Übung 1: Einfache Vektordaten

CAS FAB: Räumliche Daten in R

Nils Ratnaweera

Forschungsgruppe Geoinformatik

2021-11-30

Vorbereitung

- Erstelle ein neues RStudio Projekt
- Erstelle ein neues R-Script mit dem Namen `Uebung_1.R`
- Lade dir `Vegauf_Aussenberg_2019_Kopfdaten.csv` (von Moodle, Kurstag 12) herunter

Übung 1.1

- Importiere die CSV `Vegauf_Aussenberg_2019_Kopfdaten.csv` gewohnt als `data.frame` in R.
- Speichere die `data.frame` in der Variabel `ausserberg`

Übung 1.1

- Importiere die CSV `Vegauf_Aussenberg_2019_Kopfdaten.csv` gewohnt als `data.frame` in R.
- Speichere die `data.frame` in der Variabel `ausserberg`

Lösung

```
ausserberg <- read.csv("_data/original/Vegauf_Ausserberg_2019_Kopfdaten.csv", sep = "\t")

ausserberg[1:6, 1:6] # ich zeige nur die ersten 6 Spalten und Zeilen
```

```
##     Plot Verbuschung Vegetationstyp       Date geogr..Br. geogr..L.
## 1 VS19_1   Verbuscht        Trocken 18.06.2019    2631210   1129910
## 2 VS19_2 Unverbuscht    Halbtrocken 18.06.2019    2631175   1129752
## 3 VS19_3   Verbuscht    Halbtrocken 19.06.2019    2631411   1129900
## 4 VS19_4 Unverbuscht    Halbtrocken 18.06.2019    2631377   1129922
## 5 VS19_5 Unverbuscht        Trocken 18.06.2019    2631324   1129816
## 6 VS19_6 Unverbuscht        Trocken 19.06.2019    2631510   1129969
```

Übung 1.2

Such dir die Koordinaten im `data.frame` heraus. In welchem Koordinatensystem liegen diese wohl vor?

Übung 1.2

Such dir die Koordinaten im `data.frame` heraus. In welchem Koordinatensystem liegen diese wohl vor?

Lösung

in CH1903+ LV95

Übung 1.3

Visualisiere die Erhebungsplots räumlich als Scatterplot. Die x- und y-Achsen sind jetzt räumliche Koordinaten, auf was musst du achten?

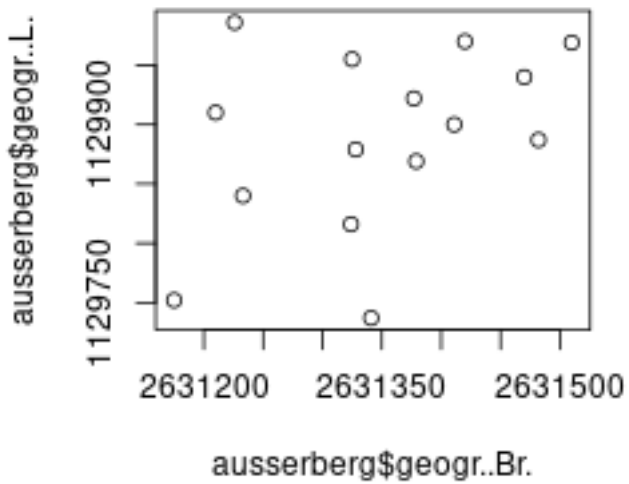
Übung 1.3

Visualisiere die Erhebungsplots räumlich als Scatterplot. Die x- und y-Achsen sind jetzt räumliche Koordinaten, auf was musst du achten?

Lösung

```
plot(ausserberg$geogr..Br., ausserberg$geogr
```

```
# zu beachten:
# - Reihenfolge der Koordinaten
# - asp = 1
```



Übung 1.4

Installiere nun das R-Package `sf` und lade es in die aktuelle Session.

Übung 1.4

Installiere nun das R-Package `sf` und lade es in die aktuelle Session.

Lösung

```
install.packages("sf") # installieren (mit Anführungs- und Schlusszeichen)
```

```
library(sf)            # laden (ohne Anführungs- und Schlusszeichen)
```

Übung 1.5

Wir machen nun aus dem `data.frame` `ausserberg` ein Vektor-Objekt und verwenden dazu die Funktion `st_as_sf()` aus der eben installierten Library `sf`.

Mit dem Argument `coords =` informieren wir dieser Funktion, wo unsere Koordinateninformation liegt. Probiere etwas rum bis es funktioniert und weise *danach* das Neue Objekt der Variabel `ausserberg_sf` zu.

Übung 1.5

Wir machen nun aus dem `data.frame` `ausserberg` ein Vektor-Objekt und verwenden dazu die Funktion `st_as_sf()` aus der eben installierten Library `sf`.

