Geodaten - zweiter Teil

Jan-Philipp Kolb

11 Oktober 2018

Inhalt dieses Abschnitts

- Vorstellung des Openstreetmap (OSM) Projekts
- Welche OSM-Daten sind erhältlich?
- Vorstellung von Forschung die mit OSM-Daten durchgeführt wurde

OpenStreetMap Projekt

OpenStreetMap.org ist ein im Jahre 2004 gegründetes internationales Projekt mit dem Ziel, eine freie Weltkarte zu erschaffen. Dafür sammeln wir weltweit Daten über Straßen, Eisenbahnen, Flüsse, Wälder, Häuser und vieles mehr.

http://www.openstreetmap.de/

OpenStreetMap

OpenStreetMap (OSM) ist ein kollaboratives Projekt um eine editierbare Weltkarte zu erzeugen.

Wikipedia - OpenStreetMap

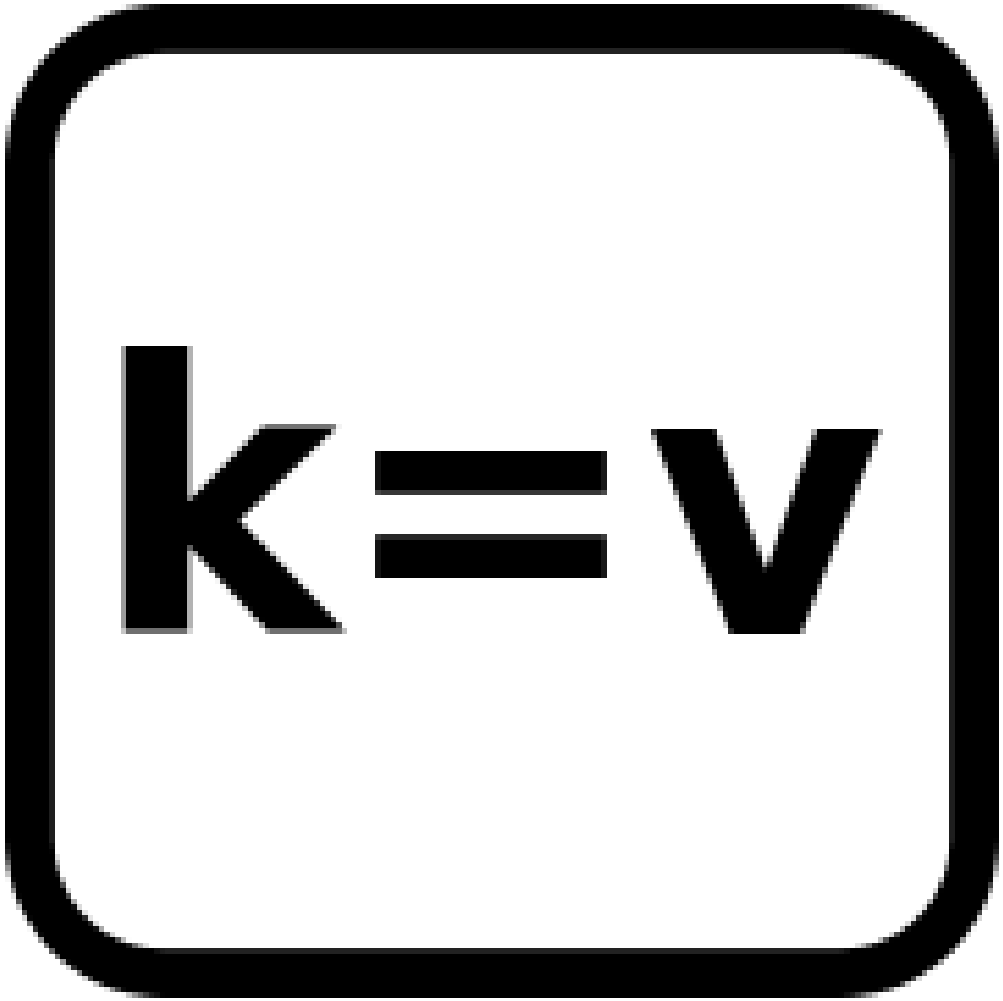
OSM Map Features

Amenity

Used to map facilities used by visitors and residents. For example: toilets, telephones, banks, pharmacies, cafes, parking and schools. See the page **Amenities** for an introduction on its usage.

Key	Value	Element	Comment	Rendering	Photo
			Sustenance		
amenity	bar		**Bar** is a purpose-built commercial establishment that sells alcoholic drinks to be consumed on the premises. They are characterised by a noisy and vibrant atmosphere, similar to a party and usually don't sell food. See also the description of the tags `amenity=pub;bar;restaurant` for a distinction between these.		
amenity	bbq		**BBQ** or **Barbecue** is a permanently built grill for cooking food, which is most typically used outdoors by the public. For example these may be found in city parks or at beaches. Use the tag `fuel=*` to specify the source of heating, such as `fuel=wood;electric;charcoal`. For mapping nearby table and chairs, see also the tag `tourism=picnic_site`. For mapping **campfires** and **firepits**, instead use the tag `leisure=firepit`.		
amenity	biergarten		**Biergarten** or **beer garden** is an open-air area where alcoholic beverages along with food is prepared and served. See also the description of the tags `amenity=pub;bar;restaurant`. A biergarten can commonly be found attached to a beer hall, pub, bar, or restaurant. In this case, you can use `biergarten=yes` additional to `amenity=pub;bar;restaurant`.		
amenity	cafe		**Cafe** is generally an informal place that offers casual meals and beverages, typically, the focus is on coffee or tea. Also known as a **coffeehouse/shop**, **bistro** or **sidewalk cafe**. The kind of food served may be mapped with the tags `cuisine=*` and `diet=*`. See also the tags `amenity=restaurant;bar;fast_food`.		
amenity	drinking_water		**Drinking water** is a place where humans can obtain potable water for consumption. Typically, the water is used for only drinking. Also known as a **drinking fountain** or **water tap**.		

Openstreetmap Tags



Objekttypen in OSM

- Es gibt prinipiell drei verschiedene Objekttypen:
-

Download von OpenStreetMap Daten

- https://mapzen.com/ - Ausschnitte von OpenStreetMap für einzelne Städte (metro extracts)
- Über Geofabrik lassen sich aktuelle Ausschnitte (auch Shapefiles) herunterladen (http://download.geofabrik.de/)
- Kartendaten (**openaprs**)

Bei großen Datenmengen

- Hier geht es nur um das Herunterladen kleiner Ausschnitte.
- Wenn größere Datenmengen benötigt werden, sollte man eine Datenbanklösung finden.
- PostgreSQL hat den Vorteil, dass es Open-Source ist.
- Download PostreSQL
- Hier ist eine Einführung in PostgreSQL zu finden
- Sehr empfehlenswert: Arbeiten mit pgAdmin III
- Beispiel: um Verknüpfung zu einer Datenbank herzustellen - Doppelklick auf den Server in pgAdmin III

PostGIS für PostgreSQL

- **Installieren** der PostGIS Erweiterung:

```
CREATE EXTENSION postgis;
```

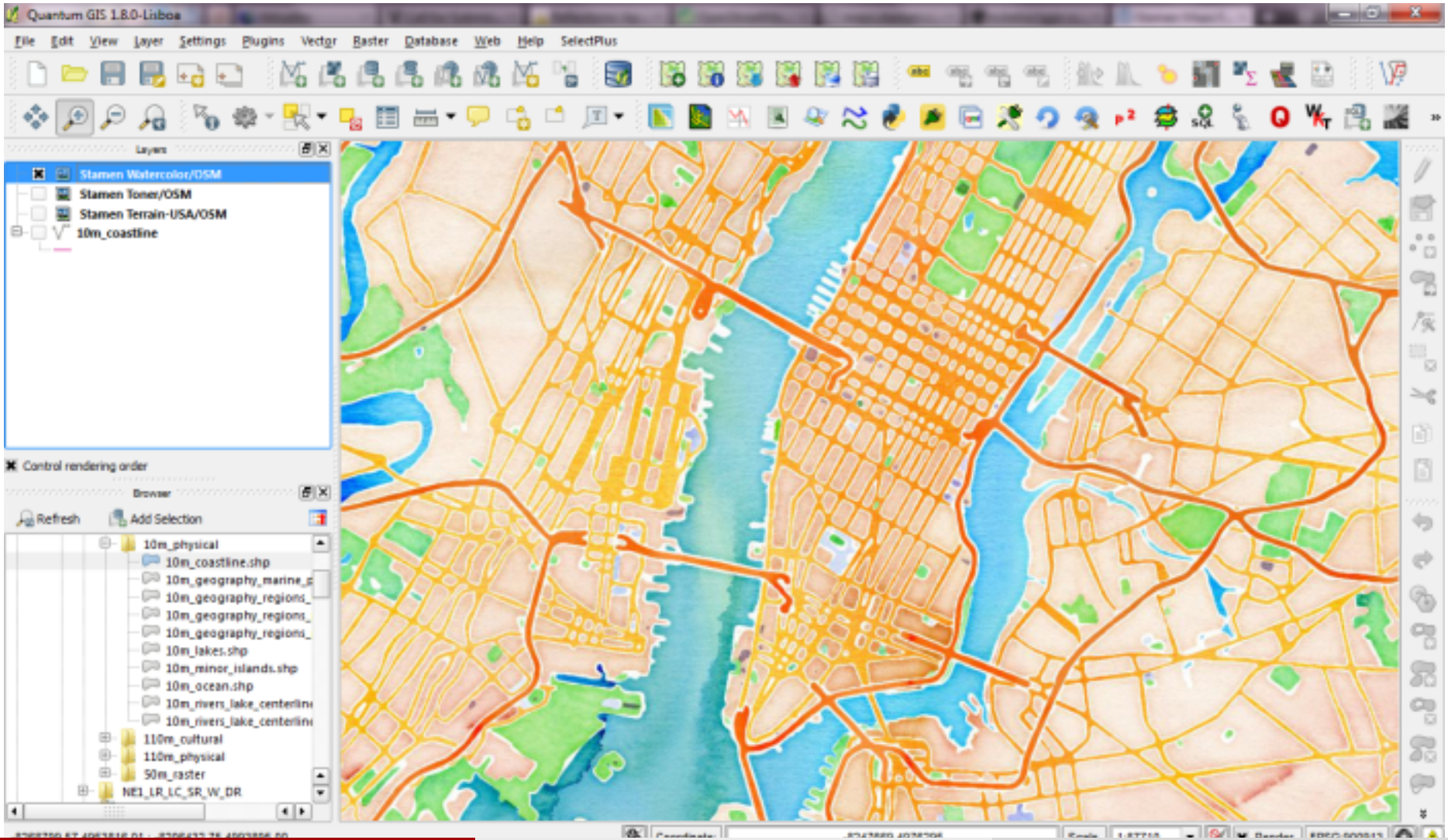
Programm zum Import der OSM Daten in PostgreSQL- osm2pgsql

- Läuft unter Linux deutlich besser
- so könnte bspw. ein Import in PostgreSQL aussehen:

```
osm2pgsql -c -d osmBerlin --slim -C  -k  berlin-latest.osm.pbf
```

Nutze bspw. QGIS um Shapefiles zu extrahieren

- Plugin OpenLayers



Links

- **Wiki zum Downlaod** von Openstreetmap Daten
- **Openstreetmap Blog**
- Liste möglicher Datenquellen für räumliche Analysen (weltweit und in **Deutschland**)
- **SALB** - Administrative Grenzen

http://wiki.openstreetmap.org/wiki/SALB

Inhalt dieses Abschnitts

- Das Konzept der Geokoordinaten erklären
- Möglichkeiten vorstellen, die Geokodierung mit R durchzuführen

Geokodierung

Wikipedia - Geocoding

Geocoding (. . .) uses a description of a location, most typically a postal address or place name, to find geographic coordinates from spatial reference data . . .

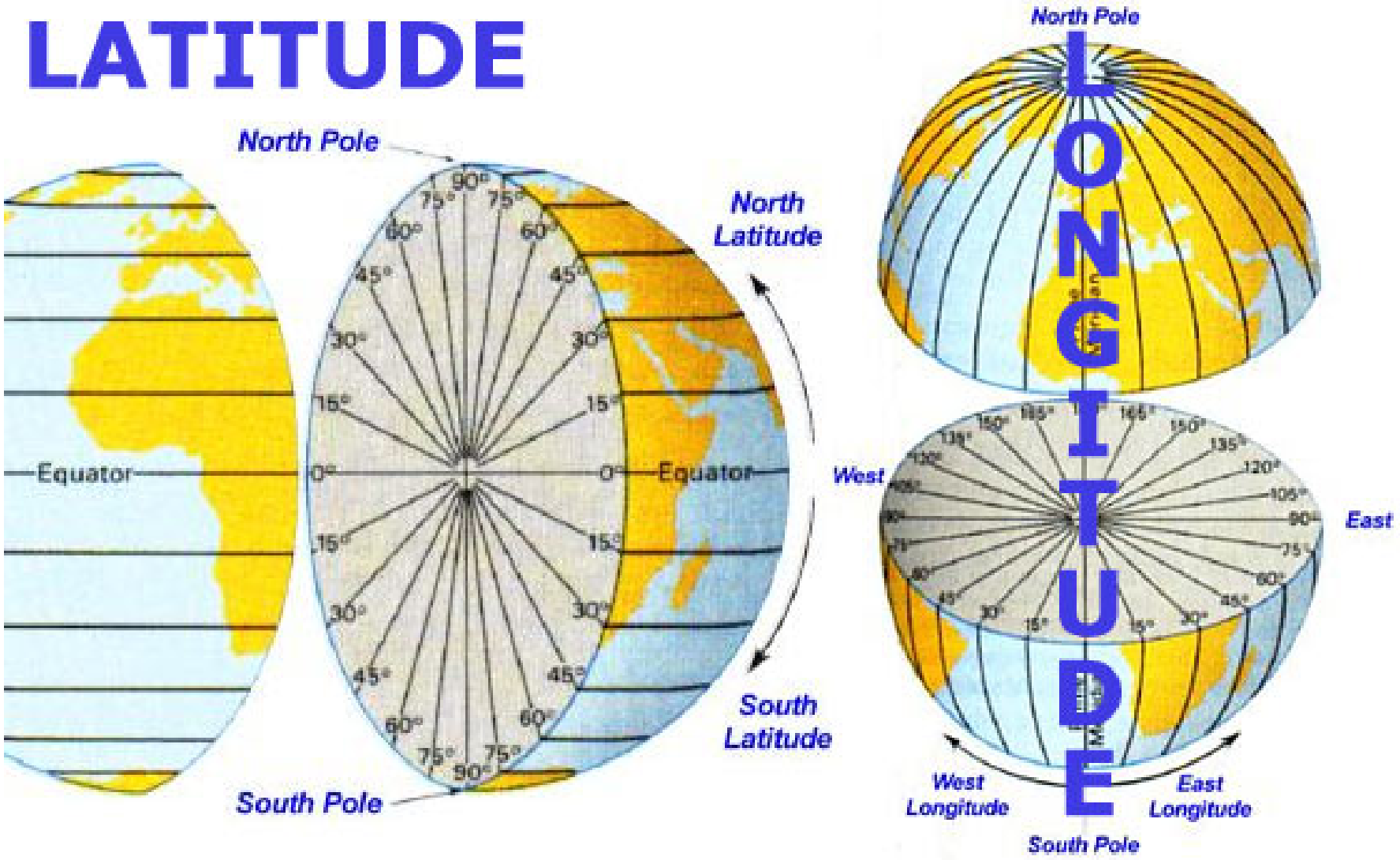
Geokodierung mit dem Paket ggmap

- Einer der ersten Ansätze Geokodierung mit R durchzuführen
- Wenn Geokodierung mit R durchgeführt wird dieses Paket wohl am häufigsten verwendet.
- Das führt auch dazu, dass im Internet zahlreiche Anwendungsbeispiele zu finden sind.

```
library(ggmap)
geocode("Heidelberg")
```

```
Information from URL : http://maps.googleapis.com/maps/api/geo
       lon      lat
1 8.672434 49.39875
```

