹Подробнее в секции 3.2.4

В Интернете распространены ресурсы, на которых можно найти данные в формате geoJSON. В качестве примера я могу привести **Портал открытых данных Правительства Москвы** (<http://data.mos.ru>). На нём можно найти различные данные об инфраструктуре Москвы в виде точек, линий и полигонов. GeoJSON можно получить с помощью API, а файлы в формате .json и .docx доступны без регистрации. Как вариант, мы можем взять файл .json и переделать его в .geojson, так как это практически один и тот же формат, а их различие лишь в синтаксисе и в типе содержащихся данных (то есть, в .json хранятся просто данные (как таблица) о, допустим, названии, посещаемости и координатах, в то время как .geojson содержит в себе информацию о точках (как объектах), с теми же параметрами, но код файла делает его пригодным для обработки в GIS). Таким образом, всё что нам нужно сделать, это создать файл, в котором мы опишем объекты, как это делается для geoJSON. Для этого нам понадобится любой редактор программного кода (например, notepad++ и его аналоги. Можно использовать и Rstudio, но подсветки синтаксиса не будет). Также будет полезно сделать шаблон файла geoJSON для каждого типа данных (точек, линий и полигонов).

Приведу пример создания точечного geojson из данных о московских кофейнях (данные можно найти здесь: <http://data.mos.ru/opendata/7710881420-kofeyni>). Для начала нам нужно получить шаблон geojson, содержащего точки. Для этого можно сохранить любой набор точек, созданный в QGIS (для примера возьму города-миллионеры). Откроем полученный файл редактором кода, и обнаружим, что geojson выглядит следующим образом (приведена часть файла):

```
"type": "FeatureCollection",
"crs": {
    "type": "name",
    "properties": {
        "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
    }
},
"features": [
{
    "type": "Feature",
    "properties": {
        "id": "21053210",
        "name": "Воронеж",
        "pop_2015": 1023570
    },
    "geometry": {
        "type": "Point",
        "coordinates": [
            39.1607149,
            51.6587727
        ]
    }
}
```

```
}
```

Данный файл можно поделить на две составные части: «шапку», где указан тип содержимого файла и система координат (1–8 строки) и перечисления объектов и их свойств (properties и geometry).

Что касается файла .json, то он состоит из объектов, описанных следующим образом:

```
[  
  {  
    "Lat": 55.78994653781263,  
    "Lon": 37.63429892286436,  
    "Id": "d9629d7f-9a30-43a7-835b-dad95ea3c7ad",  
    "Number": 1,  
    "Cells": {  
      "global_id": 20660592,  
      "Name": "Кофеня «Кофе Хаус»",  
      "IsNetObject": "да",  
      "OperatingCompany": "Кофе Хаус",  
      "AdmArea": "Центральный административный округ",  
      "District": "Мещанский район",  
      "Address": "проспект Мира, дом 77, корпус 1",  
      "PublicPhone": "(495) 221-83-81"  
      "SeatsCount": 15,  
      "SocialPrivileges": "нет",  
      "Longitude_WGS84": "37.6342989228643600",  
      "Latitude_WGS84": "55.7899465378126320"  
    }  
  }  
]
```

Очевидно, что для получения geojson с данными о кофейнях нужно просто верно написать «шапку» с системой координат и типом содержимого, и ввести соответствующие properties (в json они находятся в cells) и координаты. Аналогичным путём можно создать и свой собственный .geojson. Для большого количества точек процесс достаточно однообразный, поэтому я вписал первые 10 объектов. Чтобы быстро преобразовывать большое количество данных, можно воспользоваться скриптом (например на Python), который выполнит замену автоматически.

```
{  
  "type": "FeatureCollection",  
  "crs": {  
    "type": "name",  
    "properties": {  
      "name": "urn:ogc:def:crs:OGC:1.3:CRS84"  
    }  
  }
```

```

},
"features": [
{
  "type": "Feature",
  "properties": {
    "global_id": 20660592,
    "Name": "Кофея «Кофе Хаус»",
    "IsNetObject": "да",
    "OperatingCompany": "Кофе Хаус",
    "AdmArea": "Центральный административный округ",
    "District": "Мещанский район",
    "Address": "проспект Мира, дом 77, корпус 1",
    "PublicPhone": "(495) 221-83-81",
    "SeatsCount": 15,
    "SocialPrivileges": "нет"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [
      37.63429892286436,
      55.78994653781263
    ]
  }
}
]
}

```

Отобразим местоположения кофеен. В качестве фона можно воспользоваться картой административных округов Москвы (в формате geojson), которую можно получить по ссылке: <http://gis-lab.info/qa/moscow-atd.html>.

```

mosmap <- readOGR("Карты/geojson/Москва.geojson", "OGRGeoJSON")

## OGR data source with driver: GeoJSON
## Source: "Карты/geojson/Москва.geojson", layer: "OGRGeoJSON"
## with 12 features
## It has 3 fields

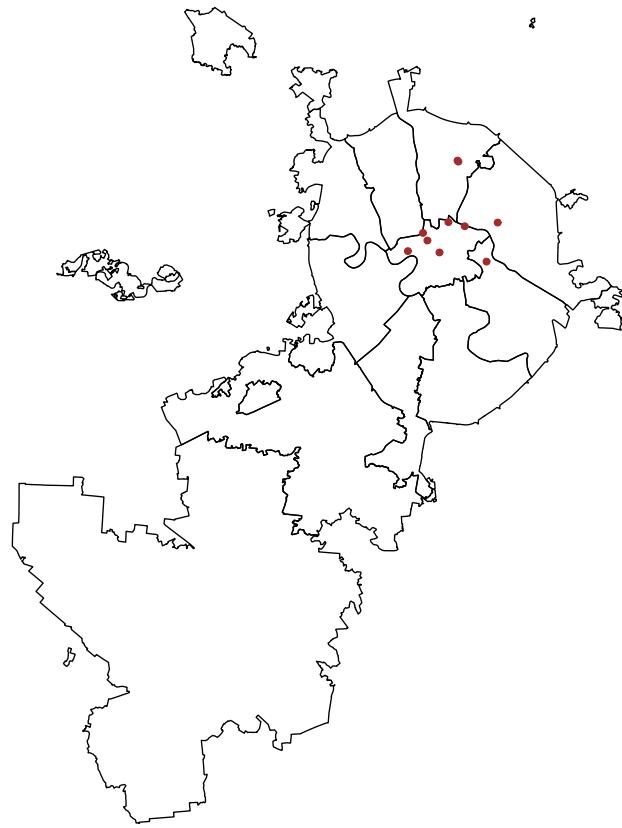
coffee <- readOGR("Карты/geojson/Кофейни.geojson", "OGRGeoJSON")

## OGR data source with driver: GeoJSON
## Source: "Карты/geojson/Кофейни.geojson", layer: "OGRGeoJSON"
## with 10 features
## It has 10 fields

par(mar = c(0, 0, 1, 0))
plot(mosmap, main = "Кофейни в Москве")
plot(coffee, add = TRUE, col = "brown", pch = 20)

```

Кофейни в Москве



В качестве альтернативы можно воспользоваться API портала. Для этого мы можем воспользоваться пакетом **datamos** (репозиторий: <https://github.com/bdemeshhev/datamos>), который можно установить с помощью команды:

```
devtools::install_github("bdemeshev/datamos")
library(datamos)
```

Теперь с помощью функции `datamos()`, введя ID данных в качестве аргумента. ID можно узнать, кликнув по вкладке «Паспорт» как на скриншоте:

A screenshot of a web application interface titled 'Кофейни'. At the top, there are tabs: 'Таблица', 'Карта', 'Паспорт' (which is highlighted in blue), 'Описание', 'Справочники', 'Скачать', and a help icon. Below the tabs, there are two main sections. The left section contains a table with one row: 'Название поля паспорта' (Name of the passport field) and 'Идентификатор набора данных:' (Data set identifier). The right section also has a table with one row: 'Значение поля паспорта' (Value of the passport field) and '1786'. A red question mark icon with the text 'Не нашли объект?' (Object not found?) is located between the two sections.

Теперь с помощью команды `names()` мы можем узнать названия столбцов и выбрать те, которые нам нужны. Функция `datamos()` возвращает набор данных, который нужно конвертировать в `SpatialPointsDataFrame`. Это можно сделать с помощью функции `coordinates()`.

```

coffee1 <- datamos(1786)
names(coffee1)

## [1] "Lat"           "Lon"           "Id"
## [4] "Number"        "global_id"       "Name"
## [7] "IsNetObject"   "OperatingCompany" "AdmArea"
## [10] "District"      "Address"        "PublicPhone"
## [13] "SeatsCount"    "SocialPrivileges" "Longitude_WGS84"
## [16] "Latitude_WGS84"

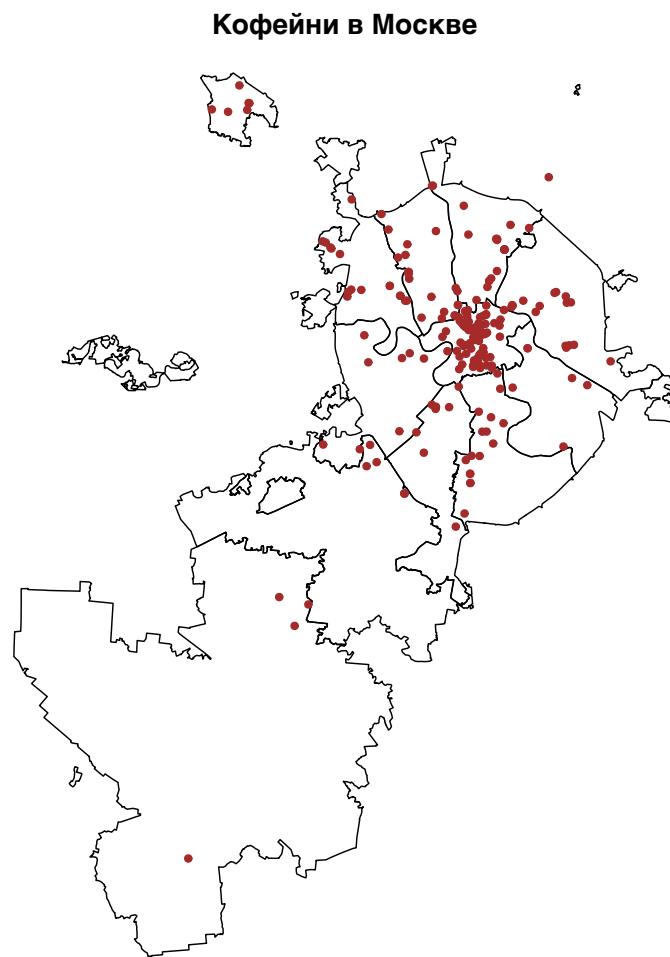
coffee1 <- coffee1[, c(1, 2, 6, 7, 8, 10, 13)]
coordinates(coffee1) <- ~Lon+Lat

```

```

par(mar = c(0, 0, 1, 0))
plot(mosmap, main = "Кофейни в Москве")
plot(coffee1, add = TRUE, col = "brown", pch = 20)

```



3.1.3 Конвертация файлов геоданных

В случаях, если Вам по какой-либо причине нужно изменить формат файла, содержащего необходимые Вам географические данные, то его необходимо конвертировать. Я приведу несколько способов как это сделать:

- Mapshaper

Удобный и практичный вариант, если Вы планируете пользоваться исключительно shape-файлами и geoJSON и не хотите пользоваться QGIS. Позволяет немного подредактировать карту (с помощью консоли) прямо в браузере и экспортить её в форматах shapefile, geoJSON, topoJSON а также CSV и SVG. Работает в Chrome, Firefox и Safari, но в последнем не поддерживает экспорт. Opera и IE не поддерживаются.

Ссылки: <http://mapshaper.org/>, <https://goo.gl/BFkjV7>

- gisconvert.com

Тут можно конвертировать «экзотику», вроде файлов AutoCAD (DXF), Google KML и другие. Также есть конвертор GPX (используется на устройствах GPS, в том числе на мобильных телефонах).

Ссылка: <https://www.gisconvert.com>

- R

Для конвертации можно пользоваться и средствами R. Для чтения файлов мы пользуемся `readOGR()` (также можно использовать `readShapePoly()` и другими функциями из пакета `maptools`). При этом мы получаем объект `SpatialPolygonsDataFrame`. Для записи файлов в формате shapefile можно воспользоваться функцией `writePolyShape()` (для полигонов), а для geojson — функциями `geojson_json()` из пакета `geojsonio` и `geojson_write()`. Можно воспользоваться и `writeOGR()`, но вышеприведённые способы проще и менее конфликтны в настройках.

```
x <- readOGR("Карты/Города_миллионники", "Воронеж")
#Конвертация в geojson
library("V8")
library("geojsonio")
x1 <- geojson_json(x)
#Сохранение как geojson
geojson_write(x1, file = "Карты/geojson/FILE.geojson")
#Сохранение как shapefile
writePolyShape(x, "Карты/Слой")
```

- QGIS

При сохранении файла в QGIS доступен выбор из 23 форматов, включая все вышеперечисленные, за исключением SVG. Учитывая возможности этой программы, я рекомендую именно этот вариант.

3.2 Работа в QGIS

В качестве альтернативы использованию готовых shape-файлов можно создать свои собственные. Такой подход позволяет создать файлы с необходимыми для работы полигонами, без необходимости что-то из них убирать. Также можно использовать QGIS для получения данных, имеющихся в базе, представленных в виде точек и линий (к примеру, различные учреждения, дороги), поскольку использование полигонов для их визуализации нецелесообразно.

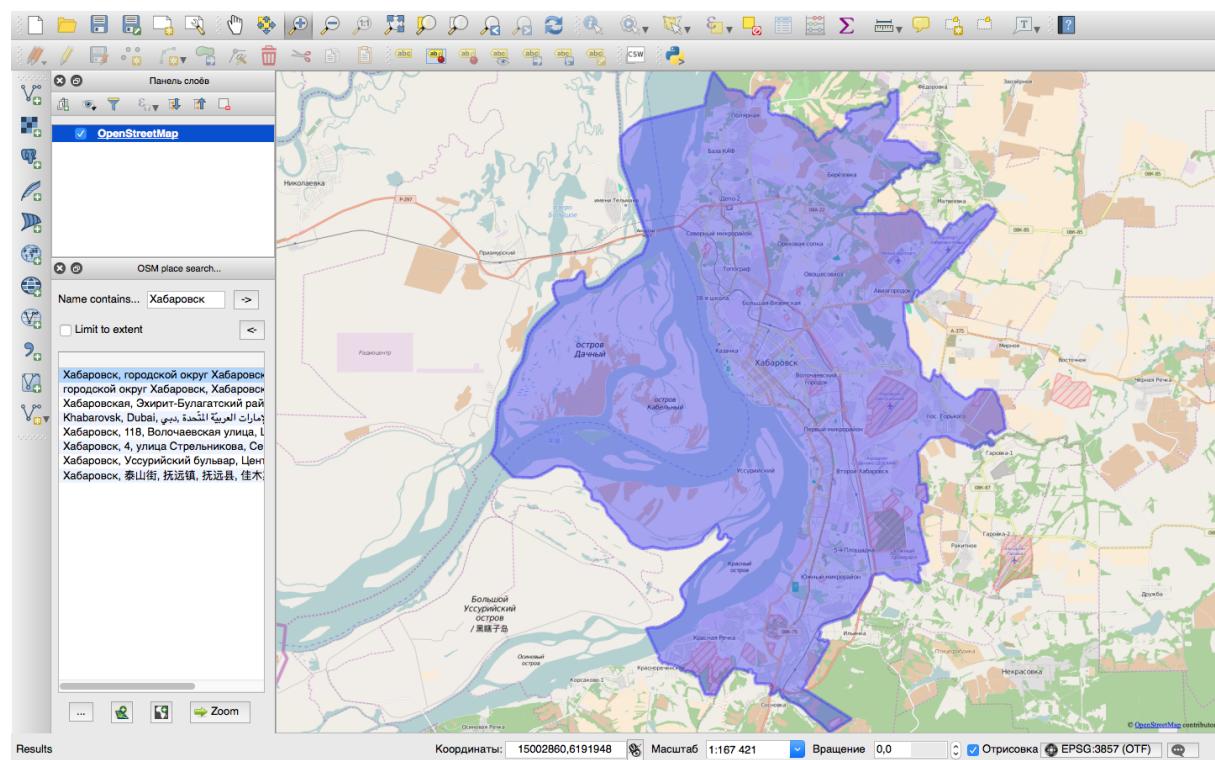
3.2.1 Получение данных OSM

В данной секции я разберу пример получения геоданных с помощью QGIS (версия 2.14.3-Essen на момент написания работы), используя данные OpenStreetMap. В качестве примера я снова возьму Хабаровск.

Для начала, загрузим карту OSM (необходимо подключение к Интернету). Для этого, перейдем по вкладкам меню:

Интернет -> OpenLayers plugin -> OpenStreetMap -> OpenStreetMap

В рабочей области появится карта мира. Введем в поле поиска на панели **OSM place search...** имя искомого города и выберем наиболее подходящий вариант. Он будет подсвечен на карте, а также зайдёт всю рабочую область:

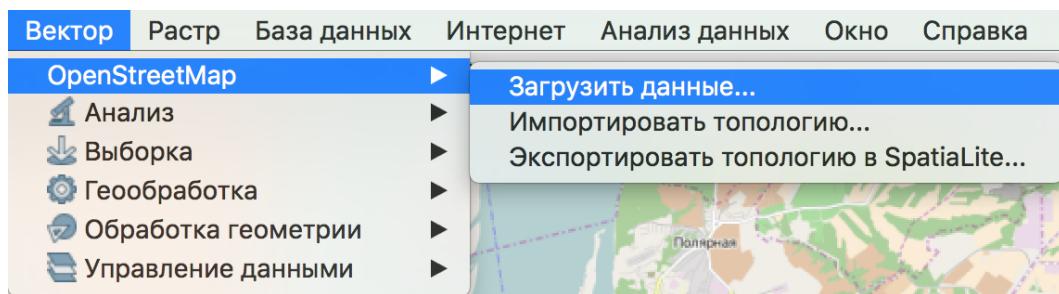


Загрузим данные¹², имеющиеся на карте. Стоит добавить, что данные загружаются с области, которая находится в рабочей области. То есть если приблизить карту с

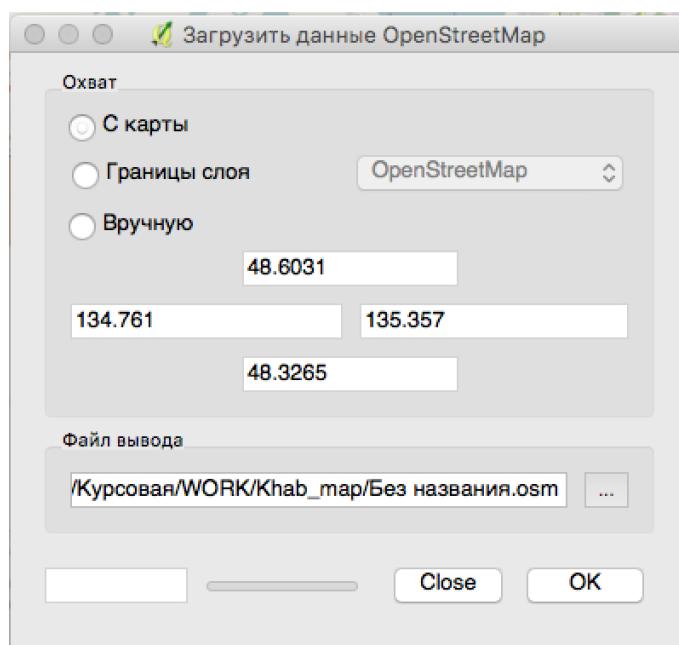
¹²В зависимости от города, размер загруженных данных варьируется и может быть весьма большим (100Мб и более для крупных городов).

помощью инструмента *Луна* то область загрузки данных уменьшится. Итак, пройдём по вкладкам меню (туда мы ещё не раз вернёмся):

Вектор -> OpenStreetMap -> Загрузить данные...



В открывшемся диалоговом окне нажав на кнопку с многоточием, выберем директорию, куда сохраним загруженные данные, а также название выходного файла. Получим файл в формате .osm (OSM XML¹³).

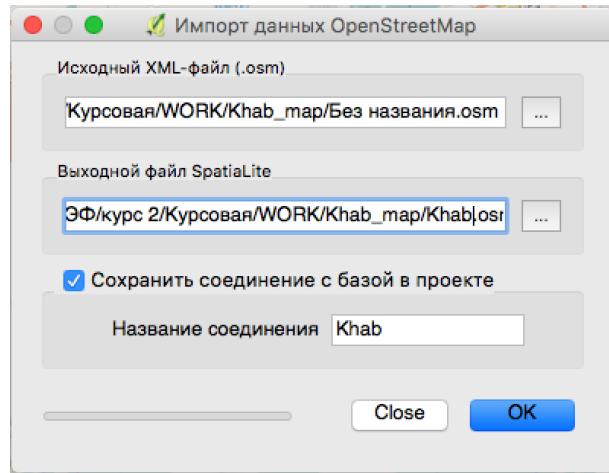


Для дальнейшей работы нам необходимо конвертировать этот файл в базу данных. Для этого, пройдем по вкладкам меню:

Вектор -> OpenStreetMap -> Импортировать топологию...