Mit dem Argument `coords =` informieren wir dieser Funktion, wo unsere Koordinateninformation liegt. Probiere etwas rum bis es funktioniert und weise *danach* das Neue Objekt der Variabel `ausserberg_sf` zu.

Lösung

```
ausserberg_sf <- st_as_sf(ausserberg, coords = c("geogr..Br.","geogr..L."))
```

Übung 1.6

Vergleiche nun `ausserberg` und `ausserberg_sf` in der Konsole. Wodurch unterscheiden sie sich?

Übung 1.6

Vergleiche nun `ausserberg` und `ausserberg_sf` in der Konsole. Wodurch unterscheiden sie sich?

Lösung

- `ausserberg_sf` hat die beiden Koordinaten-Spalten verloren und verfügt dafür neu über eine Spalte `geometry`.
- `ausserberg_sf` verfügt nun über Metadaten im header:

```
Simple feature collection with 15 features and 34 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 2631175 ymin: 1129737 xmax: 2631510 ymax: 1129986
CRS:           NA
```

Übung 1.6

```
Simple feature collection with 15 features and 34 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 2631175 ymin: 1129737 xmax: 2631510 ymax: 1129986
CRS:           NA
```

Übung 1.6

```
Simple feature collection with 15 features and 34 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 2631175 ymin: 1129737 xmax: 2631510 ymax: 1129986
CRS:           NA
```

Wir haben nirgends deklariert, in welchem Koordinatenbezugssystem sich unsere Koordinaten befinden.

Input

Nun wollen wir unserem Datensatz das richtige Koordinatenreferenzsystem zuweisen. Wie sprechen wir das korrekte Koordinatensystem `CH1903+ LV95` an?

Input

Nun wollen wir unserem Datensatz das richtige Koordinatenreferenzsystem zuweisen. Wie sprechen wir das korrekte Koordinatensystem `CH1903+ LV95` an?

Im Wesentlichen gibt es 3 Methoden, ein Koordinatenreferenzsystem anzusprechen:

- proj.4
- Well known text wkt
- EPSG

proj.4

- In einem `proj.4`-string werden alle wichtige Aspekte des Koordinatenreferensystems abgespeichert (ellipse, datum, projection units)
- der `proj.4`-strings verwenden ein `key=value` system, die mit `+` kombiniert werden
- der `proj.4`-string von `CH1903+LV95` sieht folgendermassen aus:

```
+proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333 +k_0=1
+x_0=2600000 +y_0=1200000 +ellps=bessel +towgs84=674.374,15.056,405.346,0,0,0,0
+units=m +no_defs
```

Well known text wkt

- Logik ähnlich wie `proj.4`-strings
- verwenden einen anderen Syntax (`key[value]`)
- der wkt von `CH1903+LV95` sieht folgendermassen aus

```
PROJCS["CH1903+ / LV95",
    GEOGCS["CH1903+",
        DATUM["CH1903+",
            SPHEROID["Bessel 1841",6377397.155,299.1528128,
                AUTHORITY["EPSG","7004"]],
            TOWGS84[674.374,15.056,405.346,0,0,0,0],
            AUTHORITY["EPSG","6150"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4150"]],
    PROJECTION["Hotine_Oblique_Mercator_Azimuth_Center"],
    PARAMETER["latitude_of_center",46.95240555555556],
    PARAMETER["longitude_of_center",7.439583333333333],
    PARAMETER["azimuth",90],
    PARAMETER["rectified_grid_angle",90],
    PARAMETER["scale_factor",1],
    PARAMETER["false_easting",2600000],
    PARAMETER["false_northing",1200000],
```

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen
- am besten ist, man notiert sich die EPSG Codes unserer vier Wichtigsten Koordinatenbezugssysteme:

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen
- am besten ist, man notiert sich die EPSG Codes unserer vier Wichtigsten Koordinatenbezugssysteme:
  - `CH1903+ LV95`: `2056`

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen
- am besten ist, man notiert sich die EPSG Codes unserer vier Wichtigsten Koordinatenbezugssysteme:
  - `CH1903+ LV95`: `2056`
  - `CH1903 LV03`: `21781`

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen
- am besten ist, man notiert sich die EPSG Codes unserer vier Wichtigsten Koordinatenbezugssysteme:
  - `CH1903+ LV95`: `2056`
  - `CH1903 LV03`: `21781`
  - `WGS84`: `4326`

