Modul AGI HS23

Forschungsgruppe Geoinformatik

Inhaltsverzeichnis

Willkommen!

Willkommen im Kurs *Angewandte Geoinformatik* an der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW). Wir nutzen diese Plattform (modul-agi.github.io) um euch Zugriff auf unsere Übungsunterlagen und Theorieinputs zu gewährleisten.

Der gesamte Quellcode um dieses Buch zu erstellen ist in dem folgenden github-repo verfügbar: github.com/Modul-AGI.

Teil I.

Unsicherheit

Übungsziele

- Du bist in der Lage bei zukünftigen GIS Arbeiten mit der Unsicherheit von Punktdaten umzugehen.
- Du kannst reale räumliche Objekte dahin beurteilen, ob diese grenzscharf oder nur mit einer Unschärfe abgebildet werden können.
- Du kannst mit Fuzzy-Tallness umgehen und diese definieren.
- Du kannst unscharfe Phänomene in deinen zukünftigen Projekten berücksichtigen und deren Unschärfe in den Geoverarbeitungsprozess integrieren und auswerten.
- Du kannst ein Modell mit ModelBuilder erstellen und ausführen.

1. Datengrundlage

Auf Moodle steht ein ArcGIS Pro Project Package (Name: Unsicherheit) mit den heutigen Übungsdaten zum Download zur Verfügung.

Als Packages aufbereitete ArcGIS Pro Projekte können als File anderen Personen zur Verfügung gestellt werden. Damit können auf elegante Art und Weise die eigenen Projektdaten geteilt werden.

> ℹ ArcGIS Pro Help: Create a project package
>
> https://pro.arcgis.com/en/pro-app/help/sharing/overview/project-package.htm

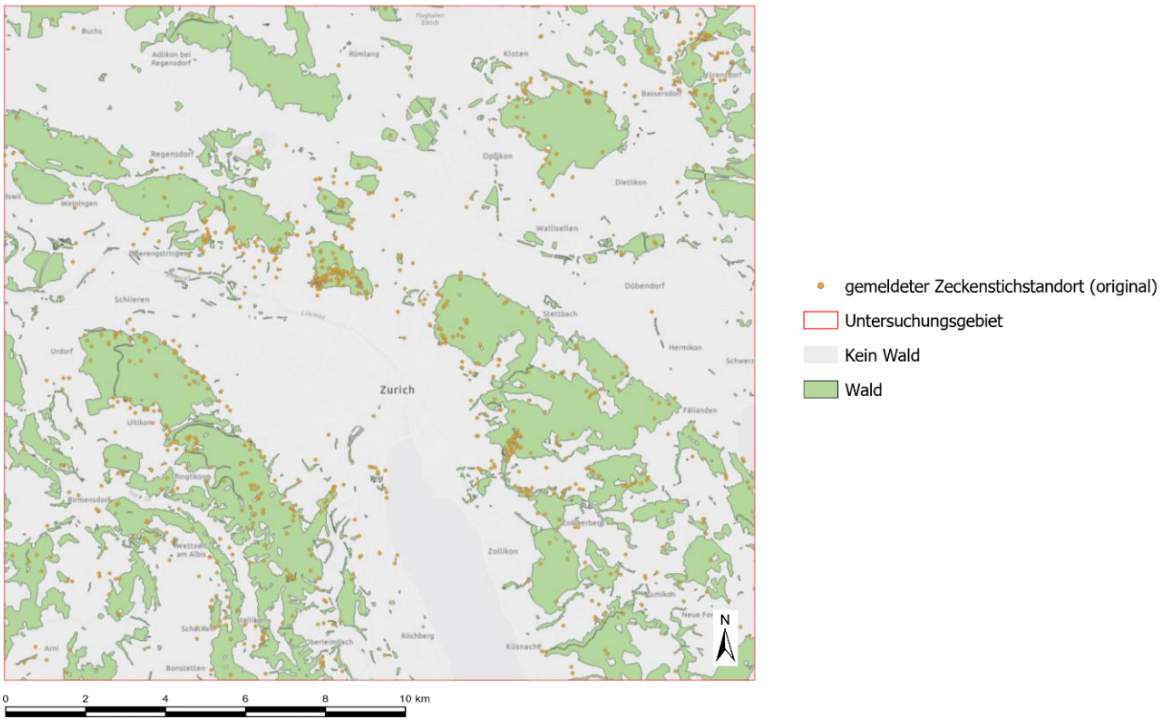
Das ArcGIS Pro Projekt "Unsicherheit" beinhaltet eine Geodatenbank mit folgenden Feature Classes:

- Feature Dataset Grundlagendaten
  - Untersuchungsgebiet (Polygon)
  - Wald_NichtWald (Polygon)
- Feature Dataset Zeckenstichdaten
  - Tick_Original (Point)
  - Tick_Run_1 bis Tick_Run_40 (Point)

1.1. Layer

Die Feature Class Tick_Original beinhaltet die Originalstandorte der via Zecken App gemeldeten Zeckenstiche im Raum Zürich aus dem Zeitraum März 2015 bis Juli 2019. Es handelt sich dabei um 1076 Meldungen. Die anderen 40 Feature Classes mit den Bezeichnungen Tick_Run_1 (bis …40) enthalten die mittels Zufall veränderten Positionen der Originalstandorte (vgl. Abschnitt Datenvorverarbeitung).

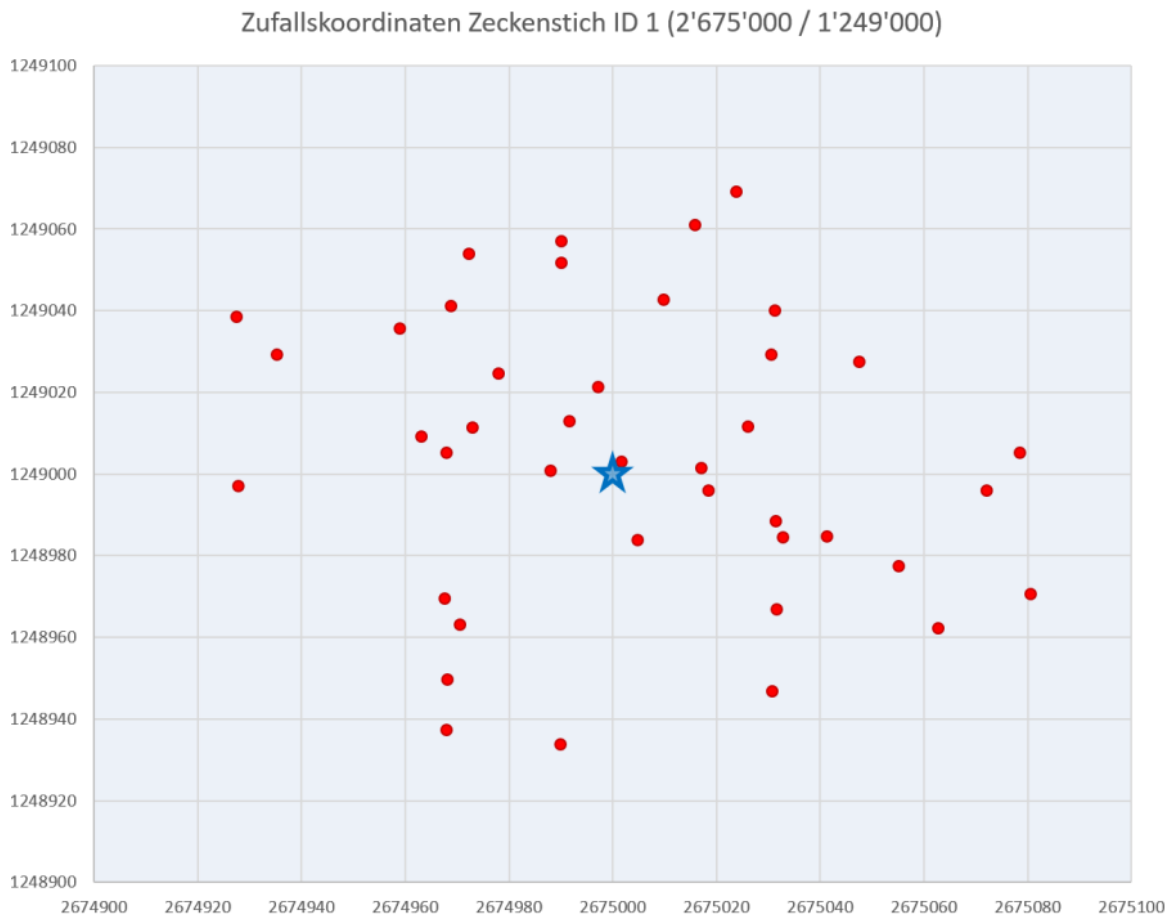
Der Layer Wald_NichtWald beinhaltet die Waldausdehnung innerhalb des Untersuchungsraums. Die Daten stammen aus dem Landschaftsmodell swissTLM3D von swisstopo (Layer Bodenbedeckung).



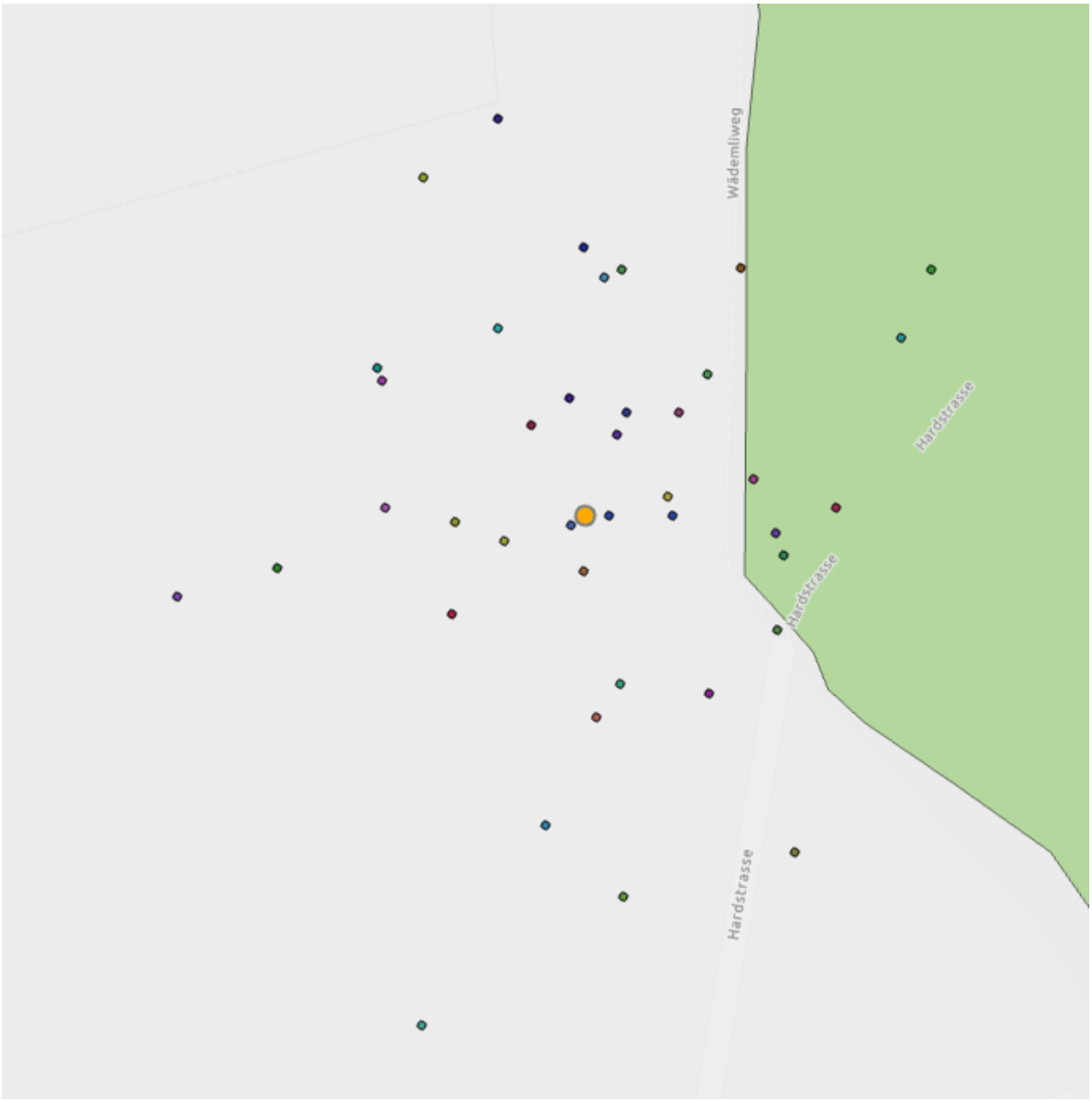
- FC Wald_Nichtwald
  - Wald: 0 = Kein Wald / 1 = Wald
- FC Tick_Original
  - ID: ID Zeckenstichmeldung
  - acc_: Zoomstufe in Karte während der Erfassung des Stiches in der App
  - x_lv95: X Koordinate (CH1903+ LV95)
  - y_lv95: Y Koordinate (CH1903+ LV95)
  - Run_Nr: 0 (Original)
- FC Tick_Run_1 bis Tick_Run_40
  - ID: ID Zeckenstichmeldung (gleich wie in Original)
  - acc_: Zoomstufe in Karte während der Erfassung des Stiches in der App (gleich wie in Original)
  - x_lv95: X Koordinate (Originalstandort)
  - y_lv95: Y Koordinate (Originalstandort)
  - Run_Nr_x: X Zufallskoordinate Run Nr
  - Run_Nr_y: Y Zufallskoordinate Run Nr
  - Run_Nr: Nummer Zufallsdurchgang

2. Datenvorverarbeitung

Da der/die App Nutzer(in) den exakten Standort des Zeckenstichs nur mit eingeschränkter Positionsgenauigkeit erfassen kann (abhängig von Zoomstufe in der Karte und individueller räumlicher Orientierungskompetenz), wird davon ausgegangen, dass die erfassten Standorte über eine Unsicherheit verfügen. Um die Auswirkung dieser Unsicherheit auf den Anteil im Wald liegender Stiche simulieren zu können, wurden die 1076 Originalkoordinaten in 40 Monte Carlo Durchgängen (Runs) zufällig in einem Bereich +/-100 Meter versetzt, bildlich gesprochen wurden die Punkte etwas "verschüttelt". Diese Zufallskoordinaten wurden in Excel generiert. Die untenstehende Abbildung zeigt beispielhaft für Zeckenstichstandort mit ID 1 (blauer Stern = Originalstandort) die 40 zufälligen Standorte (rote Punkte).



Folgende Abbildung zeigt exemplarisch wie diese künstlich simulierte Ungenauigkeit eines Punktes Auswirkungen darauf hat, ob ein Punkt sich innerhalb oder ausserhalb des Waldes befindet (orange: Originalstandort). Bei 6 von 40 Durchgängen befindet sich der Standort (ID 15611) innerhalb des Waldes. Die Frage, welche beantwortet werden soll, lautet nun: Wie gross ist die Wahrscheinlichkeit, dass sich der mit einer bekannten Ungenauigkeit erfasste Zeckenstichstandort tatsächlich innerhalb des Waldes befindet? (vgl. Kap. 3 Unsicherheit der gemeldeten Zeckenstiche)



Bezüglich Unsicherheit von Geodaten stellt sich zudem die Frage, wie genau kann die Waldrandgrenze mittels einer Polygongrenze überhaupt abgebildet werden? Handelt es sich um ein Objekt, welches eindeutig klar definiert werden kann (grenzscharf) oder verfügt die Waldrandgrenze selber über eine Unschärfe? (vgl. Kap. 4 Unschärfe des Waldrandes)

2.1. Übung 1: Daten erkunden und neues “Resultate” Feature Dataset erstellen

Downloade das ArcGIS Pro Package von Moodle, speichere es lokal auf deinem Computer und entpacke die Datei. Öffne anschliessend das Projekt-File (*.aprx), erkunde die Daten und mache dich mit ihnen vertraut.

Erstelle anschliessend in der Projekt-Geodatenbank Unsicherheit.gdb ein neues Feature Dataset mit dem Namen “Resultate”. Weise dem neuen Feature Dataset das Koordinatensystem CH1903+ LV95 zu.

In diesem neuen Feature Dataset sollen nachfolgend alle neu erstellten (Zwischen-)Resultate abgelegt werden.

3. Unsicherheit der gemeldeten Zeckenstichstandorte

Die erste Aufgabe besteht darin, zu untersuchen, wie gross der Einfluss der Erfassungsunsicherheit darauf ist, welcher Anteil der erfassten Zeckenstichstandorte inner- oder ausserhalb des Waldes liegen. Dabei soll der Fokus nur auf der Unsicherheit (der Erfassungsgenauigkeit) der Zeckenstichstandorte liegen.

3.1. Übung 2: Zusammenfügen (Append) aller Runs zu einem Datensatz

Kopiere zuerst die Feature Class *Tick_Original* in das Feature Dataset *Resultate*. Benenne die kopierte Feature Class in *Tick_Append* um, ändere auch den Alias. Nutze anschliessend das Geoverarbeitungswerkzeug "Append", um die soeben kopierten Original-Standorte mit den Zeckenstichstandorten aller Runs zusammenzufügen. Wähle unter Field Matchig Type die Option "Use the field map to reconcile field differences".

> ArcGIS Pro Help: Geoprocessing Tool Append
>
> https://pro.arcgis.com/en/pro-app/latest/tool-reference/data-management/append.htm

Überprüfe das Resultat. Die *Tick_Append* Feature Class sollte anschliessend 44'116 Features beinhalten.

3.2. Übung 3: Summe der Stiche innerhalb der Waldfläche

Um zu überprüfen wie viele Stiche sich innerhalb des Waldes befinden, muss zunächst der Punktlayer mit den Stichdaten aller Runs und den Originaldaten (Resultat aus Übung 2) mit dem Wald-Layer überlagert werden. Hierbei bietet sich das Geoverarbeitungswerkzeug "Summarize Within" (GeoAnalytics Desktop Tools) an. Dieses Werkzeug zählt alle Punkte welche sich in den jeweiligen Flächen des Input-Polygons befinden. Ausserdem kann damit eine separate Output Tabelle (Output Grouped Table) generiert werden, welche für jeden Run

angibt (Group Field = Run_Nr), wie viele Punkt pro Run inner- oder ausserhalb des Waldes liegen. Hinweis: Speichere die Output Grouped Table direkt auf Datenbank-Ebene und nicht im Feature Dataset Resultate. Ansonsten kann die Ausführung des Werkzeugs zu einem Fehler führen.

> ℹ ArcGIS Pro Help: Geoprocessing Tool Summarize Within
>
> https://pro.arcgis.com/en/pro-app/latest/tool-reference/geoanalytics-desktop/summarize-within.htm

Überprüfe anschliessend die beiden generierten Outputs (Tabelle und Feature Class). Welche Informationen sind darin enthalten? Was bedeuten die Werte in den Feldern Join ID, Run_Nr und Count of Points innerhalb der Output Grouped Table?