Latitude und Longitude



http://modernsurvivalblog.com

Die Distanz zwischen zwei Punkten

```
mapdist("Q1, 4 Mannheim","B2, 1 Mannheim")
```

```
mapdist("Q1, 4 Mannheim","B2, 1 Mannheim",mode="walking")
```

Eine andere Distanz bekommen

```
mapdist("Q1, 4 Mannheim","B2, 1 Mannheim",mode="bicycling")
```

Geokodierung mit dem Paket `tmaptools`

- Beim Paket `tmaptools` wird die Nominatim API zur Geokodierung verwendet.
- Diese Funktion hat den Vorteil, dass eine Projektion ausgewählt werden kann, in der die Geokodierungen zurück gegeben werden.

```
library("tmaptools")
```

Koordinaten verschiedener Orte in Deutschland

```
cities <- c("Hamburg","Koeln","Dresden","Muenchen")

lat <- vector()
lon <- vector()
for (i in 1:length(cities)){
  gc <- geocode_OSM(cities[i])
  lat[i] <- gc$coords[1]
  lon[i] <- gc$coords[2]
}
```

Welche Koordinaten hat der Norden

```
Dat <- data.frame(cities,lon,lat)
kable(Dat)
```

cities	lon	lat
Hamburg	53.55034	10.000654
Koeln	50.93836	6.959974
Dresden	51.04933	13.738144
Muenchen	48.13711	11.575382

Geokodierung - verschiedene Punkte von Interesse

Punkte in der Karte

```
MA_map <- qmap("Mannheim")
```

```
MA_map +
geom_point(aes(x = x, y = y),
data = ListPOI)
```

Punkte in der Karte

```
MA_map +
geom_point(aes(x = x, y = y),col="red",
data = ListPOI)
```

Reverse Geokodierung

> *Reverse geocoding is the process of back (reverse) coding of a point location (latitude, longitude) to a readable address or place name. This permits the identification of nearby street addresses, places, and/or areal subdivisions such as neighbourhoods, county, state, or country.*

Quelle: Wikipedia

```
revgeocode(c(48,8))
```

Daten einlesen

- Hier wird ein Beispieldatensatz eingelesen, den ich über räumliche Stichproben und reverse geocoding erzeugt habe.

```
load("../data/addr_list_t_68239.RData")
head(addr_list_t)
```

```
## [1] "Lilienstraße 32A, 68535 Edingen-Neckarhausen, Germany"
## [2] "Waldspitze 6, 68239 Mannheim, Germany"
## [3] "Holzweg 51, 68239 Mannheim, Germany"
## [4] "Kloppenheimer Str. 247, 68239 Mannheim, Germany"
## [5] "Mallaustraße 121, 68219 Mannheim, Germany"
## [6] "Holzweg 33A, 68239 Mannheim, Germany"
```

Die erste Adressen geokodieren

```
geocode_OSM(addr_list_t[1])
```

```
## $query
## [1] "Lilienstraße 32A, 68535 Edingen-Neckarhausen, Germany"
##
## $coords
##         x         y
##  8.584601 49.445360
##
## $bbox
##         min       max
## x  8.584494  8.584708
## y 49.445276 49.445443
```

Alle Adressen geokodieren

```
gc_list <- list()

for (i in 1:length(addr_list_t)){
  gc_list[[i]] <- geocode_OSM(addr_list_t[i])
}
```

Geokodierung mit dem R-Paket opencage

- Um dieses Paket zu nutzen muss man sich vorher bei der API registrieren

```
library(opencage)
```

```
gc_info<-opencage_forward(placename =
                          "Amsterdam, Van Woustraat")
```

- Hinweise, wie das Paket genutzt erden kann sind im **opencage Tutorial** zu finden.

Das Paket geonames

```
install.packages("geonames")
```

- Ein Account ist notwendig um die meisten Funktionen des Paketes geonameszu nutzen.

```
library(geonames)
```

```
options(geonamesUsername="myusername")
```

```
MAwiki<-GNfindNearbyWikipedia(postalcode=68239,country="DE",
                              radius=10)
```

Wikipediaeinträge in der Nähe

elevation	feature	lng	distance	countryCode	rank	lang	title
102	city	8.46711	0.1738	DE	98	en	Qua
103	landmark	8.46212	0.1986	NA	90	en	Reis
103	landmark	8.4616	0.2423	DE	13	en	Klap
104	landmark	8.46294	0.3178	DE	84	en	GES
102	city	8.4691	0.3258	DE	100	en	Mar

- Login für Geonames
- Link um mit den Geodaten zu arbeiten
- Informationen über den Download

```
library(osmdata)
bbox <- getbb("Mannheim")
```

```
erg <- geonames::GNcities(49.649591,8.627236,
```

Das Paket `googleway`

Accesses Google Maps APIs to Retrieve Data and Plot Maps

```
library(googleway)
```

- Ein API Schlüssel ist notwendig um die meisten Funktionen des Paketes zu nutzen.

Das Paket bbox

- Das Paket bbox ist auf github zu finden.
- Beispieldatensatz laden:

```
load("../data/ddat.RData")
```

- Rahmen für das räumliche Objekt bestimmen:

```
library(bbox)
b_box(ddat)
```

```
## [1]  5.866286 47.273602 15.048632 55.058262
```

```
citation("bbox")
```

Links

- Überblick von Jesse Sadler zur **Geokodierung mit R**
- Ein Schummelzettel für `ggmap`
- Die Vignette zum Paket `tmap` - **tmap: get started**
- **latlong.net** - eine Homepage um Koordinaaten zu bestimmen.

OSM Ausschnitte herunterladen

<www.openstreetmap.org/export>

Das R-Paket XML - Gaston Sanchez

```
library("XML")
```

Gaston Sanchez - Dataflow

Das Arbeiten mit XML Daten

Getting Data from the Web with R

Part 4: Parsing XML/HTML Content

Gaston Sanchez

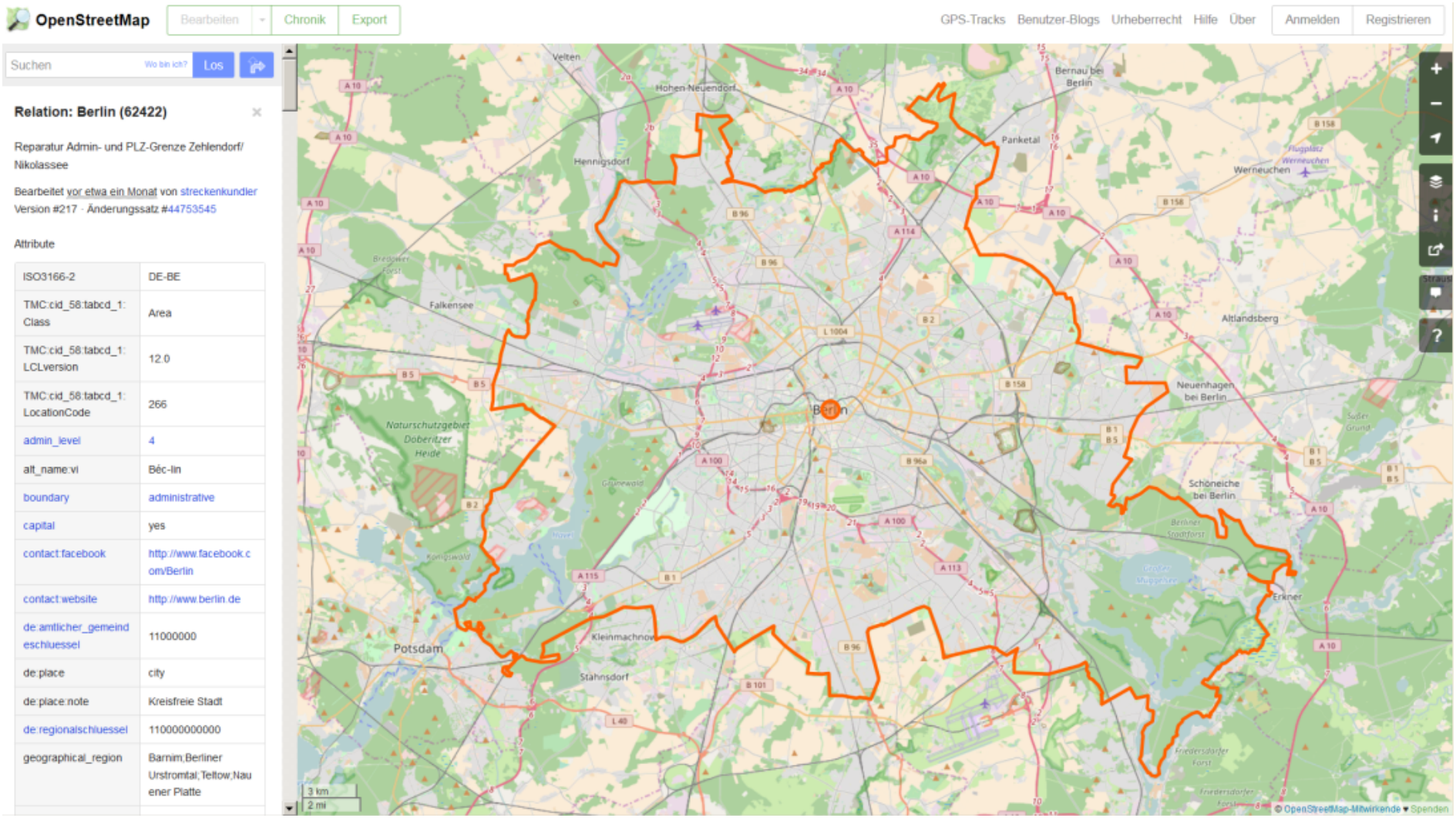
April-May 2014

Funktionen im XML Paket

Function	Description
xmlName()	name of the node
xmlSize()	number of subnodes
xmlAttrs()	named character vector of all attributes
xmlGetAttr()	value of a single attribute
xmlValue()	contents of a leaf node
xmlParent()	name of parent node
xmlAncestors()	name of ancestor nodes
getSibling()	siblings to the right or to the left
xmlNamespace()	the namespace (if there's one)

Einzelne Objekte finden

<www.openstreetmap.org/export>



Beispiel: administrative Grenzen Berlin

Administrative Grenzen für Deutschland

```
url <- "https://api.openstreetmap.org/api/0.6/relation/62422"
```

```
BE <- xmlParse(url)
```

```
BE <- xmlParse("../data/62422.xml")
```


```
- <osm version="0.6" generator="CGImap 0.4.0 (19884 thorn-03.openstreetmap.org)" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright"
  license="http://opendatacommons.org/licenses/odbl/1-0/">
  - <relation id="62422" visible="true" version="209" changeset="36072269" timestamp="2015-12-20T19:49:52Z" user="tbicr" uid="278800">
      <member type="node" ref="240109189" role="admin_centre"/>
      <member type="way" ref="50291800" role="outer"/>
      <member type="way" ref="77913336" role="outer"/>
      <member type="way" ref="315222039" role="outer"/>
      <member type="way" ref="77487568" role="outer"/>
      <member type="way" ref="315222038" role="outer"/>
      <member type="way" ref="98035898" role="outer"/>
      <member type="way" ref="77501737" role="outer"/>
```

Das XML analysieren

- Tobi Bosede - Working with XML Data in R

```
xmltop = xmlRoot(BE)
class(xmltop)
```

```
## [1] "XMLInternalElementNode" "XMLInternalNode"
## [3] "XMLAbstractNode"
```

```
xmlSize(xmltop)
```

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

```
xmlSize(xmltop[[1]])
```

```
## [1] 337
```

Nutzung von Xpath

Xpath, the XML Path Language, is a query language for selecting nodes from an XML document.

```
xpathApply(BE,"//tag[@k = 'population']")
```

```
## [[1]]
## <tag k="population" v="3440441"/>
##
## attr(,"class")
## [1] "XMLNodeSet"
```

Quelle für die Bevölkerungsgröße

```
xpathApply(BE,"//tag[@k = 'source:population']")
```

