Introduction to Applied Data Science

Lecture 6: Spatial & Network Data

Bas Machielsen
Utrecht University
2024-06-04

Lecture 6: Spatial and Network Data

Lecture 6: Spatial and Network Data

- Overview of this class:
  - Introduction to Data Science
  - Introduction to R & Programming
  - Getting Data: API’s and Databases
  - Getting Data: Web Scraping
  - Transforming and Cleaning Data
  - **This lecture**: Spatial & Network Data
  - Text as Data and Mining
  - Data Science Project

Introduction

Introduction

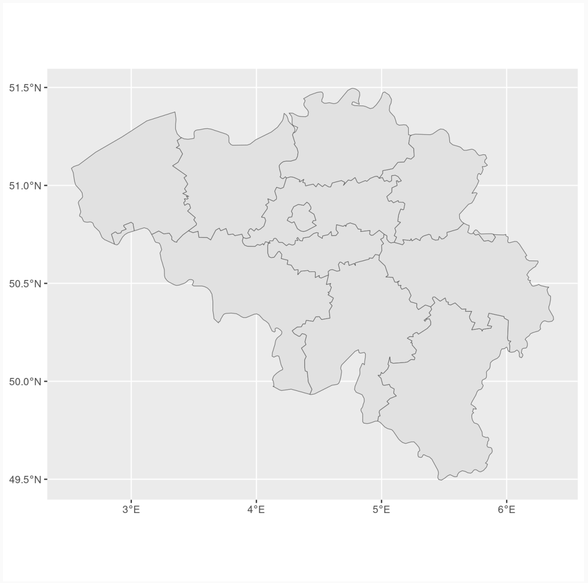
- In this lecture, we will get to know two new types of data, **spatial data** and **network data**
- **Spatial data** combines the features of data we already know with spacial features
    - Spatial data is getting more and more important, so we need tools to work with it
    - Depending on the specific spatial format, this can represent real-life geographical objects such as countries, roads, seas, or other geographical entities
    - It can also represent more abstract objects depicting distances between each other
- **Network data** describes relationships among units rather than units in isolation.
    - Examples include friendship networks among people, citation networks among academic articles, and trade and alliance networks among countries.
    - Network data is different from traditional data in that the unit of analysis is a relationship between two *nodes*

Spatial Data

Spatial Data

- As mentioned, the main example of spatial data we'll be dealing with are **maps**
- Spatial data usually does not only come with "shape" attributes, but also with *coordinates*, so the attributes can be put in relation to everything else
- For example:

```r
library(rnaturalearth); library(sf)
be ← ne_states(country="Belgium",
                returnclass = "sf")

ggplot(be) + geom_sf()
```



Spatial Data

- This is what the object `be` looks like:

```r
class(be)
```

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

```r
nrow(be)
```

```
## [1] 11
```

```r
ncol(be)
```

```
## [1] 122
```

- As you can see, `be` is a `data.frame`! It has 11 rows and 122 columns
- Each row corresponds to a particular province of Belgium, and each column to particular information about that province

Spatial Data Frame

- For example, `name_fr` contains the name (in French) of each province

```
be ▷
  select(name_fr)
```

```
## Simple feature collection with 11 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.5218 ymin: 49.49522 xmax: 6.374525 ymax: 51.49624
## Geodetic CRS:  WGS 84
## First 10 features:
##                 name_fr                      geometry
## 201  Flandre-Occidentale MULTIPOLYGON (((2.866726 50 ...
## 202              Hainaut MULTIPOLYGON (((3.023817 50 ...
## 204                Namur MULTIPOLYGON (((4.968371 49 ...
## 206           Luxembourg MULTIPOLYGON (((5.391166 49 ...
## 209                Liège MULTIPOLYGON (((6.117487 50 ...
## 761    Flandre-Orientale MULTIPOLYGON (((3.398798 51 ...
## 763             d'Anvers MULTIPOLYGON (((4.281124 51 ...
## 764             Limbourg MULTIPOLYGON (((5.551407 51 ...
## 1710  Bruxelles-Capitale MULTIPOLYGON (((4.479746 50 ...
## 1711     Brabant flamand MULTIPOLYGON (((4.099739 50 ...
```

