Tutorial: Geocomputation with R

Geographic vector data in R

Jannes Muenchow, Robin Lovelace

ERUM Budapest, 2018-05-14

Find the slides and the code

https://github.com/jannes-m/erum18_geocompr

Please install following packages:

```
install.packages(c("sf", "raster", "spData", "dplyr", "RQGIS"))
```

Or from docker:

```
docker run -d -p 8787:8787 -v ${pwd}:/data robinlovelace/geocompr
```

Simple features in R

Simple features in R

Simple feature access is a widely used ISO standard. Edzer Pebesma implemented simple features in R via the **sf** package.

Simple features in R

Simple feature access is a widely used ISO standard. Edzer Pebesma implemented simple features in R via the **sf** package.

```
library(sf)
```

```
## Linking to GEOS 3.6.1, GDAL 2.2.3, proj.4 4.9.3
```

sf automatically links to GEOS, GDAL and Proj.4.

Simple features in R

Simple feature access is a widely used ISO standard. Edzer Pebesma implemented simple features in R via the **sf** package.

```
library(sf)
```

```
## Linking to GEOS 3.6.1, GDAL 2.2.3, proj.4 4.9.3
```

sf automatically links to GEOS, GDAL and Proj.4.

```
data(random_points, package = "RQGIS")
class(random_points)
```

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

Simple features in R

Simple feature access is a widely used ISO standard. Edzer Pebesma implemented simple features in R via the **sf** package.

```
library(sf)
```

```
## Linking to GEOS 3.6.1, GDAL 2.2.3, proj.4 4.9.3
```

sf automatically links to GEOS, GDAL and Proj.4.

```
data(random_points, package = "RQGIS")
class(random_points)
```

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

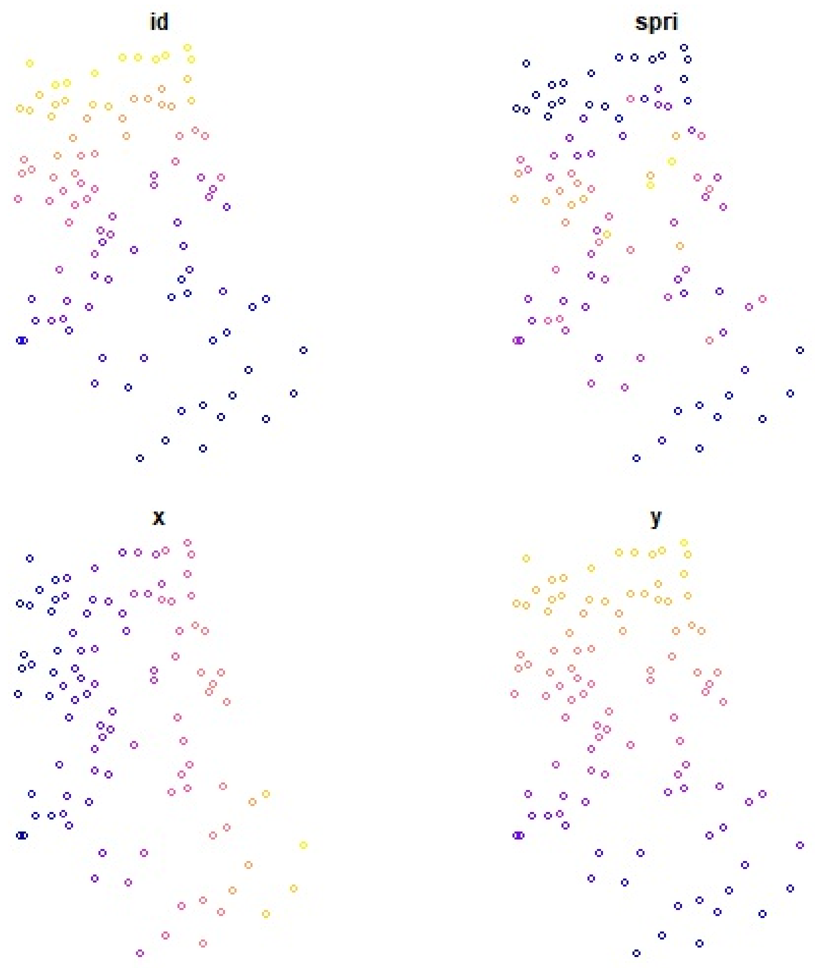
This is a `data.frame`, i.e, an S3 object (as opposed to `SpatialObjects`).