3.3. Übung 4: Prozentualer Anteil der Stiche innerhalb des Waldes

Erweitere die Output Grouped Table (Resultat aus Übung 3) um ein Feld, in dem du anschliessend den prozentualen Anteil der Punkte pro Run innerhalb des Waldes berechnen kannst.

Hinweis: Pro Run gibt es 1076 Features

> ℹ ArcGIS Pro Help: Calculate Field
>
> https://pro.arcgis.com/en/pro-app/tool-reference/data-management/calculate-field.htm

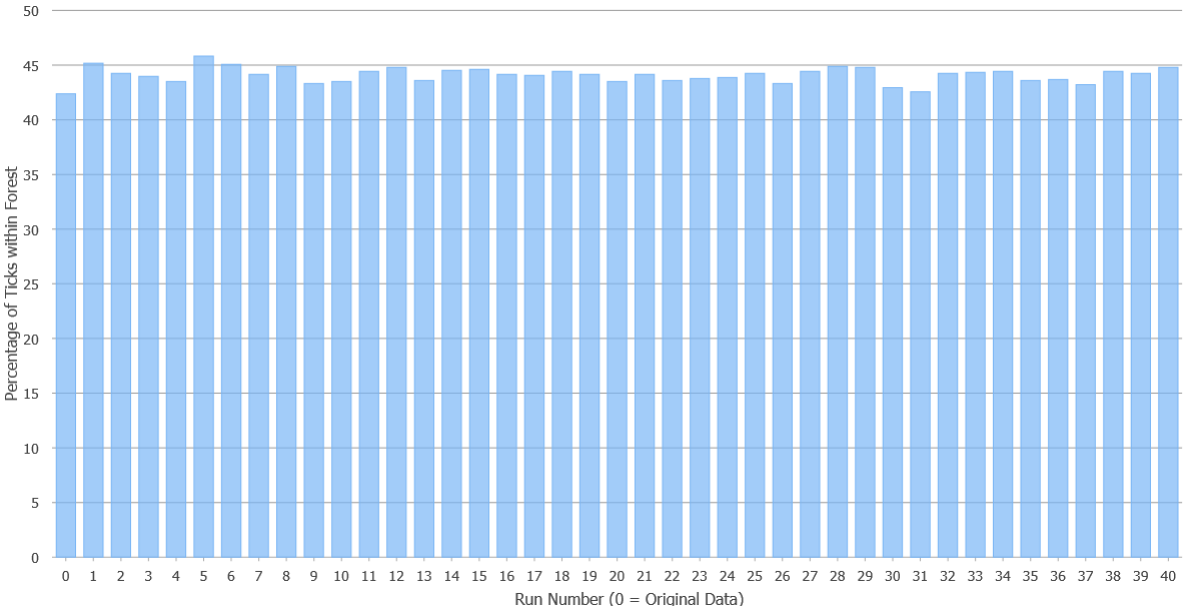
3.4. Übung 5: Diagramm (Bar Chart) mit prozentualem Anteil der individuellen Runs

Erstelle zuerst eine Kopie der Output Grouped Table in der Projekt-Geodatenbank. Entferne nachfolgend in der soeben erstellten Kopie alle Records welche die Summen der Stiche ausserhalb des Waldes beinhalten.

Hinweis: beachte die Werte im Feld *Join_ID*

> ℹ ArcGIS Pro Help: Select Layer By Attribute
>
> https://pro.arcgis.com/en/pro-app/tool-reference/data-management/select-layer-by-attribute.htm

Erstelle anschliessend ein Balkendiagramm welches die unterschiedlichen prozentualen Anteile der Stiche innerhalb des Waldes pro Run symbolisiert. Gehe hierfür in das Menüband “Standalone Table” und wähle “Create Chart”.



Beachte die Werte und deren Spannweite. Was bedeuten die Werte? Was für Aussagen bezüglich Unsicherheit (Genauigkeit) können bei den Stichdaten gemacht werden?

3.5. Übung 6: Berechnung des Mittelwertes aller Runs

Nutze die Tabelle der in Übung 3 erstellten Feature Class und berechne den prozentualen Anteil aller Stiche (über alle Runs) welche sich innerhalb des Waldes befinden. Vergleiche diesen Wert mit dem Wert aus den Originaldaten.

4. Unschärfe des Waldrandes

Neuere Forschungen haben ergeben, dass sich Zecken vermehrt im Waldrandbereich aufhalten und dadurch insbesondere dort das Risiko von einer Zecke gestochen zu werden besonders hoch ist. Deshalb sind wir interessiert daran, ob Zeckenstiche von App-Nutzern auch vermehrt im Bereich des Waldrandes gemeldet wurden. Im Gegensatz zu den vorherigen Übungen, welche die Erfassungsgenauigkeit von Punktdaten untersuchte, liegt der Fokus nun auf der Unschärfe des Waldrands. Wie wird ein Waldrand überhaupt definiert? Handelt es sich um eine scharf abgegrenzte Linie? Falls ja, wo liegt diese Linie exakt? Sind die Baumkronen, die Krautschicht oder die einzelnen Stämme ausschlaggebend? Oder handelt es sich beim Waldrand um einen Bereich mit einem bestimmten Abstand von einer zu definierenden Waldrandlinie? Wie würdest du in der untenstehenden Abbildung den Waldrand festlegen und wie würdest du dies GIS-technisch umsetzen?

Abbildung 4.1.: Bildquelle: K. Spörri (2013) in Waldrandaufwertung im Kt. Aargau (Forschungsgruppe Vegetationsökologie, ZHAW)

4.1. Übung 7: Waldrandlinie aus Waldpolygon erstellen

Die Waldrandlinie kann mit Hilfe des Geoverarbeitungswerkzeug "Polygon to Line" hergeleitet werden. Überlege dir, ob vorgängig eine Selektion im Input Feature notwendig ist.

> **i** ArcGIS Pro Help: Polygon To Line
>
> https://pro.arcgis.com/en/pro-app/tool-reference/data-management/polygon-to-line.htm

Betrachte das Resultat und überlege dir wie genau die digitale Waldrandlinie dem Waldrand in der Realität entspricht? Kann das Objekt Waldrand überhaupt grenzscharf abgebildet werden (vgl. rote Linie in Abbildung)?

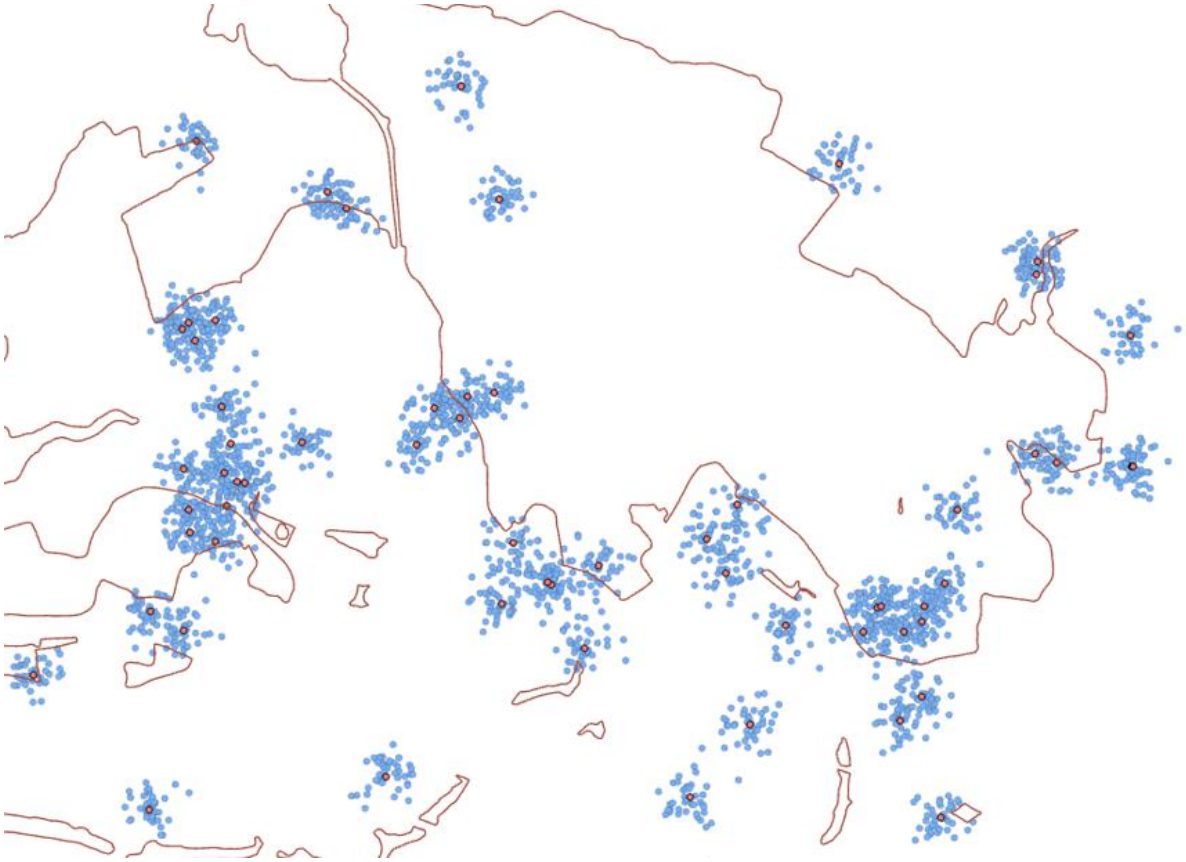


Abbildung 4.2.: Rote Linie: abgeleitete Waldrandlinie / orange Punkte: Stichstandorte original / blaue Punkte: Stichstandorte 40 Monte Carlo Runs

4.2. Übung 8: Waldrandbereich festlegen

Wir gehen davon aus, dass ein Waldrand kaum mit einer grenzscharfen Linie repräsentiert werden kann. In der Realität entspricht ein Waldrand wohl eher einem bestimmten Bereich. Der Waldrand an sich hat eine Unschärfe (Vagueness). In der GIS-Welt wird eine solche Unschärfe oft mit einem Puffer mit einem zu definierenden Abstand zu einer Linie (Waldrandlinie) abgebildet.

Erstelle basierend auf der in Übung 7 erstellten Waldrandlinie nun den Waldrandbereich mit einem äusseren und inneren Abstand von 25 Metern. Benutze hierfür das dir bereits bekannte Geoverarbeitungswerkzeug "Buffer". Achte bei der Umsetzung auf die korrekte Festlegung der Werkzeugparameter.

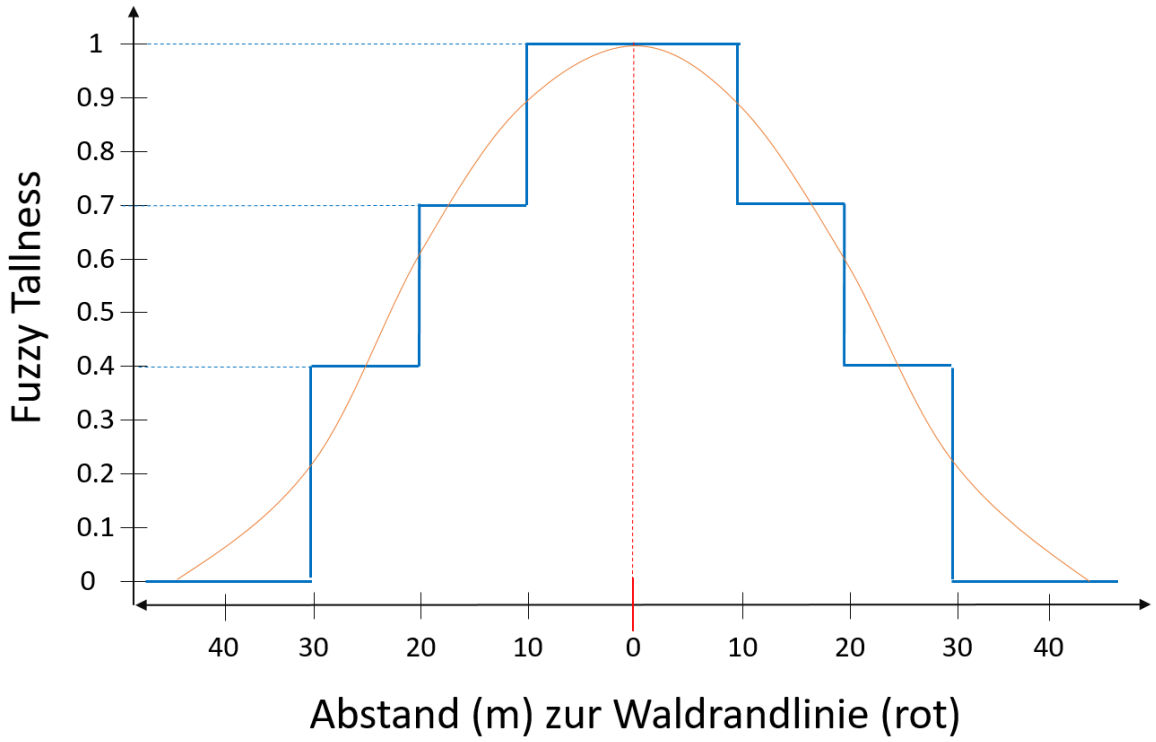
> ℹ ArcGIS Pro Help: Buffer
>
> https://pro.arcgis.com/en/pro-app/tool-reference/analysis/buffer.htm

Optional Analog wie in Übungen 3 bis 5 könntest du nun untersuchen, welchen Einfluss die Unschärfe des Waldrandes darauf hat, ob Zeckenstichstandorte im Waldrandbereich liegen oder nicht.

4.3. Übung 9: Distanz zu Waldrandlinie berechnen

Nun wollen wir diese Unschärfe des Konzepts Waldrand noch erweitern und die grenzscharfe Pufferdistanz aus der vorigen Übung mit einer graduellen Zugehörigkeit (membership function) ersetzen. Hierfür sollen folgende Fuzzy Tallness Werte definiert werden (vgl. Abbildung): Abstand zu Waldrandlinie:

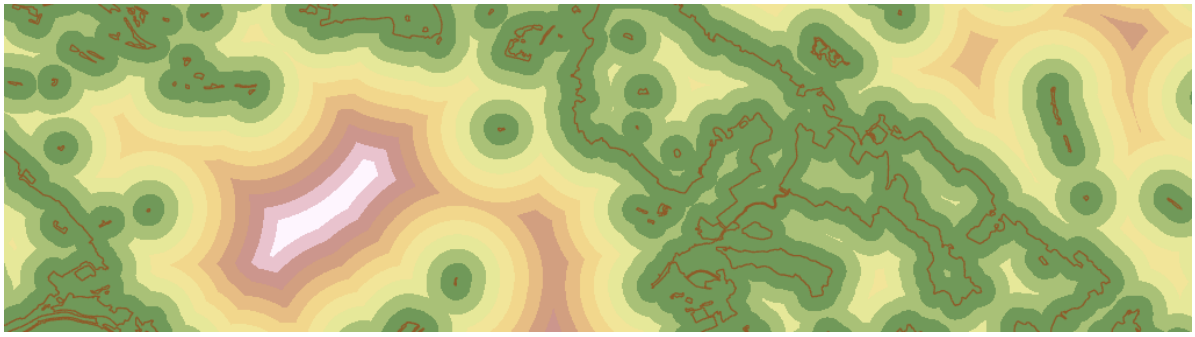
- 10 Meter = 1
- 10 bis 20 = 0.7
- 20 bis 30 = 0.4
- > 30 Meter = 0



Berechne nun als ersten Schritt für den gesamten Untersuchungsraum den Abstand zur nächstgelegenen Waldrandlinie. Nutze hierfür das Geoverarbeitungswerkzeug "Euclidean Distance". Lege die Output Cell Size auf 1 Meter fest und definiere in den Environments das Output Coordinate System und den Raum in dem diese globale Rasterfunktion ausgeführt wird (Extent = Untersuchungsgebiet).

> **i** ArcGIS Pro Help: Euclidean Distance
>
> https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/euclidean-distance.htm

Die Ausführung dieser Rasterfunktion kann einige Minuten dauern.



4.4. Übung 10: Zuweisung der Fuzzy Tallness Werte

Jeder Zelle mit einem Distanzwert soll nun ein Fuzzy Tallness Wert gemäss Abbildung in Übung 9 zugeteilt werden. Verwende hierfür das Geoverarbeitungswerkzeug “Reclassify”.

> ℹ ArcGIS Pro Help: Reclassify
>
> https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/reclassify.htm

Beim Reklassifizieren können den neuen Werten nur ganzzahlige Werte zugeteilt werden. Verwende deshalb die Werte 100, 70, 40 und 0 statt 1, 0.7, 0.4 und 0. Nutze anschliessend zuerst das Werkzeug “Float” und dann “Divide” um die ganzzahligen Werte wieder in Kommawerte umzurechnen.

- Float
- Divide

4.5. Übung 11: Unschärfe-Werte den Stichstandorten zuweisen

Die für jede Zelle definierten Unschärfe-Werte (membership zum Waldrand) werden nun den Daten mit den Stichstandorten zugeteilt. Dies erfolgt mittels “Extract Values to Points”.

> ℹ ArcGIS Pro Help: Extract Values to Points
>
> https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/extract-values-to-points.htm

Dabei sollen die Unschärfe-Werte (1 / 0.7 / 0.4 / 0) vorerst nur den Original-Stichstandortdaten zugeteilt werden (Input point feature = *Tick_Original*). Erstelle eine grafische Visualisierung (bsp. Histogramm).

Beantworte anschliessend folgende Fragen:

Wie viele Zeckenstiche im Original-Datensatz verfügen über Waldrand Unschärfe-Werte von 1, 0.7, 0.4 und 0 und welche Aussage kannst du damit machen?

5. Automatisierung mit ModelBuilder

ModelBuilder ist eine visuelle Programmiersprache für die Erstellung von Geoverarbeitungs-Workflows. ModelBuilder wird zum Modellieren und Automatisieren von räumlichen Analysen und Datenverwaltungsprozessen verwendet. Du kannst Geoverarbeitungsmodelle in ModelBuilder erstellen und ändern, wo ein Modell als ein Schema dargestellt wird, das Sequenzen von Prozessen und Geoverarbeitungswerkzeugen verkettet, indem die Ausgabe eines Prozesses als die Eingabe eines anderen Prozesses verwendet wird.

