Using the statistical language R as a Geographic Information System

Data Processing & Spatial Linking

Stefan Jünger / GESIS – Leibniz Institute for the Social Sciences

November 23, 2021

DOI: 10.5281/zenodo.5717830

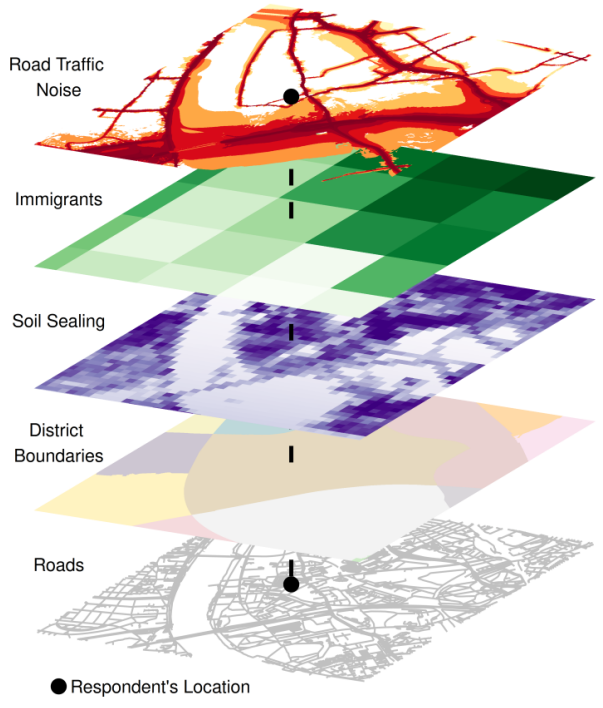
cessda.eu @CESSDA_Data

Now

Time	Title
09:00-09:30	Introduction: Data Management & Geospatial Data
09:30-09:35	Exercise 1: R Warm up
09:35-10:00	Data Processing & Spatial Linking
10:00-10:30	Exercise 2: Geospatial Data Wrangling
10:30-10:45	Break
10:45-11:15	Easy Maps
11:15-11:45	Excercise 3: Build your own map
11:45-12:00	Closing, Q & A

Why Care About Data Types and Formats?

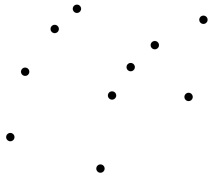
- Different commands to import and handle the data
- Spatial linking techniques and analyses partially determined by the data format
- Visualization of data can differ



Data Sources: OpenStreetMap / GEOFABRIK (2018), City of Cologne (2014), Leibniz Institute of Ecological Urban and Regional Development (2018), Statistical Offices of the Federation and the Länder (2016), and German Environmental Agency / EIONET Central Data Repository (2016)

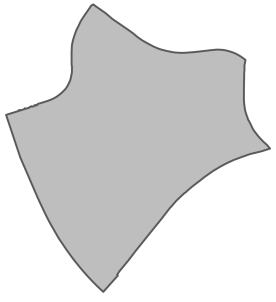
Visual Difference of Vector and Raster Data

(a) Points
e.g., addresses

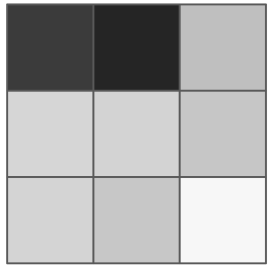


(b) Lines
e.g., roads

(c) Polygon
e.g., boundaries



(d) Grids
e.g., uniform areas



More Formally, What Are...

...Vector Data?

- Each real-world feature is represented as one of three types of geometries:
  - Points: discrete location (f.e., city)
  - Lines: linear feature (f.e., river)
  - Polygons: enclosed area (f.e., country)
- Geometries are not fixed

...Raster Data?

