

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	4
Übung 1.3	4
Übung 1.4	5
Übung 1.5	5
Übung 1.6	5
Input CRS ansprechen	5
Übung 1.8	7
Übung 1.9	7
Input tmap	7
Übung 1.10	8
Input CRS wechseln	10
Übung 1.11	10
Übung 1.12 (Optional)	10
Input Räumliche Datenformate	10
Übung 1.13	11
Input Small Multiples	11
Übung 1.14	12
Fazit	12
Übung 2	14
Vorbereitung	14
Übung 2.1	14
Übung 2.2	14
Übung 2.3	15
Übung 2.4	15
Übung 2.5	15
Übung 2.6	15
Übung 3	16
Vorbereitung	16

Übung 3.1	16
Input: Raster Datenformate	16
Raster in R	18
Übung 3.2	19
Übung 3.2	19
Lösung	19
Übung 3.3	20
Input: Koordinatenbezugssystem <i>festlegen</i>	20
Input: Koordinatenbezugssystem <i>transformieren</i>	20
Übung 3.4	21
Übung 3.5	21
Übung 3.6	22
Input Raster exportieren	22
Übung 3.7	22
Übung 4	24
Übung 4.1	24
Übung 4.2	25
Input: RGB Plots mit <code>tmap</code>	26
Input	27
Übung 4.4	27
Übung 4.5 (Optional und Open End)	27
Übung 4.6 (Optional und Open End)	27
Übung 4.7	28
Übung 5	29
Übung 5.1	29
Übung 5.2	29
Übung 5.3	30
Übung 5.4	30
Input: Rasterwerte extrahieren	30
Übung 5.5	30
Übung 5.6	30
Input: Vektordaten zuschneiden	31
Übung 5.8	31
Übung 5.9	31
Übung 5.10	31
Übung 5.11	31
Input: Vektordaten selektieren	32
Übung 5.12	32
Übung 5.13	32

Übung 6	33
Vorbereitung	34
Zufällige Verteilung	35
Zufällig - Nicht stratifiziert	36
Zufällig - Stratifiziert	36
Systematisch / Regelmässig	37
Übung	38

Einführung

Willkommen bei den Kursunterlagen zum Kurstag 17 *Geographische Kartendarstellungen mit R*, im Rahmen des Modul 2 im CAS Vegetationsanalyse & Feldbotanik. Diese Kursunterlagen sollen Vegetationsökologen und Feldbotaniker in die Lage versetzen, ihre Daten mit Hilfe von R zu analysieren und zu visualisieren. Der Kurs ist als Workshop aufgebaut: Die Inputs sind relativ kurz, dafür programmieren wir *viel* und *gemeinsam*. Dabei werden am Kurstag die Übungen 1 bis Übungen 6 so weit wie möglich gemeinsam durchgearbeitet: Was wir nicht gemeinsam schaffen kann selbstständig als Hausaufgabe gelöst werden.

Jede Übung wird auf dem Beamer projiziert und sofern nötig mündlich ergänzt. Danach haben die Kursteilnehmer Zeit, die Übung selbstständig zu lösen. Nach einer Weile wird eine Lösung präsentiert (live coding) und die Teilnehmer können Fragen stellen. Für Personen, denen das Tempo zu langsam ist, können die Übungen auch selbstständig durcharbeiten.

Diese Unterlagen stehen sowohl als Website cas-vegetationsanalyse-feldbotanik.github.io sowie auch als PDF zur Verfügung. Die PDF-Version kann auf der genannten Website heruntergeladen werden (siehe PDF-Symbol oberhalb des Menübalkens links).

Nach der Durchführung des Kurses werden Musterlösungen

Tabelle 1: Ungefähre Zeitplan für den Kurstag

von	bis	Thema
09:00	09:35	Begrüßung und Einführung räumliche Daten
09:50	10:35	Vektordaten: Übung 1
10:50	12:00	Vektordaten: Übung 1 & Übung 2
12:00	13:00	Mittagspause
13:00	13:35	Rasterdaten: Input und Übung 3 beginnen
13:50	14:35	Rasterdaten: Übung 3 & Übung 4
14:50	15:35	Integration von Geodaten: Übung 5
15:50	17:00	Sampling Design: Übung 6

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`

Übung 1.2

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

Übung 1.3

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

Übung 1.4

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

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 1.6

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

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.