EPSG Code

- die European Petroleum Survey Group (EPSG): ein wissenschaftliche Organisation (Geodäsie, Vermessung und Kartographie)
- öffentliche Datenbank um Koordinatenbezugssysteme (sowie Ellipsoide und Geodätisches Datumsangaben) festzuhalten
- jede Einträge hat einen Referenz Code (siehe epsg.io)
- Wie lautet der EPSG Code für `CH1903+LV95`?
- der EPSG Code ist der einfachste Weg, ein Koordinatenbezugssystem anzusprechen
- am besten ist, man notiert sich die EPSG Codes unserer vier Wichtigsten Koordinatenbezugssysteme:
  - `CH1903+ LV95`: `2056`
  - `CH1903 LV03`: `21781`
  - `WGS84`: `4326`
  - `WGS 84 / Pseudo-Mercator`: `3857`

Übung 1.8

Weise nun unserem Datensatz das richtige Koordinatensystem zu. Dafür brauchst du die Funktion `st_crs` sowie den EPSG Code des Koordinatensystems.

Übung 1.8

Weise nun unserem Datensatz das richtige Koordinatensystem zu. Dafür brauchst du die Funktion `st_crs` sowie den EPSG Code des Koordinatensystems.

Lösung

```
st_crs(ausserberg_sf) <- 2056
```

```
ausserberg_sf
```

```
Simple feature collection with 15 features and 34 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 2631175 ymin: 1129752 xmax: 2631210 ymax: 1129910
Projected CRS: CH1903+ / LV95
```

Übung 1.9

- `R` weiss nun, das es sich bei `aussenberg_sf`um einen Vektordatensatz handelt
- `aussenberg_sf` reagiert nun anders auf gewisse functions
- teste die Funktion `plot` mit `aussenberg_sf`

Übung 1.9

- `R` weiss nun, das es sich bei `aussenberg_sf`um einen Vektordatensatz handelt
- `aussenberg_sf` reagiert nun anders auf gewisse functions
- teste die Funktion `plot` mit `aussenberg_sf`

Lösung

```
# plot(ausserberg_sf) <- macht einen Plot pro Spalte, maximal 9
plot(ausserberg_sf["Verbuschung"]) # Plottet nur die ausgewählte Spalte
```

Input

- In `R` gibt es dezidierte libraries, um geografische Daten zu visualisieren
- Wir werden im Unterricht die library `tmap` verwenden.
- Installiere dieses Package und lade es in die aktuelle session.

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

```
library(tmap)
```

Input

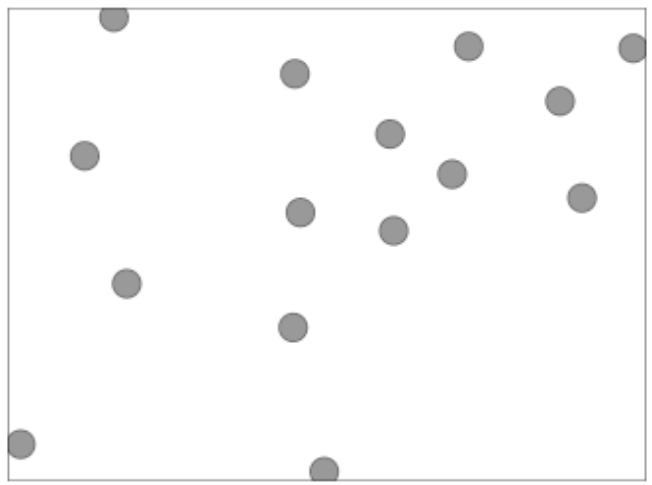
- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

```
tm_shape(ausserberg_sf) + # datensatz
tm_dots()               # darstellungsfor
```

Input

- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

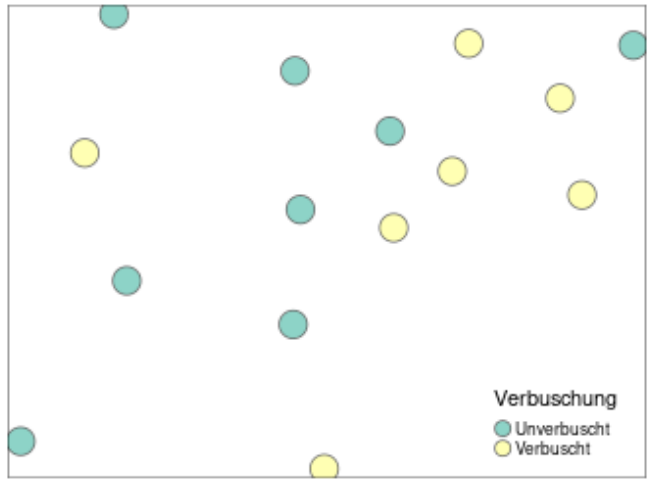
```
tm_shape(ausserberg_sf) + # datensatz
tm_bubbles()              # darstellungsfor
```



