Visualização de Dados Espaciais

Prof. Dr. Anderson C. Carniel - Universidade Federal de São Carlos (UFSCar)

1 Introdução

A visualização de dados espaciais é uma etapa bastante essencial na análise de resultados de consultas espaciais emitidas sobre bancos de dados espaciais. Ainda, essa etapa permite a criação de mapas, que possibilitam a combinação da análise convencional dos dados aliado a análise espacial.

Esta apostila objetiva discutir a visualização de objetos espaciais através da utilização da linguagem de programação R. Este método permite que os objetos espaciais sejam usados em *análises exploratória de dados (espaciais)* (e.g., usando o superpacote `tidyverse`). Entretanto, é importante enfatizar que outras linguagens de programação poderiam ser aplicadas para este fim desde que sejam respeitadas suas respectivas sintaxes e utilização apropriada de seus pacotes e bibliotecas. Ademais, outra forma bastante comum de se produzir mapas a partir de visualizações de objetos espaciais é através da utilização de *Sistemas de Informação Geográfica* (SIG) como o QGIS (https://qgis.org/en/site/).

Para atingir o objetivo da apostila, o pacote `sf` é usado juntamente com o superpacote `tidyverse` e o pacote `RPostgres` . A instalação desses pacotes podem ser feita como qualquer outro pacote do R, seja pela interface do RStudio ou via comandos como:

```
install.packages("tidyverse")
install.packages("RPostgres")
install.packages("sf") #principal pacote para manipular dados espaciais
```

Mais detalhes sobre a instalação do `sf` pode ser obtida em sua documentação oficial aqui (https://r-spatial.github.io/sf/index.html#installing).

Ainda, esta apostila utiliza a base de dados criada na disciplina e mantida no PostgreSQL/PostGIS.

Após a instalação desses pacotes, é necessário carregá-los na memória ao executar os *scripts* contidos nesta apostila:

```
#lembre-se, é necessário carregar os pacotes que serão usados na aplicação antes de tudo e antes de executar os demais comandos
library("tidyverse")
```

```
## ── Attaching packages ─────────────────────────────────────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.0.3     ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ─────────────────────────────────────────────────────────────────
──────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library("RPostgres")
library("sf")
```

```
## Linking to GEOS 3.8.0, GDAL 3.0.4, PROJ 6.3.1
```

2 Executando consultas espaciais e armazenando seus resultados no R

Nesta seção, iremos compreender como estabelecer uma conexão do R com o PostgreSQL/PostGIS e então recuperar dados de um banco de dados espacial. Com isso, é possível recuperar objetos espaciais oriundos de consultas espaciais específicas.

Primeiro, é necessário estabelecer uma conexão ao banco de dados espacial de interesse (nesse caso chamado `bd_espacial`), possibilitando a execução de consultas SQL. Dessa forma, a estrutura da nossa aplicação terá o seguinte formato:

```
#aqui estamos definindo nossa conexão e atribuindo essa conexão à variável con
con <- dbConnect(Postgres(),
                 user = "postgres", #nome do usuário
                 password = "123", #senha
                 host = "localhost",
                 port = 5432,
                 dbname = "bd_espacial")
#visualizando alguns dados sobre a conexão estabelecida
con
## <PqConnection> bd_espacial@localhost:5432

#depois de estabelecida a conexão, podemos executar vários comandos SQL sobre a nossa base de dados espacial!
#o objetivo é "rechear" essa parte com consultas espaciais, armazenando os seus resultados em dados tabulares do R

#assim que terminarmos de executar todos os comandos SQL que nos interessam, devemos fechar a conexão:
dbDisconnect(con)
```

Agora que sabemos a estrutura geral de recuperação de dados espaciais, vamos de fato recuperá-los! Para isso, vamos usar a função `st_read()` do pacote `sf`. É importante enfatizar que essa função aceita mais de 200 formatos de bases de dados espacial que podem ser importadas ao R (https://r-spatial.github.io/sf/articles/sf2.html)! Dessa forma, é uma função **muito flexível**, aceitando diversos formatos (e.g., *shapefiles*, representação textuais de objetos espaciais, etc.). Assim, é recomendado uma leitura de sua documentação que contém vários exemplos (https://r-spatial.github.io/sf/reference/st_read.html).

No nosso caso, queremos recuperar dados espaciais de uma base mantida no PostgreSQL/PostGIS. Dessa forma, vamos usar o `st_read()` com 2 parâmetros:

1. o primeiro parâmetro é a conexão com o banco de dados espacial
2. o segundo parâmetro, com nome `query`, informa a consulta espacial que deve ser processada

A seguir, é mostrado um exemplo prático:

```
#estabelecendo a conexão com o bd_espacial
con <-dbConnect(Postgres(),
                user = "postgres",
                password = "123",
                host = "localhost",
                port = 5432,
                dbname = "bd_espacial")

#executando a seguinte consulta:
# Recupere todas as cidades do estado de SP
cidades_SP <- st_read(con,
                      query = "SELECT * FROM br_municipios_2019 WHERE sigla_uf = 'SP'")
#aqui, terminamos de executar comandos SQL, então, vamos terminar a conexão com o banco
dbDisconnect(con)
```

Afinal, como a variável `cidades_SP` do código acima é representada, armazenada e manipulada pelo R (ou mais precisamente, pelo pacote `sf`)?

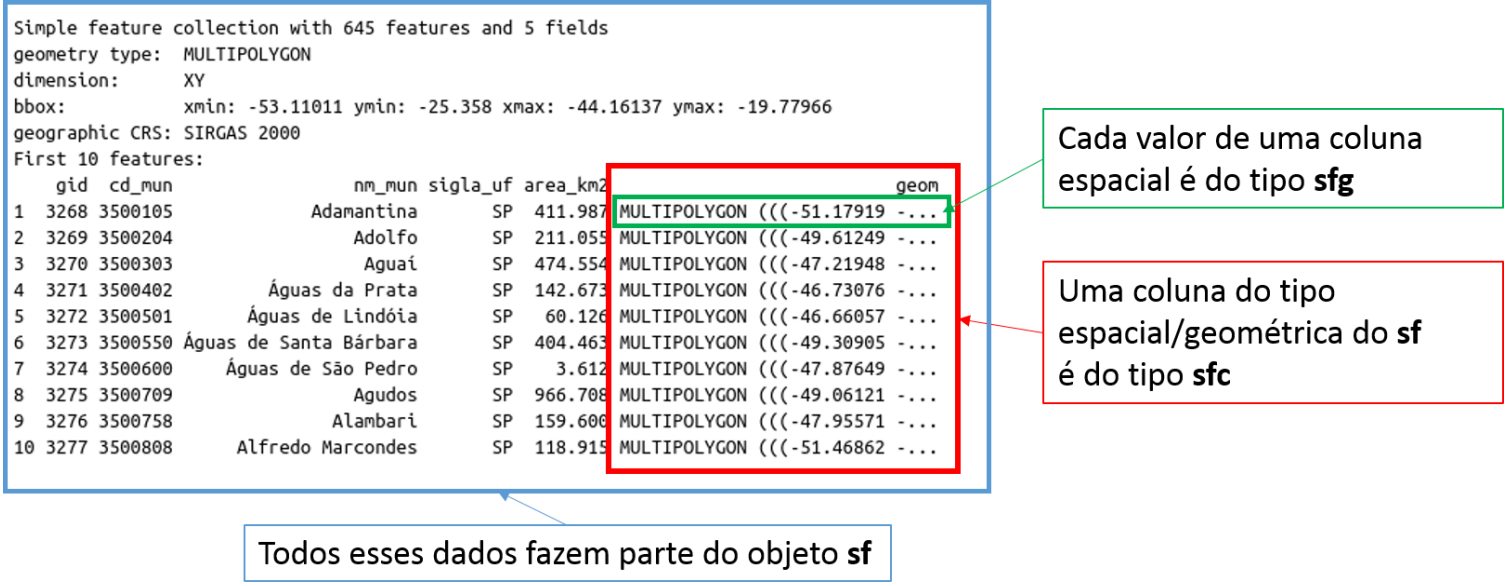
Vamos investigar o tipo de dado dessa variável:

```
#vamos visualisar o conteúdo dessa variável
print(cidades_SP)
```

```
## Simple feature collection with 645 features and 5 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -53.11011 ymin: -25.358 xmax: -44.16137 ymax: -19.77966
## geographic CRS: SIRGAS 2000
## First 10 features:
##     gid  cd_mun                 nm_mun sigla_uf area_km2
## 1  3268 3500105             Adamantina       SP  411.987
## 2  3269 3500204                 Adolfo       SP  211.055
## 3  3270 3500303                  Aguaí       SP  474.554
## 4  3271 3500402         Águas da Prata       SP  142.673
## 5  3272 3500501       Águas de Lindóia       SP   60.126
## 6  3273 3500550 Águas de Santa Bárbara       SP  404.463
## 7  3274 3500600     Águas de São Pedro       SP    3.612
## 8  3275 3500709                 Agudos       SP  966.708
## 9  3276 3500758               Alambari       SP  159.600
## 10 3277 3500808      Alfredo Marcondes       SP  118.915
##                               geom
## 1  MULTIPOLYGON (((-51.17919 -...
## 2  MULTIPOLYGON (((-49.61249 -...
## 3  MULTIPOLYGON (((-47.21948 -...
## 4  MULTIPOLYGON (((-46.73076 -...
## 5  MULTIPOLYGON (((-46.66057 -...
## 6  MULTIPOLYGON (((-49.30905 -...
## 7  MULTIPOLYGON (((-47.87649 -...
## 8  MULTIPOLYGON (((-49.06121 -...
## 9  MULTIPOLYGON (((-47.95571 -...
## 10 MULTIPOLYGON (((-51.46862 -...
```

Perceba que `cidades_SP` é um *simple feature collection* (mesmo nome dado para a documentação da OGC sobre a especificação de dados espaciais!), ou seja, um tipo de dado empregado pelo pacote para representar dados espaciais com informações tabulares associados a eles. Nesse caso, essa coleção contém 645 linhas e 5 colunas que descrevem o tipo geométrico armazenado (coluna `geom`), que nesse caso é do tipo MULTIPOLYGON de 2 dimensões (XY). Note ainda que o SRID é adequadamente preservado pelo `sf` , onde nesse caso usa-se o nome como **CRS**.

O tipo de dado `sf` na verdade é um "supertipo" de dado que tem outros tipos de dados atrelados a ele, chamados `sfc e sfg` . Vamos entendê-los melhor conforme a imagem a seguir:



Tela Inicial do RStudio

Portanto, a partir da imagem acima, podemos observar a existência da seguinte hierarquia de tipos de dados do pacote `sf`:

1. um **objeto sf** é uma tabela que contém ao menos 1 coluna com objetos espaciais!
2. um **objeto sfc** é uma lista de objetos espaciais. Logo, **toda coluna espacial/geométrica de um sf** é do tipo `sfc`.
3. um **objeto sfg** é um único objeto espacial. Logo, **cada elemento de um sfc é um sfg**.

Com o pacote `sf` é possível realizar operações espaciais assim como realizamos no PostGIS. Note que diversos comandos se assemelham bastante com o nome usado pelo PostGIS, devido ao padrão da OGC. Por exemplo, o `sf` possui uma função chamada `st_union()` (https://r-spatial.github.io/sf/reference/geos_combine.html) que computa a união geométrica entre objetos espaciais.