Spatial Data Features

- But in addition to being a `data.frame`, `be` also has class `sf`, short for **spatial features**

```
st_geometry(be)
```

```
## Geometry set for 11 features
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.5218 ymin: 49.49522 xmax: 6.374525 ymax: 51.49624
## Geodetic CRS:  WGS 84
## First 5 geometries:
```

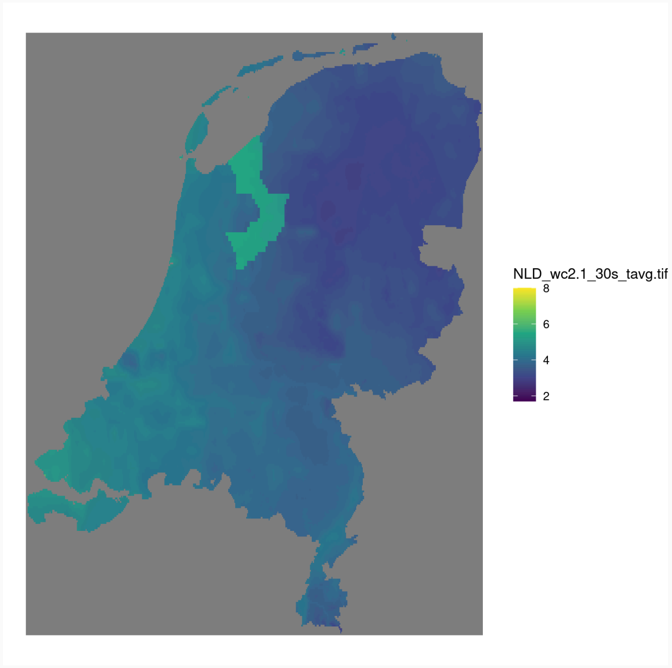
- This dataset contains 11 features, corresponding to each of these provinces
- Each of these provinces is represented as a *polygon* and is geocoded
- The **bounding box** of `be` is represented by coordinates in a certain *coordinate system*
- In this case, the **C**oordinate **R**eference **S**ystem is called WGS84
  - There exist different coordinate systems: we'll talk about this more later

Kinds of Spatial Data

- In general, there exist roughly two types of spatial data:
    - **Vector data**: represents things with *points, lines and polygons*. Vector data can scale and stretch and transform those easily with mathematical operations. Can increase precision to arbitrary levels (can always zoom in futher). It allows allows us to ask questions about statial relations, such as what is the area of an object, what is the distance from one object to another, or which objects border other objects.
    - **Raster data**: fixed-size tiles or cells (like a mosaic, or like pixels), which form a grid. Fixed resolution. Raster data is like an image with geo-coded pixels. Satellite images, for example, are usually released in the form of raster data. Other examples of raster data include population density images, species occurence data, and meteorological data.
- As you might have guessed, `be` is an example of **vector data**.

Example Raster Data

- Temperature data is an example of raster data:
    - Each "pixel" is geocoded and has a particular **value** for a particular variable, in this case, a temperature
    - Each geocoded value represents a particular temperature as measured (inferred) in a particular area marked by the "pixel"
    - Raster data can differ in terms of its granularity, i.e. how detailed the data is



CRS

- We all agree the earth is round (hopefully)
- However, you need coordinates to describe *where is what*

- Usually, this is based on a three-dimensional model of the earth

    - Coordinates are given in latitude and longitude.
    - An example CRS is EPSG:4326 (also known WGS 84).

- There also exist CRS that are "projected":

    - Transforms the earth’s curved surface onto a flat surface.
    - This advantage is that coordinates are given in linear units (e.g., meters).
    - Disadvantage: distorts surface
    - Example: Mercator projection, EPSG:3857

CRS in R

- Setting a CRS:

```r
library(sf)
point ← st_point(c(4.8897, 52.3740))  # Amsterdam coordinates
sf_point ← st_sfc(point, crs = 4326)
```

- Checking a CRS:

```r
st_crs(sf_point)[1] # Remove the [1] to see full output
```

```
## $input
## [1] "EPSG:4326"
```

- *Changing* a CRS from one to another:

```r
sf_point_transformed ← st_transform(sf_point, 3857)
```

