

CAS Vegetationsanalyse und Feldbotanik

Geografische Kartendarstellung mit R

Nils Ratnaweera

2023-11-17

Inhaltsverzeichnis

Einführung	3
Übung 1	4
Vorbereitung	4
Übung 1.1	4
Übung 1.2	5
Übung 1.3	5
Übung 1.4	5
Übung 1.5	6
Übung 1.6	6
Input CRS ansprechen	7
Übung 1.8	9
Übung 1.9	10
Input tmap	10
Übung 1.10	11
Input CRS wechseln	13
Übung 1.11	14
Übung 1.12 (Optional)	14
Input Räumliche Datenformate	15
Übung 1.13	16
Input Small Multiples	17
Übung 1.14	17
Fazit	18
Übung 2	20
Vorbereitung	20
Übung 2.1	20
Übung 2.2	21
Übung 2.3	21
Übung 2.4	21
Übung 2.5	21
Übung 2.6	22
Übung 3	24
Vorbereitung	24

Übung 3.1	24
Input: Raster Datenformate	25
Raster in R	26
Übung 3.2	27
Übung 3.2	27
Lösung	28
Übung 3.3	28
Input: Koordinatenbezugssystem <i>festlegen</i>	29
Input: Koordinatenbezugssystem <i>transformieren</i>	30
Übung 3.4	30
Übung 3.5	31
Übung 3.6	32
Input Raster exportieren	32
Übung 3.7	33
Übung 4	34
Übung 4.1	34
Übung 4.2	35
Input: RGB Plots mit <code>tmap</code>	36
Input	37
Übung 4.4	37
Übung 4.5 (Optional und Open End)	38
Übung 4.6 (Optional und Open End)	38
Übung 4.7	39
Übung 5	41
Übung 5.1	41
Übung 5.2	42
Übung 5.3	42
Übung 5.4	43
Input: Rasterwerte extrahieren	44
Übung 5.5	44
Übung 5.6	45
Input: Vektordaten zuschneiden	47
Übung 5.8	47
Übung 5.9	48
Übung 5.10	49
Übung 5.11	49
Input: Vektordaten selektieren	50
Übung 5.12	51
Übung 5.13	52

Übung 6	53
Vorbereitung	54
Zufällige Verteilung	55
Zufällig - Nicht stratifiziert	56
Zufällig - Stratifiziert	56
Systematisch / Regelmässig	57
Übung	58

Einführung

Musterlösungen

Bei diesen Unterlagen handelt es sich um die Musterlösungen zu den Aufgaben vom Kurstag 17 (*Geografische Kartendarstellung mit R*) des [CAS Vegetationsanalyse und Feldbotanik](#).

Die Musterlösungen stehen auch als PDF zur Verfügung (siehe PDF-Symbol oberhalb des Menübalkens links). Die Aufgabenstellungen selbst sind den Übungen jeweils vorangestellt, eine Version der Übungen ohne Musterlösungen befindet sich hier: cas-vegetationsanalyse-feldbotanik.github.io/HS23.

Übung 1

Einfache Vektordaten

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

Musterlösung

```
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?

 Musterlösung

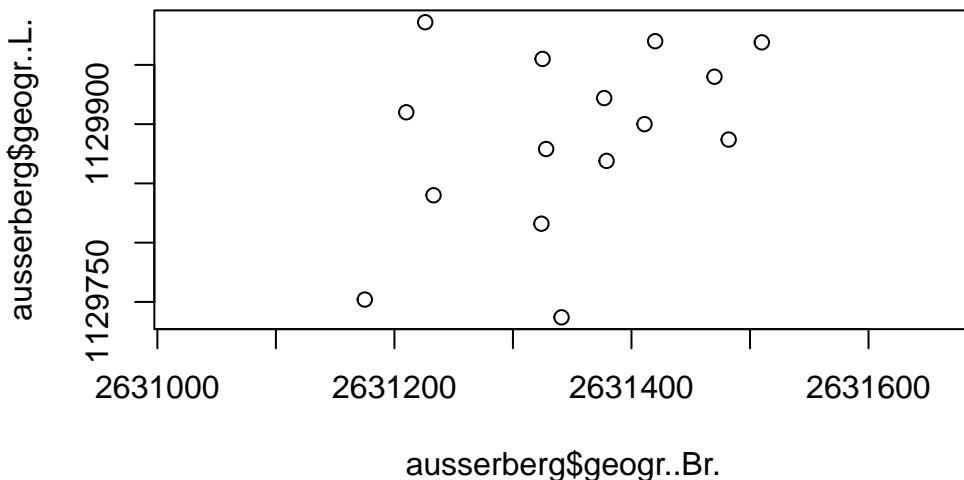
In CH1903+ LV95

Übung 1.3

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

 Musterlösung

```
plot(ausserberg$geogr..Br., ausserberg$geogr..L., asp = 1)
```



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

Übung 1.4

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

Musterlösung

```
install.packages("sf")
```

```
library("sf")
```

```
Linking to GEOS 3.12.0, GDAL 3.7.1, PROJ 9.2.1; sf_use_s2() is TRUE
```

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.

Musterlösung

```
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?

Musterlösung

- `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
```

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

Input CRS ansprechen

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

i proj.4 (optional)

- In einem proj.4-string werden alle wichtige Aspekte des Koordinatenreferenzsystems 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
```

i Well known text wkt (optional)

- 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], 
            UNIT["metre",1,
                AUTHORITY["EPSG","9001"]], 
            AXIS["Y",EAST], 
            AXIS["X",NORTH], 
            AUTHORITY["EPSG","2056"]]

```

! EPSG Code (wichtig)

- 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 wichtigsten Koordinatenbezugssysteme:

Koordinatenbezugssystem	EPSG Code	Kommentar	Einheit
CH1903+ LV95	2056	Neues Koordinatensystem der Schweiz	Meter
CH1903 LV03	21781	Altes Koordinatensystem der Schweiz	Meter
WGS84	4326	Weltweites Koordinatensystem	Grad

Übung 1.8

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

i Musterlösung

```
st_crs(ausserberg_sf) <- 2056
ausserberg_sf[1:4, 1:6]
```

```
Simple feature collection with 4 features and 6 fields
Geometry type: POINT
Dimension: XY
Bounding box: xmin: 2631175 ymin: 1129752 xmax: 2631411 ymax: 1129922
Projected CRS: CH1903+ / LV95
  Plot Verbuschung Vegetationstyp      Date Genauigkeit Meereshoehe
1 VS19_1   Verbuscht      Trocken 18.06.2019      5       1298
2 VS19_2 Unverbuscht    Halbtrocken 18.06.2019      5       1263
3 VS19_3   Verbuscht    Halbtrocken 19.06.2019     10       1282
4 VS19_4 Unverbuscht    Halbtrocken 18.06.2019      3       1295
  geometry
1 POINT (2631210 1129910)
2 POINT (2631175 1129752)
3 POINT (2631411 1129900)
4 POINT (2631377 1129922)
```

```
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
```

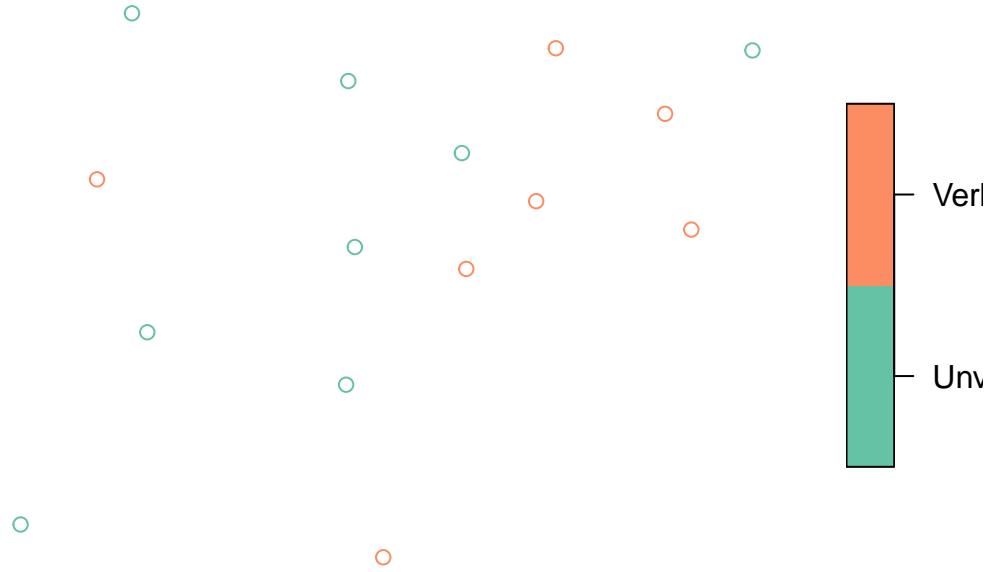
Übung 1.9

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

i Musterlösung

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

Verbuschung



Input tmap

- 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")
```

- **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*

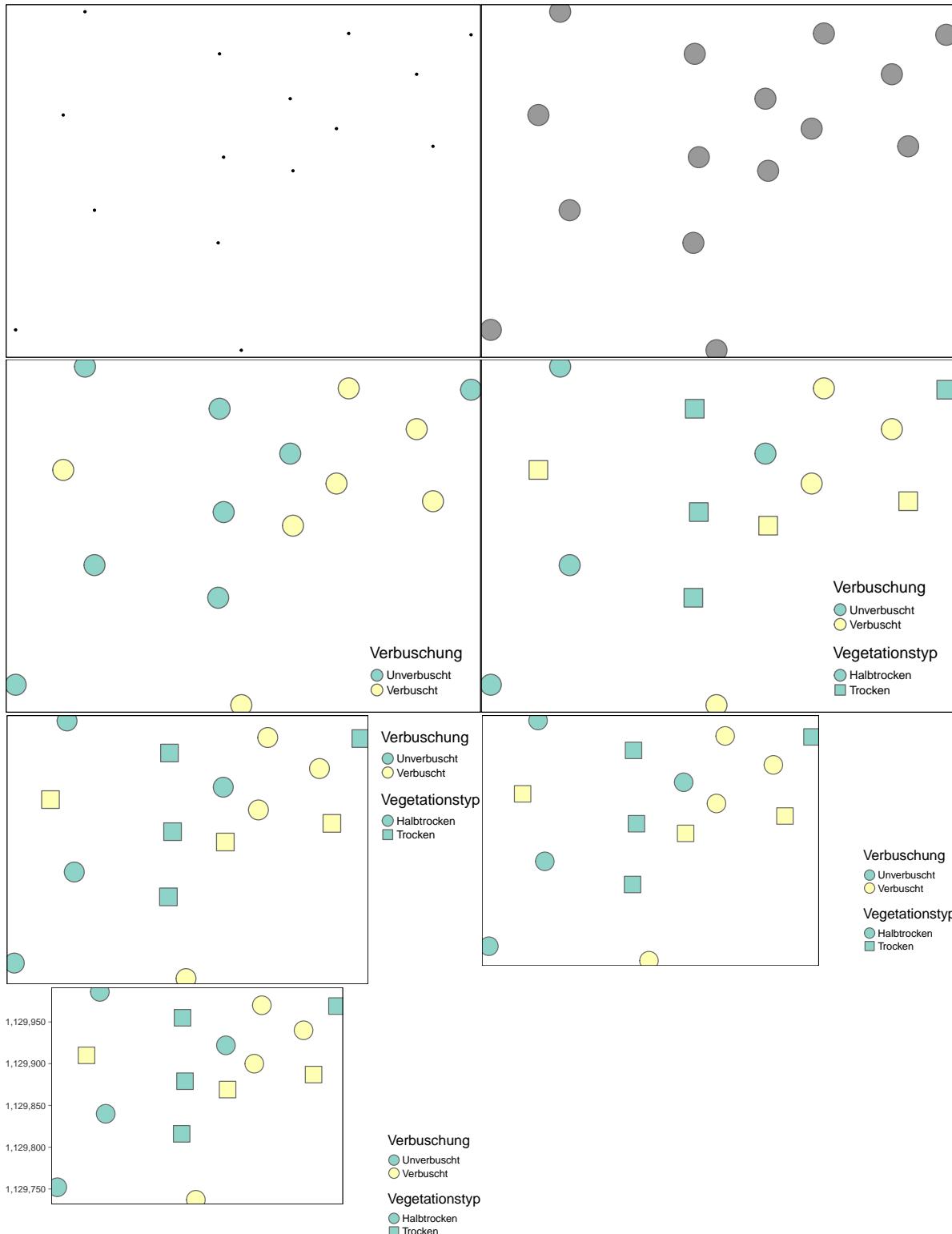
```

tm_shape(ausserberg_sf) + # datensatz
  tm_dots()               # darstellungsform
tm_shape(ausserberg_sf) +
  tm_bubbles()
tm_shape(ausserberg_sf) +
  tm_bubbles(col = "Verbuschung")
tm_shape(ausserberg_sf) +
  tm_bubbles(col = "Verbuschung",
             shape = "Vegetationstyp")
tm_shape(ausserberg_sf) +
  tm_bubbles(col = "Verbuschung",
             shape = "Vegetationstyp") +
  tm_layout(legend.outside = TRUE)
tm_shape(ausserberg_sf) +
  tm_bubbles(col = "Verbuschung",
             shape = "Vegetationstyp") +
  tm_layout(legend.outside = TRUE,
            legend.position = c("right","bottom"))
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 = FALSE)