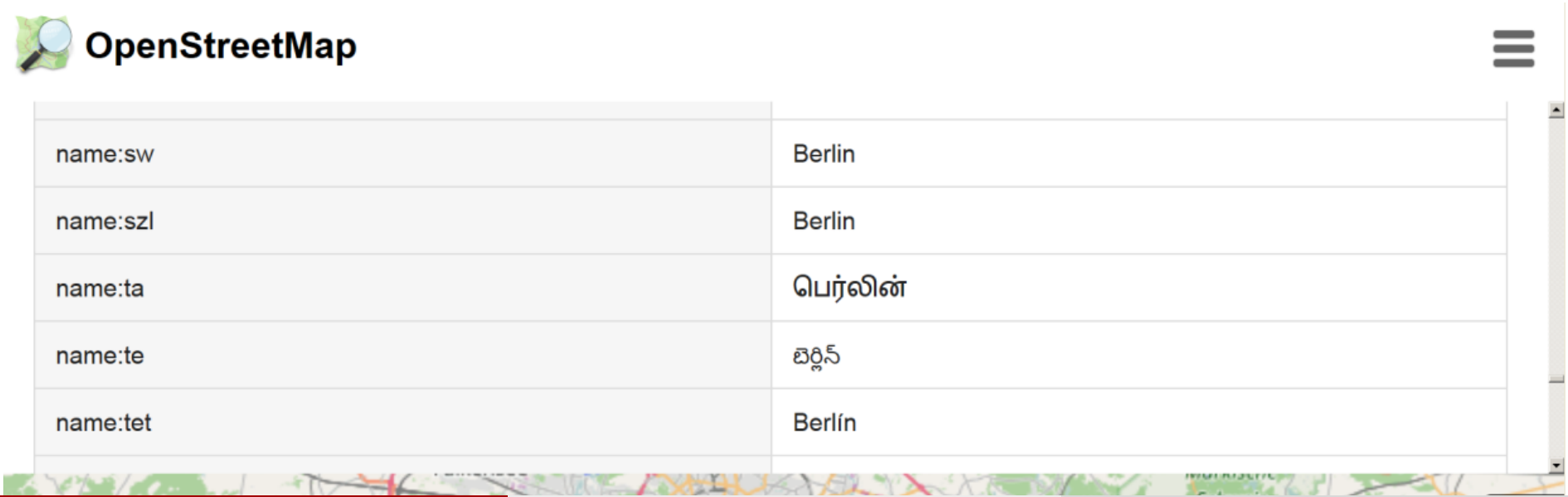
```
## [[1]]
## <tag k="source:population" v="http://www.statistik-berlin-b
##
## attr(,"class")
## [1] "XMLNodeSet"
```

-**Statistik Berlin Brandenburg**

Etwas überraschend:

```
xpathApply(BE,"//tag[@k = 'name:ta']")
```

```
## [[1]]
## <tag k="name:ta" v="<U+0BAA><U+0BC6><U+0BB0><U+0BCD><U+0BB2
##
## attr(,"class")
## [1] "XMLNodeSet"
```



Geographische Region

```
region <- xpathApply(BE,
  "//tag[@k = 'geographical_region']")
# regular expressions
region[[1]]
```

```
## <tag k="geographical_region" v="Barnim;Berliner Urstromtal;
```

```
<tag k="geographical_region"
  v="Barnim;Berliner Urstromtal;
  Teltow;Nauener Platte"/>
```

Landkreis

Weiteres Beispiel

```
url2<-"http://api.openstreetmap.org/api/0.6/node/25113879"
obj2<-xmlParse(url2)
obj_amenity<-xpathApply(obj2,"//tag[@k = 'amenity']")[[1]]
obj_amenity
```

```
## <tag k="amenity" v="university"/>
```

Wikipedia Artikel

```
xpathApply(obj2,"//tag[@k = 'wikipedia']")[[1]]
```

```
## <tag k="wikipedia" v="de:Universität Mannheim"/>
```

```
xpathApply(obj2,"//tag[@k = 'wheelchair']")[[1]]
```

```
xpathApply(obj2,"//tag[@k = 'name']")[[1]]
```

Das C und das A

```
url3<-"http://api.openstreetmap.org/api/0.6/node/303550876"
obj3 <- xmlParse(url3)
xpathApply(obj3,"//tag[@k = 'opening_hours']")[[1]]
```

```
## <tag k="opening_hours" v="Mo-Sa 09:00-20:00; Su,PH off"/>
```

Hin und weg

```
url4<-"http://api.openstreetmap.org/api/0.6/node/25439439"
obj4 <- xmlParse(url4)
xpathApply(obj4,"//tag[@k = 'railway:station_category']")[[1]]

## <tag k="railway:station_category" v="2"/>
```

- **Wikipedia Artikel Bahnhofskategorien**

Stufe	Bahnsteigkanten	Bahnsteiglänge	Reisende/Tag	Zughalte/Tag
6	1	bis 90 m	bis 49	bis 10
5	2	> 90 bis 140 m	50 bis 299	11 bis 50
4	3 bis 4	> 140 bis 170 m	300 bis 999	51 bis 100
3	5 bis 9	> 170 bis 210 m	1000 bis 9999	101 bis 500
2	10 bis 14	> 210 bis 280 m	10.000 bis 49.999	501 bis 1000
1	ab 15	> 280 m	ab 50.000	ab 1001

Prozent	Kategorie
> 90 %	1
> 80 bis 90 %	2
> 60 bis 80 %	3
> 50 bis 60 %	4
> 40 bis 50 %	5
> 25 bis 40 %	6
bis 25 %	7

Exkurs: Bahnhofskategorien

- **rvest: Easily Harvest (Scrape) Web Pages**

```
library(rvest)
bhfkat<-read_html(
  "https://de.wikipedia.org/wiki/Bahnhofskategorie")
df_html_bhfkat<-html_table(
  html_nodes(bhfkat, "table")[[2]],fill = TRUE)
```

Bahnhofskategorien Übersicht

Stufe	Bahnsteigkanten	Bahnsteiglänge[Anm 1]	Reisende/Tag
(0)	—	—	—
1	01	> 000 bis 090 m	00.000 bis 00.049
2	02	> 090 bis 140 m	00.050 bis 00.299
3	03 bis 04	> 140 bis 170 m	00.300 bis 0.0999
4	05 bis 09	> 170 bis 210 m	01.000 bis 09.999
5	10 bis 14	> 210 bis 280 m	10.000 bis 49.999
6	00i ab 15	> 280 m bis 000	000000 ab 50.000
Gewichtung	20 %	20 %	20 %

Nur fliegen ist schöner

```
url5<-"http://api.openstreetmap.org/api/0.6/way/162149882"
obj5<-xmlParse(url5)
xpathApply(obj5,"//tag[@k = 'name']")[[1]]
```

```
## <tag k="name" v="City-Airport Mannheim"/>
```

```
xpathApply(obj5,"//tag[@k = 'website']")[[1]]
```

```
## <tag k="website" v="http://www.flugplatz-mannheim.de/"/>
```

```
xpathApply(obj5,"//tag[@k = 'iata']")[[1]]
```

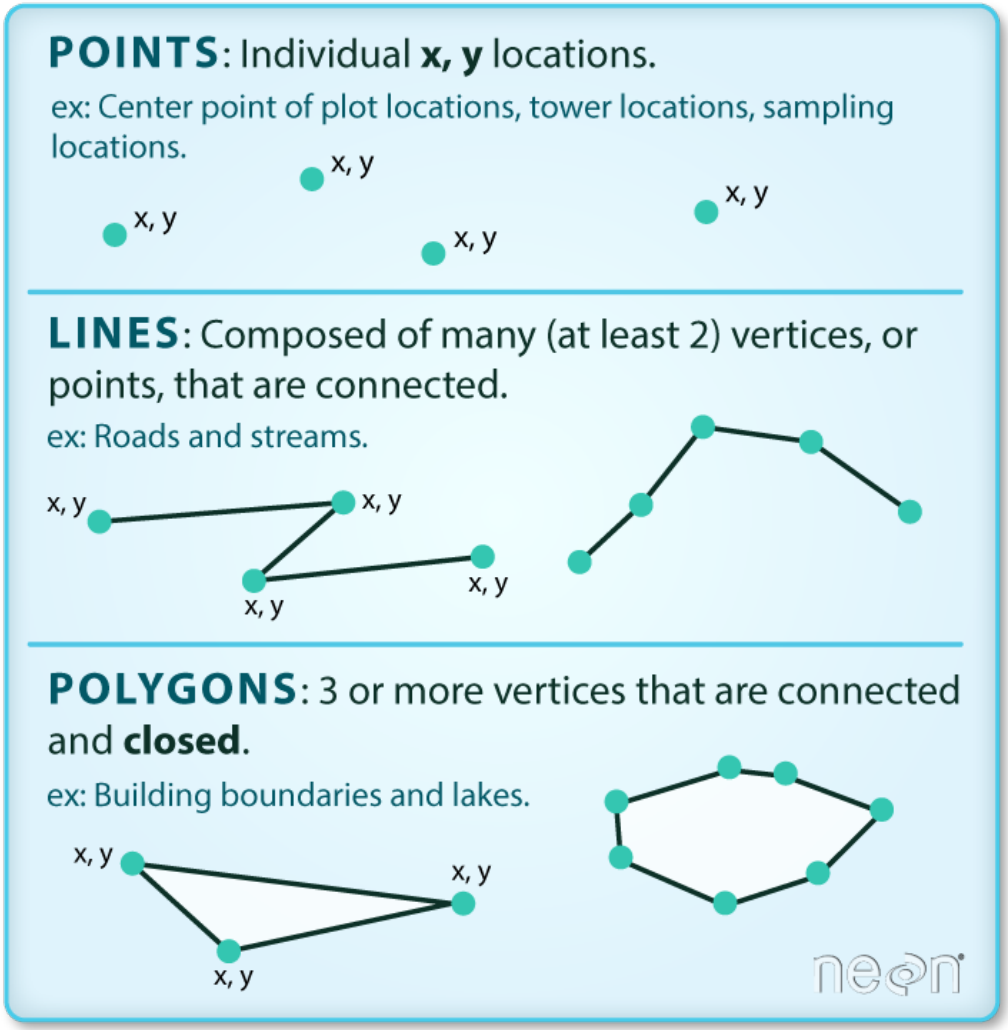
```
## <tag k="iata" v="MHG"/>
```

Das Paket osmar benutzen

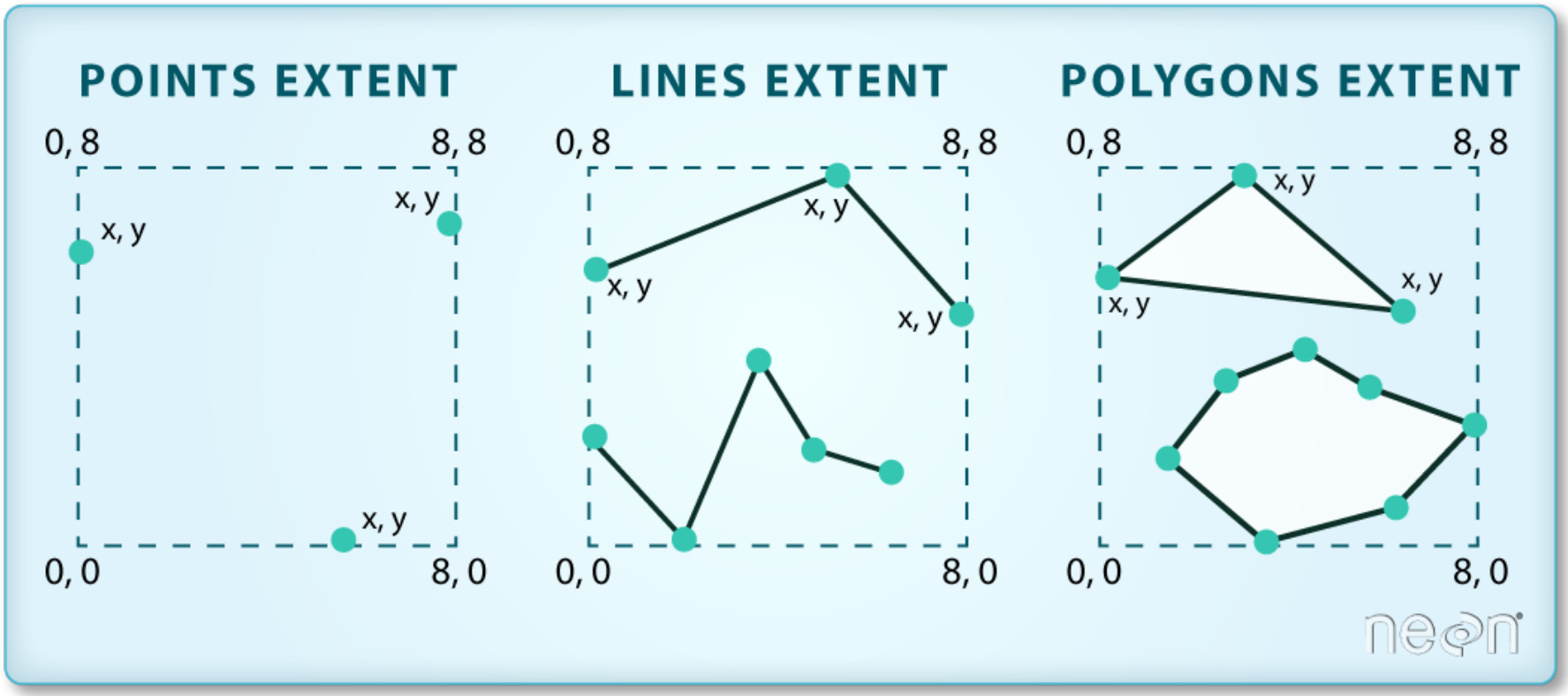
```
library("osmar")
node_ <- xmlParse("../data/162149882.xml")
node_osmar <- as_osmar(node_)
node_osmar
```

```
## osmar object
## 0 nodes, 1 ways, 0 relations
```

Drei Typen von Vektorobjekten



Die Ausdehnung



Import mit dem Paket sf

```
library(sf)
```

- Mit dem Befehl st_layers kann man sehen, welche Layer verfügbar sind:

```
st_layers("../data/Amsterdam_highway_primary.osm")
```

```
## Driver: OSM
## Available layers:
##         layer_name       geometry_type features fields
## 1           points               Point       NA     10
## 2            lines         Line String       NA      9
## 3 multilinestrings   Multi Line String       NA      4
## 4    multipolygons        Multi Polygon       NA     25
## 5  other_relations Geometry Collection       NA      4
```

Import von Layer `lines`

```
dat <- st_read("../data/Amsterdam_highway_primary.osm","lines"
```

```
## Reading layer `lines' from data source `D:\Daten\GitHub\geo
## Simple feature collection with 1464 features and 9 fields
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 8.333102 ymin: 49.32801 xmax: 8.62799
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
plot(dat$geometry)
```