Simple features in R

```
plot(random_points)
```

Simple features in R

```
plot(random_points)
```

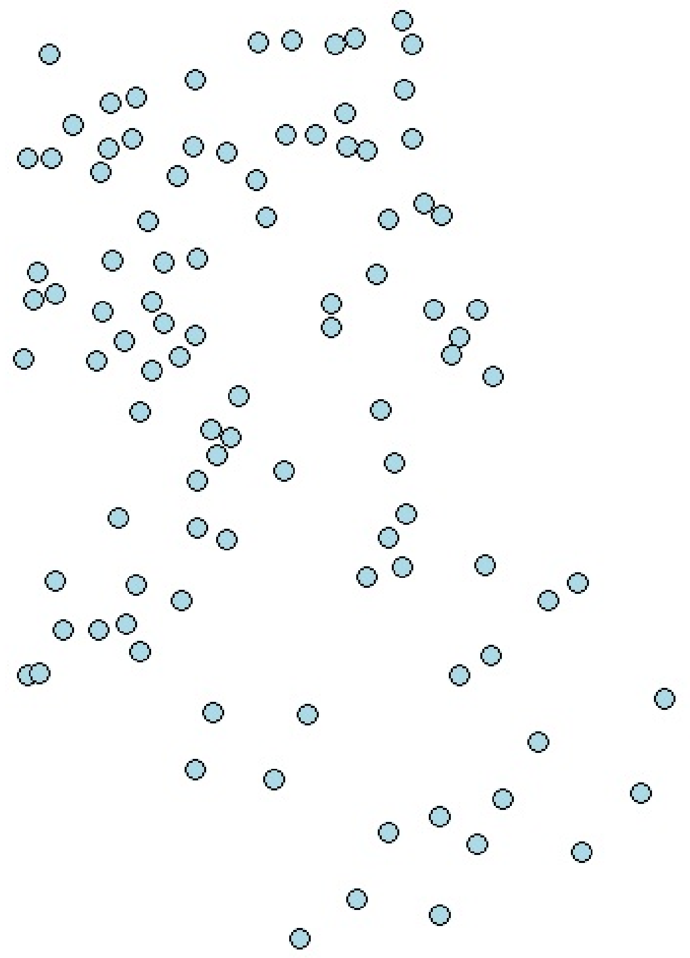


Simple features in R

```
plot(
  st_geometry(random_points),
  pch = 16, cex = 2,
  col = "black"
  bg = "lightblue"
  )
```

Simple features in R

```
plot(
  st_geometry(random_points),
  pch = 16, cex = 2,
  col = "black"
  bg = "lightblue"
  )
```



Simple features in R

```
library(dplyr)
select(random_points, 1:2) %>%
  head(2)
```

```
## Simple feature collection with 2 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 796749.3 ymin: 8932621 xmax: 797178.6 ymax: 893275
## epsg (SRID):    32717
## proj4string:    +proj=utm +zone=17 +south +datum=WGS84 +units=m +no_defs
##   id spri                  geometry
## 1  1    4 POINT (797178.6 8932755)
## 2  2    4 POINT (796749.3 8932621)
```

A few things to note:

- **sf** works with the **tidyverse**.

Simple features in R

```r
library(dplyr)
select(random_points, 1:2) %>%
  head(2)
```

```
## Simple feature collection with 2 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 796749.3 ymin: 8932621 xmax: 797178.6 ymax: 893275
## epsg (SRID):    32717
## proj4string:    +proj=utm +zone=17 +south +datum=WGS84 +units=m +no_defs
##   id spri                  geometry
## 1  1    4 POINT (797178.6 8932755)
## 2  2    4 POINT (796749.3 8932621)
```

A few things to note:

- **sf** works with the **tidyverse**.
- Geometry is **just** another column.

Simple features in R

```r
library(dplyr)
select(random_points, 1:2) %>%
  head(2)
```

```
## Simple feature collection with 2 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 796749.3 ymin: 8932621 xmax: 797178.6 ymax: 893275
## epsg (SRID):    32717
## proj4string:    +proj=utm +zone=17 +south +datum=WGS84 +units=m +no_defs
##   id spri                  geometry
## 1  1    4 POINT (797178.6 8932755)
## 2  2    4 POINT (796749.3 8932621)
```

A few things to note:

- **sf** works with the **tidyverse**.
- Geometry is **just** another column.
- The geometry column is **sticky**.

Things to note continued:

- Each observation (row) has a geometry (which can consist of multiple features, think of polygons with holes or multi-part polygons).

Things to note continued:

- Each observation (row) has a geometry (which can consist of multiple features, think of polygons with holes or multi-part polygons).
- The geometry column is a so-called **list-column**.

Things to note continued:

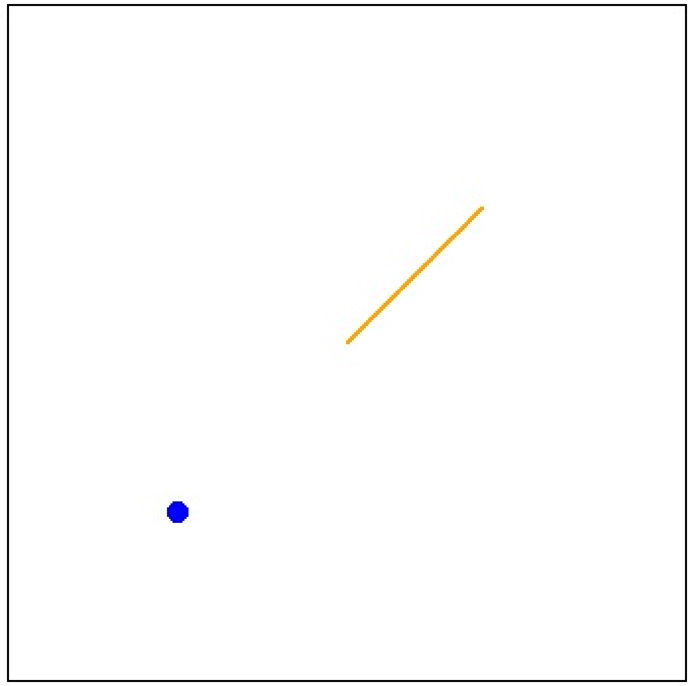
- Each observation (row) has a geometry (which can consist of multiple features, think of polygons with holes or multi-part polygons).
- The geometry column is a so-called **list-column**.
- The geometry is build up of **simple** R structures.

Geometries

```r
# one point (a numeric vector)
p = st_point(c(1.25, 1.25))
# one line (a matrix consisting of at
# least two points)
mat = matrix(c(1.5, 1.5, 1.7, 1.7),
             ncol = 2, byrow = TRUE)
l = st_linestring(mat)
# one polygon
mat = matrix(c(1, 1, 1, 2, 2, 2,
               2, 1, 1, 1),
             ncol = 2, byrow = TRUE)
# a list of one or more matrices
# consisting of points
poly = st_polygon(list(mat))
# plot it
plot(poly)
plot(p, pch = 16, col = "blue",
     cex = 2, add = TRUE)
plot(l, cex = 2, col = "orange",
     lwd = 2, add = TRUE)
```

Geometries

```r
# one point (a numeric vector)
p = st_point(c(1.25, 1.25))
# one line (a matrix consisting of at
# least two points)
mat = matrix(c(1.5, 1.5, 1.7, 1.7),
             ncol = 2, byrow = TRUE)
l = st_linestring(mat)
# one polygon
mat = matrix(c(1, 1, 1, 2, 2, 2,
               2, 1, 1, 1),
             ncol = 2, byrow = TRUE)
# a list of one or more matrices
# consisting of points
poly = st_polygon(list(mat))
# plot it
plot(poly)
plot(p, pch = 16, col = "blue",
     cex = 2, add = TRUE)
plot(l, cex = 2, col = "orange",
     lwd = 2, add = TRUE)
```



Putting it all together

sf uses three classes to represent simple features in R:

- `sf` is the `data.frame` with the attributes and the geometry list-column

Putting it all together

sf uses three classes to represent simple features in R:

- `sf` is the `data.frame` with the attributes and the geometry list-column
- The geometry list column is of class `sfc`.

```
lc = random_points %>%
  st_geometry
class(lc)
```

```
## [1] "sfc_POINT" "sfc"
```

Putting it all together

sf uses three classes to represent simple features in R:

- `sf` is the `data.frame` with the attributes and the geometry list-column
- The geometry list column is of class `sfc`.

```
lc = random_points %>%
  st_geometry
class(lc)
```

```
## [1] "sfc_POINT" "sfc"
```

- Each feature of the list column is of class `sfg`.

```
class(lc[[1]])
```

```
## [1] "XY"    "POINT" "sfg"
```

Putting it all together

sf uses three classes to represent simple features in R:

- `sf` is the `data.frame` with the attributes and the geometry list-column
- The geometry list column is of class `sfc`.

```
lc = random_points %>%
  st_geometry
class(lc)
```

```
## [1] "sfc_POINT" "sfc"
```

- Each feature of the list column is of class `sfg`.

```
class(lc[[1]])
```

```
## [1] "XY"    "POINT" "sfg"
```

For more information, refer to `vignette("sf1", package = "sf")` and https://geocompr.robinlovelace.net/spatial-class.html#vector-data

Attribute operations

Attribute operations

- `sf` objects are basically dataframes and thus can be handled like any other R object.

Attribute operations

- `sf` objects are basically dataframes and thus can be handled like any other R object.

```
dim(random_points)
```

```
## [1] 100   5
```

Attribute operations

- `sf` objects are basically dataframes and thus can be handled like any other R object.

```
dim(random_points)
```

```
## [1] 100   5
```

```
str(random_points)
```

```
## Classes 'sf' and 'data.frame':    100 obs. of  5 variables:
##  $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ spri    : int  4 4 3 2 4 5 6 2 3 3 ...
##  $ x       : num  797179 796749 796816 797023 796647 ...
##  $ y       : num  8932755 8932621 8932739 8932600 8932692 ...
##  $ geometry:sfc_POINT of length 100; first list element: Classes 'XY', 'I
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA
##   ..- attr(*, "names")= chr  "id" "spri" "x" "y"
```

Subsetting

```
# first 2 rows and first 2 columns
random_points[1:2, 1:2]
```

```
## Simple feature collection with 2 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 796749.3 ymin: 8932621 xmax: 797178.6 ymax: 893275
## epsg (SRID):    32717
## proj4string:    +proj=utm +zone=17 +south +datum=WGS84 +units=m +no_defs
##   id spri                  geometry
## 1  1    4 POINT (797178.6 8932755)
## 2  2    4 POINT (796749.3 8932621)
```

Tidyverse

- When **dplyr** is also attached to the global environment, a number of generic methods of the tidyverse become available for `sf`-objects, most notably the one-table verbs `select`, `slice`, `filter`, `arrange`, `mutate`, `summarize` (and `group_by`).

