

# **Geodaten - erster Teil**

Jan-Philipp Kolb

11 Oktober 2018

# Das Thema Geodatenlandschaft

## Georeferenzierung von Daten

Situation und Zukunft der  
Geodatenlandschaft in  
Deutschland

Herausgegeben vom Rat für Sozial- und Wirtschaftsdaten

RatSWD.

Rat für Sozial- und  
WirtschaftsDaten

GEFÖRDERT VOM

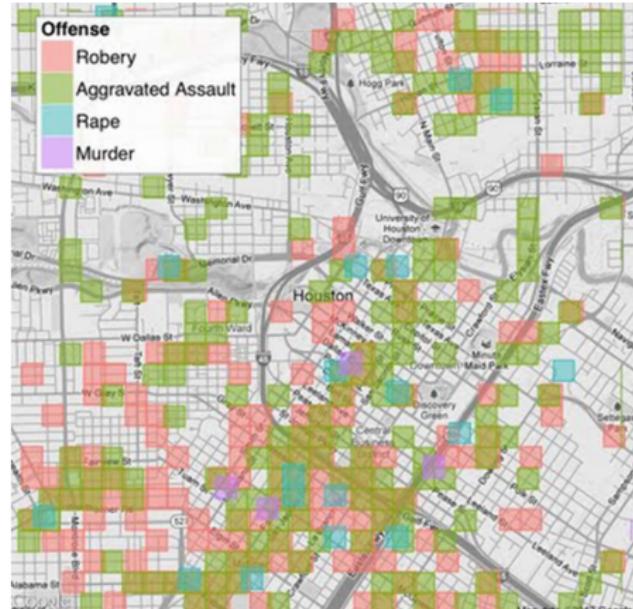
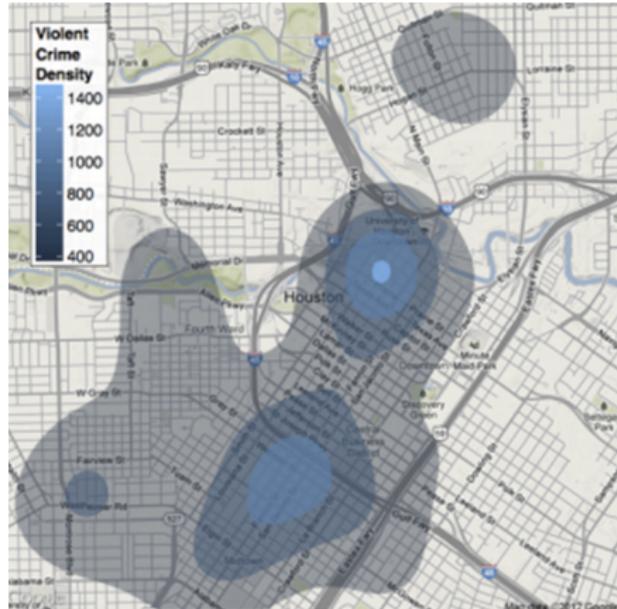


Bundesministerium  
für Bildung  
und Forschung

## R-Pakete - Zum Download von Geo-Information

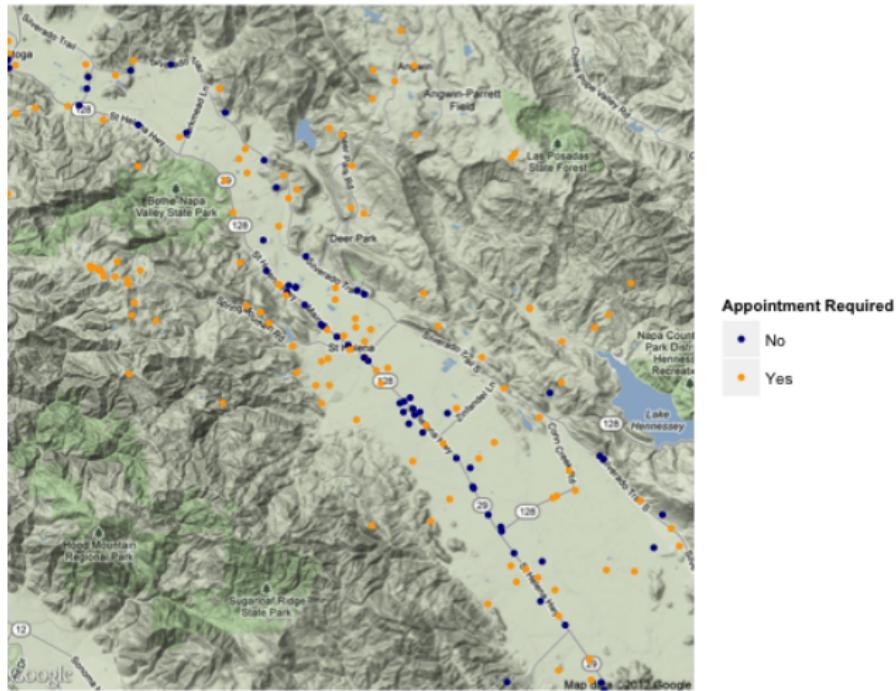
## Das Paket ggmap

- David Kahle and Hadley Wickham: `ggmap` - Spatial Visualization with `ggplot2`



# Worum geht es?

## Weine probieren im Napa Valley?



# Ziel dieses Kurses

## Vorgestellt werden:

- Möglichkeiten für den Download, den Import, die Verarbeitung und die Visualisierung von Geodaten
- Die wichtigsten Programmierschnittstellen (APIs) um die Daten zu bekommen
- R-Pakete um diese Daten zu verarbeiten und zu visualisieren

# Motivation

- Sekundäranalyse für bestehenden Daten
- Raumbezug herstellen/nutzen
- Analysepotentiale der Geokodierung vorstellen
- Verbindung von sozial- mit raumwissenschaftlichen Daten

# Laws of Spatial Science

## Tobler's law

*everything is related to everything else, but near things are more related than distant things.*

## Spatial Turn

*Spatial turn is a term used to describe an intellectual movement that places emphasis on place and space in social science and the humanities.*

# Ergebnisse des Zensus 2011 zum Download



## Gemeindeebene

- Bevölkerung nach Geschlecht, Altersgruppe, Familienstatus, Staatsangehörigkeit und Religion

## 1 km<sup>2</sup> Raster

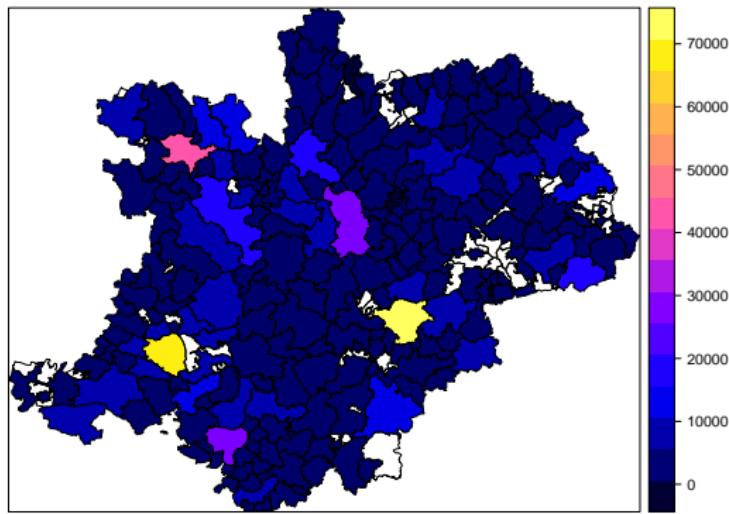
- Bevölkerung, Leerstandsquote, Wohnfläche und Haushaltsgröße

## 100 m<sup>2</sup> Raster

Bevölkerung

# Zensus Ergebnisse

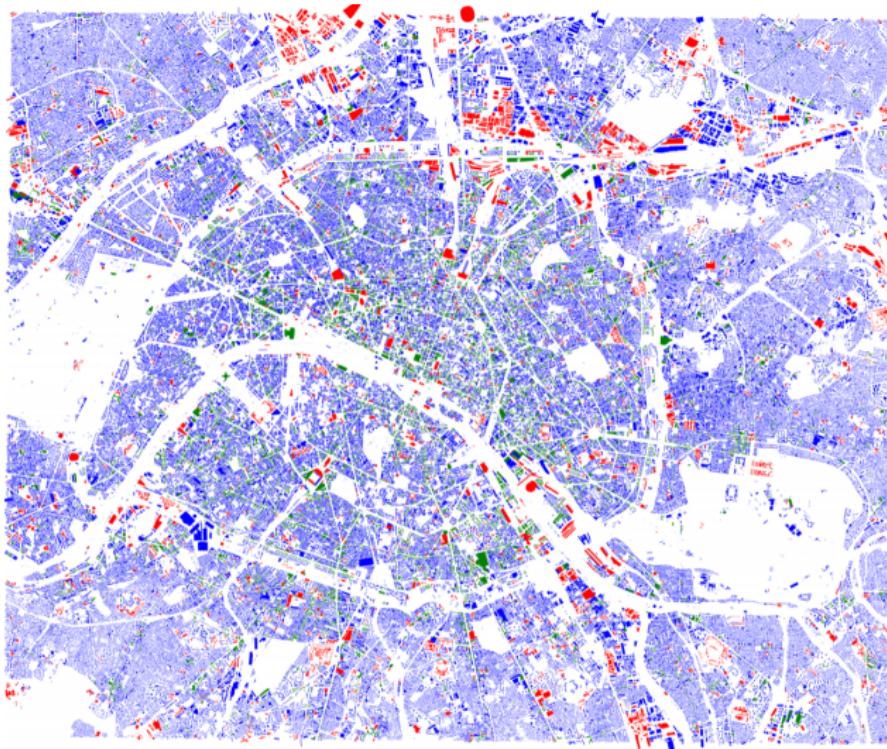
Beispiel Anteil der Personen aus EU27 Land an Einwohnerzahl pro Gemeinde in Oberfranken



# Motivation - Warum die Darstellung in Karten

- Darstellung in Karten ermöglicht besseres Verständnis bspw. sozialwissenschaftlicher Phänomene.
- Attraktiver Output
- Durch die INSPIRE Richtlinie und *Collaborative Mapping* wächst der verfügbare Bestand an Geodaten.
- Daten sind oft frei verfügbar im Internet (z.B. durch die Nutzung von APIs)
- Die Daten sind allerdings oft wenig oder gar nicht strukturiert (z.B. Internet Dokumente), heterogen und
- meistens nicht für die Nutzung zur räumlichen Visualisierung vorgesehen, beinhalten aber implizit geographische Informationen (Web 2.0)
- Oftmals sind wenig oder keine Metadaten vorhanden

# Openstreetmap Projekt



# Das Openstreetmap Projekt...

- Durch kollaboratives Mapping ist eine riesige Datenmenge zugänglich.
- Viele Menschen tragen jeden Tag Informationen bei.
- ... ermöglicht Zugang zu Big Data der Geographie.
- Die wachsende Menge an Geodaten wird von Freiwilligen gesammelt oder über Crowd-sourcing gewonnen.