Import von Layer points

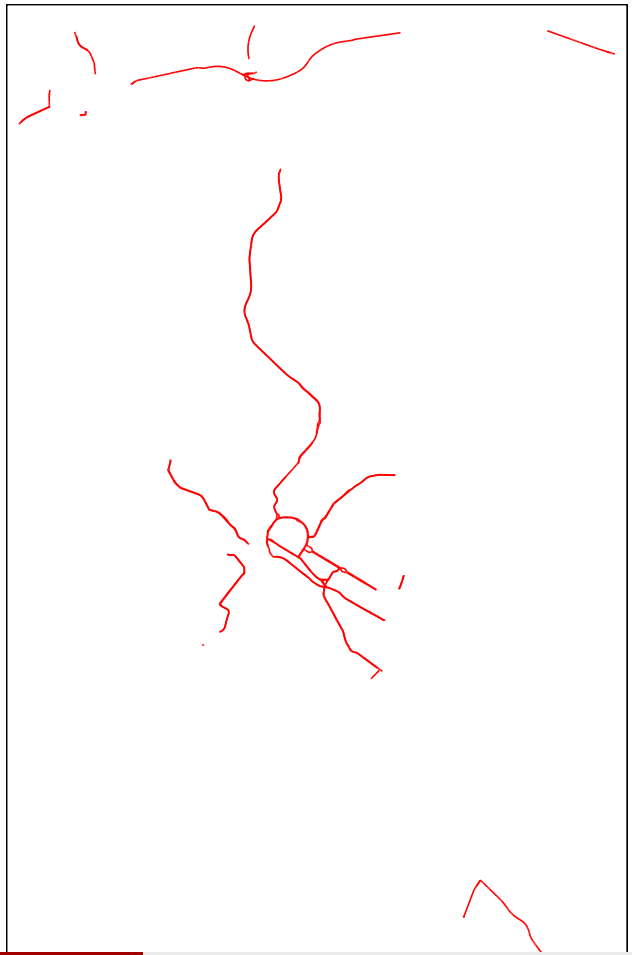
```
datp <- st_read("../data/Amsterdam_highway_primary.osm","point
```

```
## Reading layer `points' from data source `D:\Daten\GitHub\ge
## Simple feature collection with 800 features and 10 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 8.33654 ymin: 49.32801 xmax: 8.62696
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
plot(dat$geometry,pch=20,col=rgb(0,0,1,.1))
```

Mit einem anderen Paket plotten

```
library(tmap)
qtm(dat$geometry)
```



```
st_layers("../data/ams_centraal.osm")
```

```
## Driver: OSM
## Available layers:
##         layer_name       geometry_type features fields
## 1           points               Point       NA     10
## 2            lines         Line String       NA      9
## 3 multilinestrings   Multi Line String       NA      4
## 4    multipolygons       Multi Polygon       NA     25
## 5  other_relations Geometry Collection       NA      4
```

```
datm <- st_read("../data/ams_centraal.osm","multipolygons")
```

```
## Reading layer `multipolygons' from data source `D:\Daten\Gi
## Simple feature collection with 2796 features and 25 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 4.874776 ymin: 52.36088 xmax: 4.92975
```

Mehr Beispiele, wie man mit XML Daten umgeht:

- Deborah Nolan - **Extracting data from XML**
- Duncan Temple Lang - **A Short Introduction to the XML package for R**

Noch mehr Informationen

- **Web Daten manipulieren**
- **Tutorial zu xquery**
- **R und das Web (für Anfänger), Teil II: XML und R**
- Gaston Sanchez - **String Manipulation**
- **Nutzung, Vor- und Nachteile OSM**
- **Forschungsprojekte im Zusammenhang mit OpenStreetMap**

Referenzen

```
citation("XML")
```

```
##
## To cite package 'XML' in publications use:
##
##   Duncan Temple Lang and the CRAN Team (2018). XML: Tools f
##   Parsing and Generating XML Within R and S-Plus. R package
##   version 3.98-1.12. https://CRAN.R-project.org/package=XMl
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {XML: Tools for Parsing and Generating XML Witl
##     author = {Duncan Temple Lang and the CRAN Team},
##     year = {2018},
##     note = {R package version 3.98-1.12},
```

Das neuere Paket

```
citation("xml2")
```

```
##
## To cite package 'xml2' in publications use:
##
##   Hadley Wickham, James Hester and Jeroen Ooms (2018). xml2
##   XML. R package version 1.2.0.
##   https://CRAN.R-project.org/package=xml2
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {xml2: Parse XML},
##     author = {Hadley Wickham and James Hester and Jeroen Oo
##     year = {2018},
##     note = {R package version 1.2.0},
```

Themen dieses Abschnitts

- Das R-Paket `osmdata` mit dem man OSM-Daten herunterladen kann und das auf der **Overpass API** beruht.
- Das Paket `osmplotr`

Das `osmdata` Paket

Mark Padgham - Import 'OpenStreetMap' Data as Simple Features or Spatial Objects

Das osmdata Paket

- Mit dem Paket kann man Daten von OpenStreetMap importieren
- Die OSM Daten sind unter **ODbL licence** zu haben

```
install.packages("osmdata")
```

```
library(osmdata)
```

```
citation("osmdata")
```

Das Paket osmplotr

```
library("osmplotr")
library("osmdata")
```

Beispiel Kindergärten in Mannheim

```
bbox <- getbb("Mannheim")
dat_osm <- extract_osm_objects(key='building',
                               value="kindergarten",
                               bbox=bbox)
```

Einen Rahmen definieren um Daten zu bekommen

- Der Rahmen kann entweder erstellt werden, indem die Koordinaten angegeben werden:

```
q <- opq(bbox = c(52.3, 4.7, 52.4, 5.1))
```

- oder indem man den Befehl `getbb` verwendet:

```
bb <- getbb('Ladenburg')
```

- In `bb` sind nun vier Werte gespeichert, die den Rahmen definieren
- Befehl `opq` - eine Overpass Anfrage erstellen

```
q <- opq(bbox = bb)
```

Die Grenze von Mannheim

- Erst mit dem Argument `format_out=polygon` Befehl `getbb` erhält man das Polygon:

```
bb_poly <- getbb(place_name = "Ladenburg",
                 format_out = "polygon")
```

- Das Ergebnis ist sind zwei Vektoren mit den Longitude und Latitude Koordinaten.

```
##           [,1]     [,2]
## [1,] 8.569720 49.49107
## [2,] 8.569858 49.49101
## [3,] 8.569999 49.49096
## [4,] 8.570342 49.49085
```

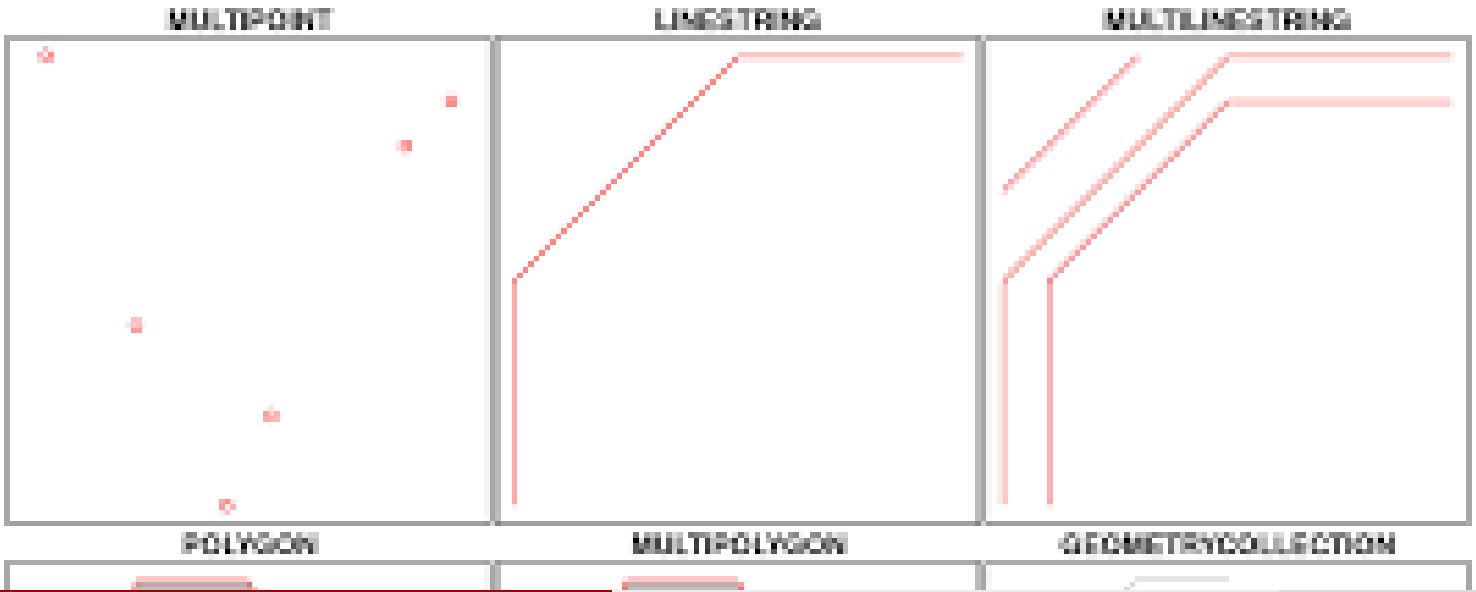
Das Paket für simple feature (`sf`)

Simple Features for R

- Das Paket `sf` ist ein Paket um geometrische Operationen durchzuführen.

```
library(sf)
```

- **Vignette für das Paket `sf`**



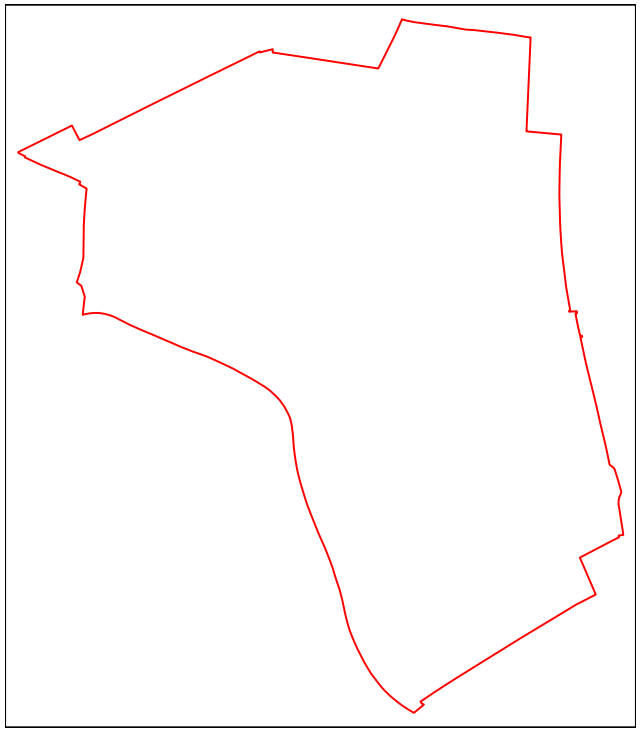
Die Funktion `st_linestring`

Create simple feature from a numeric vector, matrix or list

```
library(sf)
ls <- st_linestring(bb_poly)
sfc <- st_sfc(ls)
```

Den linestring plotten

```
library(tmap)
qtm(sfc)
```



Einrichtungen (amenity)

OSM map features

- Alle benannten Objekte findet man, wenn man OSM map features in eine Suchmaschine eingibt.
- Achtung, wenn man bspw. alle Objekte mit dem Schlüssel `amenity` für eine Stadt heraussucht, bekommt man einen recht großen Datensatz

```
q <- add_osm_feature (q, key = 'amenity')
osmdata_xml(q, '../data/Ladenburg_amenity.osm')
```

Was dahinter steckt

Die Funktion osmdata_sf

- Die Funktion osmdata_sf gibt ein osmdata ObjeKt im sf Format.

```
library(magrittr)
dat1 <- opq(bbox = 'Ladenburg') %>%
    add_osm_feature(key = 'shop', value = 'bakery') %>%
    osmdata_sf ()
```

```
unlist(lapply(dat1,nrow))
```

```
##          osm_points           osm_lines       osm_polygons     os
##                  16                   0                   0
## osm_multipolygons
##                   0
```

Alles in eine Karte plotten

**Der Start mit dem Paket `tmap`

```
library(tmap)
tm_shape(sfc)
tm_bubbles(dat, size=2)
```

Beispiel Fahrradparkplätze

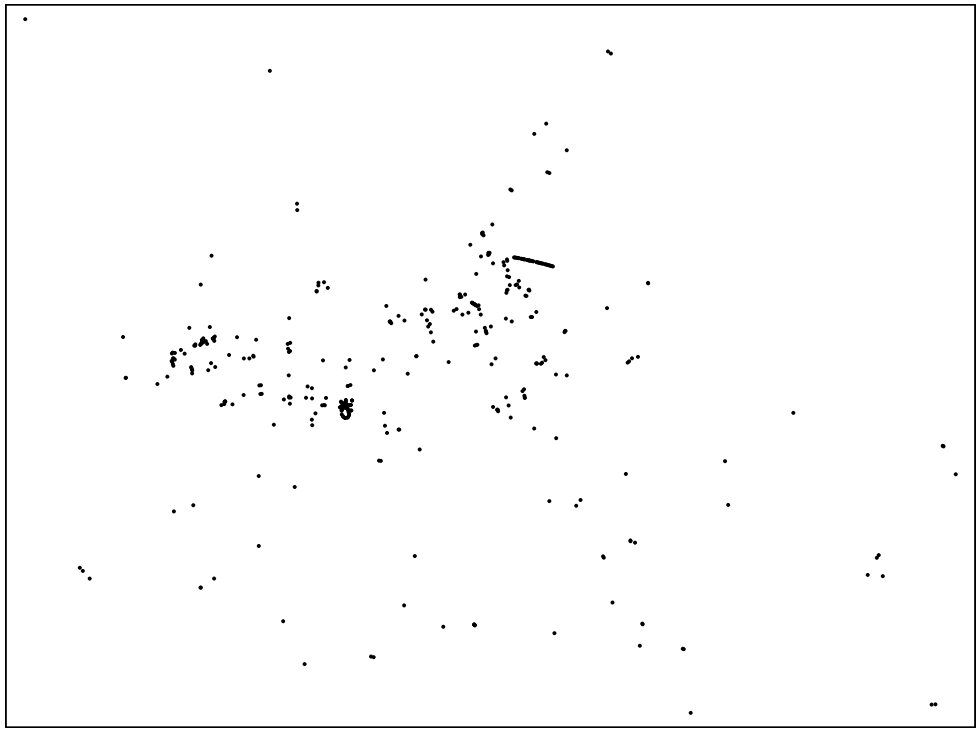
- **OSM map features**

```
q <- add_osm_feature (q, key = 'amenity',value = "bicycle_park
osmdata_xml(q, '../data/Amsterdam_amenity_bicycle_parking.osm'
```

```
dat <- sf::st_read ('../data/Amsterdam_amenity_bicycle_parking
                    layer = 'points',
                    quiet = TRUE)
```

Die Daten plotten

```
library(tmap)
qtm(dat)
```



Sehen was dahinter steckt

```
dat <- sf::st_read ('../data/Amsterdam_amenity.osm',
                    layer = 'points',
                    quiet = TRUE)
```

```
nrow(dat)
names(dat)
```

Bar's in Mannheim

```
?add_osm_feature
```

```
q <- opq (bbox = 'Mannheim')
q <- add_osm_feature (q, key ="amenity",value = 'bar')
osmdata_xml (q, 'data/Mannheim_bar.osm')
```

```
dat_bar <- sf::st_read ('../data/Mannheim_bar.osm',
                        layer = 'points', quiet = TRUE)
```