Input

- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

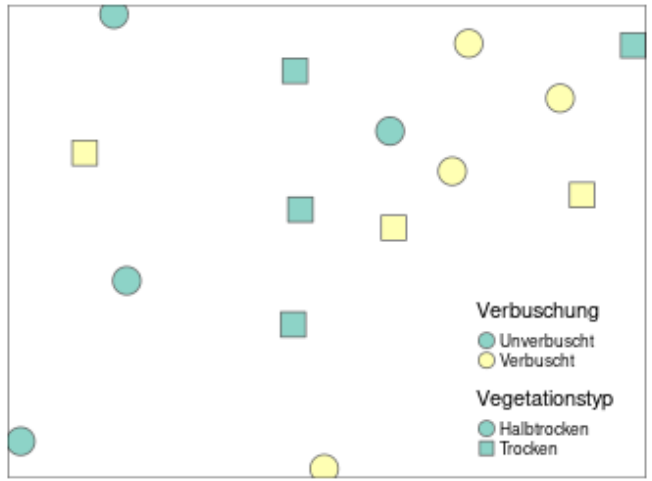
```
tm_shape(ausserberg_sf) +
tm_bubbles(col = "Verbuschung")
```



Input

- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

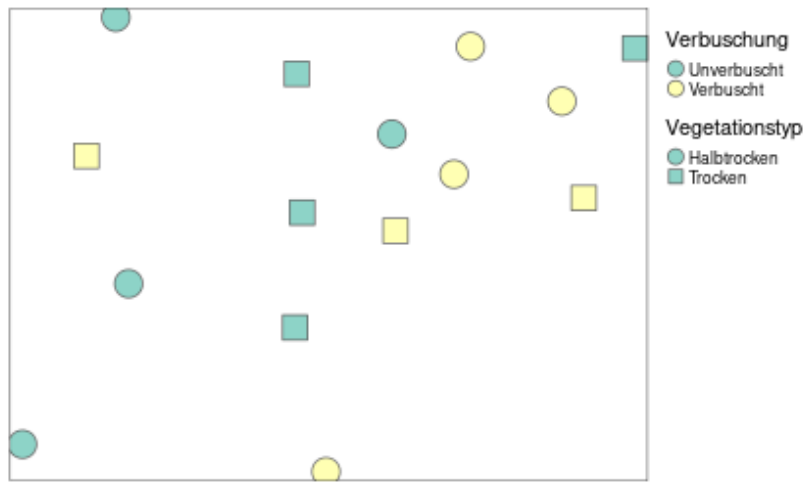
```
tm_shape(ausserberg_sf) +
tm_bubbles(col = "Verbuschung",
     shape = "Vegetationstyp")
```



Input

- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

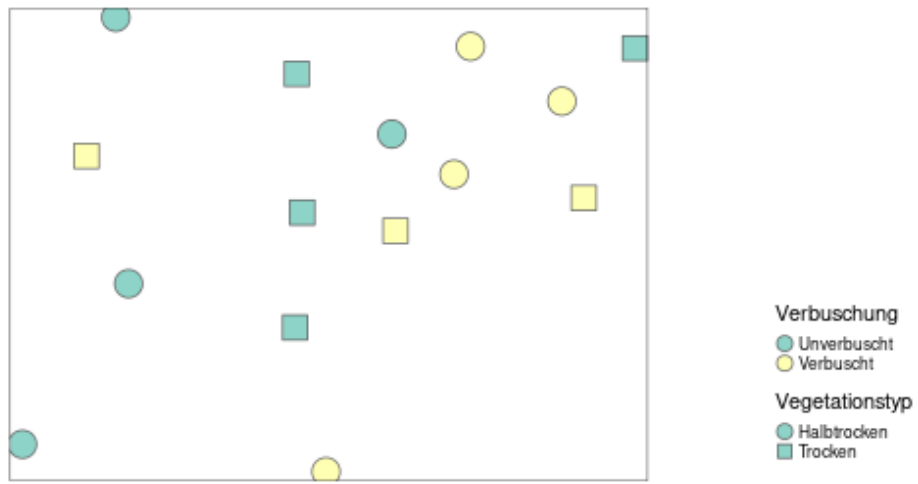
```
tm_shape(ausserberg_sf) +
tm_bubbles(col = "Verbuschung",
     shape = "Vegetationstyp") +
tm_layout(legend.outside = TRUE)
```