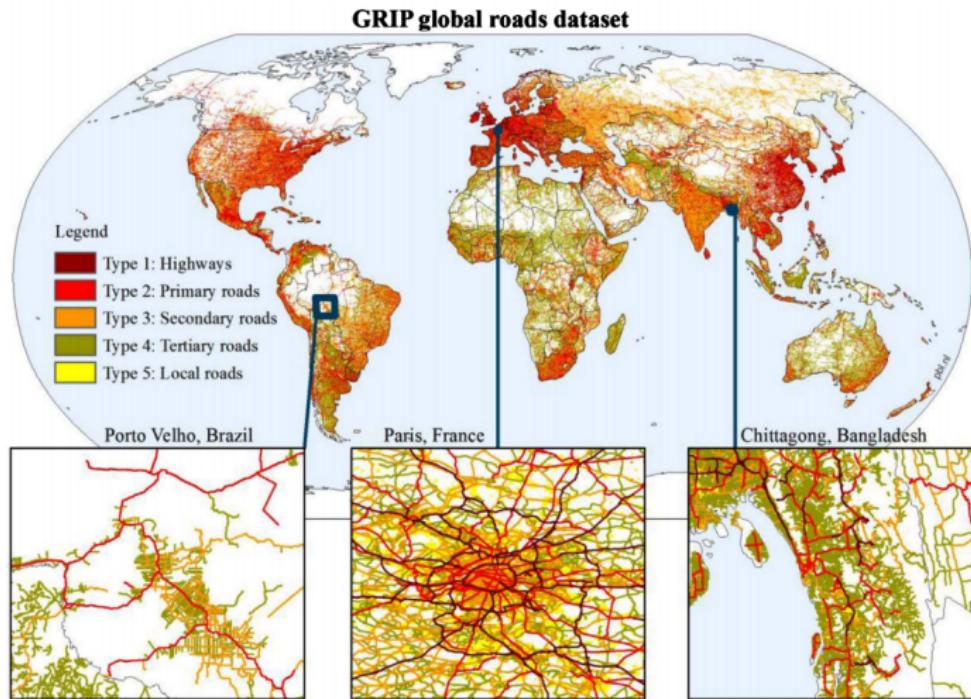
# Was ist das Ziel - Straßen in Berlin

Dargestellt werden OpenStreetMap Daten, die mit der Overpass API heruntergeladen wurden.



# Globale Muster der Straßeninfrastruktur

Studie von Johan Meijer, Mark AJ Huijbregts, Kees Schotten, und Aafke Schipper



# Links mit Beispielen

- Shiny App zu **Indikatoren** für Europa
- Räumliche Visualisierung in den USA - **Walmarts in den USA**
- **Race Gap Police USA - Wahl USA**
- Zeit Artikel zum Zustand der **Eisenbahnbrücken**
- **Fahrradunfälle** in Berlin
- **Verteilung Fußballfans**
- **Plastiktüten im Meer**

## Datenquellen:

- Datensätze zu **Pegelständen** in Deutschland
- Viele Datensätze auf **driven by data**

## Resourcen

- Andreas Plank - **Grafiken und Statistik in R**

# Inhalt dieses Abschnitts

Arten von räumlichen Daten:

- **Straßenkarten**
- **Satelliten Bilder**
- **Physische Daten und Karten**
- **Abstrakte Karten**
- ...

Das R-paket `ggmap` wird im folgenden genutzt um verschiedene Kartentypen darzustellen.

Mit `qmap` kann man eine schnelle Karte erzeugen.

# Installieren des Paketes

- Zur Erstellung der Karten brauchen wir das Paket ggmap:

```
devtools::install_github("dkahle/ggmap")
devtools::install_github("hadley/ggplot2")
install.packages("ggmap")
```

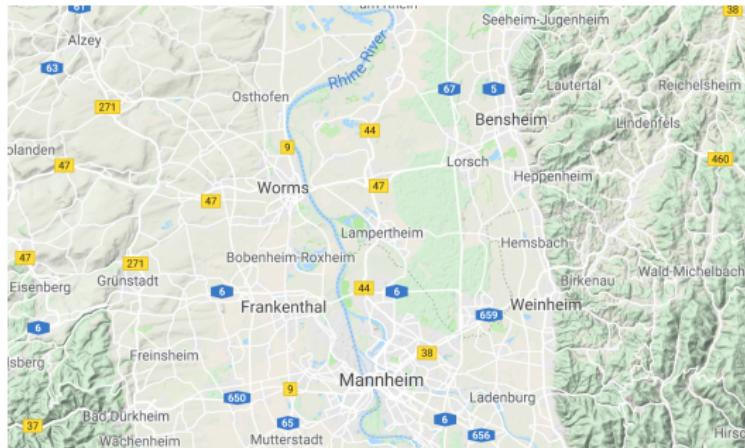
# Paket ggmap - Hallo Welt

- Um das Paket zu laden verwenden wir den Befehl library

```
library(ggmap)
```

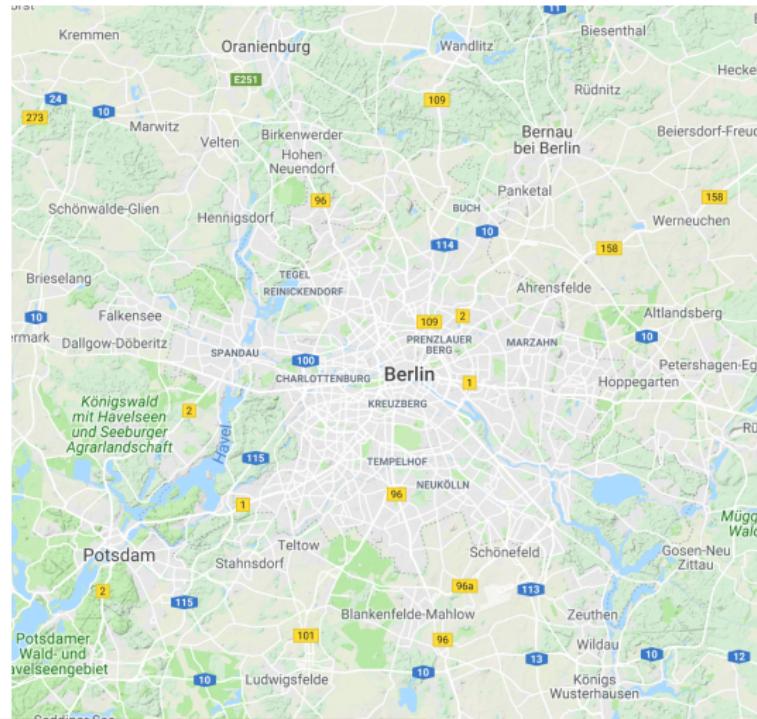
Und schon kann die erste Karte erstellt werden:

```
qmap("Mannheim")
```



# Karte für eine Sehenswürdigkeit

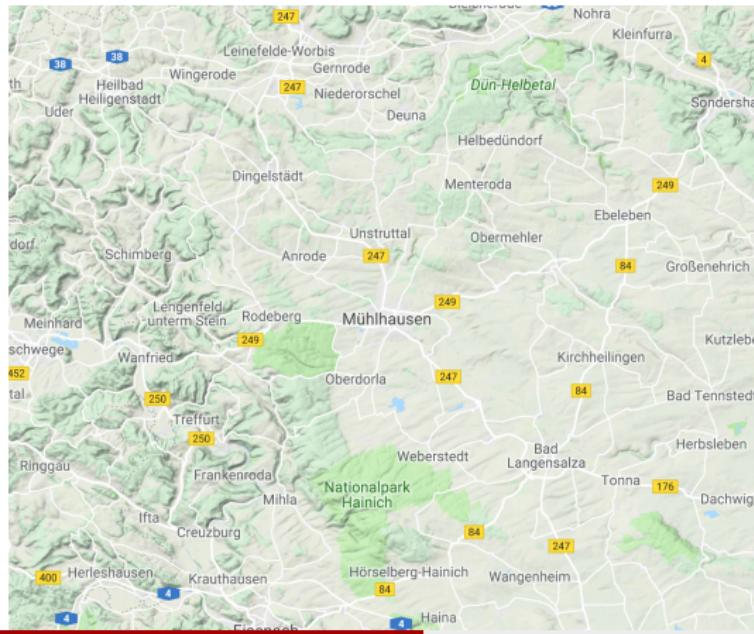
`qmap("Berlin Brandenburger Tor")`



# Karte für einen ganzen Staat

```
qmap("Germany")
```

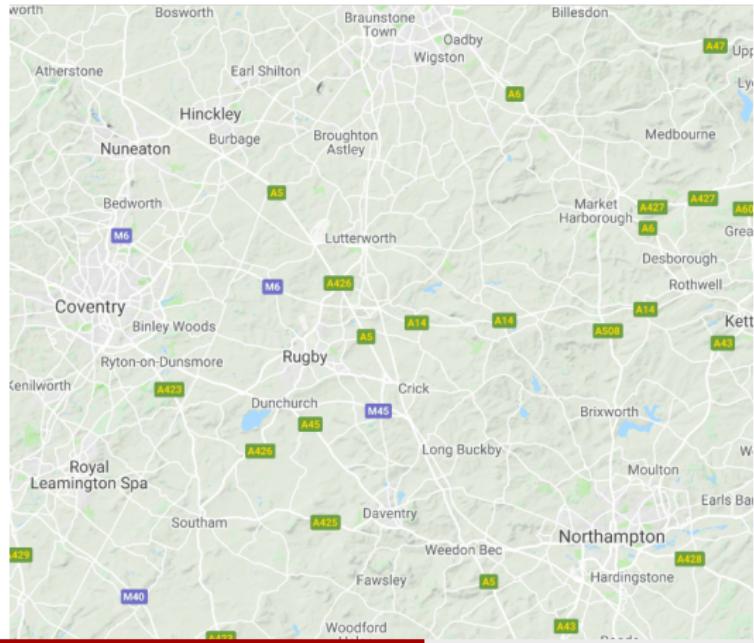
- Wir brauchen ein anderes *zoom level*



# Ein anderes *zoom level*

- level 3 - Kontinent / level 10 - Stadt / level 21 - Gebäude

```
qmap("England", zoom = 6)
```



# Hilfe bekommen wir mit dem Fragezeichen

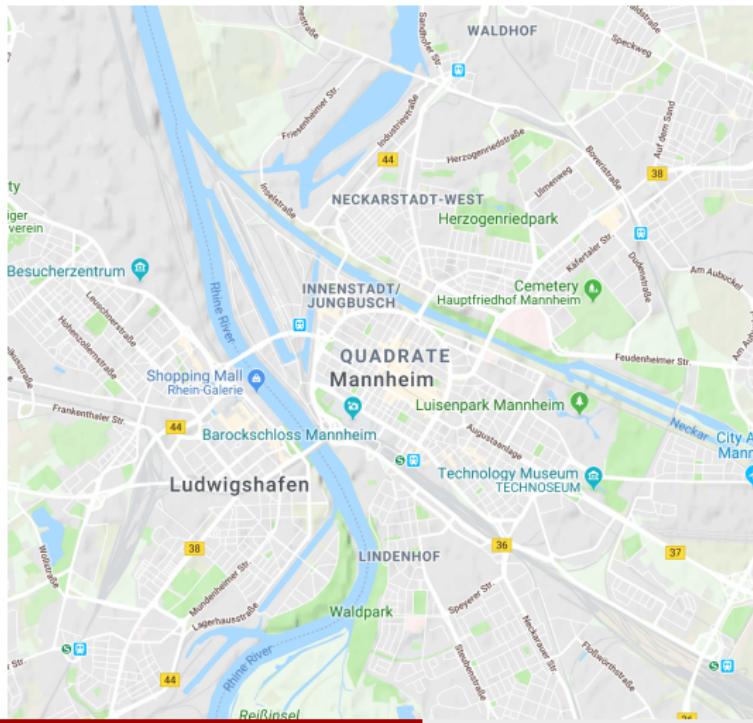
?qmap

Verschiedene Abschnitte in der Hilfe:

- Description
- Usage
- Arguments
- Value
- Author(s)
- See Also
- Examples

# Ganz nah dran

```
qmap('Mannheim', zoom = 20)
```



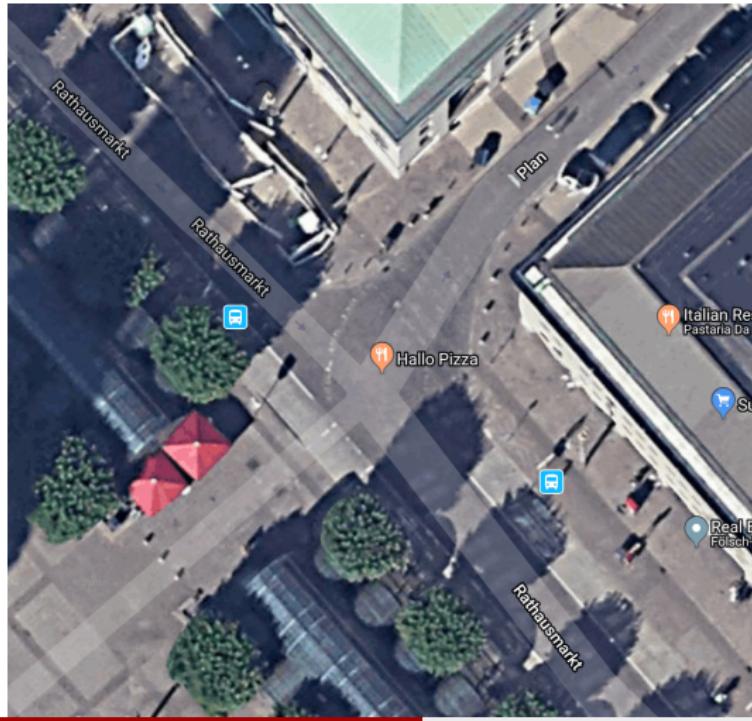
## ggmap - maptype satellite

```
qmap('Hamburg', zoom = 14, maptype="satellite")
```



# ggmap - maptype satellite zoom 20

```
qmap('Hamburg', zoom = 20, maptype="hybrid")
```



# Terrain/physical maps

- Aus Physischen Karten kann man Informationen über Berge, Flüsse und Seen ablesen.
- Farben werden oft genutzt um Höhenunterschiede zu visualisieren

```
qmap('Arequipa', maptype="terrain")
```

# Eine physische Karte von Arequipa



# Abstrahierte Karten (<http://www.designfaves.com>)



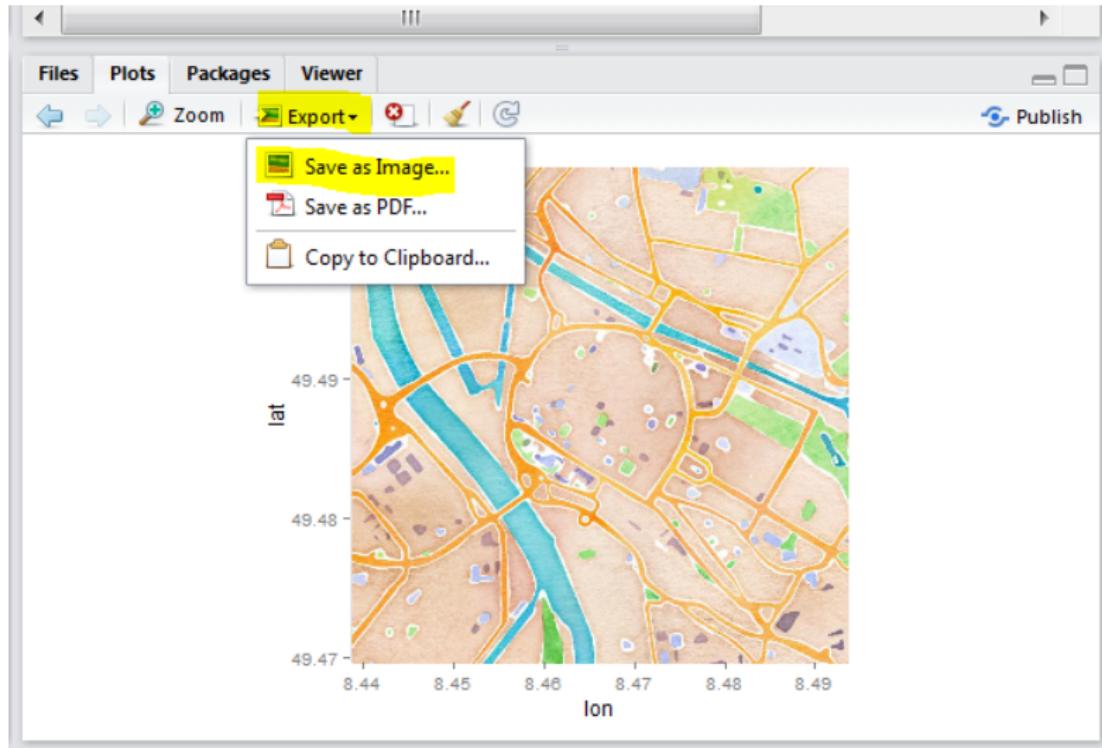
- Abstraktion wird genutzt um nur die essentiellen Informationen einer Karte zu zeigen.
- Bsp. U-Bahn Karten - wichtig sind Richtungen und wenig Infos zur Orientierung

# ggmap - maptype watercolor

```
qmap('Los Angeles', zoom = 14,  
      maptype="watercolor",source="stamen")
```



# Graphiken speichern



# ggmap - ein Objekt erzeugen

- <- ist der Zuweisungspfeil um ein Objekt zu erzeugen
- Dieses Vorgehen macht bspw. Sinn, wenn mehrere Karten nebeneinander gebraucht werden.

```
MA_map <- qmap('Mannheim',
                 zoom = 14,
                 maptype="toner",
                 source="stamen")
```

# Eine Karte für die USA

- Mit dem Befehl OSM\_scale\_lookup bekommt man heraus, welchen Wert man für scale angeben muss.

```
OSM_scale_lookup(zoom = 10)
qmap(location = "Trier", zoom = 10, source = "osm",
      scale=575000)
```

# Cheatsheet

## • Cheatsheet zu data visualisation

<https://www.rstudio.com/>

### Data Visualization with ggplot2

Cheat Sheet



#### Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build even the most complex plots from the same few components: a **data set**, a set of **geom**s—visual marks that represent data points, and a **coordinate system**.

#### Geoms

Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

##### One Variable

###### Continuous

`a <- ggplot(mpg, aes(hwy))`



###### Frequentist

`b <- ggplot(mpg, aes(wt))`



###### Discrete

`c <- ggplot(mpg, aes(cty))`



##### Graphical Primitives

###### Continuous X, Continuous Y

`d <- ggplot(economics, aes(date, unemploy))`



###### Discrete X, Continuous Y

`e <- ggplot(mpg, aes(displ, hwy))`



###### Discrete X, Discrete Y

`f <- ggplot(diamonds, aes(cut, color))`



##### Two Variables

###### Continuous X, Continuous Y

`T <- ggplot(mtcars, aes(wt, mpg))`



###### Quantitative

`g <- geom_quantile(aes(ymin = q, ymax = p), 3)`



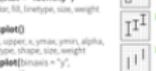
###### Rug

`h <- geom_rug(aes(x = x, xbreaks = 100))`



###### Smooth

`i <- geom_smooth(method = lm)`



###### Text

`j <- geom_text(aes(label = y))`



###### Step

`k <- geom_step(direction = "thru")`



##### Continuous Bivariate Distribution

`L <- geom_bivar(binsize = c(0.5, 0.5))`



###### Density 2D

`M <- geom_density2d()`



###### Hex

`N <- geom_hex()`



##### Function

`J <- ggplot(economics, aes(date, unemploy))`



###### Area

`O <- data.frame(date = c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"), unemploy = c(10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295, 300, 305, 310, 315, 320, 325, 330, 335, 340, 345, 350, 355, 360, 365, 370, 375, 380, 385, 390, 395, 400, 405, 410, 415, 420, 425, 430, 435, 440, 445, 450, 455, 460, 465, 470, 475, 480, 485, 490, 495, 500, 505, 510, 515, 520, 525, 530, 535, 540, 545, 550, 555, 560, 565, 570, 575, 580, 585, 590, 595, 600, 605, 610, 615, 620, 625, 630, 635, 640, 645, 650, 655, 660, 665, 670, 675, 680, 685, 690, 695, 700, 705, 710, 715, 720, 725, 730, 735, 740, 745, 750, 755, 760, 765, 770, 775, 780, 785, 790, 795, 800, 805, 810, 815, 820, 825, 830, 835, 840, 845, 850, 855, 860, 865, 870, 875, 880, 885, 890, 895, 900, 905, 910, 915, 920, 925, 930, 935, 940, 945, 950, 955, 960, 965, 970, 975, 980, 985, 990, 995, 1000))`



##### Crossbar

`P <- geom_crossbar(aes(ymin = l, ymax = u), 3)`



##### Errorbar

`Q <- geom_errorbar(aes(ymin = l, ymax = u), width = 1)`



##### Linearrange

`R <- geom_linearrange(aes(l, u), 3)`



##### Pointrange

`S <- geom_pointrange(aes(l, u), 3)`



##### Maps

`data <- data.frame(state = USArrests$Murder,`

`state = USArrests$Assault,`

`map = map_data("us_states")`

`map <- map_data("us_states")`

`map <- map_data("us_states") +`

`expand_limits(x = map$x, y = map$y)`

`map_id, alpha, color, fill, linetype, size`



##### Contour

`T <- geom_contour(aes(z = z))`



##### Raster

`m <- geom_raster(data = z, height = 0.5,`

`width = 0.5, interpolate = FALSE)`

`x, y, alpha, fill`



Build a graph with `spplot()` or `ggplot()`

`ggplot(mpg, aes(hwy, cyl, color = cty)) +`  
`geom_point()`

Creates a complete plot with given data, geom, and mappings. Supports many useful details.

`ggplot(mpg, aes(cty, cyl, fill = hwy)) +`  
`geom_point()`

Begins a plot that you finish by adding layers to. No details, but provides more control than `spplot()`.