Quando o objeto `sf` possui 2 colunas com dados espaciais. Note que somente uma delas "está ativa". Isso significa que será esta coluna a ser usada pelas operações sobre um objeto `sf`. Veja um exemplo abaixo:

```
#estabelecendo a conexão com o bd_espacial
con <-dbConnect(Postgres(),
                user = "postgres",
                password = "123",
                host = "localhost",
                port = 5432,
                dbname = "bd_espacial")

#executando a seguinte consulta:
# Recupere 100 construções e suas respectivas cidades do estado do PR.
#esta consulta é do tipo junção espacial
construcoes_PR <- st_read(con,
                      query = "SELECT * FROM br_municipios_2019, buildings
                     WHERE sigla_uf = 'PR' AND st_intersects(st_transform(geom, 3857), way) LIMIT
 100")

#aqui, terminamos de executar comandos SQL, então, vamos terminar a conexão com o banco
dbDisconnect(con)
```

A variável `construcoes_PR` possui duas colunas geométricas, sendo a coluna `geom` a coluna geométrica ativa no momento. Dessa forma, todas as operações que forem feitas sobre esse `sf` serão realizadas utilizando essa coluna. Para realizar uma operação na coluna `way`, ou seja, a outra coluna que contém objetos espaciais, duas opções são possíveis:

1. Deixar a coluna `way` como sendo a principal pelo comando:

```
#o parâmetro sf_column_name determina quem é a coluna geométrica ativa de um objeto sf
construcoes_PR2 <- st_sf(construcoes_PR, sf_column_name = "way")
#assim, ao usar a variável construcoes_PR2, podemos fazer operações sobre a coluna way
#vamos checar as informações de construcoes_PR2
print(construcoes_PR2)
```

```
## Simple feature collection with 100 features and 7 fields
## Active geometry column: way
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -6013287 ymin: -3011554 xmax: -5421617 ymax: -2745331
## projected CRS:  WGS 84 / Pseudo-Mercator
## First 10 features:
##     gid  cd_mun              nm_mun sigla_uf area_km2    osm_id building
## 1  3914 4100202        Adrianópolis       PR 1349.311 891349911      yes
## 2  3914 4100202        Adrianópolis       PR 1349.311 891349913      yes
## 3  3915 4100301       Agudos do Sul       PR  192.261 374978886   church
## 4  3915 4100301       Agudos do Sul       PR  192.261 293800699      yes
## 5  3915 4100301       Agudos do Sul       PR  192.261 373466249   church
## 6  3916 4100400 Almirante Tamandaré       PR  194.888 739568122      yes
## 7  3916 4100400 Almirante Tamandaré       PR  194.888 739568124      yes
## 8  3916 4100400 Almirante Tamandaré       PR  194.888 739568117      yes
## 9  3916 4100400 Almirante Tamandaré       PR  194.888 739568116      yes
## 10 3916 4100400 Almirante Tamandaré       PR  194.888 693775235      yes
##                               geom                                way
## 1  MULTIPOLYGON (((-49.00349 -... POLYGON ((-5423247 -2858092...
## 2  MULTIPOLYGON (((-49.00349 -... POLYGON ((-5421635 -2858023...
## 3  MULTIPOLYGON (((-49.35543 -... POLYGON ((-5489091 -3011488...
## 4  MULTIPOLYGON (((-49.35543 -... POLYGON ((-5492938 -2998103...
## 5  MULTIPOLYGON (((-49.35543 -... POLYGON ((-5491673 -2997732...
## 6  MULTIPOLYGON (((-49.34403 -... POLYGON ((-5489827 -2922247...
## 7  MULTIPOLYGON (((-49.34403 -... POLYGON ((-5489954 -2922119...
## 8  MULTIPOLYGON (((-49.34403 -... POLYGON ((-5491069 -2920874...
## 9  MULTIPOLYGON (((-49.34403 -... POLYGON ((-5491045 -2920894...
## 10 MULTIPOLYGON (((-49.34403 -... POLYGON ((-5492927 -2918100...
```

2. Capturar a coluna `way` como sendo um `sfc` pelo comando:

```
#capturando uma coluna espacial/geométrica de um sf é igual a capturar uma coluna de data frame do
 R
coluna_way <- construcoes_PR$way
#vamos verificar as informações dessa variável
print(coluna_way)
## Geometry set for 100 features
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -6013287 ymin: -3011554 xmax: -5421617 ymax: -2745331
## projected CRS:  WGS 84 / Pseudo-Mercator
## First 5 geometries:
## POLYGON ((-5423247 -2858092, -5423227 -2858096,...
## POLYGON ((-5421635 -2858023, -5421633 -2858029,...
## POLYGON ((-5489091 -3011488, -5489086 -3011549,...
## POLYGON ((-5492938 -2998103, -5492918 -2998116,...
## POLYGON ((-5491673 -2997732, -5491662 -2997747,...
```

Se você quiser somente manipular objetos espaciais, talvez a opção 2 seja mais adequada. Entretanto, se você manipular a tabela de resultados como um todo, talvez seja mais interessante a opção 1.

Com a opção 2, podemos ainda capturar somente um único objeto espacial do objeto `sfc` .

```
#a coluna_way foi definida no trecho de código anterior

#se certificando do tipo de dado da variável coluna_way
class(coluna_way)
## [1] "sfc_POLYGON" "sfc"

#capturando o primeiro objeto espacial de coluna_way
coluna_way[[1]]
## POLYGON ((-5423247 -2858092, -5423227 -2858096, -5423225 -2858082, -5423229 -2858081, -5423244 -
2858078, -5423247 -2858092))

#se certificando do tipo de dado de um objeto espacial da coluna_way
class(coluna_way[[1]])
## [1] "XY"      "POLYGON" "sfg"
```

Nota importante sobre manipulação

É possível manipular dados tabulares com objetos espaciais (ou seja, variáveis do tipo `sf`) usando métodos do pacote `tidyverse` (mais precisamente do `dplyr`). Assim, manipula-se tais dados de uma forma muito similar ao como é feito com dados convencionais, porém agora com o poder e inclusão de funções que nativamente manipulam objetos espaciais (e.g., checagem se um objeto contém o outro, ou computar a união geométrica entre objetos espaciais).

Note que esse tipo de processamento e manipulação de dados foi vista na disciplina utilizando consultas SQL no PostGIS. Para aqueles que desejam saber mais como efetuar esse tipo de processamento e manipulação no R, recomenda-se a leitura:

Tidyverse methods for sf objects (https://r-spatial.github.io/sf/reference/tidyverse.html)

Note que, ao usar um objeto `sf` em um método do `dplyr`, como o `select()`, o significado é ligeiramente adptado devido a inclusão de dados espaciais. No caso do `select()`, a coluna contendo dados espaciais **sempre** será mantida, mesmo ela não estando inclusa no `select()`.

Para manter o comportamento original das funções do `dplyr`, é necessário a conversão de um objeto `sf` para `data.frame`, usando a função `as.data.frame()` e então usar o resultado dessa conversão como entrada para as funções do `dplyr`.

3 Visualizando objetos espaciais

Na seção anterior, vimos como carregar objetos espaciais no R a partir de resultados de consultas SQL executados no PostgreSQL/PostGIS. Nesta seção, veremos como visualizar de forma gráfica tais resultados, propiciando a formulação de **mapas**.

Para executar o restante dos comandos da apostila, é necessário a execução do seguinte código R abaixo, o qual estabelece a conexão com a base de dados usada na disciplina. Isso possibilita o processamento de consultas espaciais já vistas ao longo da disciplina.

```
#estabelecendo a conexão com o bd_espacial
con <-dbConnect(Postgres(),
                user = "postgres",
                password = "123",
                host = "localhost",
                port = 5432,
                dbname = "bd_espacial")
```

3.1 Primeiros passos

Vamos gerar nossa primeira visualização gráfica ao seguir os passos abaixo:

1. Busque os dados de interesse na base de dados espacial mantida no PostgreSQL/PostGIS. Para isso, execute uma consulta espacial dentro da função `st_read()` já apresentada e armazene o resultado em uma variável (lembre-se, este resultado será do tipo `sf`).
2. Use a função `geom_sf()` do `ggplot2` (https://ggplot2.tidyverse.org/reference/ggsf.html) para mostrar na tela os objetos espaciais de interesse. Essa função é "somada" a um objeto `ggplot` e possibilita a especificação de uma estética sobre objetos

espaciais (será discutido mais a seguir). De forma mínima, a função `geom_sf()` precisa receber, via parâmetro `data`, os dados a serem utilizados na visualização (por exemplo, os dados tabulares retornados na etapa 1). Opcionalmente, via parâmetro `geometry` da função `aes()` do parâmetro `mapping`, é possível especificar qual é a coluna geométrica que será mostrada no gráfico (caso ele não for informado, a geometria ativa do `sf` será a usada). Este parâmetro será necessário caso seu conjunto de dados possuir mais que uma coluna geométrica, por exemplo.

3. Estilize seu gráfico conforme necessário! Por exemplo, além da função `geom_sf()`, o `ggplot2` possibilita adicionar rótulos aos objetos via função `geom_sf_label()`. É possível ainda adicionar outras camadas de objetos espaciais. Veremos sobre alguns tipos de estilos ao longo da apostila.

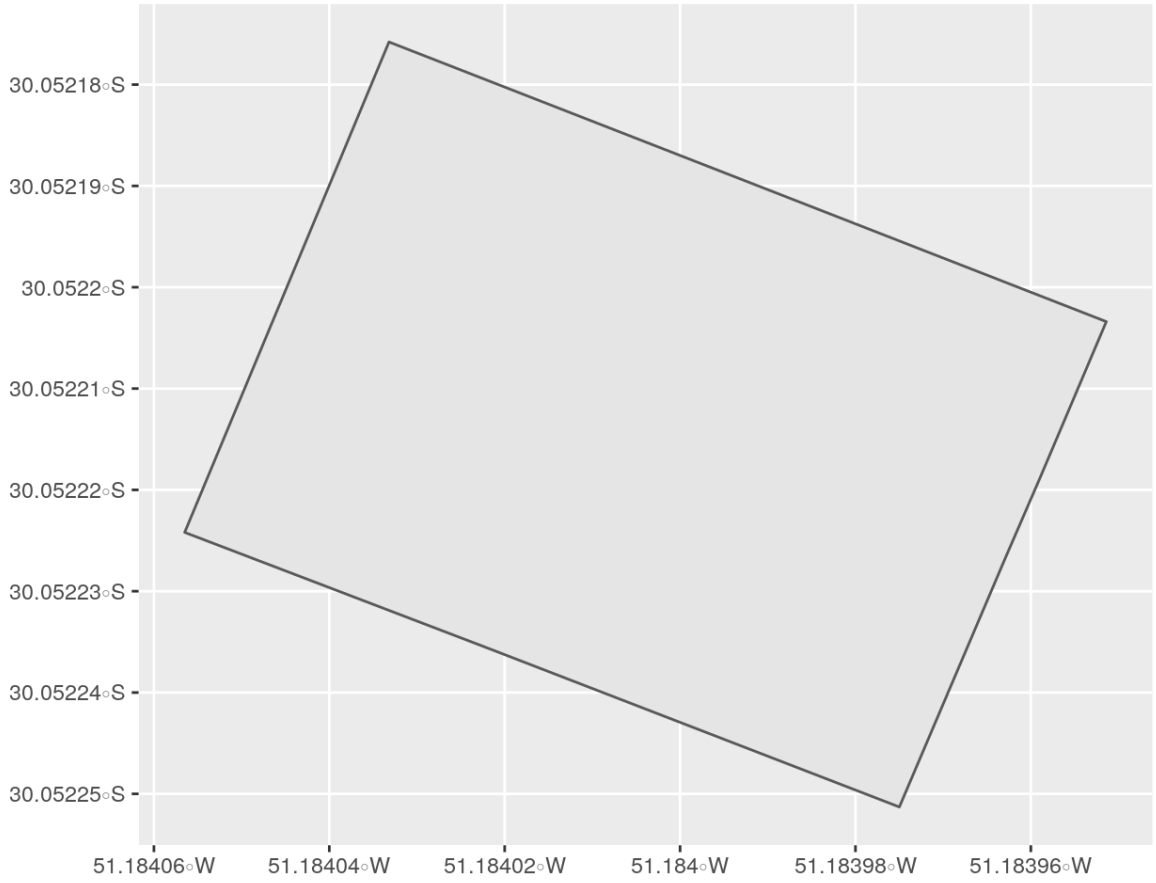
Vamos gerar nossas primeiras visualizações! A seguir, vamos apresentar graficamente o resultado de algumas consultas espaciais que foram executadas na disciplina.