```

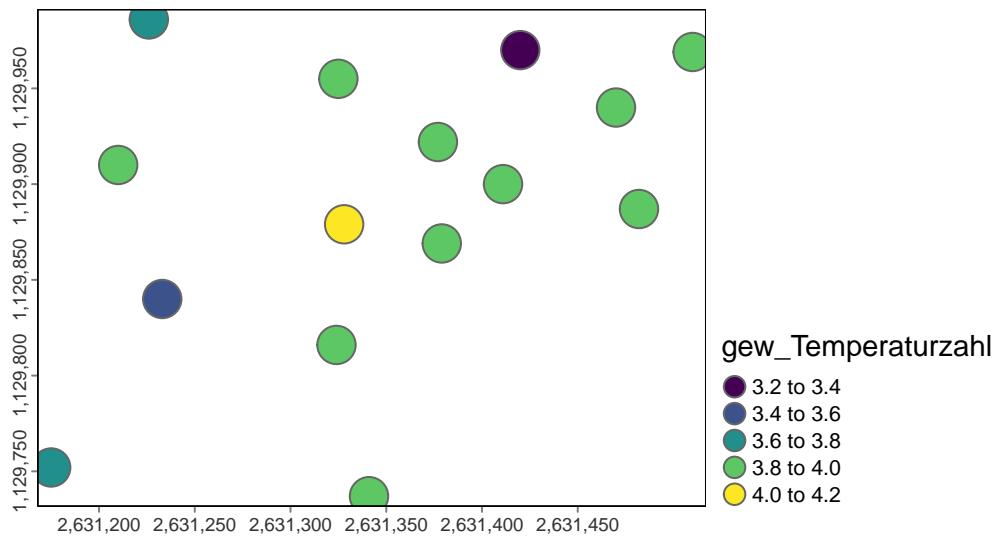
Übung 1.10

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



Musterlösung

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



Input CRS wechseln

- Bisher: Dem Datensatz ein Koordinatensystem *zuweisen*.
- Auch häufig: Koordinaten vom einen Koordinatensystem 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?
- Tipp: Nutzt dafür die Funktion `st_transform()`

i Musterlösung

```
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
```

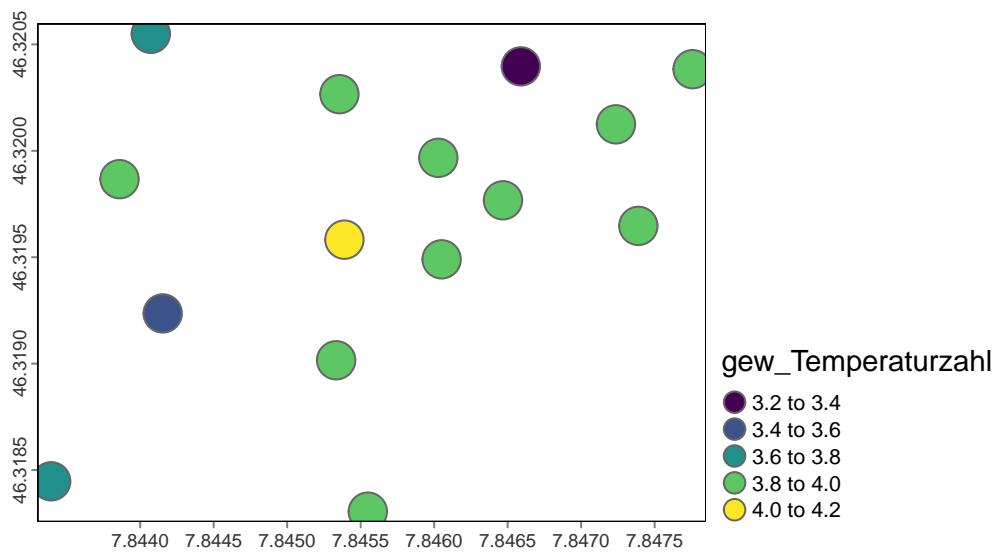
Zudem haben sich die Koordinatenwerte verändert. Neu: POINT (7.843859 46.31987)

Übung 1.12 (Optional)

Wiederhole nochmal den letzten `tmap plot`, diesmal mit dem Datensatz `ausserberg_sf_wgs84`. Wie unterscheiden sich die Plots?

i Musterlösung

```
tm_shape(ausserberg_sf_wgs84) +
  tm_bubbles(col = "gew_Temperaturzahl", palette = "viridis") +
  tm_layout(legend.outside = TRUE,
            legend.position = c("right", "bottom")) +
  tm_grid(labels.rot = c(0, 90),
          lines = FALSE)
```



- Die markierten Koordinaten haben sich verändert
- Kaum merklich haben sich auch die Positionen der Punkte verändert (unterschiedliches Datum)

Input Räumliche Datenformate

- 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.

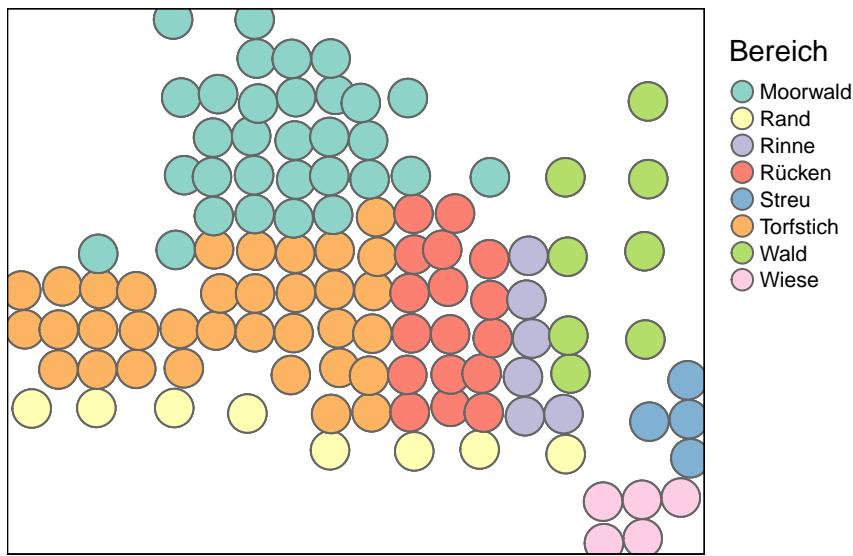
```
ausserberg_sf <- read_sf("data/processed/ausserberg.gpkg")
```

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 Koordinatensystem zu
- Transformiere die Koordinaten anschliessend in WGS84
- erstelle eine Karte mit `tmap`

i Musterlösung

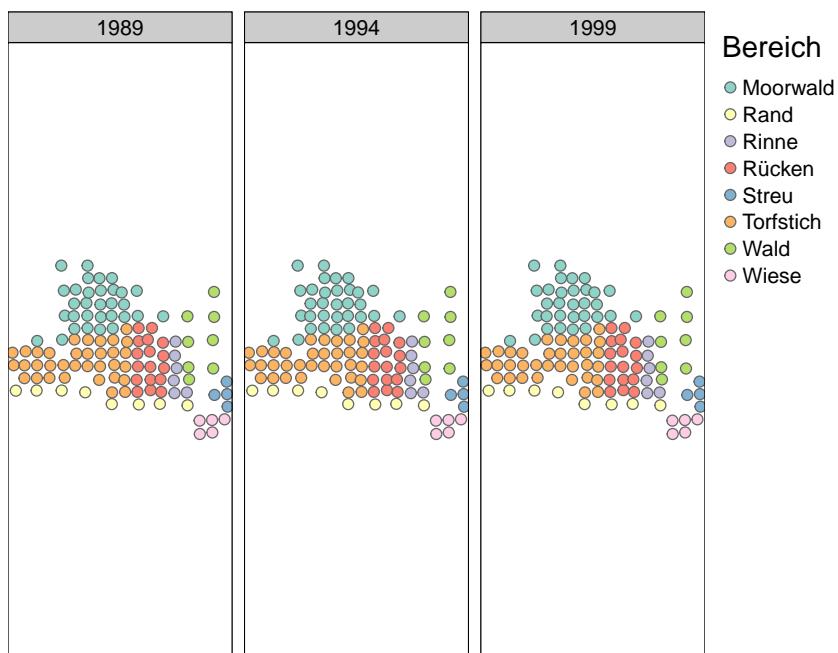
```
hangenmoos <- readxl::read_excel("data/original/daten_vegedaz/Hagenmoos.xlsx", "KopfdatenVe  
  
hangenmoos_sf <- st_as_sf(hangenmoos, coords = c("X", "Y"))  
st_crs(hangenmoos_sf) <- 21781  
  
hangenmoos_sf_wgs84 <- st_transform(hangenmoos_sf, 4326)  
  
tm_shape(hangenmoos_sf_wgs84) +  
  tm_bubbles(col = "Bereich") +  
  tm_layout(legend.outside = TRUE)
```



Input Small Multiples

- 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)
```

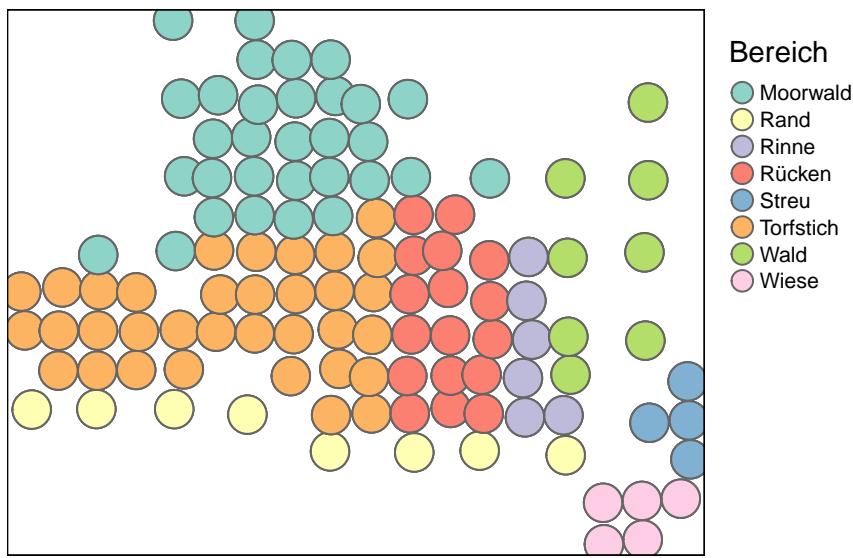


Übung 1.14

- Bisher haben wir nur s
- 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

i Musterlösung

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



Fazit

i 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 Punkt 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 sinnvollereres Format ist deshalb *geopackage*

Übung 2

Komplexe Vektordaten

Vorbereitung

Erstelle ein neues R Script mit dem Namen `Uebung_2.R` und lade darin die libraries `sf` und `tmap`