`ggplot(mpg, aes(hwy, cyl, fill = cty)) +`  
`geom_point() +`  
`add_inset(mpg, aes(cty, cyl, fill = hwy),`  
`size = 10, border = "black",`  
`stroke = "black", angle = 45)`

Add a new layer to a plot with a geom, `"f"` or `"stat"`, `"Q"`-functions. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

`last_plot()`  
Returns the last plot  
`ggplot("plot.png", width = 5, height = 5)`

# Resourcen und Literatur

- Artikel von **David Kahle und Hadley Wickham** zur Nutzung von **ggmap**.
- **Schnell eine Karte bekommen**
- **Karten machen mit R**

# Das Paket **tmap**

- Mit dem Paket **tmap** kann man thematische Karten erzeugen
- Die folgenden Beispiele sind auf der **Vignette** des Paketes basiert.

```
install.packages("tmap")
```

```
library(tmap)
```

# Schnelle thematische Karte

- Mit dem Befehl **qtm** kann man eine schnelle thematische Karte erzeugen
- Beispiel aus der **Vignette** zum Paket **tmap**

```
data(Europe)  
qtm(Europe)
```



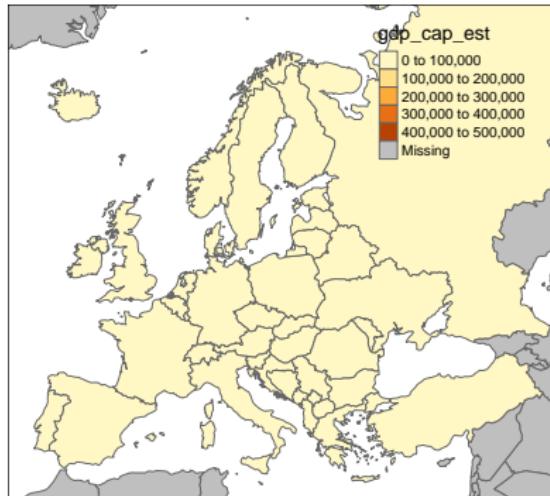
# Der Europa-Datensatz

	iso_a3	name	sovereign	continent
5	ALB	Albania	Albania	Europe
6	ALA	Aland	Finland	Europe
7	AND	Andorra	Andorra	Europe
10	ARM	Armenia	Armenia	Asia
17	AUT	Austria	Austria	Europe
18	AZE	Azerbaijan	Azerbaijan	Asia
20	BEL	Belgium	Belgium	Europe
24	BGR	Bulgaria	Bulgaria	Europe
27	BIH	Bosnia and Herz.	Bosnia and Herzegovina	Europe
29	BLR	Belarus	Belarus	Europe

# Um mehr Farbe in die Karte zu bekommen

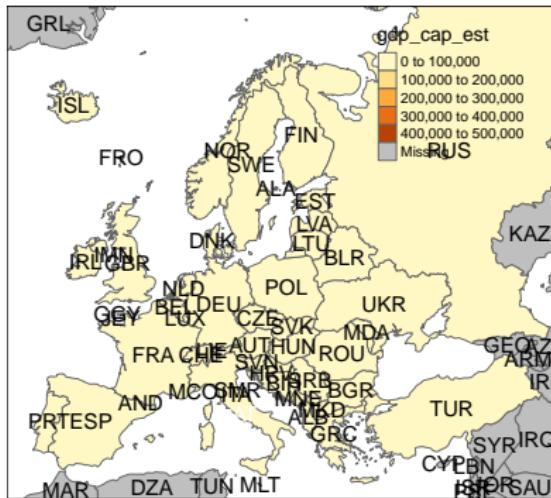
- Visualisierung von **Natural Earth** Daten

```
qtm(Europe, fill="gdp_cap_est")
```



# Eine Karte mit Text

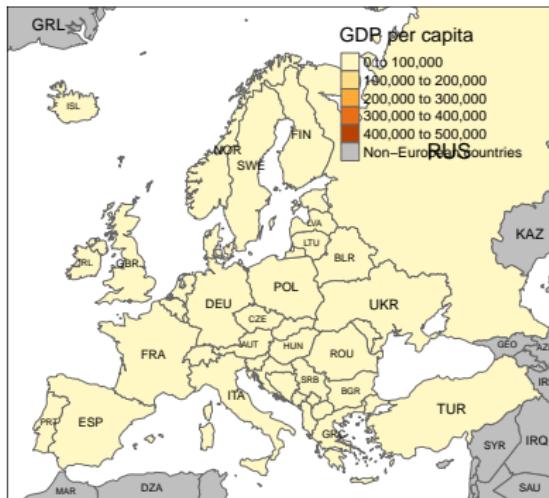
```
qtm(Europe, fill="gdp_cap_est", text="iso_a3")
```



# Dieses Schema passt besser:

## Bevölkerungsdichte

```
qtm(Europe, fill="gdp_cap_est", text="iso_a3",
     text.size="AREA", root=5, fill.title="GDP per capita",
     fill.textNA="Non-European countries", theme="Europe")
```



# Themen des Europa-Datensatzes

- ISO Klassifikation
- Ländername
- Ist das Land Teil Europas?
- Fläche, Bevölkerung, Bevölkerungsdichte,
- **Bruttoinlandsprodukt**
- Bruttoinlandsprodukt **zu Kaufkraftparitäten**
- Ökonomie, Einkommensgruppe

# Der Europa Datensatz - Variablen und was dahinter steckt

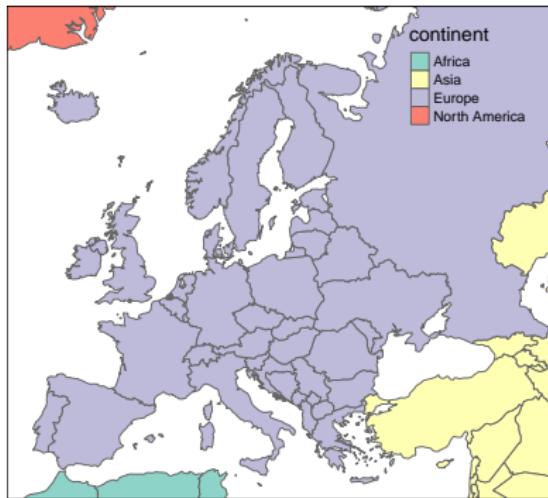
---

	iso_a3	name	sovereign	continent	part
5	ALB	Albania	Albania	Europe	Southern Europe
6	ALA	Aland	Finland	Europe	Northern Europe
7	AND	Andorra	Andorra	Europe	Southern Europe
10	ARM	Armenia	Armenia	Asia	NA
17	AUT	Austria	Austria	Europe	Western Europe
18	AZE	Azerbaijan	Azerbaijan	Asia	NA
20	BEL	Belgium	Belgium	Europe	Western Europe
24	BGR	Bulgaria	Bulgaria	Europe	Eastern Europe

---

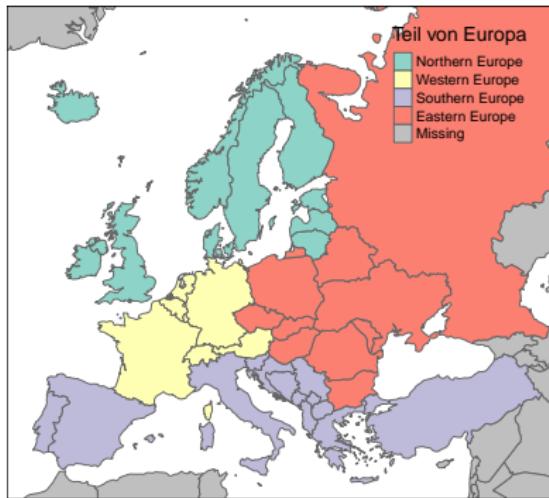
# Die Variable continent

```
qtm(Europe, fill="continent")
```



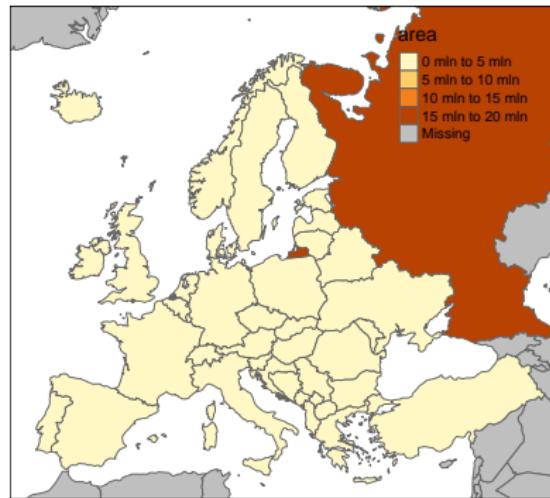
# Die Variable part

```
qtm(Europe, fill="part",fill.title="Teil von Europa")
```



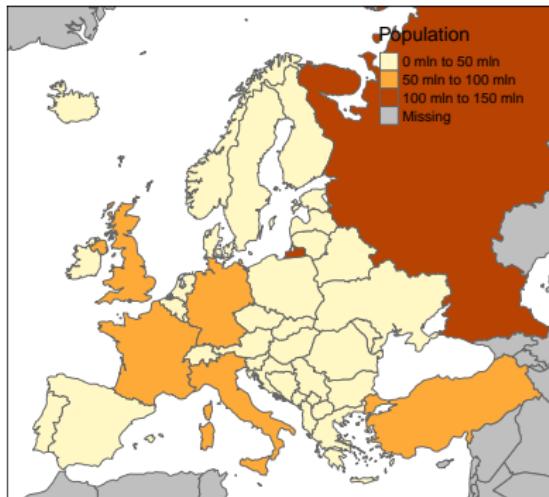
## Die Variable area

```
qtm(Europe, fill="area") # Russia is huge
```



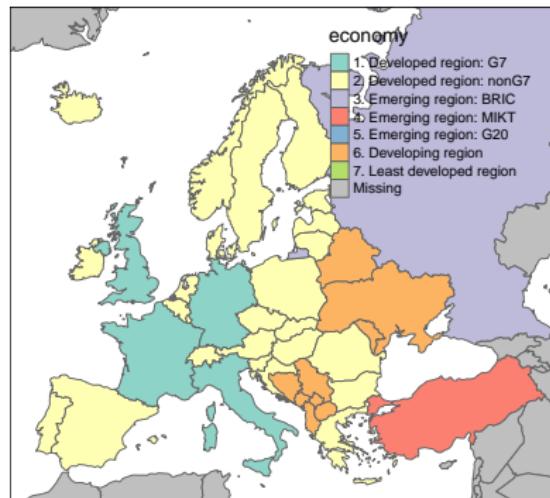
# Bevölkerung

```
qtm(Europe, fill="pop_est",fill.title="Population")
```



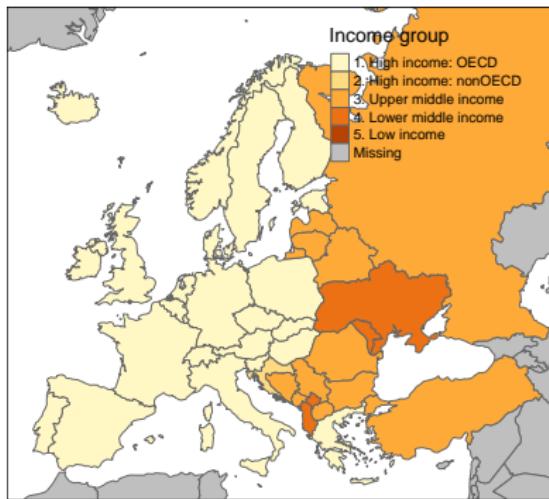
# Ökonomie

```
qtm(Europe, fill="economy")
```



# Einkommensgruppe

```
qtm(Europe, fill="income_grp", fill.title="Income group")
```

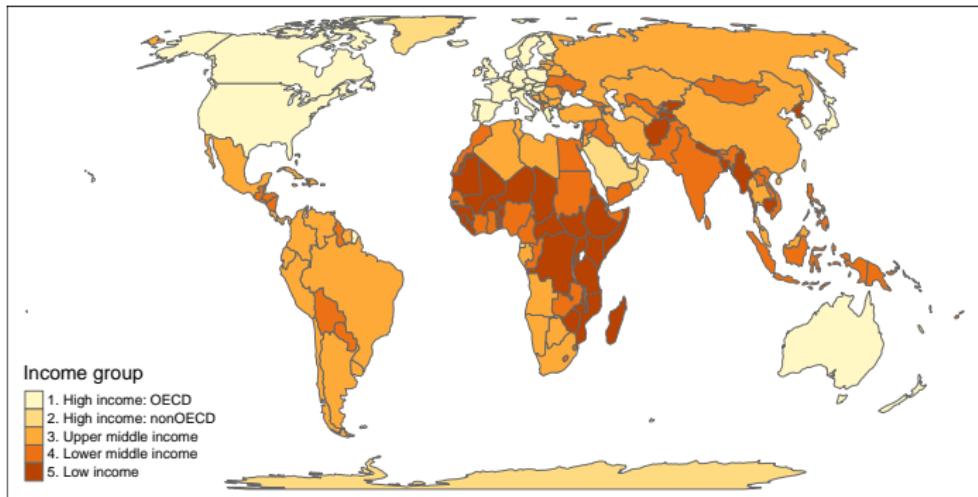


# Der Welt-Datensatz im Paket tmap

	iso_a3	name	sovereign	continent
2	AFG	Afghanistan	Afghanistan	Asia
3	AGO	Angola	Angola	Africa
5	ALB	Albania	Albania	Europe
8	ARE	United Arab Emirates	United Arab Emirates	Asia
9	ARG	Argentina	Argentina	South America
10	ARM	Armenia	Armenia	Asia
12	ATA	Antarctica	Antarctica	Antarctica
14	ATF	Fr. S. Antarctic Lands	France	Seven seas (o)
16	AUS	Australia	Australia	Oceania
17	AUT	Austria	Austria	Europe
18	AZE	Azerbaijan	Azerbaijan	Asia
19	BDI	Burundi	Burundi	Africa
20	BEL	Belgium	Belgium	Europe
21	BEN	Benin	Benin	Africa
22	BFN	Burkina Faso	Burkina Faso	Africa