- Hold information on (most of the time) evenly shaped grid cells
- Basically, a simple data table
- Each cell represents one observation

Vector Data's Main Characteristic

Geometries' 3 types of information:

- location (points, lines, and polygons)
- length (lines and polygons)
- area (polygons)

We need to assign attributes to each geometry:

- attribute tables (i.e., data tables)
- rows = geometric object (i.e., observation, case, feature)
- column = attribute (i.e., variable)

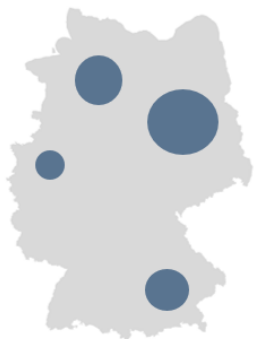
Longitude	Latitude
13.404954	52.520008
6.961899	50.933594
11.576124	48.137154
9.993682	53.551086
….	……

+

City Name	Population
Berlin	3,669,491
Cologne	1,087,863
Munich	1,484,226
Hamburg	1,847,253
…	……

Geometric Information
(Point Coordinates)

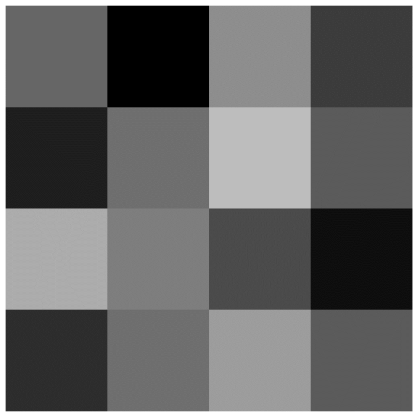
Attribute Table



Raster Data's Main Characteristic

- Information about geometries are globally stored
- Their location in space is defined by their cell location in the data table
- Information about raster cells' resolution are stored as metadata
- Usually, rasters only comprise one attribute
- One raster layer can comprise several bands though (e.g., rgb-bands)

82	923	36	99
784	73	1	83
7	72	87	876
234	73	35	83



File Formats

Vector Data

- Shapefile (.shp & Co)
- .geojson
- ...
- .kml
- .gml

Raster Data

- Gtiff/GeoTiff
- JPEG2000
- ...
- .grd
- netCDF

Importing Data: We Only Need Two Commands

Vector data

```
sf::st_read()
```

Raster data

```
stars::read_stars()
```

Reading Vector Data in Practise: COVID-19 Cases in Cologne

```
corona_cologne <-
  sf::read_sf(
    "./data/corona_cologne.shp"
  )

corona_cologne
```

```
## Simple feature collection with 86 features and 8 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 6.77253 ymin: 50.83045 xmax: 7.162028 ymax: 51.08496
## Geodetic CRS:  WGS 84
## # A tibble: 86 x 9
##       id name        anzhl_7 a_7_100 anzhl_g anzhl_k einwhnr stand
##    <int> <chr>         <int>   <dbl>   <int>   <int>   <int> <chr>                                                                        <MUL
##  1     1 Marienburg       15    207.     407      21    7258 17.11.2021 (((6.971316 50.90478, 6.971546 50.90483, 6.971546 50
##  2     2 Bayenthal        19    182.     619      39   10426 17.11.2021 (((6.973431 50.91707, 6.975959 50.91287, 6.979068 50
##  3     3 Weiß             17    288.     212      21    5907 17.11.2021 (((7.046571 50.89097, 7.047746 50.89014, 7.048239 50
##  4     4 Hahnwald          3    145.      93       7    2066 17.11.2021 (((6.969714 50.88297, 6.970253 50.88073, 6.970377 50
##  5     5 Meschenich       12    150.     786      14    8024 17.11.2021 (((6.925416 50.87073, 6.927486 50.86942, 6.927528 50
##  6     6 Immendorf         2     97       99       2    2062 17.11.2021 (((6.955817 50.86776, 6.955897 50.86773, 6.955938 50
##  7     7 Godorf           10    366.     242      11    2729 17.11.2021 (((6.994359 50.85836, 6.994369 50.85836, 6.994383 50
##  8     8 Lövenich         30    327.     368      50    9186 17.11.2021 (((6.835101 50.95726, 6.83883 50.95419, 6.842222 50.
```