```
library("sf")
library("tmap")
```

Übung 2.1

- Suche dir die Gemeindegrenzen der Schweiz.
- Drei nützliche Adressen hierfür sind:
 - [opendata.swiss](#)
 - [map.geo.admin.ch](#)
 - [swisstopo.admin.ch](#)
- Wenn du die Wahl hast, versuche das File als Geopackage herunterzuladen. Ansonsten als Shapefile oder als File Geodatabase
- Entzippe das File (sofern nötig) und schau dir den Inhalt an

Musterlösung

https://data.geo.admin.ch/ch.swisstopo.swissboundaries3d/swissboundaries3d_2023-01/swissboundaries3d_2023-01_2056_5728.gpkg.zip

Shortlink (für diesen Kurs): <https://bit.ly/47wCOzo>

Übung 2.2

Importiere das den Datensatz mit `read_sf()` und speichere den output in der Variabel `gemeindegrenzen`

i Musterlösung

```
gemeindegrenzen <- read_sf("data/original/ch.swisstopo.swissboundaries3d-gemeinde-flaeche.gml")
```

Übung 2.3

Betrachte den importierten Datensatz in der Konsole. Was für Informationen kannst du entnehmen?

i Musterlösung

- Die Koordinaten sind im Bezugssystem CH1903+ LV95 abgespeichert
- Es handelt sich um 159 Gemeinden und 21 Attribute, die meisten unbrauchbar

Übung 2.4

Entferne alle Spalten bis auf `NAME`, `EINWOHNERZ` und `geometry`.

i Musterlösung

```
gemeindegrenzen <- gemeindegrenzen[, c("NAME", "EINWOHNERZ", "geometry")]
```

Übung 2.5

Visualisiere die Gemeindegrenzen mit `plot()` und `tmap`.

i Musterlösung

```
tm_shape(gemeindegrenzen) + tm_polygons()
```

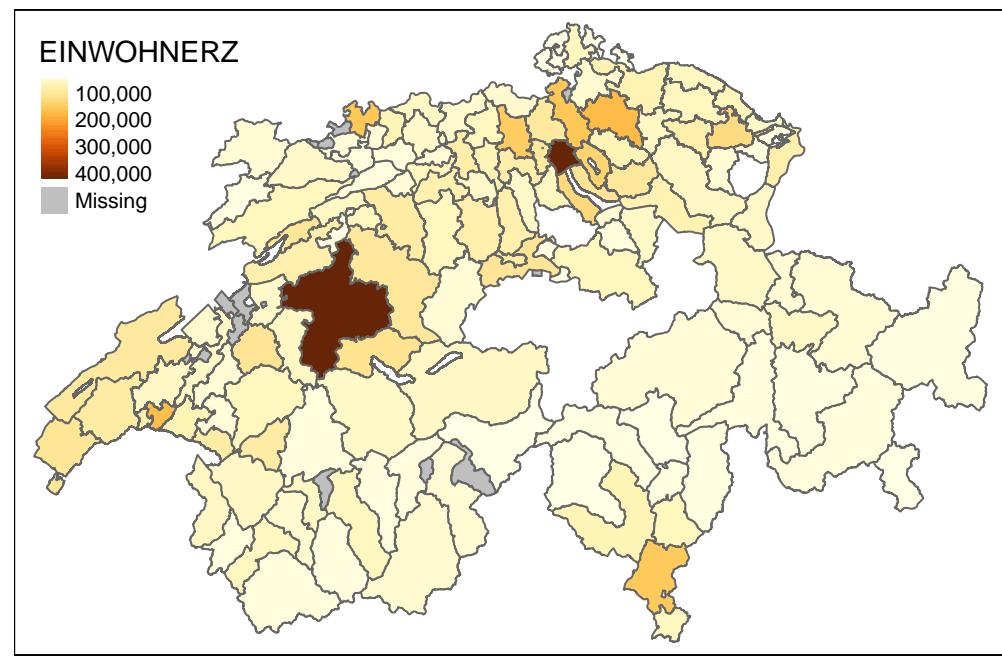


Übung 2.6

Färbe die Polygone nach der Einwohnerzahl ein. Spiele mit der Option `style` herum.

i Musterlösung

```
tm_shape(gemeindegrenzen) + tm_polygons(col = "EINWOHNERZ", style = "cont")
```



Übung 3

Einfache Rasterdaten

Vorbereitung

Installiere zudem das R Package **terra**

Erstelle dann ein neues R Script mit dem Namen **Uebung_3.R** und lade darin die libraries **sf** sowie **tmap**.

```
library("sf")
library("tmap")
```

Übung 3.1

- Such das digitale Höhenmodell der Schweiz (200m Auflösung)
- Auch hier kannst du die folgenden Adressen nutzen:
 - [opendata.swiss](#)
 - [map.geo.admin.ch](#)
 - [swisstopo.admin.ch](#)
- Entzippe das File (sofern nötig) und schau dir den Inhalt an

i Musterlösung

- <https://www.swisstopo.admin.ch/de/geodata/height/dhm25200.html>
- Shortlink (für diesen Kurs): <https://bit.ly/3Hj4X0K>

Input: Raster Datenformate

Inhalt des heruntergeladenen zip-Files:

- Eigentliche Daten:
 - DHM200_polyface.dxf
 - DHM200.asc
 - DHM200.xyz
- Metadaten und Lizenzbedingungen:
 - license.txt
 - Metadata_gm03.xml
 - Metadata_PDF.pdf
 - Metadata_xml_iso19139.xml

Der gleiche Datensatz (DHM25 200) in 3 unterschiedlichen Datenformaten:

- DHM200.asc
- DHM200.xyz
- **DHM200_polyface.dxf** (← CAD bereich)

ESRI ArcInfo ASCII Grid

- Dateierweiterung *.asc
- ein Datenformat von ESRI (siehe die [Spezifikationen](#))
- beginnt mit mehreren Zeilen Metaadaten, darauf folgen die eigentlichen Werte
- kann in einem Texteditor geöffnet werden:

```
NCOLS 1926
NROWS 1201
XLLCORNER 479900.
YLLCORNER 61900.
CELLSIZE 200.
NODATA_VALUE -9999.
-9999. -9999. -9999. -9999. -9999. -9999. -9999. -9999. -9999. -9999.
...
...
...
835.415 863.55 887.424 869.213 855.539 845.878 829.714 815.258 807.458 799.816 799.2
```

ASCII Gridded XYZ

- Dateierweiterung *.xyz
- Ein offenes Format
- Beinhaltet 3 Spalten: x- und y- Koordinaten sowie Zellwert
- kann in einem Texteditor geöffnet werden:

```
655000.00 302000.00 835.01  
655200.00 302000.00 833.11  
655400.00 302000.00 831.20
```

Raster in R

Um Rasterdaten in R zu importieren verwenden wir das Package `terra`.

```
install.packages("terra")
```

```
library("terra")
```

```
terra 1.7.39
```

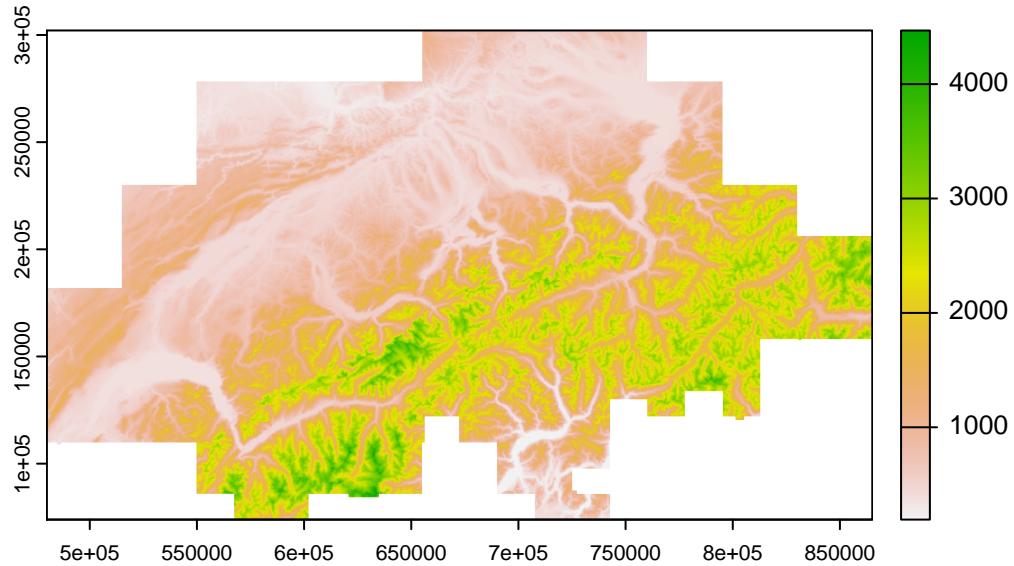
```
dhm200 <- rast("data/original/dhm25_200/DHM200.xyz")
```

- Aus `terra` benötigen wir die Funktion `rast`
- das Importieren funktioniert gleich, unabhängig von der Dateierweiterung
- eine summarische Zusammenfassung erhält man via Konsole:

```
dhm200
```

```
class      : SpatRaster  
dimensions : 1141, 1926, 1  (nrow, ncol, nlyr)  
resolution : 200, 200  (x, y)  
extent     : 479900, 865100, 73900, 302100  (xmin, xmax, ymin, ymax)  
coord. ref. :  
source     : DHM200.xyz  
name       : DHM200  
min value  : 193.00  
max value  : 4556.63
```

```
plot(dhm200) # für einfache visualisierungen
```



Übung 3.2

In welchem Koordinatensystem befindet sich dieses Höhenmodell?

Tipp: Konsultiere die Metadaten!

Musterlösung

Referenzsystem

Identifikator des Bezugssystems

Code

EPSG:21781

Referenzsystem

Identifikator des Bezugssystems

→ im alten Schweizer Koordinatensystem CH1903 LV03

Übung 3.2

Wie hoch ist die Auflösung?

Lösung

Räumliche Auflösung

Auflösung

Distanz | 200

→ 200 Meter

Übung 3.3

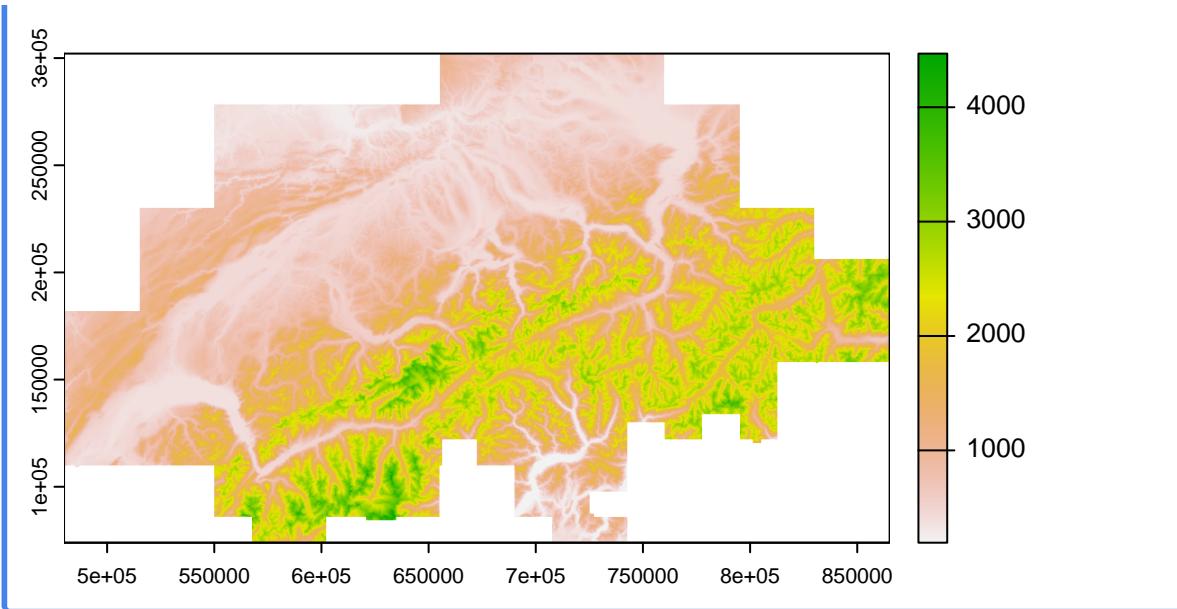
Importiere DHM200 in R und schau dir das Objekt in der Konsole sowie mit `plot()` an.

i Musterlösung