Mache dich auf der Online Hilfe von ArcGIS Pro mit den Möglichkeiten des ModelBuilders vertraut:

> ℹ ArcGIS Pro Help: What is ModelBuilder
>
> https://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/modelbuilder/what-is-modelbuilder-.htm

5.1. Übung 12: Waldrand-Fuzzy-Modell in ModelBuilder erstellen

Erstelle ein Modell mit ModelBuilder in dem die 5 Prozesse von Übung 9 bis 11 automatisiert ausgeführt werden können. Das Modell kann anschliessend wie ein normales Werkzeug aufgerufen werden. Als Modellparameter müssen die Fuzzy Tallness Distanzen resp. Werte angepasst sowie ein neuer Output Datensatz erstellt werden können.

Vorgehen:

1. Gehe in das Menüband “Analysis”.
2. Wähle den Button ModelBuilder. Damit wird ein neues, noch leeres Modell erstellt.
3. Gehe in das Catalog Pane und navigiere zu “Toolboxes > Unsicherheit.tbx” und benenne das Modell sinnvoll (Name und Label). Hinweis: keine Sonderzeichen und Leerschläge erlaubt.
4. Gehe im Catalog Pane in das Register “History”.
5. Füge die letzten 5 Prozesse (Übungen 9 bis 11) nacheinander mittels Drag und Drop dem neuen Modell hinzu.
6. Klicke im Modell mit rechter Maustaste auf das Tool Reclassify und wähle im Kontextmenü Create Variable > From Parameter > Reclassification

7. Klicke mit rechter Maustaste auf die soeben erstellte Modellvariable und wähle im Kontextmenü “Parameter”. Diese Modellvariable erscheint nun beim Aufrufen des Modells als veränderbarer Parameter, d.h. die Fuzzy Tallness Distanzen und Werte können angepasst werden.
8. Klicke mit rechter Maustaste auf den Final Output und definiere ihn ebenfalls als Modellparameter.
9. Speichere das Modell via Menüband ModelBuilder > Save
10. Gehe im Catalog auf Toolboxes > Unsicherheit.tbx > und öffne dein Modell mittels Doppelklick.
11. Gebe die Modellparameter ein (Fuzzy Tallness Distanzen & Werte und Output Dataset) und führe das Modell mit Run aus.

Mit Hilfe von solchen Modellen kannst du deine Geoverarbeitungsketten auf einfache Art und Weise automatisieren .

6. Leistungsnachweis

Als Leistungsnachweis wird eine schriftliche Dokumentation der Lösung dieser Übung erwartet. Die Dokumentation sollte vier A4-Seiten nicht überschreiten und muss Folgendes beinhalten:

- Name und Vorname der Autorin oder des Autors.
- Resultate der Übung aufbereitet mittels ansprechenden Grafiken, Tabellen und evtl. Karten.
- Beantwortung der in dieser Übung enthaltenen Fragen.
- Kurze Diskussion der Resultate.
- Beschreibung des ModelBuilder Modells inklusive:
  - Grafische Abbildung des ModelBuilder Modells
  - Welche Modellparameter wurden definiert und weshalb
  - Stichwortartige Beschreibung was das Modell berechnet

Abgabeform und -termin

Lade deine vollständige Dokumentation spätestens bis Freitag 6. Oktober 2023 als PDF über Moodle hoch (Abgabe Leistungsnachweis Datenqualität und Unsicherheit).

Teil II.

Programmieren I

In diesem Block bekommt ihr euren ersten Kontakt mit Python und lernt dabei auch gerade JupyterLab kennen, um mit Python zu interagieren. Er soll einen Einstieg in die Programmierwelt von Python bieten und spezifisch zeigen wie man räumliche Fragestellungen mit frei verfügbarer Software lösen kann. Die Voraussetzung für diesen Kurs ist eine Offenheit, neue Tools und Ansätze kennen zu lernen, die Bereitschaft für lösungsorientiertes Arbeiten sowie etwas Hartnäckigkeit.

Vorbereitung

Im Modul Angewandte Geoinformatik benutzen wir Python mit Jupyter Lab. Mit der Installation von ArcGIS sollte die nötige Software bereits installiert sein. Es muss also lediglich getestet werden, ob dies auch wie erwartet funktioniert. Bitte führt die Instruktionen unter dem Punkt Mit ArcGIS Installation durch.

Wer lieber auf einem anderen Betriebssystem als Windows arbeiten möchte, sollte es für den Teil Programmieren I - III auch können. In diesem Fall braucht es aber ein bisschen Vorbereitung. Befolgt in diesem Fall die Schritte unter dem Punkt Ohne ArcGIS Installation durch.

Mit ArcGIS Installation (v.a. Windows Nutzer)

Um zu testen, ob die Installation bei Dir auch funktioniert, musst Du die paar Schritte im Video (s.u.) ausführen:

1. Python Command Prompt suchen und starten
2. Mittels dem Befehl `cd` in dein Persönliches Laufwerk wechseln (`cd C:\Users\DeinUserName\Desktop`, siehe dazu diesen Video)
3. Mittels des Befehls `jupyter lab` das Programm Juypter Lab starten
4. Mit Klick auf den Button ein neues Notebook erstellen

Ohne ArcGIS Installation (v.a. Mac / Linux Nutzer)

Wer keine ArcGIS installation hat, sollte folgende Software installieren:

1. Überprüfe, ob Python > 3.9 installiert ist. Tippe dafür folgenden Befehl in den Terminal:
   `bash  python3 --version`
2. Wenn die Python nicht vorhanden oder die Version älter / kleiner als 3.9 ist, installiert eine aktuelle Version von Python: https://www.python.org/downloads/
3. Installiert Miniconda von dieser Website: https://docs.conda.io/projects/miniconda/en/latest/

4. Tippt folgende Befehle in der Commandozeile ein:

```
conda create --name geopython1
conda activate geopython1   # Windows nutzer: conda activate geopython1
conda install -c conda-forge jupyterlab pandas
```

Daten

Im Kurs werdet ihr nachstehende Datensätze benötigen, die ihr im Moodlekurs unter *Programmieren I → Datensätze* herunterladen könnt.

Tabelle 6.1.: Datensätze für den Teil "Programmieren I - III"

Datensatz (inkl. Link)	Beschreibung
zeckenstiche.csv	Eine CSV mit 10 Zeckenstich-Meldungen im Kanton Zürich
zeckenstiche_full.csv	Eine CSV mit 1'076 Zeckenstich-Meldungen im Kanton Zürich
wald.gpkg	Ein Geodatensatz mit einem flächendeckenden (lückenlosen) Polygon, welche den Kanton Zürich in "Wald" und "nicht Wald" unterscheidet

Übungsziele

- Python kennen lernen, erste Interaktionen mit Python durch die Commandline
- Erste Erfahrungen mit JupyterLab sammeln
- Die wichtigsten Datentypen in Python kennen lernen (`bool`, `str`, `int`, `float`, `list`, `dict`)
- Pandas DataFrames kennen lernen und einfache Manipulationen durchführen

7. Python Command Prompt

Um mit `R` zu Programmieren habt ihr bisher vermutlich in RStudio gearbeitet. RStudio ist eine sogenannte "Integrierte Entwicklungsumgebung" (i.e. IDE), wo ihr eure R-Scripts entwicklen konntet. Zu Beginn werden wir in diesem Kurs *ohne* Entwicklungsumgebung arbeiten und Python von der Konsole aus bedienen.

Startet dazu den *Python Command Prompt* indem ihr `Windowstaste` drückt und nach dieser Applikation sucht (einfach den Namen eingeben). Nach dem öffnen der Applikation sollte sich ein schwarzes Fenster öffnen und auf eure Eingabe warten (siehe Abbildung 7.1).

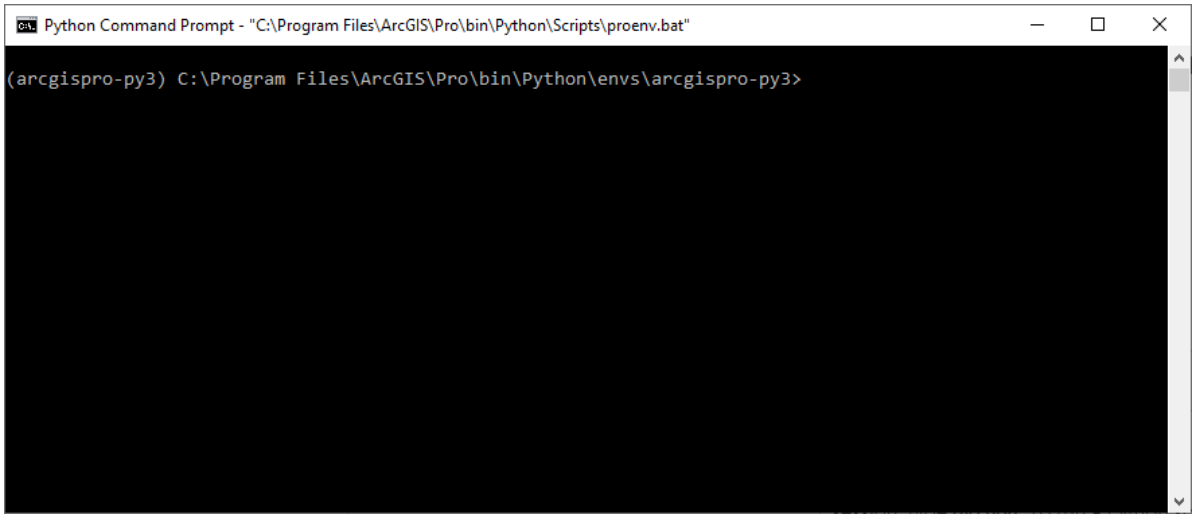


Abbildung 7.1.: Python Command Prompt (i.e. "Eingabeaufforderung")

Dieses Fenster ermöglicht es uns, auf sehr einfache (aber auch etwas trockene) Weise mit Python zu interagieren. Dazu müssen wir aber zuerst noch in das Python Programm einsteigen, indem wir `python` eintippen und `Enter` drücken.

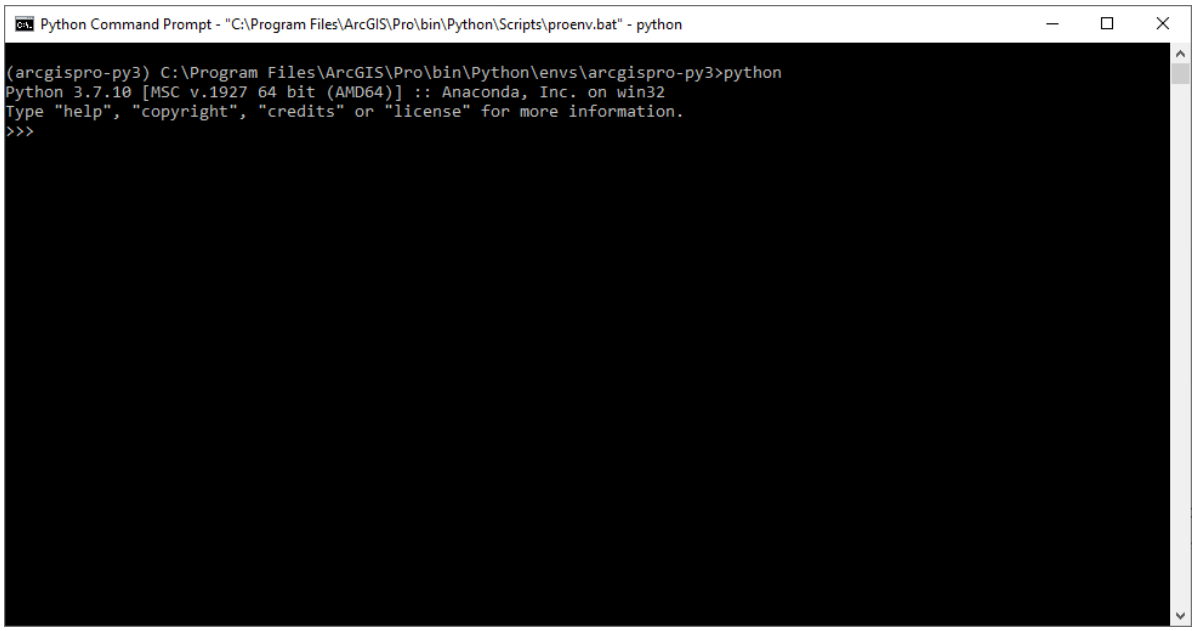


Abbildung 7.2.: Python Command Prompt im Python Programm (beachte die drei >>>)

8. Primitive Datentypen

Bei primitiven Datentypen handelt es sich um die kleinste Einheit der Programmiersprache, sie werden deshalb auch "atomare Datentypen" genannt. Alle komplexeren Datentypen (Tabellarische Daten, Bilder, Geodaten) basieren auf diesen einfachen Strukturen. Die für uns wichtigsten Datentypen lauten: *Boolean*, *String*, *Integer* und *Float*. Das sind ähnliche Datentypen wie ihr bereits aus R kennt:

Python	R	Beschreibung	Beispiel	in Python
Boolean	Logical	Logische Werte ja / nein	Wahr / Falsch	`regen = True`
String	Character	Textinformation	Bern, Luzern	`stadt = "Bern"`
Integer	Integer	Zahl ohne Nachkommastelle	Anzahl Einwohner	`bern = 133115`
Float	Double	Zahl mit Nachkommastelle	Temperatur	`temp = 22.5`

8.1. Boolean

Hierbei handelt es sich um den einfachsten Datentyp. Er beinhaltet nur zwei Zustände: Wahr oder Falsch. In Python werden diese mit `True` oder `False` definiert (diese Schreibweise muss genau beachtet werden). Beispielsweise sind das Antworten auf geschlossene Fragen.

```
regen = True # "es regnet"

sonne = False # "die Sonne scheint nicht"

type(sonne)
```

```
bool
```

Um zu prüfen, ob ein bestimmter Wert `True` oder `False` ist verwendet man `is True`. Will man also fragen ob es regnet, wir dies folgendermassen formuliert:

```
# regnet es?
regen is True
```

```
True
```

Ob die Sonne scheint, lautet folgendermassen (natürlich müssen dazu die Variabel `sonne` bereits existieren):

```
# scheint die Sonne?
sonne is True
```

```
False
```

8.2. String

In sogenannten *Strings* werden Textinformationen gespeichert. Beispielsweise können das die Namen von Ortschaften sein.

```
stadt = "Bern"
land = "Schweiz"

type(stadt)
```

```
str
```

Strings können mit `+` miteinander verbunden werden

```
stadt + " ist die Hauptstadt der " + land
```

```
'Bern ist die Hauptstadt der Schweiz'
```

8.3. Integer

In *Integer* werden ganzzahlige Werte gespeichert, beispielsweise die Anzahl Einwohner einer Stadt.

```
bern_einwohner = 133115

type(bern_einwohner)
```

```
int
```

8.4. Float

Als *Float* werden Zahlen mit Nachkommastellen gespeichert, wie zum Beispiel die Temperatur in Grad Celsius.

```
bern_flaeche = 51.62

type(bern_flaeche)
```

```
float
```

9. Übung 1

9.1. Übung 1.1

Erstelle eine Variabel `vorname` mit deinem Vornamen und eine zweite Variabel `nachname` mit deinem Nachnamen. Was sind `vorname` und `nachname` für Datentypen?

```
vorname = "Guido"
nachname = "van Rossum"

type(vorname) # es handelt sich um den Datentyp "str", also String (Text)
```

9.2. Übung 1.2

"Klebe" Vor- und Nachname mit einem Leerschlag dazwischen zusammen.

```
vorname + " " + nachname
```

9.3. Übung 1.3

Erstelle eine Variabel `groesse_cm` mit deiner Körpergrösse in Zentimeter. Was ist das für ein Datentyp?

```
groesse_cm = 184
type(groesse_cm) # es handelt sich hierbei um den Datentyp "integer"
```

9.4. Übung 1.4

Ermittle deine Grösse in Fuss auf der Basis von `groesse_cm` (1 Fuss entspricht 30.48 cm). Was ist das für ein Datentyp?

```
groesse_fuss = groesse_cm / 30.48
type(groesse_fuss) # es handelt sich um den Datentyp "float"
```

9.5. Übung 1.5

Erstelle eine boolsche Variable blond und setzte sie auf `True` wenn diese Eigenschaft auf dich zutrifft und `False` falls nicht.

```
blond = False
```

9.6. Übung 1.6

Erstelle eine Variabel `einwohner` mit der Einwohnerzahl der Schweiz (8'603'900, per 31. Dezember 2019). Erstelle eine zweite Variabel `flaeche` (ohne Umlaute!) mit der Flächengrösse der Schweiz (41'285 km^2). Berechne nun die Einwohnerdichte.

```
einwohner = 8603900
flaeche = 41285

dichte = einwohner / flaeche

dichte
```

9.7. Übung 1.7

Erstelle eine Variabel `gewicht_kg` (kg) und `groesse_m` (m) und berechne aufgrund von `gewicht_kg` und `groesse_m` ein BodyMassIndex ($BMI = \frac{m}{l^2}$, m: Körpermasse in Kilogramm, l: Körpergrösse in Meter).

```
gewicht_kg = 85
groesse_m = groesse_cm / 100

gewicht_kg / (groesse_m * groesse_m)
```

10. Komplexe Datentypen

Im letzten Kapitel haben wir primitive Datentypen angeschaut. Diese stellen eine gute Basis dar, in der Praxis haben wir aber meistens nicht *einen* Temperaturwert, sondern eine Liste von Temperaturwerten. Wir haben nicht *einen* Vornamen sondern eine Tabelle mit Vor- und Nachnamen. Dafür gibt es in Python komplexere Datenstrukturen die als Gefässe für primitive Datentypen betrachtet werden können. Auch hier finden wir viele Ähnlichkeiten mit R:

Python	R	Beschreibung	Beispiel
List	(Vector)	werden über die Position abgerufen	Beachtet die eckigen Klammern: `hexerei = [3,2,1]`
Dict	List	werden über ein Schlüsselwort abgerufen	Beachtet die geschweiften Klammern: `langenscheidt = {"trump":"erdichten"}`
DataFrame	Dataframe	Tabellarische Daten	Spezialfall einer `dict` `pd.DataFrame({"x":[1,2,3], "y": [3,4,5]}`

In Python gibt es noch weitere komplexe Datentypen wie *Tuples* und *Sets*. Diese spielen in unserem Kurs aber eine untergeordnete Rolle. Ich erwähne an dieser Stelle zwei häufig genannte Typen, damit ihr sie schon mal gehört habt:

- *Tuples*:
  - sind ähnlich wie *Lists*, nur können sie nachträglich nicht verändert werden. Das heisst, es ist nach der Erstellung keine Ergänzung von neuen Werten oder Löschung von bestehenden Werten möglich.
  - sie werden mit runden Klammern erstellt: `mytuple = (2,2,1)`
- *Sets*

- sind ähnlich wie *Dicts*, verfügen aber nicht über `keys` und `values`
- jeder Wert wird nur 1x gespeichert (Duplikate werden automatisch entfernt)
- sie werden mit geschweiften Klammern erstellt: `myset = {3,2,2}`

Wohl das einfachste Gefäss, um mehrere Werte zu speichern sind Python-Listen, sogenannte *Lists*. Diese *Lists* werden mit eckigen Klammern erstellt. Die Reihenfolge, in denen die Werte angegeben werden, wird gespeichert. Das erlaubt es uns, bestimmte Werte aufgrund ihrer Position abzurufen.