The geometry Column

```
corona_cologne["geometry"]
```

```
## Simple feature collection with 86 features and 0 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 6.77253 ymin: 50.83045 xmax: 7.162028 ymax: 51.08496
## Geodetic CRS:  WGS 84
## # A tibble: 86 x 1
##                                                                    geometry
##                                                          <MULTIPOLYGON [°]>
##  1 (((6.971316 50.90478, 6.971546 50.90483, 6.971546 50.90483, 6.971546 50.90483, 6.971...
##  2 (((6.973431 50.91707, 6.975959 50.91287, 6.979068 50.90879, 6.979459 50.9083, 6.9797...
##  3 (((7.046571 50.89097, 7.047746 50.89014, 7.048239 50.88973, 7.048942 50.88909, 7.049...
##  4 (((6.969714 50.88297, 6.970253 50.88073, 6.970377 50.88074, 6.970513 50.8806, 6.9708...
##  5 (((6.925416 50.87073, 6.927486 50.86942, 6.927528 50.86939, 6.92764 50.86935, 6.9285...
##  6 (((6.955817 50.86776, 6.955897 50.86773, 6.955938 50.86775, 6.958341 50.86696, 6.960...
##  7 (((6.994359 50.85836, 6.994369 50.85836, 6.994383 50.85835, 6.994429 50.85834, 6.994...
##  8 (((6.835101 50.95726, 6.83883 50.95419, 6.842222 50.95549, 6.842611 50.9551, 6.84289...
##  9 (((6.849502 50.9422, 6.84951 50.94038, 6.849356 50.93803, 6.849289 50.93654, 6.84924...
## 10 (((6.854198 50.94052, 6.854394 50.94052, 6.854395 50.94055, 6.854396 50.94056, 6.854...
## # ... with 76 more rows
```

Reading Vector Data in Practise: Hospitals in Cologne

```
hospitals_cologne <-
  sf::read_sf(
    "./data/hospitals_cologne.shp"
  )

hospitals_cologne
```

```
## Simple feature collection with 35 features and 10 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 6.897622 ymin: 50.88123 xmax: 7.056884 ymax: 50.99195
## Geodetic CRS:  WGS 84
## # A tibble: 35 x 11
##    objectd objktnm  name                    nutzung     adrss_n     adresse        stdtbzr  stadttl stdtvrt  pstzstl
##      <int> <chr>    <chr>                   <chr>       <chr>       <chr>          <chr>    <chr>   <chr>    <chr>
##  1       1 11510005 Eduardus-Krankenhaus    Krankenhaus 00663000300 Custodisstr. 3 Innenst~ Deutz   Germane~ 50679   (6.98
##  2       2 11510016 Ev. Krankenhaus Weyertal Krankenhaus 03310007600 Weyertal 76    Lindent~ Linden~ Uni-Vie~ 50931   (6.92
##  3       3 11510028 MediaPark Klinik        Krankenhaus 03724000300 Im Mediapark 3 Innenst~ Neusta~ Media-P~ 50670    (6.9
##  4       4 11510013 Krankenhaus Merheim     Krankenhaus 02436020000 Ostmerheimer ~ Kalk     Merheim GE Merh~ 51109   (7.05
##  5       5 11510022 Klinik am Ring          Krankenhaus 01472002800 Hohenstaufenr~ Innenst~ Neusta~ Student~ 50674   (6.94
##  6       6 11510031 RehaNova                Krankenhaus 02436020000 Ostmerheimer ~ Kalk     Merheim GE Merh~ 51109   (7.05
##  7       8 11510009 Universitätskliniken    Krankenhaus 01675000900 Joseph-Stelzm~ Lindent~ Linden~ Uni-Vie~ 50931   (6.91
##  8       9 11510018 Krankenhaus Porz am Rhein Krankenhaus 05620001900 Urbacher Weg ~ Porz     Porz    GE Porz  51149   (7.05
```