В открывшемся диалоговом окне выберем полученный ранее .osm файл, установим флажок (он здесь один) и нажмём OK.

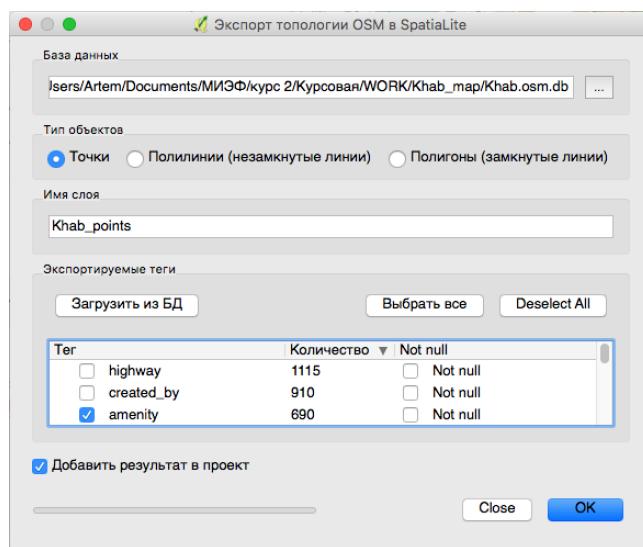
¹³Подробнее по ссылке: http://wiki.openstreetmap.org/wiki/OSM_XML



Наконец, выведем слои геометрии SpatiaLite¹⁴ пройдя по вкладкам меню:

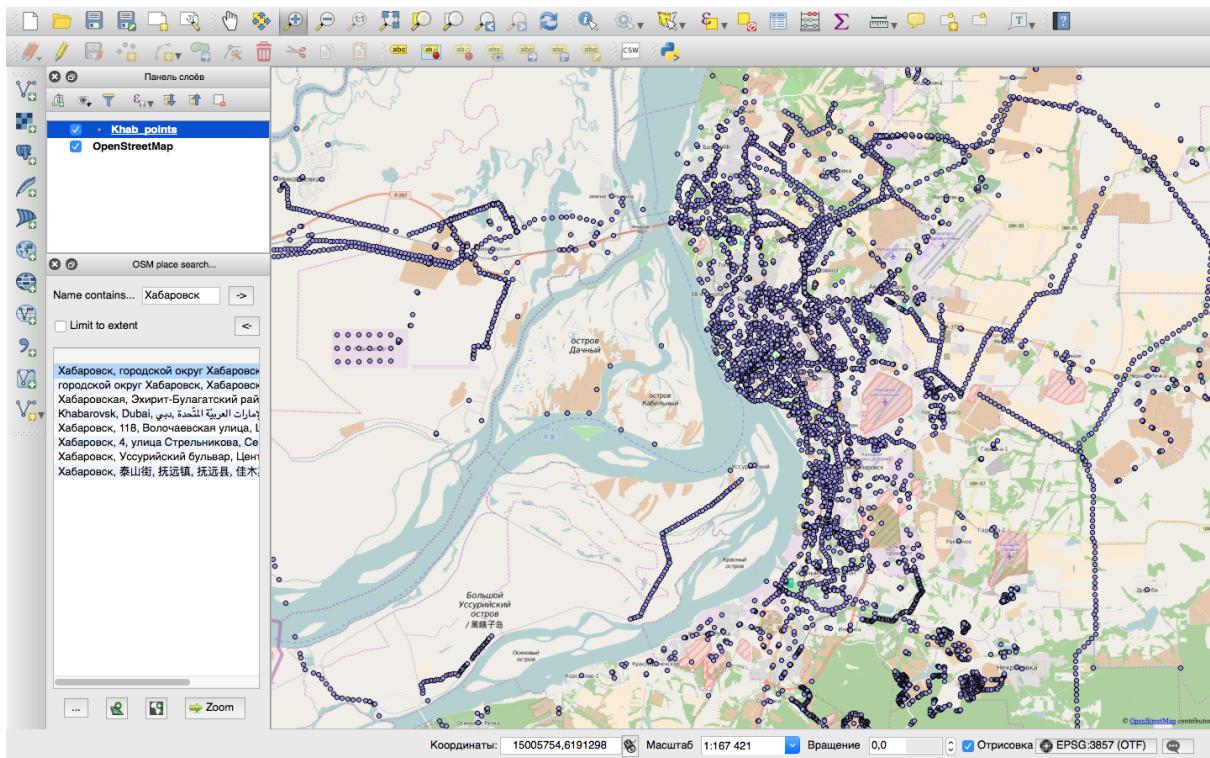
Вектор -> OpenStreetMap -> Экспортировать топологию в SpatiaLite...

В открывшемся диалоговом окне выберем созданный ранее файл базы данных (.db), кликнув по кнопке с многоточием. Здесь нам предстоит выбор, какие пространственные объекты нам нужны: точки, линии или полигоны. От себя скажу, что каких-то элементов в базе может недоставать (или могут отсутствовать значения в одной из ячеек строки). К примеру, если Вам нужно провести исследование по микрорайонам города, то выбрав «полYGONы» Вы можете разочароваться в результате. С другой стороны, вариант с полигонами неплох для исследований землепользования. В остальных случаях имеет смысл взять точечные данные (местоположения заправочных станций, школ, больниц), хотя и они могут быть неполными. В любом случае, отмечаем выбор слева от необходимых объектов (в моём случае это будут точки). Щелкнув по кнопке «Загрузить из БД» получим список имеющихся тегов. Нас интересуют теги **name** (название) и **amenity** (удобства).

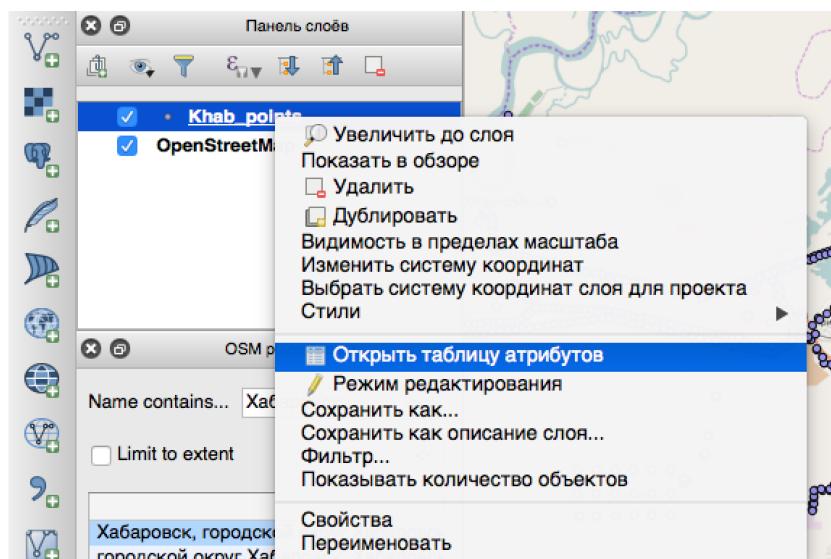


¹⁴SpatiaLite - расширение ядра SQLite (Система управления базами данных), позволяющее работать с геоданными.

Результат выглядит следующим образом:



Обратим внимание на панель слоёв. В ней появился объект **Khab_points**, который содержит все точки, находящиеся в рабочей области. Разумеется, все они нам не нужны, и чтобы выбрать только нужные, необходимо создать запрос. Но сперва я рекомендую ознакомиться с содержащимися в слое элементами. Для этого щёлкнем правой клавишей мыши (ПКМ) по слою **Khab_points** и выберем опцию *Открыть таблицу атрибутов*.

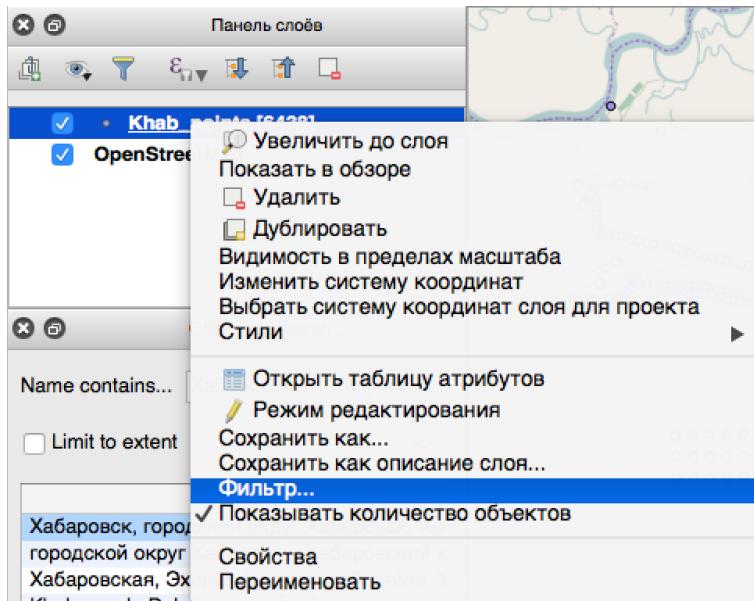


На экране появится таблица, в нашем случае состоящая из трёх столбцов. Два из них

- **name** и **amenity** - выбраны по тэгам, которые мы выбрали при экспорте топологии. То есть в зависимости от количества выбранных тэгов будет различаться число столбцов, и, как следствие, возможности в построении запроса. Щёлкнув по названию колонки, список в ней будет отсортирован по алфавиту, причём связь между ячейками в строках не нарушится. То есть щёлкнув по «amenity» мы получим список, отсортированный по параметру «удобства».

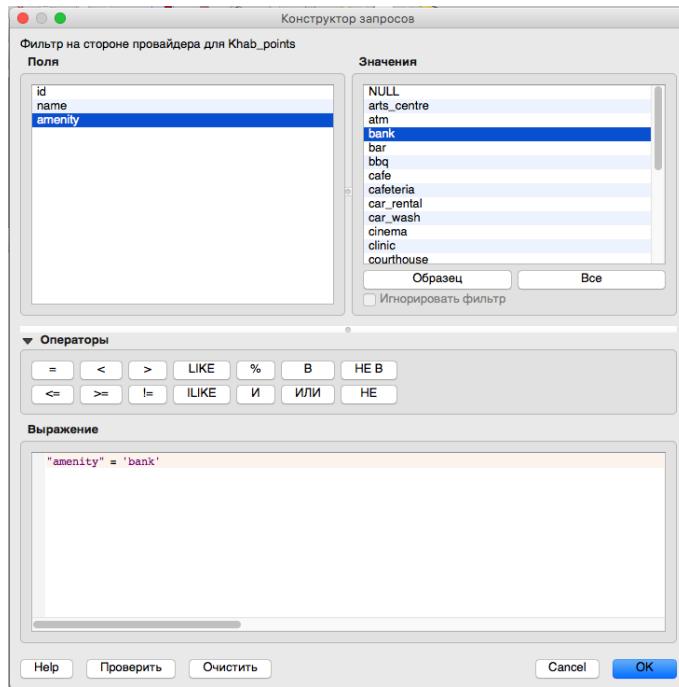
																id	name	amenity
3687																1812082908	Росбанк	bank
4164																1969646880	Росбанк	bank
4201																1973354092	Росбанк	bank
4202																1973354093	Росбанк	bank
4203																1973354094	Росбанк	bank
4204																1973354095	Росбанк	bank
4205																1973354096	Росбанк	bank
4206																1973354097	Росбанк	bank
4207																1973354098	Росбанк	bank
4208																1973354099	Росбанк	bank
1820																1259505625	Промсвязьб...	bank
1822																1259505630	Промсвязьб...	bank
1463																1022733419	Номос-Реги...	bank
3685																1805582590	Номос-Реги...	bank

Теперь перейдём непосредственно к созданию запроса. Для этого закроем таблицу атрибутов, щёлкнем правой клавишей мыши по слову **Khab_points** и выберем опцию *Фильтр...*

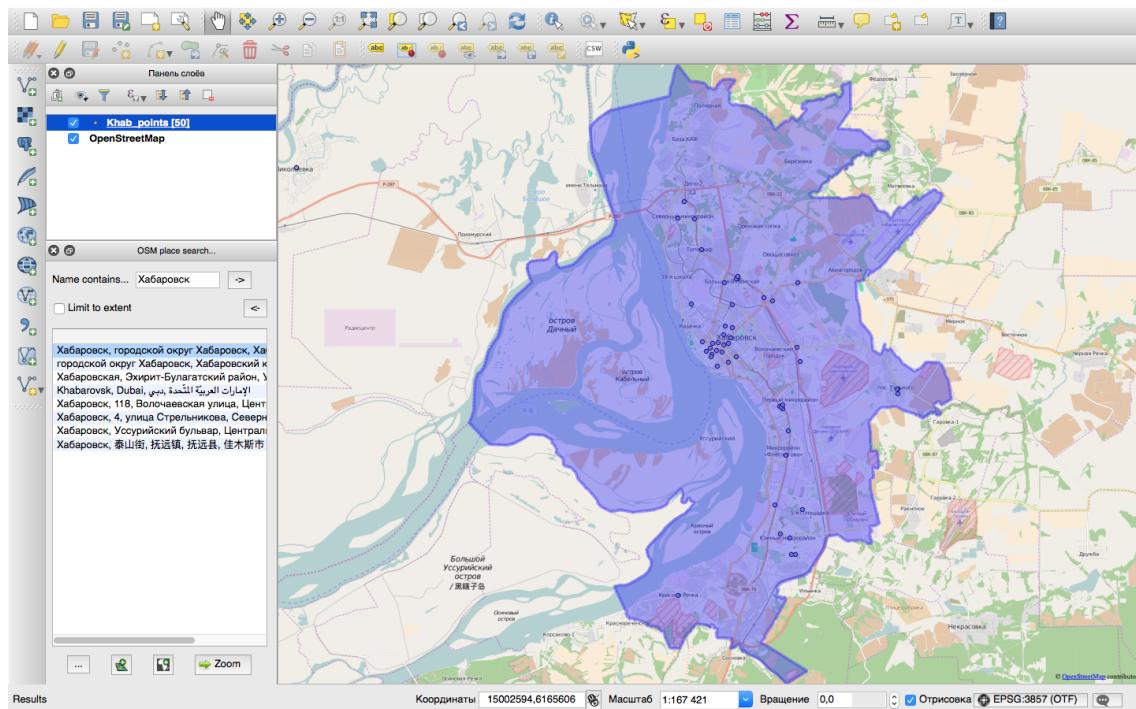


В появившемся окне в поле «Выражение» запишем команду для запроса: `"amenity" = 'bank'`. Очень важно, чтобы название параметра (**amenity**) было заключено в двойные кавычки, а значение параметра (**bank**) - в апострофы. В качестве альтернативного способа ввода выражения запроса можно выбрать имя параметра из группы «Поля», логические операции вводить имеющимися кнопками, а значения параметра

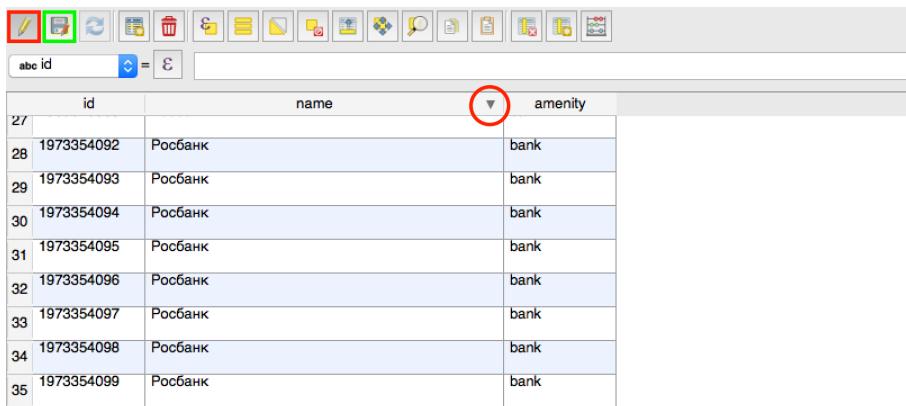
выбрать из группы «Значения» (предварительно отобразив их, щёлкнув по кнопке «Образец»). Для данного примера я выбрал банки, поскольку их достаточно много, а также почти у всех объектов есть имя, что увеличивает ценность данных.



В результате количество точек на карте существенно уменьшится. Теперь на ней отображены только точки по запросу, то есть те, что обозначают местоположение банков в Хабаровске.



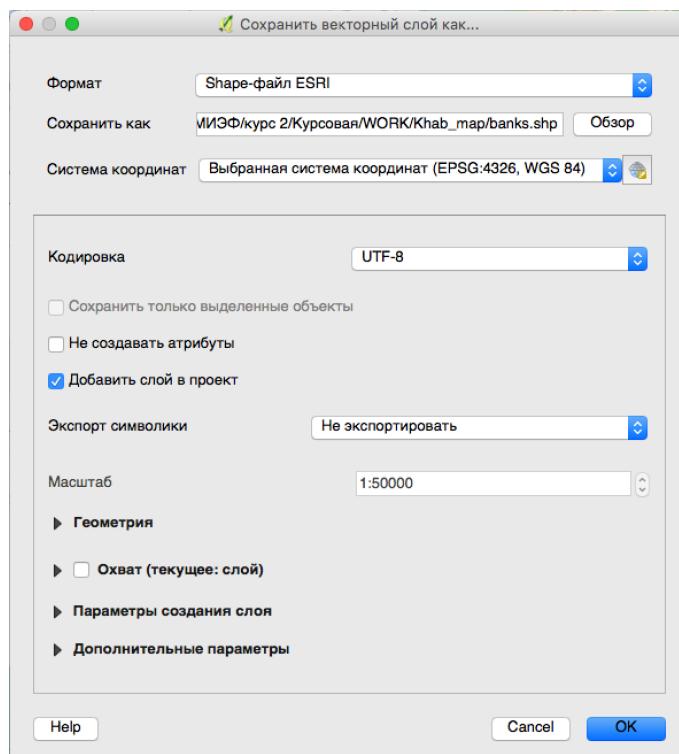
Теперь нам нужно убедиться, что наши данные «не содержат брака». Под словом «брак» я буду понимать опечатки. Для этого откроем таблицу атрибутов для точек, выбранных по запросу (по тому же **Khab_points**).



	id	name	amenity
27			
28	1973354092	Росбанк	bank
29	1973354093	Росбанк	bank
30	1973354094	Росбанк	bank
31	1973354095	Росбанк	bank
32	1973354096	Росбанк	bank
33	1973354097	Росбанк	bank
34	1973354098	Росбанк	bank
35	1973354099	Росбанк	bank

Теперь, как мы это делали с **amenity**, отсортируем список по названиям банков (красный кружок), и проверим, нет ли опечаток. Названия одинаковых банков должны быть одинаковыми и располагаться в списке вместе. Щёлкнув по иконке карандаша на панели инструментов, мы перейдём в режим редактирования. Исправим все опечатки. После этого, щёлкнем по иконке с дискетой, чтобы сохранить изменения.

Нам остаётся лишь сохранить выбранные точки в shapefile (при желании, есть и другие форматы, в том числе и geoJSON). Для этого щёлкнем ПКМ по слою **Khab_points** и выберем опцию *Сохранить как...*



Теперь испытаем наш собственноручно созданный shape-файл. В Rstudio мы построим карту Хабаровска и выполним наслойение координат банков:

```
khab.banks <- readOGR("Карты/Хабаровский_край", "banks")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Хабаровский_край", layer: "banks"
## with 48 features
## It has 3 fields

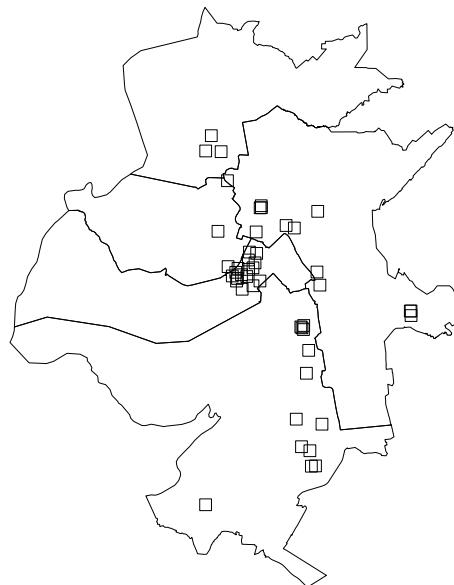
# С помощью функции order() выполним сортировку по названиям
khab.banks@data <- khab.banks@data[order(khab.banks@data$name), ]
head(khab.banks@data, 30)

##           id                  name amenity
## 42 3285118467 Азиатско-Тихоокеанский банк   bank
## 1  332389820    Восточный экспресс банк   bank
## 26 1969457318    Восточный экспресс банк   bank
## 45 3285254639    Восточный экспресс банк   bank
## 2   364067570          ВТБ 24   bank
## 4   473470364          ВТБ 24   bank
## 24 1931563391          ВТБ 24   bank
## 18 1455866884      Далькомбанк   bank
## 15 1358573694      Москва Банк   bank
## 6   822498846       МТС Банк   bank
## 19 1750920606       МТС Банк   bank
## 22 1907865423       МТС Банк   bank
## 8   1022733419     Номос-Региобанк   bank
## 20 1805582590     Номос-Региобанк   bank
## 25 1949019669     Номос-Региобанк   bank
## 10 1259505625     Промсвязьбанк   bank
## 11 1259505630     Промсвязьбанк   bank
## 21 1812082908     Росбанк   bank
## 28 1969646880     Росбанк   bank
## 29 1973354092     Росбанк   bank
## 30 1973354093     Росбанк   bank
## 31 1973354094     Росбанк   bank
## 32 1973354095     Росбанк   bank
## 33 1973354096     Росбанк   bank
## 34 1973354097     Росбанк   bank
## 35 1973354098     Росбанк   bank
## 36 1973354099     Росбанк   bank
## 3   473469687     Сбербанк   bank
## 5   821714023     Сбербанк   bank
## 7   992099163     Сбербанк   bank
```

Как видим, мы получили полноценный shape-файл. Теперь добавим точки, содержащиеся в файле на карту Хабаровска.

```
par(mar = c(0, 0, 1, 0))
plot(khab2, main = "Банки в г.Хабаровск")
plot(khab.banks, add = TRUE, pch = 0)
```

Банки в г.Хабаровск



Таким образом, мы улучшили нашу карту Хабаровска, добавив на неё дополнительные геоданные. Но всё же полученный результат даёт представление лишь о том, в каких районах столько банков а также плотность их расположения. Поэтому, мы сделаем карту с помощью пакета **ggmap**, который является расширением пакета **ggplot2**.