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`

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")
```

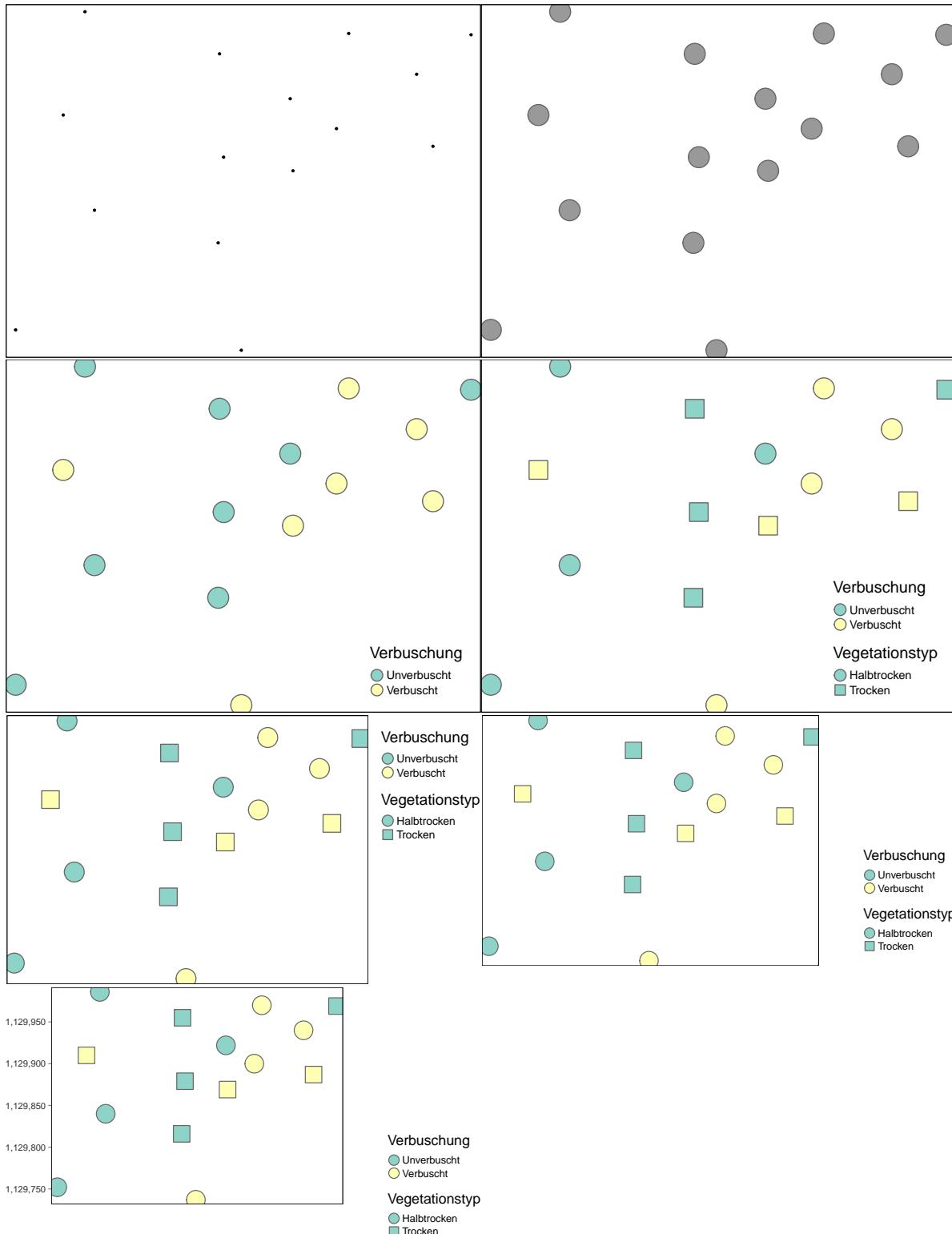
```
library("tmap")
```

- `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).



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()`

Übung 1.12 (Optional)

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

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")
```