Input

- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

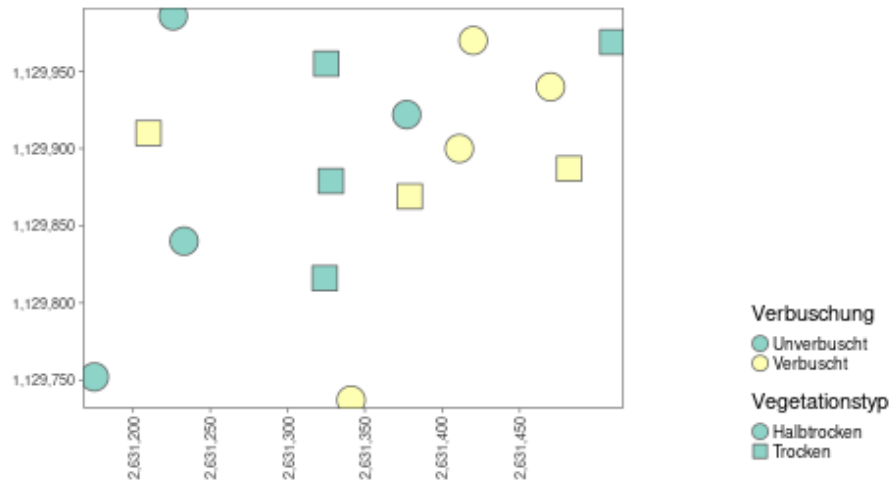
```
tm_shape(ausserberg_sf) +
tm_bubbles(col = "Verbuschung",
     shape = "Vegetationstyp") +
tm_layout(legend.outside = TRUE,
    legend.position = c("right","bottom")
```



Input

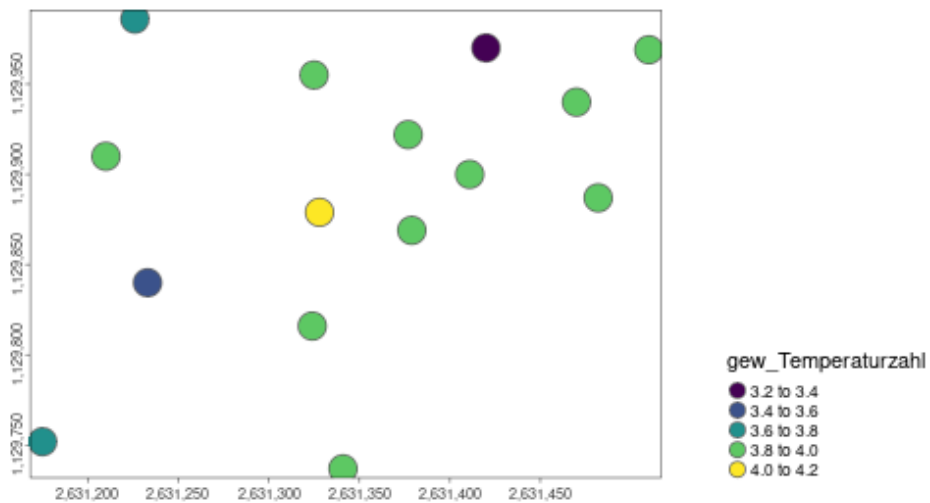
- `tmap` funktioniert nach einem "layer"-Prinzip
- ein Layer besteht aus 2 Komponenten:
  - `tm_shape()`: der Datensatz
  - `tm_dots` (oder `tm_lines`, `tm_polygons`...): die *Darstellungsform*

```
tm_shape(ausserberg_sf) +
tm_bubbles(col = "Verbuschung",
     shape = "Vegetationstyp") +
tm_layout(legend.outside = TRUE,
    legend.position = c("right","bottom")
tm_grid(labels.rot = c(90, 0), lines = FA
```



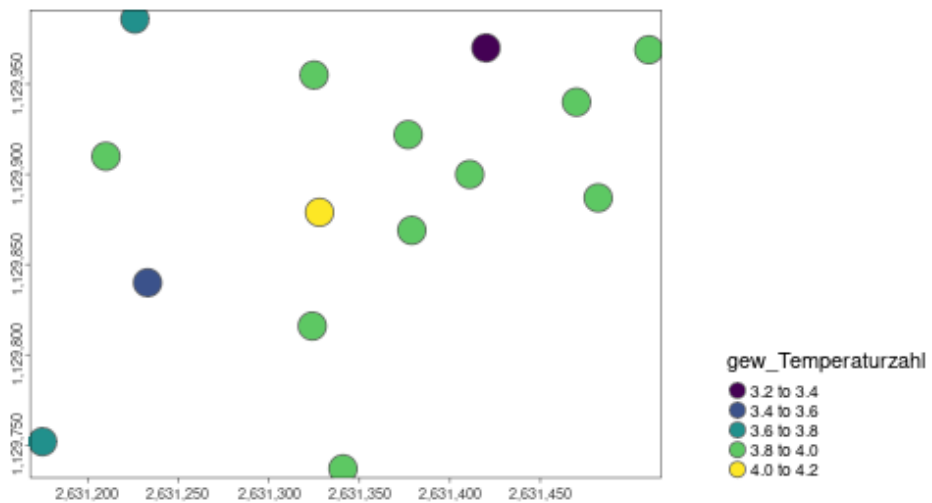
Übung 1.10

Erstellt nun eine eigene Karte mit `tmap` und euren Daten. Versucht, den unten stehenden Plot zu rekonstruieren (oder probiert was eigenes).