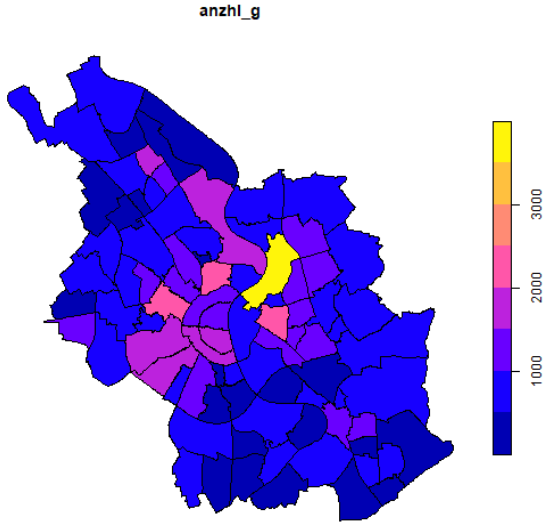
The geometry Column

```
hospitals_cologne["geometry"]
```

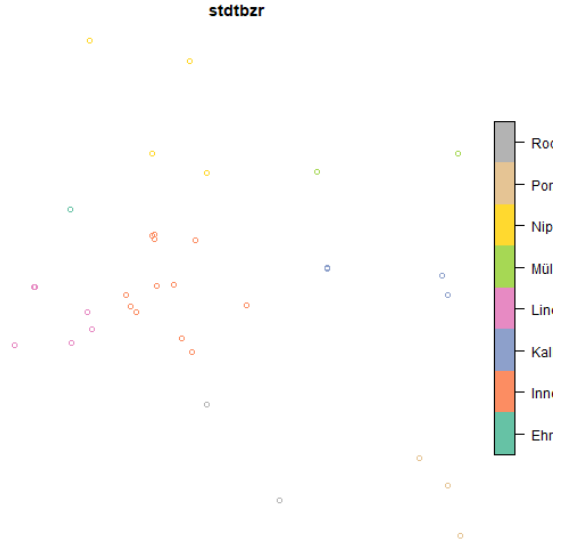
```
## Simple feature collection with 35 features and 0 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 6.897622 ymin: 50.88123 xmax: 7.056884 ymax: 50.99195
## Geodetic CRS:  WGS 84
## # A tibble: 35 x 1
##                geometry
##             <POINT [°]>
##  1 (6.980327 50.93277)
##  2 (6.925405 50.92739)
##  3  (6.94752 50.94856)
##  4 (7.050221 50.93935)
##  5 (6.941231 50.93114)
##  6 (7.050371 50.93951)
##  7 (6.918107 50.92434)
##  8 (7.052081 50.89247)
##  9   (6.948554 50.937)
## 10 (7.042189 50.89845)
## # ... with 25 more rows
```

How Do I Know They Are Vectors? First Check: Plot Layers With `plot()`!

```
plot(corona_cologne["anzhl_g"])
```

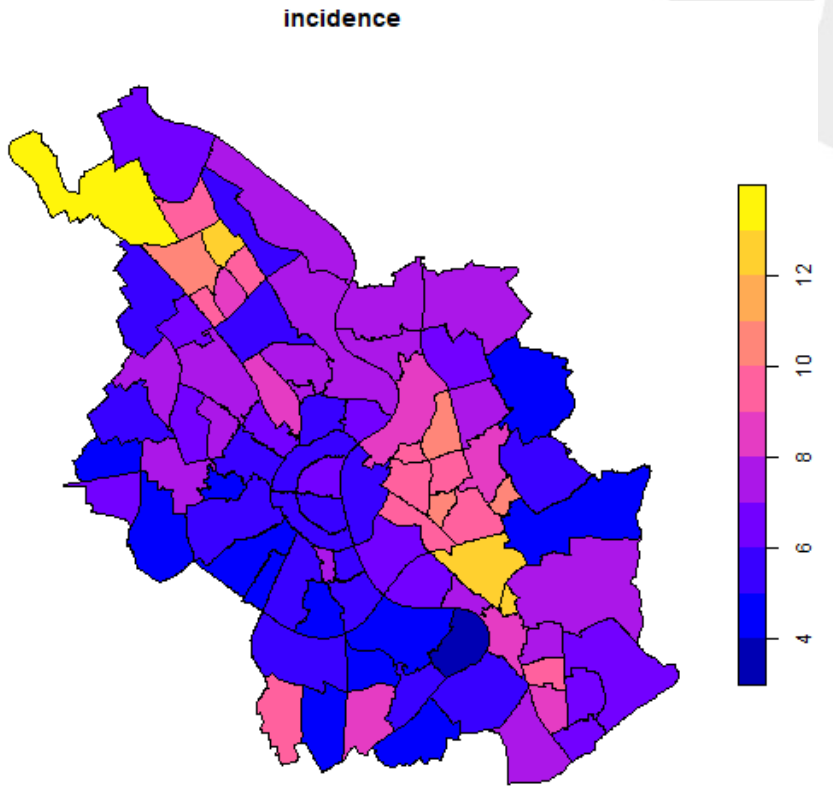



```
plot(hospitals_cologne["stdtbzr"])
```