Для начала нам необходимо получить фон для карты. Для этого воспользуемся функцией `get_map()`. Более того, функция `ggmap()` принимает только результат `get_map()`. По умолчанию загружается GoogleMaps, но при желании, указав `source = "osm"` можно вывести карту OSM. Для разнообразия, сделаем на базе карты Google:

```
#В get_map работает поиск по городу.
mapsource <- get_map(location = "Khabarovsk", zoom = 10)
```

Теперь нам нужно сделать небольшие приготовления для построения объекта (**ggmap**). А именно перевести данные о координатах точек (банков) в **dataframe**, поскольку **ggplot2** не работает с объектами класса **SpatialPointsDataFrame**. Функция `fortify()` здесь не подойдёт, но можно сохранить слот `@coords` как **dataframe**. Для этого введём команду:

```
banks.coords <- as.data.frame(khab.banks@coords)
```

Теперь перейдём непосредственно к построению:

```

khab.bank.map <- ggmap(mapsource) +
  #alpha = 0.7 НЕ должно быть в aes(),
  #иначе получим лишние элементы легенды
  geom_polygon(data = map.df, alpha = 0.7, color = "white",
                aes(long, lat, group = group, fill = id)) +
  geom_point(data = banks.coords, size = 2, pch = 1,
             aes(coords.x1, coords.x2)) +
  labs(x = "Долгота", y = "Широта", fill = "Название района") +
  ggtitle("Банки в г.Хабаровск")
khab.bank.map + plot5theme

```



В результате получаем весьма красивую (и информативную) карту. Теперь это действительно карта, а не схема. Но расположение точек на ней выглядит некрасиво. Чтобы это исправить, нужно создать объект **dataframe**, который будет содержать в себе названия банков и их координаты. Для этого выполним следующие команды:

```

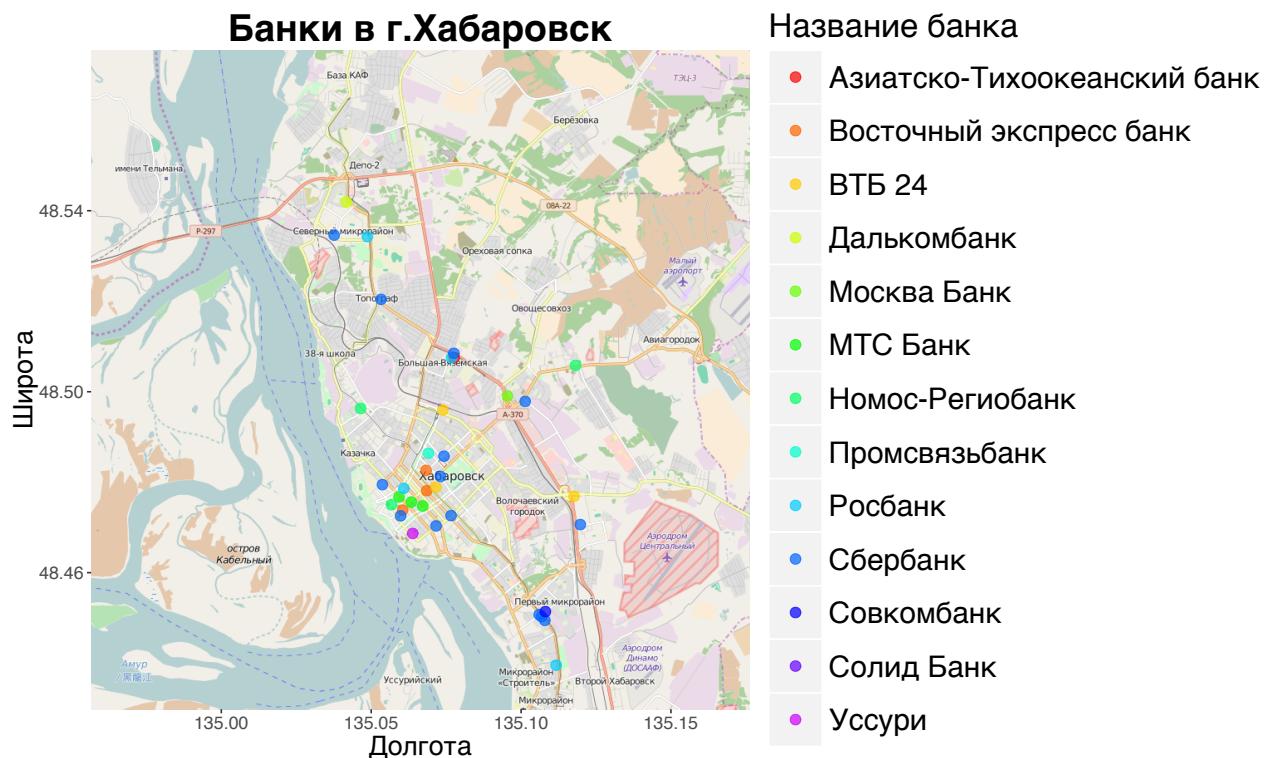
khab.banks.df <- as.data.frame(khab.banks@data)
khab.banks.df [, 4] <- banks.coords[, 1]
khab.banks.df [, 5] <- banks.coords[, 2]
names(khab.banks.df)[4] <- "coords.x"
names(khab.banks.df)[5] <- "coords.y"
head(khab.banks.df, 4)

```

##	id		name	amenity	coords.x	coords.y
## 42	3285118467	Азиатско-Тихоокеанский банк		bank	135.0777	48.50754
## 1	332389820	Восточный экспресс банк		bank	135.0683	48.48262
## 26	1969457318	Восточный экспресс банк		bank	135.0685	48.47811
## 45	3285254639	Восточный экспресс банк		bank	135.0605	48.47383

Как мы видим, в объекте **khab.banks.df** хранятся координаты и названия банков. Теперь мы можем построить карту (для удобства, полигоны будут удалены). На этот раз, сделаем её в OSM:

```
mapsource <- get_map(location = "Khabarovsk", zoom = 12, source = "osm")
khab.banks.points <- ggmap(mapsource) +
  geom_point(data = khab.banks.df,
             aes(coords.x, coords.y, color = name),
             size = 3, alpha = 0.7) +
  labs(x = "Долгота", y = "Широта",
       color = "Название банка") +
  #rainbow(x) задаёт палитру цветов.
  scale_color_manual(values = rainbow(15)) +
  ggtitle("Банки в г.Хабаровск")
khab.banks.points + plot5theme
```



Теперь мы имеем представление о том, как визуализировать точечные данные, а также познакомились с пакетом **ggmap**. Предлагаю сделать карту дорог Хабаровска. Процедура получения shape-файла в данном случае аналогична, только вместо точек нужно выбрать **полилинии** в соответствующем окне. Загружать данные и конвертировать их в базу данных уже не нужно, если Вы её не удалили. Из тегов возьмите **highway** и **name**. Далее нужно продублировать слой **Khab_polylines** дважды и сделать к каждому из полученных слоёв **отдельный** запрос ('**highway**' = '**primary**', '**highway**' = '**secondary**' и '**highway**' = '**tertiary**' соответственно) и сохранить как отдельные объекты, чтобы было проще их строить (пусть и в ущерб длине кода). В качестве объектов построения я выбрал три категории дорог по их значимости¹⁵.

Для начала, загрузим shape-файлы и сконвертируем их в dataframe:

```
khab.roads1 <- readOGR("Карты/Хабаровский_край", "primary_roads")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Хабаровский_край", layer: "primary_roads"
## with 208 features
## It has 3 fields

khab.roads2 <- readOGR("Карты/Хабаровский_край", "secondary_roads")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Хабаровский_край", layer: "secondary_roads"
## with 166 features
## It has 3 fields

khab.roads3 <- readOGR("Карты/Хабаровский_край", "tertiary_roads")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Хабаровский_край", layer: "tertiary_roads"
## with 253 features
## It has 3 fields

#конвертация
roads1.df <- fortify(khab.roads1)
roads2.df <- fortify(khab.roads2)
roads3.df <- fortify(khab.roads3)
```

Поскольку схематичная карта (построенная командой **ggplot()**) в данном случае не совсем удобна, я предлагаю вновь использовать **ggmap**.

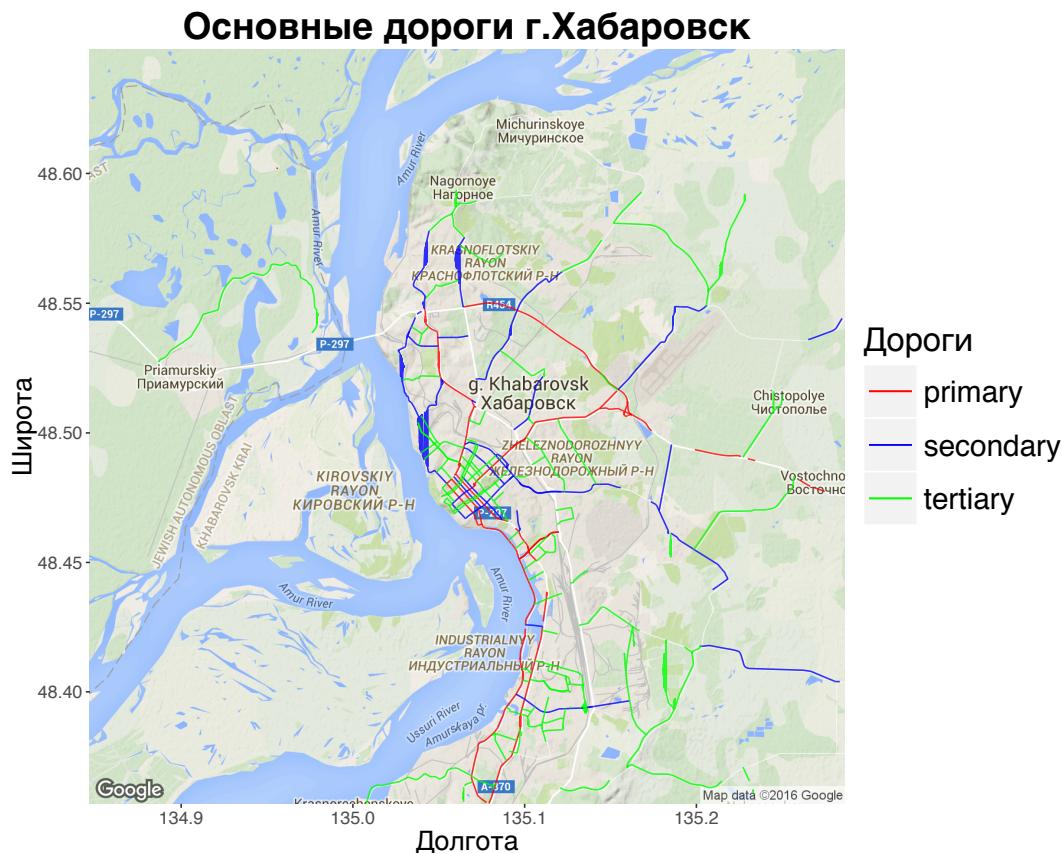
```
mapsource <- get_map(location = "Khabarovsk", zoom = 11)
khab.roads.map <- ggmap(mapsource) +
  geom_line(data = roads1.df,
```

¹⁵ Подробнее на <http://wiki.openstreetmap.org/wiki/RU:Key:highway>

```

aes(long, lat, group = group, color = "primary"),
alpha = 0.8) +
geom_line(data = roads2.df,
aes(long, lat, group = group, color = "secondary"),
alpha = 0.8) +
geom_line(data = roads3.df,
aes(long, lat, group = group, color = "tertiary"),
alpha = 0.8) +
labs(x = "Долгота", y = "Широта") +
ggtitle("Основные дороги г.Хабаровск") +
scale_colour_manual(values = c("red", "blue", "green"),
labels = c("primary", "secondary", "tertiary"),
name = "Дороги")
khab.roads.map + plot5theme

```



Таким образом, мы научились получать и строить объекты любого типа, используя как обычные графические средства R (`plot()`), так и более продвинутые (`ggplot2` и `ggmap`).

3.2.2 Рисование векторных слоёв

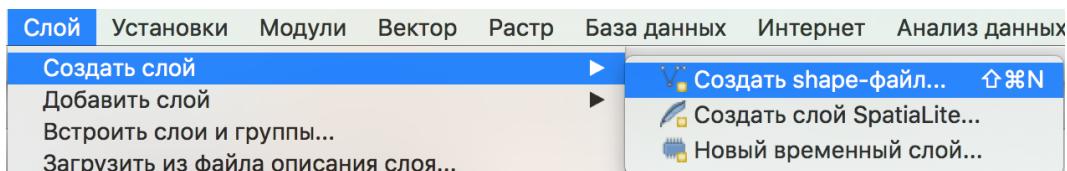
Бывают случаи, когда нужного shape-файла (или другого готового материала) по-просту не существует. К примеру, речь идёт об исследовании труднодоступного и ма-

лоисследованного участка, или же есть необходимость разделить какой-либо регион на сектора по особым правилам. В таких случаях нужно нарисовать необходимые полигоны, точки или линии самостоятельно, в зависимости от требуемых данных.

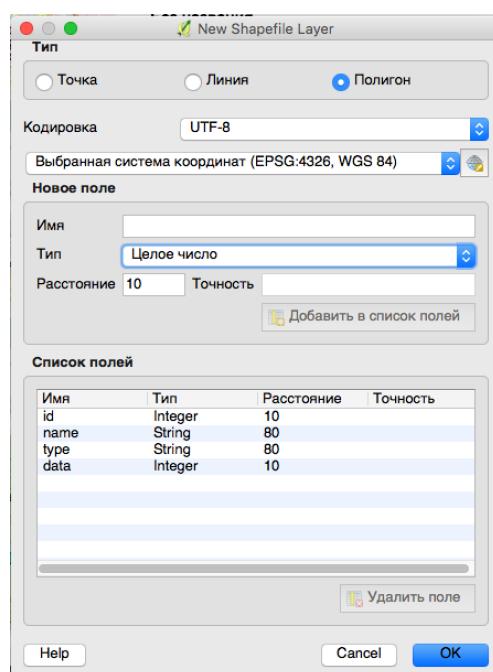
В этом разделе я разберу пример создания shape-файла, содержащего полигоны. Порядок действий для точечных и линейных данных аналогичен, за некоторыми исключениями¹⁶. В качестве абстрактного примера я придумал следующий сценарий: допустим, я хочу сопоставить численность популяций некоторого вида животных в разных секторах Большехехцирского заповедника (расположен к югу от Хабаровска) и сравнить её с популяциями в секторах вне заповедника (сравнение данных, относящихся к разным группам регионов, мы разберём подробнее в секции визуализации экономических данных (см. секцию 4.4)).

Приступим. Откроем QGIS и загрузим слой OSM (*Интернет -> OpenLayers plugin -> OpenStreetMap -> OpenStreetMap*). Затем, создадим новый векторный слой. Для этого перейдём по вкладкам меню:

Слой -> Создать слой -> Создать shape-файл...



Откроется диалоговое окно:

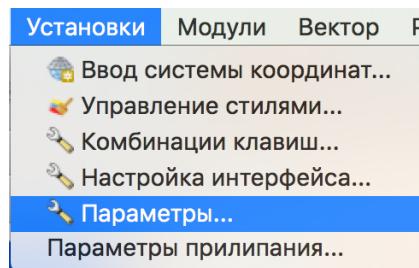


¹⁶Полная инструкция для всех видов данных доступна по ссылке: <http://gis-lab.info/qa/qgis-vector.html>

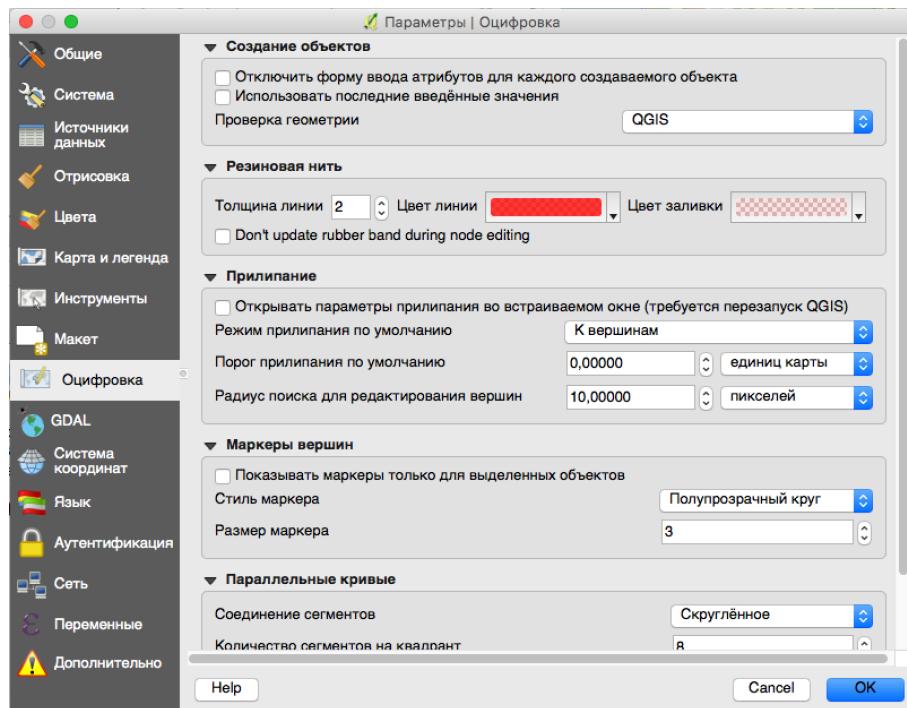
Здесь нам необходимо выбрать тип данных (в нашем случае это будут полигоны). Далее, чтобы создать файл, нужно добавить хотя бы одно поле (поле id, как правило, ставится по умолчанию), иначе создаваемые объекты (например, полигоны), будет невозможно идентифицировать по какому-либо признаку. Я добавил поля name (имя), type (тип) и data (данные, то есть численность популяции). При желании, поле с данными можно создать в R и задать им формулу генерации (или добавить из другой таблицы). Поля лучше продумать заранее. Определившись, щёлкаем по кнопке OK. Новый слой отобразится на панели слоёв.

При необходимости, можно настроить параметры построения (оцифровки). Для этого, перейдём по вкладкам меню:

Установки -> Параметры...