Vector Data

Working With Vector Data: Select

- You can work with vector data using the `tidyverse` library
- For example, data can be selected:

```r
library(rnaturalearth)
nl ← ne_states("Netherlands", returnclass="sf")
nl_short ← nl ▷
  select(name, latitude, longitude)

nl_short ▷ head(4)
```

```
## Simple feature collection with 4 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 5.00448 ymin: 51.73483 xmax: 7.198506 ymax: 53.55809
## Geodetic CRS:  WGS 84
##           name latitude longitude                       geometry
## 408  Groningen  53.2790   6.73067 MULTIPOLYGON (((7.194591 53 ...
## 410    Drenthe  52.9046   6.60064 MULTIPOLYGON (((7.072151 52 ...
## 411 Overijssel  52.4311   6.41649 MULTIPOLYGON (((6.719083 52 ...
## 413 Gelderland  52.0635   5.96001 MULTIPOLYGON (((6.771576 52 ...
```

Working With Vector Data: Filter

- Or data can be filtered:

```r
nl_short |>
  filter(name == "Utrecht")
```

```
## Simple feature collection with 1 feature and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 4.767854 ymin: 51.94117 xmax: 5.61974 ymax: 52.28549
## Geodetic CRS:  WGS 84
##      name latitude longitude                       geometry
## 1 Utrecht  52.0749    5.1938 MULTIPOLYGON (((5.408922 52 ...
```

Working With Vector Data: Mutate

- Or new variables can be added
    - There are some special functions in the `sf` package that calculate distances and areas: `st_distance()` and `st_area()`

```
nl_short ▷
  mutate(area = st_area(geometry)) ▷
  select(name, area) ▷
  head(4)
```

```
## Simple feature collection with 4 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 5.00448 ymin: 51.73483 xmax: 7.198506 ymax: 53.55809
## Geodetic CRS:  WGS 84
##           name             area                       geometry
## 408  Groningen 2386242755 [m^2] MULTIPOLYGON (((7.194591 53 ...
## 410    Drenthe 2626372215 [m^2] MULTIPOLYGON (((7.072151 52 ...
## 411 Overijssel 3331334171 [m^2] MULTIPOLYGON (((6.719083 52 ...
## 413 Gelderland 5113884777 [m^2] MULTIPOLYGON (((6.771576 52 ...
```

Working With Vector Data: Join

- You can also **join** a spatial data.frame with another data.frame using the `_join()` functions
- I download labor market participation data using the `cbsodataR` package
    - I select only provinces and focus on the year 2022
    - I use the `clean_names()` function from the `janitor` package to clean variable names, and do a minor clean-up of a variable:

```
library(cbsodataR); library(janitor)
labor ← cbs_get_data("85268NED",
                     Perioden = "2022JJ00",
                     RegioS = has_substring("PV")) ▷
  clean_names() ▷
  mutate(regio_s = str_trim(regio_s))
```

```
labor ▷ head(2)
```

```
## # A tibble: 2 × 18
##   regio_s perioden beroeps_en_niet_beroepsbevolking_1 beroepsbevolking_2 werkzame_be
##   <chr>   <chr>                                 <int>              <int>
## 1 PV20    2022JJ00                                450                330
## 2 PV21    2022JJ00                                481                357
## # ℹ 11 more variables: positie_in_de_werkkring_onbekend_6 <int>, beroepsniveau1_7 <i
```

Working With Vector Data: Join

- These data should be **merged** with a spatial data.frame
- I also download the Dutch provinces using the `cbs_get_sf()` function
    - I am using this because I get the same *identifier* as in the `labor` data.frame:

```
provincies ← cbs_get_sf("provincie", year = 2022)
provincies ▷ head(5)
```

```
## Simple feature collection with 5 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 118774 ymin: 459728 xmax: 277529 ymax: 619172
## Projected CRS: Amersfoort / RD New
## # A tibble: 5 × 3
##   statcode statnaam
##   <chr>    <chr>
## 1 PV20     Groningen  (((269919 540356, 268516 541104, 266297 544126, 264580 544106,
## 2 PV21     Fryslân    (((139834 589987, 138042 588615, 137661 590369, 139834 589987)
## 3 PV22     Drenthe    (((269919 540356, 268563 537232, 268290 527517, 267857 518738,
## 4 PV23     Overijssel (((244564 516409, 245758 514996, 245235 512060, 248533 509328,
## 5 PV24     Flevoland  (((182511 535552, 184106 533460, 186243 533127, 188252 531868,
```