Übung 1.10

Erstellt nun eine eigene Karte mit `tmap` und euren Daten. Versucht, den unten stehenden Plot zu rekonstruieren (oder probiert was eigenes).



Lösung

```
tm_shape(ausserberg_sf) +
  tm_bubbles(col = "gew_Temperaturzahl", pal
  tm_layout(legend.outside = TRUE,
            legend.position = c("right","bot
  tm_grid(labels.rot = c(0, 90),
          lines = FALSE)
```

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*
  - verändert die Koordinatenwerte *nicht*,

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*
  - verändert die Koordinatenwerte *nicht*,
  - ist nur sinnvoll, wenn das Koordinatensystem nicht oder falsch zugewiesen wurde

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*
  - verändert die Koordinatenwerte *nicht*,
  - ist nur sinnvoll, wenn das Koordinatensystem nicht oder falsch zugewiesen wurde
- Koordinatenbezugssystem*transformieren*

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*
  - verändert die Koordinatenwerte *nicht*,
  - ist nur sinnvoll, wenn das Koordinatensystem nicht oder falsch zugewiesen wurde
- Koordinatenbezugssystem*transformieren*
  - verändert die Koordinatenwerte

Input

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinantensystem in ein anderes übersetzen (*transformieren*)
- Wichtig unterschied!
- Koordinatenbezugssystem *zuweisen*
  - verändert die Koordinatenwerte *nicht*,
  - ist nur sinnvoll, wenn das Koordinatensystem nicht oder falsch zugewiesen wurde
- Koordinatenbezugssystem*transformieren*
  - verändert die Koordinatenwerte
  - ist unter verschiedenen Szenarien sinnvoll (um versch. Datequellen zu integrieren)

Übung 1.11

- Transformiert `ausserberg_sf` in das Koordinatenbezugssystem WGS84
- Speichert den output in einer neuen Variabel (z.B `ausserberg_sf_wgs84`)
- Schaut euch diesen Datensatz an, was hat sich verändert?

Übung 1.11

- Transformiert `ausserberg_sf` in das Koordinatenbezugssystem WGS84
- Speichert den output in einer neuen Variabel (z.B `ausserberg_sf_wgs84`)
- Schaut euch diesen Datensatz an, was hat sich verändert?
- Tipp: Nutzt dafür die Funktion `st_transform()`

Übung 1.11

- Transformiert `ausserberg_sf` in das Koordinatenbezugssystem WGS84
- Speichert den output in einer neuen Variabel (z.B `ausserberg_sf_wgs84`)
- Schaut euch diesen Datensatz an, was hat sich verändert?
- Tipp: Nutzt dafür die Funktion `st_transform()`

Lösung

```
ausserberg_sf_wgs84 <- st_transform(ausserberg_sf, 4326)
```

Die Metadaten haben sich verändert (vorher: `Projected CRS: CH1903+ / LV95`)

```
Simple feature collection with 15 features and 34 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 7.843394 ymin: 46.3183 xmax: 7.847758 ymax: 46.32055
Geodetic CRS:  WGS 84
```

Zudem haben sich die Koordinatenwerte verändert. Neu: `POINT (7.843859 46.31987)`

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
    1. CSV als Dataframe einlesen

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
    1. CSV als Dataframe einlesen
    2. CSV in `sf` objekt konvertieren

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
    1. CSV als Dataframe einlesen
    2. CSV in `sf` objekt konvertieren
    3. CRS Zuweisen

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
  1. CSV als Dataframe einlesen
  2. CSV in `sf` objekt konvertieren
  3. CRS Zuweisen
- Wir können `ausserberg_sf` in einem explizit *räumlichen* Datenformat abspeichern, sodass die obigen Schritte beim importieren nicht nötig sind:

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
  1. CSV als Dataframe einlesen
  2. CSV in `sf` objekt konvertieren
  3. CRS Zuweisen
- Wir können `ausserberg_sf` in einem explizit *räumlichen* Datenformat abspeichern, sodass die obigen Schritte beim importieren nicht nötig sind:

```
write_sf(ausserberg_sf, "_data/processed/ausserberg.gpkg")
```

Input

- CSVs eignen sich nur bedingt um räumliche Daten abzuspeichern
- Um aus `Vegauf_Ausserberg_2019_Kopfdaten.csv` ein räumliches Objekt zu machen mussten wir verschiedene Schritte erledigen
  1. CSV als Dataframe einlesen
  2. CSV in `sf` objekt konvertieren
  3. CRS Zuweisen
- Wir können `ausserberg_sf` in einem explizit *räumlichen* Datenformat abspeichern, sodass die obigen Schritte beim importieren nicht nötig sind:

```
write_sf(ausserberg_sf, "_data/processed/ausserberg.gpkg")
```