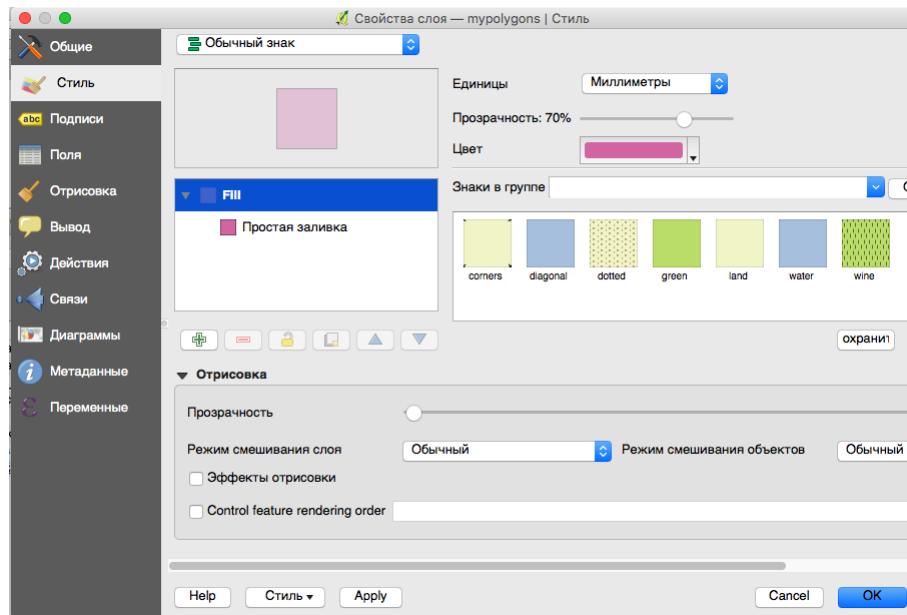


и откроем вкладку «Оцифровка»:

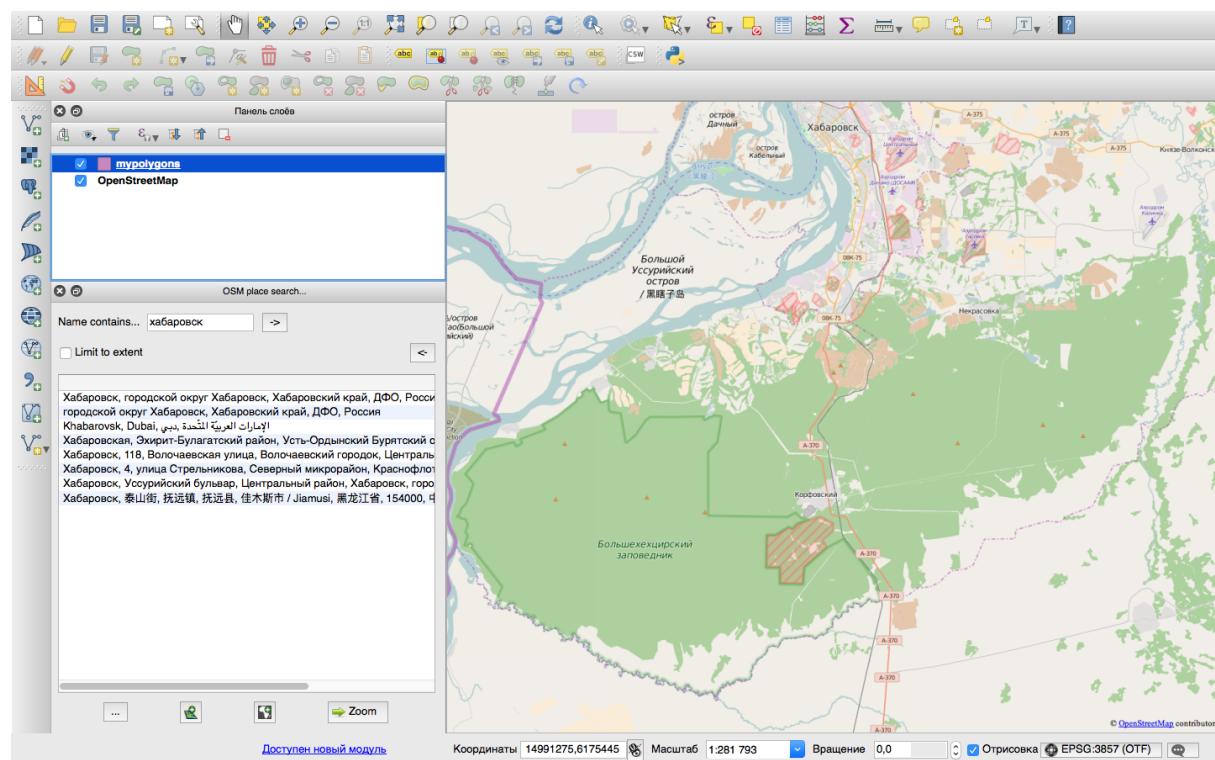


Здесь нас интересуют три параметра для настройки. Первое — ширина Резиновой нити. Резиновая нить — линия между курсором и вершинами (последней и начальной). Также можно настроить прилипание — автоматическое нацеливание на вершину или

сегмент между вершинами. И наконец, измените маркер на удобный Вам. Щёлкнув правой клавишей мыши по слою с будущими полигонами, зайдите на вкладку «Стиль» и измените значение прозрачности до 60–80%, чтобы карта просвечивала через полигоны.

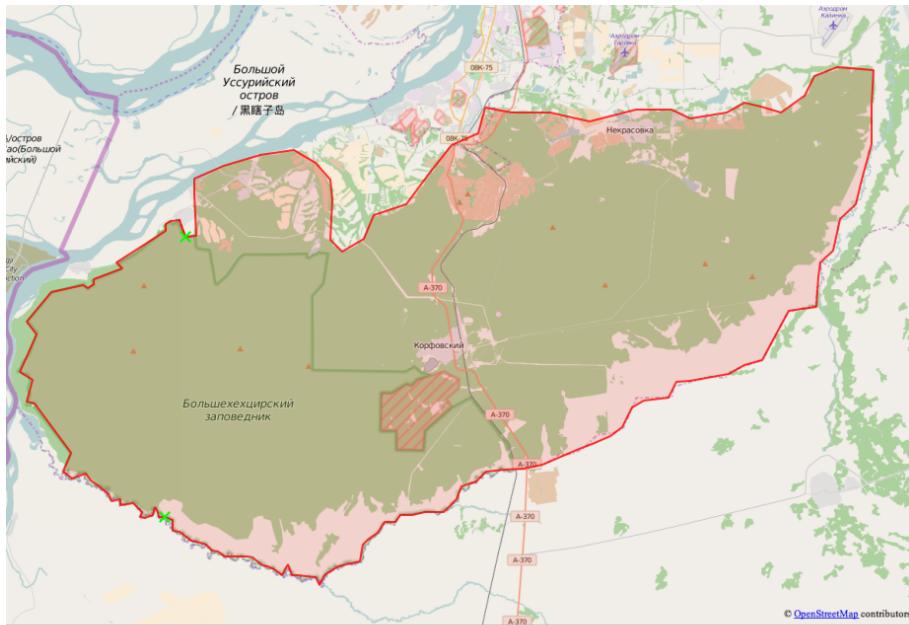


Итак, наше рабочее пространство выглядит примерно так:

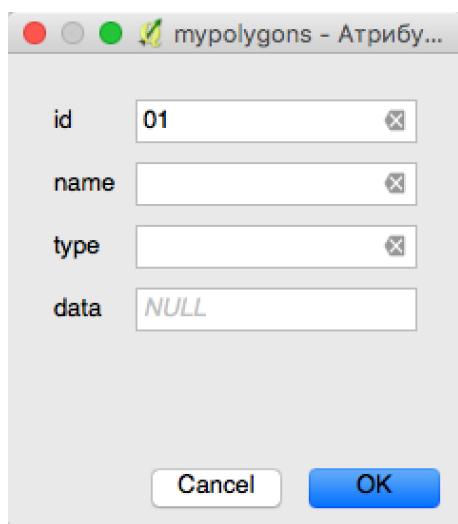


Отчётчиво видно, что граница Большехцирского заповедника выделена зеленым цве-

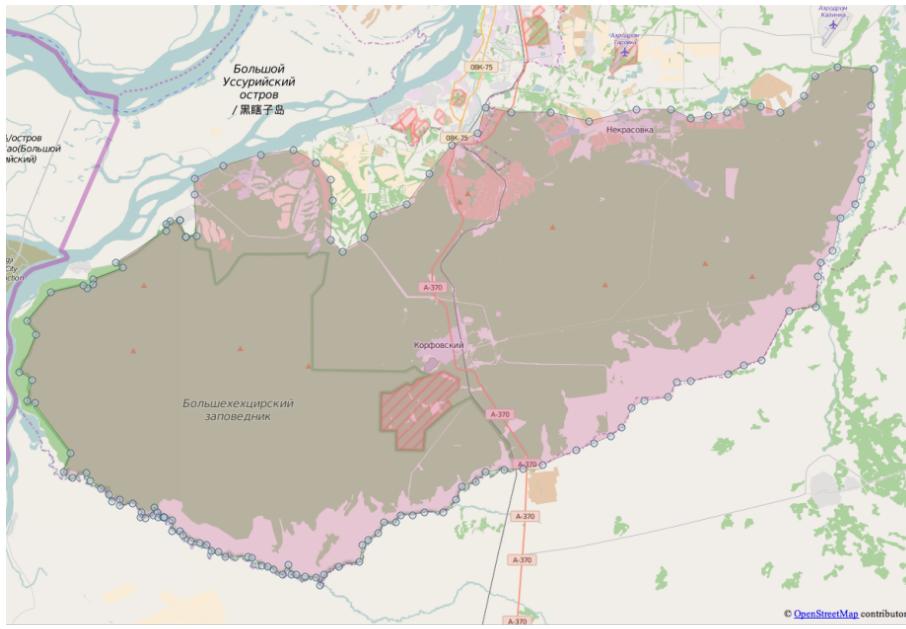
том. Мысленно разделим эту область на регионы. Для перехода в режим редактирования нажмём на иконку с жёлтым карандашом в верхнем левом углу . Выберем инструмент *Добавить объект* и приступим к рисованию. Будет лучше, если вначале мы нарисуем большой полигон, покрывающий всю область, а затем поделим его на сектора. В противном случае между границами секторов могут быть пустоты, что некрасиво. Выделение образуется из опорных точек, которые ставятся левой клавишей мыши.



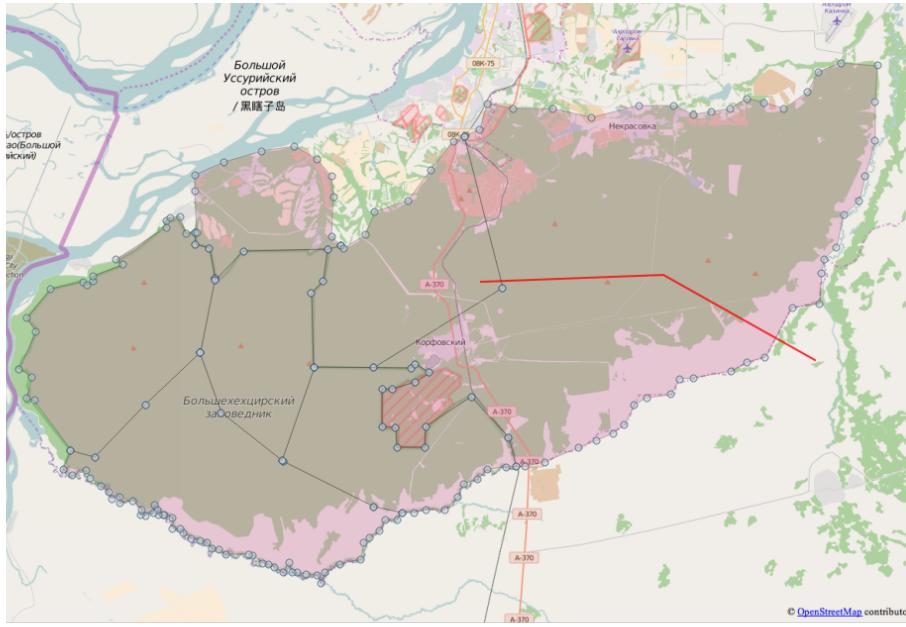
Чтобы закончить выделение, нужно нажать на правую клавишу. После нажатия, появится вот такое окно:



Пока его можно не заполнять, чтобы введённые значения не унаследовались всеми секторами, которые мы создадим инструментом *Разбить объекты* . А пока мы получили следующий результат:



Теперь нам нужно поделить большой полигон на части. Для начала, руководствуясь просвечивающей зеленой границей заповедника, отделим его. Затем, поделим два разных по типу региона на сектора. Для этого достаточно построить ломаную, чтобы она пересекала две уже существующие границы секторов, например вот так:



В результате мы получим несколько маленьких полигонов с общими границами (что было бы крайне трудно осуществимым, если бы полигоны рисовались отдельно друг от друга). Если объектов слишком много, то их можно объединить, выделив их инструментом на иконке () и «спив» Объединением (). Нам осталось лишь внести данные. Для этого щёлкнем по иконке () и нажмём на сектор. Появится панель «Результат определения». Отметим флажок «Открывать форму». Теперь, при щелчке

по сектору, будет отображаться уже знакомое нам окно с вводом данных. Введём данные для всех объектов. В результате мы получим примерно такую таблицу атрибутов (data я вводил случайным образом, насколько это понятие применимо к человеку):

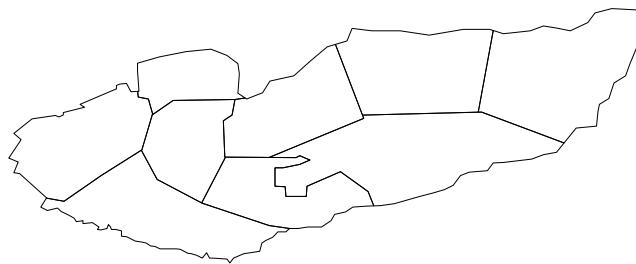
	id	name	type	data
0	1	БХЗ-01	Заповедник	34
1	7	НЭТ-03	Не заповед...	56
2	2	БХЗ-02	Заповедник	35
3	4	БХЗ-04	Заповедник	24
4	3	БХЗ-03	Заповедник	26
5	5	НЭТ-01	Не заповед...	12
6	6	НЭТ-02	Не заповед...	34
7	9	НЭТ-05	Не заповед...	46
8	8	НЭТ-04	Не заповед...	27

Проверим наш результат в R:

```
zapoved <- readOGR("Карты/Хабаровский_край", "mypolygons")

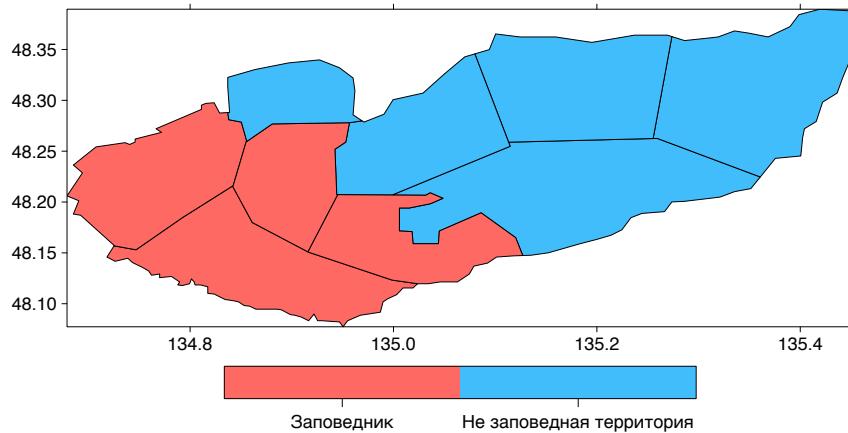
## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Хабаровский_край", layer: "mypolygons"
## with 9 features
## It has 4 fields

par(mar = c(0, 0, 1, 0))
plot(zapoved)
```



Как видим, получился уже привычный shape-файл с полигонами. Теперь отобразим имеющиеся на нём данные. Для этого мы воспользуемся новой для нас функцией `spplot()`, содержащейся в пакете `sp`, который загружается вместе с пакетом `rgdal`. В отличии от `plot()` она не способна возвращать графики и диаграммы и предназначена исключительно для работы с пространственными данными.

```
spplot(zapoved, "type", col.regions = c("#FE6A63", "#42BDFB"),
       par.settings = list(fontsize = list(text = 25)),
       scales = list(draw = TRUE),
       colorkey = list(space = "bottom", height = 0.6))
```



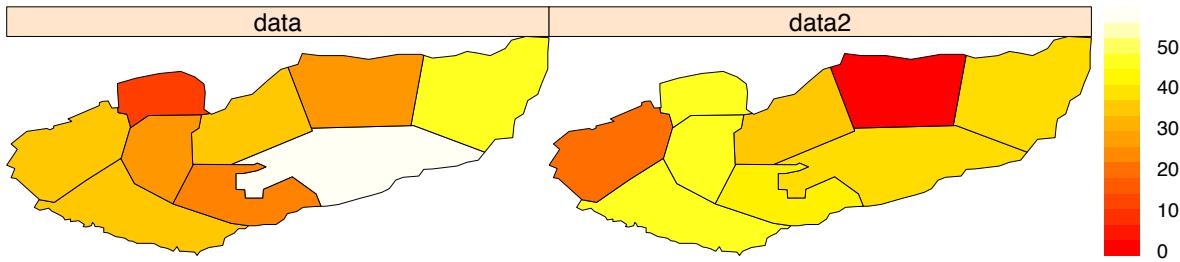
Мы получим знакомую нам карту, но уже с автоматически готовой легендой. Замечу, что при отображении важно указать отображаемую переменную (в нашем случае это `type`), иначе получим карты по всем переменным. Чтобы показать больше возможностей данной функции, сгенерируем ещё один вектор данных (столбец). Для этого, выполним следующие команды:

```
#Сгенерируем 9 случайных действительных чисел от 0 до 50
zapoved@data[, 5] <- runif(9, 0, 50)
names(zapoved@data)[5] <- "data2"
head(zapoved@data)

##   id    name          type  data  data2
## 0  1  БХ3-01  Заповедник  34 19.16884
## 1  7  НЗТ-03  Не заповедная территории  56 38.91783
## 2  2  БХ3-02  Заповедник  35 45.99400
## 3  4  БХ3-04  Заповедник  24 40.75088
## 4  3  БХ3-03  Заповедник  26 47.74708
## 5  5  НЗТ-01  Не заповедная территории  12 45.18828
```

Теперь у нас есть два набора данных. Построим график со сравнением (заодно удалим оси координат командой `list(axis.line = list(col = NA))`). Замечу, что в зависимости от заданного пространства для построения, график может быть как в одну, так и в несколько колонок, причем распределение и построение делается автоматически. Также стоит учесть, что количество цветов в палитре (в нашем случае это `heat.colors()`) должно соответствовать данным, поскольку шкалы для категорических (`categoryc`), и числовых (целочисленных (`integer`) и непрерывных (`continuous`)) величин отличны, поэтому не стоит комбинировать карты с разными типами данных.

```
spplot(zapoved[c("data", "data2")], col.regions = heat.colors(50),
       par.settings = list(fontsize = list(text = 25),
                           axis.line = list(col = NA)))
```

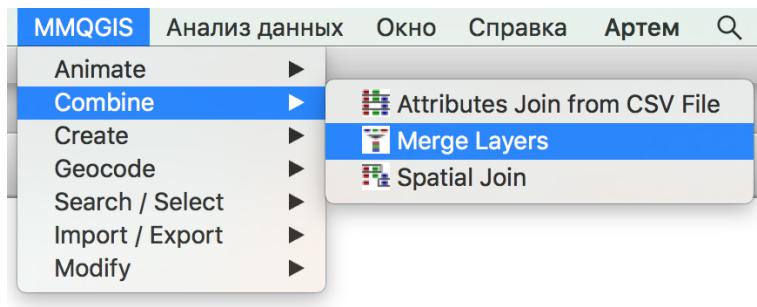


Таким образом, мы получили две картограммы, позволяющие провести сравнение распределений `data` и `data2`.

3.2.3 Получение административных границ регионов

В отличие от данных, представленных точками или линиями, получение полигонов (а именно административных границ регионов) лучше осуществить другим способом. В противном случае, при попытке получить данные OSM с рабочей области, вмещающей в себя всю территорию государства (тем более такого, как Россия), придётся загрузить крайне большое количество информации. Чтобы этого избежать, слой с границами можно получить напрямую с помощью плагина OSM Place search. В данном случае я буду получать границы районов Владивостокского городского округа (в связи с их небольшим количеством, а также для разнообразия примеров).

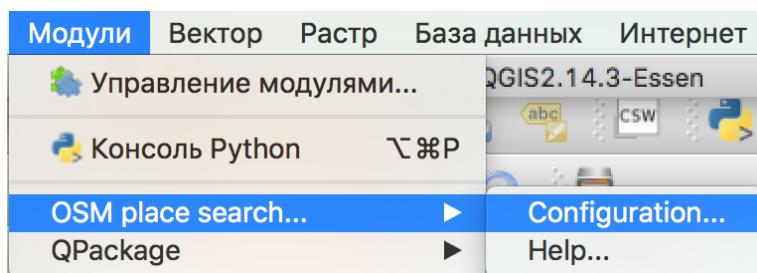
Но сперва нам понадобится новый модуль, который даст нам дополнительные инструменты для работы со слоями, а именно объединение слоев в один shape-файл. Этот модуль называется MMQGIS, получить его можно тем же путём, что и другие модули. После установки он отобразится в виде отдельной вкладки меню:



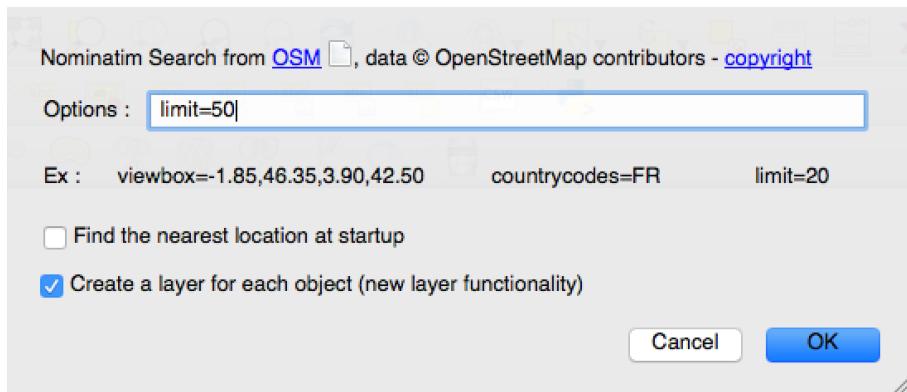
Далее я рекомендую установить систему координат проекта на WGS 84 (EPSG:4326). Объекты в этой системе могут показаться немного сплющенными, но в определении координат будут использоваться широта и долгота, что избавит нас от необходимости переопределять проекцию при работе с shape-файлом в R).