Eine *List* wird folgendermassen erstellt:

```
hexerei = [3,1,2]
```

Der erste Wert wird in Python mit `0` (!!!) aufgerufen:

```
hexerei[0]
```

```
3
```

```
type(hexerei)
```

```
list
```

Im Prinzip sind *Lists* ähnlich wie *Vectors* in R, mit dem Unterschied das in Python-Lists unterschiedliche Datentypen abgespeichert werden können. Zum Beispiel auch weitere, verschachtelte Lists:

```
chaos = [23, "ja", [1,2,3]]
```

```
# Der Inhalt vom ersten Wert ist vom Typ "Int"
type(chaos[0])
```

```
int
```

```
# Der Inhalt vom dritten Wert ist vom Typ "List"

type(chaos[2])
```

```
list
```

11. Übung 2

11.1. Übung 2.1

1. Erstelle eine Variable `vornamen` bestehend aus einer *List* mit 3 Vornamen
2. Erstelle eine zweite Variable `nachnamen` bestehend aus einer *List* mit 3 Nachnamen
3. Erstelle eine Variable `groessen` bestehend aus einer *List* mit 3 Grössenangaben in Zentimeter.

```
vornamen = ["Christopher", "Henning", "Severin"]
nachnamen = ["Annen", "May", "Kantereit"]

groessen = [174, 182, 162]
```

11.2. Übung 2.2

Wie erhältst du den ersten Eintrag in der Variable `vornamen`?

```
vornamen[0]
```

11.3. Übung 2.3

Listen können durch die Methode `append` ergänzt werden (s.u.). Ergänze die Listen `vornamen`, `nachnamen` und `groessen` durch je einen Eintrag.

```
vornamen.append("Malte")
```

```
nachnamen.append("Huck")

groessen.append(177)
```

11.4. Übung 2.4

Ermittle die Summe aller Werte in `groessen`. Tip: Nutze dazu `sum()`

```
sum(groessen)
```

11.5. Übung 2.5

Ermittle die Anzahl Werte in `groessen` . Tip: Nutze dazu `len()`

```
len(groessen)
```

11.6. Uebung 2.6

Berechne die durchschnittliche Grösse aller Personen in `groessen`. Tip: Nutze dazu `len()` und `sum()`.

```
sum(groessen) / len(groessen)
```

11.7. Übung 2.7

Ermittle nun noch die Minimum- und Maximumwerte aus `groessen` (finde die dazugehörige Funktion selber heraus).

```
min(groessen)
max(groessen)
```

12. JupyterLab

Bisher haben wir ohne Enwicklungsumgebung (IDE) gearbeitet, was auf die Dauer sehr mühsam sein kann. Erstens wird unser Code nirgens gespeichert und zweitens wird bei der Entwicklung des Codes wenig unterstützt (z.B. keine Autovervollständigung).

Deshalb arbeiten wir von nun an in einer ordentlichen Entwicklungsumgebung, und zwar in JupyterLab. JupyterLab ist eine unter *Data Scientists* sehr beliebte Entwicklungsumgebung. Sie hat zwei Eigenheiten, die anfänglich zu etwas Verwirrung sorgen können. JupyterLab wird:

1. mit dem "Command Prompt" gestartet
2. über den Webbrowser bedient

Um JupyterLab zu starten verlässt ihr Python im Command Prompt in dem ihr `exit()` oder `quit()` eintippt und mit Enter bestätigt. Danach könnt ihr mit dem Befehl `jupyter lab` eben diesen starten.

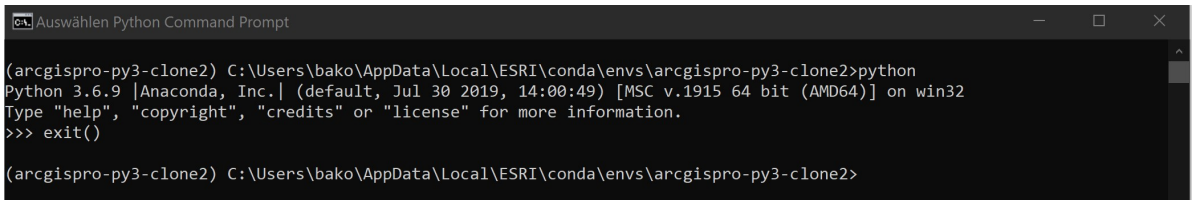


Abbildung 12.1.: Beendet den Python mit dem Befehl `quit()` oder `exit()`

Hinweis

- JupyterLab wird nun in eurem Browser angezeigt, das Programm läuft aber in eurer Konsole. Das heisst:
  - ihr düft die Konsole nicht schliessen, während ihr mit JupyterLab arbeitet
  - um JupyterLab zu beenden verwendet ihr in der Konsole die Tastenkombination Ctrl + C
- Beachtet den Pfad, der in Python Command Prompt dargestellt wird (in Abbildung 12.1 wird der Pfad `C:\Users\bako\AppData\Local\ESRI\conda\envs\arcgispro-py3-clone2` angezeigt)

- Dieser Pfad ist das “Arbeitsverzeichnis”, in welchem JupyterLab gestartet wird
- Dieser Pfad ist nicht sinnvoll, wir werden ihn in einem nächsten Schritt ändern.

Wie erwähnt ist das automatisch gewählte Arbeitsverzeichnis (im Beispiel `C:\Users\bako\AppData\Local\ESRI` nicht ideal. Es macht viel mehr Sinn, ein vertrautes Verzeichnis zu wählen (z.B. den Desktop oder wo ihr eure Unterrichtsunterlagen abspeichert). Dieses Verzeichnis müssen wir wählen, *bevor* wir JupyterLab starten. Schliesst deshalb JupyterLab wieder in dem ihr in der Konsole die Tastenkombination Ctrl + C verwendet.

Wählt nun ein sinvolleres Verzeichnis und kopiert dessen Pfad aus dem File Explorer. Gebt nun in der Konsole den Befehl `cd` *(change directory)*, gefolgt von dem Pfad zu dem von uns gewünschten Verzeichnis. Im Beispiel werden wir auf den Desktop wechseln `cd C:\Users\bako\Desktop` (siehe Abbildung 12.2).

Nun sollte sich der angezeigte Pfad (vor dem `>`) in eurer Konsole ändern. **Ist dies nicht der Fall, hat das Wechseln des Verzeichnisses nicht funktioniert**. War das wechseln des Verzeichnisses erfolgreich, könnt ihr erneut JupyterLab starten: Im Unterschied zu vorher sollten die Dateien, die Ihr nun seht (siehe Abbildung 12.3) vertrauter sein.

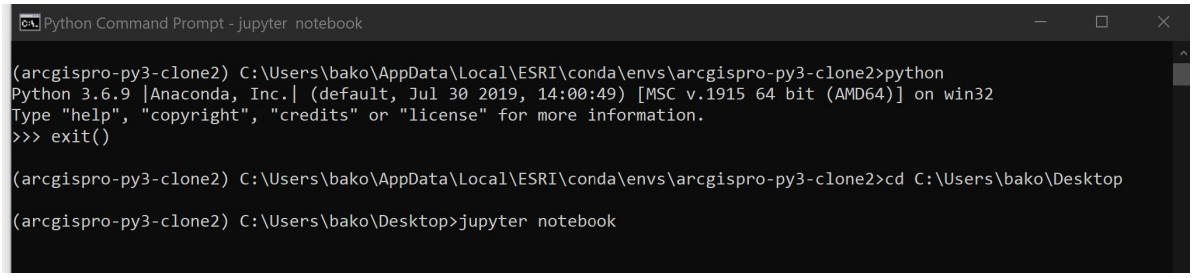


Abbildung 12.2.: JupyterLab starten

Öffnet nun ein neues Jupyter Notebook File in dem ihr auf *New* > *Python 3* klickt (siehe Abbildung 12.2).

Das Bedienen von JupyterLab lässt sich nicht gut in Worte fassen. Wir wollen diese deshalb in einer kurzen Demo vorzeigen, alternativ (falls du selbständig arbeitest) kannst du dir auch nachstehenden Video anschauen (ab 1:46):

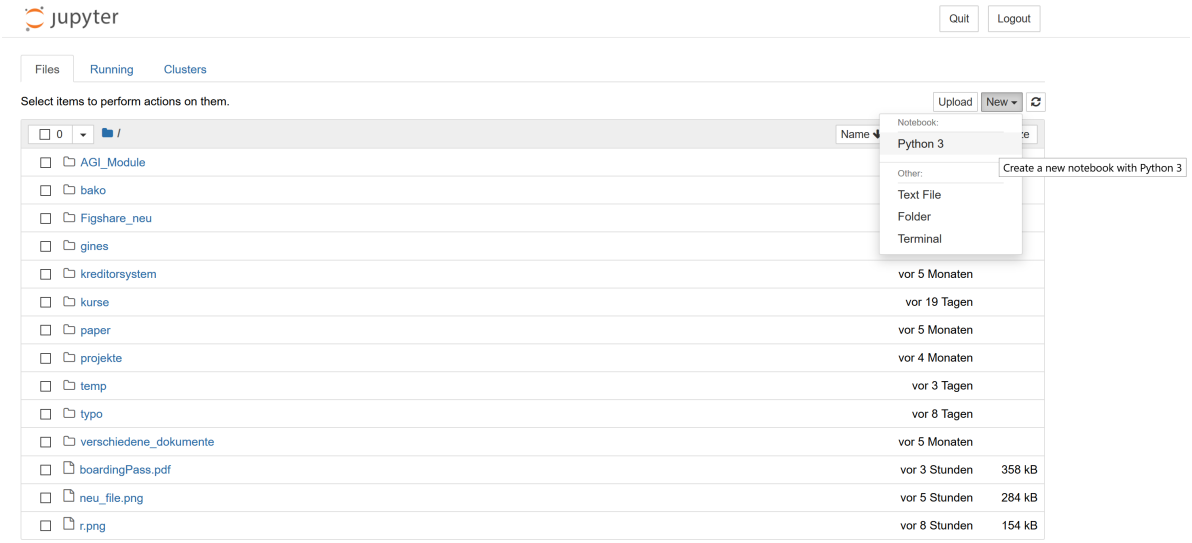


Abbildung 12.3.: Öffnen eines neuen Jupyter Notebook Files

13. Dictionaries

In den letzten Übungen haben wir einen Fokus auf Listen gelegt. Nun wollen wir ein besonderen Fokus auf den Datentyp *Dictionary* legen.

Ähnlich wie eine List, ist eine Dictionary ein Behälter wo mehrere Elemente abgespeichert werden können. Wie bei einem Wörterbuch bekommt jedes Element ein "Schlüsselwort", mit dem man den Eintrag finden kann. Unter dem Eintrag "trump" findet man im Langenscheidt Wörterbuch (1977) die Erklärung "erdichten, schwindeln, sich aus den Fingern saugen".


III. v/i. 5. Trumpf ausspielen, trumpfen, stechen.
trump² [trʌmp] v/t. ~ up erdichten, zs.-schwindeln, sich aus den Fingern saugen; 'trumped-up [trʌmpt] adj. erfunden, erlogen, falsch: ~ charges.
trump·er·y ['trʌmpəri] I. s. 1. Plunder m, Ramsch m, Schund m; 2. fig. Gewäsch n, Quatsch m; II. adj. 3. Schund..., Kitsch..., kitschig, geschmacklos; 4. fig. billig, nichtssagend: ~ arguments.
trum·pet ['trʌmpit] I. s. 1. ♪ Trom'pete f: to blow one's own ~ fig. sein eigenes Lob singen; the last ~ die Posaune des Jüngsten Gerichts; 2.

In Python würde man diese *Dictionary* folgendermassen erstellen:

```
langenscheidt = {"trump":"erdichten- schwindeln- sich aus den Fingern saugen"}
```

Schlüssel (von nun an mit *Key* bezeichnet) des Eintrages lautet "trump" und der dazugehörige Wert (*Value*) "erdichten- schwindeln- aus den Fingern saugen". Beachte die geschweiften Klammern (`{` und `}`) bei der Erstellung einer Dictionary.

Eine *Dictionary* besteht aber meistens nicht aus einem, sondern aus mehreren Einträgen: Diese werden Kommagetrennt aufgeführt.

```
langenscheidt = {"trump":"erdichten- schwindeln- sich aus den Fingern saugen", "trumpery":"Pl
```

Der Clou der *Dictionary* ist, dass man nun einen Eintrag mittels dem *Key* aufrufen kann. Wenn wir also nun wissen wollen was "trump" heisst, ermitteln wir dies mit der nachstehenden Codezeile:

```
langenscheidt["trump"]
```

```
'erdichten- schwindeln- sich aus den Fingern saugen'
```

Um eine *Dictionary* mit einem weiteren Eintrag zu ergänzen, geht man sehr ähnlich vor wie beim Abrufen von Einträgen.

```
langenscheidt["trumpet"] = "trompete"
```

Ein *Key* kann auch mehrere Einträge enthalten. An unserem Langenscheidts Beispiel: Das Wort "trump" ist zwar eindeutig, doch "trumpery" hat vier verschiedene Bedeutungen. In so einem Fall können wir einem Eintrag auch eine *List* von Werten zuweisen. Beachte die Eckigen Klammern und die Kommas, welche die Listeneinträge voneinander trennt.

```
langenscheidt["trumpery"] = ["Plunder- Ramsch- Schund",
                             "Gewäsch- Quatsch",
                             "Schund- Kitsch",
                             "billig- nichtssagend"]
langenscheidt["trumpery"]
```

```
['Plunder- Ramsch- Schund',
 'Gewäsch- Quatsch',
 'Schund- Kitsch',
 'billig- nichtssagend']
```

```
len(langenscheidt["trumpery"])
```

```
4
```

14. Übung 3

14.1. Übung 3.1

Erstelle eine *Dictionary* mit folgenden Einträgen: Vorname und Nachname von (d)einer Person. Weise diese Dictionary der Variable `me` zu.

```
me = {"vorname": "Guido", "nachname": "van Rossum"}
```

14.2. Übung 3.2

Rufe verschiedene Elemente aus der Dictionary via dem *Key* ab.

```
me["nachname"]
```

14.3. Übung 3.3

Nutze `me` um nachstehenden Satz (mit **deinen** *Values*) zu erstellen:

```
"Mein Name ist "+me["nachname"] +", "+ me["vorname"]+" "+me["nachname"]
```

```
'Mein Name ist van Rossum, Guido van Rossum'
```

14.4. Übung 3.4

Ergänze die Dictionary `me` durch einen Eintrag "groesse" mit (d)einer Grösse.

```
me["groesse"] = 181
```

14.5. Übung 3.5

Erstelle eine neue Dictionary `people` mit den *Keys* “vornamen”, “nachnamen” und “groesse” und jeweils 3 Einträgen pro *Key*.

```
people = {"vornamen": ["Christopher", "Henning", "Severin"], "nachnamen": ["Annen","May", "Ka
```

14.6. Übung 3.6

Rufe den **ersten** Vornamen deiner *Dict* auf. Dazu musst du dein Wissen über Listen und Dictionaries kombinieren.

```
people["vornamen"][0]
```

14.7. Übung 3.7

Rufe den **dritten** Nachname deiner *Dict* auf.

```
people["nachnamen"][2]
```

14.8. Übung 3.8

Berechne den Mittelwert aller grössen in deiner Dict

```
sum(people["groessen"])/len(people["groessen"])
```

15. DataFrames

Schauen wir uns nochmals die *Dictionary* `people` aus der letzten Übung an. Diese ist ein Spezialfall einer Dictionary: Jeder Eintrag besteht aus einer Liste von gleich vielen Werten. Wie bereits erwähnt, kann es in einem solchen Fall sinnvoll sein, die Dictionary als Tabelle darzustellen.

```
people = {"vornamen": ["Christopher", "Henning", "Severin"], "nachnamen": ["Annen","May", "Ka
```

```
import pandas as pd # Was diese Zeile beudeutet lernen wir später

people_df = pd.DataFrame(people)

people_df
```

```
ModuleNotFoundError: No module named 'pandas'
```

16. Übung 4

16.1. Übung 4.1

Importiere `pandas` und nutze die Funktion `DataFrame` um `people` in eine DataFrame umzuwandeln (siehe dazu das Beispiel unten). Weise den Output der Variable `people_df` zu und schau dir `people_df` an.

```
import pandas as pd

people = {"vornamen": ["Christopher", "Henning", "Severin"], "nachnamen": ["Annen","May", "Ka

people_df = pd.DataFrame(people)
```

16.2. Übung 4.2

In der Praxis kommen Tabellarische Daten meist als "csv" Dateien daher. Wir können aus unserer eben erstellten DataFrame sehr einfach eine csv Datei erstellen. Führe das mit folgendem Code aus und suche anschliessend die erstellte csv-Datei.

```
people_df.to_csv("people.csv")
```

> **Achtung!**
>
> Falls ihr nicht wisst, wo das csv abgespeichet ist solltet ihr das Kapitel Kapitel 12 nochmals durchlesen.
> Die *Working Directory* zu kennen ist wichtig, besonders auch für die nächste Aufgabe. Falls ihr die aktuelle Working Directory neu setzen und dafür JupyterLab schliessen müsst, denkt daran die Notebook vorgängig zu speichern und im File Explorer aufzusuchen.

16.3. Übung 4.3

Genau so einfach ist es eine csv zu importieren. Lade die Datei "zeckenstiche.csv" (siehe Tabelle 6.1) herunter und speichere es im aktuellen Arbeitsverzeichnis ab. Importiere mit folgendem Code die Datei "zeckenstiche.csv".

```
# ich habe die Daten in einem Unterordner "data" abgespeichert
zeckenstiche = pd.read_csv("data/zeckenstiche.csv")
```

16.4. Übung 4.4

Die *DataFrame* `zeckenstiche` beinhaltet x und y Koordinaten für jeden Unfall in den gleichnamigen Spalten. Wir können die Stiche mit einem Scatterplot räumlich visualisieren. Führe dazu folgenden Code aus.

```
fig = zeckenstiche.plot.scatter("x","y")

fig.axis("equal")
# "equal" stellt sicher, das die x und y Achsen gleich skaliert sind
# dies ist sinnvoll, da es sich ja um Schweizer Koordinaten (Meter) handelt
```

16.5. Übung 4.5

Um eine einzelne Spalte zu selektieren (z.B. die Spalte "ID"), kann man gleich vorgehen wie bei der Selektion eines Eintrags in einer *Dictionary*. Probiere es aus.

```
zeckenstiche["ID"]
```

16.6. Übung 4.6

Auch das Erstellen einer neuen Spalte ist identisch mit der Erstellung eines neuen *Dictionary* Eintrags. Erstelle eine neue Spalte "Stichtyp" mit dem Wert "Zecke" auf jeder Zeile (s.u.).

```
zeckenstiche["Stichtyp"] = "Zecke"
```

```
zeckenstiche
```

17. Übung 5

Diese letzte Übung dient lediglich der Vorbereitung auf Programmieren II. Ihr müsst die Übung nicht verstehen, sondern nur ausführen. Melde dich, falls es Probleme gibt.