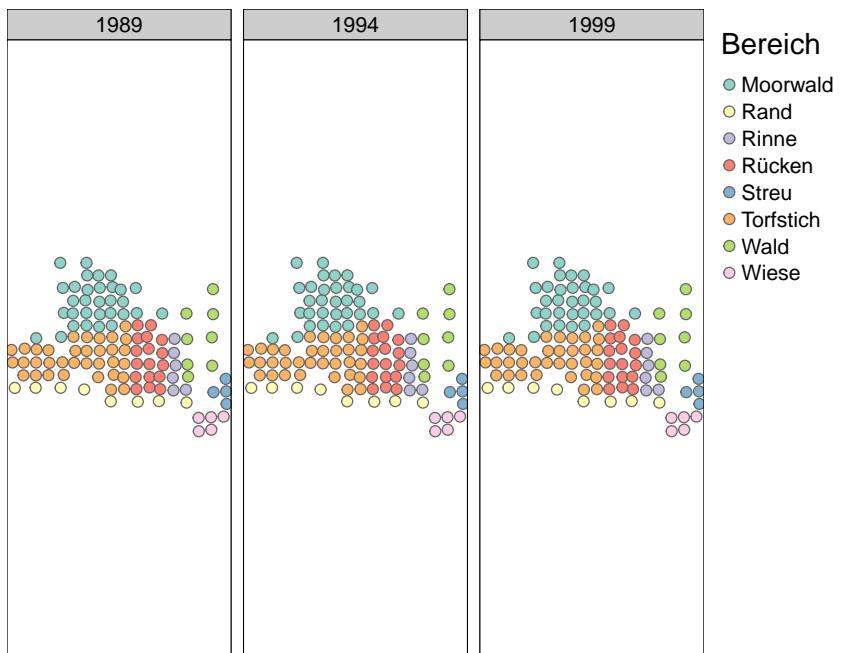
Übung 1.13

- 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`

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

Fazit

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

Übung 2.2

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

Übung 2.3

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

Übung 2.4

Entferne alle Spalten bis auf NAME, EINWOHNERZ und geometry.

Übung 2.5

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

Übung 2.6

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

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

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. -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
```

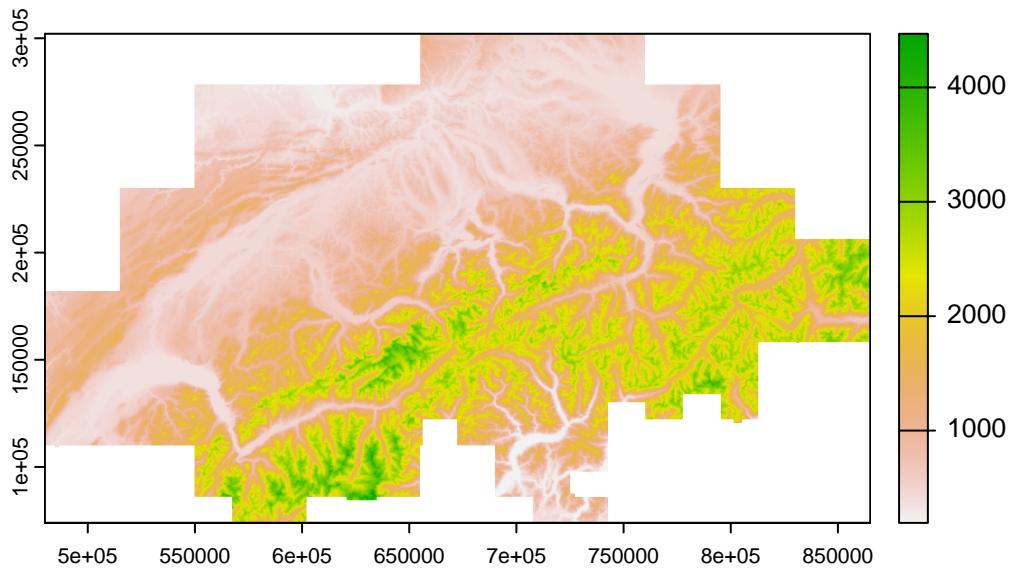
```
dhm200 <- rast("data/original/dhm25_200/DHM200.xyz")
```

- 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!

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.

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`

Übung 3.5

Visualisiere `dhm200_2056` mit `tmap`.

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

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

Übung 3.6

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

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

Rückblick

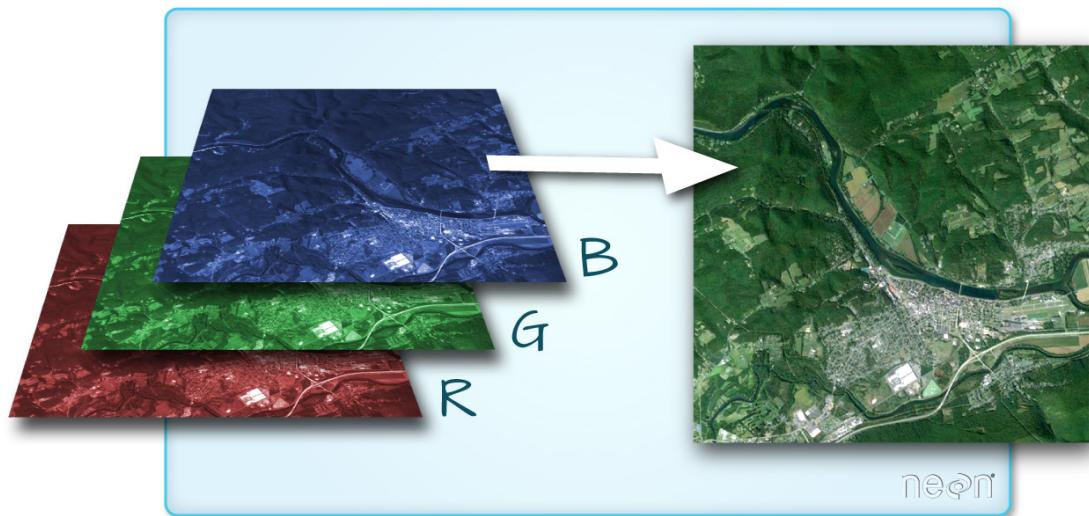
Wir haben...

- 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?

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