И наконец, зайдя в меню

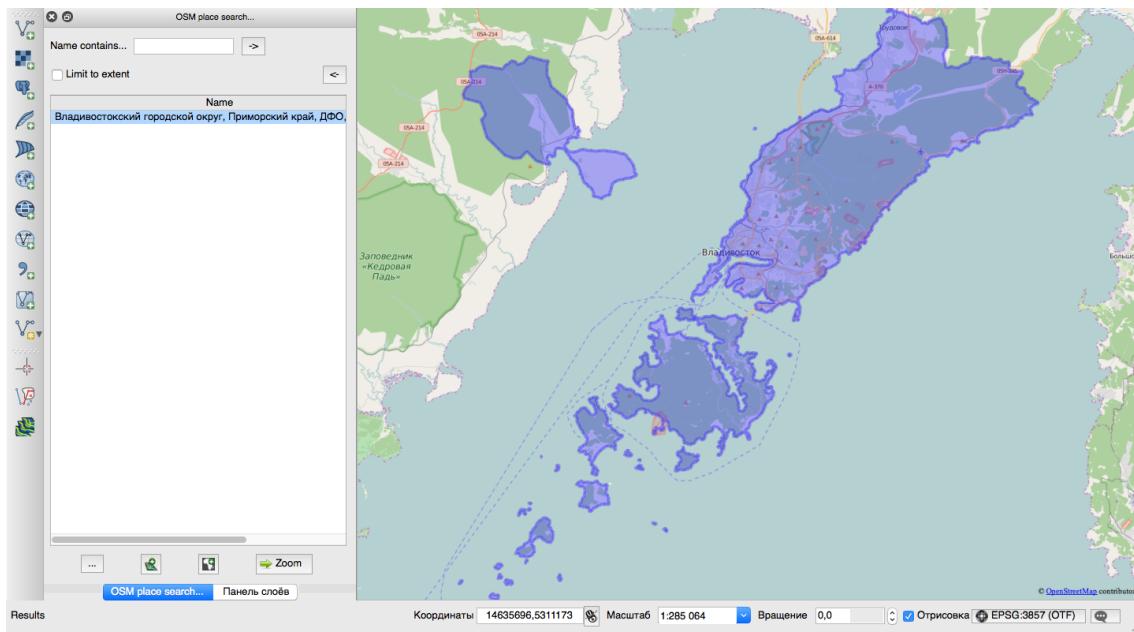
Модули -> OSM place search... -> Configuration...



в поле Options вводим limit=50. Это даст нам больше результатов поиска и уменьшит шанс того, что искомого района не окажется среди выведенных.



Теперь можно приступить к получению административных границ. Для этого, аналогично тому, как мы это делали в предыдущих подсекциях, вводим имя искомого района в поле поиска панели **OSM place search...** (для удобства в ориентировке можно подгрузить слой с картой OpenStreetMap, но учтите, что система координат сменится на WGS 84/Pseudo Mercator (EPSG:3857)). Обращаю Ваше внимание, что понятия «Владивосток» и «Владивостокский городской округ» отличны друг от друга, но для краткости будет «Владивосток».



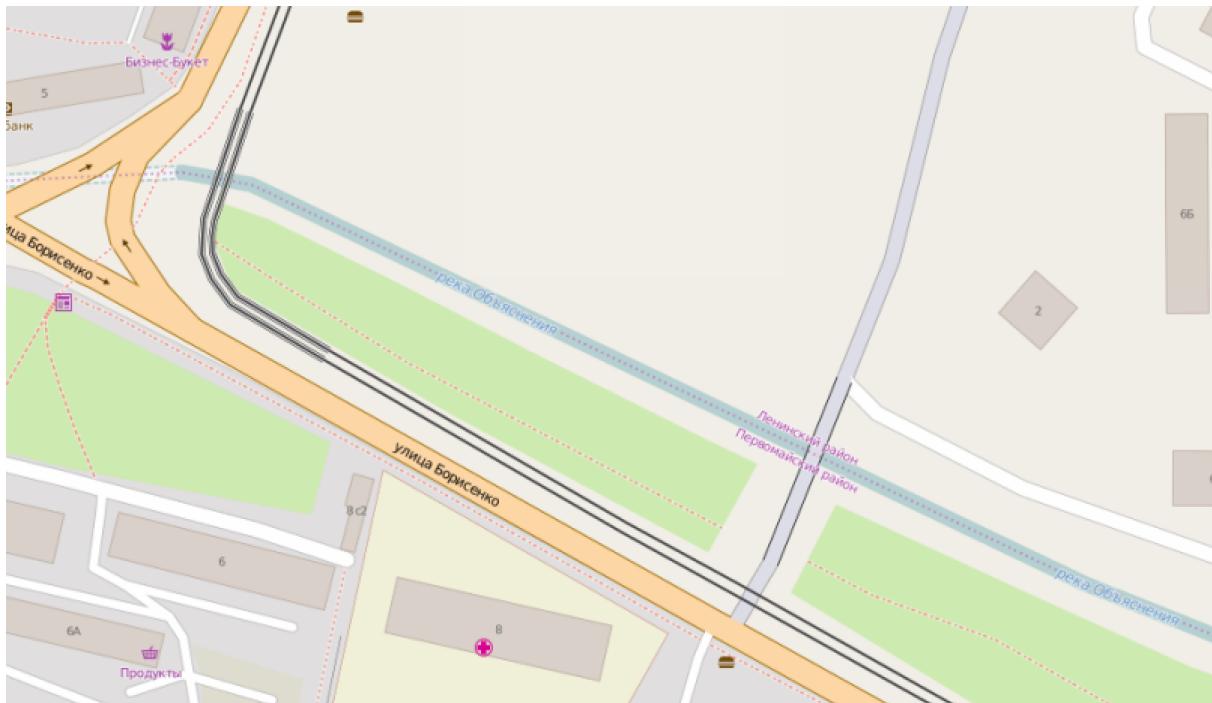
Чтобы сохранить выделение как слой, щёлкнем по иконке , расположенной в нижней части панели. Если система координат по умолчанию отсутствует, то появится окно, в котором будет предложено её выбрать. Выбираем WGS 84 (или WGS 84/Pseudo Mercator, если используете карту OSM на фоне). В таком случае, если слой не отображается, щёлкаем по нему правой клавишей мыши и в *Свойства -> Общие* выбираем эту систему координат). Полученному слою лучше дать какое-нибудь имя и сделать полупрозрачным, он будет служить фоном для районов города.

Аналогичным способом мы можем получить слои районов Владивостока (как и любого другого города или региона). Отмету, что административные районы Владивостока не покрывают всю территорию Владивостокского городского округа, поэтому некоторые места могут отсутствовать (к примеру, острова к югу от материка и небольшая область соседствующая с Фрунзенским районом, отмечена на скриншоте ниже). В том редком случае, когда поиск не находит запрашиваемый район, убедитесь в следующем:

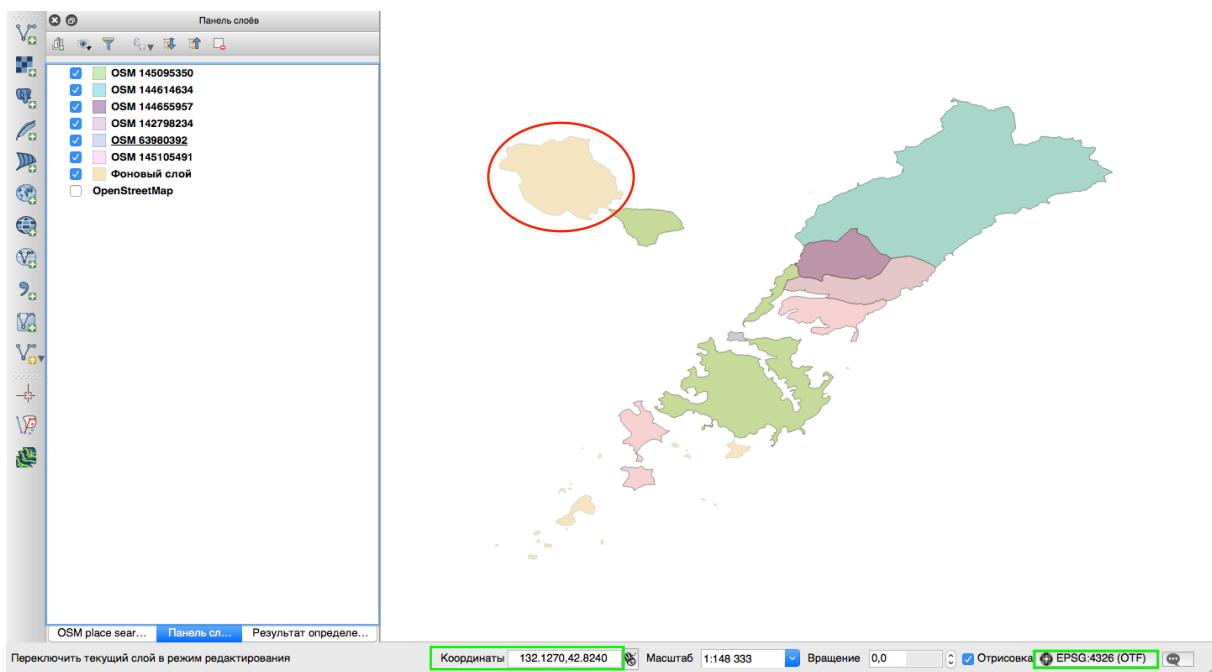
- Отсутствие опечаток
- Соответствие введённого значения официальному названию района (то есть, если вы хотите получить границы Ленинского района, нужно вводить «Ленинский район», так как одного лишь «Ленинский» недостаточно)
- В настройках модуля OSM place search в строке options указано limit=50

Если же все вышеперечисленные критерии выполнены, попробуйте ввести другое принятное название района (например, чтобы найти республику Северная Осетия, нужно ввести «республика Северная Осетия - Алания»), дополнить поиск (вместо «Ленинский район» ввести «Ленинский район, Владивосток»), увеличить лимит результатов поиска или же ввести имя региона на английском. Также можно попробовать приблизить карту и найти название района (в случае с нашим примером — остров Елены) В крайнем случае границы можно отрисовать вручную: для этого на

слой OpenStreetMap в достаточно большом масштабе нужно найти границы районов (фиолетовая пунктирная линия, как на скриншоте ниже). Замечу, что это очень трудоёмкий вариант.

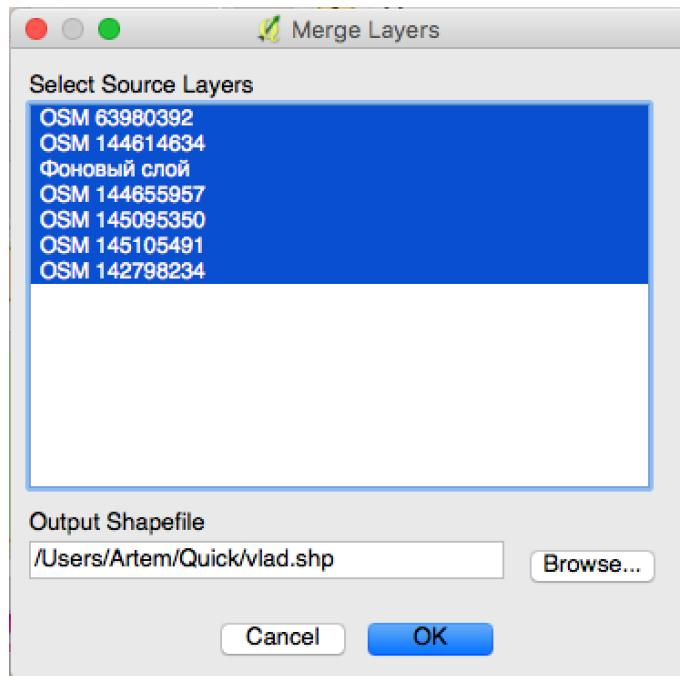


Получив слои границ всех районов города, при необходимости, нужно перевести их обратно в систему координат WGS 84 (EPSG:4326). Для этого щёлкнем по соответствующему полю в нижней правой стороне окна и выберем систему в открывшемся окне. Как мы можем заметить, теперь координаты измеряются в градусах:



Чтобы сохранить слои, перейдём по вкладкам меню:

MMQGIS -> Combine -> Merge Layers



В открывшемся окне выделяем нужные слои и выбираем директорию для экспорта shape-файла. Также он отобразится на панели слоев. Теперь можно открыть таблицу атрибутов слоя и отредактировать её на свое усмотрение. Напоследок, если при создании файла Вы использовали слои с проекцией Pseudo Mercator, то слой нужно сохранить с новой проекцией. Для этого, сперва убедитесь в том, что система координат проекта WGS 98, и затем, щёлкнув правой клавишей мыши по слою файла и выбрав «Сохранить как», из списка систем координат выберите WGS 84 (EPSG:4326) и нажмите OK.

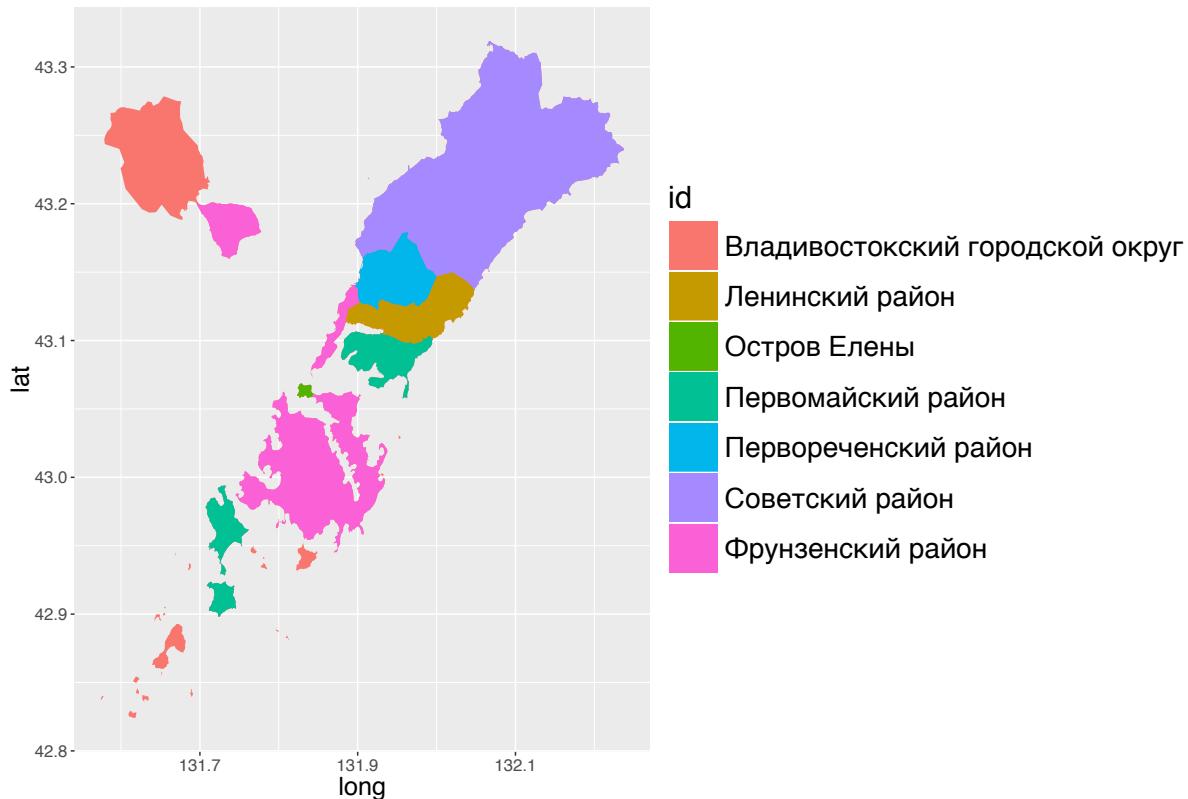
В R результат выглядит следующим образом (в QGIS я подредактировал имена районов для компактности легенды):

```
vlad <- readOGR("Карты/Владивосток", "vlad")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Владивосток", layer: "vlad"
## with 7 features
## It has 2 fields

vlad.df <- fortify(vlad, region = "name")
```

```
ggplot(vlad.df,
       aes(long, lat, group = group, fill = id)) +
  geom_polygon() + plot5theme
```



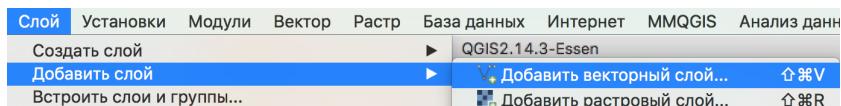
3.2.4 Упрощение карт

В данной подсекции пойдёт речь о картографической генерализации¹⁷, а именно об упрощении границ (другими словами, об упрощении геометрии слоёв границ). Это актуально для тех случаев, когда на карте (или картограмме) большое число полигонов или излишне подробная граница для целевого масштаба. В этих случаях полигоны, изображающие границы регионов, содержат большое количество вершин, данные координат которых необходимо обработать при прорисовке, замедляя составление картограмм. С технической точки зрения, процесс упрощения границ состоит в том, что часть этих вершин удаляются. В качестве примера я возьму файл с границами субъектов Федерации из проекта OSM в формате geoJSON с сайта GISGeo (<http://gisgeo.org/index.php?id=7#GovOpen>), поскольку это достаточно большой (27Мб) файл, который весьма долго обрабатывается в R и, следовательно, требует упрощения. Для упрощения я буду использовать QGIS, однако замечу, что существуют другие варианты генерализации карт¹⁸.

¹⁷Картографическая генерализация — упрощение карты в рамках её цели.

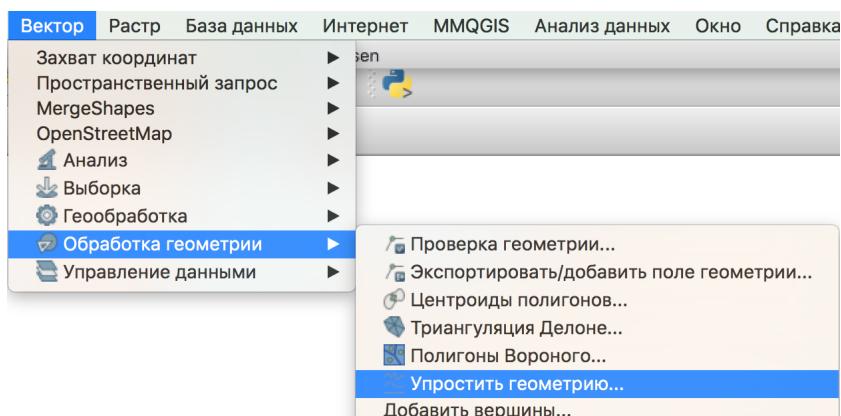
¹⁸Использовать GRASS или функцию `thinnedSpatialPoly()` в R.

Загрузим файл и откроем его в QGIS, как слой (*Слой -> Добавить слой -> Добавить векторный слой...*).

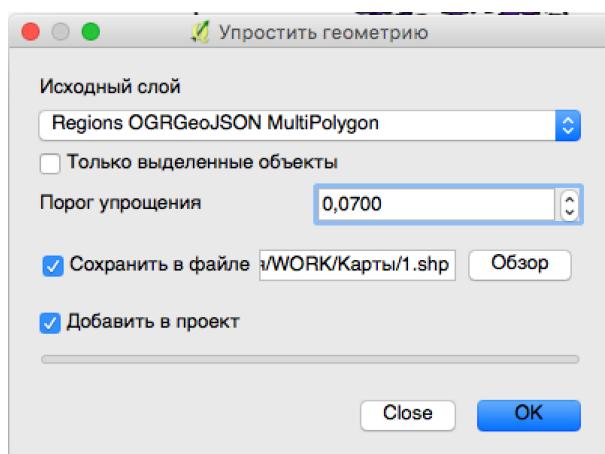


Как и прошлой подсекции, убедитесь в том, что система координат слоя WGS 84. Теперь выполним упрощение границ, для этого перейдём по вкладкам меню:

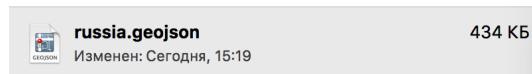
Вектор -> Обработка геометрии -> Упростить геометрию...



В открывшемся окне выбираем слой, геометрию которого мы хотим упростить и порог упрощения. Последний зависит от системы координат, в которой загружен слой, то есть, в случае с WGS 84, в которой координаты измеряются в градусах, порог 0.1 будет означать одну десятую градуса, что в зависимости от широты может составлять от 111 км до 0 км. То есть, выбрав в качестве порога 0.1 мы удалим все вершины, расстояние между которыми меньше чем 0.1° . От себя замечу, что в R достаточно быстро обрабатываются файлы с 10000–25000 точками (в зависимости от вычислительной мощности компьютера), поэтому стоит выбрать значение, которое сократит количество вершин до этого числа. В нашем случае это около 0.07. Результат сохраним в shp (за отсутствием выбора) и добавим в проект.



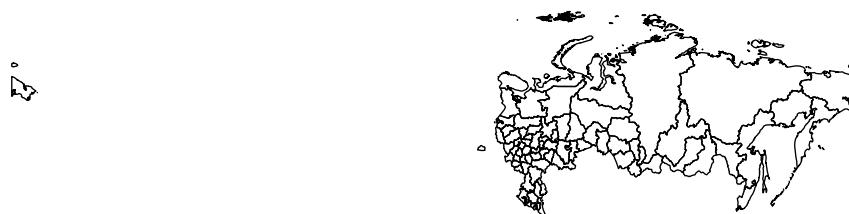
Отмечу, что данный способ генерализации не идеален, так как упрощает полигоны по-отдельности, тем самым создавая щели между ними. Однако, при определённом масштабе это будет незаметно, а потери оправдают себя уменьшенней скоростью прорисовки. В любом случае, полученный слой сохраняется в формате geoJSON (не забудьте о системе координат). Назовём его «russia.geojson». Размер файла с границами существенно уменьшился: теперь он занимает всего 434Кб.



