Nutzung von GeoDaten in den Sozialwissenschaften - Kartentypen mit ggmap

Jan-Philipp Kolb

07 April 2016

Gliederung

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.

Straßenkarten

- Straßenkarte werden sehr häufig verwendet.
- Diese Karten zeigen Haupt- und Nebenstraßen (abhängig vom Detail)
- oft sind auch weitere Informationen enthalten. Wie beispielsweise Flughäfen, Städte, Campingplätze oder andere Orte von Interesse
- Beispiel einer Straßenkarte für Mannheim.

Installieren des Paketes

- Zur Erstellung der Karten brauchen wir das Paket `ggmap`:

1. Möglichkeit:

```
install.packages("ggmap")
```

2. Möglichkeit:

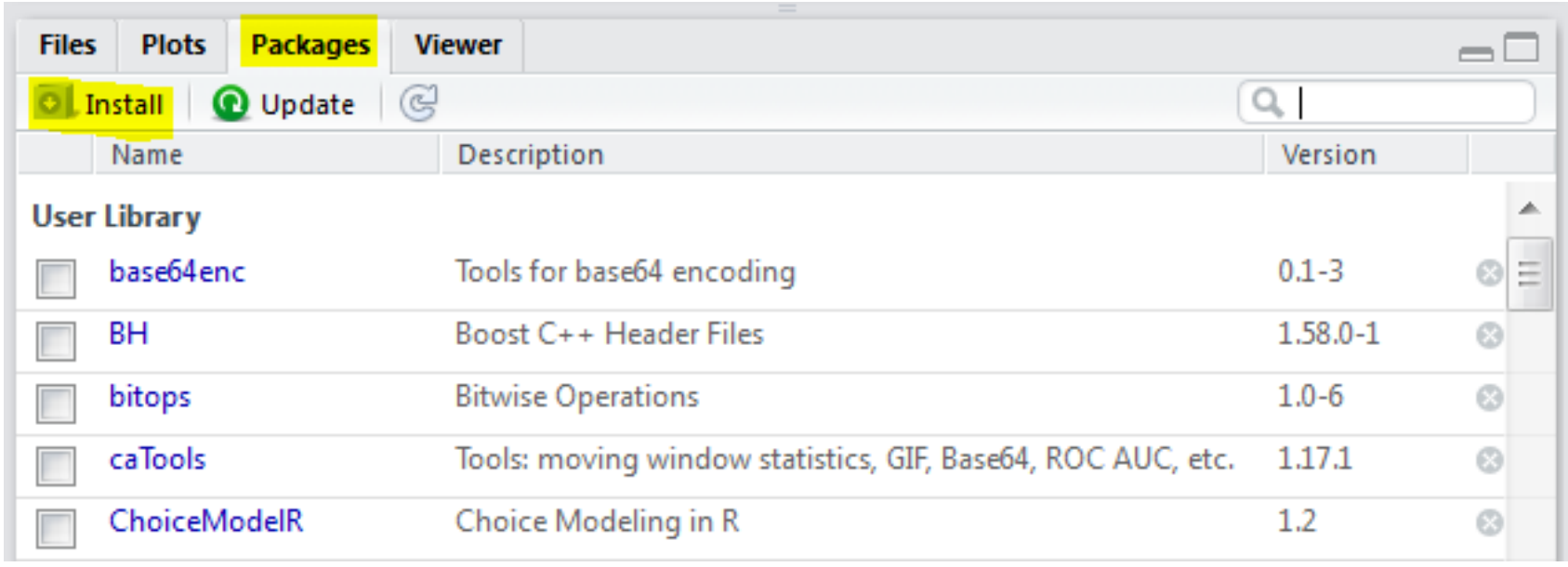


Figure 1: Install packages

Pakete installieren

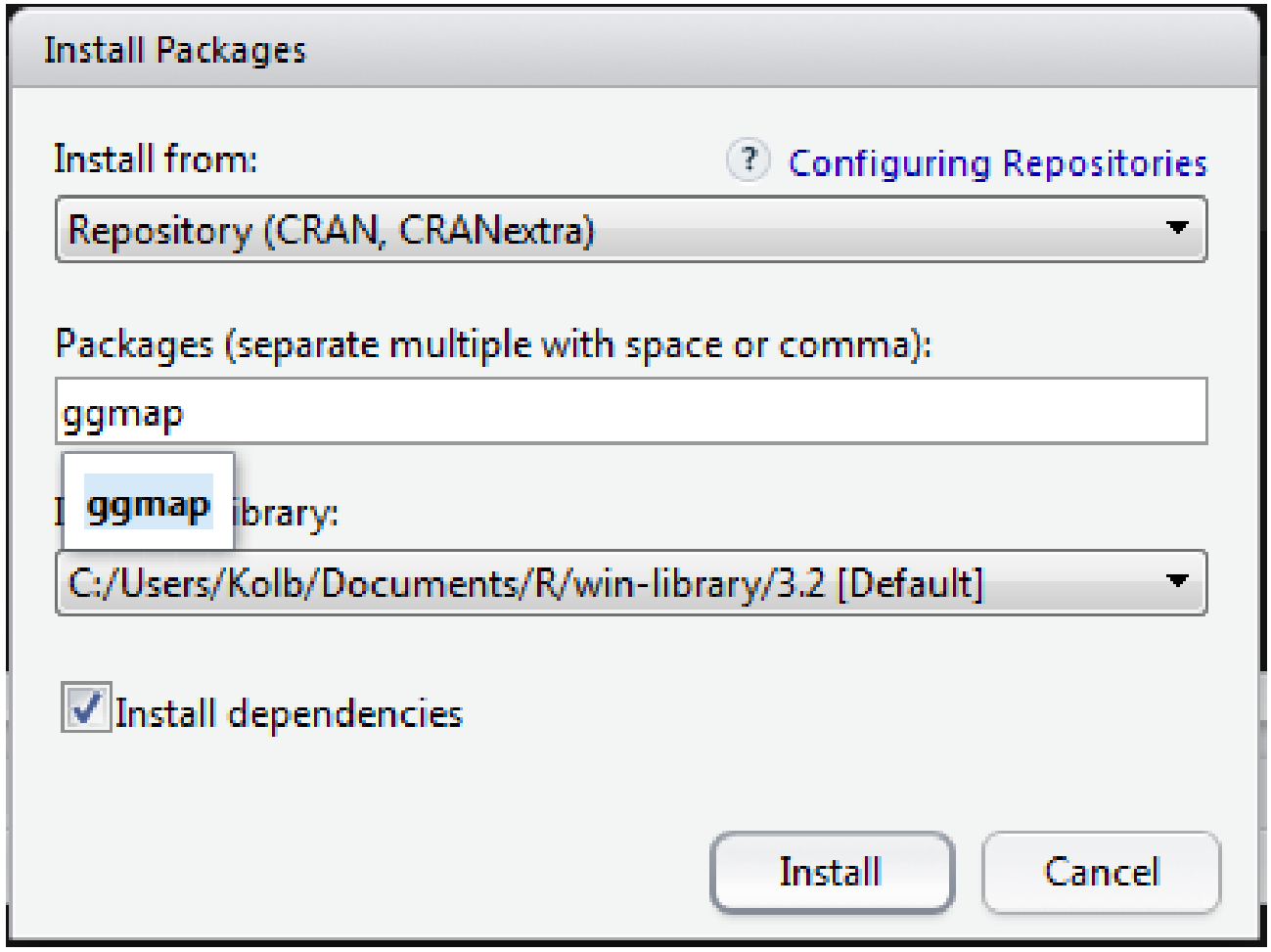


Figure 2: Install packages

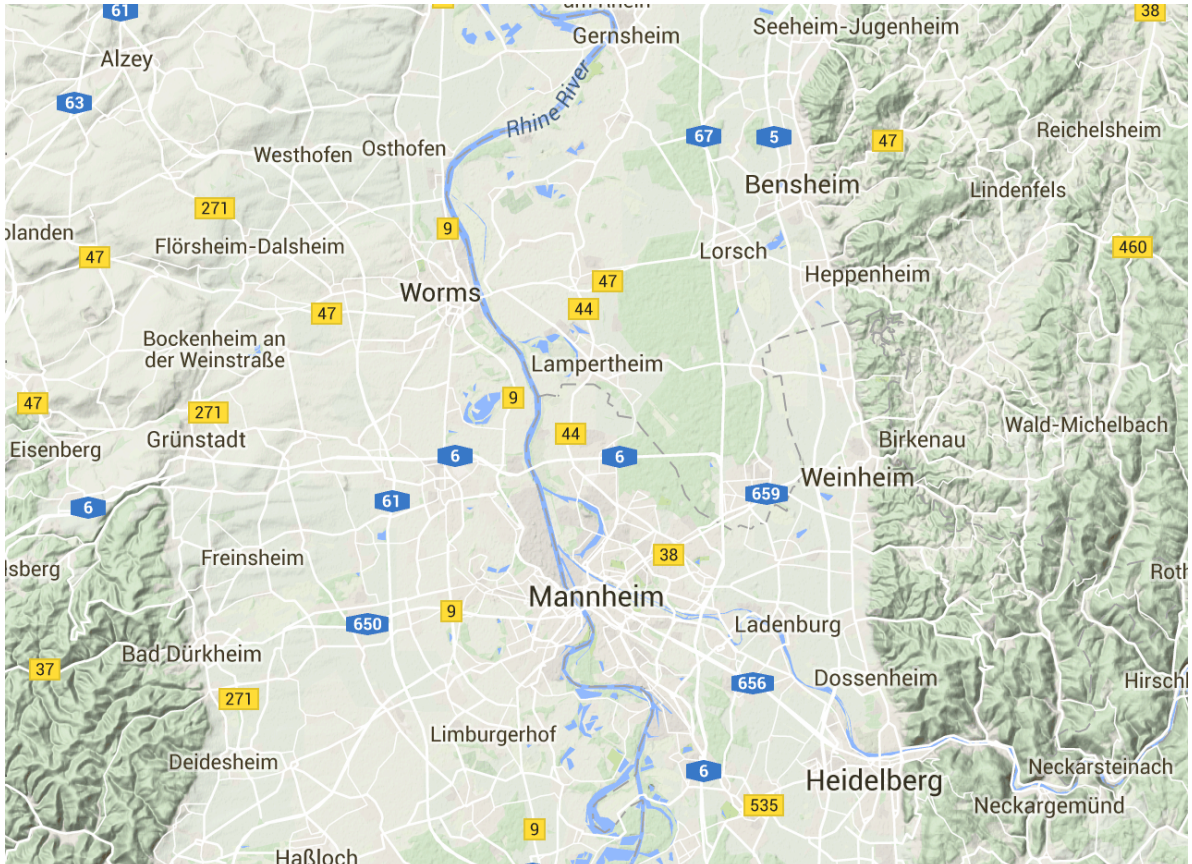
Paket ggmap - Hallo Welt

- Um das Paket zu laden verwenden wir den Befehl `library`

```
library(ggmap)
```

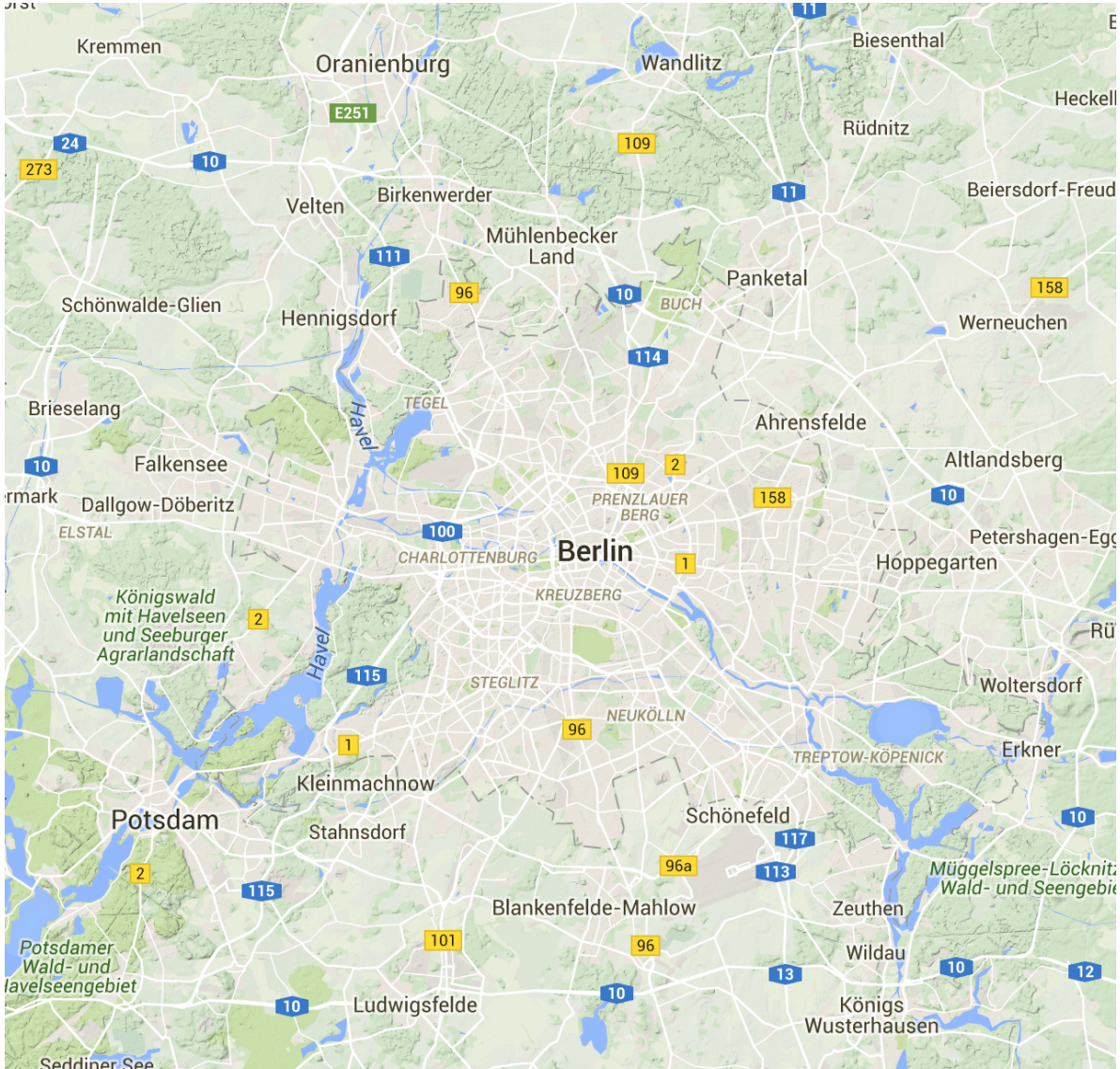
Und schon kann die erste Karte erstellt werden:

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



Karte für eine Sehenswürdigkeit

```
BBT <- qmap("Berlin Brandenburger Tor")
BBT
```



Karte für einen ganzen Staat

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

Ein anderes *zoom level*

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

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

Hilfe bekommen wir mit dem Fragezeichen

```
?qmap
```

Verschiedene Abschnitte in der Hilfe:

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

Die Beispiele in der Hilfe

Ausschnitt aus der Hilfe Seite zum Befehl qmap:

Examples

```
## Not run:
# these examples have been excluded for checking efficiency

qmap(location = "baylor university")
qmap(location = "baylor university", zoom = 14)
qmap(location = "baylor university", zoom = 14, source = "osm")
```

Figure 3: qmap Example

Das Beispiel kann man direkt in die Konsole kopieren:

```
# qmap("baylor university")
qmap("baylor university", zoom = 14)
# und so weiter
```

Ein anderes *zoom level*

```
qmap("Mannheim", zoom = 12)
```

Näher rankommen

```
qmap('Mannheim', zoom = 13)
```

Ganz nah dran

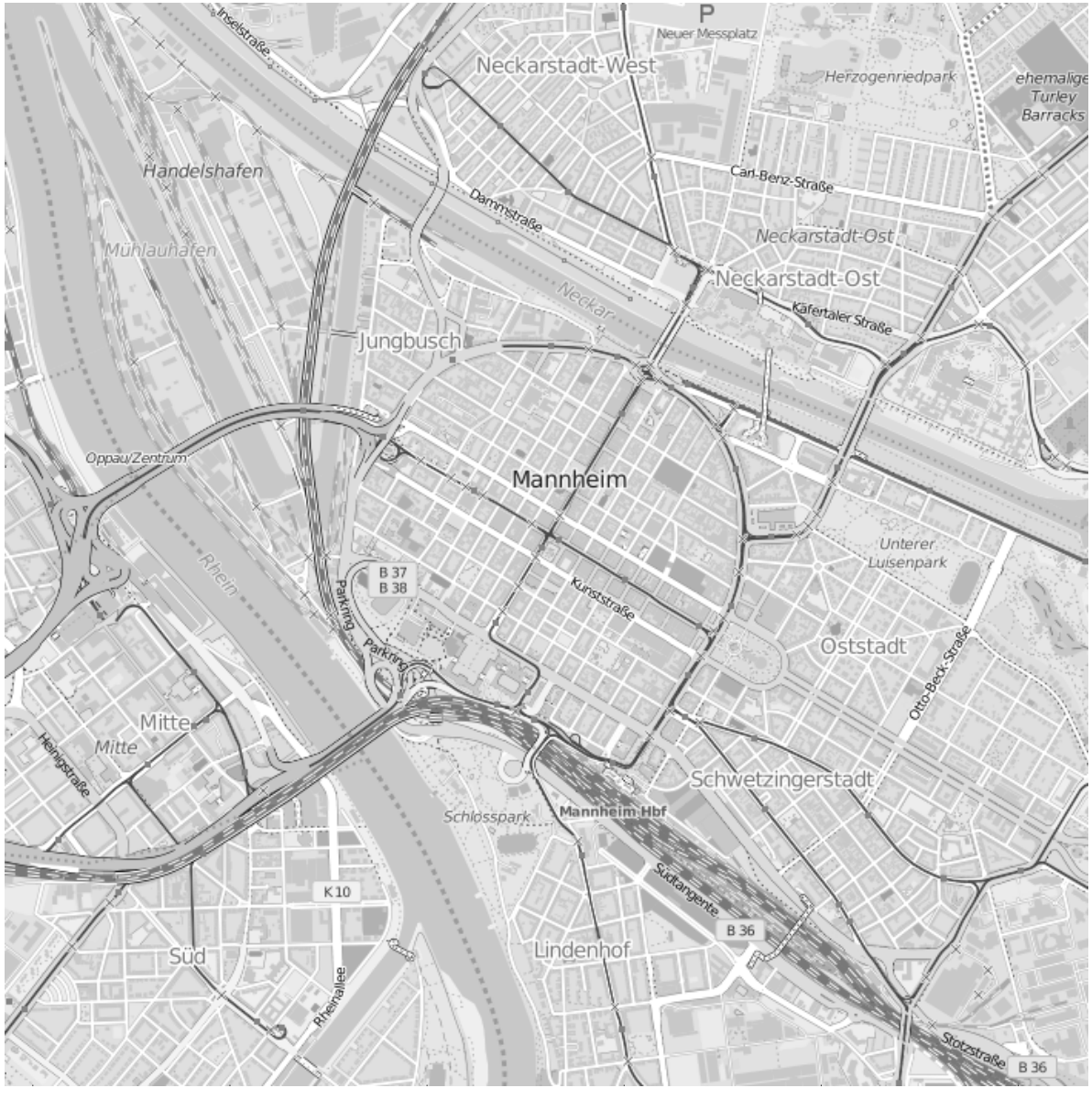
```
qmap('Mannheim', zoom = 20)
```

ggmap - Quelle OpenStreetMap

```
qmap('Mannheim', zoom = 14, source="osm")
```

ggmap - OpenStreetMap - schwarz/weiß

```
qmap('Mannheim', zoom = 14, source="osm",color="bw")
```