Bus stations in Amsterdam

```
q <- opq (bbox = 'Amsterdam')
q <- add_osm_feature (q, key ="amenity",
                         value = 'bus_station')
osmdata_xml (q, 'data/Amsterdam_bus_station.osm')
```

```
dat_bus <- sf::st_read ('../data/Amsterdam_bus_station.osm',
                        layer = 'points', quiet = TRUE)
nrow(dat_bus)
```

```
?sf::st_read
```

An alternative

- Main vignette osmdata
- OpenStreetMap Data Structure

```
q <- opq (bbox = 'Amsterdam')
q <- add_osm_feature (q, key ="public_transport",
                      value = 'station')
osmdata_xml (q, '../data/Amsterdam_bus_pubtrans.osm')
```

```
dat_bus <- sf::st_read ('../data/Amsterdam_bus_pubtrans.osm',
                        layer = 'points', quiet = TRUE)
nrow(dat_bus)
```

Further information about public transport

Stop area

```
dat3 <- opq(bbox = 'Amsterdam') %>%
    add_osm_feature(key = 'railway',
                    value = 'tram_stop') %>%
    osmdata_sf ()
```

```
dat3$osm_points$geometry
```

Plotting the result

```
# install.packages("osmplotr")
library("osmplotr")
bbox <- getbb("Amsterdam")
dat_pa <- extract_osm_objects(key='highway',
                              value="primary",
                              bbox=bbox)
dat_sa <- extract_osm_objects(key='highway',
                              value="secondary",
                              bbox=bbox)
dat_ta <- extract_osm_objects(key='highway',
                              value="tertiary",
                              bbox=bbox)
```

```
map <- osm_basemap(bbox = bbox, bg = c("#F5F5DC"))
map <- add_osm_objects(map, dat_pa, col = c("#00008B"))
map <- add_osm_objects(map, dat_sa, col = "green")
```

Get an overview of the available features

```
features <- available_features()
head(features,n=20)
```

```
##  [1] "4wd only"                  "abandoned"
##  [3] "abutters"                  "access"
##  [5] "addr"                      "addr:city"
##  [7] "addr:conscriptionnumber"   "addr:country"
##  [9] "addr:district"             "addr:flats"
## [11] "addr:full"                 "addr:hamlet"
## [13] "addr:housename"            "addr:housenumber"
## [15] "addr:inclusion"            "addr:interpolation"
## [17] "addr:place"                "addr:postcode"
## [19] "addr:province"             "addr:state"
```

Changing the API

```
api_list <- c('http://overpass-api.de/api/interpreter',
              'https://lz4.overpass-api.de/api/interpreter',
              'https://z.overpass-api.de/api/interpreter',
              'https://overpass.kumi.systems/api/interpreter')

api_to_use <- sample(1:length(api_list), 1)

set_overpass_url(api_list[api_to_use])
```

Die wichtigsten Funktionen im Paket `osmdata`

```
# https://rdrr.io/cran/osmdata/man/osmdata_sp.html
?osmdata_sp
```

Links

- Github repo of the `osmdata` package
- Vignette for the package `osmdata` on github
- osmdata Homepage
- Overpass API - query form
- Overpass API/Language Guide
- Overpass Turbo
- **`osmplotr` tutorial
- **Geocomputation with R**
- **osmar - JOS**

Beispiel zu Campingplätzen

- Die Daten stammen von:

http://www.openstreetmap.de/

- Dabei wird die Overpass API genutzt:

http://wiki.openstreetmap.org/wiki/Overpass_API

```
url <- "https://raw.githubusercontent.com/Japhilko/
GeoData/master/2015/data/CampSites_Germany.csv"
```

```
CampSites <- read.csv(url)
```

Überblick über Daten zu Campingplätzen

X	name	tourism	website
1	Campingplatz Winkelbachtal	camp_site	http://www.gruibir
2	Radler-Zeltplatz	camp_site	NA
3	Campingplatz des Naturfreundehauses	camp_site	NA
4	Campingplatz Am Aichstruter Stausee	camp_site	NA
5	NA	camp_site	NA
6	Kandern	camp_site	NA
7	Campingplatz Baiersbronn-Obertal	camp_site	NA
8	Campingplatz SchwabenmÃ¼hle	camp_site	NA

Notwendige Pakete

magrittr - für den Pipe Operator in R:

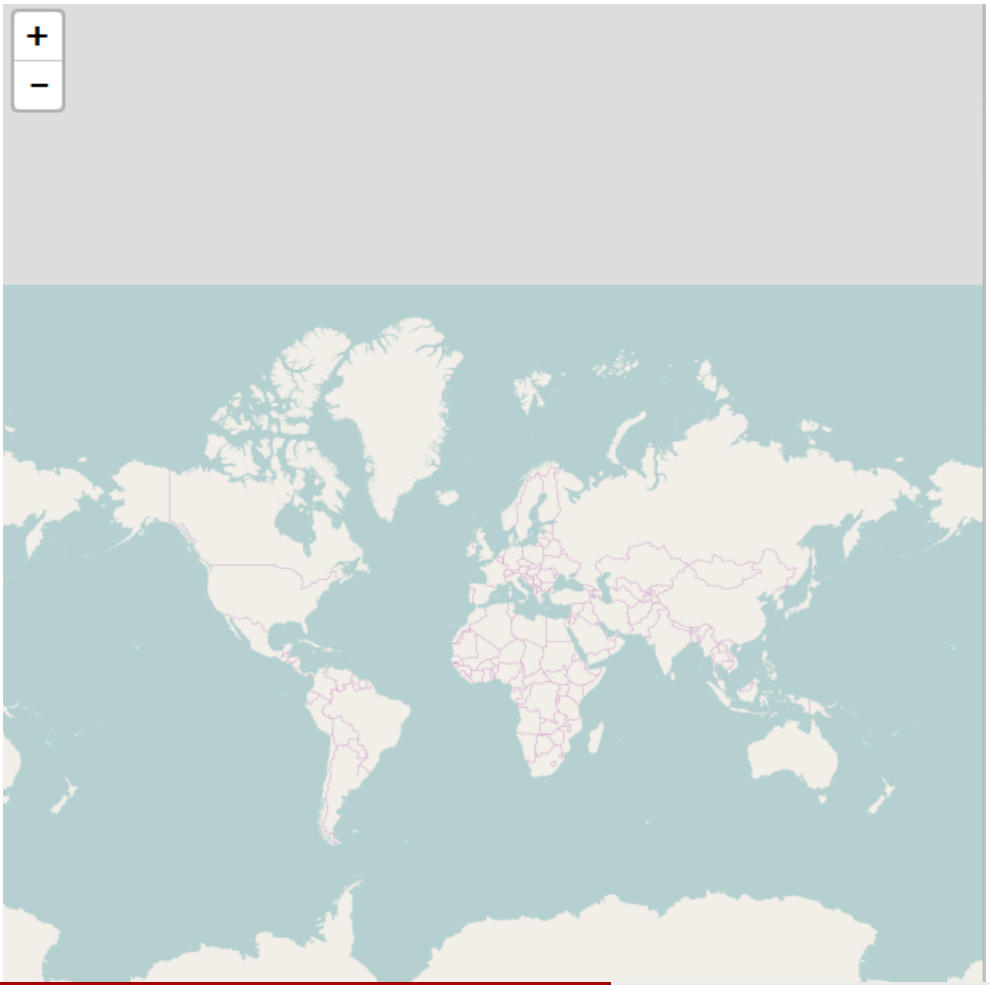
```
library("magrittr")
```

leaflet - um interaktive Karten mit der JavaScript Bibliothek ‘Leaflet’ zu erzeugen

```
library("leaflet")
```

Eine erste interaktive Karte

```
leaflet()%>%
  addTiles()
```



Auf eine Stadt zoomen

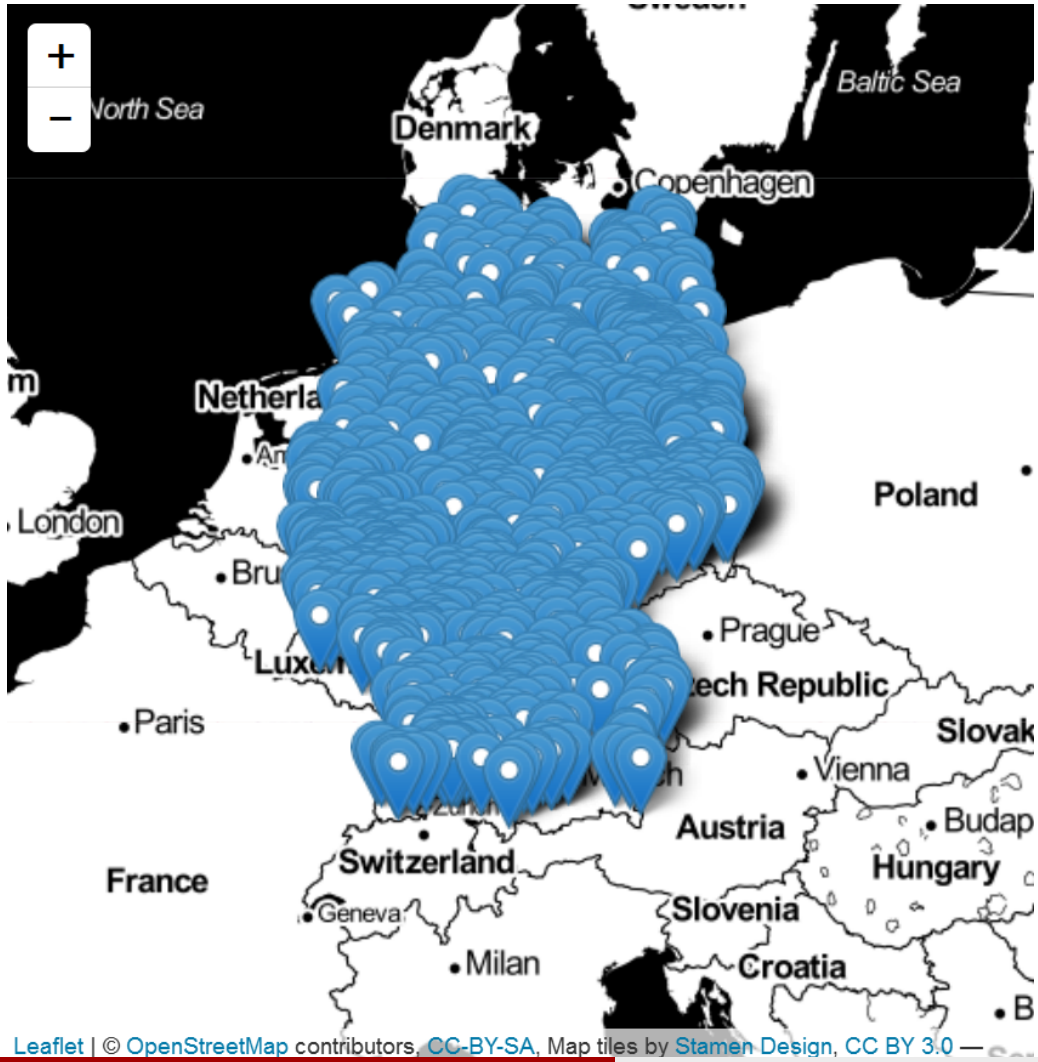
```
leaflet() %>%
  addTiles() %>%
  addMarkers(lng=8.456597, lat=49.48738,
             popup="Wo wir sind")
```

Eine interaktive Karte

```
m <- leaflet() %>%
  addTiles() %>%
  addMarkers(lng=CampSites$lon,
             lat=CampSites$lat,
             popup=CampSites$name)
m
```

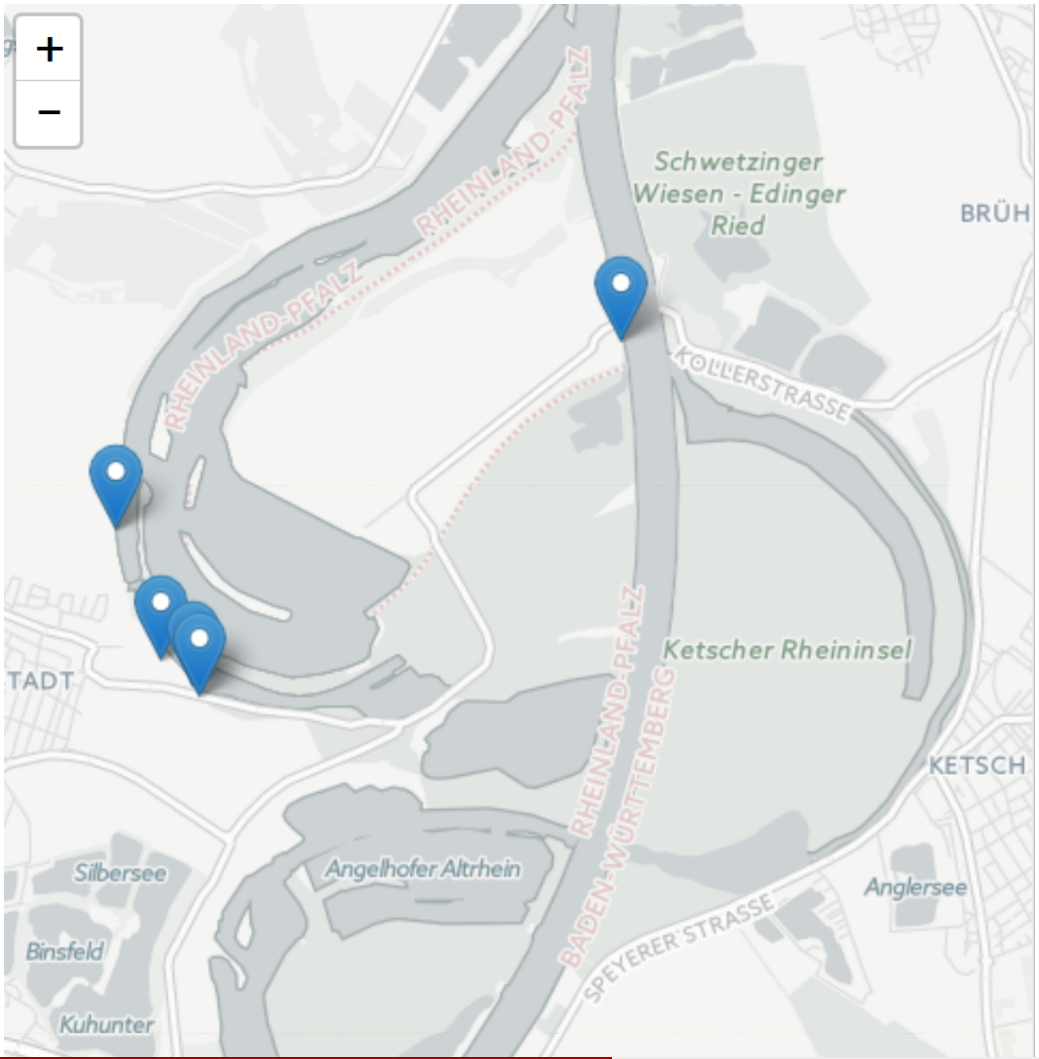
Stamen als Hintergrundkarte

```
m %>% addProviderTiles("Stamen.Toner")
```