Working With Vector Data: Join

- Now, these two data.frames can be merged:

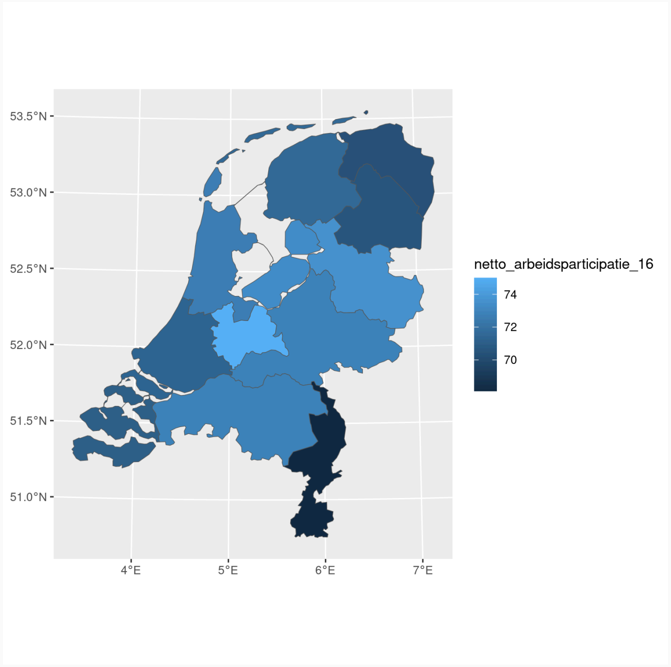
```r
prov_labor <- provincies |>
  left_join(labor, by = c('statcode' = 'regio_s')) |>
  select(statcode, statnaam, netto_arbeidsparticipatie_16)

prov_labor |> head(5)
```

```
## Simple feature collection with 5 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 118774 ymin: 459728 xmax: 277529 ymax: 619172
## Projected CRS: Amersfoort / RD New
## # A tibble: 5 × 4
##   statcode statnaam   netto_arbeidsparticipatie_16
##   <chr>    <chr>                             <dbl>
## 1 PV20     Groningen                          70.4 (((269919 540356, 268516 541104,
## 2 PV21     Fryslân                            71.6 (((139834 589987, 138042 588615,
## 3 PV22     Drenthe                            70.6 (((269919 540356, 268563 537232,
## 4 PV23     Overijssel                         73.5 (((244564 516409, 245758 514996,
## 5 PV24     Flevoland                          73.4 (((182511 535552, 184106 533460,
```

Working With Vector Data: Plot

- Finally, you can **plot** the data using a few simple commands (details are for a later course):

```
prov_labor ▷
  ggplot(aes(fill = netto_arbeidsparticipatie_16)) + geom_sf()
```



Working With Vector Data: Spatial

- In addition to working with spatial `sf` data.frames using the tidyverse library, there are also a couple of specific operations to spatial data
- Spatial operations, including spatial joins between vector datasets and local and several operations on raster datasets, are a vital part of geocomputation.
    - For more resources on this, see the Geocomputation with R Ebook
- We'll demonstrate a couple of spatial operations including:
    - **Spatial subsetting**
    - **Topological relations**
    - **Distance relations**
- In addition, there exists two other, more advanced subjects for study in other courses:
    - **Spatial joining**
    - **Spatial aggregations**