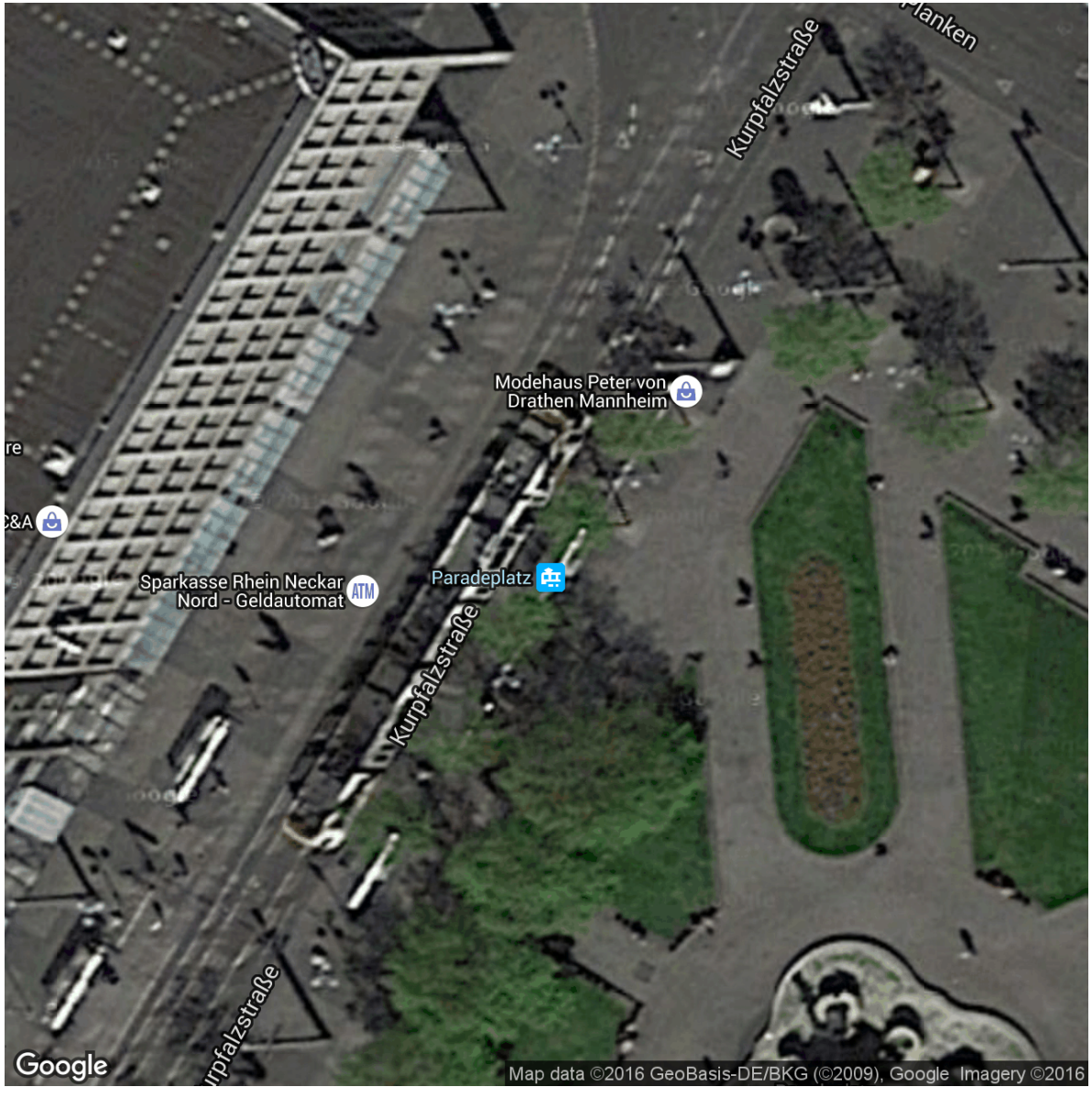


ggmap - maptype satellite

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

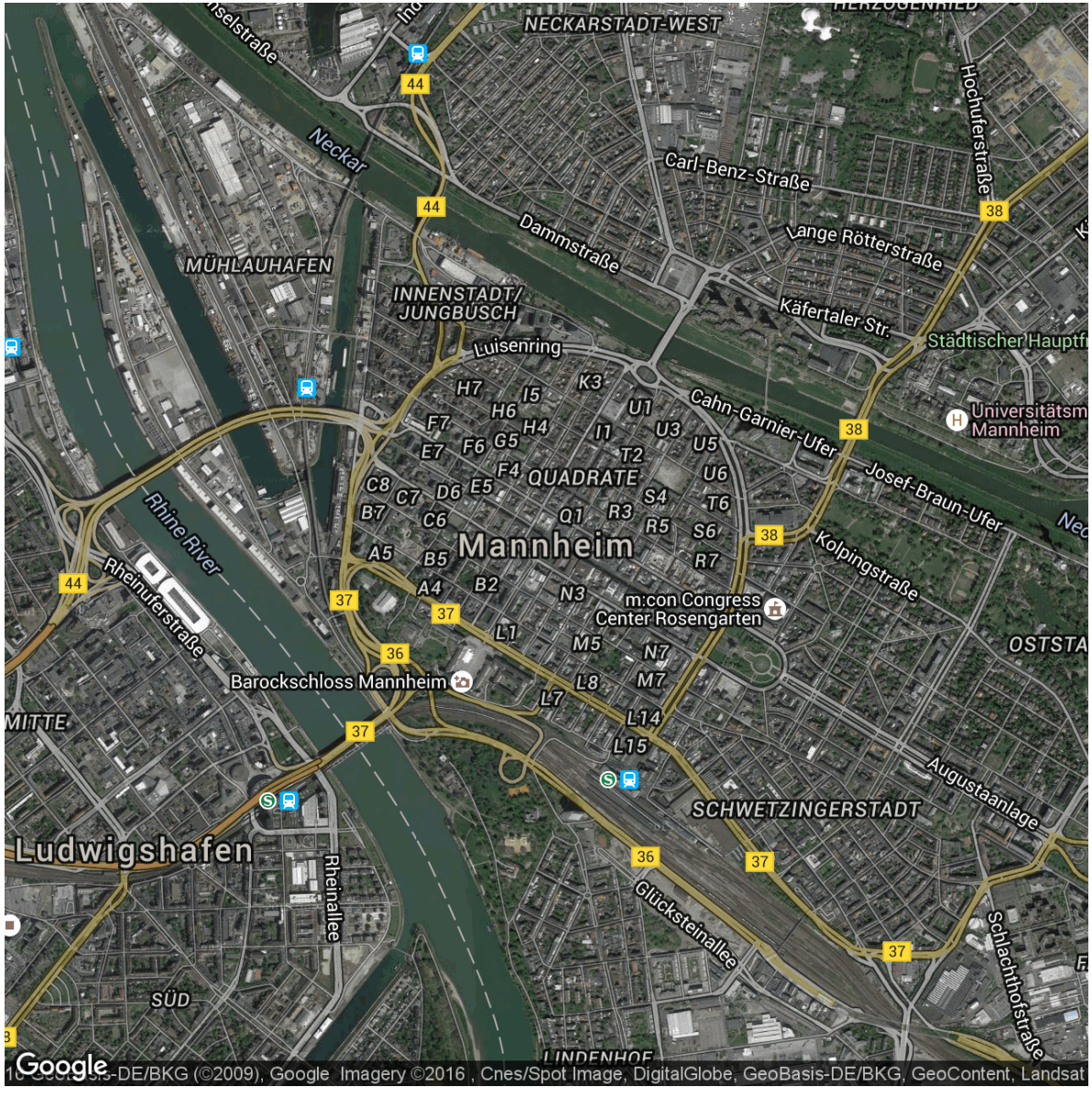
ggmap - maptype satellite zoom 20

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



ggmap - maptype hybrid

```
qmap("Mannheim", zoom = 14, maptype="hybrid")
```

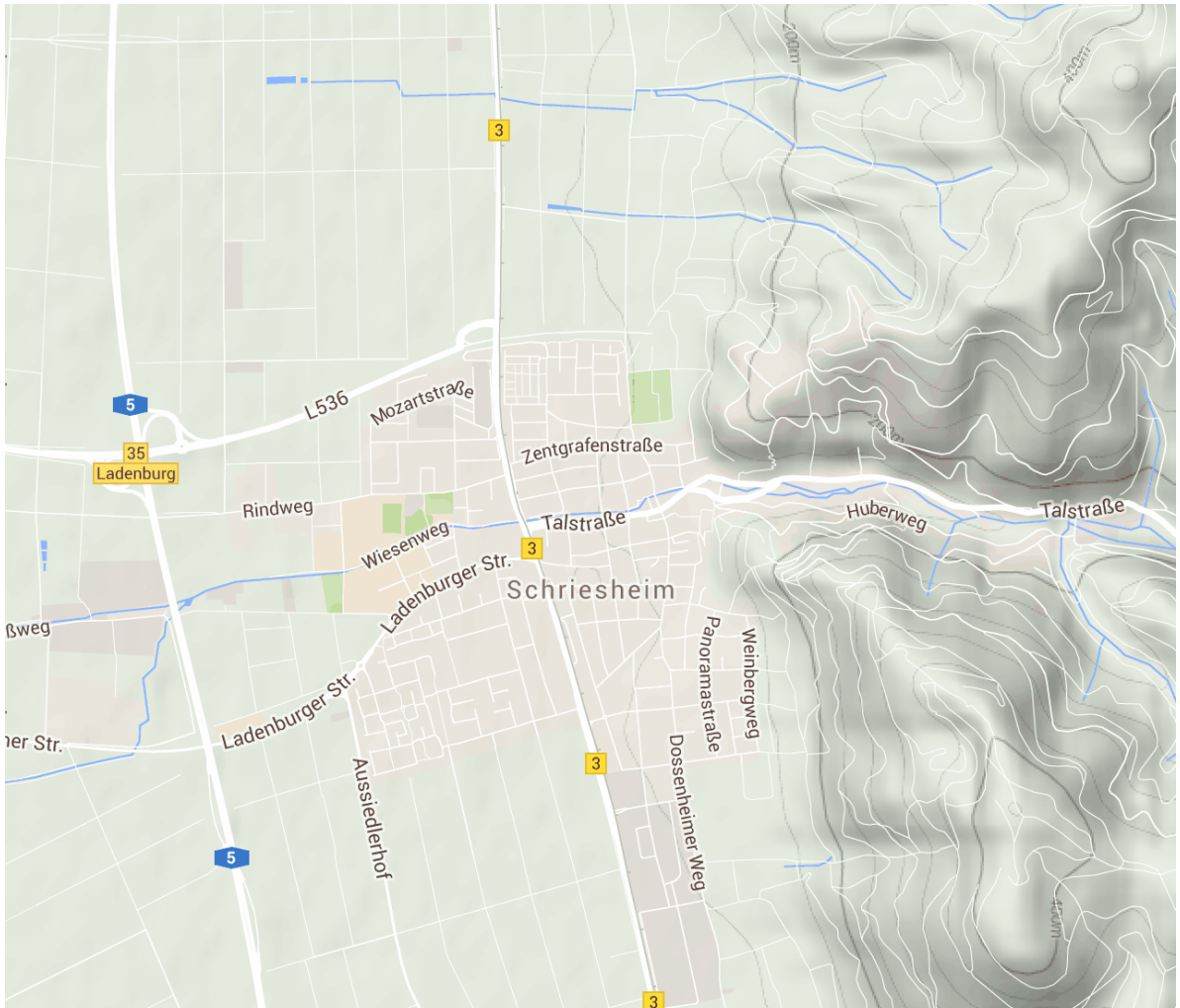


Terrain/physical maps

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

ggmap - terrain map

```
qmap('Schriesheim', zoom = 14,
 maptype="terrain")
```



Abstrahierte Karten (http://www.designfaves.com)

Figure 5: New York

- 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('Mannheim', zoom = 14,
 maptype="watercolor",source="stamen")
```

ggmap - source stamen

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

ggmap - maptype toner-lite

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

ggmap - maptype toner-hybrid

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

ggmap - maptype terrain-lines