```
dhm200 <- rast("data/original/dhm25_200/DHM200.xyz")
dhm200

class      : SpatRaster
dimensions : 1141, 1926, 1 (nrow, ncol, nlyr)
resolution : 200, 200 (x, y)
extent     : 479900, 865100, 73900, 302100 (xmin, xmax, ymin, ymax)
coord. ref. :
source     : DHM200.xyz
name       : DHM200
min value  : 193.00
max value  : 4556.63

plot(dhm200)
```



Input: Koordinatenbezugssystem festlegen

- Das Koordinatenbezugssystem haben wir bereits für Vektordaten festgelegt
- dabei haben wir folgenden Befehl verwendet:
- `st_crs(meinvektordatensatz) <- 21781` (\leftarrow für das alte Schweizer Koordinatenbezugssystem)
- für Rasterdaten funktioniert es leicht anders:

```
crs(dhm200) <- "epsg: 21781"
```

- `crs()` statt `st_crs`
- `"epsg: 21781"` (mit Anführungs- und Schlusszeichen) statt `21781`

```
dhm200
```

```
class      : SpatRaster
dimensions : 1141, 1926, 1  (nrow, ncol, nlyr)
resolution : 200, 200  (x, y)
extent     : 479900, 865100, 73900, 302100  (xmin, xmax, ymin, ymax)
coord. ref. : CH1903 / LV03 (EPSG:21781)
source     : DHM200.xyz
name       : DHM200
min value  : 193.00
max value  : 4556.63
```

Input: Koordinatenbezugssystem transformieren

- Koordinatenbezugssystem von `dhm200`: CH1903 LV03 bzw. EPSG: 21781
- Analog Vektordaten: in das *neue* Schweizer Koordinatenbezugssystem transformieren
- Vektordaten: Funktion `st_transform`
- Rasterdaten: Funktion `project`

```
dhm200_2056 <- project(dhm200, "epsg: 2056")
```

```
dhm200
```

```
class      : SpatRaster
dimensions : 1141, 1926, 1 (nrow, ncol, nlyr)
resolution : 200, 200 (x, y)
extent     : 479900, 865100, 73900, 302100 (xmin, xmax, ymin, ymax)
coord. ref. : CH1903 / LV03 (EPSG:21781)
source     : DHM200.xyz
name       : DHM200
min value  : 193.00
max value  : 4556.63
```

```
dhm200_2056
```

```
class      : SpatRaster
dimensions : 1141, 1926, 1 (nrow, ncol, nlyr)
resolution : 200, 200 (x, y)
extent     : 2479900, 2865100, 1073900, 1302100 (xmin, xmax, ymin, ymax)
coord. ref. : CH1903+ / LV95 (EPSG:2056)
source(s)   : memory
name       : DHM200
min value  : 193.000
max value  : 4555.624
```

Übung 3.4

- Transformiere `dhm200` in das Koordinatenbezugssystem CH1903+ LV95
- Speichere den Output als `dhm200_2056`

i Musterlösung

```
dhm200_2056 <- project(dhm200, "epsg: 2056")
```

Übung 3.5

Visualisiere dhm200_2056 mit tmap.

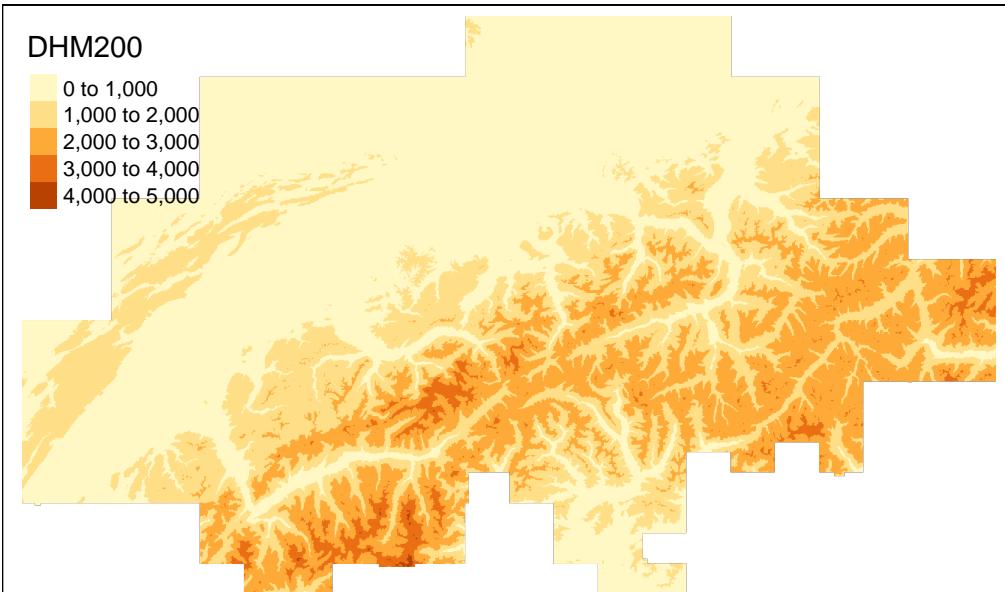
Tipp: Um ein Polygon zu visualisieren sind wir wie folgt vorgegangen

```
tm_shape(gemeindegrenzen) + tm_polygons()
```

i Musterlösung

```
# tmap_mode("view") # optional  
tm_shape(dhm200_2056) +  
  tm_raster()
```

stars object downsampled to 1299 by 770 cells. See tm_shape manual (argument raster.downsample)



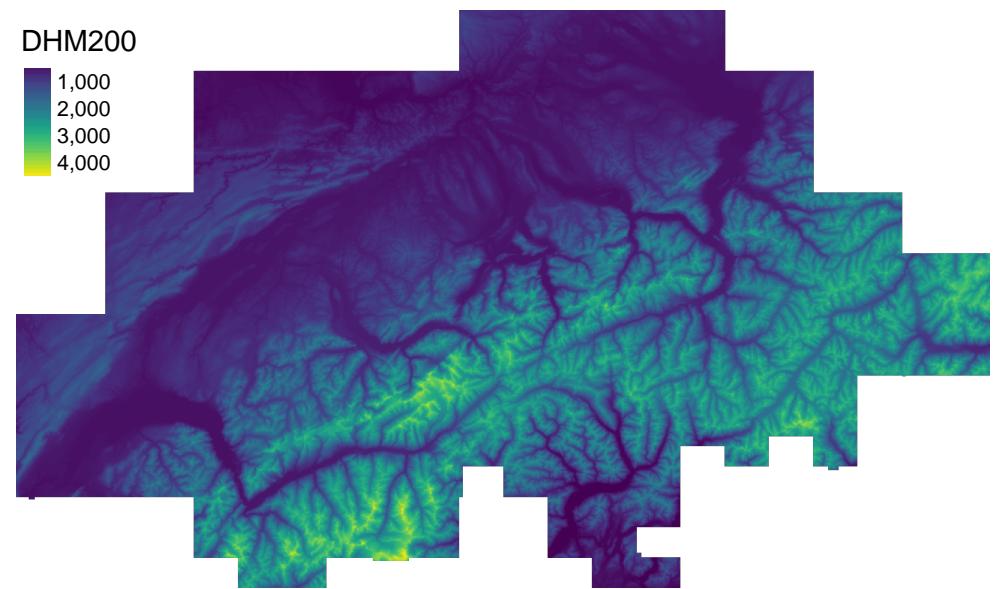
Übung 3.6

Verändere die Darstellungsweise des Rasters mithilfe von `style` und `palette`. Tipp, schau dir die Hilfe von `?tm_raster` an.

Musterlösung

```
tm_shape(dhm200_2056) + tm_raster(style = "cont", palette = "viridis")
```

stars object downsampled to 1299 by 770 cells. See `tm_shape` manual (argument `raster.downsample`)



Input Raster exportieren

- Wir haben das DHM auf unsere Bedürfnisse angepasst (CRS gesetzt und transformiert)
- Wir können unser verändertes Objekt (`dhm200_2056`) exportieren, so dass diese Änderungen abgespeichert werden

```
#| eval: false
#|
writeRaster(dhm200_2056, "data/processed/dhm200_2056.tif", overwrite = TRUE)
```

- beim Import ist die CRS Information bekannt (CRS setzen und transformieren ist nicht mehr nötig)

```
dhm200_2056 <- rast("data/processed/dhm200_2056.tif")
```

```
dhm200_2056
```

```
class      : SpatRaster
dimensions : 1141, 1926, 1 (nrow, ncol, nlyr)
resolution : 200, 200 (x, y)
extent     : 2479900, 2865100, 1073900, 1302100 (xmin, xmax, ymin, ymax)
coord. ref. : CH1903+ / LV95 (EPSG:2056)
source     : dhm200_2056.tif
name       : DHM200
min value  : 193.000
max value  : 4555.624
```

Übung 3.7

Exportiere `dhm200_2056` als `tif` File

i Musterlösung

```
writeRaster(dhm200_2056,"data/processed/dhm200_2056.tif")
```

i Rückblick

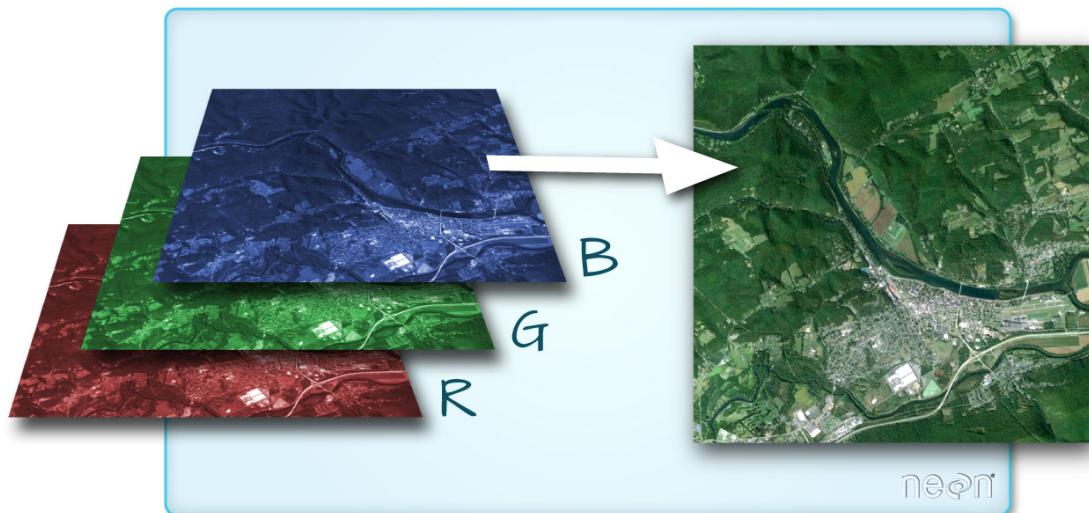
Wir haben...

- ein Höhenmodell der Schweiz heruntergeladen
- 3 unterschiedliche Datenformaten von Rasterdaten kennengelernt
- ein Rasterdatensatz mithilfe von `rast` aus `terra` in R importiert
- diesem Rasterdatensatz das korrekte Koordinatenbezugssystem zugewiesen (`crs`)
- diesen Rasterdatensatz in ein anderes Koordinatensystem transformiert (`project`)
- diesen Rasterdatensatz mit `plot()` sowie `tmap` visualisiert
- mit verschiedenen Darstellungformen in `tmap` gearbeitet (optionen `style` und `palette`)
- DHM: *ein* Wert pro Zelle. Es gibt aber Situationen, wo wir mehreren Werten pro Zelle benötigen

Übung 4

Komplexe Rasterdaten

- Satelliten und Drohnen nehmen meist verschiedene Spektren von Elektromagnetischen Wellen auf
- diese Spektren werden in unterschiedlichen Datensätzen abgespeichert
- diese Datensätze müssen wieder zusammengefügt werden um ein Gesamtbild zu erhalten
- Beispiel: Rot, Grün und Blau werte fügen sich zu einem Farbluftbild zusammen



Übung 4.1

- Ladet euch einen Ausschnitt aus dem Datensatz `swissimage 10` von Swisstopo herunter:
<https://www.swisstopo.admin.ch/de/geodata/images/ortho/swissimage10.html>

- Shortlink: <https://bit.ly/40Fy0Wj>
- Enzipped den Inhalt in euer RStudio Projekt und schaut den Inhalt an
- Was ist das Koordinatenbezugssystem? Wie hoch ist die räumliche Auflösung?

i Musterlösung

1. Koordinatenbezugssystem: EPSG 2056
2. räumliche Auflösung: 10cm oder 2m verfügbar

Übung 4.2

- Erstelle ein neues R Script mit dem Namen Uebung_4.R
- Lade die libraries `sf`, `tmap` und `terra`.
- Importiere den Swissimage Datensatz
- Weise dem importierten Datensatz das Korrekte Koordinatenbezugssystem zu
- Schau dir den Datensatz in der Konsole sowie mit `plot()` an

i Musterlösung