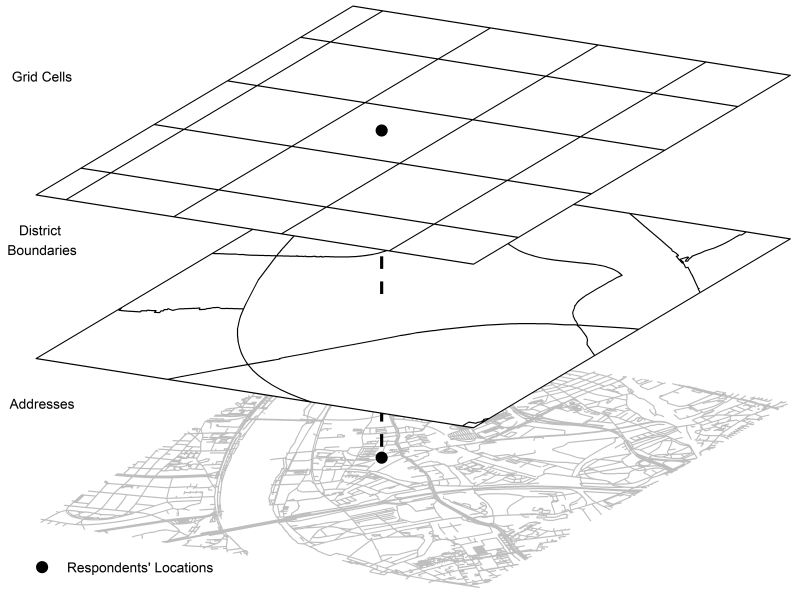
Creating New Variables

```
corona_cologne <-
  corona_cologne %>%
  dplyr::mutate(
    incidence = (anzhl_g / einwhnr) * 100
  )
```



Geometric Operations and Spatial Linking

The main advantage of geospatial data is the flexibility of their geometric properties. They make spatial linking of geospatial data straightforward.



Linking Hospitals to Our Corona Data

How many hospitals are located in each of Cologne's districts? How would we do that?

- We project the corona and hospital data in one space
- We extract the hospitals' information and transfer it to the corona data by counting how many of them fall into the corona datas' polygons

(...but first, we first have to check whether our data are valid.

```
corona_cologne <-
  sf::st_make_valid(corona_cologne)

hospitals_cologne <-
  sf::st_make_valid(hospitals_cologne)
```

)

Detect Containing Geometries

```
containing <-
  sf::st_contains(
    corona_cologne,
    hospitals_cologne
  )
```

```
## Sparse geometry binary predicate list of length 86, where the p
## first 23 elements:
##  1: (empty)
##  2: 11
##  3: (empty)
##  4: (empty)
##  5: (empty)
##  6: (empty)
##  7: (empty)
##  8: (empty)
##  9: (empty)
##  10: (empty)
##  11: (empty)
##  12: (empty)
##  13: (empty)
##  14: (empty)
##  15: (empty)
##  16: (empty)
##  17: (empty)
##  18: (empty)
##  19: (empty)
##  20: (empty)
##  21: 28
##  22: (empty)
##  23: 4, 6, 29
```