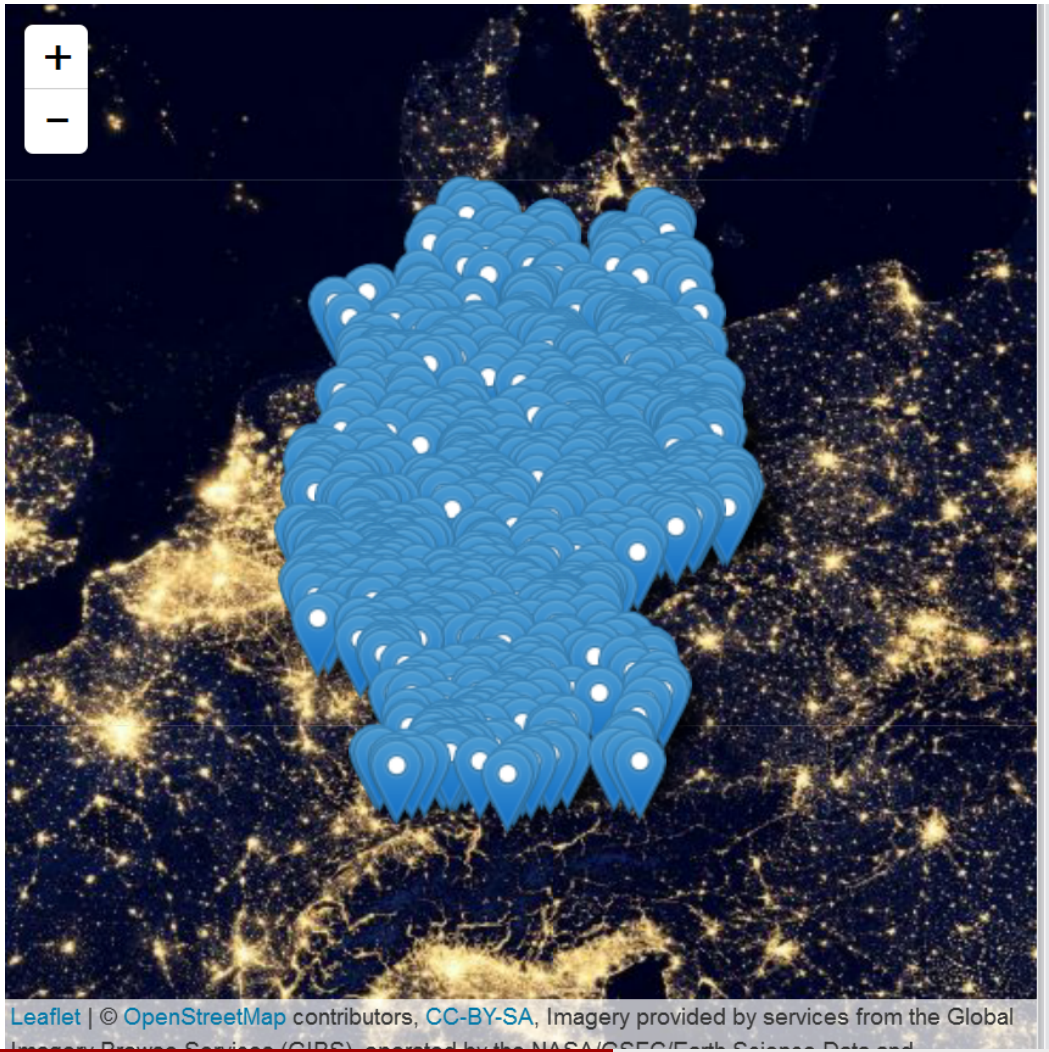
CartoDB als Hintergrund

```
m %>% addProviderTiles("CartoDB.Positron")
```



Mehr Hintergründe

```
m %>% addProviderTiles("NASAGIBS.ViirsEarthAtNight2012")
```



Mehr Informationen hinzufügen

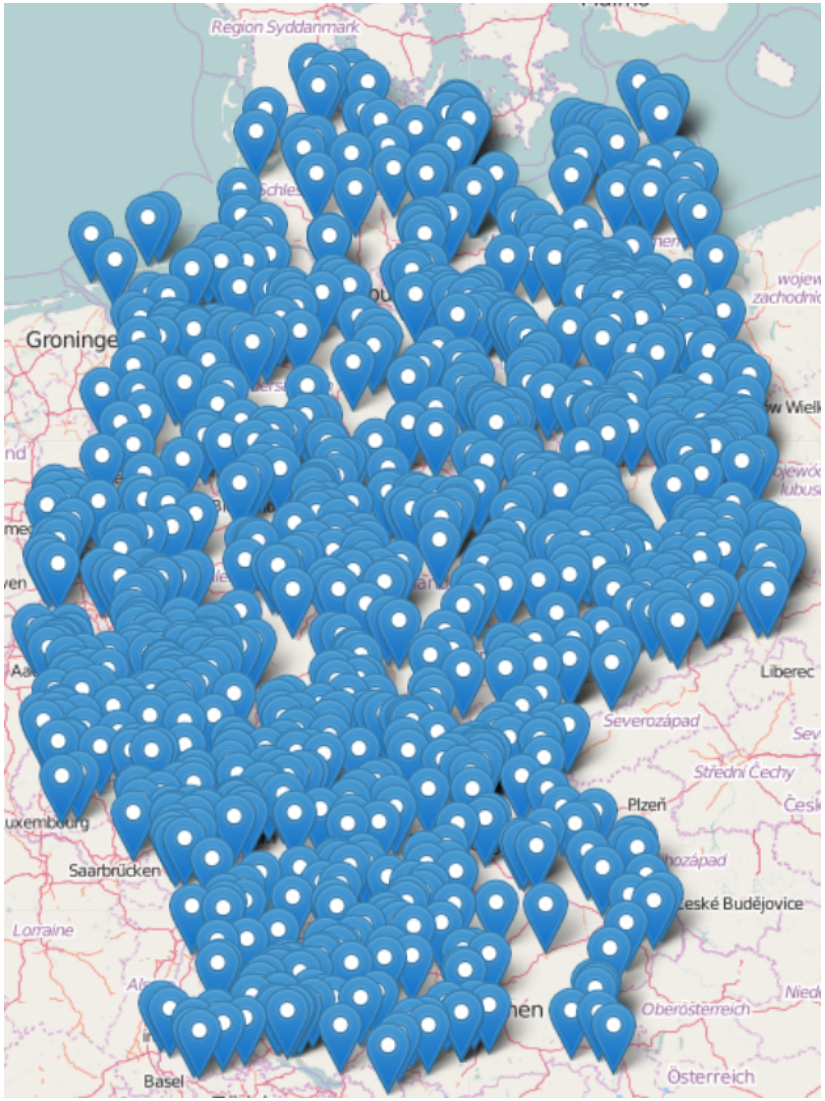
```
popupInfo <- paste(CampSites$name,"\n",CampSites$website)
```

```
m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=CampSites$lon,
             lat=CampSites$lat,
             popup=popupInfo)
m
```

Das Ergebnis ist hier:

http://rpubs.com/Japhilko82/CampSitesHL

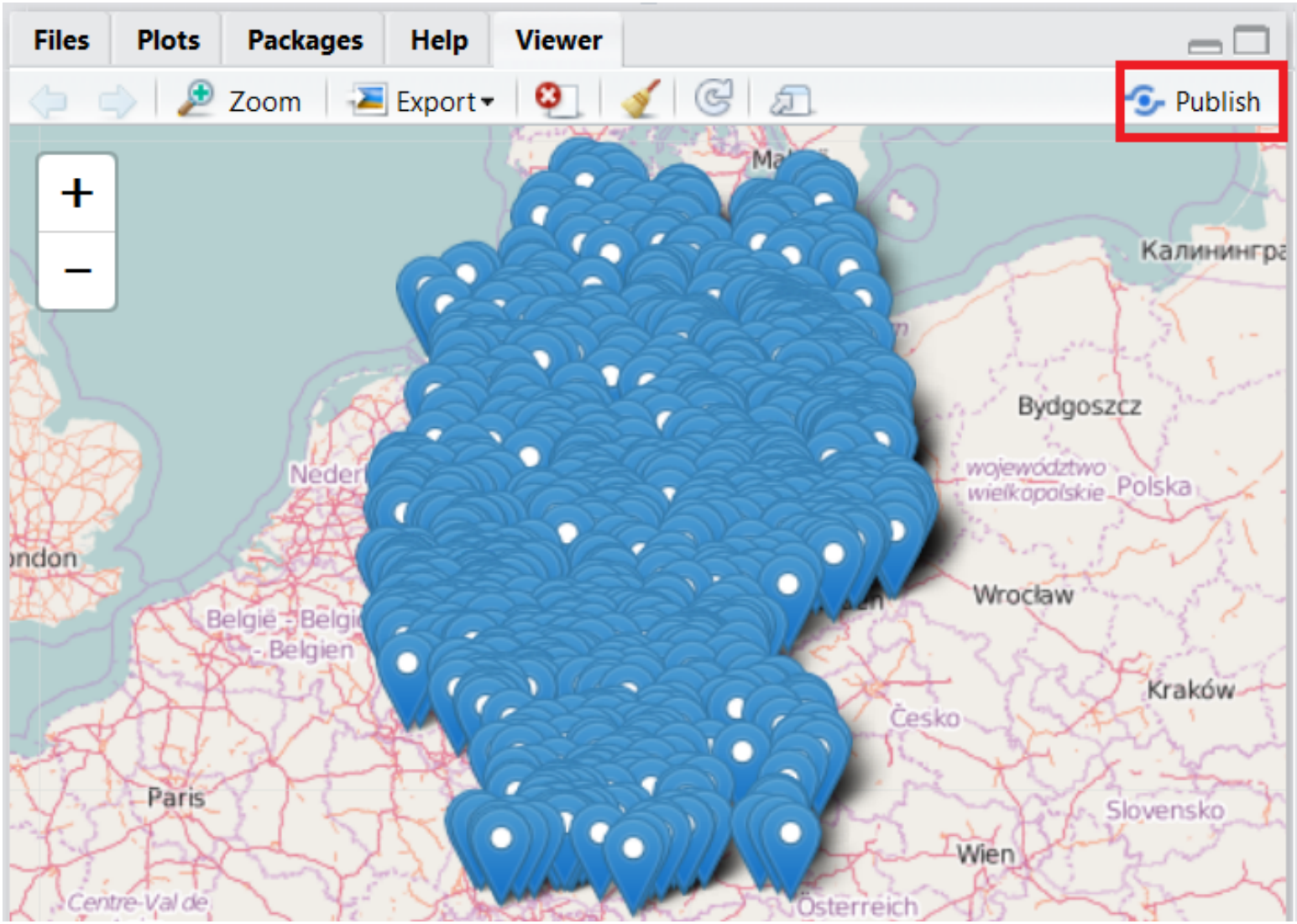
Die resultierende Karte



Popups in einer interactiven Karte

Ich hab die Ergebnisse hochgeladen:

Wie man auf Rpubs publizieren kann



Ein weiteres Beispiel - Weltkulturerbe

```
url <- "https://raw.githubusercontent.com/Japhilko/
GeoData/master/2015/data/whcSites.csv"

whcSites <- read.csv(url)
```

Eine interaktive Karte erstellen

```
m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=whcSites$lon,
             lat=whcSites$lat,
             popup=whcSites$name_en)
m
```

Die Karte zeigen

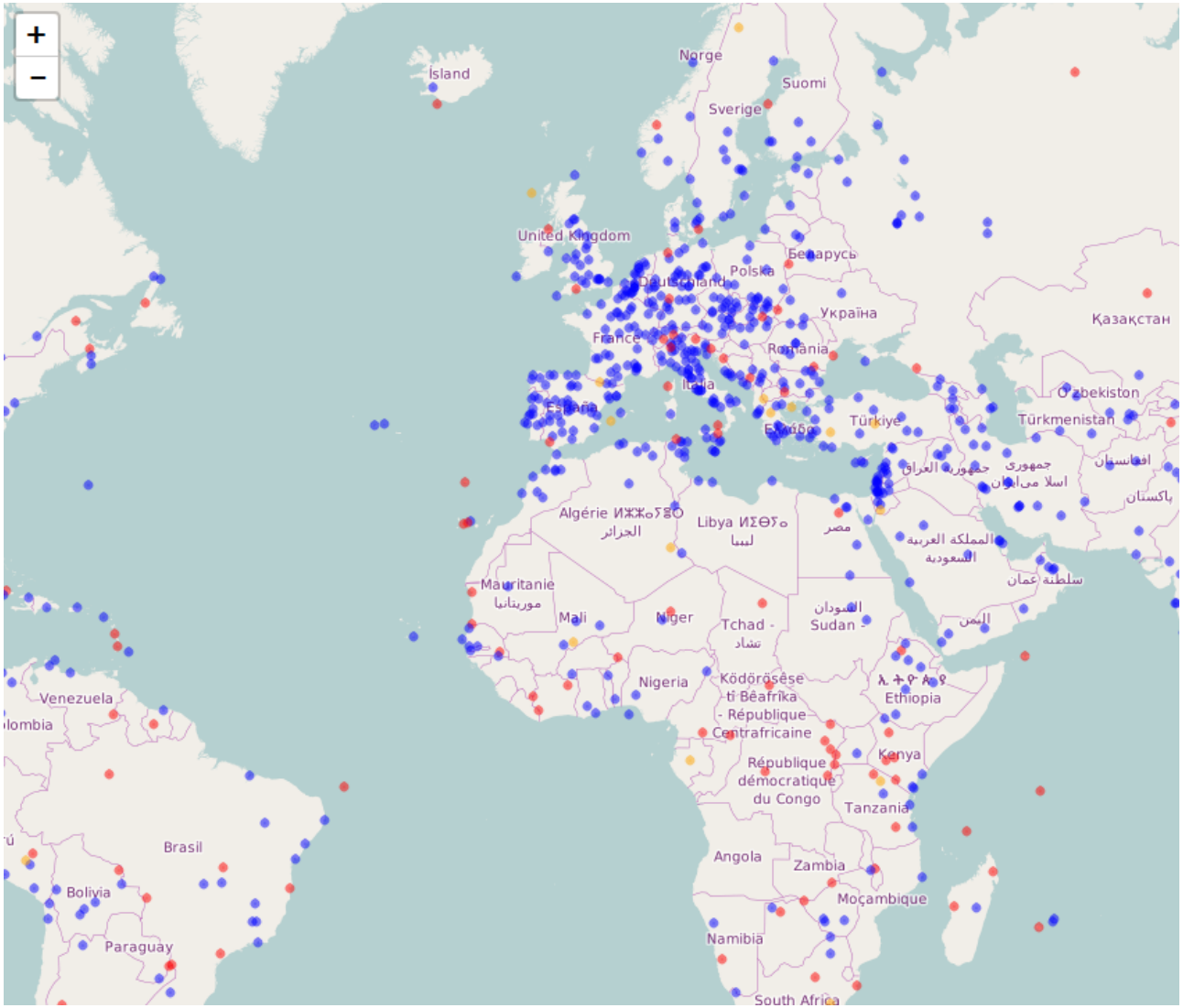
Farbe hinzu

```
whcSites$color <- "red"
whcSites$color[whcSites$category=="Cultural"] <- "blue"
whcSites$color[whcSites$category=="Mixed"] <- "orange"
```