Beim Einlesen von `ausserberg.gpkg` ist R nun sofort klar, dass es sich um Punktdaten im Koordinatenbezugssystem `EPSG 2056` handelt.

```
ausserberg_sf <- read_sf("_data/processed/ausserberg.gpkg")
```

Übung 1.13

- Importiere nun aus dem Excel Hagenmoos.xlsx das Datenblatt `KopfdatenVertikal` als `data.frame`.
- Konvertiere den Dataframe in ein `sf` objekt
- Weise das korrekte Koordinatensytem zu
- Transformiere die Koordinaten anschliessend in WGS84
- erstelle eine Karte mit `tmap`

Übung 1.13

- Importiere nun aus dem Excel Hagenmoos.xlsx das Datenblatt `KopfdatenVertikal` als `data.frame`.
- Konvertiere den Dataframe in ein `sf` objekt
- Weise das korrekte Koordinatensytem zu
- Transformiere die Koordinaten anschliessend in WGS84
- erstelle eine Karte mit `tmap`

Lösung

```
hangenmoos <- readxl::read_excel("_data/original/daten_vegedaz/Hagenmoos.xlsx", "KopfdatenVerti

hangenmoos_sf <- st_as_sf(hangenmoos, coords = c("X", "Y"))
st_crs(hangenmoos_sf) <- 21781

hangenmoos_sf_wgs84 <- st_transform(hangenmoos_sf, 4326)

tm_shape(hangenmoos_sf_wgs84) +
  tm_bubbles(col = "Bereich")  +
  tm_layout(legend.outside = TRUE)
```

Input

- Der Datensatz `Hangenmoos` beinhaltet Erhebungen an den gleichen Standorten in verschiedenen Jahren.

Input

- Der Datensatz `Hangenmoos` beinhaltet Erhebungen an den gleichen Standorten in verschiedenen Jahren.
- Dies führt dazu, dass sich Punkte überlagern (gleiche Koordinaten)

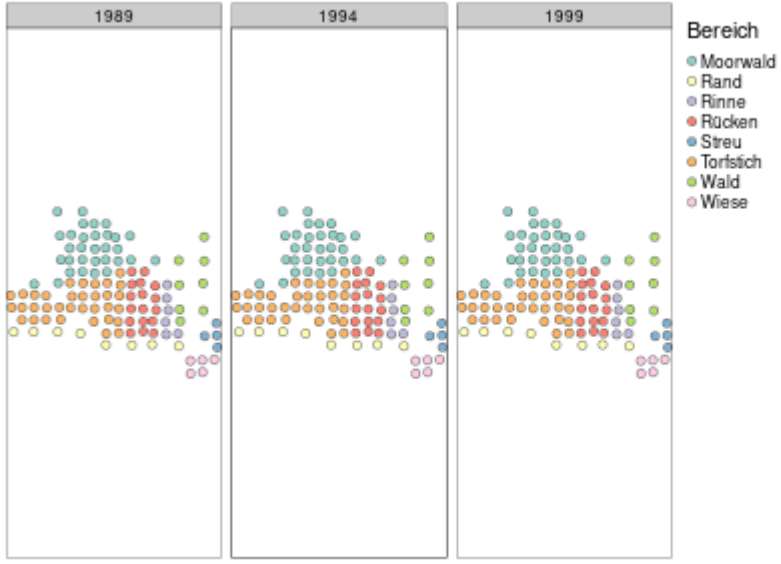
Input

- Der Datensatz `Hangenmoos` beinhaltet Erhebungen an den gleichen Standorten in verschiedenen Jahren.
- Dies führt dazu, dass sich Punkte überlagern (gleiche Koordinaten)
- Um dies zu vermeiden, können wir mit der `facet` option in `tmap` arbeiten

Input

- Der Datensatz `Hangenmoos` beinhaltet Erhebungen an den gleichen Standorten in verschiedenen Jahren.
- Dies führt dazu, dass sich Punkte überlagern (gleiche Koordinaten)
- Um dies zu vermeiden, können wir mit der `facet` option in `tmap` arbeiten

```
tm_shape(hangenmoos_sf_wgs84) +
  tm_bubbles(size = .2, col = "Bereich")  +
  tm_layout(legend.outside = TRUE) +
  tm_facets("Jahr",nrow = 1)
```



Aufgabe 1.14

- Bisher haben wir nur statische Karten (ohne Hintergrundkarte) erstellt.