Tidyverse

- When **dplyr** is also attached to the global environment, a number of generic methods of the tidyverse become available for `sf`-objects, most notably the one-table verbs `select`, `slice`, `filter`, `arrange`, `mutate`, `summarize` (and `group_by`).
- Piped operations are also supported (%>%).

Tidyverse

- When **dplyr** is also attached to the global environment, a number of generic methods of the tidyverse become available for `sf`-objects, most notably the one-table verbs `select`, `slice`, `filter`, `arrange`, `mutate`, `summarize` (and `group_by`).
- Piped operations are also supported (%>%).

```
select(random_points, 1:2) %>%
  slice(1:2)
```

```
## Simple feature collection with 2 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 795551.4 ymin: 8932370 xmax: 797242.3 ymax: 893480
## epsg (SRID):    32717
## proj4string:    +proj=utm +zone=17 +south +datum=WGS84 +units=m +no_defs
## # A tibble: 2 x 3
##      id  spri          geometry
##   <int> <int>       <POINT [m]>
## 1     1     4 (797178.6 8932755)
## 2     2     4 (796749.3 8932621)
```

Vector attribute operations

Further reading: https://geocompr.robinlovelace.net/attr.html#vector-attribute-manipulation

Your turn

- Select all observations of `random_points` (`data("random_points, package = "RQGIS")`) which have more than 10 species (column `spri`). Plot the geometry of all points and add your selection to the plot in another color.
- Based on `spri` add a categorical column to `random_points` with 0-5 corresponding to `low`, 5-10 to `medium` and >10 to `high`.
- Optional: create two points of class `sfg` and convert them into an object of class `sf` which has an `id` and a `geometry` column.

Spatial attribute operations

Spatial attribute operations

Spatial operations make use of spatial relationship between objects (features). In the following we will address:

Spatial attribute operations

Spatial operations make use of spatial relationship between objects (features). In the following we will address:

- Spatial subsetting

Spatial attribute operations

Spatial operations make use of spatial relationship between objects (features). In the following we will address:

- Spatial subsetting
- Topological or neighborhood operations

Spatial attribute operations

Spatial operations make use of spatial relationship between objects (features). In the following we will address:

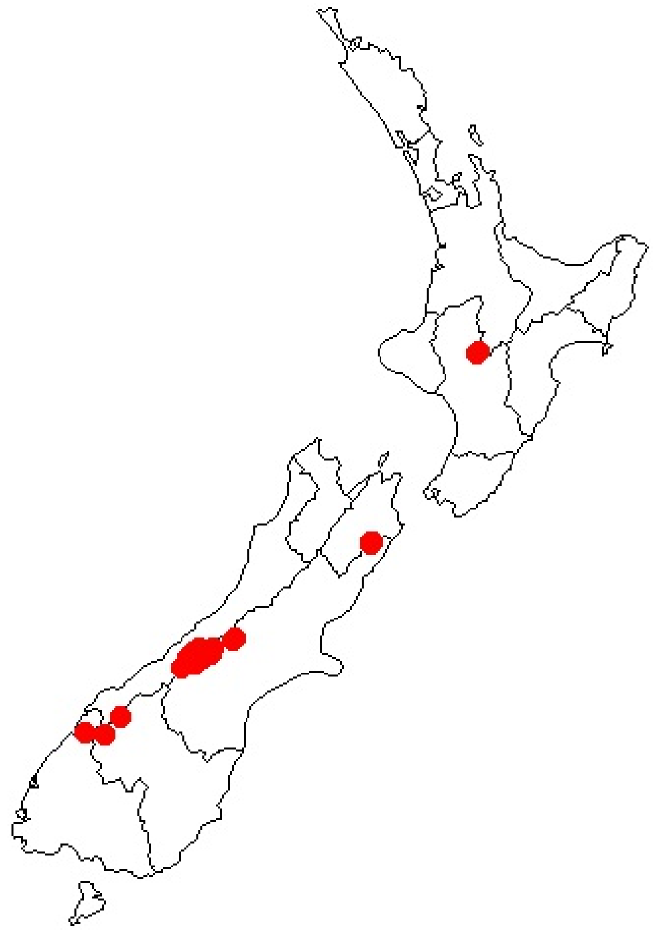
- Spatial subsetting
- Topological or neighborhood operations
- Spatial joins (spatial overlay)