3.1.1 Point Query

Retorne todas as construções que contém um ponto especificado pelo usuário

```
#executando a consulta, passo 1
point_query <- st_read(con,
                       query = "SELECT way
FROM buildings
WHERE ST_Intersects('SRID=3857;POINT(-5697777.26702199 -3510263.18413228)', way);")

#visualizando seu resultado, utilizando o passo 2
plot <- ggplot() + geom_sf(data = point_query) #estamos armazenando o gráfico na variável plot
#vamos ver o gráfico
plot
```

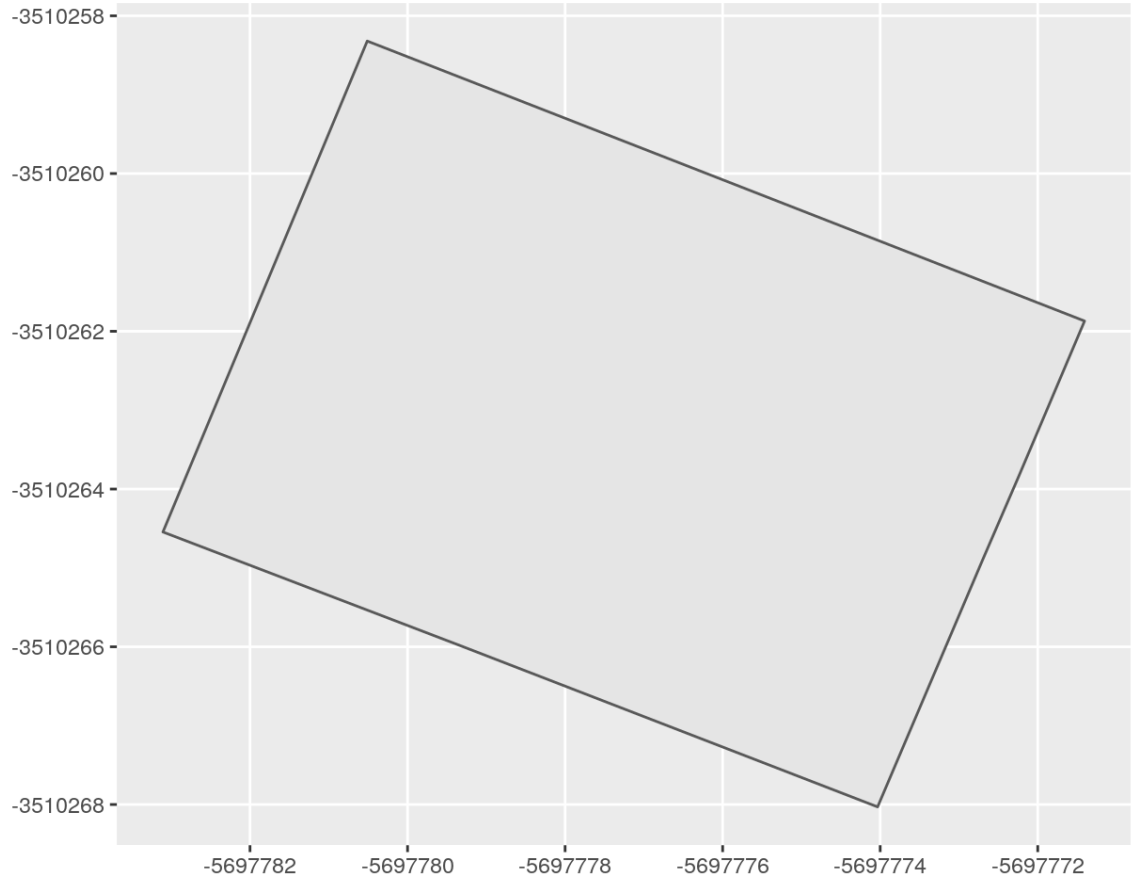



```
#ou ainda, poderíamos fazer a forma completa, conforme abaixo
#ggplot() + geom_sf(data = point_query, mapping = aes(geometry = way))
#ou
#ggplot() + geom_sf(point_query, mapping = aes(geometry = way))
```

Interessante! Esse é o nosso primeiro objeto espacial visualizado de forma gráfica. Entretanto, note o seguinte fato: **as coordenadas foram transformada para graus**. Isso aconteceu, pois, internamente o `geom_sf()` utiliza a função `coord_sf()` que atribui qual é o sistema de projeção será usada na visualização, tendo o SRID 4326 como padrão. Dessa forma, interna e automaticamente, as coordenadas foram transformadas do SRID 3857 para 4326.

Para visualizar os objetos no mesmo sistema de projeção usado no armazenamento dos objetos, é preciso adicionar ao `ggplot` a função `coord_sf()` informando ao seu parâmetro `datum` qual é o sistema de projeção de interesse (e.g., que pode ser obtido através da função `st_crs()`). Um exemplo é mostrado abaixo:

```
plot2 <- ggplot() + geom_sf(data = point_query) + coord_sf(datum = st_crs(point_query))
plot2
```




```
#na função coord_sf está sendo indicado que o datum a ser usado deve ser o mesmo que o do objeto po
int_query
```

Como resultado, temos o objeto espacial na projeção desejada.

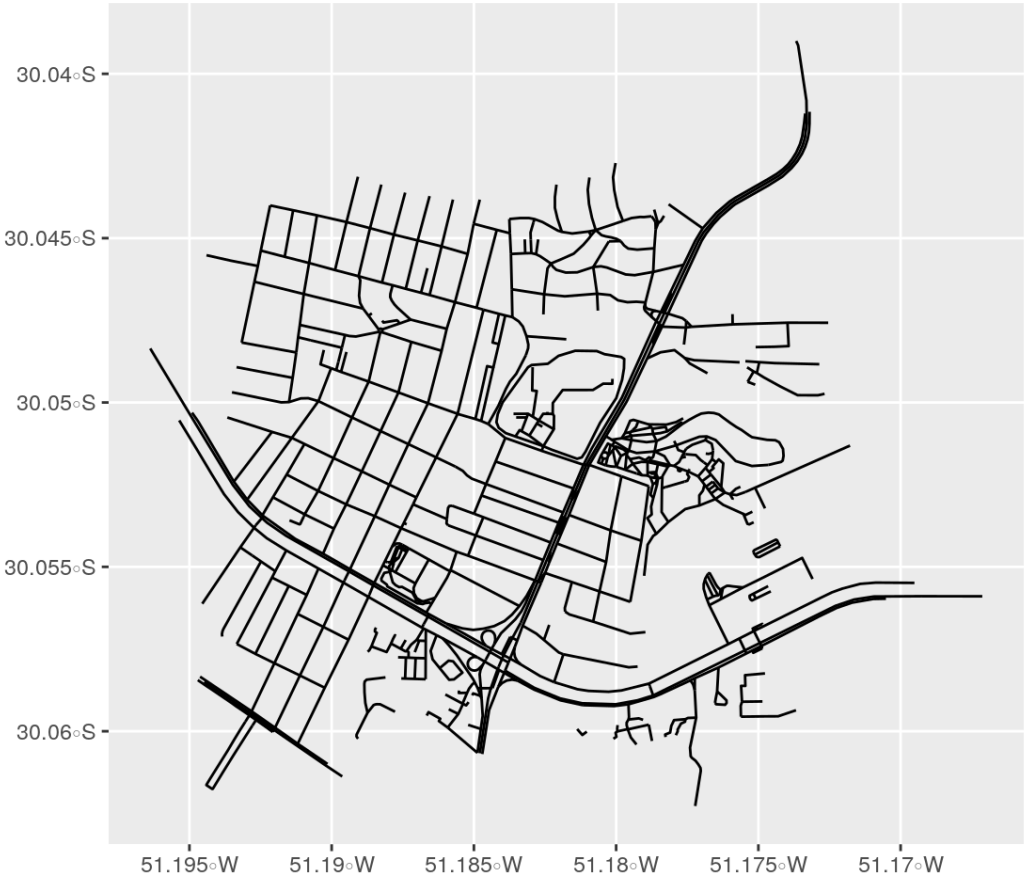
A partir desse ponto, a apostila não realizará qualquer interferência na projeção que o `geom_sf()` adotar para a visualização dos objetos espaciais.

3.1.2 Window Query

Retorne todos os highways que intersectam uma janela de consulta

```
#executando a consulta, passo 1
window_query <- st_read(con,
                        query = "SELECT way
FROM highways
WHERE ST_Intersects('SRID=3857;POLYGON((-5698777.26702199
-3511263.18413228,-5698777.26702199 -3509263.18413228,-5696777.26702199
-3509263.18413228,-5696777.26702199 -3511263.18413228,-5698777.26702199
-3511263.18413228))', way);")

# vamos somente visualizar direto, sem armazenar o plot em uma variável
ggplot() + geom_sf(data = window_query)
```

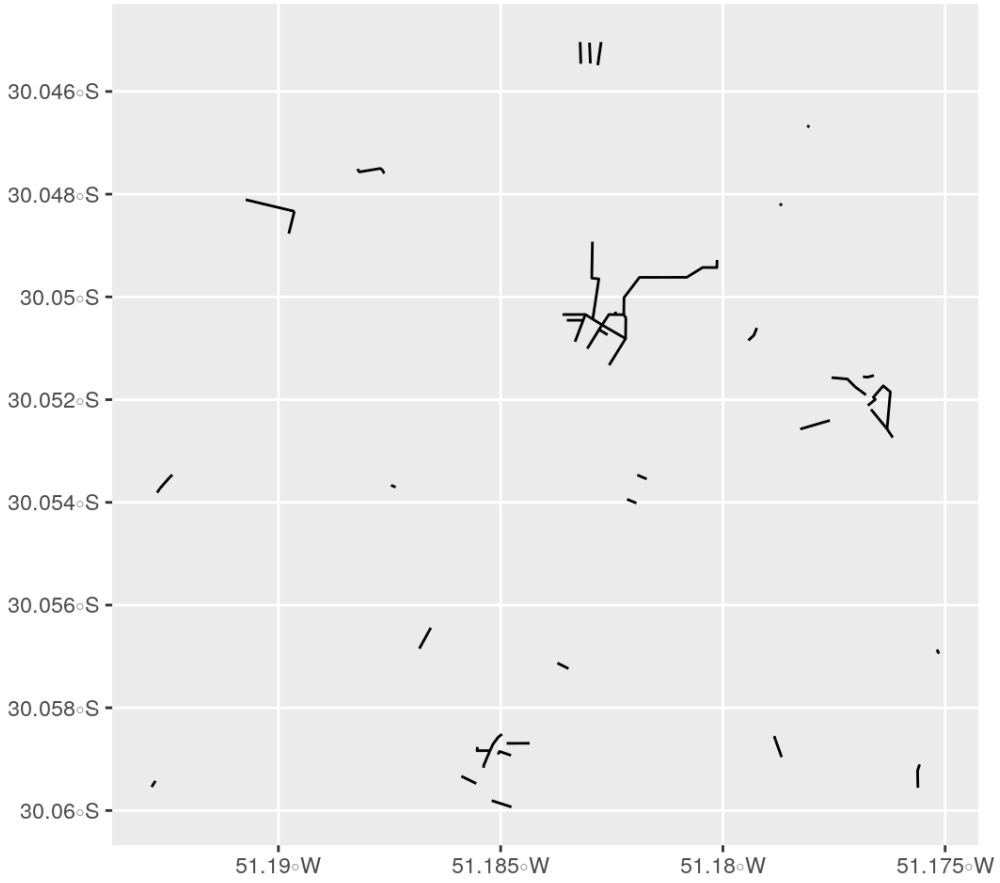


3.1.3 Containment Range Query

Retorne todos os caminhos para pedestres contidos dentro de uma janela de consulta

```
#executando a consulta, passo 1
containment_query <- st_read(con,
                        query = "SELECT way
FROM highways
WHERE highway = 'footway' AND
ST_Contains('SRID=3857;POLYGON((
-5698777.26702199 -3511263.18413228,
-5698777.26702199 -3509263.18413228,
-5696777.26702199 -3509263.18413228,
-5696777.26702199 -3511263.18413228,
-5698777.26702199 -3511263.18413228))', way);")

# vamos somente visualizar direto, sem armazenar o plot em uma variável
ggplot() + geom_sf(data = containment_query)
```