17.1. Übung 5.1 Neue Conda Umgebung erstellen

1. Starte Python Command Prompt (siehe Python Command Prompt)
2. Führe den Befehl: `conda create --name geopython2` aus
3. Bestätige die Installation mit `y`
4. Prüfe, ob u.a. folgende Meldung im Terminal erscheint (melde dich falls nicht):

```
#
# To activate this environment, use
#
#     $ activate geopython2
#
# To deactivate an active environment, use
#
#     $ deactivate
```

5. Führe den Befehl aus, der in der Meldung steht:

```
activate geopython2               # (Windows)
conda activate geopython2         # (MacOS, Linux)
```

6. Prüfe, ob die Eingabeforderung im Terminal nun mit `(geopython2)` beginnt (melde dich falls nicht)

```
# ↓ vorher
(arcgispro-py3) C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3>

# ↓ nachher
(geopython2)  C:\Users\rata\AppData\Local\ESRI\conda\envs\geopython>
```

7. Führe den folgenden Befehl aus:

```
conda install -c conda-forge jupyterlab geopandas matplotlib descartes
```

8. Bestätige die Installation mit `y`

9. Prüfe, ob die installation erfolgreich war (erscheint folgende Meldung im Terminal?):

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Retrieving notices: ...working... done
```

17.2. Übung 5.2 JupyterLab Desktop installieren

Die Tatsache, dass JupyterLab via dem *Command Prompt* gestartet wird und im Browser läuft, ist ungewohnt und auch etwas umständlich. Um nur schon eines der soeben erstellten Jupyter Notebooks (*.ipynb File) zu *betrachten* muss (1) der Terminal geöffnet, (2) in den richtigen Ordner navigiert (3) die richtige Conda Umgebung aktiviert und (4) JupyterLab gestartet werden.

Deshalb empfehlen wir die Installation von JupyterLab Desktop, welches JupyterLab als Desktop Applikation zur Verfügung stellt. JupyterLab Desktop kann man gemäss folgender Anleitung installieren: https://github.com/jupyterlab/jupyterlab-desktop#installation.

18. Anhang

18.1. Anhang 1: Arbeiten mit GoogleCollab

In diesem Modul fokussieren wir uns darauf, dass ihr eure Lokale Programmierumgebung aufsetzen und pflegen könnt. Falls das nicht möglich ist, könnt ihr auch mit GoogleCollab arbeiten. GoogleCollab ist eine kostenlose Online-Plattform, die euch die Möglichkeit gibt, Python-Notebooks zu erstellen und auszuführen. Ihr könnt dort auch eure Notebooks speichern und mit anderen teilen.

1. **Daten auf GoogleDrive Hochladen**: Erstellt einen Ordner auf GoogleDrive mit dem Namen `Modul-AGI` und ladet die Daten (siehe Tabelle 6.1) dort hoch.
2. **GoogleCollab öffnen**: Öffnet GoogleCollab unter https://colab.research.google.com/ und clickt auf den Button "New Notebook" / "Neues Notebook".
3. **Google Drive Mounten**: Mountet GoogleDrive, indem ihr folgenden Code in die erste Zelle eures Notebooks schreibt und ausführt:

   ```
   from google.colab import drive
   drive.mount('/content/drive')
   ```

4. **Libraries laden**: Auf GoogleCollab sind die meisten Libraries bereits installiert. Ihr müsst sie nur noch laden. Fügt dazu folgenden Code in die nächste Zelle ein und führt ihn aus:

   ```
   import pandas as pd
   import geopandas as gpd
   ```

   Unter Umständen müsst ihr zusätzliche Libraries installieren. GoogleCollab wird euch darauf hinweisen und liefert auch den passenden `pip`-Befehl (`pip` ist eine alternative zu `conda`). Um diesen Befehl auf GoogleCollab auszuführen, führt ihr den Befehl *in eurem Notebook* mit vorangestelltem Ausrufezeichen aus. Z.B:

   ```
   !pip install folium matplotlib mapclassify
   ```

5. **Daten laden**: Ladet die Daten, indem ihr folgenden Code in die nächste Zelle einfügt und ausführt:

```
# Solltet ihr der Ordner nicht Modul-AGI benannt haben, Pfad anpassen
zeckenstiche = pd.read_csv("/content/drive/MyDrive/Modul-AGI/zeckenstiche.csv")
```

Teil III.

Programmieren II

Letzte Woche habt ihr die wichtigsten Datentypen in Python kennen gelernt und sowohl den Command Prompt wie auch Juypter Notebook verwendet. Diese Woche werden wir Python Module genauer beleuchten und dabei *conda* kennenlernen. Zudem widmen wir uns dem Erstellen von Funktionen (*Functions*), Fallunterscheidungen (`if` und `else`) sowie dem Erstellen von Zufallszahlen.

Übungsziele

- Conda verstehen und beherrschen
- Python Module verstehen
- *Functions* kennenlernen und beherrschen
- *Function* auf eine ganze Spalte einer DataFrame anwenden können.

19. Conda

19.1. Was ist Conda?

Conda ist ein (von im Wesentlichen zwei[1]) Verwaltungssystem für Python[2] Bibliotheken welches unter Windows, macOS und Linux läuft. Mit Conda lassen sich diese Libraries und deren Abhängigkeiten schnell installieren, ausführen und aktualisieren. Ein zusätzliches, wichtiges Feature von Conda ist die Verwaltung von sogenannten "Virtuellen Umgebungen". Diese Umgebungen ermöglichen es, dass Package Installationen in abgeschottenen "Container" erfolgen. Das heisst, dass man unterschiedliche Versionen des gleichen Packages installieren kann, ohne das dies Probleme bereitet.

19.2. Conda installieren

Mit der Installation von ArcGIS wird (Mini-) Conda bereits mitgeliefert. Wer ArcGIS also schon hat, muss Conda nicht mehr installieren. Wer aber auf einem anderen Betriebssystem arbeiten möchte (oder ArcGIS auf Windows nicht installiert hat), kann Conda sehr einfach herunterladen und installieren: https://docs.conda.io/en/latest/miniconda.html.

19.3. Conda environment erstellen

Wenn man den Python Command Prompt startet, ist Conda bereits aktiviert. Dies ist daran erkennbar, dass vor dem Pfad (`C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3`) noch etwas in Klammern steht (`arcgispro-py3`, siehe Abbildung 19.1).

`arcgispro-py3` ist der Name des Conda Environments, das automatisch aktiviert wurde. Gleichzeitig ist dies das "default Python Environment", welches ArcGIS Pro jeweils nutzt (siehe Abbildung 19.2). Das heisst, dieses Environment wird verwendet wenn man Python innerhalb von ArcGIS nutzen möchte und hat dafür gewisse Libraries vorinstalliert. Dabei macht ArcGIS aber folgenden Hinweis: *Note: Cannot modify the default Python environment*

[1] Neben Conda gibt es noch den "Package Manager" `pip`, welchen wir im Unterricht aber nicht nutzen werden.
[2] Genau genommen ist Conda universell einsetzbar und kann auch Libraries von R und weiteren Programmiersprachen installieren.

(arcigspro-py3). Clone then activate a new environment first. Genau diese Freiheit wollen wir aber haben und erstellen deswegen ein eigenes Environment.

Um ein neues Environment mit dem Namen `geopython2` zu erstellen und aktivieren geht man wie folgt vor:

```
conda create --name geopython2          # erstellt das Environment
activate geopython                      # aktiviert das Environment (Windows)
conda activate geopython                # aktiviert das Environment (Mac / Linux)
```

Nun sollte in der Klammer der Name des eben erstellten Environment erscheinen (`geopython`). Selbstverständlich kann man den Namen des Environments selbst wählen, dabei sollten aber Umlaute, Sonderzeichen und Grossbuchstaben vermieden werden.

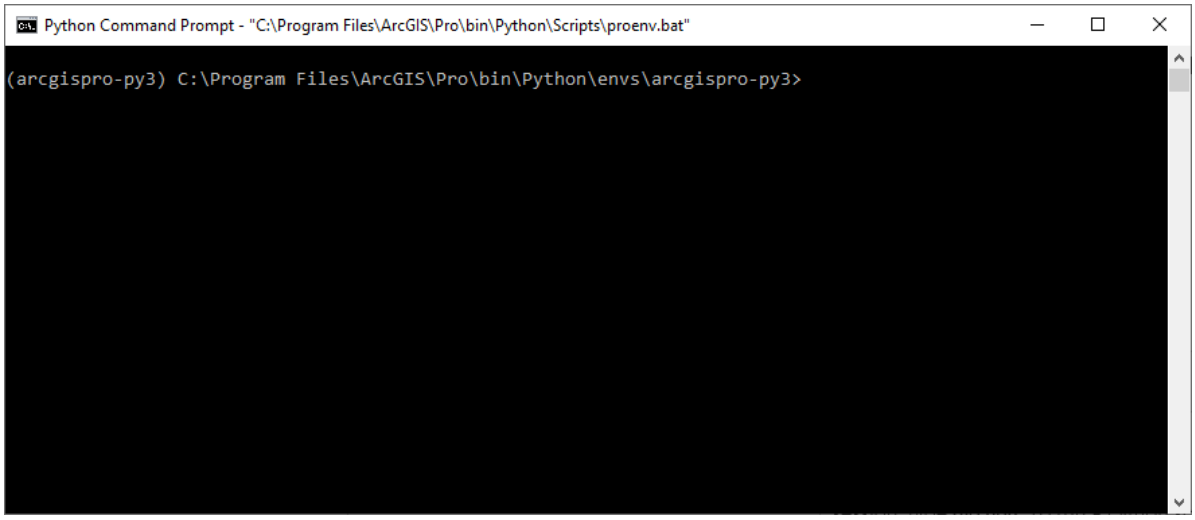


Abbildung 19.1.: Python Command Prompt (i.e. “Eingabeaufforderung”)

19.4. Conda packages installieren

Jetzt wo wir ein eigenes Conda Environment erstellt haben (`geopython`) müssen wir dieses mit Libraries befüllen. Aktuell ist es nämlich noch ziemlich leer, wie ihr mit `conda list` sehen könnt (versucht es aus!).

Letzte Woche hatten wir die Packages `pandas` sowie `jupyter lab` gebraucht. Diese Packages waren im Environment `arcgispro-py3` bereits vorinstalliert.

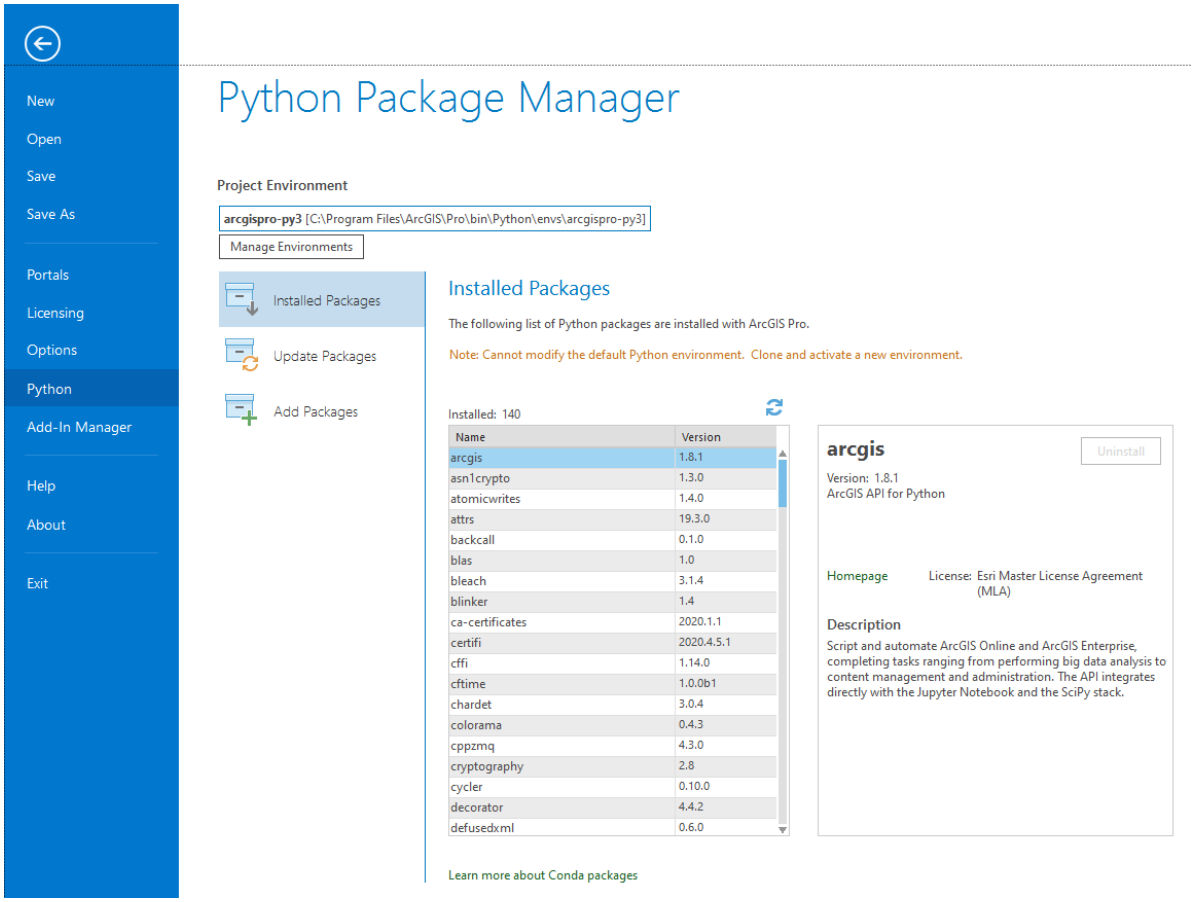


Abbildung 19.2.: ArcGIS nutzt ebenfalls Conda. Welches Conda Environment in der aktuellen ArcGIS Session benutzt wird ist ersichtlich, wenn man in ArcGIS zu *Projekt* > *Python* navigiert.

```
conda install -c conda-forge jupyterlab   # installiert jupyterlab
conda install -c conda-forge pandas       # installiert pandas
```

`-c` ist in den obigen `install` Befehlen ein sogenannter Flag welcher ankündigt, dass gleich der "Channel" angegeben wird. In beiden Fällen nutzen wir den "Channel" `conda-forge`, d.h. die Packages werden von conda-forge geladen. Welcher Channel angegeben werden muss kann über eine kurze Internetsuche in Erfahrung gebracht werden (z.B. nach "conda install pandas" suchen).

Bestätigt die Rückfragen (`Proceed ([y]/n)?`) mit y + Enter. Schaut euch nun euer Environment mit `conda list` an und startet danach `jupyter lab` indem ihr eben diesen Befehl in der Konsole eingebt. Wechselt aber vorher das Verzeichnis zu einem sinvolleren Ort (wie in Kapitel Kapitel 12 beschrieben).

19.5. Conda cheat sheet

In der folgenden Tabelle werden die Einzelschritte in der Verwendung von Conda nochmal zusammengefasst. Wichtig ist vor allem, wann dieser Schritt nötig ist und wie er ausgeführt wird. Um die Tabelle kompakt zu halten werden gewisse Details als Fussnote verlinkt.

Schritt	Wann ist dies nötig?	Details zum Vorgehen / Befehl für die Konsole[3]
1. Conda installieren (installiert das Program *conda*)	einmalig (ist nicht nötig, wenn ArcGIS Pro installiert ist)	Miniconda (empfohlen) oder anaconda herunterladen und installieren
2. *Virtual environment* erstellen (erstellt eine neue Arbeitsumgebung)	einmal pro Projekt nötig (wobei eine environment auch wiederverwendet werden kann)	in der Konsole: `conda create --name geopython`
3. *Virutal environment* aktivieren (schaltet den "Bearbeitungsmodus" ein)	jedes mal nöig wenn ein *Erweiterung installiert **oder** jupyter lab gestartet** werden soll	in der Konsole [4]: `activate geopython`
4. Jupyter lab installieren (fügt der virtuellen Umgebung diese IDE hinzu)	1x pro environment*	in der Konsole [5]: `conda install -c conda-forge jupyterlab`

[4] Unter Linux: `conda activate geopython`

[5] Falls die richtige environment noch nicht aktiviert ist, muss dies zuerst noch erfolgen (z.B `activate geopython`).

Schritt	Wann ist dies nötig?	Details zum Vorgehen / Befehl für die Konsole[3]
5. Jupyter lab starten (startet die IDE "JupyterLab")	jedes mal, wenn am Projekt gearbeitet wird	in der Konsole [6]: `jupyter lab`
6. Jupyter lab (JL) beenden (beendet "JupyterLab" in der Console)	wenn ihr die Konsole wieder braucht	Während JL läuft, ist die Konsole blockiert. Um JL zu beenden und die Konsole freizugeben: Tastenkombination CTRL + C
7. weitere Module [7] **installieren** (fügt der *environment* zB `pandas` hinzu)	jedes mal nötig, wenn ein Modul in einer Environment fehlt[8]	in der Konsole [9],[10]: `conda install -c conda-forge pandas`

[6] Falls die richtige environment noch nicht aktiviert ist, muss dies zuerst noch erfolgen (z.B `activate geopython`).

[7] In Programmieren I - III brauchen wir die Module `pandas`, `matplotlib`, `geopandas` und `descartes`

[8] Dies macht sich bemerkbar duch die Fehlermeldung `ModuleNotFoundError: No module named 'pandas'`

[9] Falls die richtige environment noch nicht aktiviert ist, muss dies zuerst noch erfolgen (z.B `activate geopython`).

[10] Falls JupyterLabs läuft und dadurch die Konsole blockiert ist, startet ihr am besten eine neue Konsole und aktiviert dort die entsprechende *environment*. Danach könnt ihr das Modul installieren

[3] Mit Konsole ist unter Windows *cmd* gemeint (Windowstaste > cmd). Unter Linux wird bash, auf Mac der Terminal verwendet.

20. Python Modules

20.1. Vergleich R vs. Python

Der Umgang mit Modulen ist in Python in vielerlei Hinsicht ähnlich wie in R. An dieser Stelle möchten wir die Unterschiede in einem Direktvergleich beleuchten. Dafür verwenden wir **ein fiktives Modul** namens `maler`, in Anlehnung an die Analogie des Hausbauens mit Spezialisten (siehe Vorlesungsfolien). Nehmen wir an, dieses Modul existiert als Python Modul wie auch als R Library.

20.1.1. Erweiterung installieren

In R ist die Installation einer *Library* selbst ein R-Befehl und wird innerhalb von R ausgeführt. Wenn wir keine Quelle angeben, woher die Library heruntergeladen werden soll, wird eine Default-Quelle verwendet, die im System hinterlegt ist (z.B. "https://cloud.r-project.org").

In Python ist dies leider etwas komplizierter, es braucht für die Installation einer Python *Library* eine Zusatzsoftware wie zum Beispiel `conda` (siehe dazu das Kapitel Kapitel 19). Es gibt auch noch andere Wege, wie zum Beispiel `pip`, aber diese lassen wir der Einfachheit an dieser Stelle weg.

in R:

```
install.packages("maler")
```

In Python:

```
conda install -c conda-foge maler
```

20.1.2. Erweiterung laden

Um eine Erweiterung nutzen zu können, müssen wir diese sowohl in R wie auch in Python in die aktuelle Session importieren. In R und Python sehen die Befehle folgendermassen aus:

in R:

```
library(maler)
```

in Python:

```
import maler
```

20.1.3. Erweiterung verwenden

Um eine Funktion aus einer *Library* in R zu verwenden, kann ich diese *Function* direkt aufrufen. In Python hingegen muss ich entsprechende Erweiterung der *Function* mit einem Punkt voranstellen.

Das ist zwar umständlicher, dafür aber weniger fehleranfällig. Angenommen zwei leicht unterschiedliche Funktionen heissen beide `wand_bemalen()`. Die eine stammt aus der Erweiterung `maler`, die andere aus der Erweiterung `maurer`. Wenn die Funktion in R aufgerufen wird ist nicht klar, aus welcher Library die Funktion verwendet werden soll. In Python ist im nachstehenden Beispiel unmissverständlich, dass `wand_bemalen()` aus dem Modul `maler` gemeint ist.

in R:

```
wand_bemalen()
```

in Python:

```
maler.wand_bemalen()
```

20.2. Python Eigenheiten

In Python gibt es in Bezug auf die Verwendung von Modulen ein paar Eigenheiten, die wir aus der R Welt nicht kennen. Es ist wichtig diese Eigenheiten zu kennen, denn man trifft sie immer wieder an.

20.2.1. Modul mit Alias importieren

Da es umständlich sein kann, jedesmal `maler.wand_bemalen()` voll auszuschreiben, können wir dem Modul beim Import auch einen "Alias" vergeben. Für gewisse populäre Module haben sich solche Aliasse eingebürgert. Beispielsweise wird `pandas` meist mit dem Alias `pd` importiert. Es ist sinnvoll, sich an diese Konventionen zu halten. Übertragen auf unser `maler` beispiel sieht der Import mit einem Alias folgendermassen aus:

```
import maler as m                # importiert "maler" mit dem Alias "m"
m.wand_bemalen()                 # nun wird "m." vorangestellt statt "maler."
```

20.2.2. Einzelne *Function* importieren

Es gibt noch die Variante, explizit eine spezifische *Function* aus einem Modul zu laden. Wenn man dies macht, kann man die Funktion ohne vorangestelltes Modul nutzen (genau wie in R). Dies sieht folgendermassen aus:

```
from maler import wand_bemalen  # importiert nur die Funktion "wand_bemalen"
wand_bemalen()                  # das Voranstellen von "maler." ist nun nicht nötig
```

20.2.3. Alle *Functions* importieren

Zusätzlich ist es möglich, **alle** *Functions* aus einem Modul so zu importieren, dass der Modulname nicht mehr erwähnt werden muss. Diese Notation wird nicht empfohlen, aber es ist wichtig sie zu kennen.

```
from maler import *             # importier alle Funktionen (*) von "maler"
wand_bemalen()                  # das Voranstellen von "maler." ist nun nicht nötig
```

21. JupyterLab Desktop

Letzte Woche haben wir JupyterLab von der Konsole aus gestartet und im Browser verwendet. Dies was für den Einstieg sehr praktisch, da nichts installiert werden musste. Die Erfahrung hat aber gezeigt, dass die Verwendung von JupyterLab im Browser gerade für Einsteiger sehr verwirrend sein kann, weshalb wir euch letzte Woche in Auftrag gegeben haben, JupyterLab Desktop zu installieren.

Um diese Woche in Python zu coden, könnt ihr als erstes JupyterLab Desktop starten. Diese Anwendung solltet ihr in eurem Startmenü finden.

22. *Function* Basics

Ein Grundprinzip von Programmieren ist "DRY" (*Don't repeat yourself*). Wenn unser Script viele ähnliche Codezeilen enthält ist das ein Zeichen dafür, dass man besser eine *Function* schreiben sollte. Das hat viele Vorteile: Unter anderem wird der Code lesbarer, einfacher zu warten und kürzer.

Um mit Python gut zurechtzukommen ist das Schreiben von eigenen *Functions* unerlässlich. Dies ist auch nicht weiter schwierig: Eine *Function* wird mit `def` eingeleitet, braucht einen Namen, einen Input und einen Output.

Wenn wir zum Beispiel eine Function erstellen wollen die uns grüsst, so geht dies folgendermassen:

```
def sag_hallo():
    return "Hallo!"
```

- Mit `def` sagen wir: "Jetzt definiere ich eine Function".
- Danach kommt der Name der *Function*, in unserem Fall `sag_hallo` (mit diesem Namen können wir die *Function* später wieder aufrufen).
- Als Drittes kommen die runden Klammern, wo wir bei Bedarf Inputvariablen (sogenannte Parameter) festlegen können. In diesem ersten Beispiel habe ich keine Parameter festgelegt.
- Nach der Klammer kommt ein Doppelpunkt was bedeutet: "jetzt wird gleich definiert, was die Funktion tun soll".
- Auf einer neuen Zeile wird eingerückt festgelegt, was die Function eben tun soll. Meist sind hier ein paar Zeilen Code vorhanden.
- Die letzte eingerückte Zeile (in unserem Fall ist das die einzige Zeile) gibt mit `return` an, was die *Function* zurück geben soll (der Output). In unserem Fall soll sie "Hallo!" zurück geben.

Das war's schon! Jetzt können wir diese *Function* schon nutzen:

```
sag_hallo()
```

```
'Hallo!'
```

Diese *Function* ohne Input ist wenig nützlich. Meist wollen wir der *Function* etwas - einen Input - übergeben können. Beispielsweise könnten wir der *Function* unseren Vornamen übergeben, damit wir persönlich gegrüsst werden:

```
def sag_hallo(vorname):
    return "Hallo " + vorname + "!"
```

Nun können wir der Function ein Argument übergeben. In folgendem Beispiel ist `vorname` ein Parameter, "Guido" ist sein Argument.

```
sag_hallo(vorname = "Guido")
```

```
'Hallo Guido!'
```

Wir können auch eine *Function* gestalten, die mehrere Parameter annimmt. Beispielweise könnte `sag_hallo()` zusätzlich noch einen Parameter `nachname` erwarten:

```
def sag_hallo(vorname, nachname):
    return "Hallo " + vorname + " " + nachname + "!"
```

```
sag_hallo(vorname = "Guido", nachname = "van Rossum")
```

```
'Hallo Guido van Rossum!'
```

23. Übung 5

Bevor du mit den Übungen anfägst, solltest du bereits eine eigene Conda Environment nach der Anleitung in Kapitel "Kapitel 19" erstellt haben.

23.1. Übung 5.1

Erstelle eine Function, die `gruezi` heisst, einen Nachnamen als Input annimmt und per Sie grüsst.

```
def gruezi(nachname):
    return "Guten Tag, " + nachname
```

```
# Das Resultat soll in etwa folgendermassen aussehen:
gruezi(nachname = "van Rossum")
```

```
'Guten Tag, van Rossum'
```

23.2. Übung 5.2

Erstelle eine neue Funktion `gruezi2` welche im Vergleich zu `gruezi` einen weiteren Parameter namens `anrede` annimmt.

```
def gruezi2(nachname, anrede):
    return "Guten Tag, " + anrede + " " + nachname
```

```
# Das Resultat soll in etwa folgendermassen aussehen:
gruezi2(nachname = "van Rossum", anrede = "Herr")
```

```
'Guten Tag, Herr van Rossum'
```

23.3. Übung 5.3

Erstelle eine Funktion `add` die zwei Zahlen summiert.

```
def add(zahl1, zahl2):
    return zahl1 + zahl2
```

```
# Das Resultat sollte folgendermassen aussehen:
add(zahl1 = 2, zahl2 = 10)
```

```
12
```

23.4. Übung 5.4

Erstelle eine Funktion `square`, welche den Input quadriert.

```
def square(zahl):
    return zahl * zahl

# oder

def square(zahl):
    return zahl**2
```

```
# Das Resultat sollte folgendermassen aussehen:
square(zahl = 5)
```

```
25
```

23.5. Übung 5.5

Erstelle eine Funktion `meter_zu_fuss`, die eine beliebige Zahl von Meter in Fuss konvertiert. Zur Erinnerung: 30.48 cm ergeben 1 Fuss.

```
def meter_zu_fuss(meter):
    fuss = meter * 100 / 30.48
    return fuss
```

```
# Das Resultat sollte folgendermassen aussehen:

meter_zu_fuss(meter = 1.80)
```

```
5.905511811023622
```

24. *Function* Advanced

24.1. Standart-Werte

Man kann für einzelne (oder alle) Parameter auch Standardwerte festlegen. Das sind Werte die dann zum Zug kommen, wenn der Nutzer der Funktion das entsprechende Parameter leer lässt. Schauen wir dazu nochmals `sag_hallo()` an.

```
def sag_hallo(vorname):
    return "Hallo " + vorname + "!"
```

Um diese Funktion zu nutzen muss dem Parameter `vorname` ein Argument übergeben werden, sonst erhalten wir eine Fehlermeldung.

```
sag_hallo()
```

```
TypeError: sag_hallo() missing 1 required positional argument: 'vorname'
```

Wenn wir möchten, dass gewisse Parameter auch ohne Argument auskommen, dann könnnen wir einen Standartwert festlegen. So wird der Parameter optional. Bespielsweise könnte `sag_hallo()` einfach *Hallo Du!* zurückgeben, wenn kein Vorname angegeben wird. Um dies zu erreichen, definieren wir den Standartwert bereits innerhalb der Klammer, und zwar folgendermassen:

```
def sag_hallo(vorname = "Du"):
    return "Hallo " + vorname + "!"

# Wenn "vorname" nicht angegeben wird:
sag_hallo()
```

```
'Hallo Du!'
```

> **Wichtig**
>
> Wenn mehrere Parameter in einer Funktion definiert werden, dann kommen die optionalen Parameter **immer zum Schluss**.

24.2. Reihenfolge der Argumente

Wenn die Argumente in der gleichen Reihenfolge eingegeben werden, wie sie in der *Function* definiert sind, müssen die Parameter **nicht** spezifiziert werden (z.B: `anrede=`, `nachname=`).

```
def gruezi2(nachname, anrede):
    return "Guten Tag, " + anrede + " "+nachname

gruezi2("van Rossum", "Herr")
```

```
'Guten Tag, Herr van Rossum'
```

Wenn wir die Reihenfolge missachten, ist der Output unserer Funktion fehlerhaft:

```
gruezi2("Herr", "van Rossum")
```

```
'Guten Tag, van Rossum Herr'
```

Aber wenn die Parameter der Argumente spezifiziert werden, können wir sie in jeder beliebigen Reihenfolge auflisten:

```
gruezi2(anrede = "Herr", nachname = "van Rossum")
```

```
'Guten Tag, Herr van Rossum'
```

24.3. Funktionen auf mehreren Zeilen

Bisher waren unsere Funktionen sehr kurz und einfach und wir benötigten dafür immer nur zwei Zeilen: Die erste Zeile begann die *Function*-Definition (`def..`) und die zweite Zeile retournierte bereits die Lösung `return(...)`.

Zwischen diesen beiden Komponenten haben wir aber viel Platz, den wir uns zu Nutze machen können. Wir können hier Kommentare hinzufügen wie auch unsere Funktion in Einzelschritte aufteilen um den Code lesbarer zu machen.

```
def gruezi2(nachname, anrede):
    """
    Meine coole Grüezi funktion
    Diese Funktion soll Menschen freundlich grüssen.
    Sie nimmt zwei Inputs: nachname und andrede, beides "strings"
    """
    gruss = "Guten Tag, " + anrede + " "+nachname
    return gruss
```

Allgemeine Kommentare werden in Python mit `#` hinzugefügt, Funktionen werden aber mit drei Anführungs- und Schlusszeichen kommentiert. Diese Kommentare erscheinen eleganterweise wenn man die Hilfe zu dieser Funktion mit `help()` aufruft:

```
help(gruezi2)
```

```
Help on function gruezi2 in module __main__:

gruezi2(nachname, anrede)
    Meine coole Grüezi funktion
    Diese Funktion soll Menschen freundlich grüssen.
    Sie nimmt zwei Inputs: nachname und andrede, beides "strings"
```

24.4. Globale und Lokale Variabeln

Innerhalb einer *Function* können nur die Variabeln verwendet werden, die der *Function* als Argumente übergeben (oder innerhalb der Funktion erstellt) werden. Diese nennt man "lokale" Variabeln und sind nur lokal in der *Function* vorhanden. Im Gegensatz dazu stehen "globale" Variabeln, welche Teil der aktuellen Session sind.

Versuchen wir das mit einem Beispiel zu verdeutlichen. Angenommen wir definieren gobal die Variabel `vorname`:

```
# Wir definieren globale Variabel
vorname = "Guido"

# Nun erstellen wir eine Function, welche diese Variabel ("vorname") nutzen soll:
def sag_hallo(vorname):
    return "Hallo " + vorname

# Wenn wir jetzt aber die Function ausführen wollen, entsteht die Fehlermeldung,
```

```
# dass "vorname" fehlt (obwohl wir vorname ja schon definiert haben)
sag_hallo()
```

```
TypeError: sag_hallo() missing 1 required positional argument: 'vorname'
```

24.5. Lambda-Function

Mit dem Begriff `lambda` kann eine *Function* verkürzt geschrieben werden. Wir werden dies im Unterricht kaum verwenden, es ist aber doch gut davon gehört zu haben. Nachstehend wird die Funktion `sag_hallo()` in der bekannten, wie auch in der verkürzten Form definiert.

Herkömmliche Weise:

```
def sag_hallo(vorname):
    return "Hi "+vorname
```

Verkürzt mit `lambda`:

```
sag_hallo = lambda vorname: "Hi "+vorname
```

25. Übung 6

25.1. Übung 6.1

Erstelle eine Funktion namens `times`, die zwei Zahlen miteinander multipliziert.

```
def times(x, y):
    return x * y
```

```
times(2, 2)
```

25.2. Übung 6.2

Die eben erstellte Funktion `times` benötigt 2 Argumente (die miteinander multipliziert werden). Wandle den zweiten Parameter in einen optionalen Parameter um (mit dem Defaultwert `1`).

Zusatzaufgabe: Was passiert, wenn du den ersten Parameter in einen optionalen Parameter umwandelst?

```
def times(x, y = 1):
    return x * y
```

```
times(3)
```

```
# (Zusatzaufgabe)
def times(x = 1, y):
    return x * y
```

25.3. Übung 6.3

Erstelle eine Funktion namens `bmi`, die aus Grösse und Gewicht einen BodyMassIndex berechnet ($BMI = \frac{m}{l^2}$, m: Körpermasse in Kilogramm, l: Körpergrösse in Meter). Das Resultat soll etwa folgendermassen aussehen:

```
def bmi(groesse_m, gewicht_kg):
    return gewicht_kg / (groesse_m * groesse_m)
```

```
bmi(groesse_m = 1.8, gewicht_kg = 88)
```

```
27.160493827160494
```

25.4. Übung 6.4

Erstelle eine Funktion `mittelwert()`, welche den Mittelwert aus einer Liste (`List`) von Zahlen berechnet. Nutze dazu `sum()` und `len()` analog Kapitel 11.6. Das Resultat sollte folgendermassen aussehen:

```
def mittelwert(zahlen):
    return sum(zahlen) / len(zahlen)
```

```
meine_zahlen = [50, 100, 550, 1000]
mittelwert(meine_zahlen)
```

```
425.0
```

25.5. Übung 6.5

Erstelle eine Funktion `celsius_zu_farenheit`, welche eine beliebige Zahl von Grad Celsius in Grad Farenheit konvertiert. Zur Erinnerung: *Temperatur in °F = Temperatur in °C x 1,8 + 32*.

```
def celsius_zu_farenheit(celsius):
    farenheit = celsius * 1.8 + 32
    return farenheit
```

Das Resultat sollte folgendermassen aussehen:

```
celsius_zu_farenheit(celsius = 25)
```

```
77.0
```

25.6. Übung 6.6

Schreibe die letzte Funktion `celsius_zu_farenheit` in der *lambda* Notation.

```
celsius_in_farenheit2 = lambda celsius: celsius * 1.8 + 32
```

26. If / Else

Ein wichtiger Bestandteil von Programmieren sind Fallunterscheidungen. Mit Fallunterscheidungen können wir mit unterschiedlichen Situationen verschieden umgehen.

Die einfache Fallunterscheidung ist die bedingte Verzweigung. Die Syntax der einfachen `if`-Anweisung lautet folgendermassen:

```
if Bedingung:
    # Anweisungen 1
else:
    # Anweisungen 2
```

Zum Beispiel:

```
# hier alter eingeben:
alter = 35

if alter < 40:
    print("Backstreet Boys: 'I want it that way'")
else:
    print("The Jackson 5: 'I want you back'")
```

```
Backstreet Boys: 'I want it that way'
```

Wenn wir mehr als zwei verschiedene Fälle haben, können diese mit `elif` dazwischen geschaltet werden.