Проверим его в R.

```
json.file <- readOGR("Карты/geojson/russia.geojson", "OGRGeoJSON")  
  
## OGR data source with driver: GeoJSON  
## Source: "Карты/geojson/russia.geojson", layer: "OGRGeoJSON"  
## with 83 features  
## It has 2 fields  
  
par(mar = c(0,0,1,0))  
plot(json.file, main = "Карта России", ylim = c(40, 85))
```

Карта России



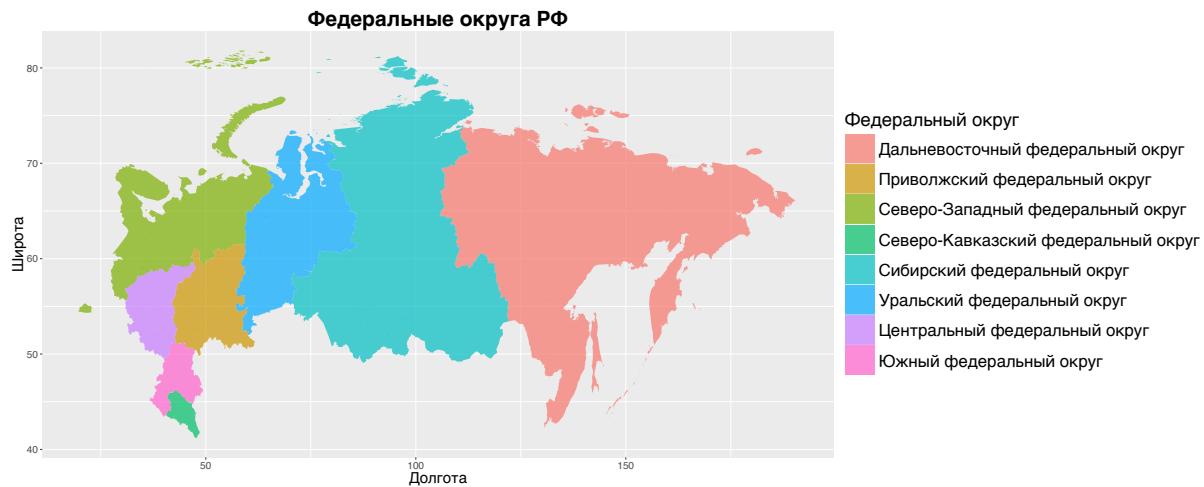
У нас получилась вполне корректная карта, но поскольку Чукотка и остров Врангеля пересекают 180° меридиан, по правилам системы координат её разрезает на две части, что портит вид карты и представляет её в меньшем масштабе. Этот недостаток можно исправить непосредственно в R с помощью следующего блока команд, который я для воспроизводимости преобразовал в функцию (команды для «сшивания» Чукотки взяты отсюда: <https://github.com/Sobach/R.rus.map.zoom/blob/master/mapzoom.R>). Также эти функции доступны в пакете rusmaps:

```
#Данная функция исправляет отрицательные значения координат,
#преобразуя их в положительные
longitude_correction <- function(x) {
  if (x < 0) {
    x <- 359.999 + x
  }
  return(x)
}

#Данная функция принимает объект SpatialPolygonsDataFrame
#и "сшивает" полигоны, пересекающие 180 меридиан
restore180 <- function(y) {
  for (i in 1:length(y@polygons)) {
    for (j in 1:length(y@polygons[[i]]@Polygons)) {
      y@polygons[[i]]@Polygons[[j]]@coords[, 1] <- sapply(
        y@polygons[[i]]@Polygons[[j]]@coords[, 1],
        longitude_correction)
    }
  }
  return(y)
}
```

Теперь нам осталось лишь применить функцию `restore180()`, чтобы соединить разрозненные части карты:

```
json.file <- restore180(json.file)
json.map <- fortify(json.file, region = "ADM3_NAME")
rus.map <- ggplot(json.map,
                  aes(long, lat, group = group, fill = id)) +
  geom_polygon(alpha = 0.7) +
  labs(x = "Долгота", y = "Широта", fill = "Федеральный округ") +
  ggtitle("Федеральные округа РФ")
rus.map + plot5theme
```



У нас получилась достаточно детализированная и быстро загружающаяся карта.

Также качестве альтернативы можно использовать mapshaper.org или другие веб-сервисы. Mapshaper, помимо конвертации, позволяет осуществить упрощение карты разными методами (алгоритмами). Для этой цели есть кнопка «simplify» в верхней правой части окна. Есть возможность выбрать степень упрощения с помощью ползунка.

4 Визуализация экономических данных

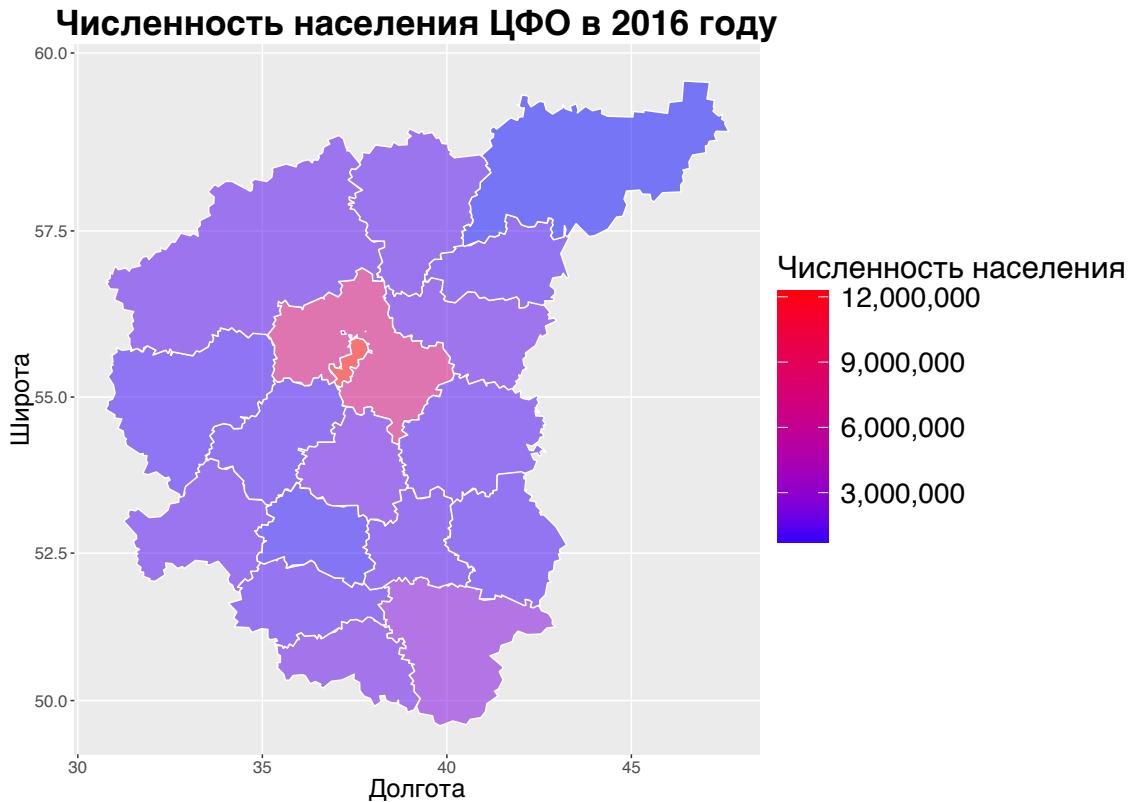
В данной секции пойдёт речь о способах визуализации пространственных данных. До этого момента мы создавали карты и легенды по категорическим данным (названиям районов и банков). Но помимо имени объект может представлять и какую-то численную информацию, которую нужно отобразить в рамках исследования. Для разных данных можно использовать разные способы. С некоторыми из них Вы можете ознакомиться ниже.

4.1 Построение карт с градиентом

Градиент лучше всего подходит для непрерывных и целочисленных данных с большим размахом, поскольку без объединения таких данных в группы их нельзя поделить на категории. В качестве примера мы рассмотрим распределение населения по Центральному федеральному округу.

```
centpop <- readOGR("Карты/Федеральные_округа", "ЦФО")  
  
## OGR data source with driver: ESRI Shapefile  
## Source: "Карты/Федеральные_округа", layer: "ЦФО"  
## with 18 features  
## It has 6 fields
```

```
#Представит метки легенды в целочисленном виде с labels = comma  
library("scales")  
pop.df <- fortify(centpop, region = "name")  
#Командой merge() можно обединить данные из разных таблиц по общему полю  
pop.df <- merge(pop.df, centpop@data, by.x = "id", by.y = "name")  
pop.map <- ggplot(pop.df,  
                   aes(long, lat, group = group, fill = pop_2016)) +  
  geom_polygon(color = "white", alpha = 0.5) +  
  coord_map() +  
  labs(x = "Долгота", y = "Широта",  
       fill = "Численность населения") +  
  ggtitle("Численность населения ЦФО в 2016 году") +  
  scale_fill_gradient(low = "blue",  
                      high = "red",  
                      labels = comma)  
pop.map + plot5theme
```



Как мы можем заметить, больше всего населения приходится на Московскую область и город Москва.

4.2 Построение карт с точечными данными

В случаях, когда данные представлены в виде точек, имеет смысл отобразить их размер в зависимости от величин данных, которые они содержат. Рассмотрим пример с населением городов миллионеров (для экономии места, я буду иногда скрывать результаты функции `readOGR()`):

```
citypop <- readOGR("Карты/Города_миллионники", "города_миллионники")
russia <- readOGR("Карты/Россия/w_fd", "рос_фо")

rus.df <- fortify(russia, region = "id")
city.df <- as.data.frame(citypop@data)
city.coords <- as.data.frame(citypop@coords)
city.df[, 4] <- city.coords$coords.x1
city.df[, 5] <- city.coords$coords.x2
names(city.df)[4] <- "coords.x"
names(city.df)[5] <- "coords.y"
```

```

#geom_text добавит подписи к точкам
citypop.map <- ggplot(rus.df) +
  geom_polygon(color = "white", fill = "#7eb92d",
               aes(long, lat, group = group)) +
  geom_point(data = city.df,
             aes(coords.x, coords.y, size = pop_2015),
             color = "#b92f51", alpha = 0.7) +
  geom_text(data = city.df, angle = 45,
            nudge_y = 0.6, nudge_x = -0.5,
            aes(coords.x, coords.y, label = name)) +
  coord_map(xlim = c(0, 100),
             ylim = c(40, 70)) +
  labs(x = "Долгота", y = "Широта",
       size = "Численность населения") +
  scale_size(labels = comma) +
  gtitle("Численность населения в городах миллионерах в 2015 году")

citypop.map + plot5theme + theme(legend.position = "bottom")

```



4.3 Построение карт с уровнями

Уровни нужны в случаях, когда есть необходимость показать плотность распределения данных. В нашем случае, мы можем использовать уже известные нам данные о местоположении банков в Хабаровске. Воспользуемся следующими командами:

```
levelmap <- ggplot() +  
  geom_polygon(data = map.df, alpha = 0.7, color = "white",  
               aes(long, lat, group = group, fill = id)) +  
  geom_point(data = khab.banks.df,  
             aes(coords.x, coords.y), size = 3) +  
  geom_density2d(data = khab.banks.df,  
                 aes(coords.x, coords.y)) +  
  coord_map(xlim = c(134.85, 135.25)) +  
  labs(x = "Широта", y = "Долгота",  
       fill = "Название района") +  
  ggtitle("Банки в г.Хабаровск")  
  
levelmap + plot5theme
```



4.4 Сравнение двух регионов

Имеет место быть случай, когда необходимо сравнить два региона. В таком случае, нам понадобится пакет **gridExtra**, который позволяет разместить карты и графики

по нескольким столбцам и(или) строкам. Чтобы это осуществить, нам понадобятся следующие команды:

```
uralpop <- readOGR("Карты/Федеральные_округа", "УФО")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Федеральные_округа", layer: "УФО"
## with 6 features
## It has 6 fields

pop.df1 <- fortify(uralpop, region = "name")
pop.df1 <- merge(pop.df1, uralpop@data, by.x = "id", by.y = "name")
#При условии одинаковой заливки, легенду у одной из карт можно убрать
pop.map1 <- ggplot(pop.df1, aes(long, lat, group = group, fill = pop_2016)) +
  geom_polygon(color = "white") +
  coord_map() +
  labs(x = "Долгота", y = "Широта",
       fill = "Численность населения") +
  ggtitle("Численность населения УФО в 2016 году") +
  theme(legend.position = "none")

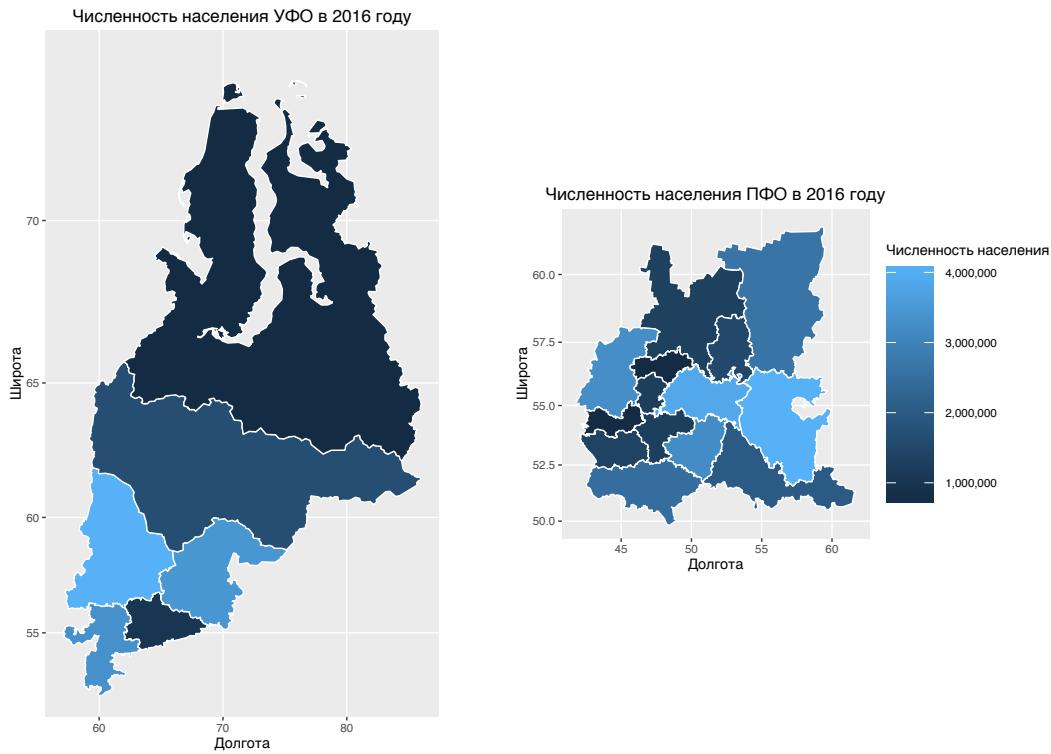
volgapop <- readOGR("Карты/Федеральные_округа", "ПФО")

## OGR data source with driver: ESRI Shapefile
## Source: "Карты/Федеральные_округа", layer: "ПФО"
## with 14 features
## It has 6 fields

pop.df2 <- fortify(volgapop, region = "name")
pop.df2 <- merge(pop.df2, volgapop@data, by.x = "id", by.y = "name")

pop.map2 <- ggplot(pop.df2, aes(long, lat, group = group, fill = pop_2016)) +
  geom_polygon(color = "white") +
  coord_map() +
  labs(x = "Долгота", y = "Широта",
       fill = "Численность населения") +
  scale_fill_gradient(labels = comma) +
  ggtitle("Численность населения ПФО в 2016 году")

grid.arrange(pop.map1, pop.map2, ncol = 2)
```



Таким же образом можно разместить карты с разными типами данных (категорическими и непрерывными), или добавить к карте гистограмму или диаграмму.

4.5 Сравнение наборов точечных данных

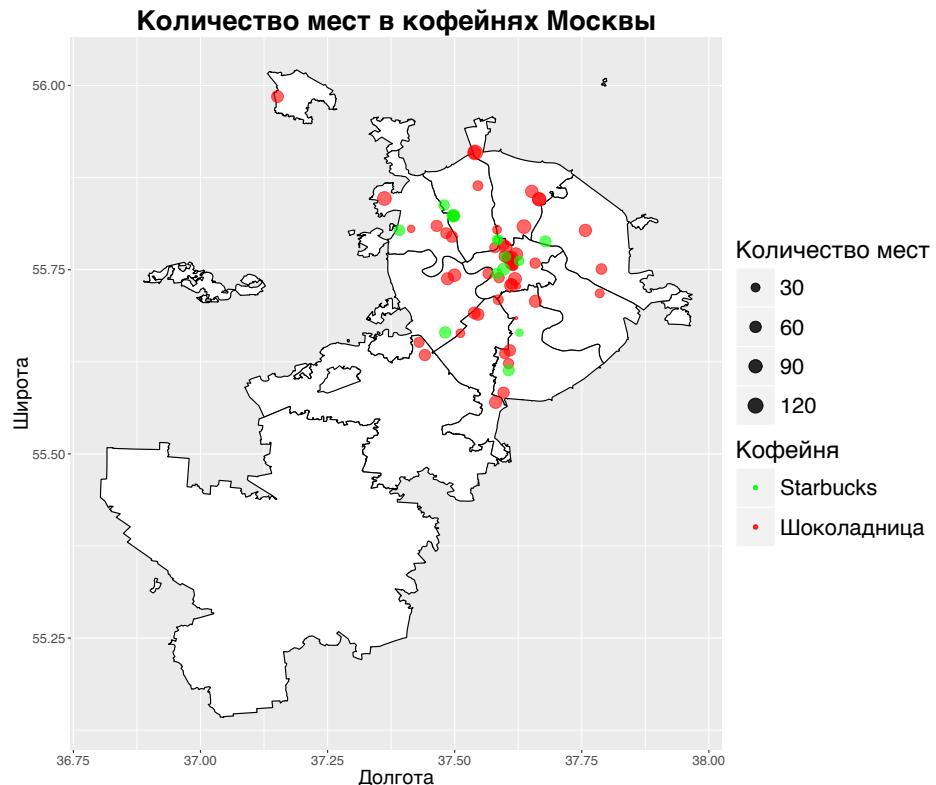
Рассмотрим такой случай: у нас есть два набора данных по количеству мест в кафе сети Шоколадница и кафе сети Starbucks. Эти данные можно разделить на две группы, и чтобы нам была понятна принадлежность кафе к первой или второй группе, точки нужно закрасить соответствующими цветами. Количество мест же можно отобразить с помощью размера точки.

```
#Допустим, что данные coffee1 и mosmap уже есть
#Переименуем (чтобы не терять данные) NA значения
coffee1@data[is.na(coffee1@data)] <- "Unknown"
shoko <- coffee1[coffee1@data$OperatingCompany == "Шоколадница", ]
starbucks <- coffee1[coffee1@data$OperatingCompany == "Старбакс", ]
#Создадим dataframes для каждого набора точек
shoko.coords <- as.data.frame(shoko@coords)
shoko.df <- as.data.frame(shoko@data)
shoko.df[, 6] <- shoko.coords[, 1]
shoko.df[, 7] <- shoko.coords[, 2]
names(shoko.df)[6] <- "coords.x"
names(shoko.df)[7] <- "coords.y"
starbucks.coords <- as.data.frame(starbucks@coords)
starbucks.df <- as.data.frame(starbucks@data)
```

```

starbucks.df[, 6] <- starbucks.coords[, 1]
starbucks.df[, 7] <- starbucks.coords[, 2]
names(starbucks.df)[6] <- "coords.x"
names(starbucks.df)[7] <- "coords.y"
mosmap.df <- fortify(mosmap, region = "NAME")
coffee.points <- ggplot() +
  geom_polygon(data = mosmap.df, fill = "white",
                aes(long, lat, group = group),
                color = "black") +
  geom_point(data = shoko.df, alpha = 0.6,
             aes(coords.x, coords.y, size = SeatsCount,
                 color = OperatingCompany)) +
  geom_point(data = starbucks.df, alpha = 0.6,
             aes(coords.x, coords.y, size = SeatsCount,
                 color = OperatingCompany)) +
  labs(x = "Долгота", y = "Широта",
       color = "Кофейня", size = "Количество мест") +
  scale_color_manual(values = c("green", "red"),
                      labels = c("Starbucks", "Шоколадница")) +
  ggttitle("Количество мест в кофейнях Москвы")
coffee.points + plot5theme

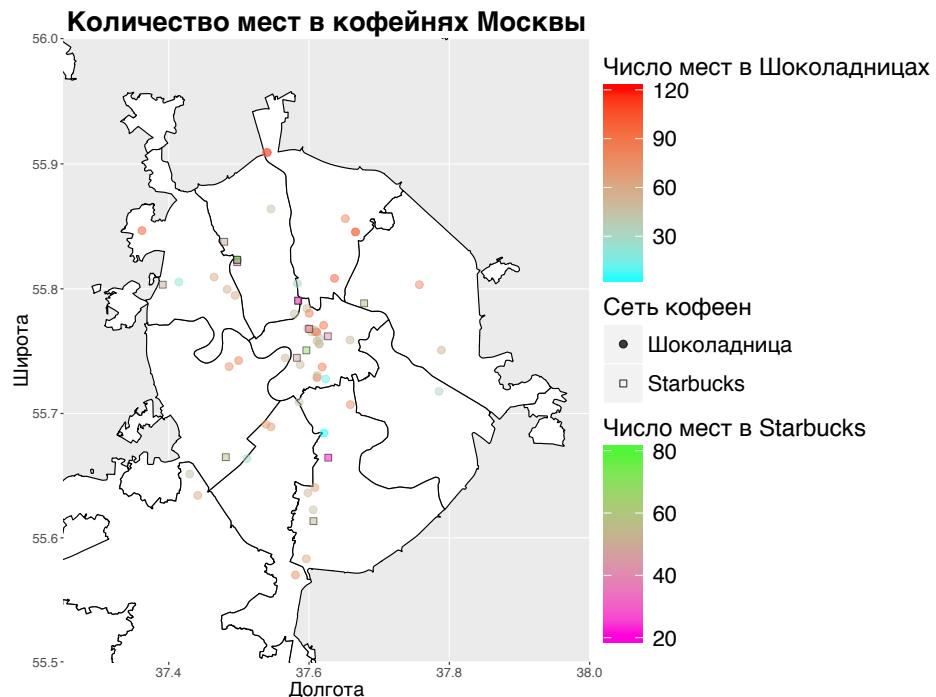
```