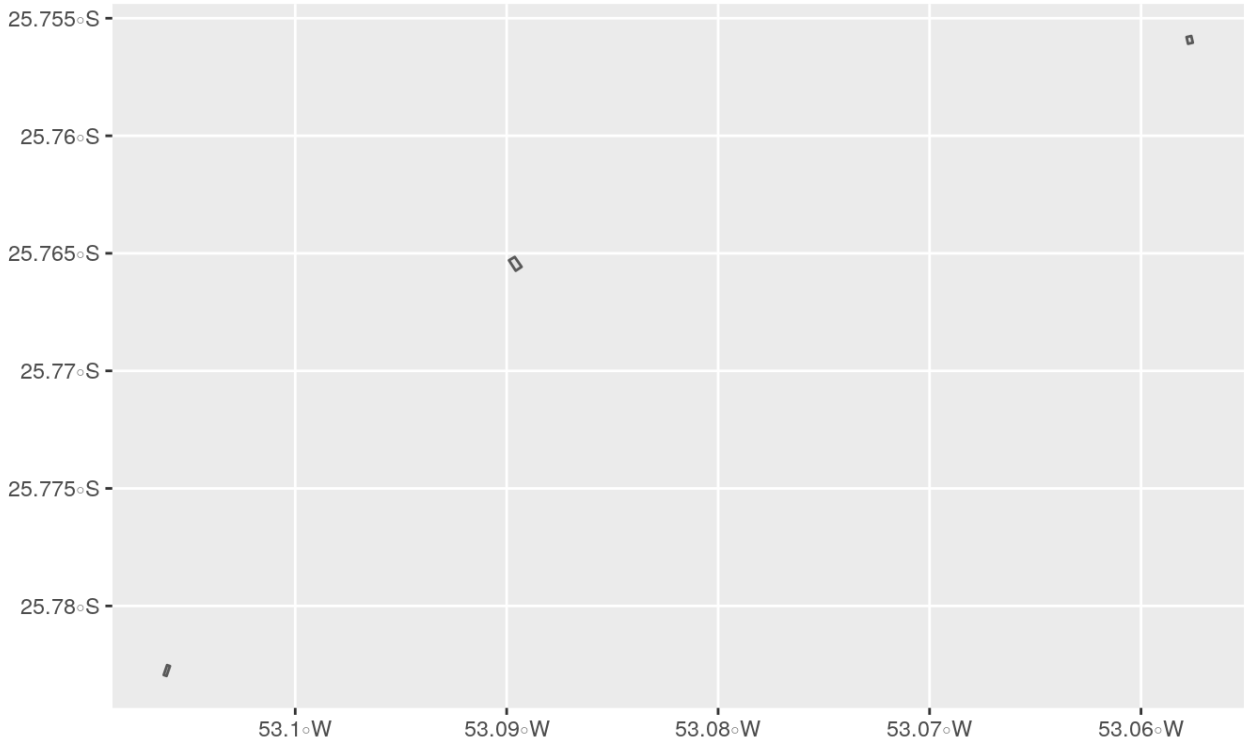


3.1.4 Consulta baseada em distância

Quais são as construções representando indústrias que estão dentro de um raio de 10km de um determinado ponto (informado pelo usuário)?

```
#executando a consulta, passo 1
distance_query <- st_read(con,
                      query = "SELECT way
FROM buildings
WHERE ST_DWithin('SRID=3857;POINT(-5908323.00630419 -2968170.2555387)', way, 10*1000) AND
  building = 'industrial';")

# vamos somente visualizar direto, sem armazenar o plot em uma variável
ggplot() + geom_sf(data = distance_query)
```

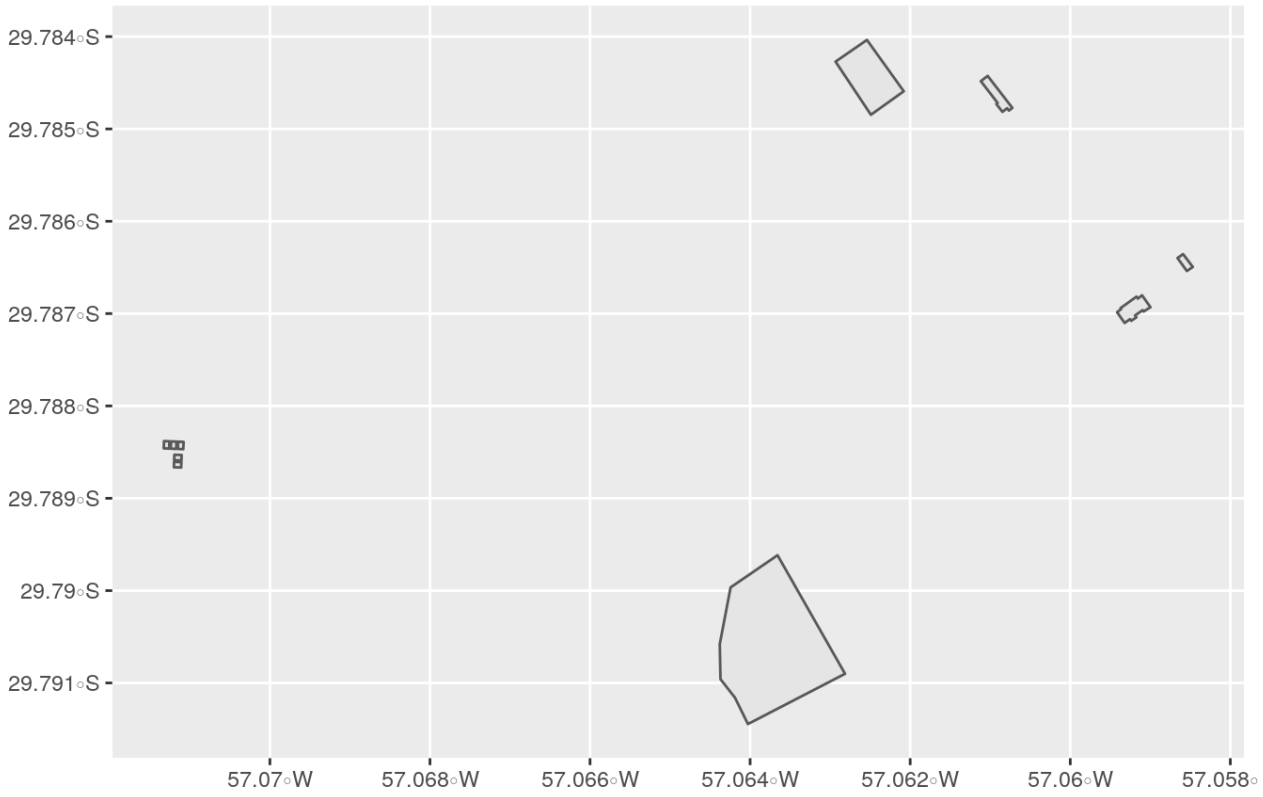


3.1.5 kNN

Retorne as 10 construções mais próximas de uma determinada escola

```
#executando a consulta, passo 1
knn_query <- st_read(con,
                     query = "SELECT way, building, ST_Distance(ST_GeomFromText('POLYGON((-635221
6.62158255 -3475849.92266669,-6352167.41836763 -3475923.72527349,-6352121.72171666 -3475891.095080
1,-6352172.93981437 -3475819.8066049,-6352216.62158255 -3475849.92266669))', 3857), way) as dist
FROM buildings
ORDER BY way <-> ST_GeomFromText('POLYGON((-6352216.62158255 -3475849.92266669, -6352167.41836763 -
3475923.72527349, -6352121.72171666 -3475891.0950801, -6352172.93981437 -3475819.8066049, -6352216.
62158255 -3475849.92266669))', 3857)
LIMIT 10;")

# vamos somente visualizar direto, sem armazenar o plot em uma variável
ggplot() + geom_sf(data = knn_query)
```

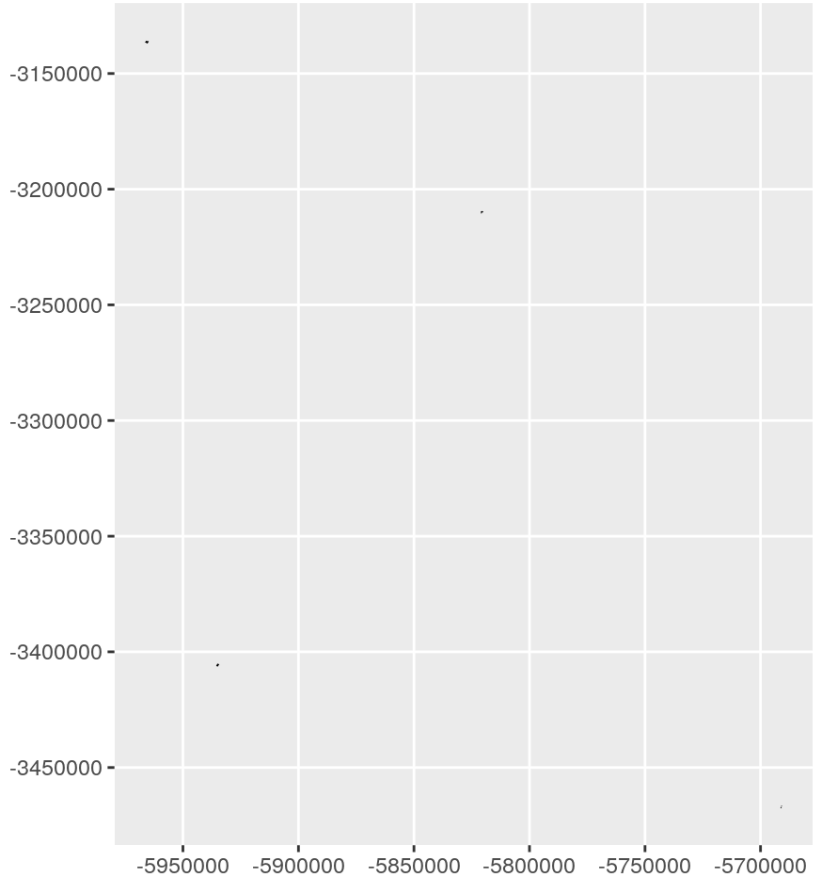


3.1.6 Junção espacial

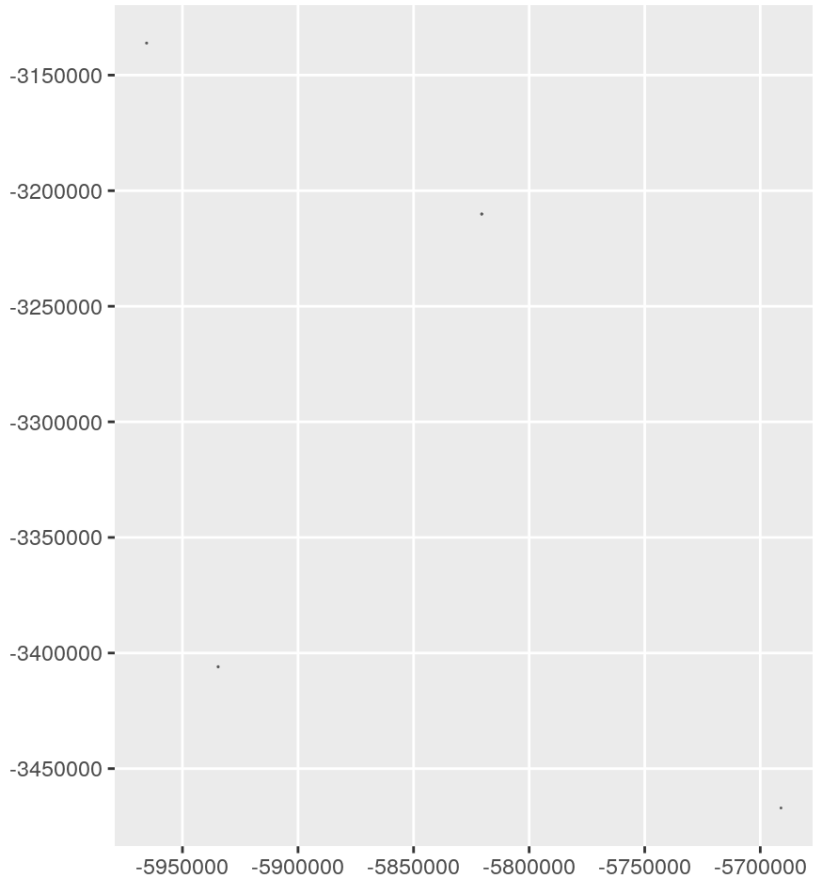
Retorne os pares de qualquer tipo de rodovia (highways) e fazendas (buildings) que se intersectam

```
#executando a consulta, passo 1
spatial_join <- st_read(con,
                        query = "SELECT h.way as highways, b.way as buildings
FROM highways as h, buildings as b
WHERE ST_Intersects(h.way, b.way)
AND b.building = 'farm';")

# vamos visualizar os highways
ggplot() + geom_sf(data = as.data.frame(spatial_join), mapping = aes(geometry = highways))
```




```
#agora, vamos visualizar os buildings
ggplot() + geom_sf(data = as.data.frame(spatial_join), mapping = aes(geometry = buildings))
```




```
#as vezes não precisa transformar para data.frame, basta informar direto a variável para data
```

É importante notar os seguintes pontos do código acima:

1. o objeto `spatial_join` é um `sf` que possui 2 colunas do tipo `sfc`, isto é, duas colunas contendo objetos espaciais (chamada de coluna espacial/geométrica ao longo da apostila)
2. para selecionar qual coluna geométrica deve ser visualizada, as seguintes modificações foram feitas:

- o `spatial_join` foi convertido para um `data.frame` através da função `as.data.frame()`
- posteriormente, foi indicado qual é a coluna geométrica que deveria ser usada na visualização, através da especificação `mapping = aes(geometry = XXX)` onde `XXX` é o nome da coluna (no nosso caso mesmo nome da coluna retornado pelo comando SQL).