```
library("sf")
library("tmap")
library("terra")

swissimage <- rast("data/original/swissimage25/SWISSIMAGE25m/SI25-2012-2013-2014.tif")

crs(swissimage) <- "epsg: 2056"
```

```
plot(swissimage)
```



```
swissimage
```

```
class      : SpatRaster
dimensions : 9480, 14000, 3  (nrow, ncol, nlyr)
resolution : 25, 25  (x, y)
extent     : 2484375, 2834375, 1062000, 1299000  (xmin, xmax, ymin, ymax)
coord. ref. : CH1903+ / LV95 (EPSG:2056)
source     : SI25-2012-2013-2014.tif
colors RGB : 1, 2, 3
names      : SI25-2012-2013-2014_1, SI25-2012-2013-2014_2, SI25-2012-2013-2014_3
```

Input: RGB Plots mit tmap

- Um ein `rgb` Datensatz mit `tmap` zu plotten, verwenden wir nicht mehr `tm_raster()` sondern `tm_rgb`

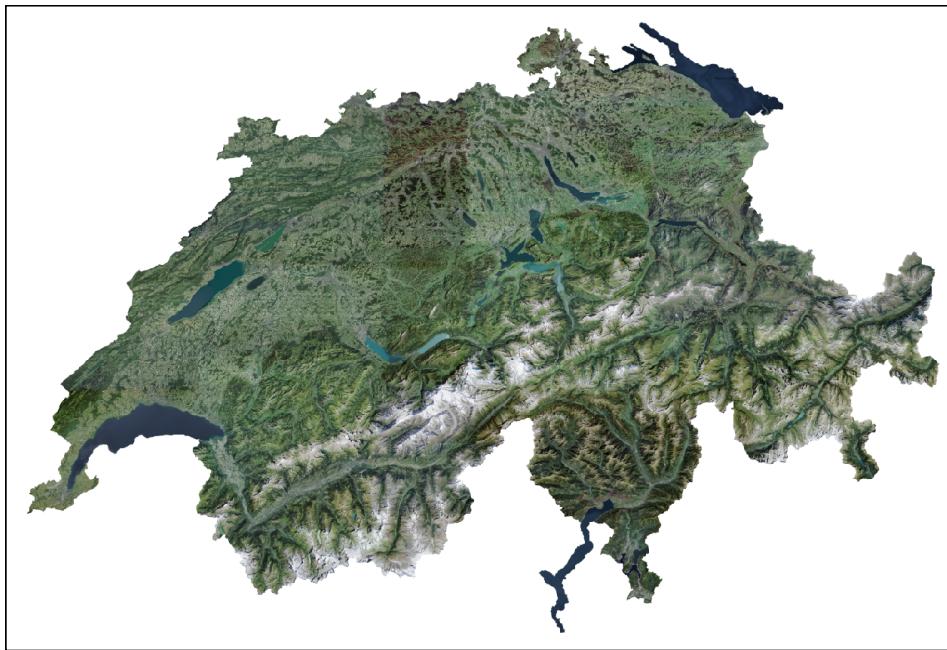
```
tmap_mode("plot")
```

```
tmap mode set to plotting
```

```
swissimage_10 <- aggregate(swissimage, fact = 10)
```

```
tm_shape(swissimage_10) +  
  tm_rgb()
```

stars object downsampled to 1215 by 823 cells. See `tm_shape` manual (argument `raster.downsample`)



Input

- Heute haben wir das Höhenmodell `dhm200` importiert
- Höhenmodell mit 200m Auflösung (→ grob!)
- swisstopo stellt zusätzlich das `dhm25` mit 25m Auflösung zur Verfügung (<https://bit.ly/3kFgZrF>)
- durch die höhere Auflösung dauert das transformieren in ein neues Koordinatensystem etwas länger

Übung 4.4

- Ladet euch das das `dhm25` mit 25m Auflösung herunter (<https://bit.ly/3kFgZrF>)

- importiert es in R
- setzt das korrekte CRS
- transformiert es in EPSG 2056 und verwendet dabei folgende Optionen:
 - mit `filename` = den Output direkt in ein File speichern
 - mit `progress` = TRUE den Fortschritt anzeigen lassen
- visualisiert es mit `tmap`

Musterlösung

```

dgm25 <- rast("data/original/DHM25_MM_ASCII_GRID/ASCII_GRID_1part/dhm25_grid_raster.asc")
crs(dhm25) <- "epsg: 21781"

dgm25

class      : SpatRaster
dimensions : 9121, 15401, 1 (nrow, ncol, nlyr)
resolution : 25, 25 (x, y)
extent     : 479987.5, 865012.5, 73987.5, 302012.5 (xmin, xmax, ymin, ymax)
coord. ref. : CH1903 / LV03 (EPSG:21781)
source     : dgm25_grid_raster.asc
name       : dgm25_grid_raster
min value  :                 193
max value  :                 4629

#terra::project(dgm25, "epsg: 2056", filename = "data/dgm25_2056.tif", progress = TRUE)

```

Übung 4.5 (Optional und Open End)

Suche dir auf den gängigen Portalen (s.u.) einen spannenden Datensatz und visualisiere diesen

- [opendata.swiss](#)
- [map.geo.admin.ch](#)
- [swisstopo.admin.ch](#)

Übung 4.6 (Optional und Open End)

- Lade dir swissimage daten in der Auflösung von 2m herunter und importiere sie in R

- Achtung! Sehr anspruchsvoll!!
- Tipps: du brauchst dazu:

- `list.files()`
- `lapply`
- `do.call`
- `mosaic`

Musterlösung

```
# Daten herunterladen:

links <- read.csv("data/original/swissimage_2m_landquart/swissimage_2m_urls.csv", header = FALSE)

links <- links$V1

filenames <- basename(links)

for (i in seq_along(links)) {
  download.file(links[i], file.path("data/original/swissimage_2m_landquart/", filenames[i]))
}

# Daten importieren:

swissimage_paths <- list.files("data/original/swissimage_2m_landquart", full.names = TRUE)

swissimage_list <- lapply(swissimage_paths, function(x){rast(x)})

swiss_mosaic <- do.call(terra::mosaic, swissimage_list)

writeRaster(swiss_mosaic, "data/processed/swissimage_2m_landquart.tif", overwrite = TRUE)
```

Übung 4.7

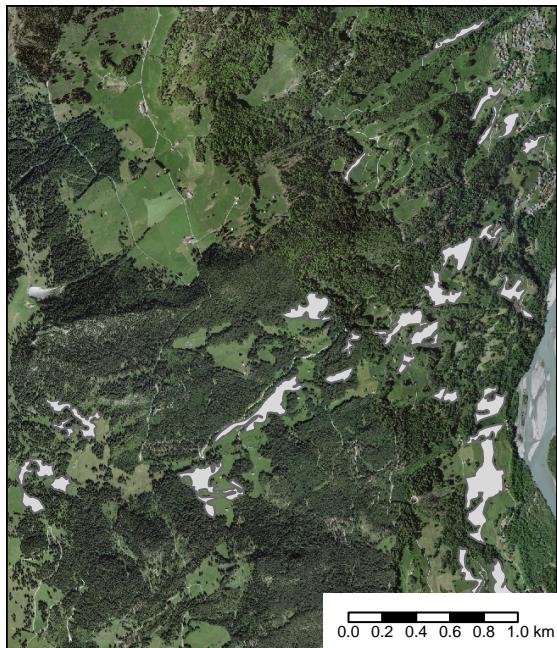
```
tww_landquart <- read_sf("data/processed/tww_landquart.gpkg")

swissimage <- terra::rast("data/processed/swissimage_2m_landquart.tif")

tm_shape(swissimage) +
  tm_rgb() +
```

```
tm_shape(tww_landquart, is.master = TRUE) +  
  tm_polygons() +  
  tm_scale_bar(position = c(1,0), just = c(1,0), bg.color = "white")
```

stars object downsampled to 1000 by 1000 cells. See `tm_shape` manual (argument `raster.downsample`)



Übung 5

Integration von Geodaten

- Bisher haben wir alle Datensätze einzeln betrachtet
- Wenn wir alle Datensätze ins gleiche Bezugssystem bringen, können wir diese *integrieren* bzw. überlagern
- Überlagern kann heissen:
 - gemeinsam Visualisieren
 - Information übertragen

Vorbereitung:

- Starte ein neues Script Uebung_5.R
- Importiere darin alle räumlichen libraries

```
library("tmap")
library("sf")
library("terra")

# tmap_options(check.and.fix = TRUE)
tmap_mode("plot")
```

Übung 5.1

Importiere die Datensätze ausserberg.gpkg (aus Übung 2) sowie dhm200_2056.tif (aus Übung 3, data/processed/ausserberg.gpkg bzw. data/processed/dhm200_2056.tif)

 Musterlösung

```
ausserberg <- read_sf("data/processed/ausserberg.gpkg")
dhm200 <- rast("data/processed/dhm200_2056.tif")
```

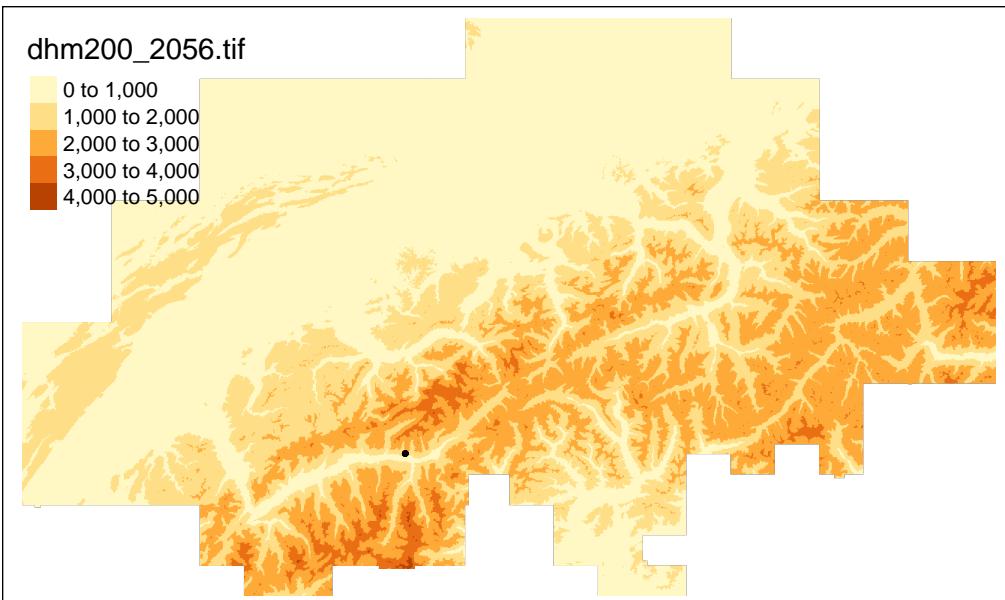
Übung 5.2

Überlagere die beiden Datensätze in einem tmap-Plot, indem du diese mit `+` verkettetest.

Musterlösung