Также можно использовать градиент для этих целей. Как и в предыдущем случае, нужно разделить точки на две группы: первую, к точкам которой будет применяться

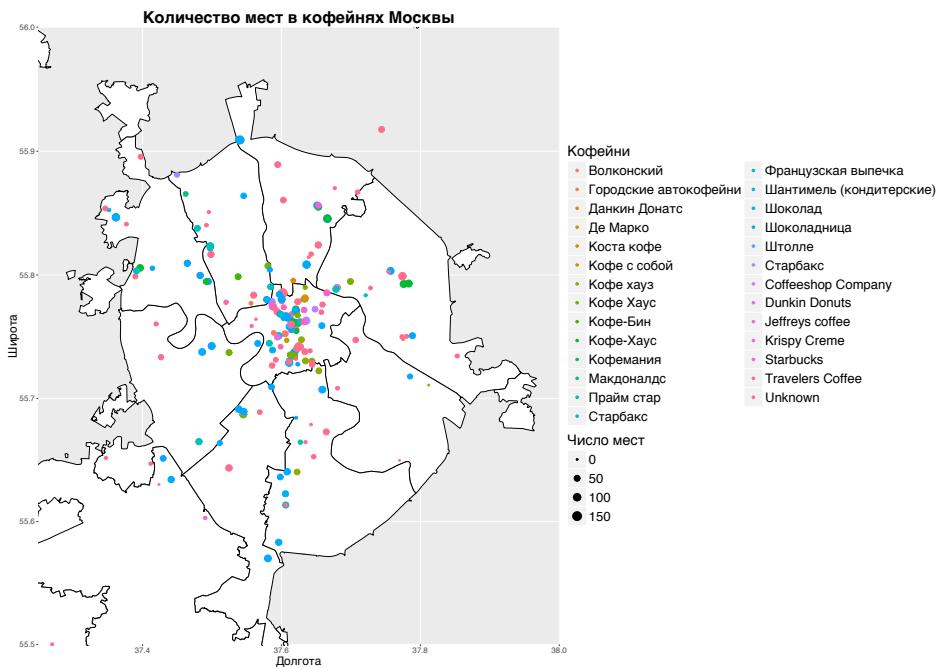
заливка, и вторую, точки которой будут иметь лишь внешний цветной контур (в случае небольшого размаха лучше выбрать контрастные цвета, также стоит сделать разные формы для точек в случае если они будут сероватого цвета). В таком случае наша команда выглядит следующим образом:

```
coffee.2grad <- ggplot() +
  geom_polygon(data = mosmap.df, fill = "white",
                aes(long, lat, group = group), color = "black") +
  coord_map(xlim = c(37.25, 38),
             ylim = c(55.50, 56)) +
  geom_point(data = shoko.df, size = 3, alpha = 0.5,
             aes(coords.x, coords.y, color = SeatsCount, shape = "19")) +
  geom_point(data = starbucks.df, size = 3, alpha = 0.5,
             aes(coords.x, coords.y, fill = SeatsCount, shape = "22")) +
  scale_color_gradient(low = "#00ffff", high = "#ff0400",
                        name = "Число мест в Шоколадницах") +
  scale_fill_gradient(low = "#ff00dc", high = "#00ff19",
                      name = "Число мест в Starbucks") +
  scale_shape_manual(values = c(19, 22),
                     labels = c("Шоколадница", "Starbucks"),
                     name = "Сеть кофеен") +
  labs(x = "Долгота", y = "Широта") +
  ggtitle("Количество мест в кофейнях Москвы")
coffee.2grad + plot5theme
```



Если же есть необходимость сравнить точки из более чем двух групп, мы можем использовать команды первого примера. Однако, полученную в результате картограмму сложно читать, и она будет намного полезнее, если сделать её интерактивной, например в `plot_ly`.

```
coffee2.coords <- as.data.frame(coffee1@coords)
coffee2.df <- as.data.frame(coffee1@data)
coffee2.df[, 6] <- coffee2.coords[, 1]
coffee2.df[, 7] <- coffee2.coords[, 2]
names(coffee2.df)[6] <- "coords.x"
names(coffee2.df)[7] <- "coords.y"
coffee.groups <- ggplot() +
  geom_polygon(data = mosmap.df, fill = "white",
                aes(long, lat, group = group), color = "black") +
  coord_map(xlim = c(37.25, 38),
             ylim = c(55.50, 56)) +
  geom_point(data = coffee2.df,
             aes(coords.x, coords.y, size = SeatsCount,
                 color = OperatingCompany)) +
  labs(x = "Долгота", y = "Широта",
       color = "Кофейни", size = "Число мест") +
  ggtitle("Количество мест в кофейнях Москвы")
coffee.groups + plot5theme
```



4.6 Отображение изменений на картах

Допустим, нам нужно сделать сравнение численности населения в Хабаровском крае за 2015 и 2016 года. Как вариант, мы можем построить две карты с помощью `gridExtra`. Но это занимает много места, и по цвету градиента не всегда ясно, изменилась ли численность или нет. В таком случае можно воспользоваться функцией `plot()`, добавив в неё цикл `for`:

```
khab3 <- readOGR("Карты/Хабаровский_край", "Хабаровск")
par(mar = c(0, 0, 1, 0))
plot(khab3, main = "Изменение численности населения в Хабаровском крае")
khab3$id <- 1:length(khab3@data$id)
for (i in 1:length(khab3@data$id)) {
  if (khab3@data$pop_2015[i] < khab3@data$pop_2016[i]) {
    plot(khab3[khab3$id == i, ], col = "green", add = TRUE)
  }
  else(plot(khab3[khab3$id == i, ], col = "red", add = TRUE))
}
legend("right", c("Уменьшилось", "Увеличилось"),
       bg = "transparent", box.lty = 0,
       fill = c("red", "green"),
       title = "Численность населения",
       cex = 1.5, title.adj = 0.22)
```

Изменение численности населения в Хабаровском крае



Теперь мы видим, что почти в всех муниципальных районах Хабаровского края по сравнению в 2015 году население упало. Но из картограммы нельзя узнать **на-**

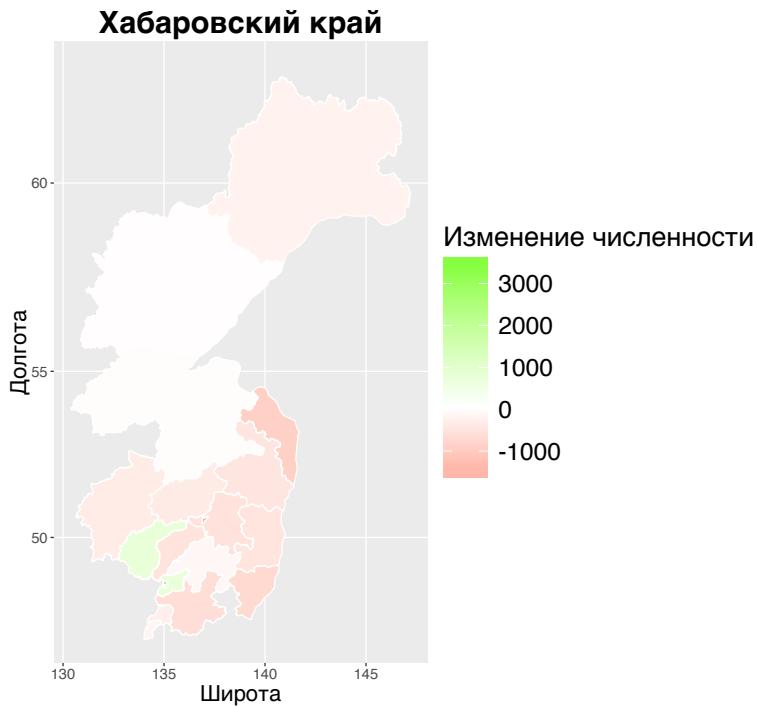
сколько оно упало. Для этой цели нам пригодится трёхцветная градиентная заливка `scale_gradient2()`, которую мы применим к объекту `ggplot2`:

```
khab3@data[, 6] <- khab3@data$pop_2016 - khab3@data$pop_2015
names(khab3@data)[6] <- "difference"

khabch.map <- fortify(khab3, region = "name")
khabch.map <- merge(khabch.map, khab3@data, by.x = "id", by.y = "name")

mapch <- ggplot() +
  geom_polygon(data = khabch.map, color = "white",
                aes(long, lat, group = group, fill = difference)) +
  coord_map() +
  labs(x = "Широта", y = "Долгота", fill = "Изменение численности") +
  scale_fill_gradient2(low = "#fe1529", mid = "white", high = "#6fff05") +
  ggtitle("Хабаровский край")

mapch + plot5theme
```



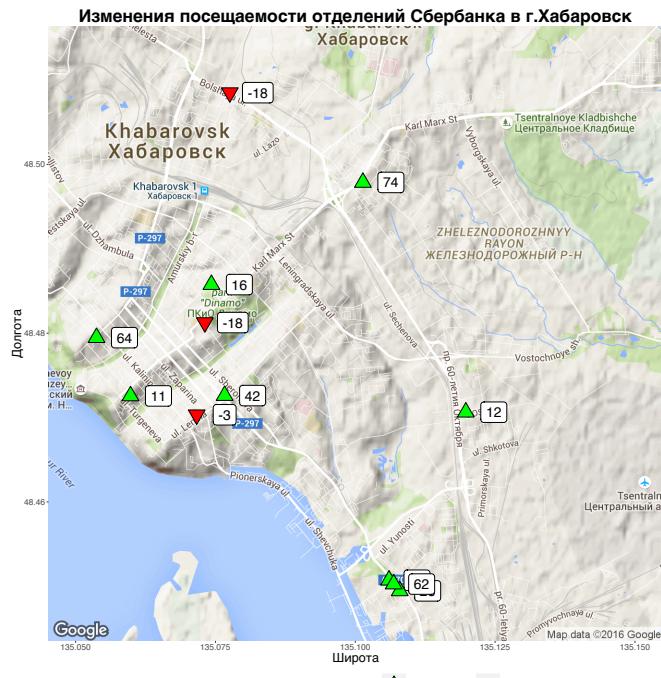
Такой подход делает картограмму более информативной: теперь нам очевидно, что большие изменения численности коснулись южных районов края.

Аналогично можно сделать и с точечными данными. Здесь нам пригодятся точки форм 24 и 25 (задаются с помощью `pch =` или `shape =`, представляют из себя стрелочки вверх и вниз). Разберём пример, создав случайные данные по посещаемости отделений Сбербанка (Сбербанк взят в качестве примера, поскольку насчитывает достаточно большое количество отделений):

```

#Зададим значения посещаемости банков до и после некоторого нововведения:
khab.banks.df[, 6] <- sample(120:200, length(khab.banks.df$name))
khab.banks.df[, 7] <- sample(130:220, length(khab.banks.df$name))
names(khab.banks.df)[6] <- "before"
names(khab.banks.df)[7] <- "after"
khab.banks.df[, 8] <- khab.banks.df$after - khab.banks.df$before
names(khab.banks.df)[8] <- "difference"
Sber <- khab.banks.df[khab.banks.df$name == "Сбербанк", ]
mapsouce_p <- get_map(location = c(lon = 135.1, lat = 48.48), zoom = 13)
pchange <- ggmap(mapsouce_p) +
  geom_point(data = Sber, size = 8,
             aes(coords.x, coords.y,
                 shape = factor(difference < 0),
                 fill = factor(difference < 0))) +
  geom_label(data = Sber, nudge_x = 0.005, size = 7,
             aes(coords.x, coords.y, label = difference)) +
  scale_shape_manual(name = "Изменение в посещаемости:",
                     values = c(24, 25),
                     labels = c("Возрасло", "Упало")) +
  scale_fill_manual(name = "Изменение в посещаемости:",
                     values = c("green", "red"),
                     labels = c("Возрасло", "Упало")) +
  labs(x = "Широта", y = "Долгота") +
  ggtitle("Изменения посещаемости отделений Сбербанка в г.Хабаровск")
pchange + plot5theme + theme(legend.position = "bottom")

```



4.7 Построение интерактивных карт

В этой подсекции мы разберёмся, как пользоваться пакетами **leaflet** и **plotly**, которые позволяют делать интерактивные карты. Сама интерактивность будет доступна в Rstudio, где можно будет увеличить/уменьшить масштаб карты, перетаскивать её, и посмотреть на место расположение точек (в нашем случае банков) вплоть до улицы, а также получить подробные данные, содержащиеся в этих точках.

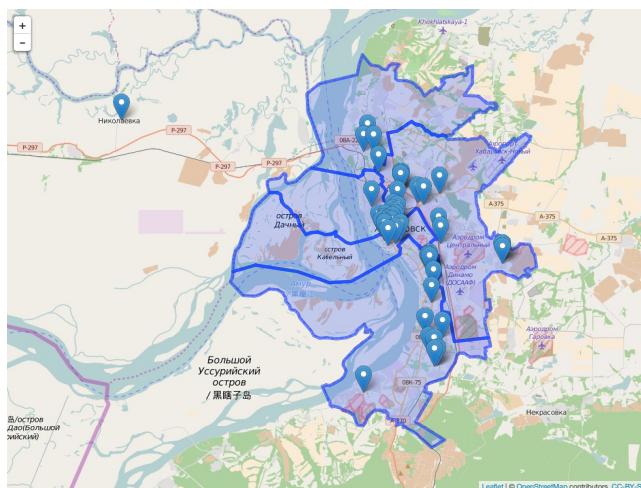
4.7.1 Создание карт с leaflet

Leaflet¹⁹ — библиотека javascript с открытым исходным кодом, предназначенная для создания интерактивных карт, которые можно поместить на веб-страницу. Поддерживает мобильные приложения. Также существует пакет leaflet для R. С ним мы и будем работать.

Чтобы в pdf документе появилось изображение карты (web-скриншот), понадобятся пакеты **webshot** и **htmlWidgets**. Работу первого мы настроили в секции «Подготовка» (заранее замечу, что для работы leaflet необходимо подключение к Интернету, и в случае падения скорости, карта может не прогрузиться полностью).

```
#Создание карты
khab.map.leaf <- leaflet() %>%
  addTiles() %>%
  #Добавим границы районов
  addPolygons(data = khab2) %>%
  #Добавим маркеры (точки) - координаты банков
  addMarkers(data = khab.banks)

#Сохранение карты
saveWidget(khab.map.leaf, "temp.html", selfcontained = FALSE)
webshot("temp.html", file = "plot13.pdf",
       cliprect = "viewport")
```

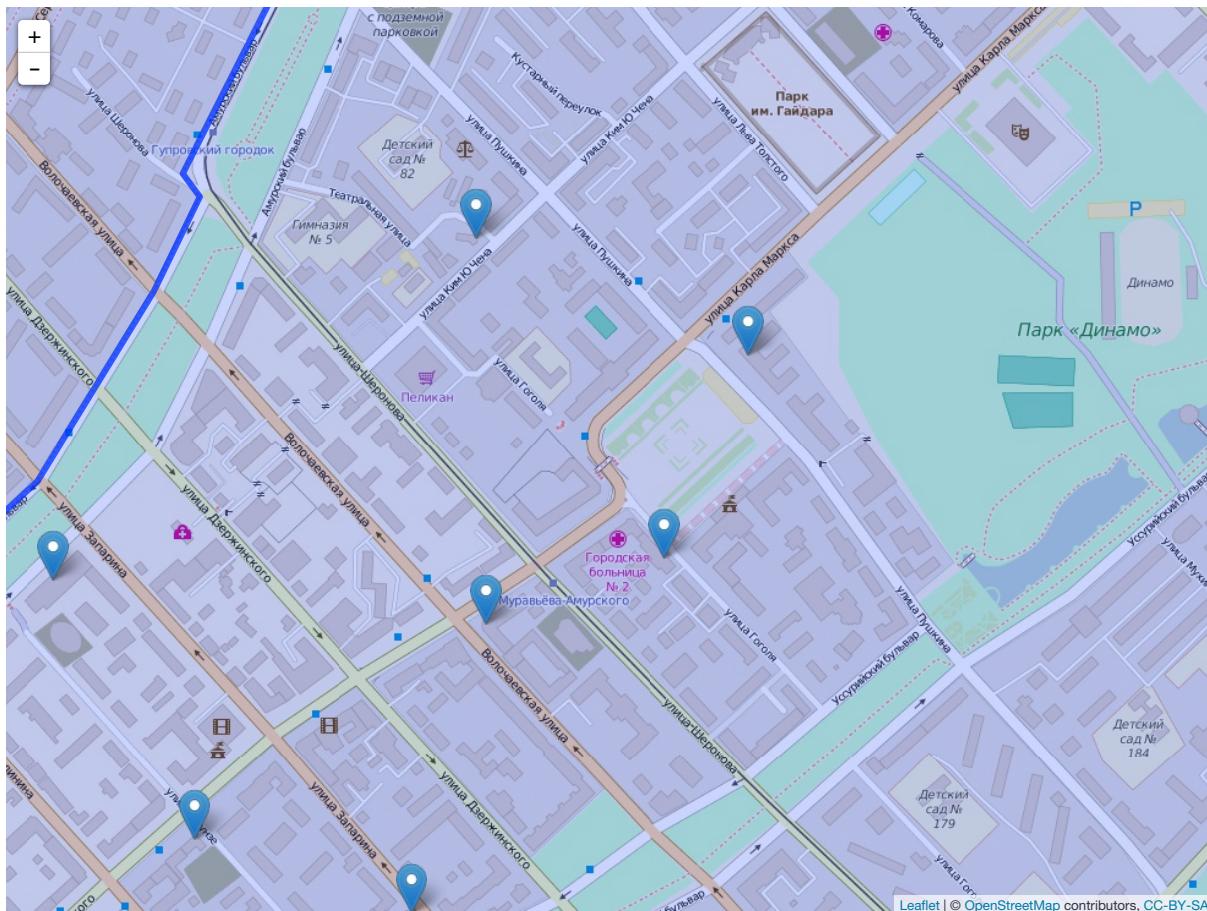


¹⁹Подробнее на <http://leafletjs.com>

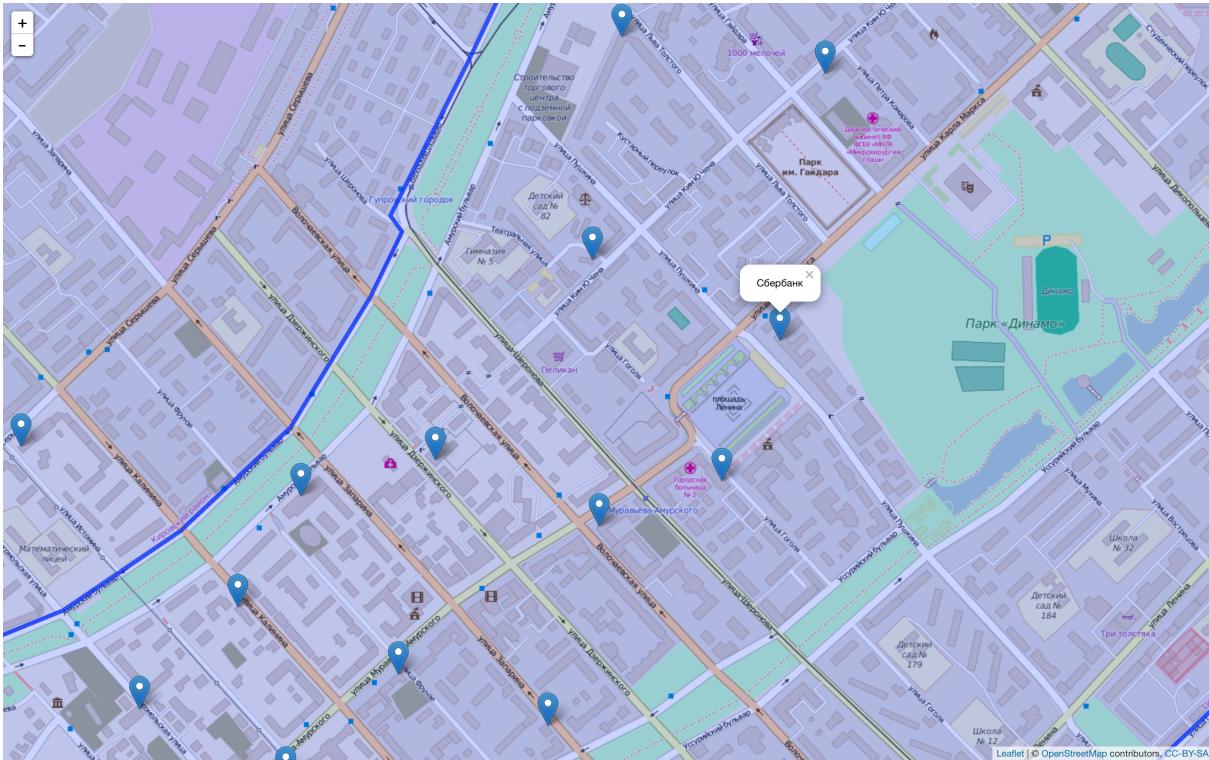
Таким образом, мы построили интерактивную карту, которая показывает границы районов Хабаровска и местоположения банков. Будет неплохо если мы её улучшим: к примеру, изменим масштаб и добавим всплывающие подсказки (popups). Для этого введём вот такую команду:

```
khab.map.leaf2 <- leaflet() %>%
  addTiles() %>%
  addPolygons(data = khab2) %>%
  addMarkers(data = khab.banks,
             popup = ~name) %>%
#lng и lat задают центр карты,
#zoom - чем больше значение, тем сильнее увеличение
  setView(lng = 135.07071, lat = 48.48002, zoom = 16)

saveWidget(khab.map.leaf2, "temp2.html", selfcontained = FALSE)
webshot("temp2.html", file = "plot14.pdf",
       cliprect = "viewport")
```