# Welt - Länder nach Einkommensgruppe

```
qtm(World, fill="income_grp",fill.title="Income group")
```

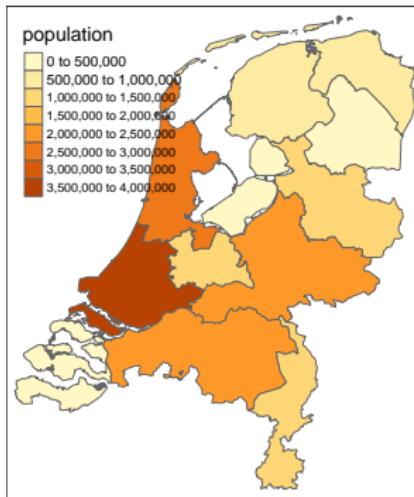


# Ein Datensatz zu den Provinzen in den Niederlanden (R-Paket tmap)

	code	name	population	pop_men	pop_women	pop_0_14
0	20	Groningen	582705	289795	292875	15
1	21	Friesland	646290	323215	323055	17
2	22	Drenthe	488970	242225	246755	17
3	23	Overijssel	1139680	570185	569465	18
4	24	Flevoland	399885	199940	199940	20
5	25	Gelderland	2019635	997805	1021790	17

# Niederlande - Bevölkerung in den Provinzen

```
qtm(NLD_prov, fill="population", fill.title="population")
```



## Anteile berechnen

```
pop <- NLD_prov@data$population  
pop
```

```
## [1] 582705 646290 488970 1139680 399885 2019635 125364  
## [9] 3576960 380610 2479220 1119980
```

```
popmen <- NLD_prov@data$pop_men  
popmen
```

```
## [1] 289795 323215 242225 570185 199940 997805 61364  
## [9] 1764855 188655 1238600 555450
```

```
prop <- popmen/pop  
prop
```

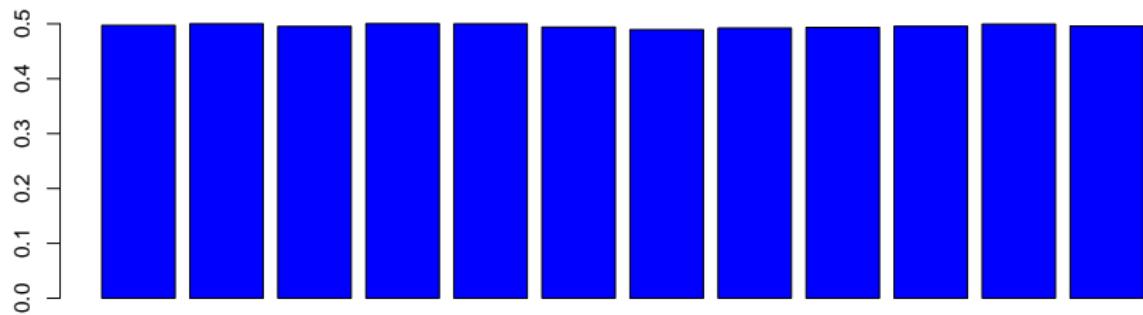
```
## [1] 0.4973271 0.5001083 0.4953780 0.5003027 0.4999937 0.49
```

# Exkurs: Barplot vom Männeranteil

```
barplot(prop)
```

## Barplot mit Farbe

```
barplot(prop, col="blue")
```

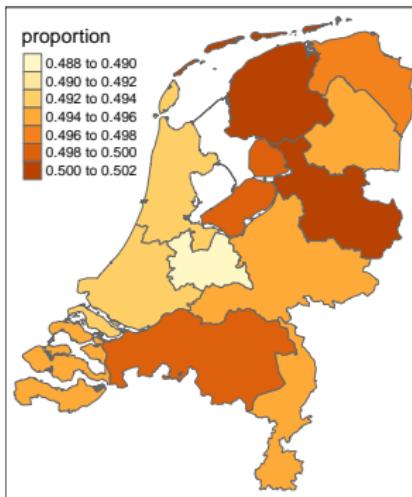


# Niederlande - Anteil Männer

Information in Datensatz einspeisen

```
NLD_prov@data$proportion <- prop
```

```
qtm(NLD_prov, fill="proportion", fill.title="proportion")
```



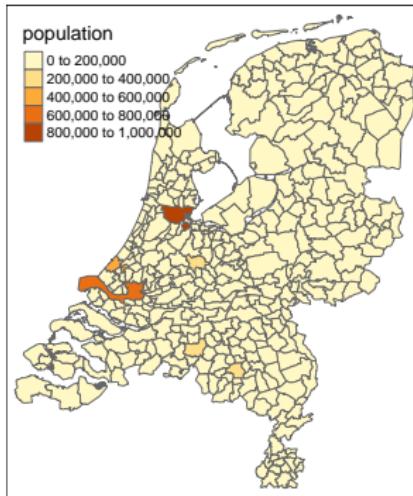
# Ein Datensatz zu den Gemeinden in den Niederlanden

```
data(NLD_muni)
```

	name	province	population
0	Appingedam	Groningen	12065
1	Bedum	Groningen	10495
2	Bellingwedde	Groningen	8920
3	Ten Boer	Groningen	7480
4	Delfzijl	Groningen	25695
5	Groningen	Groningen	198315
6	Grootegast	Groningen	12165
7	Haren	Groningen	18780
8	Hoogezand-Sappemeer	Groningen	34305
9	Leek	Groningen	19595
10	Loppersum	Groningen	10195

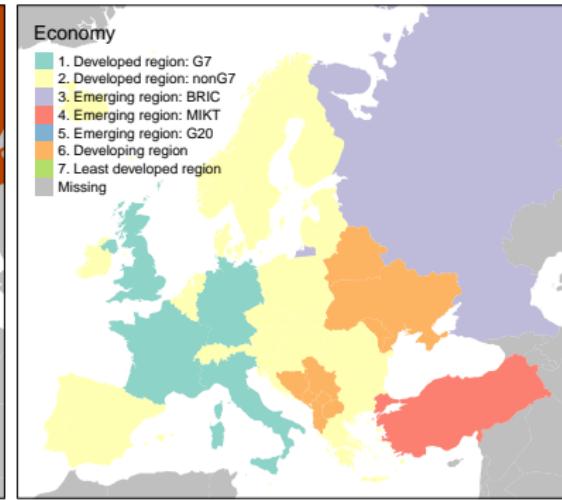
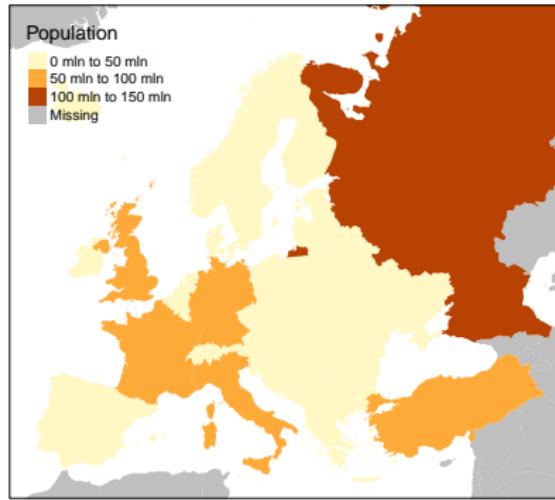
# Bevölkerung der Gemeinden in den Niederlanden

```
qtm(NLD_muni, fill="population")
```



# Zwei Karten

```
tm_shape(Europe) +  
  tm_fill(c("pop_est", "economy"),  
          title=c("Population", "Economy"))
```



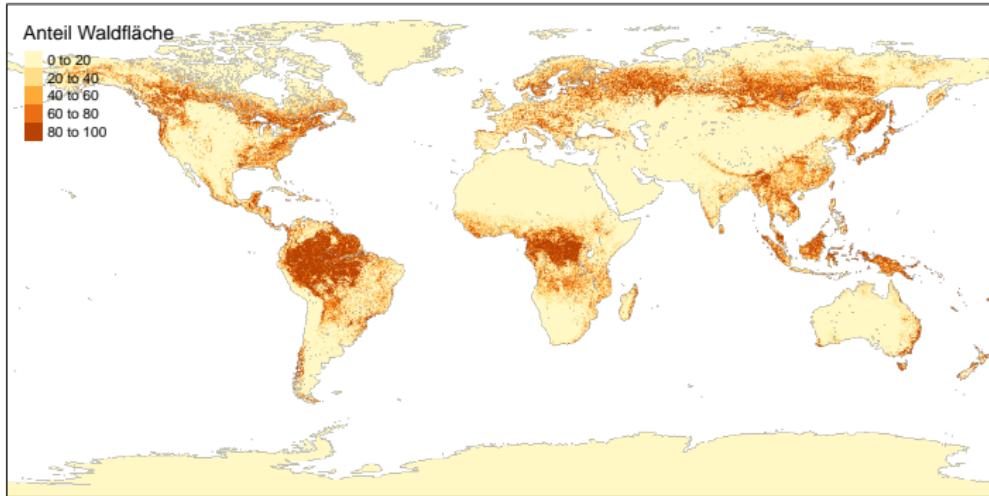
# Räumliche Daten zur Flächennutzung

```
data(land)  
data(World)
```

	cover_cls	trees
239644	Water	NA
184394	Bare area/Sparse vegetation	0
306844	Water	NA
90696	Forest	14
352519	Water	NA
546692	Snow/ice	0
375559	Water	NA
158069	Water	NA
487509	Water	NA
25804	Water	NA

# Weltweite Flächennutzung

```
tm_shape(land, relative=FALSE) +  
  tm_raster("trees", title="Anteil Waldfläche")
```



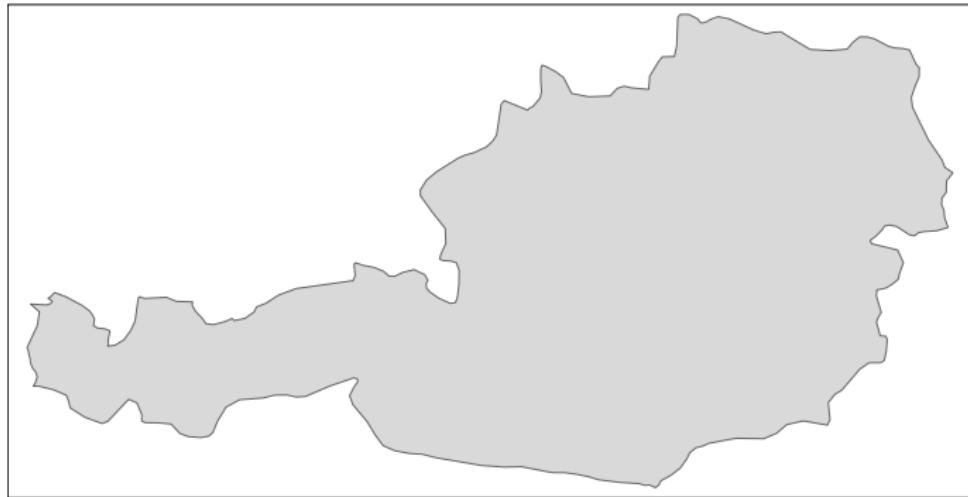
# Räumliche Daten zu Metropolregionen

data(metro)

	name	name_long	iso_a3	pop1950	pop1960	pop
2	Kabul	Kabul	AFG	170784	285352	47
8	Algiers	El Djazair (Algiers)	DZA	516450	871636	128
13	Luanda	Luanda	AGO	138413	219427	45
16	Buenos Aires	Buenos Aires	ARG	5097612	6597634	810
17	Cordoba	Cordoba	ARG	429249	605309	80
25	Rosario	Rosario	ARG	554483	671349	81
32	Yerevan	Yerevan	ARM	341432	537759	77
33	Adelaide	Adelaide	AUS	429277	571822	85
34	Brisbane	Brisbane	AUS	441718	602999	90
37	Melbourne	Melbourne	AUS	1331966	1851220	249