```
tm_shape(dhm200) +  
  tm_raster() +  
  tm_shape(ausserberg) +  
  tm_dots()
```

stars object downsampled to 1299 by 770 cells. See `tm_shape` manual (argument `raster.downsample`)



- da das `dhm200` die ganze Schweiz abdeckt, sind unsere Punkte kaum erkennbar.
- Lösung: raster mittels unseren Punktdaten „zuschneiden“ (`crop`)

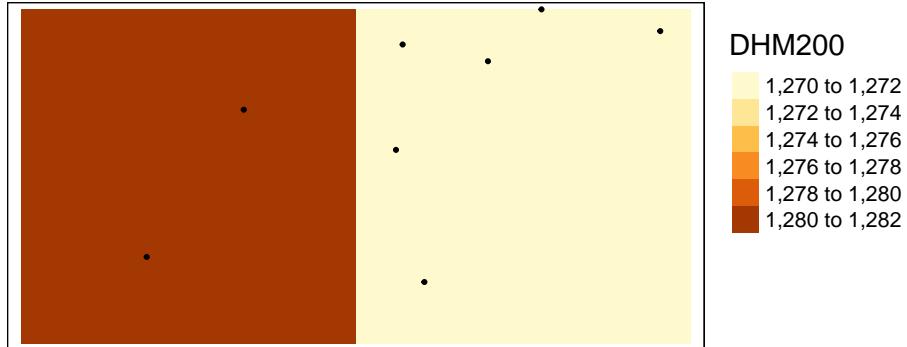
Übung 5.3

- Mit `crop()` können wir ein Raster auf den “extent” von einem Vektor Datensatz zuschneiden
- Schneide `dhm200` auf den extent von `ausserberg` zu
- Visualisiere das resultierende Raster mit `tmap` (wieder gemeinsam mit `ausserberg`)

Musterlösung

```
dhm200_cropped <- terra::crop(dhm200, ausserberg)

tm_shape(dhm200_cropped) +
  tm_raster() +
  tm_shape(ausserberg) +
  tm_dots() +
  tm_layout(legend.outside = TRUE)
```



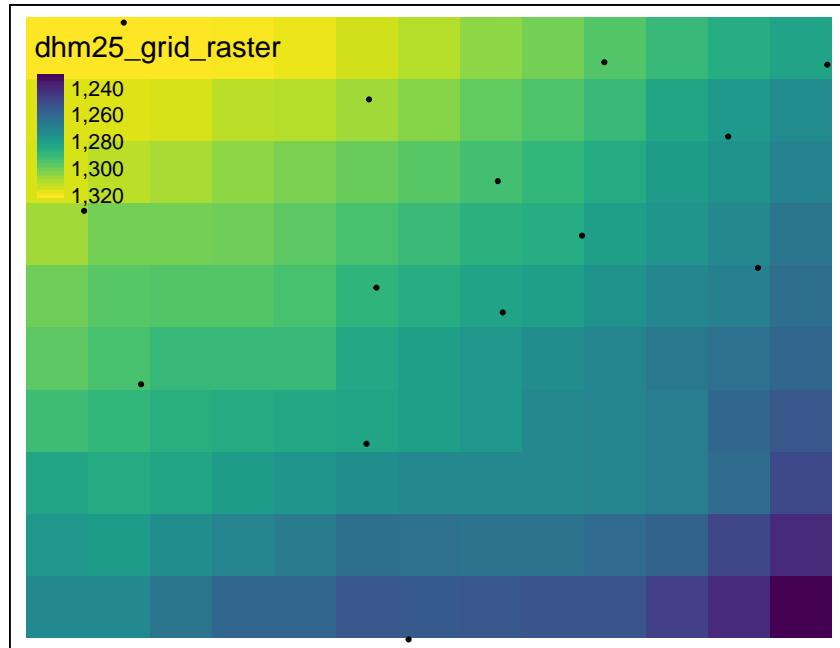
Übung 5.4

- die Auflösung des Raster Datensatzes ist zu grob!!
- Lösung: Hoch aufgelöster Datensatz `dhm25` (aus Übung 4) und einlesen (zip-File: `processed/dhm25_2056.tif`)
- wiederhole das Zuschneiden mittels `crop` sowie das Visualisieren mittels `tmap`

Musterlösung

```
dhm25 <- rast("data/processed/dhm25_2056.tif")
dhm25_crop <- crop(dhm25, ausserberg)
```

```
tm_shape(dhm25_crop) +
  tm_raster(style = "cont", palette = "viridis") +
  tm_shape(ausserberg) +
  tm_dots()
```



Input: Rasterwerte extrahieren

- bisher haben wir zwei Datensätze (Raster und Vektor) visuell überlagert
- nächster Schritt: **Information** von Raster → Punkt Datensatz übertragen
- dazu müssen wir `ausserberg` von einem `sp-` in ein `SpatVector` Objekt konvertieren
- danach können wir das `SpatVector` Objekt gemeinsam mit `extract` verwenden

Übung 5.5

- Wandle `ausserberg` mit der Funktion `vect()` in ein `SpatVector` Objekt und speichere es als `ausserberg_vect`
- Schau dir `ausserberg_vect` an, was hat sich verändert?
- Verwende die Funktion `extract` mit `ausserberg_vect` um die Höhenwerte aus `dhm25` zu extrahieren
- Speichere den output in einer Variabel und beguteachte diese

i Musterlösung

```
# ausserberg_vect <- vect(ausserberg)
elev <- extract(dhm25, ausserberg) # <- die Funktion extract() extrahiert die Information

elev                                     # <- der output ist eine data.frame mit 2 Spalten

  ID dhm25_grid_raster
1   1      1307.219
2   2      1269.035
3   3      1284.346
4   4      1292.309
5   5      1282.125
6   6      1281.335
7   7      1320.213
8   8      1292.845
9   9      1267.855
10 10     1307.206
11 11     1277.758
12 12     1281.398
13 13     1287.539
14 14     1244.080
15 15     1294.878
```

Übung 5.6

Spiele die Höheninformation aus `extract` zurück in `ausserberg`.

i Musterlösung

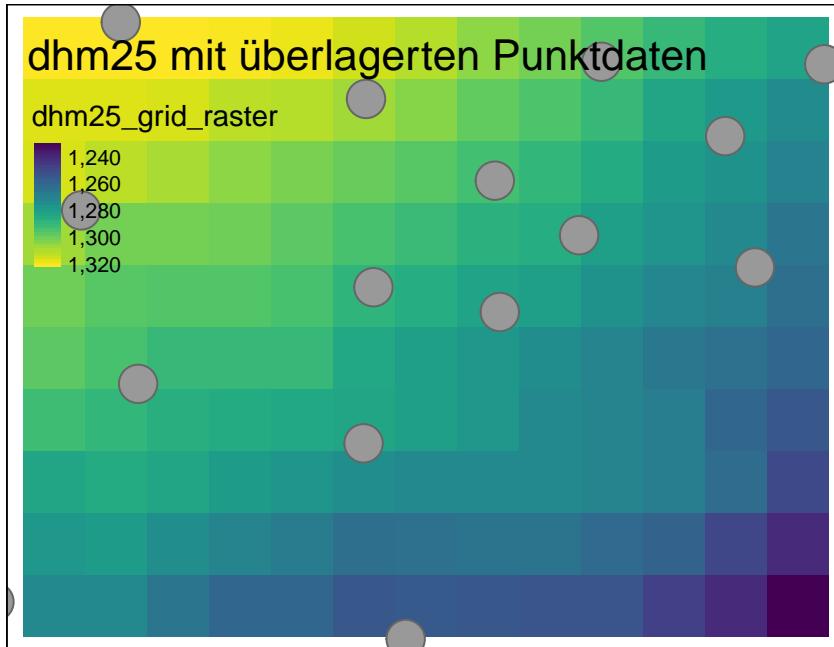
```
ausserberg$elevation <- elev[,2]          # <- die 2. Spalte aus elev auf ausserberg überträgt
```

Übung 5.7

Visualisiere nun `ausserberg` und Färbe die Punkte nach ihrer Höheninformation ein.

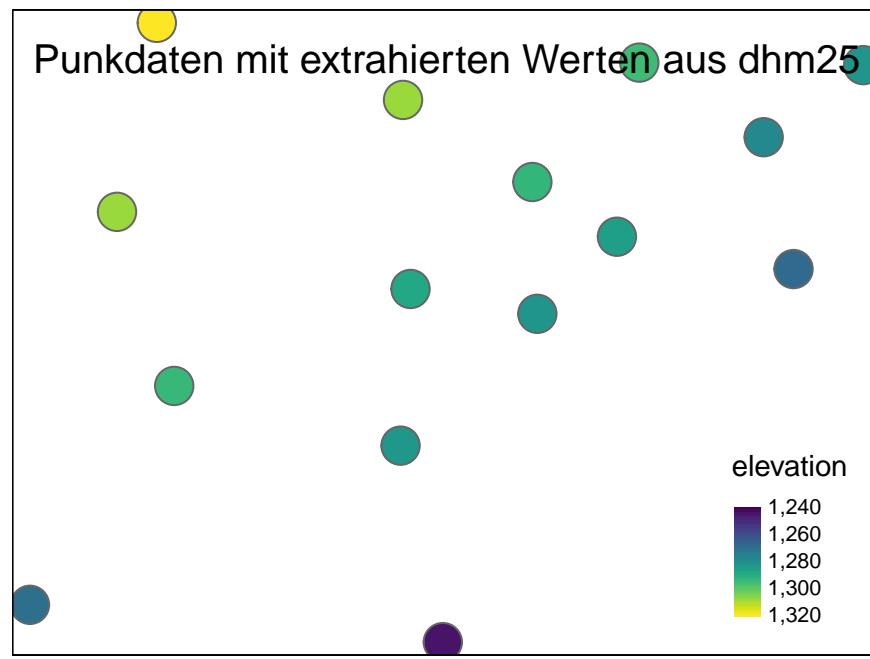
Musterlösung

```
tm_shape(dhm25_crop) +  
  tm_raster(style = "cont", palette = "viridis") +  
  tm_shape(ausserberg) +  
  tm_bubbles() +  
  tm_layout(title = "dhm25 mit überlagerten Punktdaten")
```



```
tm_shape(ausserberg) +  
  tm_bubbles(col="elevation", style="cont", palette="viridis", breaks = seq(1240, 1320, 20))  
  tm_layout(title = "Punktdaten mit extrahierten Werten aus dhm25")
```

Warning: Values have found that are higher than the highest break



Input: Vektordaten zuschneiden

- nun wollen wir zwei Vektordatensätze miteinander verschneiden
- Ausgangslage:
 - wir verfügen über einen [TWW Datensatz der Schweiz](https://bit.ly/3CqNRKT) (<https://bit.ly/3CqNRKT>)
 - wir verfügen über den [Gemeindelayer der Schweiz](https://bit.ly/3CaAj5W) (<https://bit.ly/3CaAj5W>)
 - wir wollen alle TWW Flächen innerhalb der Gemeinde Landquart erhalten

Übung 5.8

- Lade diese beiden Datensätze herunter und importiere sie in R (swissboundaries *Hoheitsgebiet*)
- Transformiere sie in EPSG 2056

i Musterlösung

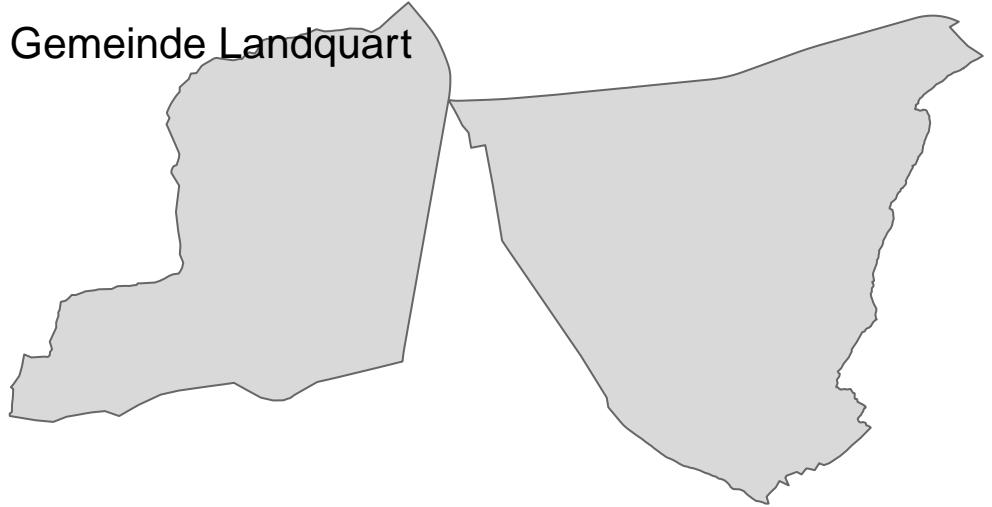
```
# Datensätze einlesen:  
tww <- read_sf("data/original/TWW/TWW_LV95/trockenwiesenweiden.shp")  
hoheitsgebiet <- read_sf("data/original/ch.swisstopo.swissboundaries3d-gemeinde-flaeche.fil  
  