```
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()`

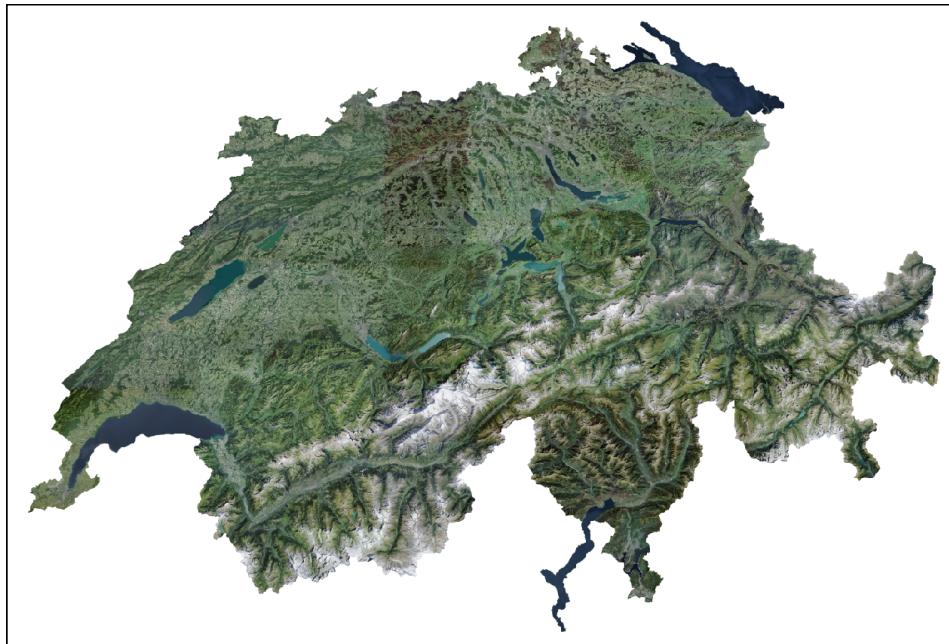
```
tmap_mode("plot")
```

```
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.downsamp
```



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`

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](#)

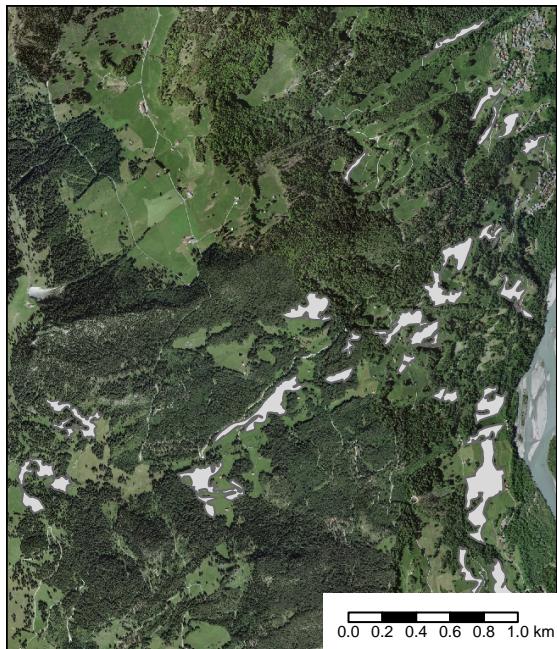
Übung 4.6 (Optional und Open End)

- 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`

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)

Übung 5.2

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

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`)

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`

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

Übung 5.6

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

Übung 5.7

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

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

Übung 5.9

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

Übung 5.10

Überlagere die TWW Flächen mit der Gemeindegrenze von Landquart.