Spatial subsetting

```
# spData makes available
# nz and nz_height
library(spData)
plot(st_geometry(nz))
plot(st_geometry(nz_height),
     pch = 16, col = "red2"
     cex = 2, add = TRUE)
```

Spatial subsetting

```r
# spData makes available
# nz and nz_height
library(spData)
plot(st_geometry(nz))
plot(st_geometry(nz_height),
     pch = 16, col = "red2"
     cex = 2, add = TRUE)
```

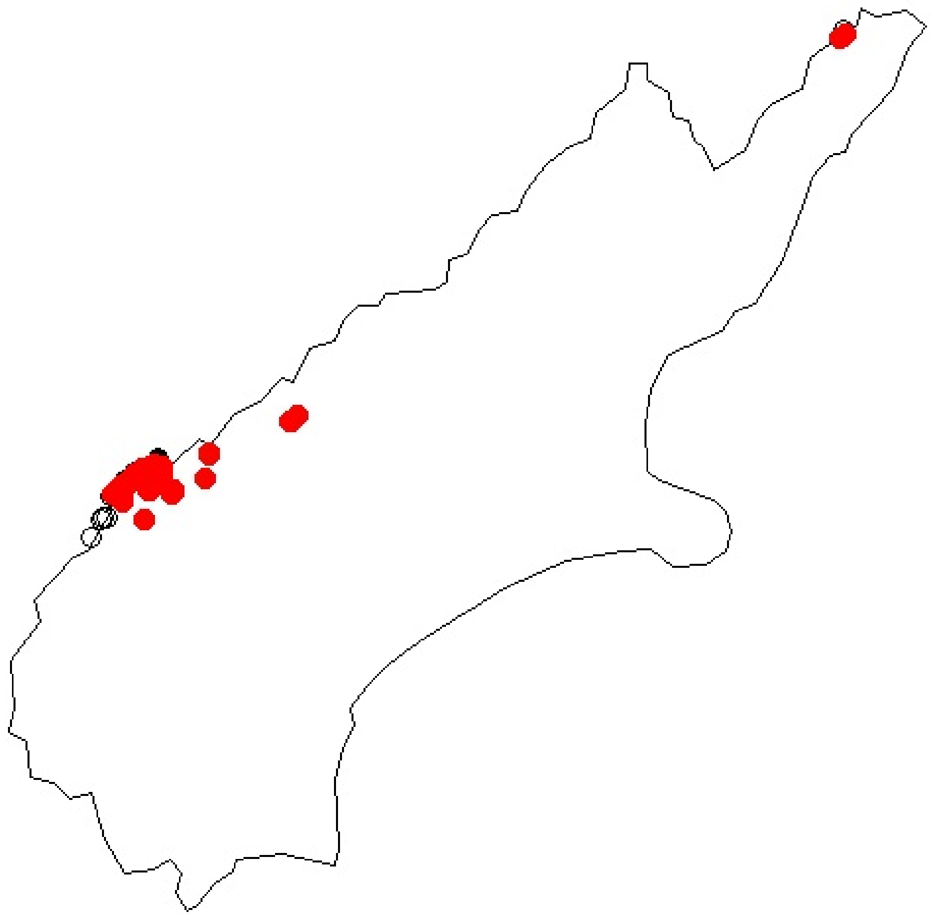


Spatial subsetting

```r
canterbury = nz %>%
  filter(Name == "Canterbury")
plot(st_geometry(canterbury))
plot(st_geometry(nz_height),
     cex = 2, add = TRUE)
# spatial subsetting
sel = nz_height[canterbury, ]
plot(st_geometry(sel), cex = 2,
     col = "red", pch = 16,
     add = TRUE)
```

Spatial subsetting

```
canterbury = nz %>%
  filter(Name == "Canterbury")
plot(st_geometry(canterbury))
plot(st_geometry(nz_height),
     cex = 2, add = TRUE)
# spatial subsetting
sel = nz_height[canterbury, ]
plot(st_geometry(sel), cex = 2,
     col = "red", pch = 16,
     add = TRUE)
```



Topological relations

Implicitly our subsetting used `st_intersects`, i.e. it returned all featured that touched or overlapped.

```
nz_height[canterbury, op = st_intersects]
# see also
?st_sf
```

Topological relations

Implicitly our subsetting used `st_intersects`, i.e. it returned all featured that touched or overlapped.

```
nz_height[canterbury, op = st_intersects]
# see also
?st_sf
```

We can use `st_intersects` individually. This returns a boolean vector if there is an intersection.