Working With Vector Data: Schools

- Spatial subsetting is the process of taking a spatial object and returning a new object containing only features that relate in space to another object.
  - For example, here's a dataset containing all schools in the Netherlands:

```
schools ← read_csv2('https://duo.nl/open_onderwijsdata/images/01•-hoofdvestiginge
  mutate(POSTCODE = str_remove(POSTCODE, " "))
schools ▷ head(3)
```

```
## # A tibble: 3 × 29
##   PROVINCIE `BEVOEGD GEZAG NUMMER` INSTELLINGSCODE INSTELLINGSNAAM           STRAAT
##   <chr>                      <dbl> <chr>           <chr>                     <chr>
## 1 Drenthe                    32073 03WU            Kindcentrum De Wegwijzer  Harm T
## 2 Drenthe                    32073 04LY            Christelijk Kindcentrum D… Molens
## 3 Drenthe                    32073 04TG            Kindcentrum Drijber       Nijenk
## # ℹ abbreviated name: ¹`HUISNUMMER-TOEVOEGING`
## # ℹ 21 more variables: GEMEENTENUMMER <chr>, GEMEENTENAAM <chr>, DENOMINATIE <chr>, `
## #   `STRAATNAAM CORRESPONDENTIEADRES` <chr>, `HUISNUMMER-TOEVOEGING CORRESPONDENTIEA
## #   `POSTCODE CORRESPONDENTIEADRES` <chr>, `PLAATSNAAM CORRESPONDENTIEADRES` <chr>,
## #   `NODAAL GEBIED NAAM` <chr>, `RPA-GEBIED CODE` <chr>, `RPA-GEBIED NAAM` <chr>, `W
## #   `WGR-GEBIED NAAM` <chr>, `COROPGEBIED CODE` <chr>, `COROPGEBIED NAAM` <chr>, `ON
## #   `ONDERWIJSGEBIED NAAM` <chr>, `RMC-REGIO CODE` <chr>, `RMC-REGIO NAAM` <chr>
```

Working With Vector Data: Schools

- And here's a dataset with spatial distributions of postal codes, allowing to link a school to a location
  - Available on the Dutch Geodata Portal

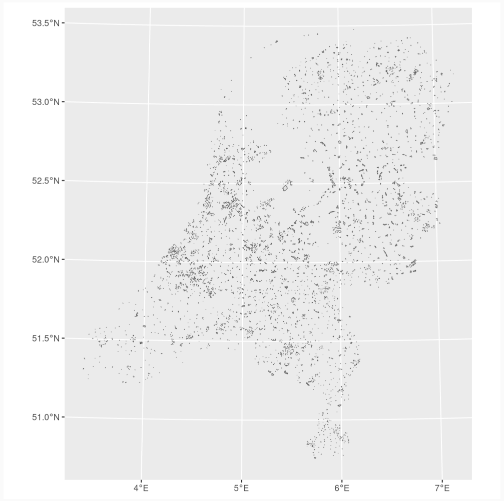
```
#post_codes ← st_read('https://service.pdok.nl/cbs/postcode6/atom/downloads/cbs_p
post_codes ← st_read('./cbs_pc6_2022_v1.gpkg', quiet=TRUE) ▷
  select(postcode)

post_codes ▷
  head(5)
```

```
## Simple feature collection with 5 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 121039.2 ymin: 487028.2 xmax: 122311 ymax: 488296.8
## Projected CRS: Amersfoort / RD New
##   postcode                           geom
## 1   1011DD MULTIPOLYGON (((122311 4877 ...
## 2   1012AC MULTIPOLYGON (((121800.6 48 ...
## 3   1012CJ MULTIPOLYGON (((121600.4 48 ...
## 4   1012MA MULTIPOLYGON (((121551.6 48 ...
## 5   1013GR MULTIPOLYGON (((121112.3 48 ...
```

Working With Vector Data: Schools

- We link them together and display the location of schools in the Netherlands:

```
schools_geocoded ← left_join(schools,
                              post_codes,
                              by=c('POSTCODE'='postcode')) ▷
  st_as_sf()

schools_geocoded ▷
  ggplot() + geom_sf()
```



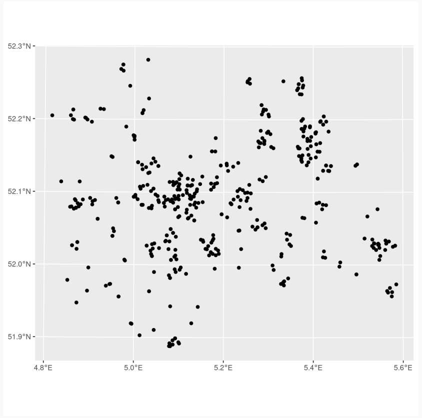
Working With Vector Data: Subsetting

- Then, suppose we're only interested only interested in schools in the Utrecht province:

```r
utrecht <- cbs_get_sf("provincie", year=2022) |>
  filter(statnaam == "Utrecht")
```

- We can now simply use **spatial subsetting** to "filter" our `schools_geocoded` data.frame to look at only observations from the Utrecht province:

```r
schools_geocoded[utrecht, ] |>
  st_centroid() |>
  ggplot() +
  geom_sf()
```