# Nur ein Land visualisieren

```
tm_shape(Europe[Europe$name=="Austria", ]) +  
  tm_polygons()
```



# Beispieldaten laden

## Datenquelle Eurostat

- Daten zur Arbeitslosigkeit in Europa

```
url <- "https://raw.githubusercontent.com/Japhilko/  
GeoData/master/2015/data/Unemployment07a13.csv"
```

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

# Überblick über die Daten

X	GEO	Val2007M12	Val2013M01
9316	EU28	6.9	10.9
9325	EU27	6.9	10.9
9334	EU25	6.9	11.0
9343	EU15	6.9	11.1
9352	EA	7.3	12.0
9361	EA19	7.3	12.0
9370	EA18	7.4	12.0
9379	EA17	7.4	12.0
9388	EA16	7.4	12.0
9397	EA15	7.3	12.0

# Nutzung des Paketes tmap mit eigenen Daten

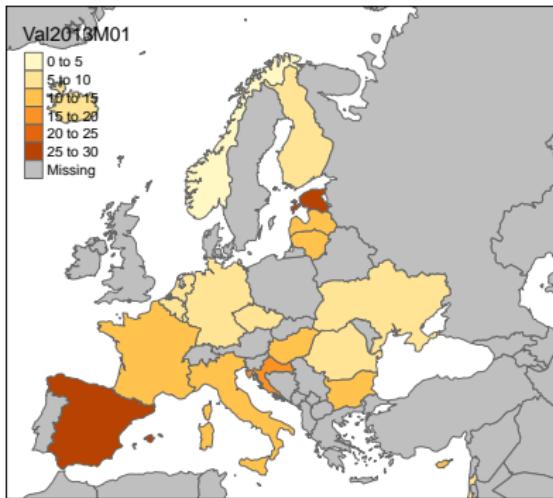
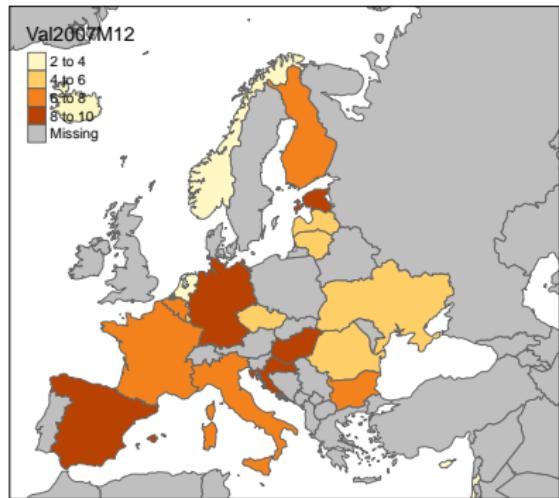
```
library("tmap")
data(Europe)
```

## Die Daten matchen

```
iso_a2<- substr(Europe@data$iso_a3,1,2)
ind <- match(iso_a2,Unemp$GEO)
Europe@data$Val2007M12 <- Unemp$Val2007M12[ind]
Europe@data$Val2013M01 <- Unemp$Val2013M01[ind]
```

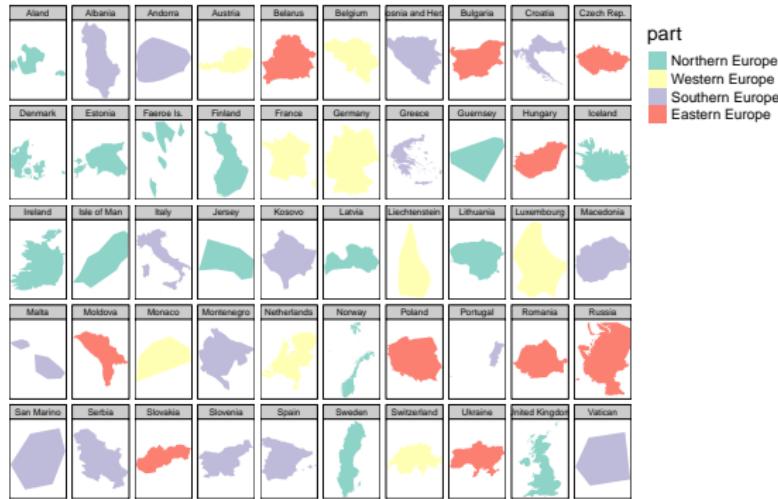
# Eine Karte erzeugen

```
qtm(Europe, c("Val2007M12", "Val2013M01"))
```



# Kleine und viele Karten

```
tm_shape(Europe[Europe$continent=="Europe",]) +  
  tm_fill("part", thres.poly = 0) +  
  tm_facets("name", free.coords=TRUE)
```



# tmap zitieren

```
citation("tmap")  
  
##  
## To cite tmap/tmaptools in publications use:  
##  
## Tennekes M (2018). "tmap: Thematic Maps in R." _Journal of  
## Statistical Software_, *84*(6), 1-39. doi: 10.18637/jss.v084.i06.  
## (URL: http://doi.org/10.18637/jss.v084.i06).  
##  
## A BibTeX entry for LaTeX users is  
##  
## @Article{,  
##   title = {{tmap}: Thematic Maps in {R}},  
##   author = {Martijn Tennekes},  
##   journal = {Journal of Statistical Software},  
##   year = {2018}}
```

# Inhalt dieses Abschnitts

- Der Beispieldatensatz `wrld_simpl` im Paket `maptools` wird vorgestellt.
- Es wird gezeigt, wie man Daten aus anderen Quellen mit Kartendaten verbinden kann.
- Mit dieser Verbindung ist es dann möglich thematische Karten - so genannte Choroplethen - zu erstellen
- Zudem wird das Paket `choroplethr` vorgestellt.

# Was ist ein Choropleth

Ein Choropleth ist eine Karte, die

- geografische Grenzen zeigt.
- bei denen Bereiche basierend auf Metriken eingefärbt werden.

Choroplethen sind nützlich für die Visualisierung von Daten, wo geografische Grenzen eine natürliche Einheit der Aggregation sind.

# Das Paket maptools

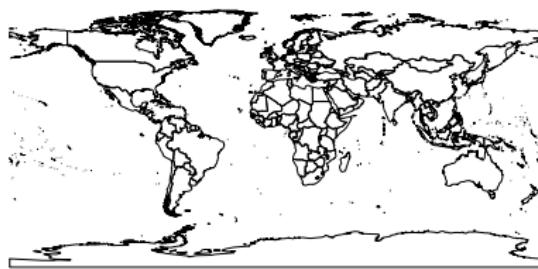
- Datensatz wrld\_simpl aus dem Paket maptools
- Polygone für fast alle Staaten der Erde

```
library(maptools)
data(wrld_simpl)
```

	ISO2	NAME	AREA	POP2005
ATG	AG	Antigua and Barbuda	44	83039
DZA	DZ	Algeria	238174	32854159
AZE	AZ	Azerbaijan	8260	8352021
ALB	AL	Albania	2740	3153731
ARM	AM	Armenia	2820	3017661
AGO	AO	Angola	124670	16095214

# Hallo Welt

```
plot(wrld_simpl)
```



# Daten zum Gini Index

- Daten von **datahub.io**
- Statistisches Maß zur Darstellung von Ungleichverteilungen

```
gini <- read.csv("../data/gini-index_csv.csv")
```

Country.Name	Country.Code	Year	Value
Albania	ALB	1996	27.0
Albania	ALB	2002	31.7
Albania	ALB	2005	30.6
Albania	ALB	2008	30.0
Albania	ALB	2012	29.0
Algeria	DZA	1988	40.2

# Der Gini Index im Jahr 2012

- Für das Jahr 2012 sind am meisten Beobachtungen vorhanden.

```
gini12 <- gini[gini$Year==2012,]  
summary(gini12$Value)
```

```
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.  
## 24.70   29.80  35.10   36.15  41.40   57.40
```

## Exkurs: der Befehl `match`

```
vec_a <- c("A", 2, 6, 1, "C")
vec_b <- c(1, "C", 2)
```

```
match(vec_a, vec_b)
```

```
## [1] NA 3 NA 1 2
```

# Die Daten matchen

- Wir matchen die Gini-Daten mit den Kartendaten

```
ind <- match(gini12$Country.Code, wrld_simpl$ISO3)
```

- Wir nehmen die Länder raus, für die keine Daten vorhanden sind:

```
ind2 <- ind[!is.na(ind)]
```

- Eine neue Karte wird erstellt:

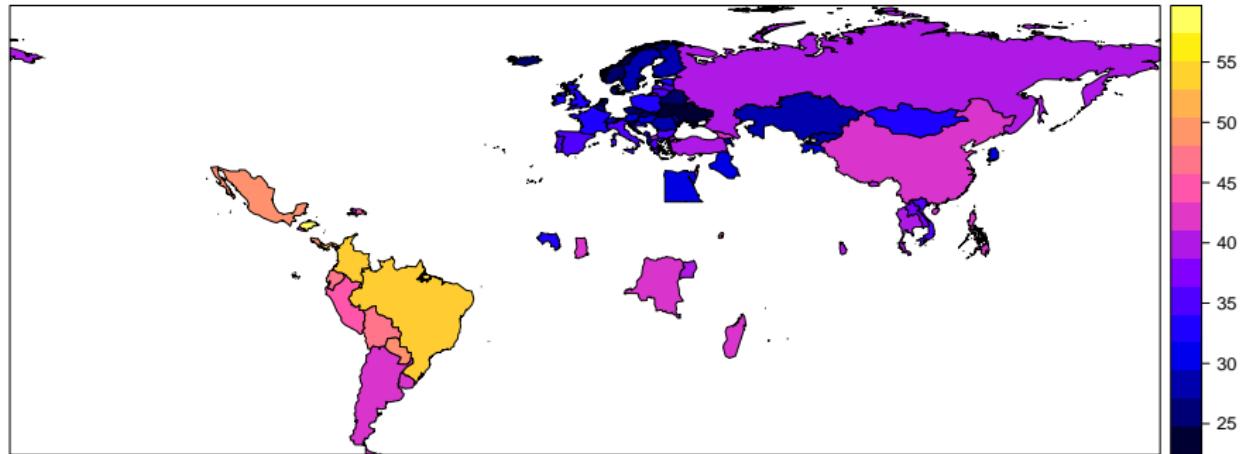
```
ginimap <- wrld_simpl[ind2, ]
```

- Die Gini-Daten werden in den Datenslot geschrieben

```
ginimap@data$gini12 <- gini12$value[!is.na(ind)]
```