Übung 5.11

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

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?

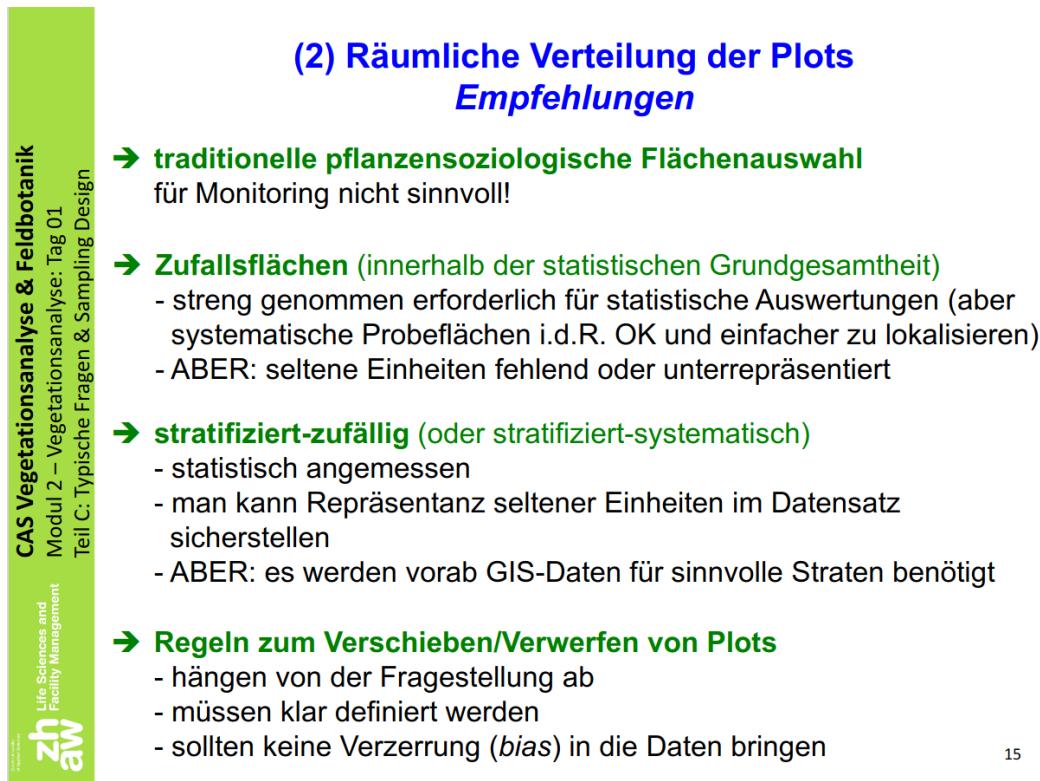
Übung 5.13

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**

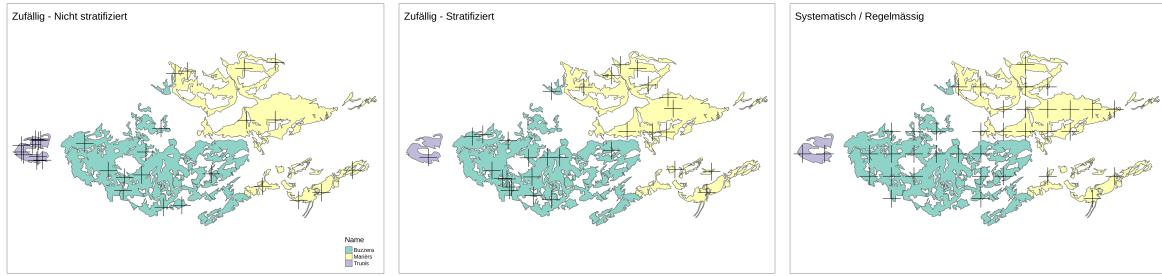
- **traditionelle pflanzensoziologische Flächenauswahl**
für Monitoring nicht sinnvoll!
- **Zufallsflächen** (innerhalb der statistischen Grundgesamtheit)