Working With Vector Data: Topology

- **Topological relations** describe the spatial relationships between objects.
- These are logical statements (in that the answer can only be TRUE or FALSE) about the spatial relationships between two objects
- A simple question is: which of the points in `schools_geocoded` intersect in some way with polygon of the city of Utrecht?
- This question can be answered with the spatial predicate `st_intersects()` as follows:

Working With Vector Data: Topology

- `st_intersects()` with the argument `sparse=FALSE` returns a TRUE/FALSE for each datapoint in `schools_geocoded`

```r
# Find Polygon of Utrecht Municipality
gem_utrecht <- cbs_get_sf("gemeente", 2023) |> filter(statnaam == "Utrecht")
schools_geocoded <- schools_geocoded |>
  mutate(is_inside_utrecht_municipality = st_intersects(schools_geocoded,
schools_geocoded |>
  select(is_inside_utrecht_municipality) |> head(3)
```

```
## Simple feature collection with 3 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 232918.9 ymin: 533975.6 xmax: 258342.5 ymax: 549358
## Projected CRS: Amersfoort / RD New
## # A tibble: 3 × 2
##   is_inside_utrecht_municipality[,1]
##   <lgl>
## 1 FALSE                                (((258188.9 547230.8, 258197.3 547225.4, 258207
## 2 FALSE                                (((249736.7 548846.2, 249735 548833.4, 249724.8
## 3 FALSE                                (((233262.5 534196, 233255.7 533985.1, 233254.7
```

Working With Vector Data: Topology

- In addition to `st_intersects()`, you can also use one of the following functions:
    - The opposite of `st_intersects()` is `st_disjoint()`, which returns only TRUE for objects that do not spatially relate in any way to the selecting object
    - The function `st_is_within_distance()` detects features that almost touch the selection object, which has an additional dist argument
    - `st_within()` and `st_touches()` returns T/F for objects in the first argument that are strictly within and strictly on the boundary of the second object

Working With Vector Data: Distance

- While the topological relations presented in the previous section are binary (TRUE/FALSE), distance relations are continuous.
- The distance between two sf objects is calculated with `st_distance()`
- `st_distance()` takes two arguments:
    - If you put in two spatial data.frames, you'll get back a **distance matrix**
    - I.e. a matrix reflecting the distance from each element in x to each element in y

```
distance_matrix ← st_distance(schools_geocoded, provincies)
dim(distance_matrix)
```

```
## [1] 6069   12
```

- Meaning this matrix contains the distance for 6065 schools to (the centroids of) 12 provinces.

Working With Vector Data: Distance

- But sometimes, you might want to calculate the *minimum* or *maximum* distance between each object in a `data.frame` and another set of points or polygons

- You can do that using the following:

  - Take the distance matrix as before
  - For each row, ask what is the *minimum* or *maximum* element:

```
max_distances ← apply(distance_matrix, 1, max)
max_distances[1:10]
```

```
##  [1] 231081.6 225551.2 203325.0 231413.8 233384.6 235838.7 232801.0 204200.9 213738.
```

Working With Vector Data: No Data?

- R also has packages that allow us to **geocode place names** or addresses: the **osmdata** package can be used to convert names to geographic coordinates.
    - Use the `getbb()` function to "Google" for a place

```
library(osmdata, quietly=TRUE)
utrecht ← getbb("Utrecht, The Netherlands")
utrecht
```

```
##         min       max
## x  4.970096  5.195155
## y 52.026282 52.142051
```

Working With Vector Data: No Data?

- Then, either get the polygon or the centroid
- And turn it into a spatial data.frame using `st_as_sf()`

```
# Pick the center of this bounding box
centroid ← data.frame(x=(utrecht[1,1]+utrecht[1,2])/2, y = (utrecht[2,1]+utrecht[
# Transform it into sf format:
st_as_sf(centroid, coords = c('x', 'y'), crs='wgs84')
```

```
## Simple feature collection with 1 feature and 0 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 5.082626 ymin: 52.08417 xmax: 5.082626 ymax: 52.08417
## Geodetic CRS:  WGS 84
##                    geometry
## 1 POINT (5.082625 52.08417)
```

Network Data

Network Data

- Next, we consider **network data**, which describes relationships among units rather than units in isolation.
- Examples include friendship networks among people, citation networks among academic articles, and trade and alliance networks among countries.
- Analysis of network data differs from the data analyses we have covered so far in that the **unit of analysis** is a relationship (Imai, 2018).