3. os objetos ficaram **muito pequenos**! Ainda, eles ficaram na projeção do SRID 3857. Por que? A razão é conversão do `sf` para um `data.frame` e então a especificação de uma coluna espacial via parâmetro `geometry`. Essa coluna espacial é do tipo `sfc`, tendo seu próprio SRID. Nessa situação, o `geom_sf()` adota esse SRID como projeção na visualização dos objetos. Para contornar o problema da visualização de objetos muito pequenos, podemos visualizar porções de objetos em gráficos separados.

Uma questão em aberta é: como visualizar os juntos? Essa questão é respondida na seção que discute a visualização de objetos espaciais em camadas.

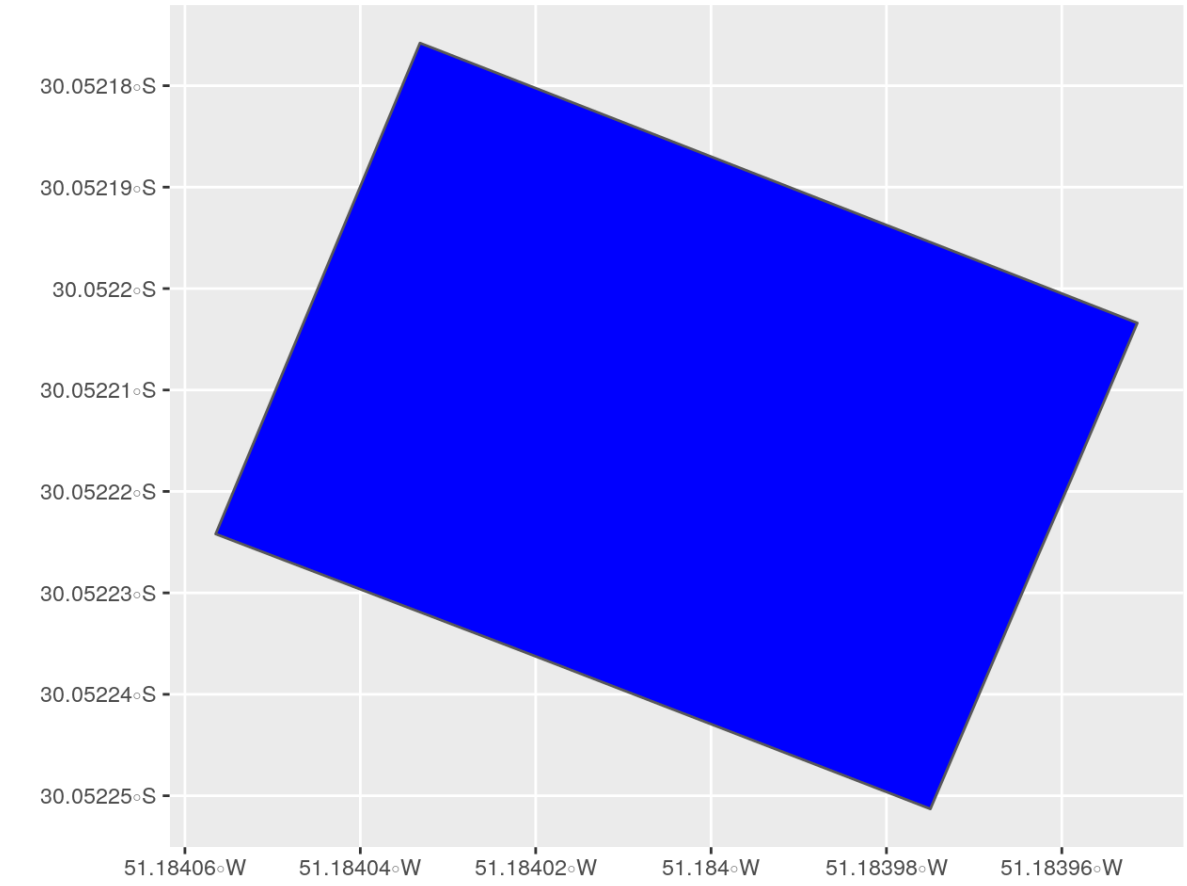
3.2 Colorindo objetos espaciais

Com a função `geom_sf()` é possível atribuir uma paleta de cores para os objetos espaciais visualizados. Podemos especificar uma cor simples para todos os objetos (ou seja, uma mesma cor que será aplicada a cada objeto espacial) ou cores baseadas nos valores de alguma coluna. No último caso, podemos criar uma escala de cores.

3.2.1 Cores fixas

Vamos atribuir algumas cores para os resultados das consultas espaciais processadas na seção anterior. Logo, atente-se que você deve ter executado todos os comandos anteriores para executar os próximos seguintes.

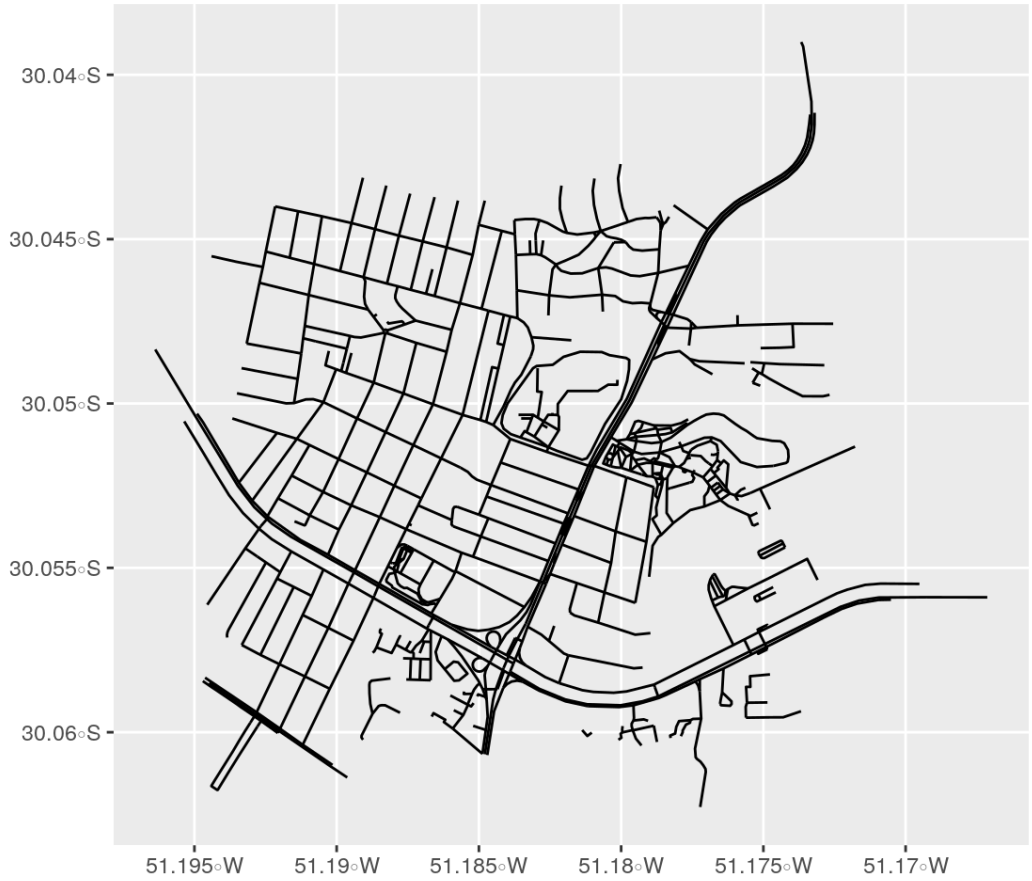
```
#neste caso a consulta retorna somente 1 objeto espacial, e vamos deixá-lo com a cor azul
ggplot() + geom_sf(data = point_query, fill = "blue")
```



No comando acima, a cor azul é denominada pelo parâmetro `fill` e ela é aplicada a **todos** os objetos espaciais sendo processados pela função `geom_sf()`. Como nesse caso existia apenas um objeto espacial, o resultado ficou bastante simples.

Veja mais um exemplo da aplicação de uma cor fixa a todos os objetos espaciais a seguir.

```
ggplot() + geom_sf(data = window_query, fill = "red")
```

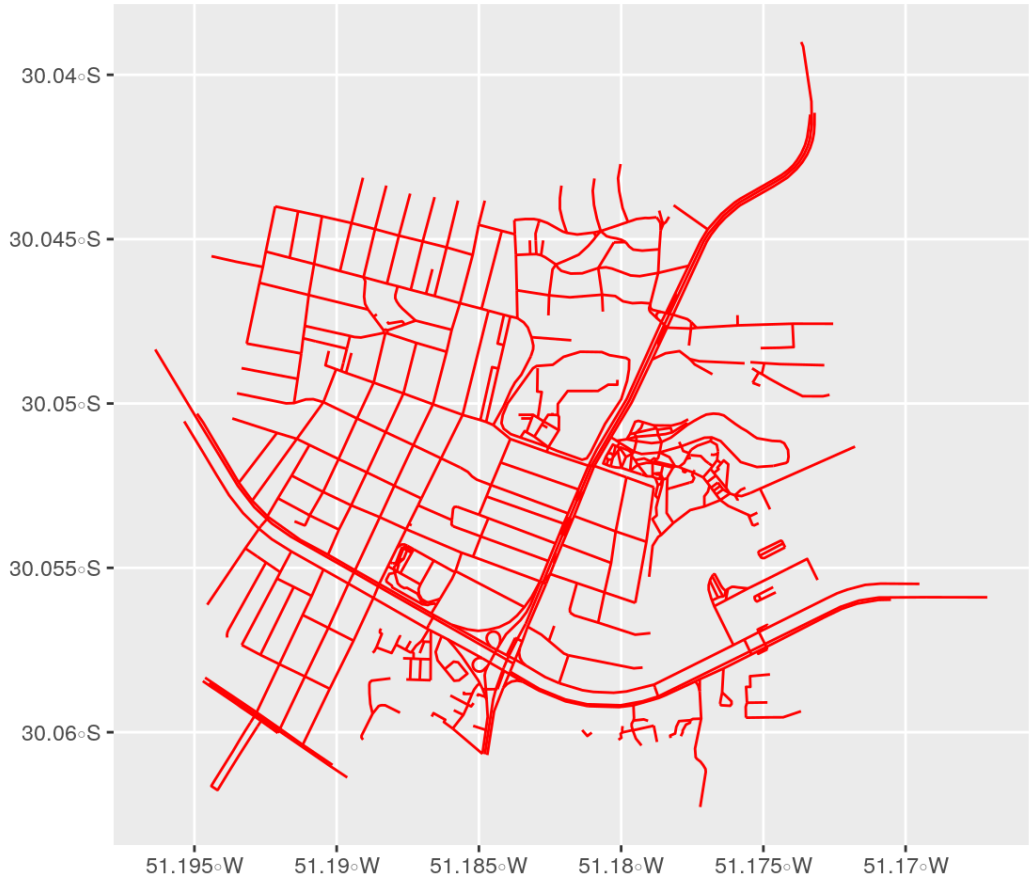


Ué, o que houve? A resposta é que o parâmetro `fill` refere-se a cor de preenchimento de um objeto espacial. No caso do `point_query`, o objeto espacial retornado é um polígono e o `gem_sf()` consegue atribuir uma cor para pintá-lo. Esse parâmetro também funcionaria para um objeto espacial do tipo ponto.