```
qmap('Mannheim', zoom = 14,
 maptype="terrain-lines",source="stamen")
```

Graphiken speichern

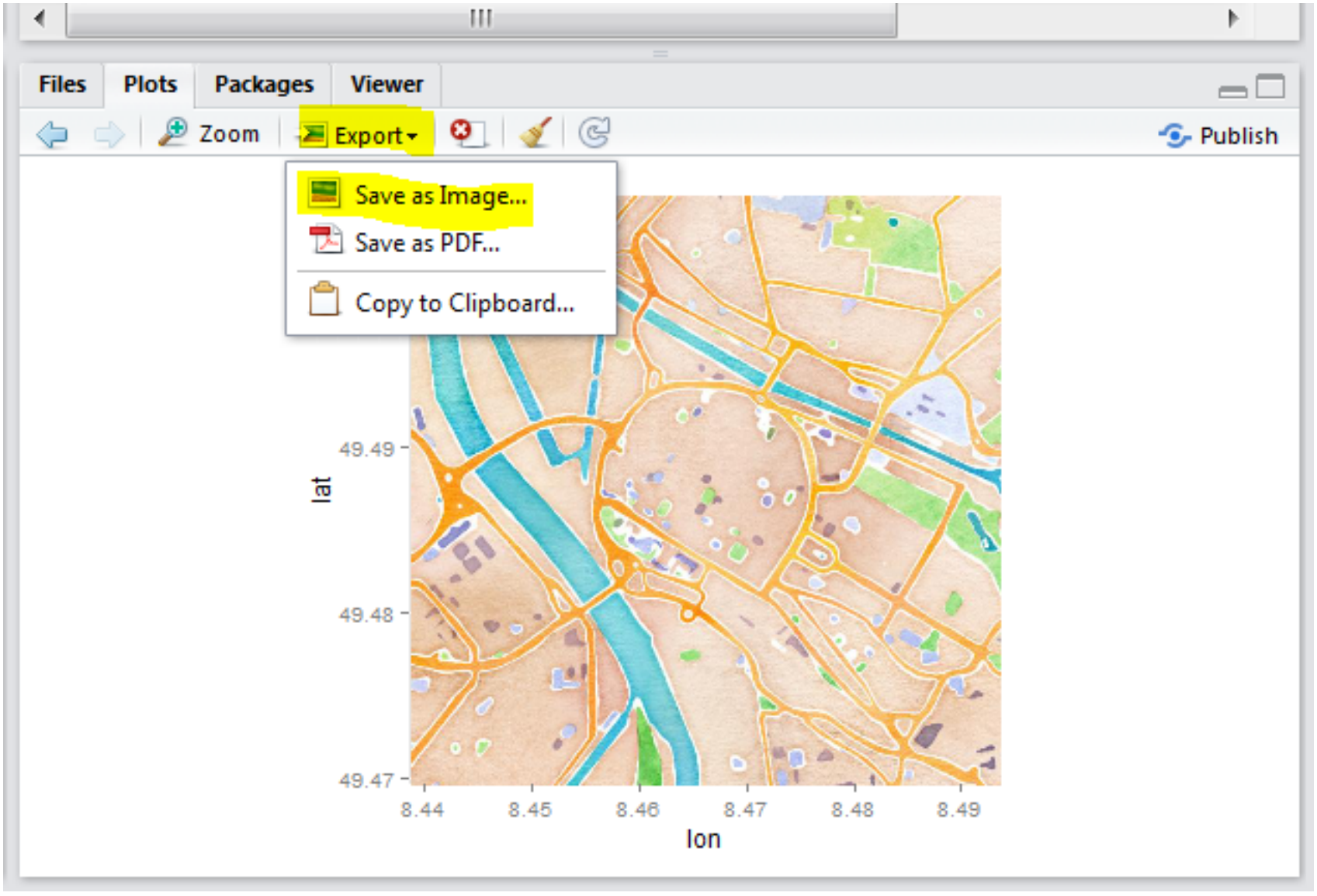


Figure 6: RstudioExport

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")
```

Geokodierung

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

Wikipedia - Geocoding

```
library(ggmap)
geocode("Mannheim Wasserturm",source="google")
```

lon	lat
34.79565	32.1221

Latitude und Longitude

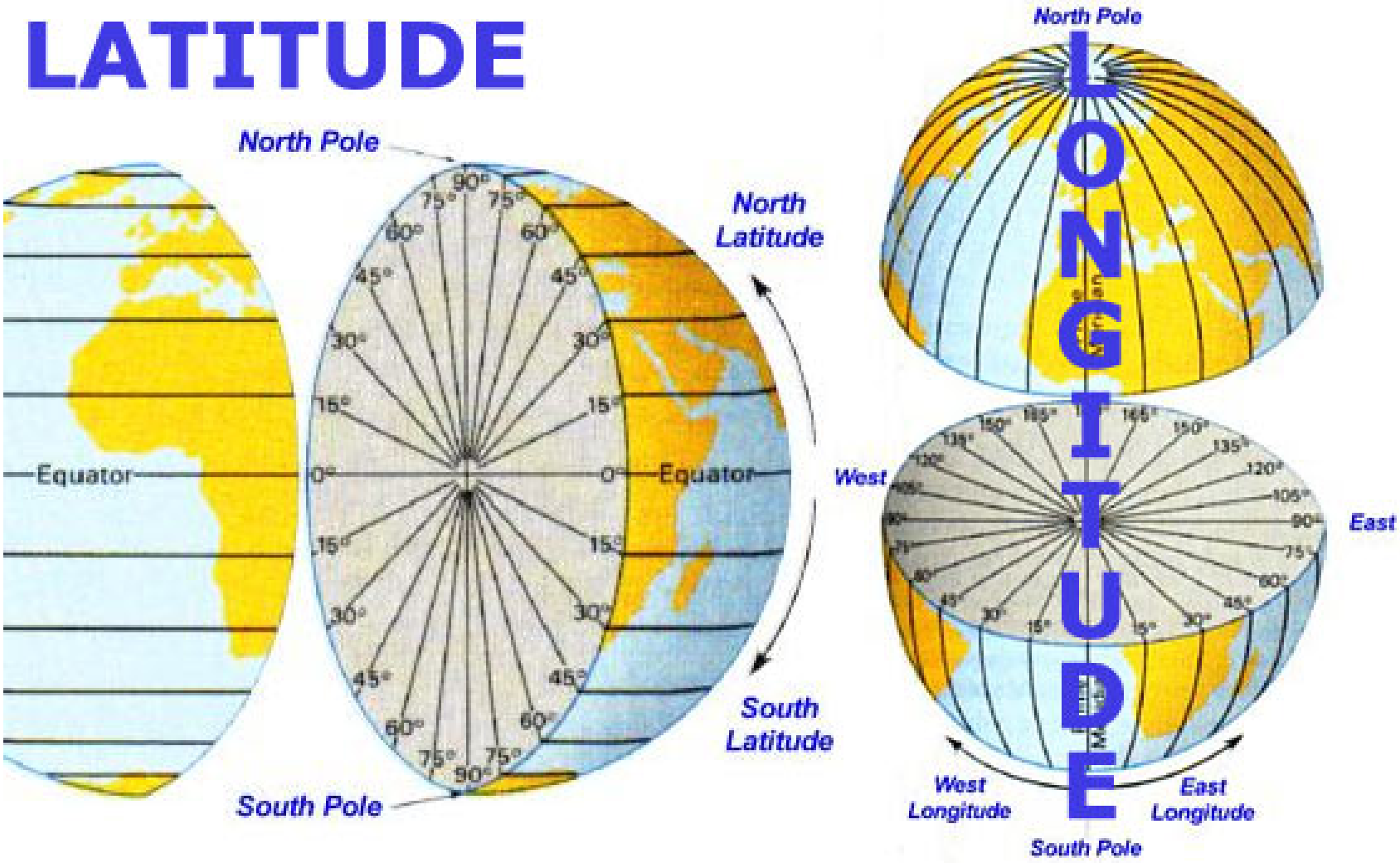


Figure 7: LatLon

http://modernsurvivalblog.com

Koordinaten verschiedener Orte in Deutschland

cities	lon	lat
Hamburg	9.993682	53.55108
Koeln	6.960279	50.93753
Dresden	13.737262	51.05041
Muenchen	11.581981	48.13513

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))
```

```
## [1] "Unnamed Road, Somalia"
```

Die Distanz zwischen zwei Punkten

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