Counting Containing Geometries

```
containing_count <-
  lengths(containing)

containing_count
```

```
##  [1] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 3 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 2 3 2 3 2 0 0 3 1 0 0 1 0 1
## [68] 0 0 0 0 0 0 0 0 0 0 1 1 4 0 0 0 1 0 1 0
```

We then can add this information as a new variable:

```
corona_cologne <-
  corona_cologne %>%
  dplyr::mutate(
    hospitals_count = containing_count
  )
```

Incidence vs. Hospital Infrastructure

```
plot(
  corona_cologne[
    c("incidence", "hospitals_count")
  ]
)
```

There Are More Geometric Confirmation Methods

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_covered_by(x, y, ...) Identifies if x is completely within y (i.e. polygon completely within polygon)

st_covers(x, y, ...) Identifies if any point from x is outside of y (i.e. polygon outside polygon)

st_crosses(x, y, ...) Identifies if any geometry of x have commonalities with y

st_disjoint(x, y, ...) Identifies when geometries from x do not share space with y

st_equals(x, y, ...) Identifies if x and y share the same geometry

st_intersects(x, y, ...) Identifies if x and y geometry share any space

st_overlaps(x, y, ...) Identifies if geometries of x and y share space, are of the same dimension, but are not completely contained by each other

st_touches(x, y, ...) Identifies if geometries of x and y share a common point but their interiors do not intersect

st_within(x, y, ...) Identifies if x is in a specified distance to y

https://github.com/rstudio/cheatsheets/blob/master/sf.pdf

Turning to Raster Data: Information About Immigrant Shares

Covid-19 is a disease disproportionaly affecting social groups.

For example, members of minority groups may be at a higher risk to get infected.

We are going to try to 'corroborate' this finding with German Census 2011 *raster* data.

However: This is by no means a serious research assessment!

Loading Raster Data: Immigrants From the German Census 2011

```
immigrants_cologne <-
  stars::read_stars("./data/immigrants_cologne.tif")

immigrants_cologne
```

```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##                         Min. 1st Qu. Median    Mean 3rd Qu. Max.  NA's
## immigrants_cologne.tif     3       3      7 15.1337      17  639 62429
## dimension(s):
##   from  to  offset delta                       refsys point values x/y
## x    1 264 4094850   100 ETRS89-extended / LAEA Eu... FALSE   NULL [x]
## y    1 289 3112950  -100 ETRS89-extended / LAEA Eu... FALSE   NULL [y]
```

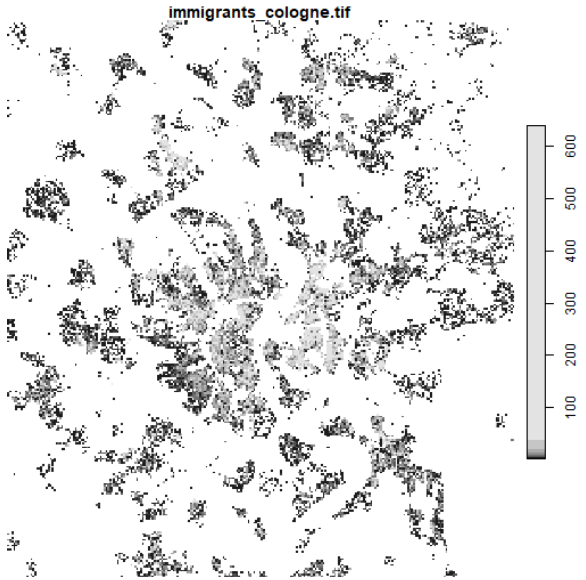
Loading Raster Data: Inhabitants From the German Census 2011

```
inhabitants_cologne <-
  stars::read_stars("./data/inhabitants_cologne.tif")

inhabitants_cologne
```