# Gemeindegrenzen in EPSG 2056 transformieren und nur Landquart selektieren  
hoheitsgebiet <- st_transform(hoheitsgebiet, 2056)
```

Übung 5.9

Erstelle ein neues Objekt `landquart`, welches nur die Gemeinde Landquart beinhaltet und visualisiere diese.

i Musterlösung

```
landquart <- hoheitsgebiet[hoheitsgebiet$NAME == "Landquart", ]  
  
tm_shape(landquart) +  
  tm_polygons() +  
  tm_layout(title = "Gemeinde Landquart")
```



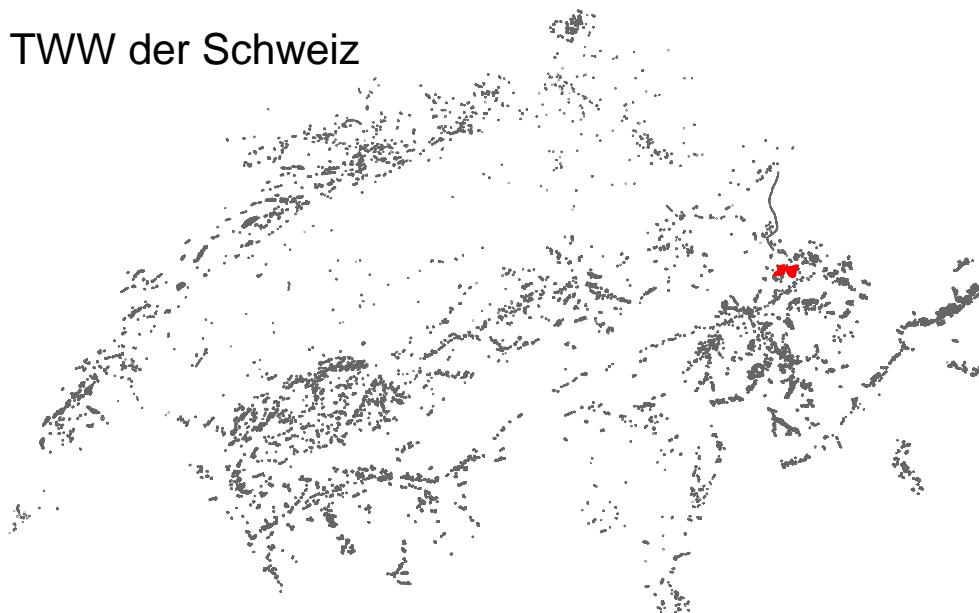
Übung 5.10

Überlagere die TWW Flächen mit der Gemeindegrenze von Landquart.

i Musterlösung

```
tm_shape(tww) +  
  tm_polygons() +  
  tm_layout(title = "TWW der Schweiz") +  
  tm_shape(landquart) +  
  tm_polygons(col = "red", border.col = "red")
```

TWW der Schweiz



Übung 5.11

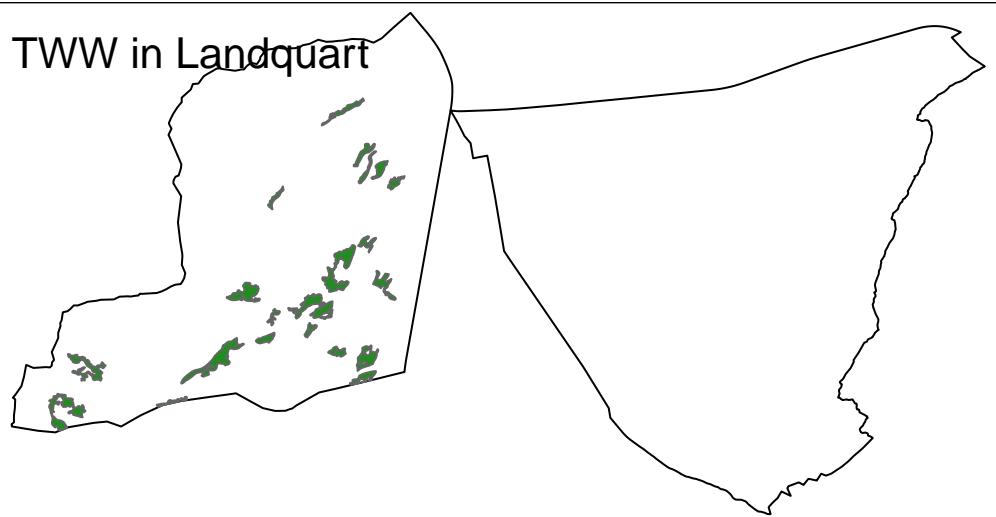
- Verwende die Funktion `st_intersection()` um die TWW-Flächen auf die Gemeindegrenze von Landquart zu zuschneiden.
- Visualisiere das Resultat

Musterlösung

```
tww_landquart <- st_intersection(tww, landquart)
```

Warning: attribute variables are assumed to be spatially constant throughout all geometries

```
tm_shape(landquart) +  
  tm_borders(col = "black") +  
  tm_shape(tww_landquart) +  
  tm_polygons(col = "forestgreen") +  
  tm_layout(title = "TWW in Landquart")
```



Input: Vektordaten selektieren

Mit `st_intersection` haben wir TWW Flächen verschnitten, da `st_intersection` die Schnittmenge beider Polygone nimmt



- Alternativ können wir alle TWW Flächen selektieren, die mindestens Teilweise innerhalb des Gemeindegebietes liegen

```
tww_landquart2 <- tww[landquart, ]
```

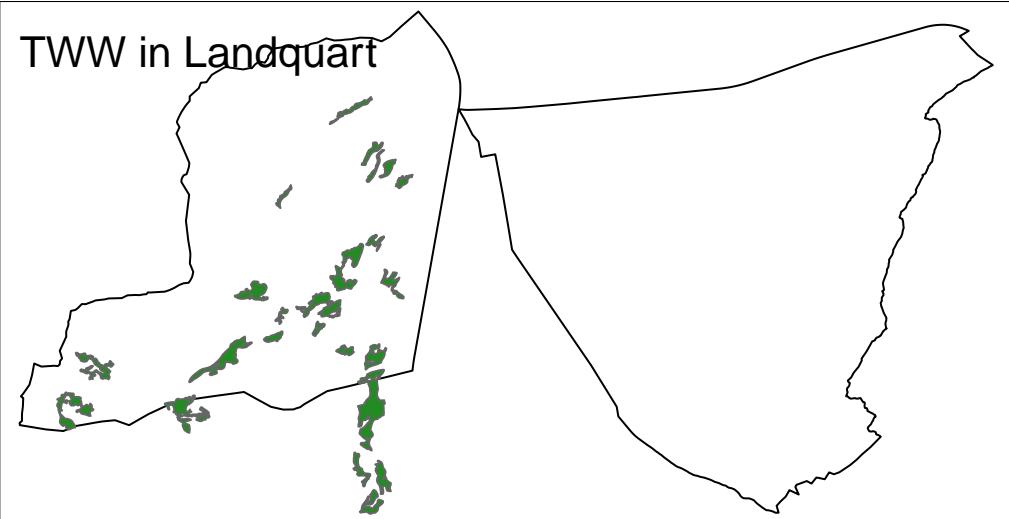
Übung 5.12

- Selektiere die TWW Flächen, welche sich zumindest Teilweise in der Gemeinde Landquart befinden und speichere den Output als `tww_landquart2`
- Visualisiere das Resultat mit `tmap`
- Vergleiche `tww_landquart2` mit `tww_landquart`. Wie unterscheiden sich diese?

i Musterlösung

```
tww_landquart2 <- tww[landquart, ]
```

```
tm_shape(landquart) +
  tm_borders(col = "black") +
  tm_shape(tww_landquart2) +
  tm_polygons(col = "forestgreen") +
  tm_layout(title = "TWW in Landquart")
```



Übung 5.13

i Musterlösung

Exportiere `tww_landquart2` als Geopackage

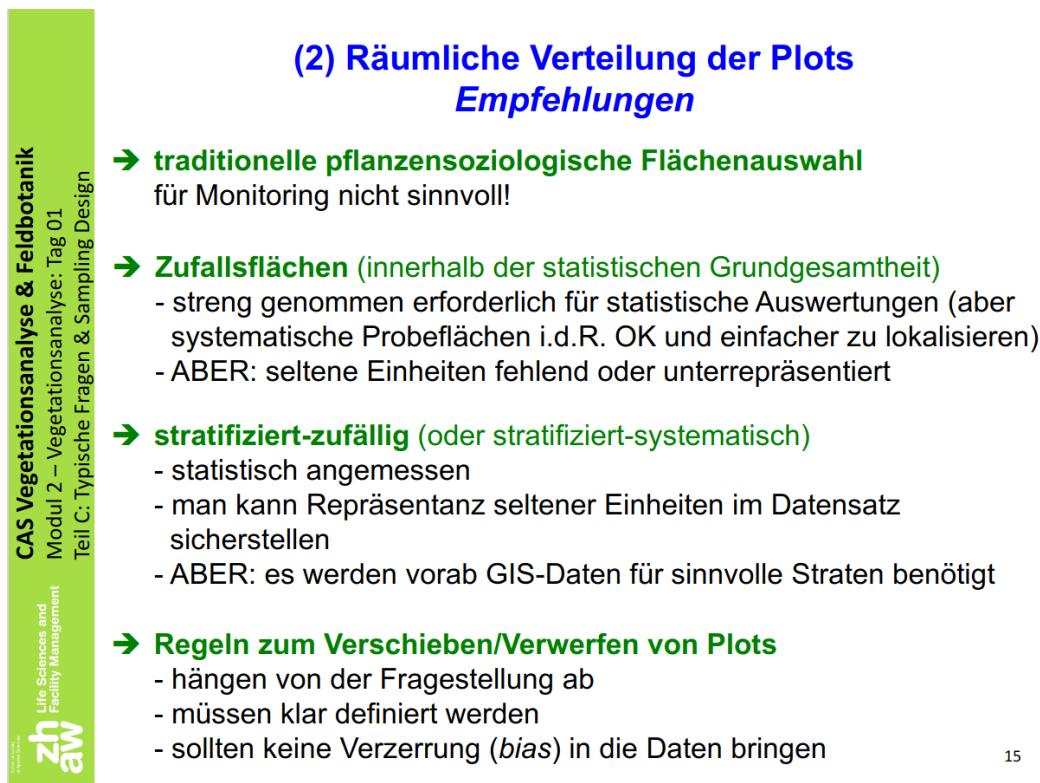
```
st_write(tww_landquart2, "data/processed/tww_landquart.gpkg", delete_layer = TRUE)
```

Übung 6

Sampling Design

R eignet sich hervorragend um ein Sampling Design umzusetzen. Am ersten Tag des CAS habt ihr etwas zu *Sampling Design* gelernt (siehe Abbildung 2).

Folie Sampling Design



The slide has a green header bar with white text. On the left side of the slide, there is a vertical sidebar with the following text:
CAS Vegetationsanalyse & Feldbotanik
Modul 2 – Vegetationsanalyse: Tag 01
Teil C: Typische Fragen & Sampling Design
zhaw Life Sciences and Facility Management

**(2) Räumliche Verteilung der Plots
Empfehlungen**

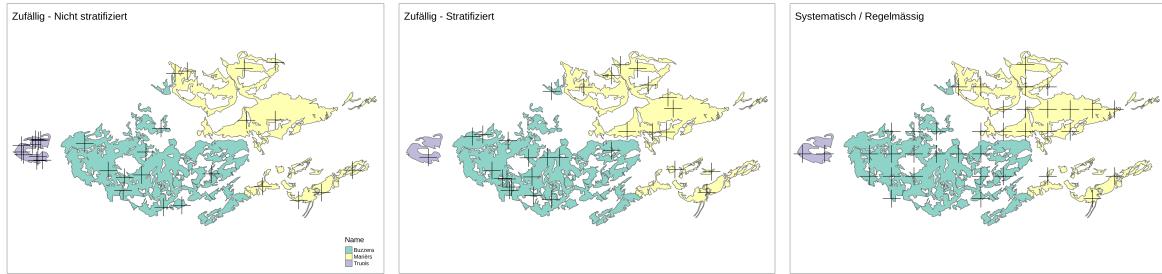
- **traditionelle pflanzensoziologische Flächenauswahl**
für Monitoring nicht sinnvoll!
- **Zufallsflächen** (innerhalb der statistischen Grundgesamtheit)
 - streng genommen erforderlich für statistische Auswertungen (aber systematische Probeflächen i.d.R. OK und einfacher zu lokalisieren)
 - ABER: seltene Einheiten fehlend oder unterrepräsentiert
- **stratifiziert-zufällig** (oder stratifiziert-systematisch)
 - statistisch angemessen
 - man kann Repräsentanz seltener Einheiten im Datensatz sicherstellen
 - ABER: es werden vorab GIS-Daten für sinnvolle Straten benötigt
- **Regeln zum Verschieben/Verwerfen von Plots**
 - hängen von der Fragestellung ab
 - müssen klar definiert werden
 - sollten keine Verzerrung (*bias*) in die Daten bringen

15

Abbildung 1

Schauen wir uns folgende *Sampling* Strategien an:

- zufällig
 - nicht stratifiziert (siehe Abbildung 2a und Kapitel)
 - stratifiziert (siehe Abbildung 2b und Kapitel)
- systematisch / regelmässig (siehe Abbildung 2c und Kapitel)



(a) 30 Samples total, 10 Samples pro Lokalität (b) 30 Samples total, verteilt nach Hektaren (c) 30 Samples total, systematisch / regelmässig verteilt

Abbildung 2: Drei verschiedene Sampling Strategien

Vorbereitung