Network Data Types

- The basic package to deal with network data in R is called `igraph`. Install it and load it
- Also install and load a package called `igraphdata`, which features some example datasets we'll use during this lecture

```r
library(igraph)
library(igraphdata)
```

- As a running example, we'll use a dataset from `igraphdata` called `Koenigsberg`:

```r
data(Koenigsberg)
```

Background

- From `?Koenigsberg`

> The Seven Bridges of Koenigsberg is a notable historical problem in mathematics. Its negative resolution by Leonhard Euler in 1735 laid the foundations of graph theory and presaged the idea of topology. The city of Koenigsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.
>
> The problem was to find a walk through the city that would cross each bridge once and only once. The islands could not be reached by any route other than the bridges, and every bridge must have been crossed completely every time (one could not walk half way onto the bridge and then turn around and later cross the other half from the other side).

Network Data Types

- The first thing we can do is display this dataset in a convenient format:
  - The number represents how many bridges there are between city part x and city part y

```
as.matrix(Koenigsberg)
```

```
## 4 x 4 sparse Matrix of class "dgCMatrix"
##                      Altstadt-Loebenicht Kneiphof Vorstadt-Haberberg Lomse
## Altstadt-Loebenicht                    .        2                  .     1
## Kneiphof                               2        .                  2     1
## Vorstadt-Haberberg                     .        2                  .     1
## Lomse                                  1        1                  1     .
```

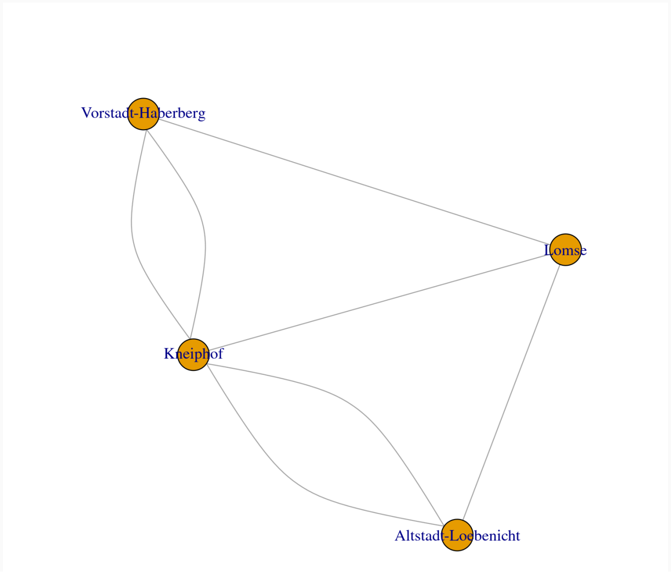
Network Data Types

- Networks are usually displayed as mathematical objects called **graphs**
- A graph $\mathcal{G}$ consists of a set of nodes (or vertices) V and a set of edges (or ties) E , i.e., $\mathcal{G} = (V, E)$.
- A node represents an individual unit and is typically depicted as a solid circle.
  - In our case: *Altstadt-Loebenicht, Kneiphof, Vorstadt-Haberbeg, Lomse*
- An edge, on the other hand, represents the existence of a relationship between any pair of nodes via a line connecting those nodes.
  - In our case: a bridge

Network Data Types

- Graphs can be contained in special matrices called **adjacency matrices**:
  - An adjacency matrix is a matrix whose entries represent the existence of relationships between two units (one unit represented by the row and the other represented by the column)
- In our case:

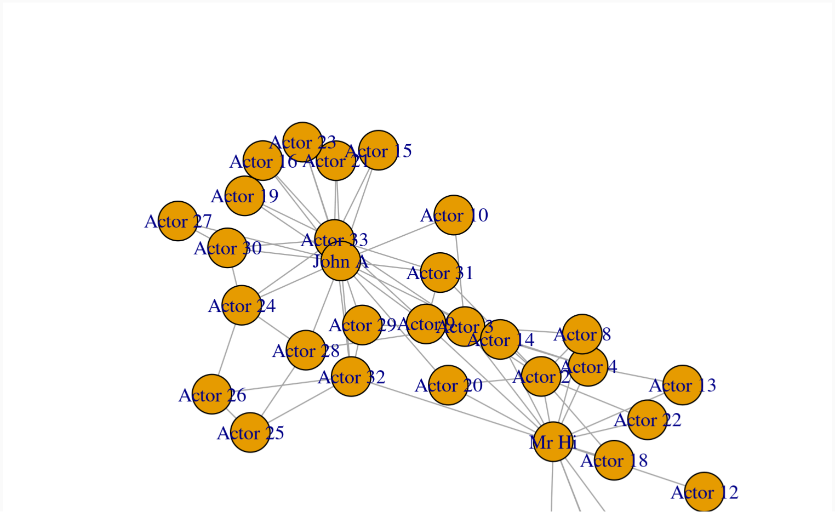
```
adjacency_graph ← graph.adjacency(as.matrix(Koenigsberg), mode="undirected")
plot(adjacency_graph)
```



Analyzing Networks

- There exist a variety of graph-based measures that can quantify **centrality**, or the extent to which each node is connected to other nodes and appears in the center of a graph.
- The **number of edges**, or degree, is perhaps the most crude measure of how well a node is connected to the other nodes in a graph.
- For this example, let's consider the `karate` dataset:

```
data(karate)
graph.adjacency(as.matrix(karate), mode="undirected") ▷ plot()
```



Analyzing Networks

- The **number of edges**, or degree, is perhaps the most crude measure of how well a node is connected to the other nodes in a graph.

```
degree(karate)
```

```
##     Mr Hi  Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7  Actor 8  Actor 9 Act
##        16        9       10        6        3        4        4        4        5
## Actor 15 Actor 16 Actor 17 Actor 18 Actor 19 Actor 20 Actor 21 Actor 22 Actor 23 Act
##         2        2        2        2        2        3        2        2        2
## Actor 29 Actor 30 Actor 31 Actor 32 Actor 33   John A
##         3        4        4        6       12       17
```

Analyzing Networks

(From Imai, 2018)

- Degree is problematically a **local measure** because it simply counts the number of edges that come out of a given node. As a result, it does not account for the structure of the graph beyond its immediate neighbors.
- As an alternative, we can count the **sum of edges** from a given node to all other nodes in a graph, including the ones that are not directly connected.
- This measure, called farness, describes how far apart a given node is from each one of all other nodes in the graph. This contrasts with degree, which counts the number of connected nodes. The inverse of farness, closeness, represents another measure of centrality:

$$\text{closeness}(v) = \frac{1}{\sum_{u \in V, u \neq v} \text{distance}(v, u)}$$

- The distance between two nodes is the number of edges in the shortest path, which is the shortest sequence of connected nodes, between the two nodes of interest.

Analyzing Networks

- In R, we use the `closeness()` function to calculate the closeness for each node:

```
(cl ← closeness(karate))
```

```
##        Mr Hi     Actor 2     Actor 3     Actor 4     Actor 5     Actor 6     Actor 7
## 0.007692308 0.006060606 0.005952381 0.005347594 0.004629630 0.004608295 0.004651163
##    Actor 11    Actor 12    Actor 13    Actor 14    Actor 15    Actor 16    Actor 17
## 0.005319149 0.004424779 0.006211180 0.005780347 0.005181347 0.004166667 0.003289474
##    Actor 21    Actor 22    Actor 23    Actor 24    Actor 25    Actor 26    Actor 27
## 0.006172840 0.005347594 0.004807692 0.004201681 0.004784689 0.003745318 0.005128205
##    Actor 31    Actor 32    Actor 33      John A
## 0.005263158 0.006329114 0.006060606 0.007633588
```

- We can also calculate the correlation between different measures of centrality:

```
de ← degree(karate); cor(cl, de)
```

```
## [1] 0.5860469
```

Recapitulation

- We have learned a lot about **spatial data** today
    - We have learned the difference between vector and raster data
    - We have learned what projections are
    - We have learned how to work with vector data in various ways
    - Including by using spatial operations
- We have also seen various **data packages** containing spatial data
- In addition, we have seen the preliminaries of **network data**
- We have seen how graphs are used to represent network data
    - We have seen two measures of centrality ("importance"), the *degree* and the *closeness*