Topological relations

Implicitly our subsetting used `st_intersects`, i.e. it returned all featured that touched or overlapped.

```
nz_height[canterbury, op = st_intersects]
# see also
?st_sf
```

We can use `st_intersects` individually. This returns a boolean vector if there is an intersection.

```
st_intersects(nz_height, canterbury, sparse = FALSE) %>% head
```

```
##       [,1]
## [1,] FALSE
## [2,] FALSE
## [3,] FALSE
## [4,] FALSE
## [5,]  TRUE
## [6,]  TRUE
```

aside from `st_intersects` there are further predicates:

- `st_disjoint`: the opposite of `st_intersects`
- `st_touches`: just touching
- ...
- have a look at `?st_intersects` for a complete list and description

Spatial join

Transfer the attribute of one spatial object to another spatial object based on intersecting geometries. For example, let us add the region name from `nz` to `nz_height` (so far consisting of columns `t50_fid`, `elevation` and `geometry`).

Spatial join

Transfer the attribute of one spatial object to another spatial object based on intersecting geometries. For example, let us add the region name from `nz` to `nz_height` (so far consisting of columns `t50_fid`, `elevation` and `geometry`).

```
join = st_join(nz_height, select(nz, Name))
```

Spatial join

Transfer the attribute of one spatial object to another spatial object based on intersecting geometries. For example, let us add the region name from `nz` to `nz_height` (so far consisting of columns `t50_fid`, `elevation` and `geometry`).

```
join = st_join(nz_height, select(nz, Name))
```

```
slice(join, 1:2)
```

```
## Simple feature collection with 2 features and 3 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 1204143 ymin: 5048309 xmax: 1822492 ymax: 5650492
## epsg (SRID):    2193
## proj4string:    +proj=tmerc +lat_0=0 +lon_0=173 +k=0.9996 +x_0=1600000 +y
## # A tibble: 2 x 4
##   t50_fid elevation Name                    geometry
##     <int>     <int> <chr>              <POINT [m]>
## 1 2353944      2723 Southland (1204143 5049971)
## 2 2354404      2820 Otago     (1234725 5048309)
```

Spatial attribute operations on vector data

Further reading: https://geocompr.robinlovelace.net/spatial-operations.html#spatial-vec

Your turn

- Filter the Canterbury region from `nz`, and find all summits of `nz_height` that do not intersect with the Canterbury region (both datasets come with the `spData` package).
- What happens if we spatially join the elevation column of `nz_height` to `nz`?