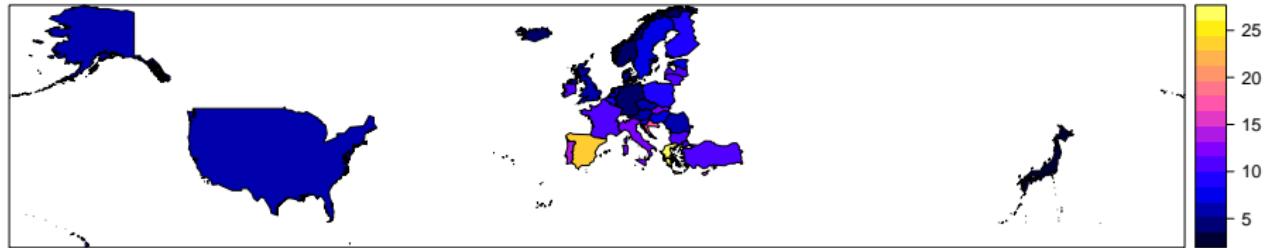
# Die Daten plotten

```
library(sp)
spplot(ginimap, "gini12")
```



## Aufgabe A4A - Eine Choroplethenkarte erzeugen

- Lade Datensatz **Unemployment Datensatz** herunter
- Matche die Daten mit einer passenden Karte
- Erzeuge mit der (Variable X2014M10) folgende Karte:



# Das Paket choroplethr

## Paket von Ari Lamstein - choroplethr

- Vereinfachung der Erstellung von Choroplethen in R
- World Development Indicators WDI (World Bank)
- Die folgenden Beispiele basieren auf der **Vignette** des choroplethr-Paketes

```
install.packages("choroplethr")
```

# Bevölkerungsschätzungen für den US-Staaten

df\_pop\_state ist ein Datensatz , der in dem Paket choroplethr enthalten ist, es enthält Schätzungen zu den US-Staaten für das Jahr 2012.

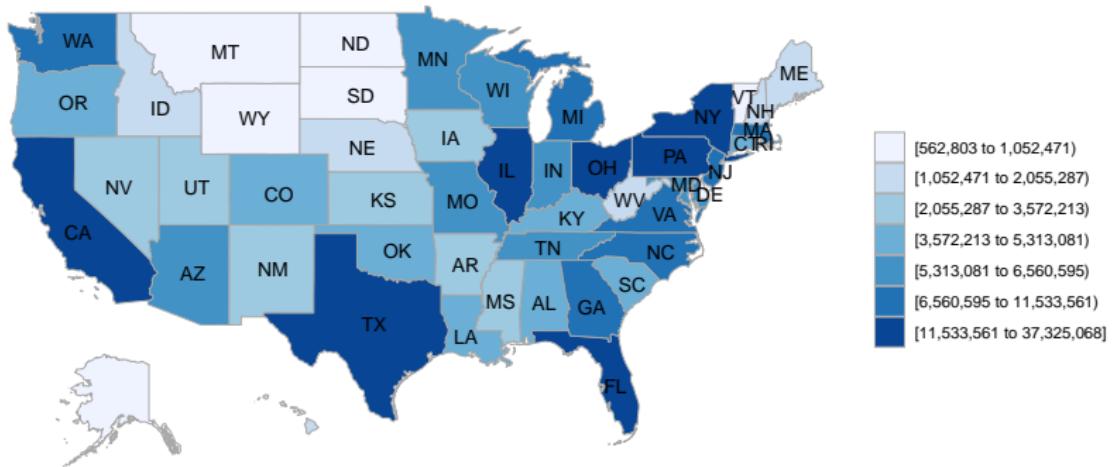
region	value
alabama	4777326
alaska	711139
arizona	6410979
arkansas	2916372
california	37325068
colorado	5042853

# choroplethr - Hallo Welt

Die Karte zeigt die US Bevölkerungsschätzung für die US-Staaten und das Jahr 2012:

Wir bekommen eine Choroplethenkarte mit nur einem Argument:

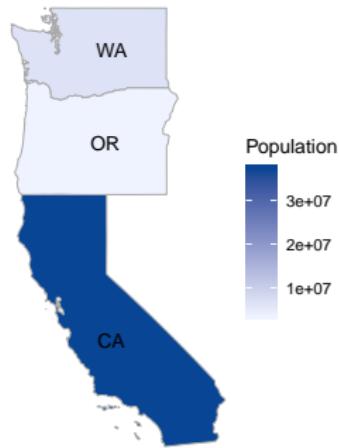
```
state_choropleth(df_pop_state)
```



# Nur drei Staaten darstellen

```
state_choropleth(df_pop_state,  
                  title= "2012 Population Estimates",  
                  legend= "Population", num_colors = 1,  
                  zoom=c("california","washington","oregon"))
```

2012 Population Estimates



# US County Chroplethen

## Choroplethen der US Counties

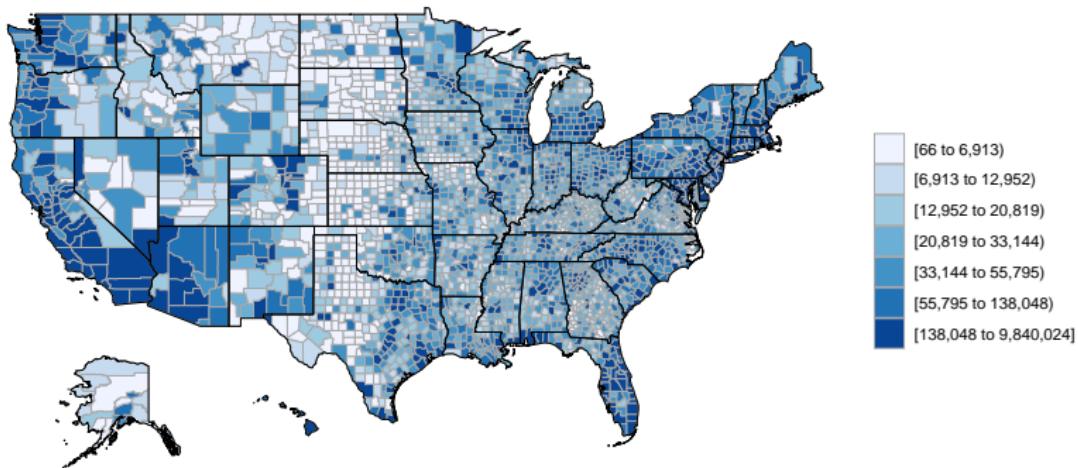
- Vignette des Pakets

```
# A data.frame containing population estimates for US Counties  
?df_pop_county
```

```
# Create a choropleth of US Counties  
?county_choropleth
```

# Eine Karte der US Counties

```
data(df_pop_county)  
county_choropleth(df_pop_county)
```



# Choroplethen Länder

```
data(df_pop_country)
country_choropleth(df_pop_country,
                    title      = "2012 Population Estimates",
                    legend     = "Population",
                    num_colors = 1,
                    zoom       = c("united states of america",
                                "mexico", "canada"))
```

# Choroplethen Länder

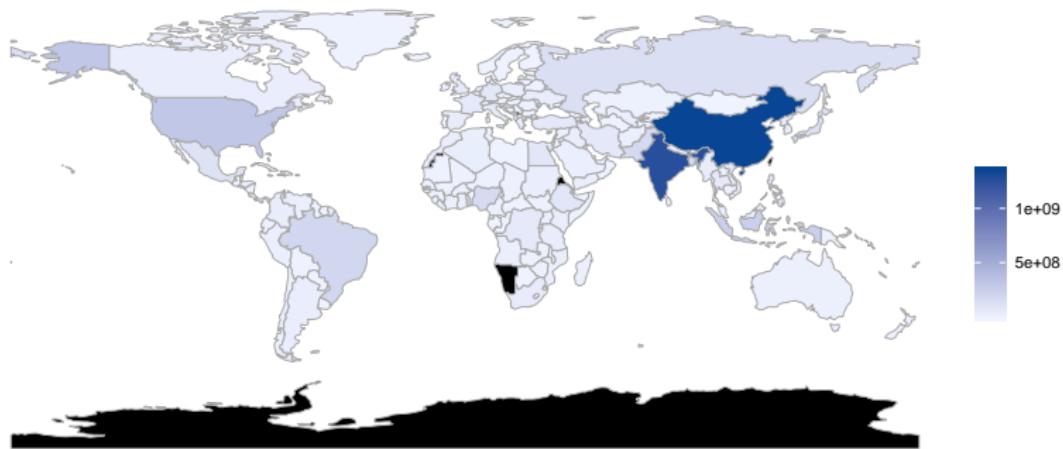
2012 Population Estimates



# Weltbank Daten

```
library(WDI)
choroplethr_wdi(code="SP.POP.TOTL", year=2012,
                 title="2012 Population",
                 num_colors=1)
```

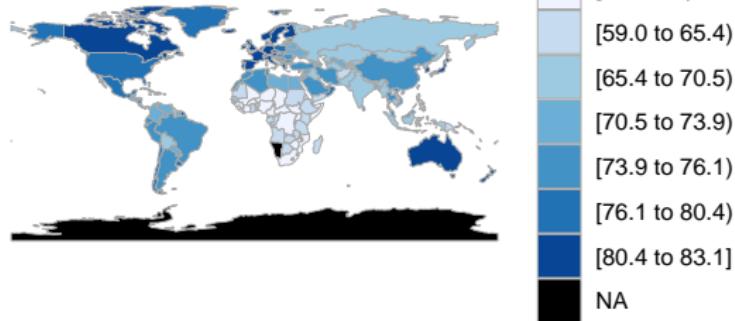
2012 Population



# Lebenserwartung

```
choropblethr_wdi(code="SP.DYN.LE00.IN", year=2012,  
                  title="2012 Life Expectancy")
```

2012 Life Expectancy



# Ein weiterer Datensatz

*A data.frame containing all US presidential election results from 1789 to 2012*

```
data(df_president_ts)
```

D = Democratic; R = Republican; PR = Progressive;

	region	1908	1912	1916	1920	1924	1928	1932
42	south dakota	R	PR	R	R	R	R	D
43	tennessee	D	D	D	R	D	R	D
44	texas	D	D	D	D	D	R	D
45	utah	R	R	D	R	R	R	D
46	vermont	R	R	R	R	R	R	R
47	virginia	D	D	D	D	D	R	D
48	washington	R	PR	D	R	R	R	D

# Resourcen

```
citation("choroplethr")  
  
##  
## To cite package 'choroplethr' in publications use:  
##  
##   Ari Lamstein (2018). choroplethr: Simplify the Creation of  
##   Choropleth Maps in R. R package version 3.6.3.  
##   https://CRAN.R-project.org/package=choroplethr  
##  
## A BibTeX entry for LaTeX users is  
##  
## @Manual{,  
##   title = {choroplethr: Simplify the Creation of Choropleth Maps in R},  
##   author = {Ari Lamstein},  
##   year = {2018},  
##   note = {R package version 3.6.3}
```

# Resources / Links

- **Einführung - Was sind Choroplethen**
- **Beschreibung** der Nutzung des choroplethr Paketes
- Die **US Staaten** plotten mit choroplethr
- **Weltbankdaten in Karten darstellen** mit choroplethr
- **Revolutions-Blog** über das choroplethr Paket
- **trulia-blog** über das choroplethr Paket
- **Präsentation von Ari Lamstein** über das choroplethr Paket

# Worum geht es in diesem Abschnitt

- Was sind Shapefiles
- Shapefiles mit Vorwahl- und PLZ-Bereichen importieren
- Einzelne Polygonzüge zusammenfassen
- Adressen für die gezogenen Punkte bestimmen
- Adressdatensatz bereinigen
- Entfernung zum Hauptbahnhof bestimmen

# Das shapefile Format ...

- ... ist ein beliebtes Format räumlicher Vektordaten für geographisches Informationssysteme (GIS).
- Es wurde entwickelt und reguliert von ESRI
- (meist) offene Spezifikation um Daten Interoperabilität zwischen Esri und anderen Formaten zu sichern.
- Es können Punkte, Linien und Polygone beschrieben werden
- Jedes Element hat Attribute, wie bspw. Name oder Temperatur die es beschreiben.