Entretanto, o `fill` não funciona para objetos espaciais do tipo linha. Nesse caso, deve-se usar o parâmetro `colour` no lugar!

Vamos modificar o código anterior para:

```
ggplot() + geom_sf(data = window_query, colour = "red")
```



Portanto, é importante tomar nota que:

1. `fill` é somente para polígonos e pontos
2. `colour` é somente para linhas

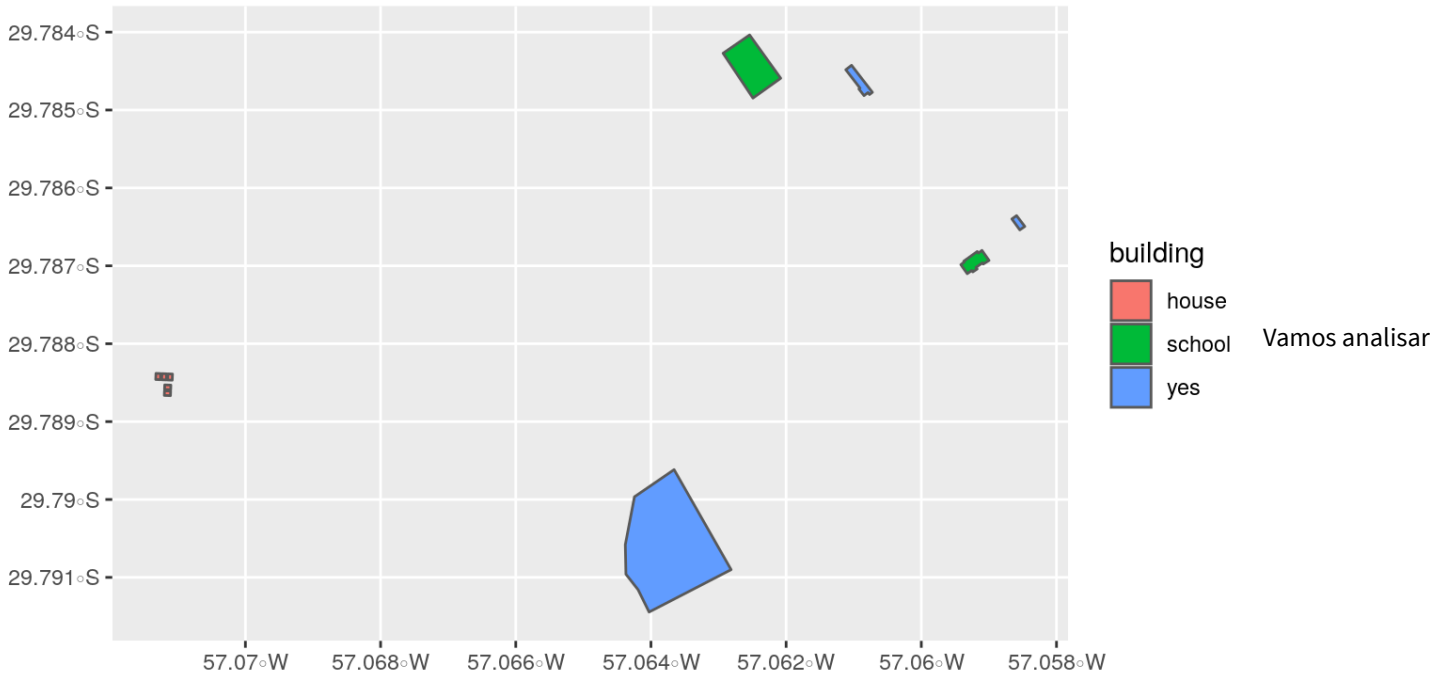
3.2.2 Cores variáveis

Como comentado anteriormente, é possível colorir cada objeto espacial com base em algum valor associado a ele. Por exemplo, alguma coluna que a descreve (e.g., categoria) ou baseando-se em valores numéricos.

Nos exemplos a seguir, vamos utilizar o resultado da nossa consulta kNN para demonstrar esses 2 casos.

Primeiro, vamos atribuir cores às construções retornadas com base na coluna `building` , a qual é uma coluna do tipo `character` que descreve que tipo de construção o objeto espacial está representando.

```
ggplot() + geom_sf(data = knn_query, mapping = aes(fill = building))
```

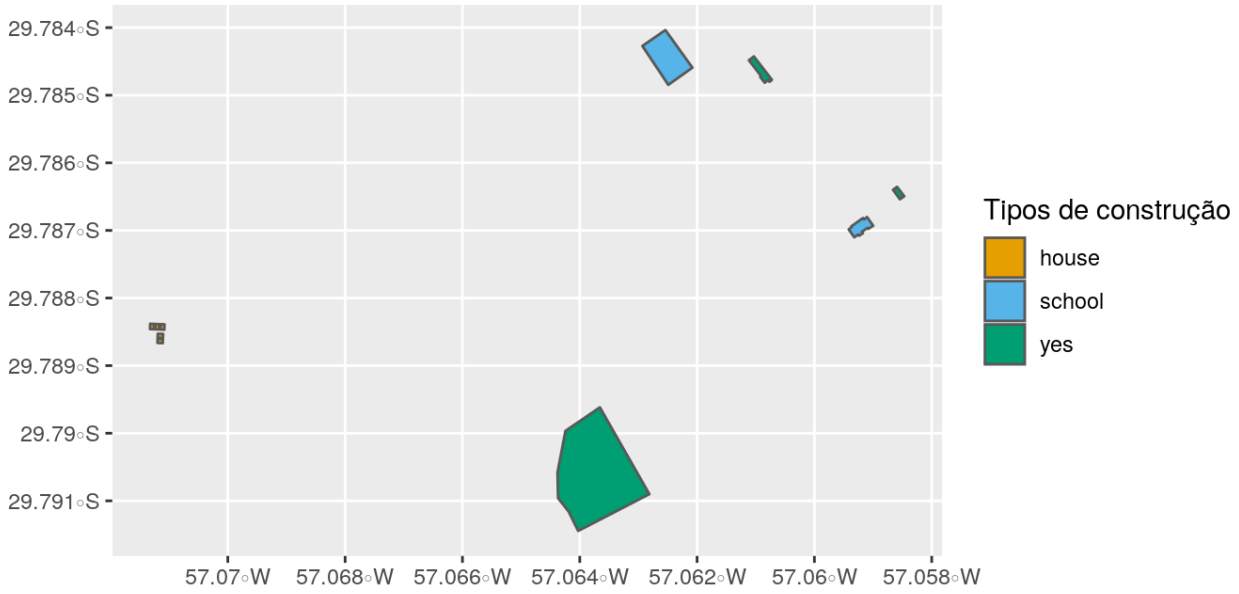


rapidamente o código acima:

1. agora a cor foi especificada pela função `aes()` através do parâmetro `mapping`. A função `aes()` especifica a estética da visualização, sendo que dentro dela foi especificada que o preenchimento da cor (atributo `fill`) deve ser feita com base nos valores distintos da coluna `building`.
2. Nesse caso, cada valor distinto da coluna `building` recebe uma cor de forma automática, criando uma legenda para ela.

Podemos ir melhorando a visualização do gráfico anterior. Por exemplo, alterando o esquema de cores e atribuindo um nome mais semântico à legenda criada:

```
ggplot() + geom_sf(data = knn_query, mapping = aes(fill = building)) +
  scale_fill_discrete(name = "Tipos de construção", type = c("#E69F00", "#56B4E9", "#009E73"))
```

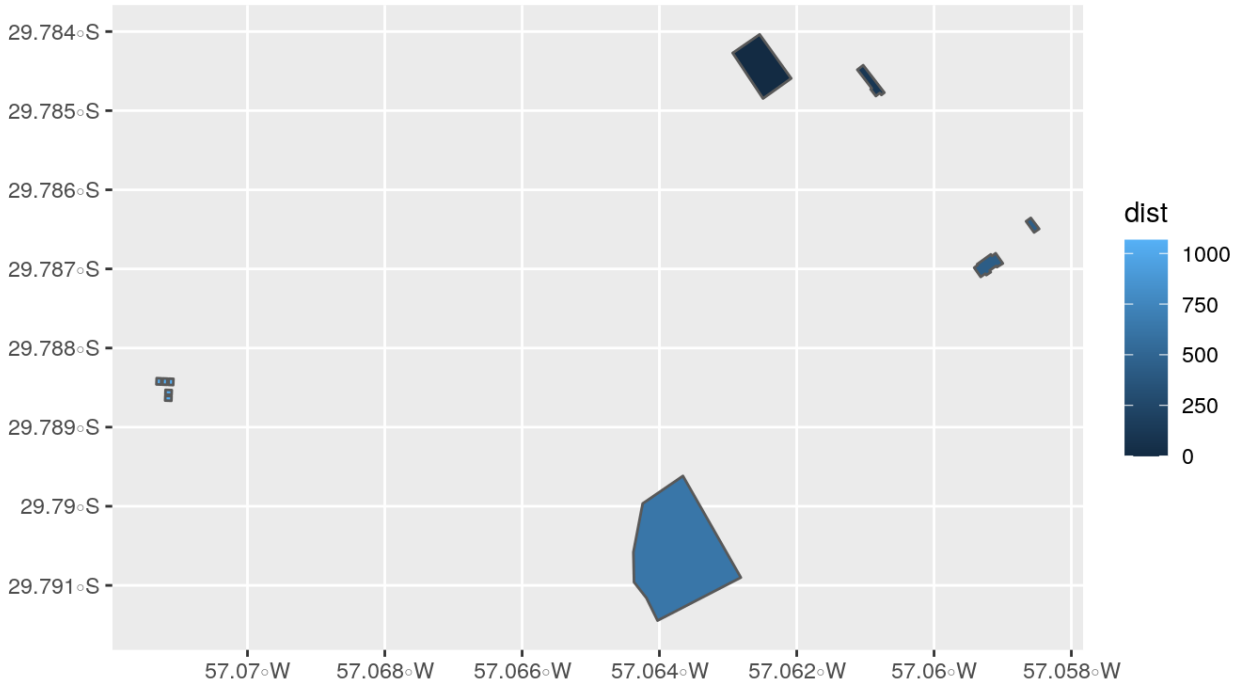


3. A função `scale_fill_discrete()` foi usada para atribuir um nome para a legenda criada. **IMPORTANTE:** note que seu nome tem um **fill**, indicando que essa escala se aplica ao parâmetro **fill** do `aes()` e **discrete** indica que a escala possui valores discretos! Essa função, assim como outras funções do `ggplot()`, é simplesmente "somada" ao gráfico. Ademais, essa função, por meio do parâmetro `type`, possibilita especificar cores para cada valor discreto associado ao objeto espacial. Nesse caso, temos 3 valores, logo, temos 3 cores. Se quiséssemos manter as cores originais, teriamos que simplesmente omitir o parâmetro `type`.

No segundo exemplo, vamos atribuir cores às construções retornadas com base na coluna `dist`, a qual é uma coluna do tipo `numeric` que mantém o quão distante um objeto espacial está do objeto de consulta. Dessa forma, essa coluna tem o domínio contínuo (embora sejam apenas 10 valores, imagine se tivéssemos mais objetos espaciais retornados).