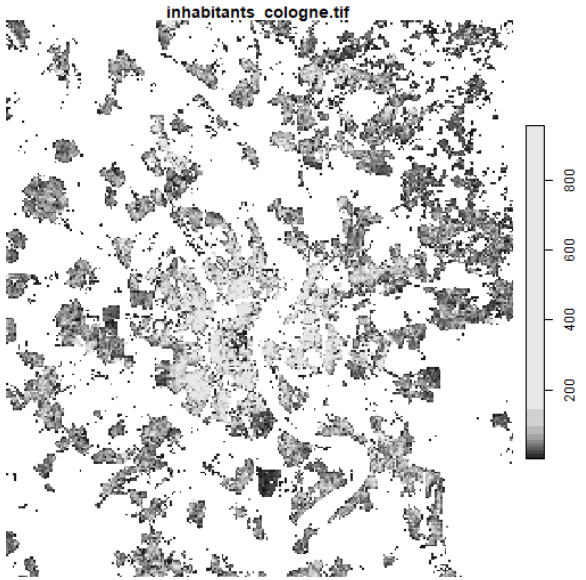
```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##                          Min. 1st Qu. Median     Mean 3rd Qu. Max.  NA's
## inhabitants_cologne.tif     3      20     44 62.53469      81  956 52731
## dimension(s):
##   from  to  offset delta                       refsys point values x/y
## x    1 264 4094850   100 ETRS89-extended / LAEA Eu... FALSE   NULL [x]
## y    1 289 3112950  -100 ETRS89-extended / LAEA Eu... FALSE   NULL [y]
```

Compare Layers by Plotting

```
plot(immigrants_cologne)
```




```
plot(inhabitants_cologne)
```



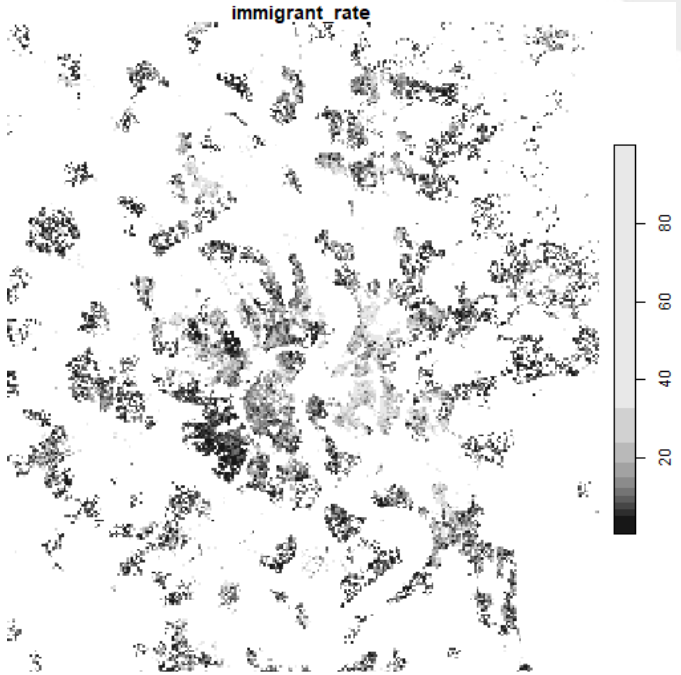
Combining Raster Layers to Calculate New Values: Immigrant Rates

```
immigrant_rate <- immigrants_cologne * 100 / inhabitants_cologne

names(immigrant_rate) <- "immigrant_rate"

immigrant_rate
```

```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##                      Min. 1st Qu.   Median     Mean  3rd Qu. Max.  NA's
## immigrant_rate  0.6637168     7.5 12.32877 16.48389 21.05263  100 62429
## dimension(s):
##   from  to  offset delta                       refsys point values x/y
## x    1 264 4094850   100 ETRS89-extended / LAEA Eu... FALSE   NULL [x]
## y    1 289 3112950  -100 ETRS89-extended / LAEA Eu... FALSE   NULL [y]
```