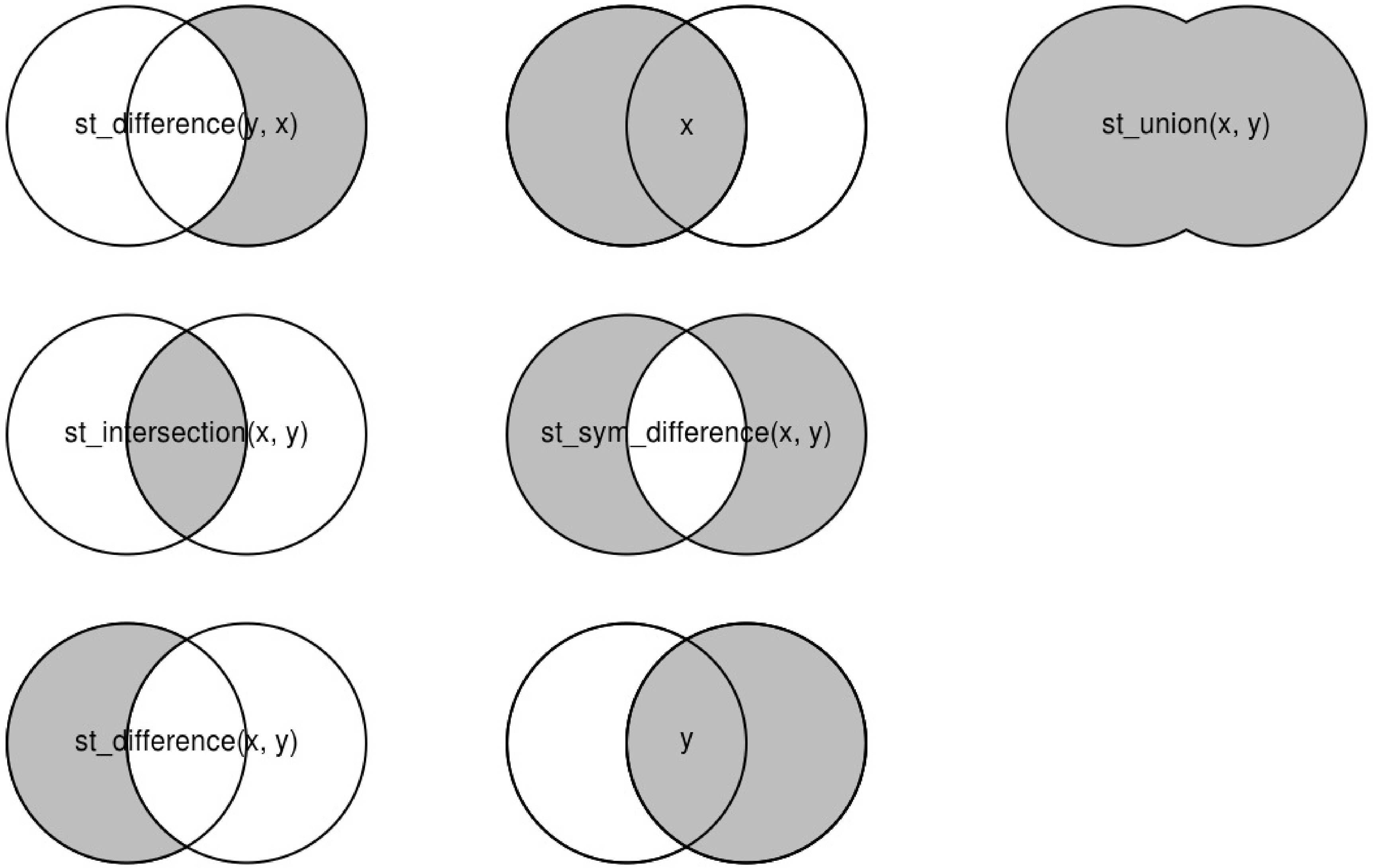
Geometric operations

Geometric operations

What if we want the geometric intersection of two overlapping spatial objects instead of a boolean vector?

Geometric operations

What if we want the geometric intersection of two overlapping spatial objects instead of a boolean vector?

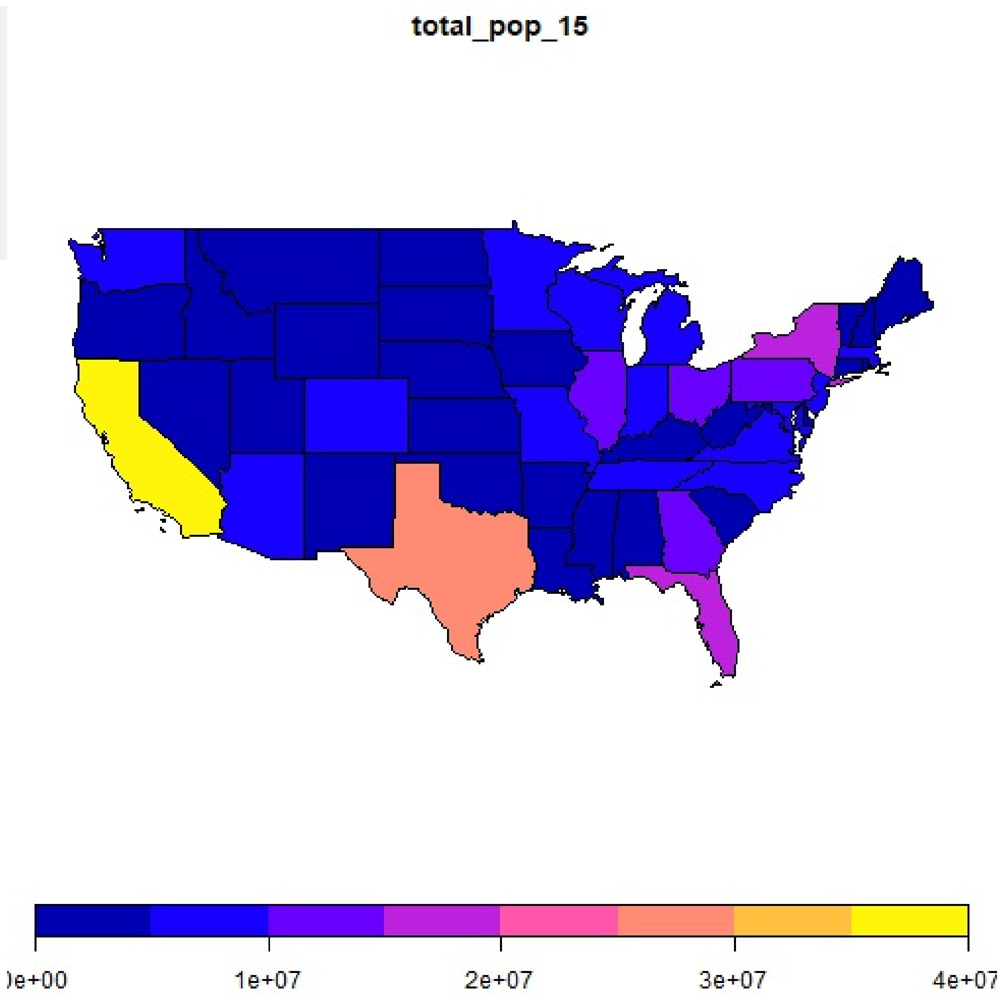


Spatial aggregation (dissolving polygons)

```
library(spData)
us_states %>%
  select(total_pop_15) %>%
  plot
```

Spatial aggregation (dissolving polygons)

```
library(spData)
us_states %>%
  select(total_pop_15) %>%
  plot
```