O procedimento é bastante similar ao exemplo anterior:

```
ggplot() + geom_sf(data = knn_query, mapping = aes(fill = dist))
```

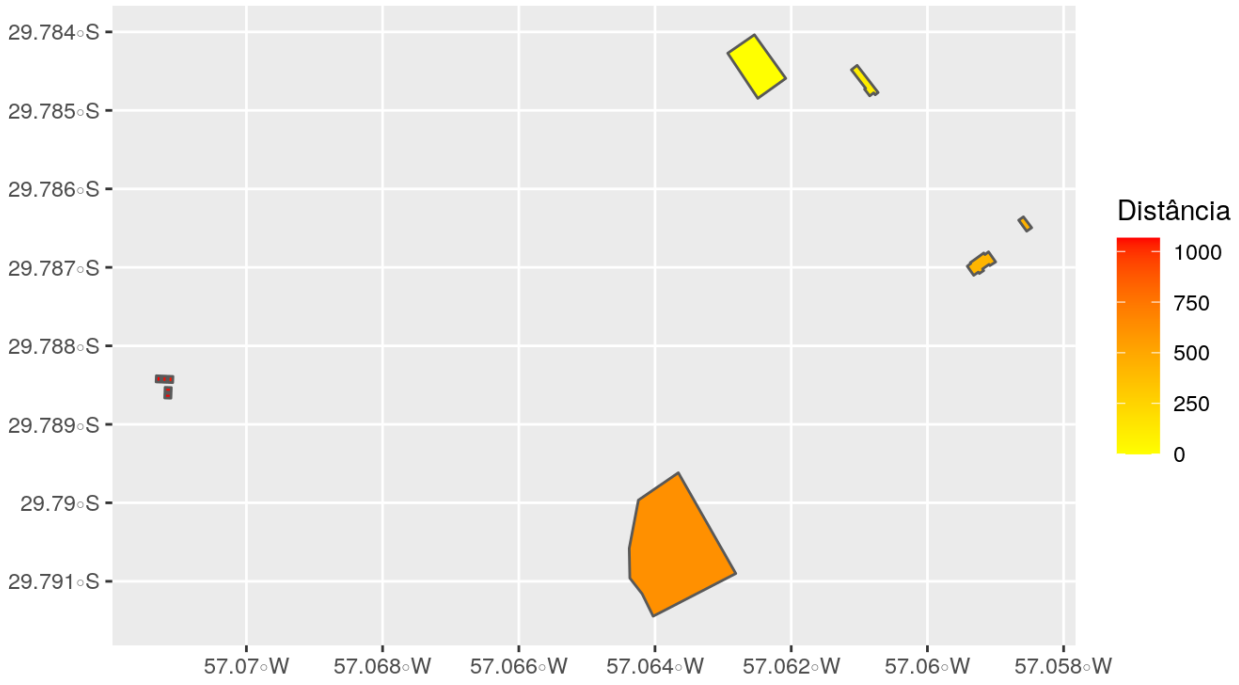


Porém, note a diferença:

- uma vez que a coluna indicado no parâmetro `fill` é do tipo numérico, uma escala contínua de cores foi criada. Essa escala possui um valor mínimo e vai variando até um valor máximo. Nesse caso, o valor se refere a distância. Uma cor mais escura indica que o objeto espacial retornado está bastante próximo ao objeto de consulta do kNN.

De forma similar ao exemplo anterior, podemos fazer ajustes na escala de cor. Para isso e para esse caso de variável contínua (numérica), devemos usar a função `scale_fill_continuous()` da seguinte forma:

```
ggplot() + geom_sf(data = knn_query, mapping = aes(fill = dist)) +
  scale_fill_continuous(name = "Distância", low = "yellow", high = "red")
```



Analisando o código cima observamos os seguintes pontos:

1. O uso da função `scale_fill_continuous()` que está associada ao ajuste de parâmetros de uma escala contínua para o parâmetro estético `fill`.
2. Nessa função, o parâmetro `name` indica o nome da legenda.
3. Outros parâmetros podem ser passados à essa função. Aqui, ajustamos as cores atribuindo qual é a cor a ser usada para o **menor** valor (parâmetro `low`) e qual é a cor a ser usada para o **maior** valor (parâmetro `high`). A própria função irá fazer os cálculos para produzir o *gradient* entre essas cores com base nos valores indicados. Outros parâmetros podem ser passados para a função, conforme sua documentação oficial (https://ggplot2.tidyverse.org/reference/scale_gradient.html).

4 Elaborando mapas com sobreposições (camadas)

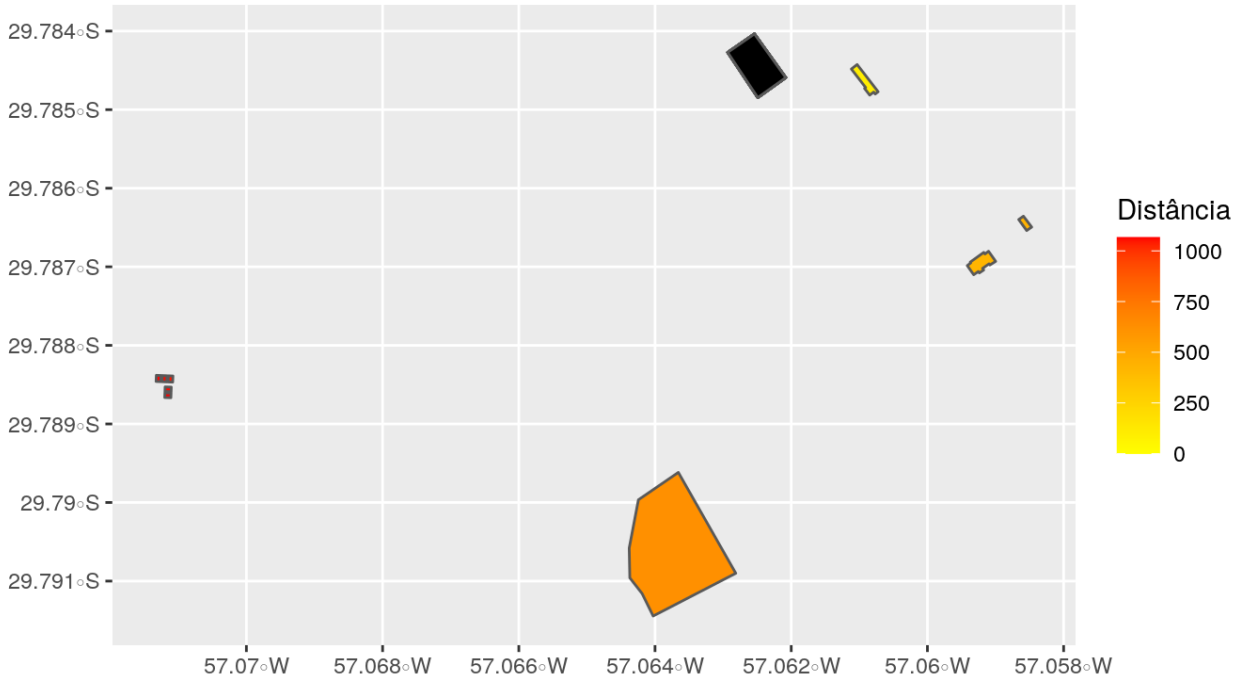
A escala de cores oferece uma boa noção de proximidade dos objetos espaciais retornados. Entretanto, como incluir nosso objeto de consulta para o kNN executado?

Iremos saber como responder essa pergunta nessa seção, ao visualizar camadas (*layers*) de objetos espaciais.

O código abaixo faz uma pequena mudança no código SQL da consulta kNN, para também adicionar o objeto de consulta como uma coluna da resposta (coluna `search_obj`):

```
knn_query_v2 <- st_read(con,
                        query = "SELECT way, building, ST_Distance(ST_GeomFromText('POLYGON((-635221
6.62158255 -3475849.92266669,-6352167.41836763 -3475923.72527349,-6352121.72171666 -3475891.095080
1,-6352172.93981437 -3475819.8066049,-6352216.62158255 -3475849.92266669))', 3857), way) as dist, S
T_GeomFromText('POLYGON((-6352216.62158255 -3475849.92266669, -6352167.41836763 -3475923.72527349,
 -6352121.72171666 -3475891.0950801, -6352172.93981437 -3475819.8066049, -6352216.62158255 -347584
9.92266669))', 3857) as search_obj
FROM buildings
ORDER BY way <-> ST_GeomFromText('POLYGON((-6352216.62158255 -3475849.92266669, -6352167.41836763 -
3475923.72527349, -6352121.72171666 -3475891.0950801, -6352172.93981437 -3475819.8066049, -6352216.
62158255 -3475849.92266669))', 3857)
LIMIT 10;")

ggplot() + geom_sf(data = knn_query_v2, mapping = aes(geometry = way, fill = dist)) +
  scale_fill_continuous(name = "Distância", low = "yellow", high = "red") +
  geom_sf(data = knn_query_v2, mapping = aes(geometry = search_obj), fill = "black")
```



Uma coisa bem interessante de notar é que o objeto de consulta do kNN (mostrado em preto) também faz parte da resposta.

Alguns pontos sobre o código acima:

1. a primeira camada de objetos espaciais contém os objetos da coluna `way`, onde cada objeto é colorido de acordo com o valor da coluna `dist`
2. a segunda camada de objetos espaciais contém o objeto de consulta indicado na coluna `search_obj` sendo que sua cor é preto.

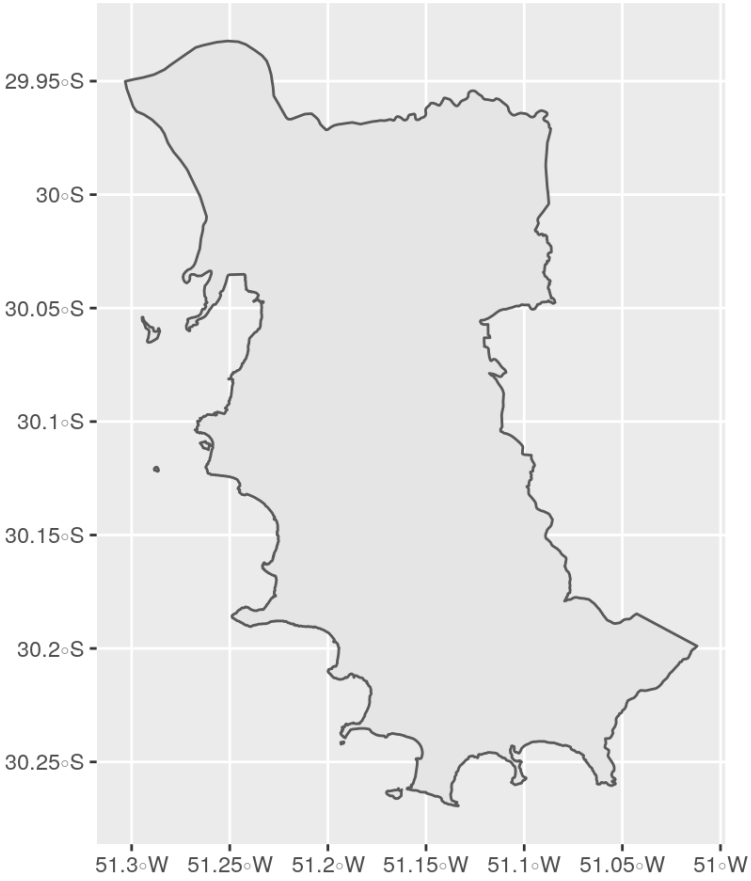
Dessa forma, nota-se que para adicionar uma camada, basta chamar novamente a função `geom_sf()` passando um outro conjunto de dados. Essa nova chamada é então "adicionada" ao gráfico sendo produzido pelo `ggplot()`.