```
if Bedingung1:
    # Anweisungen 1
elif Bedingung2:
    # Anweisungen 2
elif Bedingung3:
    # Anweisungen 3
else:
    # Anweisungen 4
```

Zum Beispiel:

```
# hier alter eingeben:
alter = 35

if alter < 20:
    print("Kesha: 'Tik Tok'")
elif alter < 30:
    print("Destiny's Child: 'Say My Name'")
elif alter < 40:
    print("Mariah Carey: 'Vision of love'")
else:
    print("Blondie: 'Call me'")
```

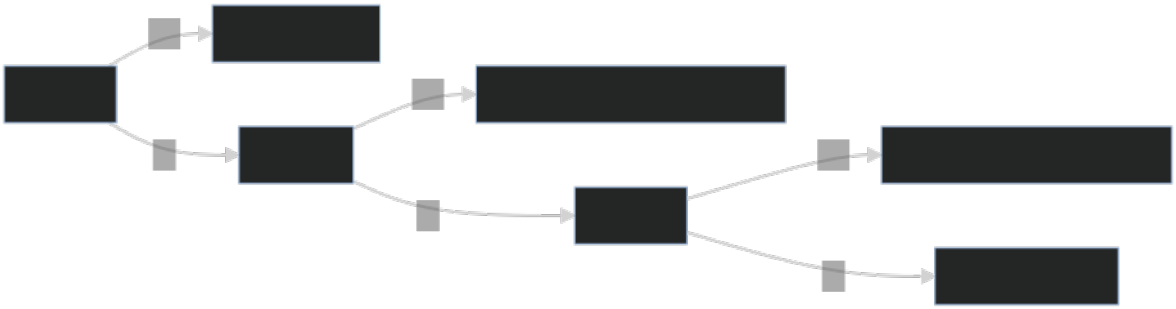
```
Mariah Carey: 'Vision of love'
```

Solche Fallunterscheidungen sind vor allem in Funktionen sehr praktisch. Aus dem obigen Beispiel können wir beispielsweise eine Funktion erstellen, die uns ein Lied in Abhängigkeit zu unserem Alter vorschlägt:

```
def suggest_song(alter):
    if alter < 20:
        song = "Kesha: 'Tik Tok'"
    elif alter < 30:
        song = "Destiny's Child: 'Say My Name'"
    elif alter < 40:
        song = "Mariah Carey: 'Vision of love'"
    else:
        song = "Blondie: 'Call me'"
    return "I suggest the song " + song + ". Enjoy!"

suggest_song(24)
```

```
"I suggest the song Destiny's Child: 'Say My Name'. Enjoy!"
```

Dabei ist wichtig, dass man sich jeweils die Ausführungslogik vor Augen führt.



27. Übung 7

27.1. Übung 7.1

Erstelle eine neue Funktion `gruezi3` die `gruezi` (aus Kapitel 23.1) um einem weiteren Parameter `uhrzeit` erweitert. Die Funktion soll vor 18 Uhr (`uhrzeit < 18`) mit "Guten Tag" und sonst mit "Guten Abend" grüssen. Um es einfach zu halten: Die Zeitangbe muss in Dezimalzahlen erfolgen (also nicht `20:15 Uhr` sondern `20.25`).

```
def gruezi3(nachname, uhrzeit):
    if(uhrzeit < 18):
        grussform = "Tag"
    else:
        grussform = "Abend"

    return "Guten " + grussform + " " + nachname
```

```
# die Lösung sollte etwa folgendermasssen funktionieren:

gruezi3("Guido", uhrzeit = 21)
```

```
'Guten Abend Guido'
```

27.2. Übung 7.2

Erweitere die Funktion `gruezi2` so, dass zwischen folgende Uhrzeiten berücksichtigt werden:

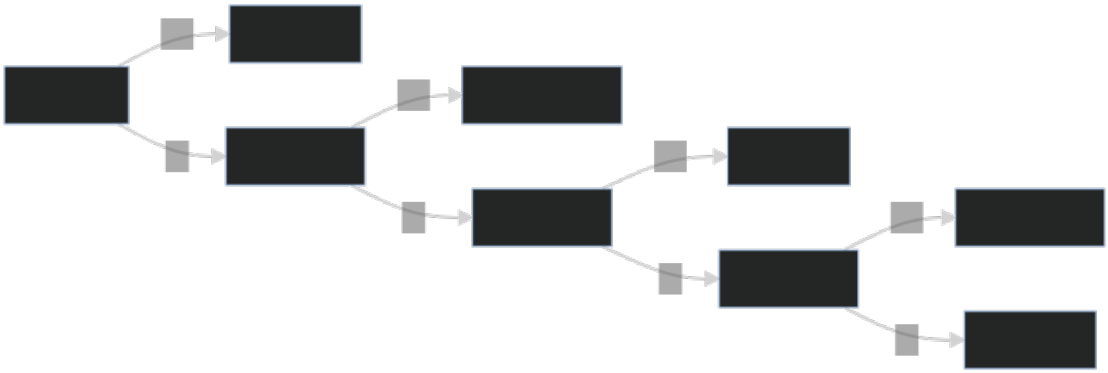
Tabelle 27.1.: Uhrzeiten und ihre Begrüssungsformen

Uhrzeit	Begrüssungsform
5 bis 11 Uhr	Guten Morgen
11 bis 18 Uhr	Guten Tag
18 bis 22 Uhr	Guten Abend

Uhrzeit	Begrüssungsform
22 bis 5 Uhr	Gute Nacht

Das Problem ist etwas schwierig zu Packen weil wir die “von - bis” Uhrzeiten in eine Ja/Nein Logik überführen müssen. Überlege dir zuerst eine Ausführungslogik und schreibe danach die `if`, `elif`und `else` Operationen. Im Dropdown unten ist unser Vorschlag.

Hinweis



```
def gruezi2(nachname, anrede, uhrzeit):
    if(uhrzeit < 5):
        was = "Gute Nacht"
    elif(uhrzeit < 11):
        was = "Guten Morgen"
    elif(uhrzeit < 18):
        was = "Guten Tag"
    elif(uhrzeit < 22):
        was = "Guten Abend"
    else:
        was = "Gute Nacht"

    return was + ", " + anrede + " "+ nachname
```

Teste die Funktion mit verschiedenen Optionen um sicherzustellen, dass sie richtig funktioniert.

27.3. Übung 7.2

Erweitere die Funktion `bmi` aus Kapitel 25.3 so, dass statt dem berechneten BMI der "Nutritional status" aufgrund der WHO Klassifizierung ausgegeben wird:

Tabelle 27.2.: WHO Nutritional status

BMI	Nutritional status
Below 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Pre-obesity
30.0–34.9	Obesity class I
35.0–39.9	Obesity class II
Above 40	Obesity class III

```
def bmi2(groesse_m, gewicht_kg):
    bmi = gewicht_kg / (groesse_m * groesse_m)
    if(bmi < 18.5):
        cat = "Underweight"
    elif(bmi <= 24.9):
        cat = "Normal weight"
    elif(bmi <= 29.9):
        cat = "Pre-obesity"
    elif(bmi <= 34.9):
        cat = "Obesity class I"
    elif(bmi <= 39.9):
        cat = "Obesity class II"
    else:
        cat = "Obesity class III"
    return "Who classification: " + cat
```

```
# Der Output sollte etwa folgedermassen aussehen
bmi2(1.7, 70)
```

```
'Who classification: Normal weight'
```

28. Zufallszahlen generieren

Im Block "*Datenqualität und Unsicherheit*" habt ihr euch bereits mit Zufallszahlen und Simulationen auseinandergesetzt. Programmiersprachen sind für eine solche Anwendung sehr gut geeignet, und deshalb werden wir in diesem Abschnitt eine Erweiterung zur Erstellung von Zufallszahlen kennenlernen. Diese Erweiterung lautet `random` und ist Teil der "Python Standard Library", was bedeutet das dieses Erweiterung bereits installiert ist, und wir sie nicht installieren müssen.

```
import random
```

```
random.seed(2)
```

Innerhalb vom `random` gibt es zahlreiche Funktionen um Zufallszahlen zu generieren, je nachdem was unsere Anforderungen an die Zufallszahl ist. Eine Anforderung könnte zum Beispiel sein, dass die Zahl innerhalb eines bestimmten Bereichs luegen soll (z.B. "*generiere eine Zufallszahl zwischen 1 und 9*"). Oder aber, dass sie eine ganze Zahl sein muss. Weiter könnte die Anforderung sein, dass sie aus einer bestimmten Verteilung kommen sollte, zum Beispiel einer Normalverteilung. In diesem letzten Fall müssen wir den Mittlwert sowie die Standartabweichung unserer Verteilung angeben.

Um eine ganzzahlige Zufallszahl zwischen 1 und 9 zu generieren, können wir die Funktion `randrange()` nutzen:

```
random.randrange(start = 1, stop = 10)
```

```
1
```

Wenn wir auf diese Weise mit `randrange()` immer wieder neue Zufallszahlen generieren fällt auf, dass die Verteilung der Zahlen ziemlich gleichmässig ist. Es ist also gleich wahrscheinlich eine 9 zu bekommen, wie eine 1 oder eine 5. Die Zahlen kommen also aus einer "uniformen" Verteilung. Um dies zu verdeutlichen generiere ich in den folgenden Codezeilen 500 Zufallszahlen zwischen 1 und 9 mit der Funktion `randrange` und visualisiere die Häufigkeit der einzelnen Zahlen in einem Histogramm.

```
# erstellt eine Liste von Zufallszahlen 1 - 9
# (lernen wir zu einem späteren Zeitpunkt)
a = [random.randrange(1, 10) for x in range(500)]

import pandas as pd

# visualisiert die zufällig generierten Zahl in Form
# eines Histogramms (lernen wir ebenfalls später)
pd.Series(a).plot(kind = "hist", bins = range(1, 11), edgecolor = "black", align = "left", x
```

```
ModuleNotFoundError: No module named 'pandas'
```

Die Funktion `randrange()` generiert nur ganzzahlige Zufallszahlen. Wenn wir aber eine Zufallszahl mit Nachkommastellen brauchen, verwenden wir die Funktion `uniform()`.

Um Zufallszahlen aus einer "Normalverteilung" zu erhalten, verwenden wir die Funktion `normalvariate`. Hier müssen wir den Mittelwert und die Standartabweichung dieser Verteilung angeben. Tatsächlich können wir bei dieser Variante keine Minimum- und Maximumwerte festlegen: Theoretisch könnte der Generator jeden erdenklichen Zahlenwert rausspucken, am wahrscheinlichsten ist jedoch eine Zahl nahe am angegebenen Mittelwert.

```
# mu = Mittelwert, sigma = Standartabweichung
random.normalvariate(mu = 5, sigma = 2)
```

```
2.5056884809480864
```

Wenn wir die obige Funktion 10'000x laufen lassen und uns das Histogramm der generierten Zahlen anschauen, dann zeichnet sich folgendes Bild ab.

```
b = [random.normalvariate(mu = 5, sigma = 2) for b in range(10000)]
pd.Series(b).plot(kind = "hist", bins = 30, edgecolor = "black")
```

```
NameError: name 'pd' is not defined
```

29. Übung 8

Nun wollen wir diesen Zufallszahlengenerator `random` nutzen um eine Funktion zu entwickeln, welche einen beliebigen Punkt (mit einer x-/y-Koordinate) zufällig in einem definierten Umkreis verschiebt. Unser Fernziel ist es, den simulierten Datensatz aus "Datenqualität und Unsicherheit" zu rekonstruieren (siehe unten). Der erste Schritt dorthin ist es, einen gemeldeten Punkt (rot in Abbildung 29.1) in einem definierten Umkreis zu verschieben.

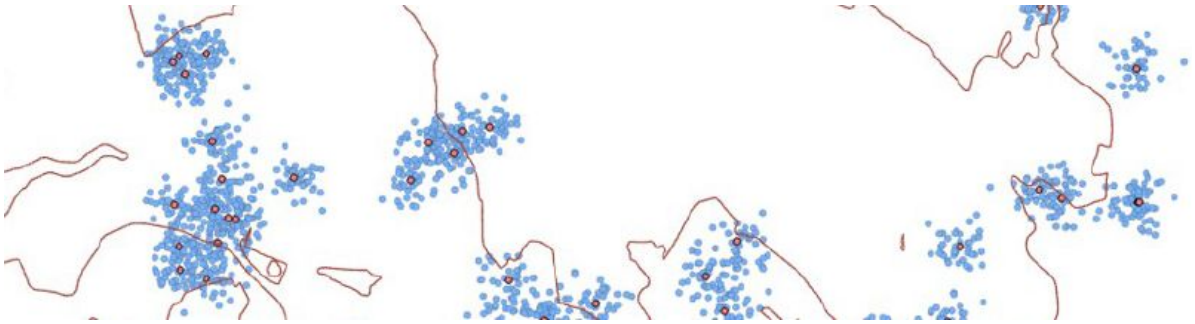


Abbildung 29.1.: Ausschnitt der simulierten Zeckenstiche. Der rote Punkt stellt jeweils den gemeldeten Zeckenstich dar, die blaue Punktwolke drum herum sind simulierte Punkte welche die Ungenauigkeit der Daten wiederspiegelt.

Das Ziel dieser Übung ist es also, dass wir eine Funktion entwickeln, die uns einen zufälligen Punkt in der Nähe eines Ursprungspunktes vorschlägt. Unser Vorgehen: Wir addieren jedem Koordinatenwert (`x`/`y`) des Ursprungspunktes einen Zufallswert, zum Beispiel zwischen -100 bis +100.

29.1. Übung 8.1

Bevor wir mit Koordinaten arbeiten wollt ihr euch zuerst mit dem Modul `random` vertraut machen. Importiere das Modul `random` und generiere eine Zufallszahl zwischen -100 und +100 aus einer uniformen Verteilung sowie aus einer Normalverteilung mit Mittelwert 100 und Standartabweichung 20.

```
import random

random.uniform(-100, 100)

random.normalvariate(100, 20)
```

29.2. Übung 8.2

Nun wollen wir uns den Koordinaten zuwenden. Erstelle als erstes zwei Dummykoordinaten `x_start` und `y_start` mit jeweils dem Wert `0`. Diese sollen als "Ursprungskoordinaten" dienen.

```
x_start = 0
y_start = 0
```

29.3. Übung 8.3

Generiere nun eine Zufallszahl, die aus einer Normalverteilung stammt und die *in etwa* zwischen -100 und +100 liegt. Weise diese Zahl der Variabel `x_offset` zu. Generiere danach eine zweite Zufallszahl (auf die gleiche Art) und weise diese `y_offset` zu.

> **Hinweis**
>
> Überlege dir, welcher *Mittelwert* Sinn macht um Werte zwischen -100 und +100 zu bekommen. Welche Zahl liegt zwischen -100 und +100?
> Überlege dir als nächstes, welche Standartabweichung sinnvoll wäre. Zur Erinnerung: Etwa 68% der Werte liegen inerhalb von +/- 1 Standartabweichung (SD), 95% innerhalb von +/- 2 SD, 99% innerhalb von 3 SD (siehe unten):

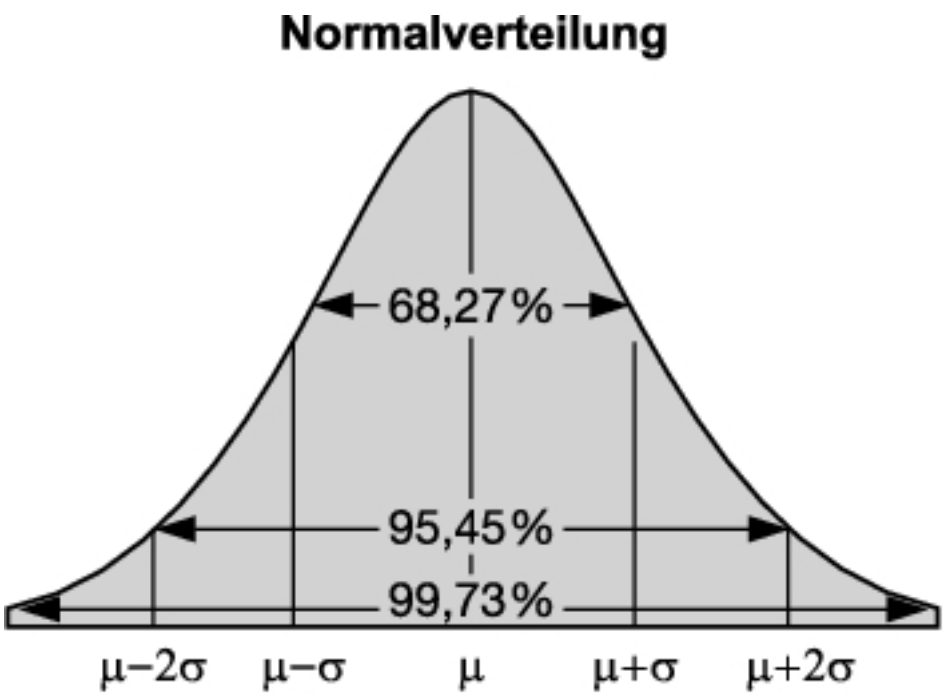


Abbildung 29.2.: Normalverteilung und die Anteile innerhalb von 1 Standartabweichung (Mittelwert μ minus Standartabweichung σ), 2 Standartabweichungen ($\mu - 2 \times \sigma$) und 2 Standartabweichungen ($\mu - 3 \times \sigma$). Quelle: cobocards

```
# Normalverteilte Werte mit Mittelwert 0 und Standartabweichung 100
# Achtung: bei dieser Standartabweichung sind ca 30% der Werte > 100!
x_offset = random.normalvariate(0, 100)
y_offset = random.normalvariate(0, 100)

x_offset
y_offset
```

29.4. Übung 8.4

Addiere nun die Zufallszahlen `x_offset` und `y_offset` **jeweils** zu den Dummykoordinaten `x_start` und `y_start` und weise diese neuen Koordinaten `x_neu` und `y_neu` zu. Die neuen Werte stellen die leicht verschobenen Ursprungskoordinaten dar. In meinem Fall sind diese um 10.2 Meter nach Osten (positiver Wert) bzw. 4.4 Meter nach Süden (negativer Wert) verschoben worden.

```
x_offset = 10.246170309600945
y_offset = -4.443904000288846

x_neu = x_start + x_offset
y_neu = y_start + y_offset
```

```
x_neu
```

```
10.246170309600945
```

```
y_neu
```

```
-4.443904000288846
```

```
import pandas as pd

mypoints = pd.DataFrame({
    "x": [x_start, x_neu],
    "y": [y_start, y_neu],
    "typ": ["start", "neu"]
    }
)

from matplotlib import pyplot as plt

mypoints.plot("x", "y", kind = "scatter", xlim = [-100, 100], ylim = [-100, 100], c = ["red"
```

```
ModuleNotFoundError: No module named 'pandas'
```

29.5. Übung 8.5

Nun haben wir das zufällige Verschieben eines Einzelpunktes am Beispiel einer Dummykoordinaten (0/0) durchgespielt. In der nächsten Aufgabe (Kapitel 30) werden wir *alle* unsere Zeckenstichkoordinaten auf diese Weise zufällig verschieben um einen simulierten Zeckenstichdatensatz ähnlich wie Abbildung 29.1 zu erhalten.

Dafür brauchen wir die eben erarbeiteten Einzelschritte als Funktion, um diese auf alle Zeckenstiche anwenden zu können.

Erstelle jetzt eine Funktion namens `offset_coordinate` welche als Input eine `x` *oder* `y`-Achsenwert annimmt und eine leicht verschobene Wert zurückgibt.** Integriere die Standartabweichung der Verteilung als optionalen Parameter mit dem Namen `distance` und einem Defaultwert von 100.