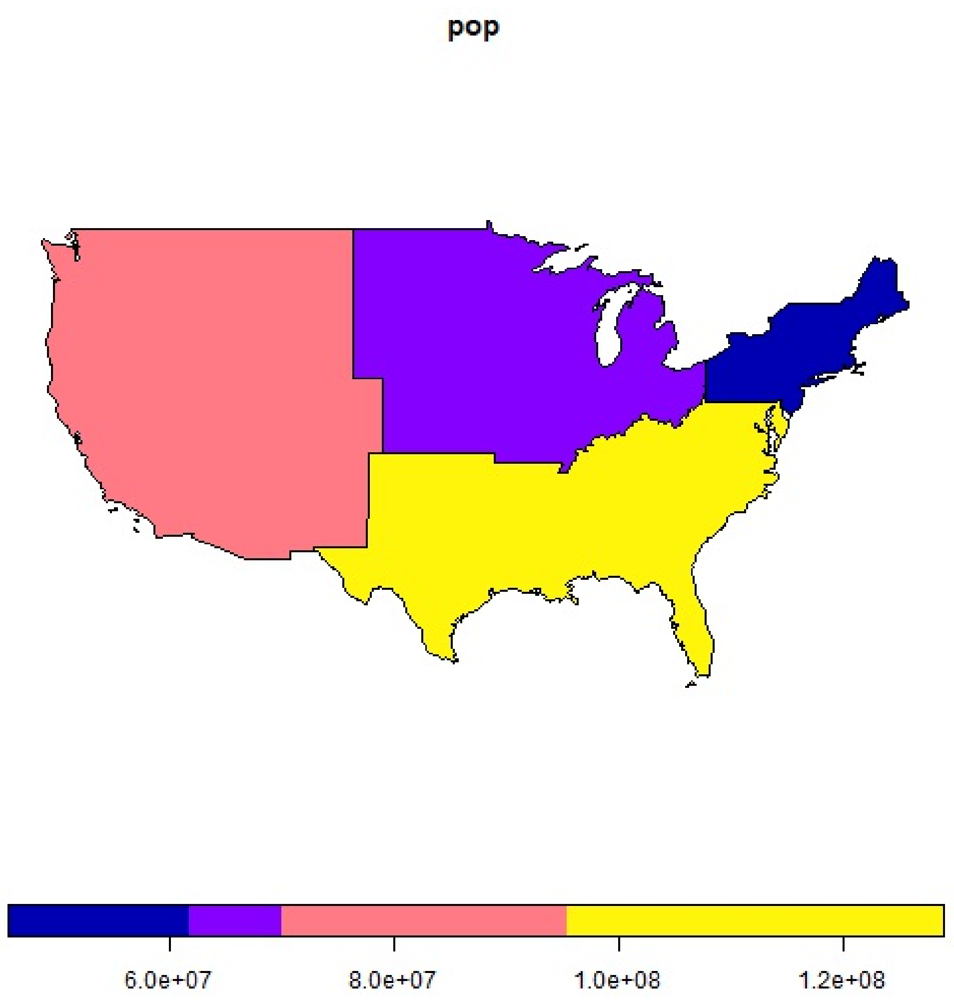


Spatial aggregation (dissolving polygons)

```
regions = us_states %>%
  group_by(REGION) %>%
  summarize(pop = sum(total_pop_15,
                      na.rm = TRUE))
regions %>%
  select(pop) %>%
  plot
```

Spatial aggregation (dissolving polygons)

```
regions = us_states %>%
  group_by(REGION) %>%
  summarize(pop = sum(total_pop_15,
                      na.rm = TRUE))
regions %>%
  select(pop) %>%
  plot
```



CRS in sf

sf lets you use CRS and change CRS (reproject) through Proj.4.

CRS in sf

sf lets you use CRS and change CRS (reproject) through Proj.4.

```
st_crs(4326)
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

CRS in sf

Find out about a projection of a spatial object:

```
st_crs(us_states)
```

```
## Coordinate Reference System:
##   EPSG: 4269
##   proj4string: "+proj=longlat +datum=NAD83 +no_defs"
```

CRS in sf

Find out about a projection of a spatial object:

```
st_crs(us_states)
```

```
## Coordinate Reference System:
##   EPSG: 4269
##   proj4string: "+proj=longlat +datum=NAD83 +no_defs"
```

Change the CRS with the help of `st_transform()`:

```
st_transform(us_states, crs = 4326)
```

Further reading

Geometric operations on vector data

Your turn

- Create two overlapping circles (see below) and compute and plot their geometric intersection. Secondly union the circles.

```
pts = st_sfc(st_point(c(0, 1)), st_point(c(1, 1))) # create 2 point
# use the buffer function to create circles from points
circles = st_buffer(pts, dist = 1)
x = circles[1, ]
y = circles[2, ]
```

- Compute the average population (`total_pop_15`) for each `REGION` of `us_states`. Plot your result.
- Find out about the CRS of `nz`, reproject it into a geographic CRS (EPSG: 4326) and plot the original `nz` object next to your transformed `nz` object.

Recap

We have learned how to perform with `sf`-objects:

- Attribute operations
- Spatial attribute operations
- Geometric operations