Теперь, если навести курсор на маркер, то мы получим всплывающее сообщение (в данном случае это название банка):



4.7.2 Создание карт с plotly

Plotly — очень мощный инструмент для интерактивной визуализации данных, способен работать не только с картами (чем мы занимаемся в рамках этой работы), но и с обычными графиками, гистограммами и диаграммами. Для работы требуется подключение к Интернету, так как создание инфографики происходит с помощью Вашей учётной записи plotly. Чтобы вывести полученный результат в pdf, нужно выполнить кое-какие настройки окружения:

```
Sys.setenv("plotly_username" = "ИМЯ ПОЛЬЗУВАТЕЛЯ")
#Ключ API можно найти на странице своего аккаунта на вкладке "API KEYS"
Sys.setenv("plotly_api_key" = "КЛЮЧ API")
```

Начнём с крайне полезной возможности plotly — конвертации объектов ggplot в объекты plotly. То есть, имея график или карту, созданную с помощью `ggplot()` (но не `ggmap()`), можно вывести её в интерактивном виде через `ggplotly()`. Для этого (при условии, что у Вас уже есть карта Хабаровска), введём следующую команду²⁰:

```
khab.map.plotly <- ggplotly(khab.map + plot5theme)
plotly_IMAGE(khab.map.plotly, width = 1200, height = 800,
             format = "png", out_file = "plot15.png")
```

²⁰Обратите внимание на ограничение API: <http://help.plot.ly/api-rate-limits/>

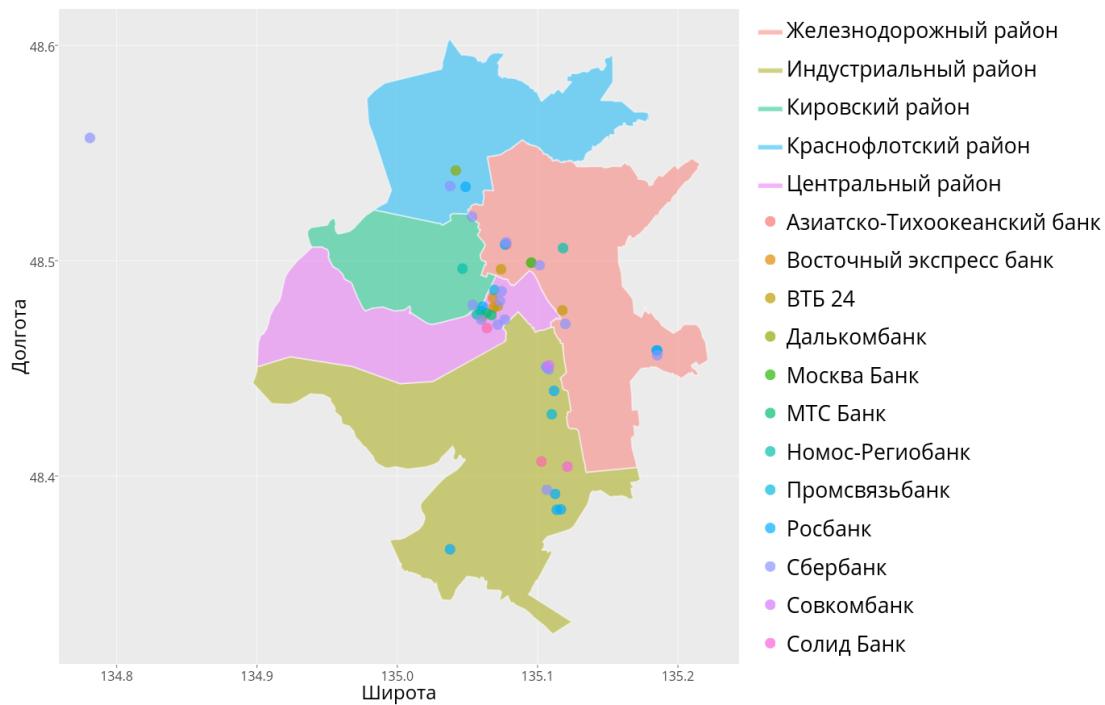
Хабаровск



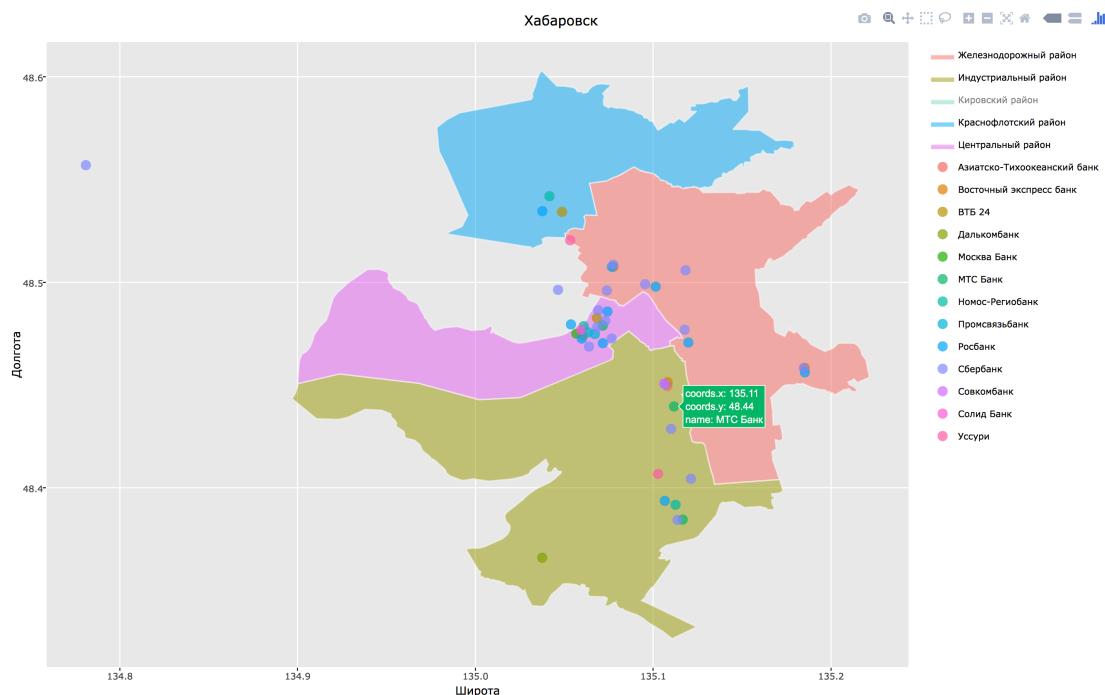
Карта на рисунке похожа на ту, что мы получили в помощью `ggplot()`, но с некоторыми упрощениями. Теперь, добавим в карту координаты банков. Для этого нам нужно будет немного видоизменить объект `khab.bank.map`. Замечу, что параметр `aes()`, в который задаёт `fill` (заливку) для полигонов и `color` (цвет) для точек участвует в создании легенды, поэтому, чтобы в легенде были корректно указаны разные объекты, не стоит создавать единый `aes()` в команде `ggplot()`. Назовём новый объект `khab.bank2`:

```
khab.bank2 <- ggplot(map.df) +  
  geom_polygon(color = "white", alpha = 0.5,  
               aes(long, lat, group = group, fill = id)) +  
  geom_point(data = khab.banks.df,  
             aes(coords.x, coords.y, color = name),  
             size = 3, alpha = 0.7) +  
  labs(x = "Широта", y = "Долгота",  
       color = NULL, fill = NULL) +  
  ggtitle("Хабаровск")  
  
khab.banks.plotly <- ggplotly(khab.bank2 + plot5theme)  
plotly_IMAGE(khab.banks.plotly, width = 1380, height = 920,  
            format = "png", out_file = "plot16.png")
```

Хабаровск



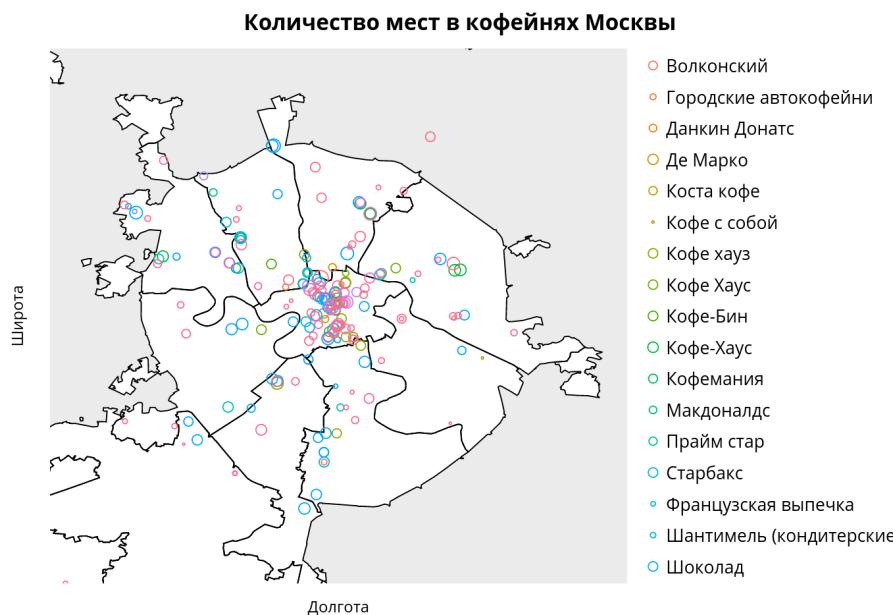
В карте можно убирать ненужные элементы, щёлкнув по ним на легенде, а при наведении курсора появится детальная информация, содержащаяся в объекте (например, наведя курсор на точку, мы получим её координаты и название банка, который эта точка отмечает.)



Как я уже отметил в одной из предыдущих подсекций, некоторые картограммы мо-

гут быть настолько загруженными, что их пригодность можно спасти интерактивностью, как например с картой кофеен в Москве (**coffee.groups**).

```
coffee.groups <- ggplot() +  
  geom_polygon(data = mosmap.df, fill = "white",  
               aes(long, lat, group = group), color = "black") +  
  coord_map(xlim = c(37.25, 38),  
            ylim = c(55.50, 56)) +  
  geom_point(data = coffee2.df, shape = 21,  
             aes(coords.x, coords.y, size = SeatsCount,  
                  color = OperatingCompany)) +  
  labs(x = "Долгота", y = "Широта",  
        color = NULL, size = NULL) +  
  ggttitle("Количество мест в кофейнях Москвы")  
  
coffee.groups.plotly <- ggplotly(coffee.groups + plot5theme)  
plotly_IMAGE(coffee.groups.plotly, width = 1380, height = 920,  
            format = "png", out_file = "plot17.png")
```



Так наша карта стала намного полезнее!

Согласно нашему опыту, мы можем вывести небольшое заключение: leaflet удобен в случаях, когда нужно просто отобразить данные с привязкой к карте, в то время как plotly создаёт схематичные карты, с привязкой данных непосредственно к объектам изучения - регионам, областям и так далее. Оба пакета просты в использовании, но совместимость ggplot2 с plotly несомненно, даёт последнему существенный перевес.

5 О пакете `rusmaps`

К данной работе прилагается пакет `rusmaps`, разработанный мною при поддержке моего научного руководителя, Бориса Демешева. Этот пакет предназначен для работы в R и содержит в себе готовый набор shape-файлов, среди которых есть:

- Города-миллионеры (согласно данным Википедии):

Москва (2015, 2016)

Санкт-Петербург (2015, 2016)

Новосибирск (2016)

Екатеринбург (2015)

Нижний Новгород (2015)

Казань (2015)

Челябинск (2015)

Омск (2010)

Самара (2015)

Ростов-на-Дону

Уфа (2015)

Красноярск (2010)

Пермь (2015)

Воронеж (2010)

Волгоград (2015)

- Федеральные округа (с субъектами Федерации, данные по оценке численности населения даны за 2015 и 2016)

Центральный федеральный округ

Северо-Западный федеральный округ

Южный федеральный округ

Северо-Кавказский федеральный округ

Приволжский федеральный округ

Уральский федеральный округ

Сибирский федеральный округ

Дальневосточный федеральный округ

Крымский федеральный округ

- Российская Федерация в целом (доступны версии с Крымом и без)

С федеральными округами

С субъектами Федерации

- Хабаровский край (2016)

Содержащиеся в данных городов полигоны были получены с помощью QGIS и модулей OSM Place Search и OpenLayers. Это же касается и данных по Хабаровскому краю. Подробную инструкцию Вы можете прочитать в подсекции «Получение административных границ регионов». В файлах переименованы некоторые районы, для соответствия официальным названиям. Также были добавлены данные по оценке численности населения на 1 января 2010, 2015 и 2016 годов (указано в скобках), взятые с сайта Росстата: www.gks.ru и его региональных филиалах. Данные федеральных округов и субъектов Федерации были получены с ресурса: <http://gis-lab.info/qa/osm-adm.html> (файлы с обрезкой по береговой линии). Таблицы данных изменены и дополнены аналогично таблицам файлов городов.

Репозиторий пакета: <https://github.com/akondrashov96/rusmaps>.

Для загрузки пакета понадобится следующая команда:

```
devtools::install_github("akondrashov96/rusmaps")
library("rusmaps")
```

Теперь, подключив пакет rusmaps, можно использовать его данные для создания картограмм. Замечу, что большая часть текста работы была написана до создания пакета, поэтому в своей работе я обращался к shape-исходникам. Список карт (и их "программных позывных") можете увидеть, введя следующее (колонка format была скрыта для компактности):

```
library("rusmaps")
rusmaps.dataframe[, c(1, 2, 4)]
```

##	map_name	description	date
## 1	cities	Cities of Russia with 1mil population	27.06.2016
## 2	Voronezh	Voronezh city districts	27.06.2016
## 3	Novgorod	Novgorod city districts	27.06.2016
## 4	Novosibirsk	Novosibirsk city districts	27.06.2016
## 5	Omsk	Omsk city districts	27.06.2016
## 6	Perm	Perm city districts	27.06.2016
## 7	Rostov	Rostov-na-Donu city districts	27.06.2016

## 8	Samara	Samara city districts	27.06.2016
## 9	StPetersburg	Saint-Petersburg city districts	27.06.2016
## 10	Ufa	Ufa city districts	27.06.2016
## 11	Chelyabinsk	Chelyabinsk city districts	27.06.2016
## 12	Moscow	Moscow city districts	27.06.2016
## 13	Krasnoyarsk	Krasnoyarsk city districts	27.06.2016
## 14	Kazan	Kazan city districts	27.06.2016
## 15	Ekaterinburg	Ekaterinburg city districts	27.06.2016
## 16	Volgograd	Volgograd city districts	27.06.2016
## 17	rus_fd_nc	Federal Districts of Russia (without Crimea)	27.06.2016
## 18	rus_fd	Federal Districts of Russia	27.06.2016
## 19	rus_sub	Federal subjects of Russia	27.06.2016
## 20	rus_sub_nc	Federal subjects of Russia (without Crimea)	27.06.2016
## 21	DFO	Regions in Far Eastern Federal District	27.06.2016
## 22	KFO	Regions in Crimea Federal District	27.06.2016
## 23	VFO	Regions in Volga Federal District	27.06.2016
## 24	NWFO	Regions in North-Waestern Federal District	27.06.2016
## 25	NCFO	Regions in North-Caucasus Federal District	27.06.2016
## 26	SFO	Regions in Siberia Federal District	27.06.2016
## 27	UFO	Regions in Ural Federal District	27.06.2016
## 28	CentralFO	Regions in Central Federal District	27.06.2016
## 29	SouthFO	Regions in Soutern Federal District	27.06.2016
## 30	Khabarovsk	Municipal districts of Khabarovsk region	27.06.2016

6 Заключение

В данной работе я рассмотрел и привёл примеры получения и визуализации данных на картах. Также, в рамках этой работы я разработал пакет **rusmaps**, в который включил границы территории городов и регионов России. Возможности визуализации пространственных данных в R на настоящий момент достаточно развиты и способны решить широкий спектр задач. Тем не менее, это не мешает создавать новые инструменты, в том числе и пакеты данных, для упрощения работы и дальнейшего расширения среды R в этой области. Кроме того, разумным решением станет использовать QGIS в паре с R: во-первых, QGIS это тоже свободно распространяемое ПО, функционал которого можно расширять посредством различных модулей, а во-вторых, это экономит время: в QGIS многие задачи решаются просто и быстро несколькими щелчками мышью, тогда как в R может возникнуть необходимость разбираться в новых командах, что может занять много времени. Выражаю надежду, что читатели данной работы найдут её полезной и смогут с лёгкостью использовать R для своих исследовательских нужд.

7 Приложение 1

Административные уровни регионов России

Каждый регион обладает определённым административным уровнем, который определяет тип этого региона согласно применяемому в той или иной стране территориальному делению. Данные для России приведены в таблице ниже:

Уровень	Описание	Пример
1	N/A	
2	Государства	Российская Федерация
3	Федеральные округа	Дальневосточный федеральный округ
4	Субъекты Федерации	Хабаровский край
5	Административные округа городов федерального значения	Троицкий административный округ (Москва)
6	Муниципальные районы субъектов федерации	г. Хабаровск, Амурский район
7	N/A (для России)	
8	Внутригородские муниципальные территории городов федерального значения	Район Измайлово (Москва)
9	Административные районы городских округов и поселений	Индустримальный район (Хабаровск)
10	Микрорайоны городов ²¹	

8 Приложение 2

Полезные ресурсы

В данной секции я собрал ссылки на ресурсы и проекты, относящиеся к визуализации данных на картах, которые не понадобились мне при подготовке данной работы, но могут быть полезны и интересны в качестве дополнительного материала.

- OSM Boundaries 4.0

В данном проекте представлена большая часть (если не все) регионов мира по их административному уровню. Если возникает неуверенность, какой уровень соответствует тому или иному региону, то я советую им воспользоваться.

Ссылка: <https://osm.wno-edv-service.de/boundaries/>

- Конвертация пользовательской карты Яндекс в shape-файл

По указанной ссылке Вы найдёте инструкцию по конвертации карты Яндекс, содержащей точечные данные в ESRI shapefile.

Ссылка: <http://gis-lab.info/qa/yandex-shape.html>

²¹Обсуждается. Подробную информацию можно узнать по ссылке: <http://wiki.openstreetmap.org/wiki/Tag:boundary%3Dadministrative>.

Список литературы

- [1] Robin Lovelace, James Cheshire, Rachel Oldroyd and others «Introduction to visualizing spatial data in R» <http://eprints.ncrm.ac.uk/3295/> (на момент 13.06.2016)
- [2] GIS-Lab («ГИС Лаборатория») <http://gis-lab.info>
- [3] «Поиск и загрузка данных OpenStreetMap» http://www.qgistutorials.com/ru/docs/downloading_osm_data.html
- [4] Мастицкий С.Э., Шитиков В.К. (2014) Статистический анализ и визуализация данных с помощью R. – Электронная книга, адрес доступа: <http://r-analytics.blogspot.com>
- [5] D. Kahle and H. Wickham. ggmap: Spatial Visualization with ggplot2. The R Journal, 5(1), 144-161. URL <http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf>
- [6] Roger S. Bivand, Edzer J. Pebesma, Virgilio Gomez-Rubio. «Applied Spatial Data Analysis with R»
- [7] StackOverFlow.com (здесь можно найти ответы на многие вопросы и решения для возникающих трудностей)
- [8] Прочие ссылки, которые Вы найдете в работе.