Aufgabe 1.14

- Bisher haben wir nur statische Karten (ohne Hintergrundkarte) erstellt.
- Mit `tmap` lassen sich aber auch sehr leicht interaktive Karten erstellen

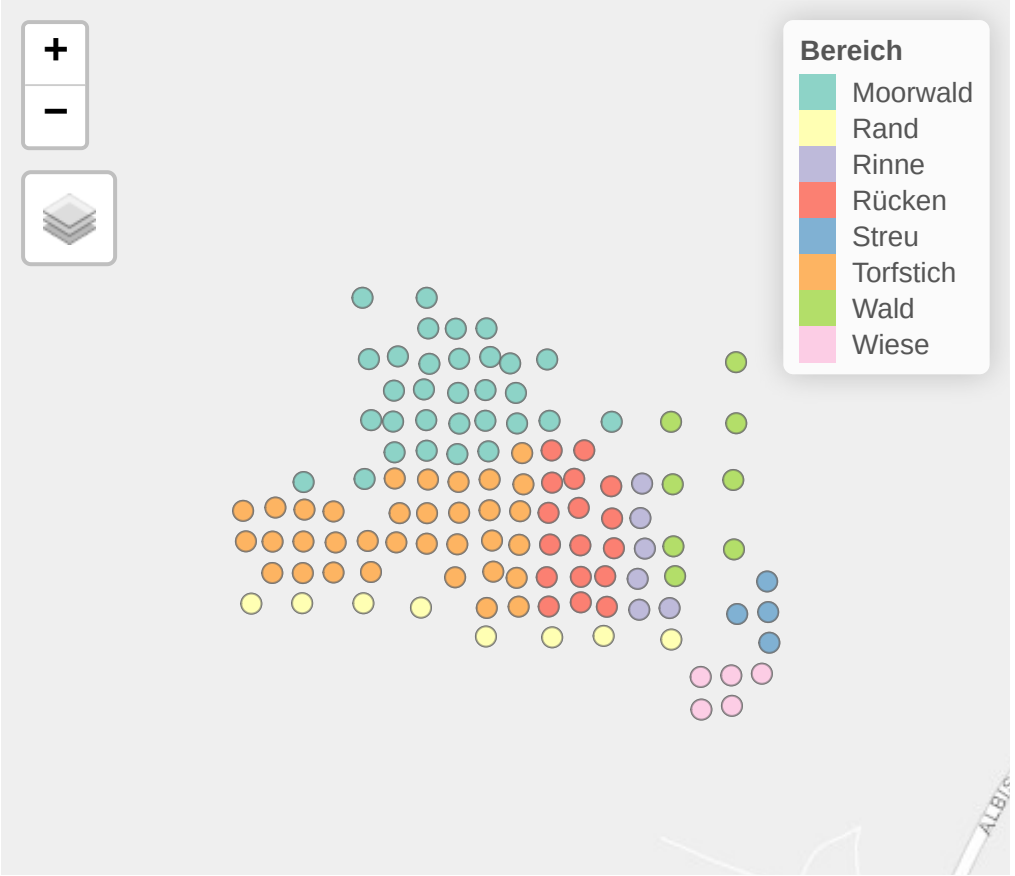
Aufgabe 1.14

- Bisher haben wir nur statische Karten (ohne Hintergrundkarte) erstellt.
- Mit `tmap` lassen sich aber auch sehr leicht interaktive Karten erstellen
  - Setze dafür `tmap_mode("view")` und führe dein letzter Code für die Erstellung eines tmap-Plots nochmals aus

Lösung

```
tmap_mode("view")

tm_shape(hangenmoos_sf_wgs84) +  tm_bubbles(col = "Bereich")  +  tm_layout(legend.outside = TRU
```



Rückblick

- Bisher haben wir mit Vektordaten vom Typ `Point` gearbeitet
- Das dem zugrundeliegende, konzeptionelle Datenmodell ist das Entitäten Modell
- Diese Punktdaten waren in einem csv sowie einem xlsx Dateiformat abgespeichert
- In `R` haben wir diese Punktdaten als `data.frame` importiert und danach in ein `sf` Objekt konvertiert
- `sf`-Objekte zeichnen sich dadurch aus, dass sie über eine Geometriespalte sowie über Metadaten verfügen

Ausblick

- Punktdaten lassen sich gut in CSV abspeichern, weil sich die Geometrie so gut vorhersehbar ist (jeder Punkte besteht aus genau einer x- und einer y-Koordinate)
- Linien und Polygone sind komplexer, sie können aus beliebig vielen Knoten bestehen
- Es bessere Wege, räumliche Daten abzuspeichern
- Das bekannteste Format für Vektordaten ist das *shapefile*
- Shapefiles haben aber Nachteile ein sinnvolleres Format ist deshalb *geopackage*