 - streng genommen erforderlich für statistische Auswertungen (aber systematische Probeflächen i.d.R. OK und einfacher zu lokalisieren)
 - ABER: seltene 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
```

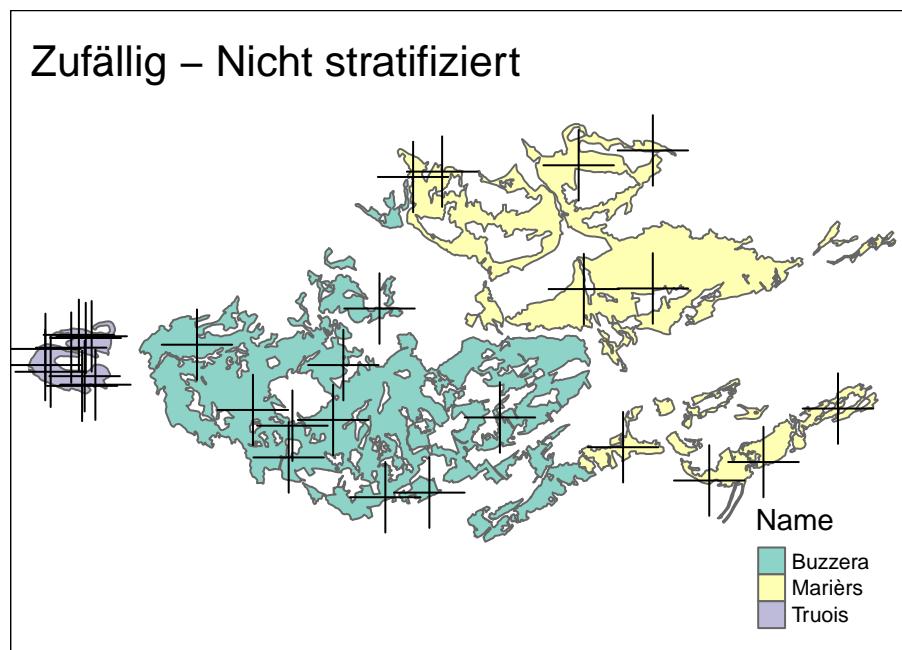


Abbildung 3

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
```

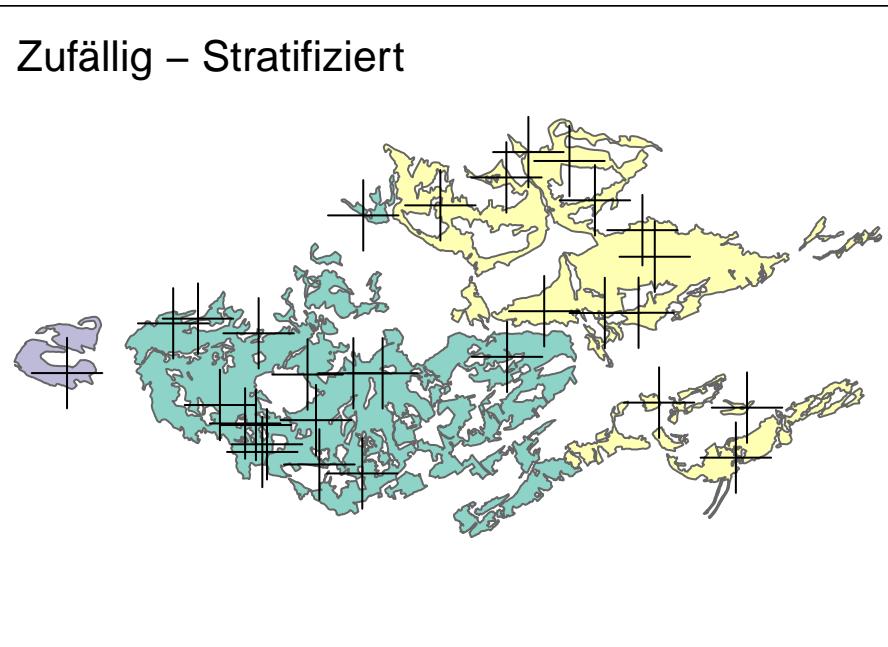


Abbildung 4

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