```
##                from                  to   m    km     miles seco
## 1 Q1, 4 Mannheim B2, 1 Mannheim 746 0.746 0.4635644
##         hours
## 1 0.05305556
```

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

```
##                from                  to   m    km     miles seco
## 1 Q1, 4 Mannheim B2, 1 Mannheim 546 0.546 0.3392844
##        hours
## 1 0.1166667
```

Eine andere Distanz bekommen

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

```
##                from                  to   m    km    miles secon
## 1 Q1, 4 Mannheim B2, 1 Mannheim 555 0.555 0.344877     2
##        hours
## 1 0.05972222
```

Geokodierung - verschiedene Punkte von Interesse

```
POI1 <- geocode("B2, 1 Mannheim",source="google")
POI2 <- geocode("Hbf Mannheim",source="google")
POI3 <- geocode("Wasserturm Mannheim",source="google")
ListPOI <-rbind(POI1,POI2,POI3)
POI1;POI2;POI3
```

```
##        lon      lat
## 1 8.462844 49.48569
```

```
##        lon      lat
## 1 8.469879 49.47972
```

```
##        lon      lat
## 1 8.466039 49.48746
```

Punkte in der Karte

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

Punkte in der Karte

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

ggmap - verschiedene Farben

```
ListPOI$color <- c("A","B","C")
MA_map +
geom_point(aes(x = lon, y = lat,col=color),
data = ListPOI)
```

ggmap - größere Punkte

```
ListPOI$size <- c(10,20,30)
MA_map +
geom_point(aes(x = lon, y = lat,col=color,size=size),
data = ListPOI)
```

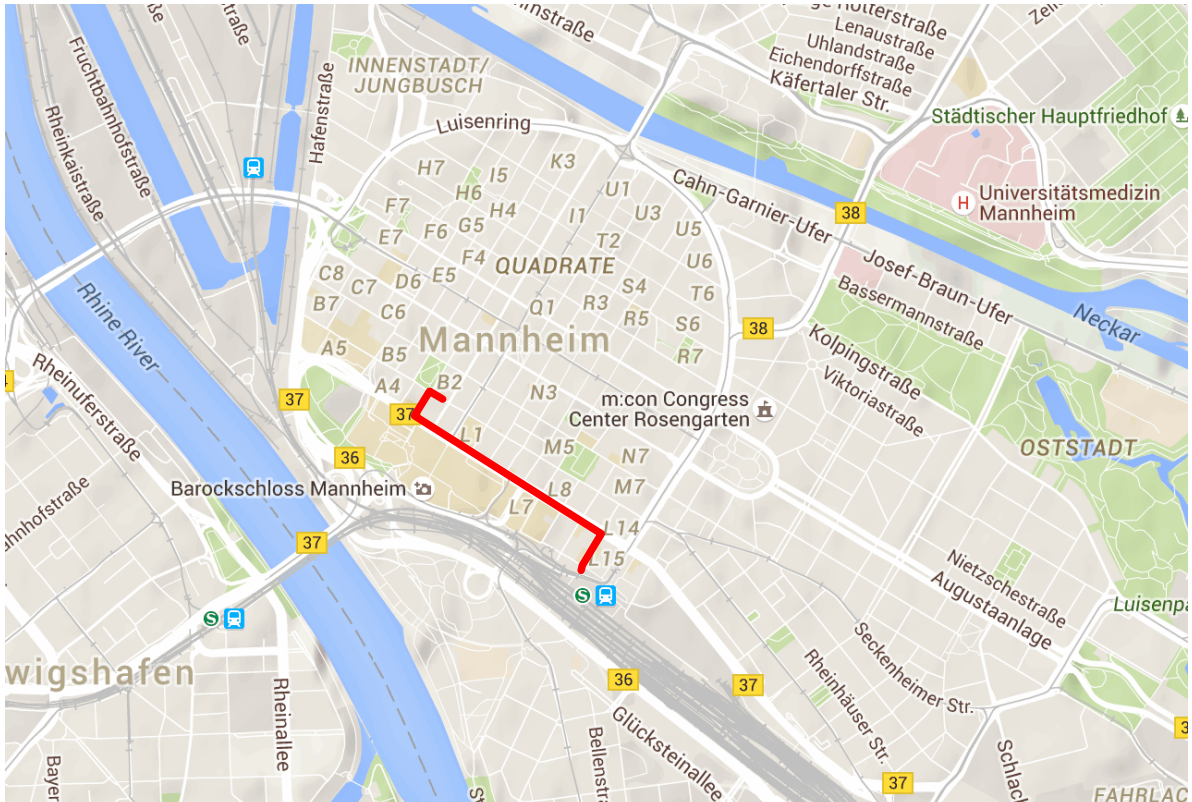
Eine Route von Google maps bekommen