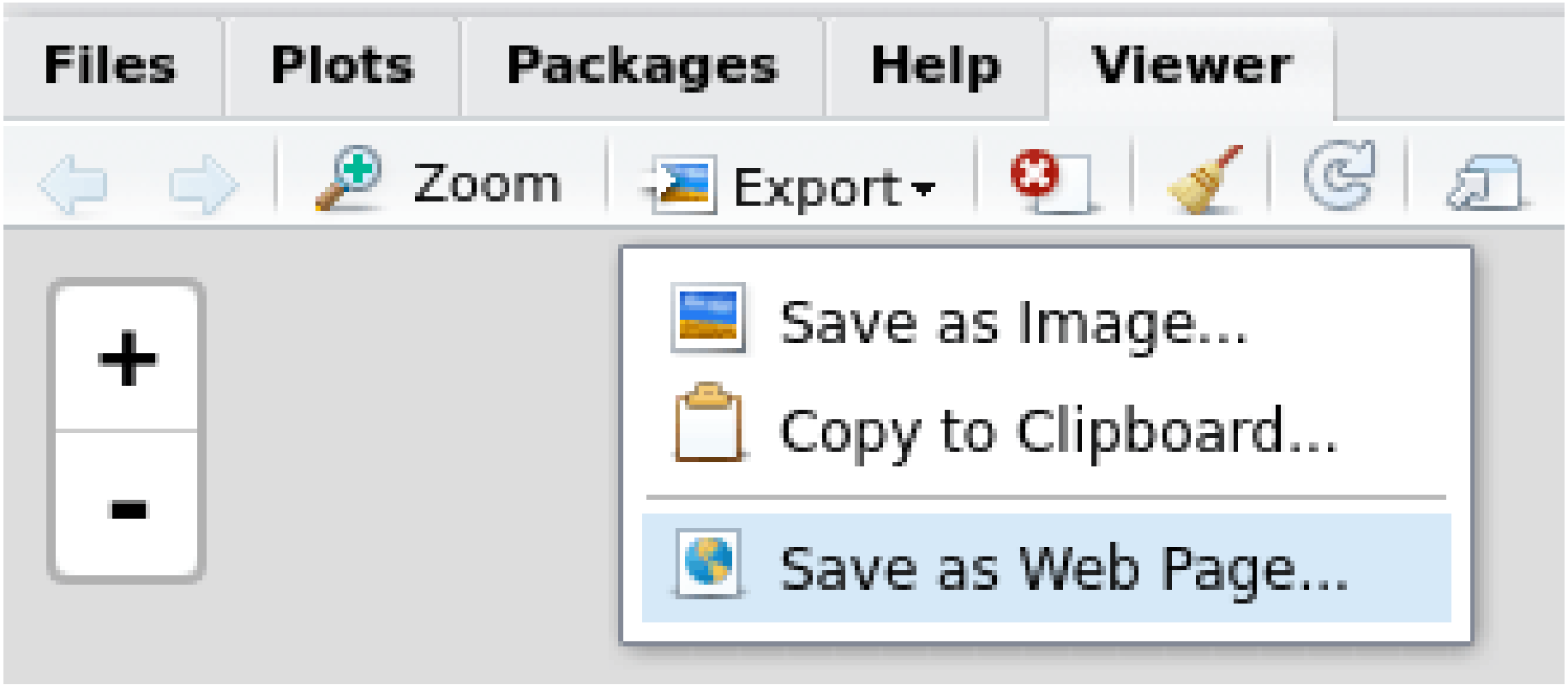
Eine Karte mit Farbe erzeugen

```
m1 <- leaflet() %>%
  addTiles() %>%
  addCircles(lng=whcSites$lon,
             lat=whcSites$lat,
             popup=whcSites$name_en,
             color=whcSites$color)
m1
```

Die Karte zeigen



Die Karte abspeichern



Das Paket mapview - Beispieldatensatz Franken

```
library(mapview)

mapview(franconia)
```

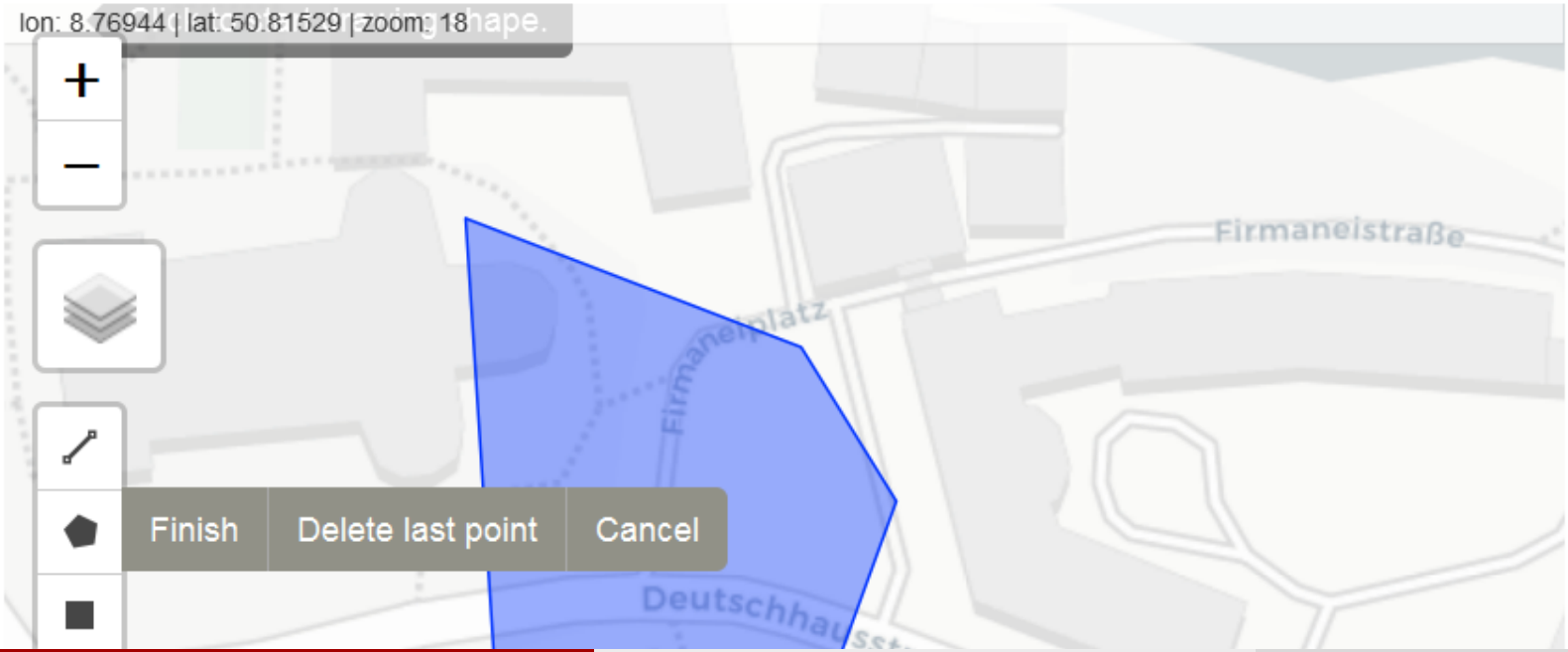
GADM und mapview

```
mapview(leaflet::gadmCHE)
```

Das Paket mapview

```
library(mapedit)
library(magrittr)

lf <- mapview()
drawing <- lf %>%
  editMap()
```



Das Paket mapview

```
load("../data/spatsamp_68239.RData")
```

```
library(mapview)
mapview(spatsamp)
```

Das Paket leaflet

```
library("tmaptools")
gc_tma <- geocode_OSM("Mannheim, GESIS")
```

```
library(leaflet)
library(magrittr)
m <- leaflet() %>%
addTiles() %>%
addMarkers(lng=8.463061 , lat=49.485736 , popup="GESIS Mannhei
m
```

Das Paket geojsonR

- JavaScript Object Notation

```
install.packages("geojsonR")
citation("geojsonR")
```

Wo bekomme ich ein geojson

- Ein **OSM map feature** heraus suchen
- z.B. `key=highway`, `value=bus_stop`
- Auf **Overpass Turbo** gehen und das Objekt herunterladen

```
bus_stops <- geojsonio::geojson_read("../data/Amsterdam_bus_st
  what = "sp")
```

Die Punkte plotten

```
sp::plot(bus_stops)
```

Das Paket `lawn`

- **Vignette** für das Paket `lawn`
- Mit dem Paket `lawn` kann die Javascript-Bibliothek turf.js in R eingebunden werden.
- Weitere genutzte Javascript Bibliotheken (geojson-random und geojsonhint), werden verwendet um GeoJSON-Objekte zufällig zu erzeugen bzw. um die GeoJSON Objekte einzufärben.

```
install.packages("lawn")
citation("lawn")
```

```
library(lawn)
```

Ein zufälliges Beispiel Objekt erstellen

- Mit der Funktion `gr_polygon` kann ein Beispielobjekt erzeugt werden.
- Anschließend kann man sich das Objekt mit der generischen Funktion `view` plotten.

```
a <- gr_polygon(n = 1, vertices = 5, max_radial_length = 5)
view(a)
```

```
b <- gr_polygon(n = 1)
view(b)
```

Interaktive Deutschland Karte

```
gcs <- geojsonio::geojson_read("../data/ddat.geojson")
view(gcs)
```

Das Paket jsonlite

- A Robust, High Performance JSON Parser and Generator for R

```
library(jsonlite)
geoc <- read_json("../data/ddat.geojson")
```

```
citation("jsonlite")
```

```
##
## To cite jsonlite in publications use:
##
##   Jeroen Ooms (2014). The jsonlite Package: A Practical and
##   Consistent Mapping Between JSON Data and R Objects.
##   arXiv:1403.2805 [stat.CO] URL https://arxiv.org/abs/1403.
##
## A BibTeX entry for LaTeX users is
##
```

Das Paket RJSONIO

```
library("RJSONIO")
con <- url("http://nominatim.openstreetmap.org/search?format=j
addressdetails=1&extratags=1&q=Amsterdam+Niederlande+Rozengrac
geoc <- fromJSON(paste(readLines(con,warn=F),
                       collapse = ''))
close(con)
```

Links und Quellen

- Rbloggers Artikel zu Leaflet
- Einführung in Leaflet für R
- Offline Karten mit RgoogleMaps und leaflet
- github Ordner für das Iwan Paket

Themen dieses Abschnitts

- Der Import von Geodaten mit dem Paket simple features (`sf`).
- Die Verarbeitung der OSM-Daten mit dem Paket `sf`.
- Die Daten visualisieren mit `sf`

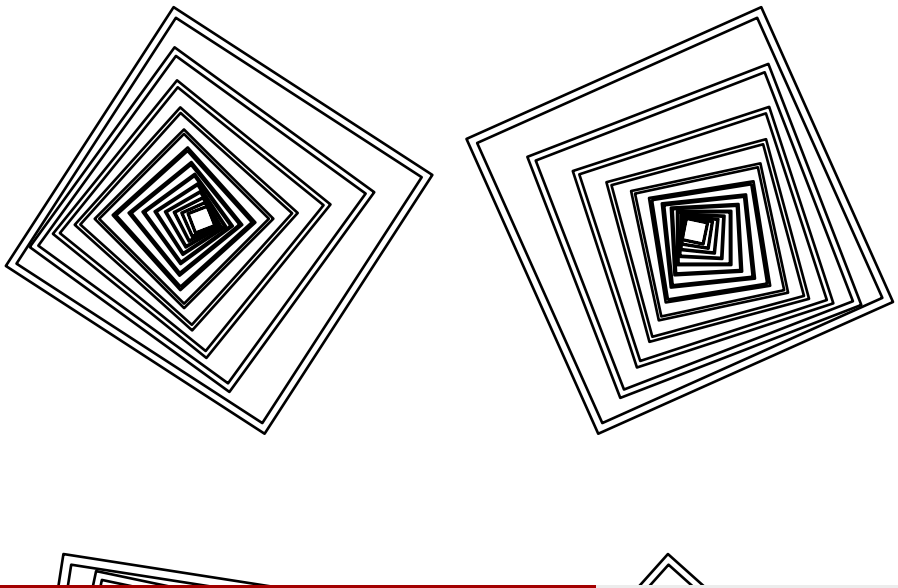
Das Paket `sf`

Simple Features for R

```
library(sf)
```

- Ein Demo ist im Paket `sf` integriert

```
demo(sf::affine)
```



Beispieldaten bekommen

```
library(osmdata)
bb_poly <- getbb(place_name = "Amsterdam",
                 format_out = "polygon")
```

```
ls <- st_multilinestring(bb_poly)
```