```

library("sf")
library("terra")
library("tmap")
tmap_mode("plot")

set.seed(1920)

tww <- read_sf("data/original/TWW/TWW_LV95/trockenwiesenweiden.shp")

filter <- c("Mariërs", "Buzzera", "Truois") ①
tww <- tww[tww$Name %in% filter,]

tww <- tww[, "Name"] ②

tww$area_ha <- as.numeric(st_area(tww))/10000 ③

samples_total <- 30 ④
  
```

```
base_plot <- tm_shape(tww) +
  tm_polygons(col = "Name") +
  tm_layout(legend.show = FALSE, asp = 7/5) ⑤
```

- ① Nur 3 Lokaliäten auswählen
- ② Nur die Spalte “Name” (=Lokalität) behalten. Die Geometrie Spalte kommt automatisch mit.
- ③ Fläche berechnen und in Hektaren umrechnen (als Vorbereitung für die das stratifizierte Sampling)
- ④ Variabel erstellen für die Summe an Samples, die wir machen können / wollen
- ⑤ Optional: Da wir immer wieder die gleiche Karte machen, können wir eine Basis Karte erstellen und immer wieder benutzen.

Zufällige Verteilung

Für die zufällige Verteilung der 30 Samples, gibt es zwei Möglichkeiten: (1) Nicht stratifiziert und (2) Stratifiziert. In jedem Fall brauchen wir eine Spalte mit der Anzahl der Samples für die jeweilige Lokalität:

```
tww$nicht_stratifiziert <- samples_total/nrow(tww) ①
```

```
tww$stratifiziert <- round(samples_total/sum(tww$area_ha)*tww$area_ha) ②
```

- ① Nicht stratifiziert: In jeder Lokalität gleich viele Samples ($\frac{30}{3} = 10$)
- ② Stratifiziert: Samples in Relation zur Fläche (A) verteilen ($\frac{30}{\sum A} \times A$)

```
knitr::kable(tww)
```

Name	geometry	area_ha	nicht_stratifiziert	stratifiziert
Mariërs	MULTIPOLYGON (((2822107 119...	240.5064	10	13
Buzzera	MULTIPOLYGON (((2820384 118...	305.1896	10	16
Truois	MULTIPOLYGON (((2815810 118...	20.1714	10	1

Zufällig - Nicht stratifiziert

```
sample_plots1 <- st_sample(tww, size = tww$nicht_stratifiziert)

nicht_stratifiziert_plot <- base_plot +
  tm_shape(sample_plots1) +
  tm_dots(shape = 3, size = 3) +
  tm_layout(title = "Zufällig - Nicht stratifiziert", legend.show = TRUE)

nicht_stratifiziert_plot
```

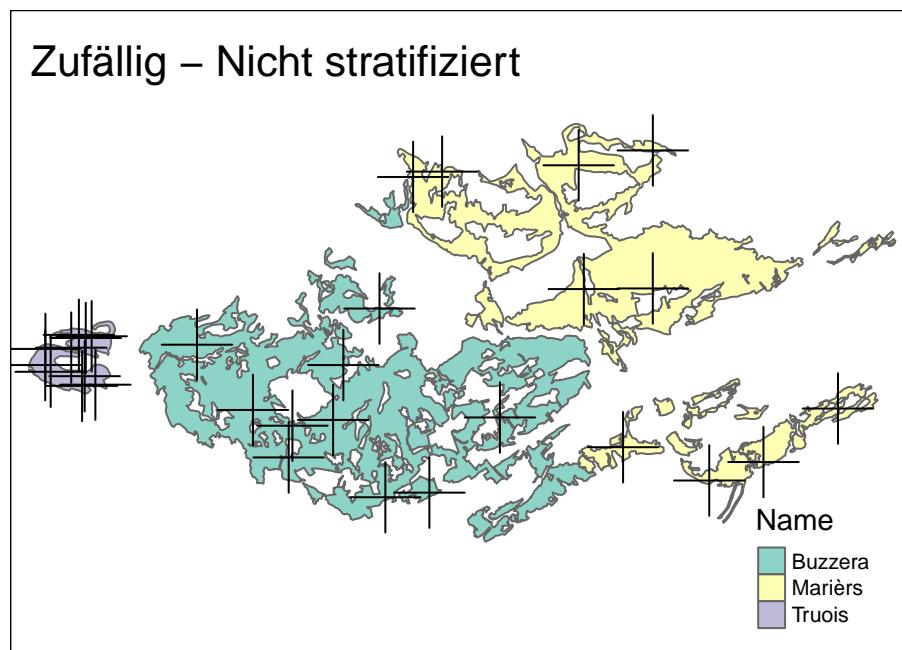


Abbildung 3

Zufällig - Stratifiziert

```
sample_plots2 <- st_sample(tww, size = tww$stratifiziert)

stratifiziert_plot <- base_plot +
  tm_shape(sample_plots2) +
  tm_dots(shape = 3, size = 3) +
```

```
tm_layout(title = "Zufällig - Stratifiziert")  
stratifiziert_plot
```

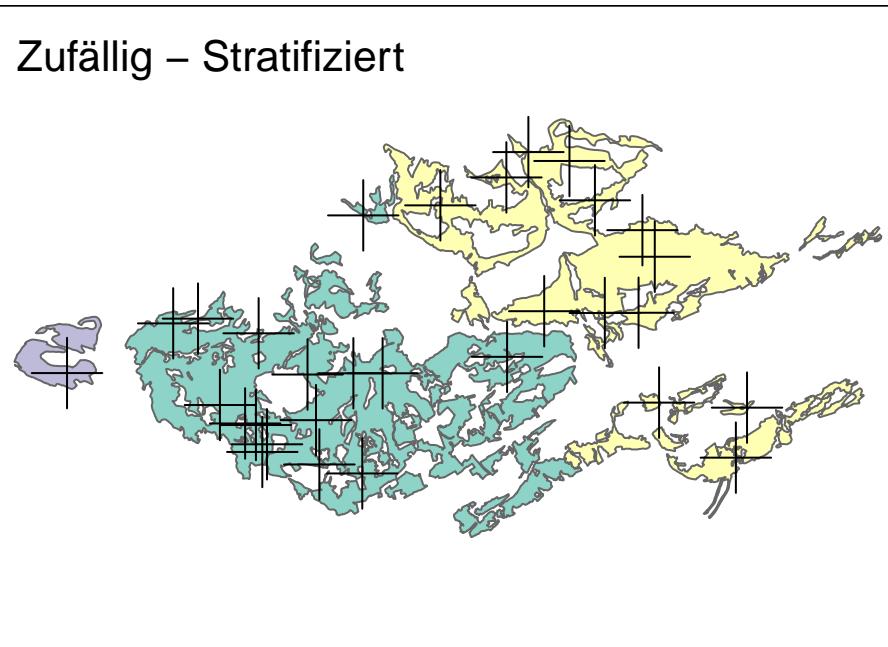


Abbildung 4

Systematisch / Regelmässig

```
sample_plots3 <- st_sample(tww, size = samples_total, type = "regular")  
  
systematisch_plot <- base_plot +  
  tm_shape(sample_plots3) +  
  tm_dots(shape = 3, size = 3) +  
  tm_layout(title = "Systematisch / Regelmässig")  
  
systematisch_plot
```

Systematisch / Regelmässig

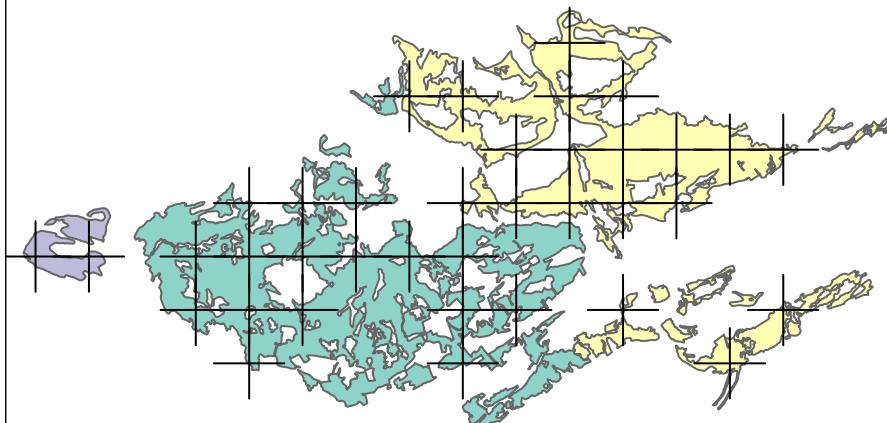


Abbildung 5

Übung

(Open End und ohne Musterlösung)

- Wähle einen kleineren Kanton oder eine Gemeinde aus
- Selektiere die TWW Standorte dieser Gemeinde / dieses Kantons
- Wähle ein sinnvolles Sampling Design und setze es mit R um
- Extrahiere die Höhenwerte für jeden Sample
- Visualisere in einer Karte:
 - die TWW Flächen
 - Gemeinde- / Kantongrenze
 - Sampling Standorte
 - Swissimage Hintergrund Karte
 - Nordpfeil, Scalebar