Result of Combination

```
plot(immigrant_rate)
```



Raster Extraction

To extract the raster values at a specific point by location we can use:

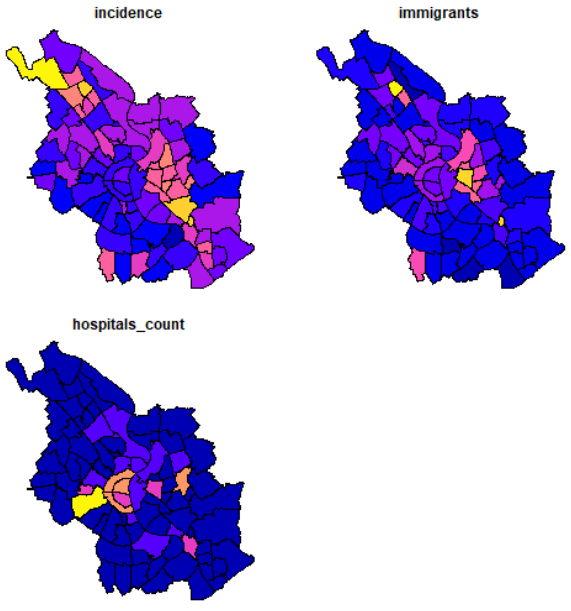
```
extracted_immigrants <-
  aggregate(
    x =
      immigrants_cologne %>%
      stars::st_transform_proj(4326),
    by = corona_cologne,
    FUN = mean,
    na.rm = TRUE
  )

extracted_immigrants
```

```
## stars object with 1 dimensions and 1 attribute
## attribute(s):
##                            Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## immigrants_cologne.tif  3.909091 8.572482 12.46615 17.73039 23.70045 69.04762
## dimension(s):
##          from to offset delta refsys point                                                                values
## geometry    1 86     NA    NA WGS 84 FALSE MULTIPOLYGON (((6.971316 50...,...,MULTIPOLYGON (((7.064916 50...
```

Add Results to Existing Dataset

This information can simply be added to an existing dataset:

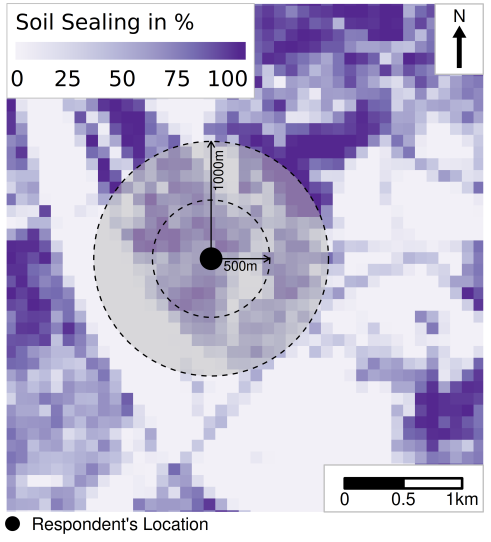
```
corona_cologne <-
  corona_cologne %>%
  dplyr::mutate(
    immigrants = extracted_immigrants[[1]]
  )

plot(
  corona_cologne[
    c(
      "incidence",
      "immigrants",
      "hospitals_count"
    )
  ]
)
```



What's Next, What's Missing?

When working with geospatial data, researcher's degree of freedom is huge

- we can vary the scale and zonation
- permutation is crazy
- do it theory-driven!



The analysis of geospatial data can get quite sophisticated

- in the end, it's about modeling and testing spatial dependence
- you may want to have a look at Tobias Rüttenauer's fantastic materials from his recent workshop to get a first idea


SO, IS THERE STILL MUCH TO LEARN ABOUT R AND GIS?
YES
BUT YOU'LL LEARN AS YOU PRACTICE (IN THE EXERCISES)
imgflip.com

Exercise 2: Geospatial Data Wrangling

Exercise

Solution

stefan.juenger@gesis.org
@StefanJuenger
StefanJuenger
https://stefanjuenger.github.io

cessda.eu

@CESSDA_Data