Quelle: <https://en.wikipedia.org/wiki/Shapefile>

# Der R Befehl `readShapePoly`

Um Shape-Dateien zu lesen, ist es notwendig, die drei Dateien mit den folgenden Dateierweiterungen im gleichen Verzeichnis zu haben:

- .shp
- .dbf
- .shx

# Vorwahlbereiche in Deutschland

[www.bundesnetzagentur.de](http://www.bundesnetzagentur.de)

- Wir verwenden das Paket `maptools`' um die Daten einzulesen:

```
setwd(geodata_path)
library(maptools)
onb <- readShapePoly("onb_grenzen.shp")
```

- Quelle Ortsnetzbereiche: **Bundesnetzagentur**

# Die Karte zeichnen

```
plot(onb)
```



# Der Datenslot

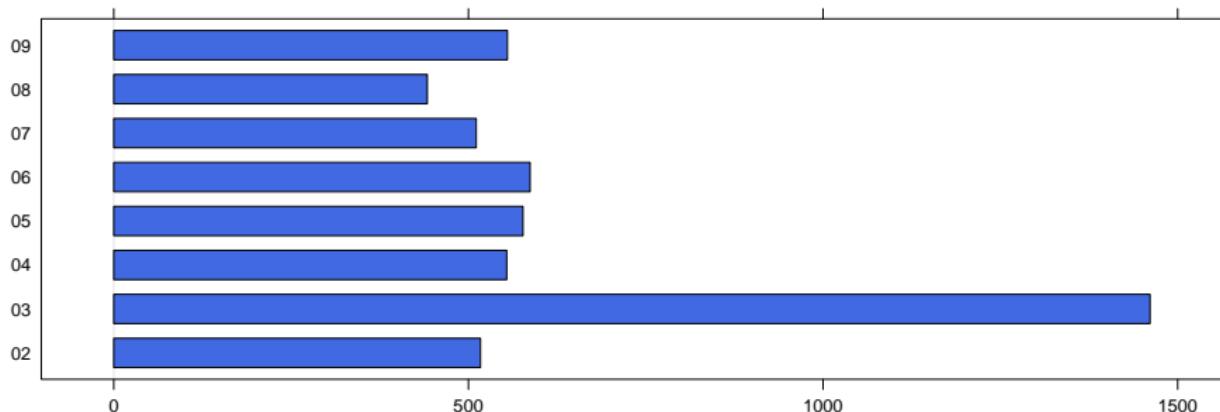
```
kable(head(onb@data))
```

	VORWAHL	NAME	KENNUNG
0	04651	Sylt	NA
1	04668	Klanxbüll	NA
2	04664	Neukirchen b Niebüll	NA
3	04663	Süderlügum	NA
4	04666	Ladelund	NA
5	04631	Glücksburg Ostsee	NA

# Einen Vorwahlbereich ausschneiden

```
vwb <- onb@data$VORWAHL  
vwb2 <- substr(vwb, 1,2)
```

```
library(lattice)  
barchart(table(vwb2), col="royalblue",  
         xlab="Häufigkeit")
```



# Vorwahlbereich ausschneiden

```
vwb6 <- onb[vwb2=="06",]  
plot(vwb6)
```



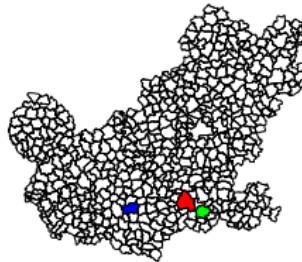
# Shapefiles zusammenfassen

```
vwb6c <- unionSpatialPolygons(vwb6,  
                                rep(1,length(vwb6)))  
plot(vwb6c,col="royalblue")
```



# Wo ist Mannheim?

```
Com <- vwb6@data$NAME  
plot(vwb6)  
plot(vwb6[Com=="Mannheim",], col="red", add=T)  
plot(vwb6[Com=="Heidelberg",], col="green", add=T)  
plot(vwb6[Com=="Kaiserslautern",], col="blue", add=T)
```



# Paket rgdal - PLZ Datensatz einlesen

## Quelle für PLZ Shapefiles

```
library(rgdal)
```

```
setwd(data_path)
plz <- readOGR ("post_pl.shp","post_pl")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "D:\Daten\Daten\GeoDaten\post_pl.shp", layer: "post_pl"
## with 8270 features
## It has 3 fields
```

# Die Daten plotten

```
plzbereich <- substr(plz@data$PLZ99, 1, 2)  
plot(plz[plzbereich=="68",])
```



# Die Grenze von Mannheim

```
ma_map <- plz[plz$PLZORT99=="Mannheim",]  
plot(ma_map)
```



# Die PLZ-Bereiche von Mannheim zusammenfassen

- Wir nutzen den Befehl `unionSpatialPolygons` im Paket `maptools`

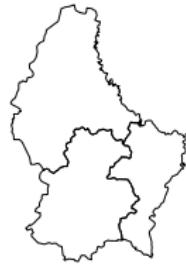
```
library(maptools)
ma_map2 <- unionSpatialPolygons(SpP = ma_map,
                                 IDs = rep(1,length(ma_map)))
plot(ma_map2)
```



# Global Administraive Boundaries - GADM - NUTS level 1

```
library(raster)
```

```
library(raster)
LUX1 <- getData('GADM', country='LUX', level=1)
plot(LUX1)
```



# Ein Blick auf die Daten

Koordinaten im polygon slot

```
LUX1@polygons[[1]]@Polygons[[1]]@coords
```

```
##           [,1]      [,2]
## [1,] 6.026519 50.17767
## [2,] 6.031361 50.16563
## [3,] 6.035646 50.16410
## [4,] 6.042747 50.16157
## [5,] 6.043894 50.16116
## [6,] 6.048243 50.16008
```

# Der Datenslot

```
head(LUX1@data)
```

```
##   OBJECTID ID_0 ISO      NAME_0 ID_1      NAME_1 HASC_1 CCM
## 1           1 131 Luxembourg  1 Diekirch LU.DI
## 2           2 131 Luxembourg  2 Grevenmacher LU.GR
## 3           3 131 Luxembourg  3 Luxembourg LU.LU
##   TYPE_1 ENGTTYPE_1 NL_NAME_1          VARNAME_1
## 1 District    District          Dikrech|Dikkrich
## 2 District    District          Gréivemaacher
## 3 District    District          Lëtzebuerg|Luxemburg
```

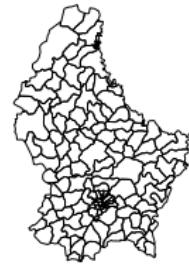
# GADM- NUTS level 3

```
LUX3 <- getData('GADM', country='LUX', level=3)  
plot(LUX3)
```



# GADM- NUTS level 4

```
LUX4 <- getData('GADM', country='LUX', level=4)  
plot(LUX4)
```



# GADM- NUTS level 3

```
DEU3 <- getData('GADM', country='DEU', level=3)  
plot(DEU3)
```



# Gemeinden in Deutschland

Bundesamt für Kartographie und Geodäsie (BKG)

```
library(maptools)
krs <- readShapePoly("vg250_krs.shp")
plot(krs)
```



# Kreise eines Bundeslandes

```
fds <- substr(krs@data$AGS, 1, 2)  
  
plot(krs[fds=="05",])
```



# Andere Quellen

- World Port Index

```
library(rgdal)
WPI <- readOGR ("WPI.shp","WPI")
plot(WPI)
```



# Weitere Quellen für Shapefiles

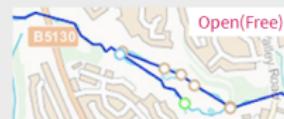
- **Eurostat Karten** - in der Regel die Europäischen Mitgliedsstaaten
- **Open linked data** - Ordnance Survey (GB)



OS Names API ›  
Addressing and Location



OS Open Roads ›  
Networks



OS Open Rivers ›  
Networks



OS Terrain 50 ›  
Height

- **World Borders Datensatz**
- **National Historical Information System**
- **Freie Polygon-Daten für die USA**
- Überblick über - **Spatial Data in R**

# Das erste Gesetz der Geographie (TFLG)

*“All things are related, but nearby things are more related than distant things” [Tobler, 1970]*

# Eine Karte von Afrika

```
library(maptools)
data(wrld_simpl)
Africa <- wrld_simpl[wrld_simpl@data$REGION==2,]
plot(Africa)
```



# Das Zentrum eines Polygonzuges

```
library(sp)
Af <- coordinates(Africa)
plot(Africa)
points(x=Af[1,1],y=Af[1,2],col="red",pch=20)
```



# Die nächsten Nachbarn finden

```
library(spdep)
Af_nb <- tri2nb(Af)
```

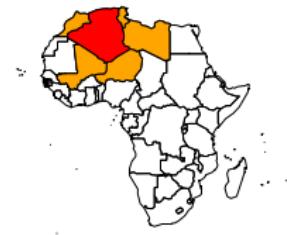
Die Nachbarn für das erste Land:

```
Af_nb[1]
```

```
## [[1]]
## [1] 24 26 27 32 48
```

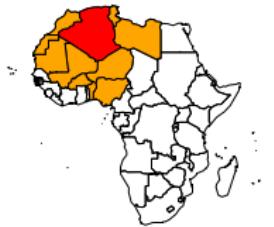
# Die Nachbarn finden

```
plot(Africa)
plot(Africa[1], col="red", add=T)
plot(Africa[Af_nb[1][[1]],], col="orange", add=T)
```



# Die 10 nächsten Nachbarn finden

```
IDs <- row.names(as(Africa, "data.frame"))
Af10_nb <- knn2nb(knearneigh(Af, k = 10), row.names = IDs)
plot(Africa)
plot(Africa[1], col="red", add=T)
plot(Africa[Af10_nb[1][[1]],], col="orange", add=T)
```



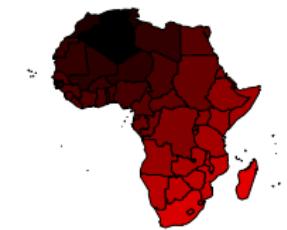
# Die Distanz berechnen

```
Af <- coordinates(Africa) # get centroid  
library(raster)  
pointDistance(Af[1:4,], lonlat=TRUE) # compute distance
```

```
##          [,1]      [,2]      [,3]  [,4]  
## [1,]        0       NA       NA   NA  
## [2,] 4763231        0       NA   NA  
## [3,] 2055609 2954497        0   NA  
## [4,] 3484053 1295173 1839191     0
```

# Berechnen/zeichnen einer Distanzmatrix

```
Dist_Af <- pointDistance(Af, lonlat=TRUE)
Af_color <- Dist_Af[,1]
Af_color <- Af_color/max(Af_color)
Af_color <- rgb(Af_color,0,0)
plot(Africa,col=Af_color)
```



# Aufgabe

```
library(sf)
lnd <- read_sf("../data/london_sport.shp")
```

# Links

- Raster, CMSAF and solaR

<https://procomun.wordpress.com/2011/06/17/raster-cmsaf-and-solar/>

- Getting rasters into shape from R

<https://johnbaumgartner.wordpress.com/2012/07/26/getting-rasters-into-shape-from-r/>