```
from <- "Mannheim Hbf"
to <- "Mannheim B2 , 1"
route_df <- route(from, to, structure = "route")
```

Mehr Information
http:
//rpackages.ianhowson.com/cran/ggmap/man/route.html

Eine Karte mit dieser Information zeichnen

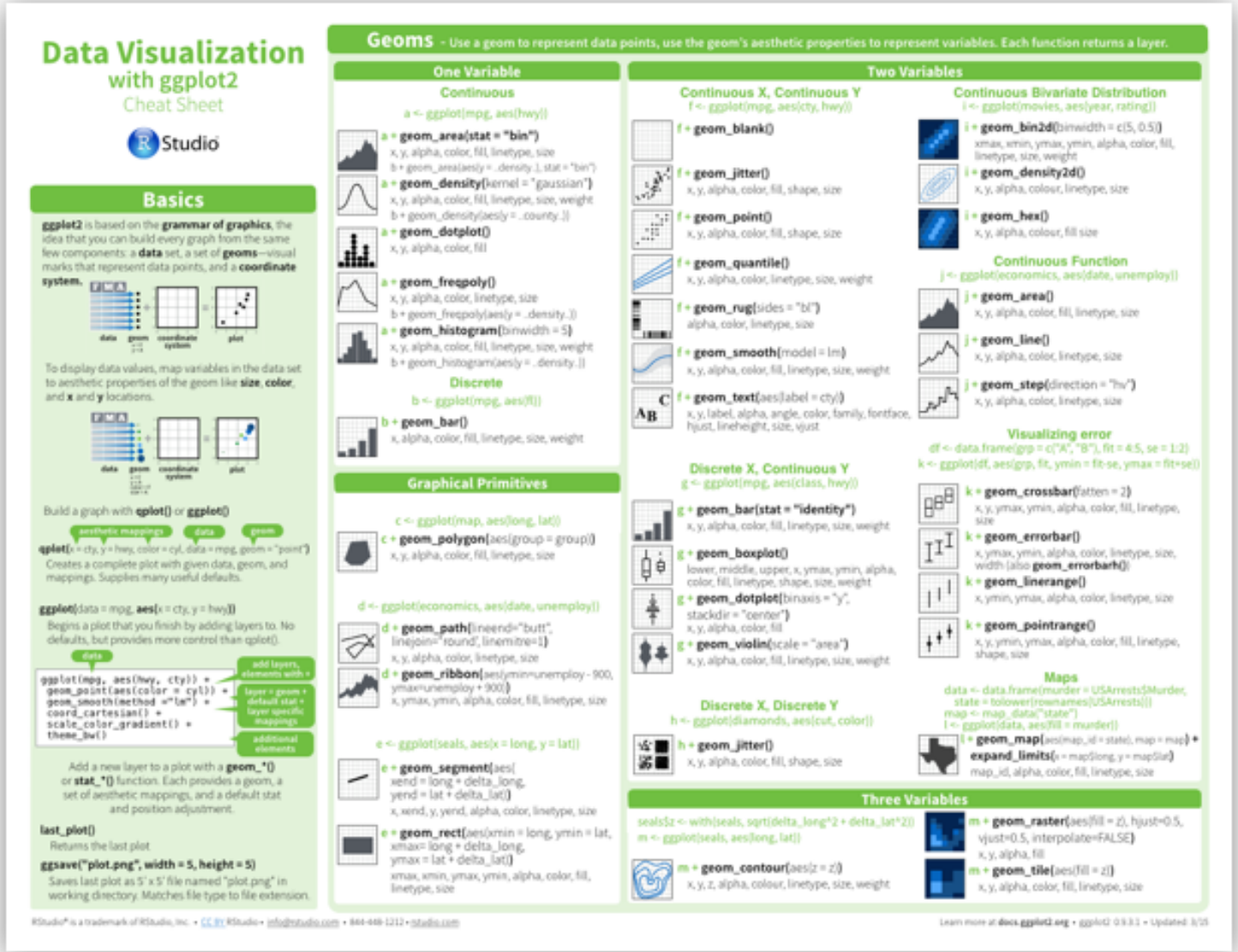
```
qmap("Mannheim Hbf", zoom = 14) +
  geom_path(
    aes(x = lon, y = lat),  colour = "red", size = 1.5,
    data = route_df, lineend = "round"
  )
```



Cheatsheet

- Cheatsheet zu data visualisation

`https://www.rstudio.com/`



Data Visualization with ggplot2 Cheat Sheet

RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.

To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.

Build a graph with **qplot()** or **ggplot()**

```
qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
```

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

```
ggplot(data = mpg, aes(x = cty, y = hwy))
```

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

```
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method ="lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

(add layers, elements with +; layer = geom + default stat + layer specific mapping(s); additional elements)

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()
Returns the last plot

ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

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))
```

- a + **geom_area(stat = "bin")** — x, y, alpha, color, fill, linetype, size
- a + **geom_density(kernel = "gaussian")** — x, y, alpha, color, fill, linetype, size, weight
- a + **geom_dotplot()** — x, y, alpha, color, fill
- a + **geom_freqpoly()** — x, y, alpha, color, linetype, size
- a + **geom_histogram(binwidth = 5)** — x, y, alpha, color, fill, linetype, size, weight

Discrete

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

- b + **geom_bar()** — x, alpha, color, fill, linetype, size, weight

Graphical Primitives

```
c <- ggplot(map, aes(long, lat))
```

- c + **geom_polygon(aes(group = group))** — x, y, alpha, color, fill, linetype, size

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

- d + **geom_path(lineend="butt", linejoin="round", linemitre=1)** — x, y, alpha, color, linetype, size
- d + **geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900))** — x, ymax, ymin, alpha, color, fill, linetype, size

```
e <- ggplot(seals, aes(x = long, y = lat))
```

- e + **geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))** — x, xend, y, yend, alpha, color, linetype, size
- e + **geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat))** — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

```
f <- ggplot(mpg, aes(cty, hwy))
```

- f + **geom_blank()**
- f + **geom_jitter()** — x, y, alpha, color, fill, shape, size
- f + **geom_point()** — x, y, alpha, color, fill, shape, size
- f + **geom_quantile()** — x, y, alpha, color, linetype, size, weight
- f + **geom_rug(sides = "bl")** — alpha, color, linetype, size
- f + **geom_smooth(model = lm)** — x, y, alpha, color, fill, linetype, size, weight
- f + **geom_text(aes(label = cty))** — x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Continuous Bivariate Distribution

```
i <- ggplot(movies, aes(year, rating))
```

- i + **geom_bin2d(binwidth = c(5, 0.5))** — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight
- i + **geom_density2d()** — x, y, alpha, colour, linetype, size
- i + **geom_hex()** — x, y, alpha, colour, fill size

Continuous Function

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

- j + **geom_area()** — x, y, alpha, color, fill, linetype, size
- j + **geom_line()** — x, y, alpha, color, linetype, size
- j + **geom_step(direction = "hv")** — x, y, alpha, color, linetype, size

Visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

- k + **geom_crossbar(fatten = 2)** — x, y, ymax, ymin, alpha, color, fill, linetype, size
- k + **geom_errorbar()** — x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)
- k + **geom_linerange()** — x, ymin, ymax, alpha, color, linetype, size
- k + **geom_pointrange()** — x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Discrete X, Continuous Y

```
g <- ggplot(mpg, aes(class, hwy))
```

- g + **geom_bar(stat = "identity")** — x, y, alpha, color, fill, linetype, size, weight
- g + **geom_boxplot()** — lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight
- g + **geom_dotplot(binaxis = "y", stackdir = "center")** — x, y, alpha, color, fill
- g + **geom_violin(scale = "area")** — x, y, alpha, color, fill, linetype, size, weight

Maps

```
data <- data.frame(murder = USArrests$Murder,
  state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))
```

- l + **geom_map(aes(map_id = state), map = map) + expand_limits(x = map$long, y = map$lat)** — map_id, alpha, color, fill, linetype, size

Discrete X, Discrete Y

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

- h + **geom_jitter()** — x, y, alpha, color, fill, shape, size

Three Variables

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))
```

- m + **geom_contour(aes(z = z))** — x, y, z, alpha, colour, linetype, size, weight
- m + **geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)** — x, y, alpha, fill
- m + **geom_tile(aes(fill = z))** — x, y, alpha, color, fill, linetype, size

Resourcen und Literatur

- Artikel von David Kahle und Hadley Wickham zur Nutzung von `ggmap`.

`http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf`

- Schnell eine Karte bekommen

`http://rpackages.ianhowson.com/cran/ggmap/man/get_map.html`

- Karten machen mit R

`http://www.kevjohnson.org/making-maps-in-r-part-2/`

Take Home Message

Was klar sein sollte:

- Wie man eine schnelle Karte erzeugt
- wie man geokodiert
- Wie man eine Distanz berechnet