Vamos ver outro exemplo a seguir. O código abaixo coloca as construções retornadas pela consulta *window query* executada anterior sobre a cidade que a janela intersecta:

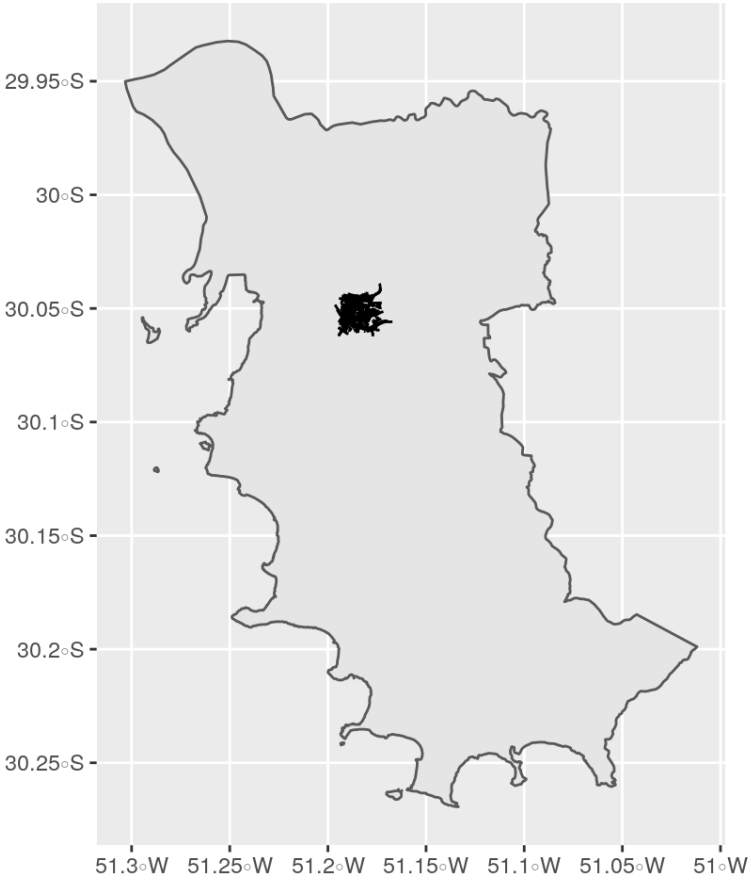
```
#qual é a cidade intersectada pela janela de consulta?
cidades_RS_wq <- st_read(con,
                      query = "SELECT * FROM br_municipios_2019
                     WHERE sigla_uf = 'RS' AND ST_Intersects('SRID=3857;POLYGON((-5698777.26702199
-3511263.18413228,-5698777.26702199 -3509263.18413228,-5696777.26702199
-3509263.18413228,-5696777.26702199 -3511263.18413228,-5698777.26702199
-3511263.18413228))', st_transform(geom, 3857));")

#antes de visualizar os elementos em camadas, vamos visualizar somente as cidades do SC:

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




```
ggplot() + geom_sf(data = cidades_RS_wq) + geom_sf(data = window_query)
```



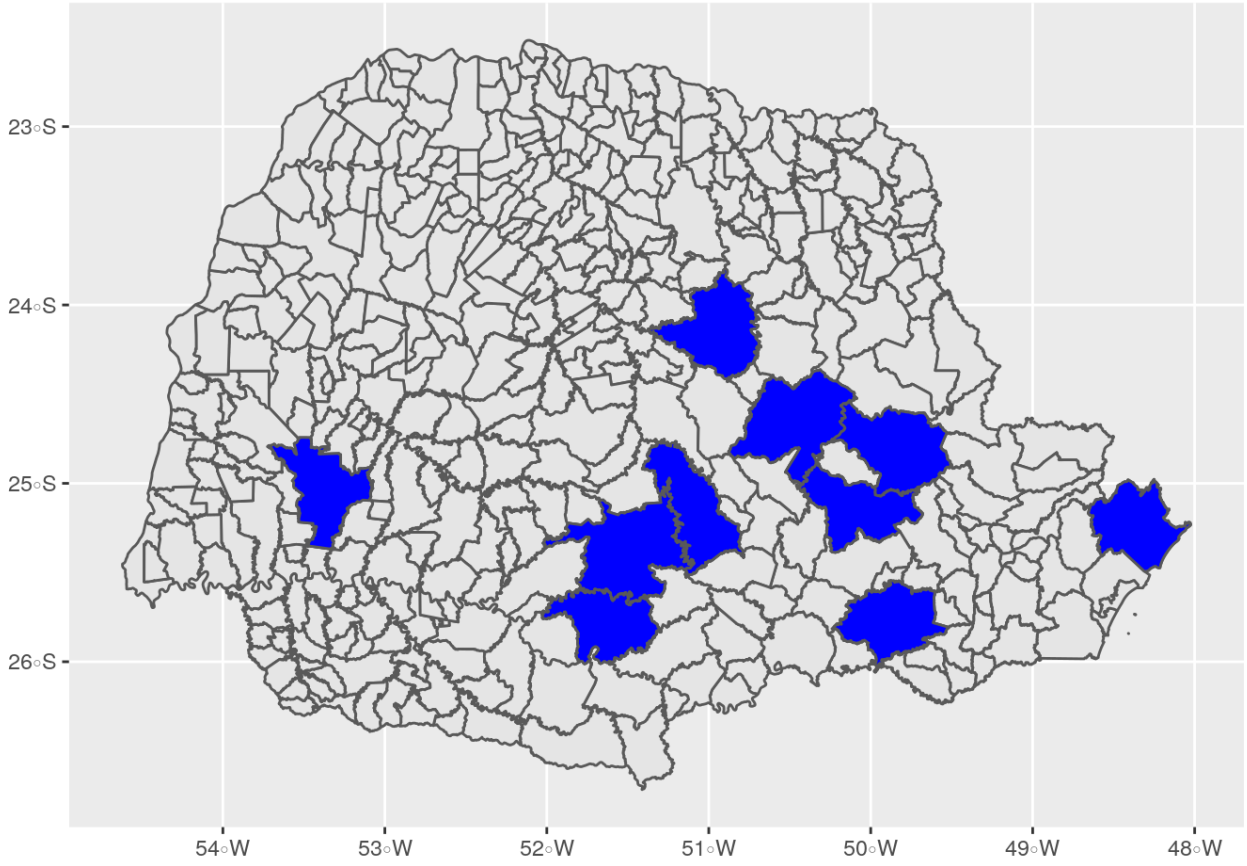
Outro tipo de sobreposição de camadas é o destaque de alguns objetos espaciais de um mesmo conjunto de dados.

Por exemplo, a visualização abaixo mostra todas as cidades do estado de PR, porém, destacando as 10 cidades que possuem as maiores áreas:

```
#todas as cidades de PR
cidades_PR <- st_read(con,
                      query = "SELECT * FROM br_municipios_2019
                     WHERE sigla_uf = 'PR';")

#capturando os 10 elementos com maior área usando o dplyr, entretanto, poderíamos ter processado es
se tipo de coisa usando SQL!
cidades_PR_area <- arrange(cidades_PR, desc(area_km2)) %>% slice(1:10)

ggplot() + geom_sf(data = cidades_PR) + geom_sf(data = cidades_PR_area, fill = "blue")
```

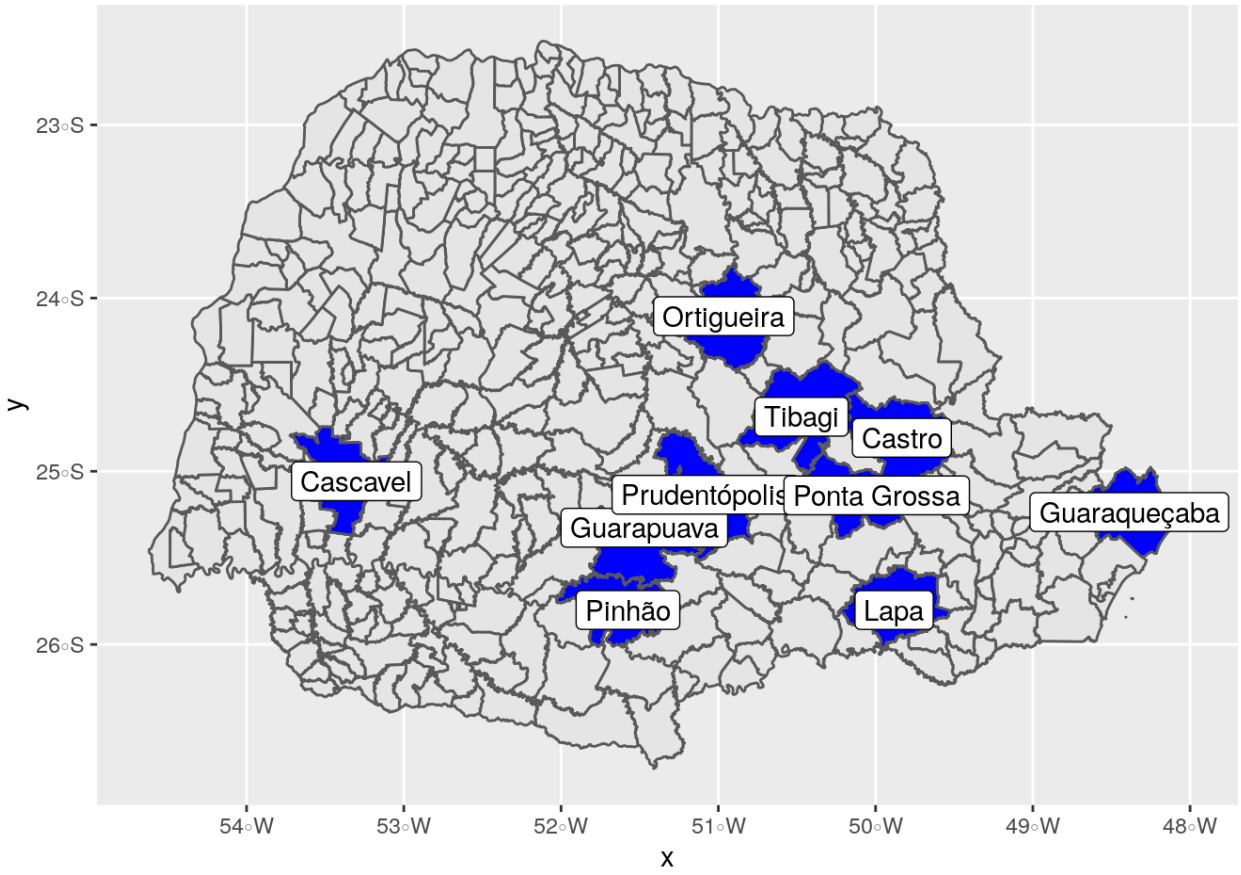


O gráfico acima realmente destaca as cidades com maiores áreas. Entretanto, **quais os nomes delas?**

Podemos incluir `labels` (identificadores) dos objetos espaciais por meio da função `geom_sf_label()`, de maneira bastante semelhante à função `geom_sf()`, mas informando pelo parâmetro `label` qual é a coluna que deve ser usado para associar um texto a um objeto espacial:

```
ggplot() + geom_sf(data = cidades_PR) + geom_sf(data = cidades_PR_area, fill = "blue") + geom_sf_la
bel(data = cidades_PR_area, mapping = aes(label = nm_mun))
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



Visualização de dados (em geral, não somente objetos espaciais) pode demandar bastante tempo e pesquisa pelas funções adequadas que realizem o tipo de visualização desejada. Nesse sentido, existem ainda muitas outras bibliotecas disponíveis para a linguagem R para visualizar dados espaciais. Na seção seguinte, uma leitura complementar sobre o tópico de visualização é indicada.

5 Leitura complementar

Para saber mais sobre como estilizar gráficos gerados pelo `ggplot`, é interessante verificar seus **temas completos** em sua documentação oficial (https://ggplot2.tidyverse.org/reference/ggtheme.html).

Para saber mais sobre o suporte das linguagens de programação mais populares para ciência de dados para manipular dados espaciais, recomenda-se a leitura do seguinte artigo:

Md Mahbub Alam, Luís Torgo, Albert Bifet. A Survey on Spatio-temporal Data Analytics Systems. Preprints and early-stage research may not have been peer reviewed yet, 2021. (https://www.researchgate.net/publication/350160598_A_Survey_on_Spatio-temporal_Data_Analytics_Systems)

Na Seção 7 desse artigo, os autores discutem o suporte a manipulação de dados espaciais oferecido por diversos pacotes e bibliotecas das linguagens de programação R e Python. É interessante destacar que o pacote `sf` possui diversos pontos positivos perante aos outros pacotes existentes.

Ainda, recomenda-se a leitura do seguinte artigo para compreender o suporte a manipulação de dados espaciais em sistemas de processamento paralelo e distribuído de dados baseados no Hadoop e Spark:

João Pedro Castro, Anderson Carniel & Cristina Ciferri. Analyzing spatial analytics systems based on Hadoop and Spark: A user perspective. Software Practice and Experience, 50 (12), 2121-2144, 2020. (https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2882)

Note que diversos sistemas que manipulam dados tendem a oferecer algum suporte mais especializado para manipular dados espaciais e eles oferecem diversas operações espaciais que foram vistas ao longo da disciplina (e.g., definidas pela OGC ou amplamente aceitas pela comunidade científica).

6 Conclusões

Esta apostila apresentou uma forma de se analisar visualmente o resultado de consultas espaciais utilizando o pacote `sf` da linguagem de programação R. Os principais pontos a serem levados em conta são:

1. é possível realizar uma consulta espacial (e.g., *Window Query*) ao emiti-la pelo R. Ela, então, pode ser eficientemente processada pelo PostgreSQL/PostGIS (devido, por exemplo, a utilização de estrutura de indexação espacial)
2. com os dados oriundos de uma consula espacial, é possível visualizar os objetos espaciais contidos nelas, adicionando informações adicionais conforme necessário
3. ainda, é possível elaborar mapas mais complexos utilizando **diversas camadas** de objetos espaciais. Este tipo de operação é conhecida como *overlay*.