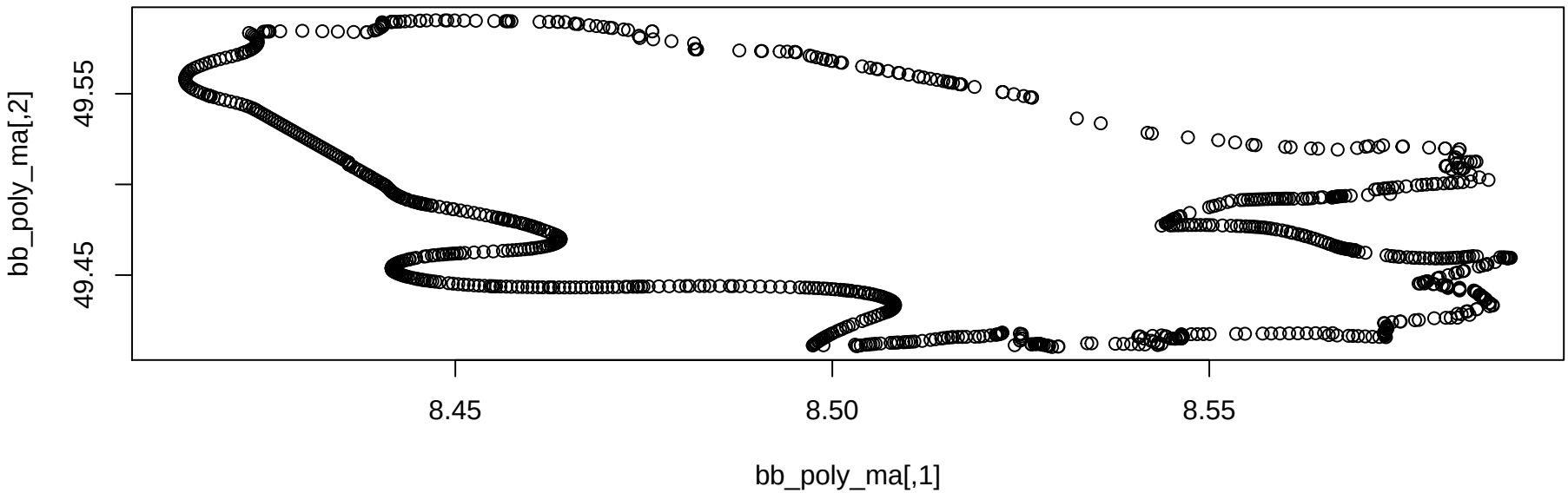
```
pol <- sf::st_polygon(bb_poly)
class(pol)
```

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

```
bb_poly_ma <- getbb(place_name = "Mannheim",
                    format_out = "polygon")
```

Das Ergebnis plotten

```
plot(bb_poly_ma)
```



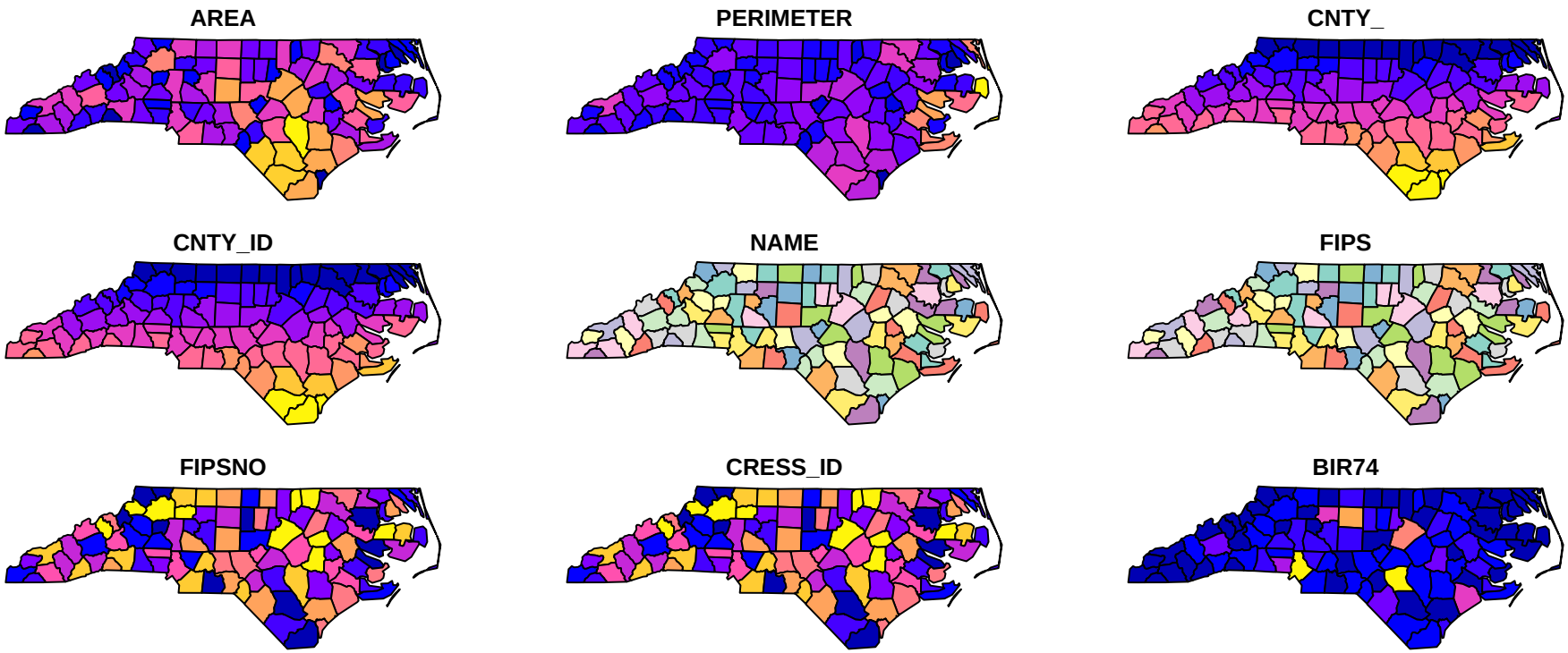
Ein Beispieldatensatz

```
demo(nc, ask = FALSE, echo = FALSE)
```

```
## Reading layer `nc.gpkg' from data source `D:\Programme\R-3.
## Simple feature collection with 100 features and 14 fields
## Attribute-geometry relationship: 0 constant, 8 aggregate, 6
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -84.32385 ymin: 33.88199 xmax: -75.45
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
```

Graphiken mit `sf`

```
plot(nc)
```



Shapefiles mit `sf` importieren

```
lon <- st_read("../data/london_sport.shp")
```

```
## Reading layer `london_sport' from data source `D:\Daten\Git
## Simple feature collection with 33 features and 4 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 503571.2 ymin: 155850.8 xmax: 561941.
## epsg (SRID):    NA
## proj4string:    +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.99960
```

Das Shapefile plotten

```
plot(lon$geometry)
```

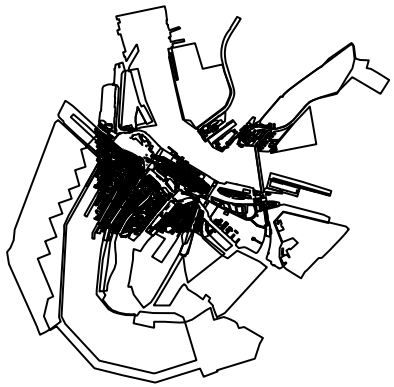
Daten vom Amsterdam Beispiel

```
datm <- st_read("../data/ams_centraal.osm","multipolygons")
```

```
## Reading layer `multipolygons' from data source `D:\Daten\Gi
## Simple feature collection with 2796 features and 25 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 4.874776 ymin: 52.36088 xmax: 4.92975
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

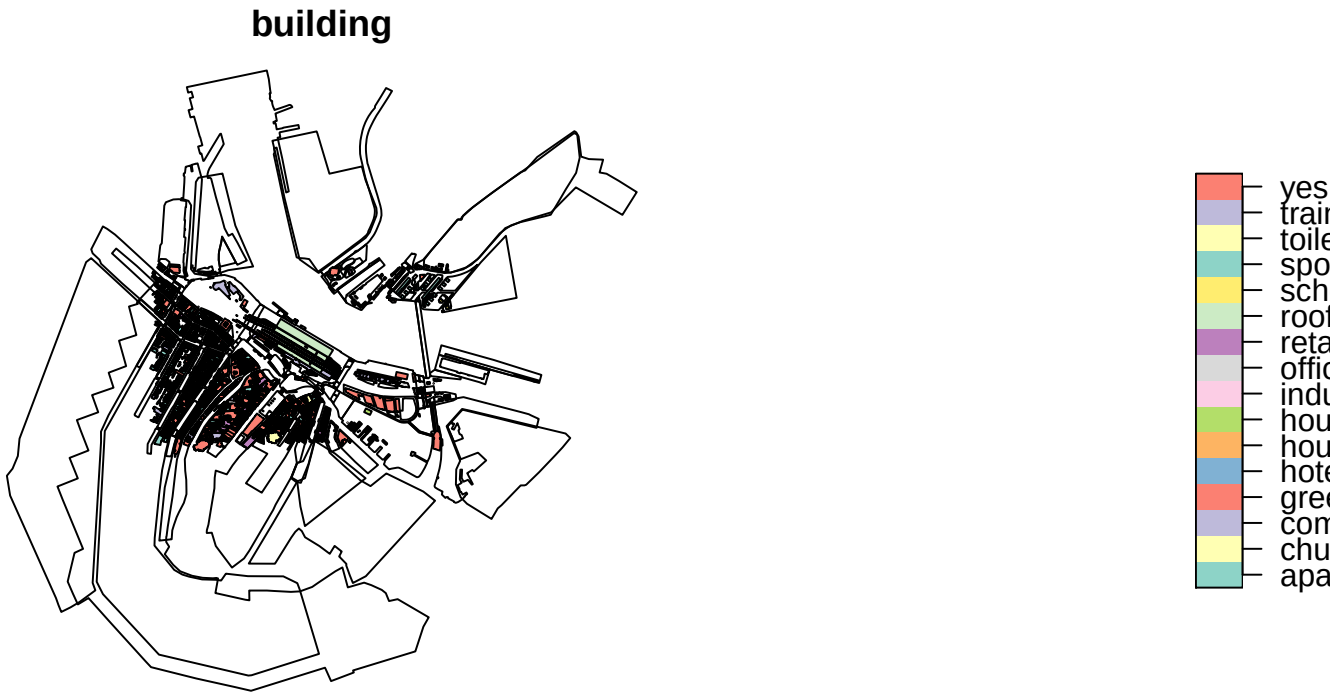
Die Funktion st_geometry

```
geom_datm <- st_geometry(datm)
plot(geom_datm)
```



Die Häuser auswählen

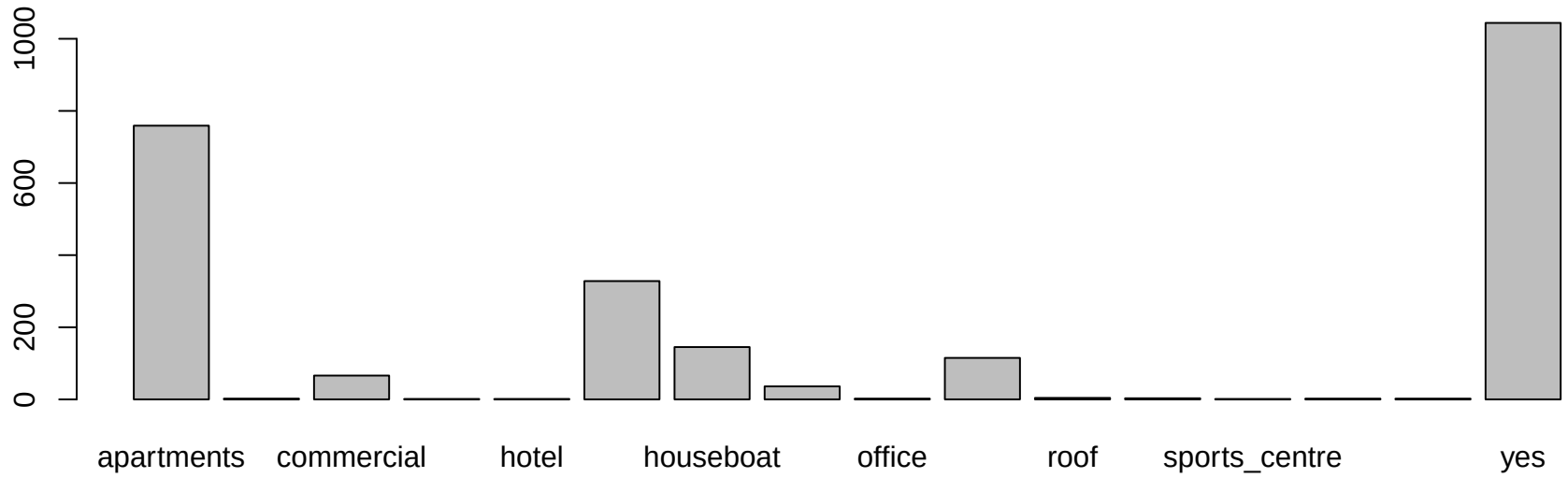
```
library(dplyr)
buis <- datm %>% select(building)
plot(buis)
```



Welche Häusertypen gibt es?

```
buis2 <- datm %>% as.data.frame %>% select(building)
```

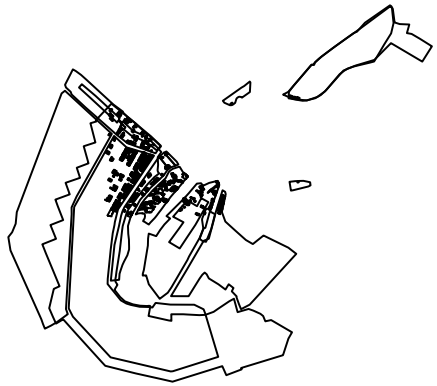
```
datbuis <- datm[, "building", drop = TRUE]
plot(datbuis)
```



```r
houses <- datm[datm$building == "house",]
class(houses)
```

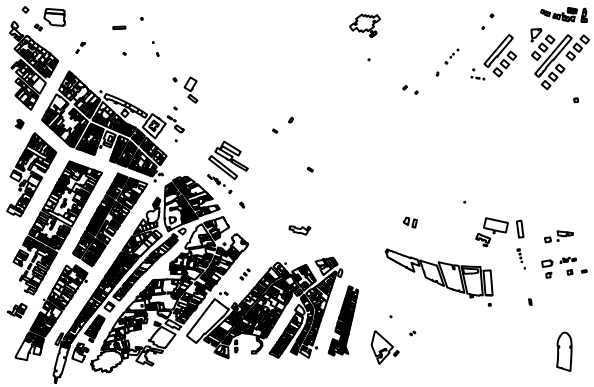
```
## [1] "sf"         "data.frame"
```

```r
## [1] "sf"         "data.frame"
dhous <- datm[houses,]
plot(dhous$geometry)
```



Alle Häuser herausnehmen

```
houses <- datm[datm$building %in% c("house","yes",
                                    "apartments"),]
plot(st_geometry(houses))
```



Die Vignetten für das Paket `sf`

https://r-spatial.github.io/sf/reference/st_as_sf.html

https://r-spatial.github.io/sf/reference/st_read.html

https://r-spatial.github.io/sf/articles/sf1.html