```
def offset_coordinate(old, distance = 100):
    new = old + random.normalvariate(0, distance)
    return new
```

```
# die Funktion sollte so funktionieren:
offset_coordinate(x_start)
```

```
-23.79232776816737
```

29.6. Übung 8.6

Nun ist es wichtig, dass wir unser Resultat visuell überprüfen. Im Beispiel unten wende ich die in der letzten Übung erstellte Funktion `offset_coordinate()` 1'000 mal auf die Dummykoordinate an. Nutze *deine* Funktion `offset_coordinate` um eine Visualisierung gemäss untenstehendem beispiel zu machen.

```
x_neu_list = [offset_coordinate(x_start) for i in range(1, 1000)]
y_neu_list = [offset_coordinate(y_start) for i in range(1, 1000)]

# Liste in eine Pandas DataFrame überführen
mysim = pd.DataFrame({"x" : x_neu_list, "y" : y_neu_list})

mysim.plot("x", "y", kind = "scatter")

from matplotlib import pyplot as plt
plt.axis("equal")
```

```
NameError: name 'pd' is not defined
```

30. Funktionen in *DataFrames*

In dieser Aufgabe haben wir das Ziel, die in der letzten Aufgabe (Kapitel 28) erstellte Funktion `offset_coordinate()` auf alle Zeckenstich-Koordinaten anwenden. Bildlich gesprochen: Wir nehmen unsere Zeckenstichdatensatz und schütteln ihn **einmal** durch. So erhalten wir einen Datensatz ähnlich wie in Abbildung 29.1 mit dem Unterschied, dass jede Zeckenstichmeldung nicht eine *Wolke* von simulierten Punkten enthält, sondern nur einen einzelnen Punkt.

Nutze hier die Datei *zeckenstiche.csv* von letzter Woche (siehe Tabelle 6.1). Erstelle ein neues Notebook und nutze nachstehenden Code um die nötigen Module und Functions zu haben:

```
import pandas as pd

def offset_coordinate(old, distance = 100):
    import random
    new = old + random.normalvariate(0,distance)

    return new

zeckenstiche = pd.read_csv("data/zeckenstiche.csv")
#           ---------------------  --------------------
#         < verwendet hier euren eigenen relativen pfad >
#           ----------------------------------------


zeckenstiche
```

```
ModuleNotFoundError: No module named 'pandas'
```

31. Übung 9

Für diese Übung brauchen wir

1. Die Python-Module `pandas` sowie `random`
2. Die Funktion `offset_coordinate` aus Kapitel 29.5
3. *zeckenstiche.csv* importiert als `pandas DataFrame` in der Variabel `zeckenstiche`

```
import pandas as pd
import random

def offset_coordinate(old, distance = 100):
    new = old + random.normalvariate(0, distance)
    return new

zeckenstiche = pd.read_csv("data/zeckenstiche.csv")
```

```
ModuleNotFoundError: No module named 'pandas'
```

31.1. Übung 9.1

Mache dich nochmals damit vertraut, einzelne Spalten zu selektieren. Schau dir Kapitel 15 nochmals an wenn du nicht mehr weisst wie das geht.

```
zeckenstiche["x"]
zeckenstiche["y"]
```

31.2. Übung 9.2

Mache dich nochmals damit vertraut, wie man neue Spalten erstellt. Schau dir Kapitel 15 nochmals an wenn du nicht mehr weisst wie das geht. Erstelle ein paar neue Spalten nach dem Beispiel unten um die Hangriffe zu üben. Lösche die Spalten im Anschluss wieder mit `del zeckenstiche['test1']` etc.

```
zeckenstiche["test1"] = "test1"

zeckenstiche["test2"] = 10

zeckenstiche["test3"] = range(10)
```

```
# zeckenstiche könnte danach folgendermassen aussehen:
zeckenstiche
```

```
NameError: name 'zeckenstiche' is not defined
```

```
# unnötigen Spalten wieder entfernen:
del zeckenstiche['test1']
del zeckenstiche['test2']
del zeckenstiche['test3']
```

```
NameError: name 'zeckenstiche' is not defined
```

31.3. Übung 9.3

`pandas` kennt eine ganze Familie von Methoden, um Spalten zu Manipulieren und Daten zu Aggregieren (`apply`, `map`, `mapapply`, `assign`). Es würde den Rahmen dieses Kurses sprengen, die alle im Detail durchzugehen, es lohnt sich aber sehr sich mit diesen auseinanderzusetzen wenn man sich näher mit Python befassen möchte.

Im unserem Fall brauchen wir lediglich die Methode `apply` um die Funktion `offset_coordinate()` auf die Zeckenstichkoordinaten anzuwenden. Dabei gehen wir wie folgt for:

```
zeckenstiche["x"].apply(offset_coordinate)
#\______1_______/ \_2_/\_______3_________/

# 1. Spalte selektieren (["x"])
# 2. Methode "apply" aufrufen
# 3. Function übergeben
```

Verwende dieses Schema um auch `offset_coordinate` auf die `y` Spalte anzuwenden und speichere den Output dieser beiden Operationen als neue Spalten `x_sim` sowie `y_sim`. Die *DataFrame* `zeckenstiche` sollte danach wie folgt aussehen:

```
zeckenstiche["x_sim"] = zeckenstiche["x"].apply(offset_coordinate)
zeckenstiche["y_sim"] = zeckenstiche["y"].apply(offset_coordinate)
```

```
zeckenstiche
```

```
NameError: name 'zeckenstiche' is not defined
```

31.4. Übung 9.4

In Kapitel 31.3 haben wir unsere Funktion `offset_coordinate` aufgerufen, ohne den Parameter `distance` zu spezifizieren. Dies war möglich, weil wir für `distance` einen Defaultwert festgelegt hatten (100 Meter). Wir können aber auch zusätzliche Parameter kommagetrennt nach der Funktion angeben. Dies sieht folgendermassen aus:

```
zeckenstiche["x"].apply(offset_coordinate, distance = 200)
```

Nutze diese Möglichkeit, um den Offset (`distance`) auf maximal 10 Meter zu reduzieren.

```
zeckenstiche["x_sim"] = zeckenstiche["x"].apply(offset_coordinate, distance = 10)
zeckenstiche["y_sim"] = zeckenstiche["y"].apply(offset_coordinate, distance = 10)
```

31.5. Übung 9.5

Um die Original `x`/`y`-Werte sowie die simulierten Daten im gleichen Plot darzustellen, wird folgendermassen vorgegangen: Der erste Datensatz wird mit `.plot()` visualisiert, wobei der Output einer Variabel (z.B. `basemap`) zugewiesen wird. Danach wird der zweite Datensatz ebenfalls mit `.plot()` visualisiert, wobei auf den ersten Plot via dem Argument `ax` verwiesen wird.

Bei den roten Punkten handelt es sich um die Original-Zeckenstichen, bei den blauen um die simulierten (leicht verschoben) Zeckenstiche. Visualisiere deine eigenen Zeckenstiche auf diese Weise.

```
from matplotlib import pyplot as plt

basemap = zeckenstiche.plot("x", "y", kind = "scatter", color = "red")
zeckenstiche.plot("x_sim", "y_sim", kind = "scatter", ax = basemap, color = "blue")

plt.axis("equal")
```

```
ModuleNotFoundError: No module named 'matplotlib'
```

32. Anhang

32.1. Anhang 1: Probleme mit Conda: Umlaute im Bentzernamen

Alle *conda environments* werden an einem zentralen Ort gespeichert. Typischerweise im folgenden Verzeichnis: `C:\Users\DEIN-WINDOWS-BENUTZERNAME\AppData\Local\ESRI\conda\envs\`. Wenn ihr Umlaute in eurem Windows Benutzernamen habt, findet `conda` eure *environment* möglicherweise nicht. Eine Möglichkeit, dieses Problem zu beheben, ist einen alternativen Speicherort für eure *conda environments* zu definieren. Dazu geht ihr wie folgt vor[1]:

1. Überlege dir eine Location, wo du deine conda nvironments abspeichern möchtest (z.B. `C:\conda-envs`). Erstelle diesen Ordnerpfad und prüfe ob du dort Schreibrechte hast indem du in diesem Orner `C:\conda-envs` eine Dateie erstellst. Wenn das klappt, hast du Schreibrechte.
2. Füge diesen Pfad deiner conda configuration hinzu:

   ```
   conda config --append envs_dirs C:\conda-envs
   ```

3. Prüfe ob der letzte Schritt funktioniert hat indem du folgendem Befehl laufen lässt und prüfst, ob `C:\conda-envs` vorkommt:

   ```
   conda config --show envs_dirs
   ```

4. Erstelle eine neue conda environment in dieser neu erstellten directory:

   ```
   conda create --prefix=C:\conda-envs\geopython
   ```

5. Prüfe, ob das geklappt hat, indem du die verfügbaren environments auflistest.

   ```
   conda info --envs
   ```

6. Wenn das geklappt hat, kannst du nun deine environment aktivieren

   ```
   conda activate geopython
   ```

[1] Aus: https://stackoverflow.com/a/67376348/4139249

32.2. Anhang 2: Conda und `arcpy`

Um die ganzen Befehle von ArcGIS direkt in Python ansteuern zu können, muss das package arcpy installiert werden. Bisher haben wir alle unsere Packages mit conda installiert, z.B:

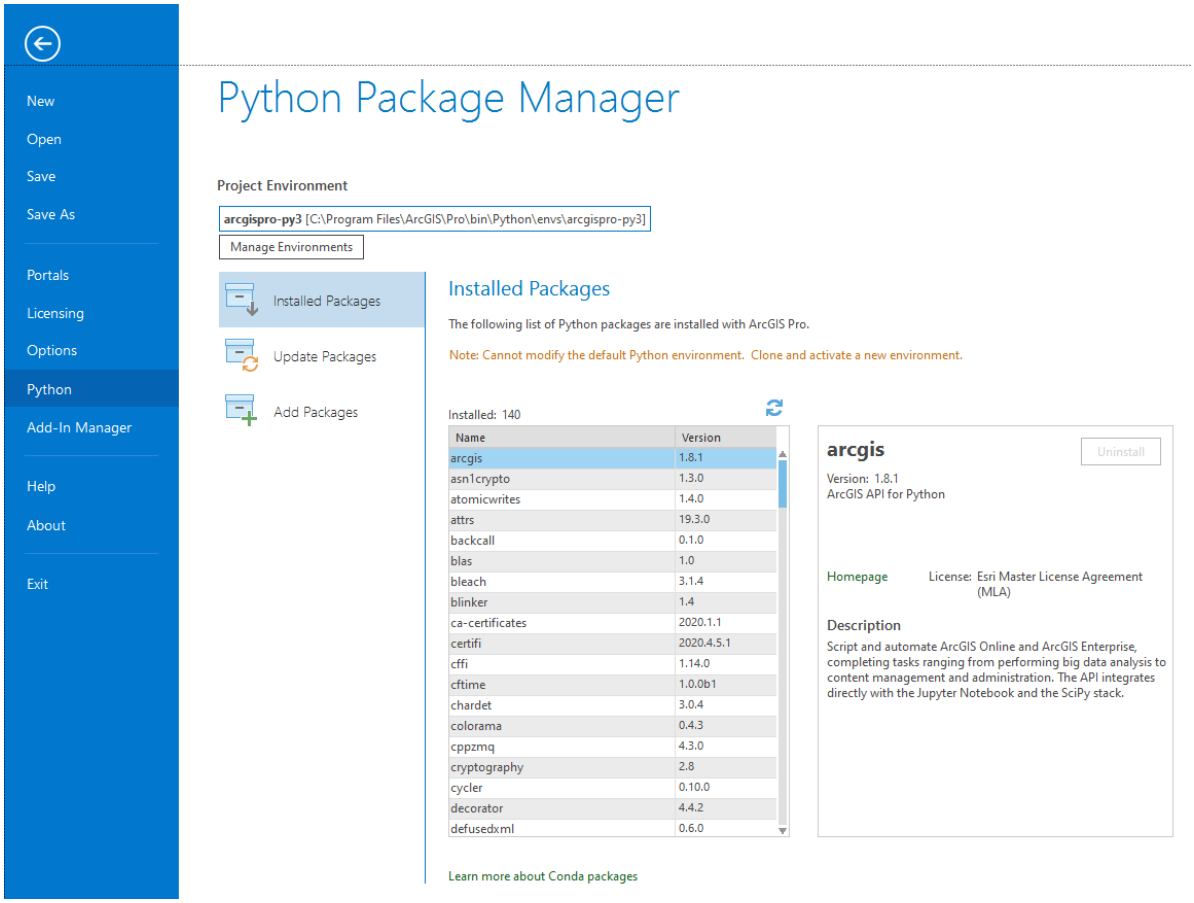
```
conda install -c conda-forge geopandas
```

Mit `arcpy` geht dies leider nicht, weil `arcpy` ein kostenpflichtiges Modul ist welches eine ArcGIS Lizenz benötigt. Glücklicherweise nutzt ArcGIS aber auch Conda Environments. Wir müssen also nur bewerkstelligen, das Juypter Lab die gleiche Conda environment verwendet wie ArcGIS. Dazu gehen wir wie folgt vor:

- Schritt 1: ArcGIS Python Umgebung Klonen
- Schritt 2: Die neue Environment aktivieren
- Schritt 3: weitere Module installieren

32.2.1. Schritt 1: ArcGIS Python Umgebung Klonen

Zuerst prüfen wir die Python Umgebung in ArcGIS. Diese findet man in ArcGIS unter Project > Python



Hier ist einerseits die Project Environement ersichtlich (1), andererseits steht aber auch, dass diese Environment “read only” ist (2). Das bedeutet, dass wir keine neuen module installieren können, wenn wir diese Environment benutzen. Wir folgen deshalb den Vorschlag “Clone and activate a new environment”. Dazu klicken wir auf “Manage Environment” (3). Übrigens: Das ArcGIS Conda benutzt sehen wir an (4).

Klicke hier auf “Clone Default” um die Umgebung zu kopieren. Das dauert eine Weile, danach kann man die neue Environment auswählen (Klick auf den Button “Active”). Notiert dir den Namen der neuen Environment, speichere das ArcGIS Projekt ab und starte das ArcGIS neu

32.2.2. Schritt 2: Die neue Environment aktivieren

Nun haben wir uns eine wunderschöne Python Umgebung parat gemacht und können diese jetzt in CMD aktivieren. Starte dazu Command Prompt / CMD und schaue dir die verfügbaren environments an:

```
conda env list
```

Bei mir sieht der output folgendermassen aus:

```
# conda environments:
#
arcgisonline             C:\Users\rata\AppData\Local\ESRI\conda\envs\arcgisonline
arcgisonline2            C:\Users\rata\AppData\Local\ESRI\conda\envs\arcgisonline2
arcgispro-py3-clone      C:\Users\rata\AppData\Local\ESRI\conda\envs\arcgispro-py3-clone
arcgispro-py3-clone1     C:\Users\rata\AppData\Local\ESRI\conda\envs\arcgispro-py3-clone1
cameratraps-detector     C:\Users\rata\AppData\Local\ESRI\conda\envs\cameratraps-detector
geopython                C:\Users\rata\AppData\Local\ESRI\conda\envs\geopython
test                     C:\Users\rata\AppData\Local\ESRI\conda\envs\test
arcgispro-py3         *  C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3
root                     C:\Program Files\ArcGIS\Pro\bin\Python
```

Aktiviere nun die eben erstelle environment mit folgendem code (ersetzte `name-der-environment` mit dem tatsächlichen Namen deiner neuen environment aus dem letzten Schritt.

```
activate name-der-environment
```

32.2.3. Schritt 3: weitere Module installieren

Glücklicherweise ist `jupyterlab` bereits in der `arcgis` environment installiert, dies können wir mit folgendem code überprüfen:

```
conda list
```

```
# packages in environment at C:\Users\rata\AppData\Local\ESRI\conda\envs\arcgispro-py3-clone1
#
affine                    2.3.0                      py_0    anaconda
arcgis                    1.8.2                 py36_1275    esri
arcgispro                 2.6                           0    esri
.
.
.
jupyterlab                2.2.9                      py_0    conda-forge
jupyterlab_pygments       0.1.2                pyh9f0ad1d_0    conda-forge
jupyterlab_server         1.2.0                      py_0    conda-forge
.
.
.
```

Wenn jetzt aber noch module fehlen (wie z.B. `geopandas`), dann können wir diese wie gewohnt installieren.

32.2.4. Schritt 4: `arcpy` verwenden

Wenn alles gewünschten Module installiert sind können wir nun JupyterLab starten.

Sobald JupyterLab gestartet ist, können wir innerhalb einer cell das Modul `arcpy` mit `import arcpy` importieren und nun auch verwenden. Zum Beispiel folgendermassen:

```
import arcpy
from arcpy import env

# Set environment settings
env.workspace = "C:/data/Habitat_Analysis.gdb"

# Select suitable vegetation patches from all vegetation
veg = "vegtype"
suitableVeg = "C:/output/Output.gdb/suitable_vegetation"
whereClause = "HABITAT = 1"
arcpy.Select_analysis(veg, suitableVeg, whereClause)
```

Der Syntax ist auf jeder jeweiligen Tool Beschreibung gut dokumentiert (Abschnitt "Code Sample", z.B. hier)

32.3. Anhang 3: JupyterLab mit `R`

Einige von euch kennen JupyterLab bereits aus dem Statistik Unterricht. Dort habt ihr JupyterLab mit `R` benutzt, dabei habt ihr aber nicht auf eurem lokalen PC, sondern auf einer *gehosteten* Version von JupyterLab gearbeitet, also auf einem Server. Dieser Server kann jeder Zeit wieder Offline geschaltet werden oder ihr könnt euren Zugriff darauf verlieren. Wenn ihr JupyterLab mit `R` lokal nutzen wollt, geht ihr wie folgt vor (Anleitung in Anlehnung an diesen Blogpost):

1. In eurem Python Command Prompt erstellt ihr eine neue `conda` environment, aktiviert diese und installiert `juypterlab` sowie `r-ikernel` (letzteres aus dem channel `r`, nicht wie gewohnt `conda-forge`).

   ```
   conda create --name r-env
   conda activate r-env
   conda install -c conda-forge jupyterlab
   conda install -c r r-irkernel
   ```

2. Nun ist `R` in eurer conda environment aktiviert. Nun könnt ihr mit folgenden Befehlen das Package `IRkernel` installieren sowie den Kernel für JuypterLab verfügbar machen.

```
Rscript -e "install.packages('IRkernel', repos = 'https://cloud.r-project.org')"
Rscript -e "IRkernel::installspec(user = FALSE)"
```

3. Optional: Wenn ihr ein paar nützliche shortcuts für `R` haben wollt, könnt ihr die extension `techrah/text-shortcuts` mit folgender Zeile installieren.

```
jupyter labextension install @techrah/text-shortcuts
```

4. Nun könnt ihr JupyterLab starten und sollet nun einen *R-Kernel* zur Verfügung haben:

```
jupyter lab
```

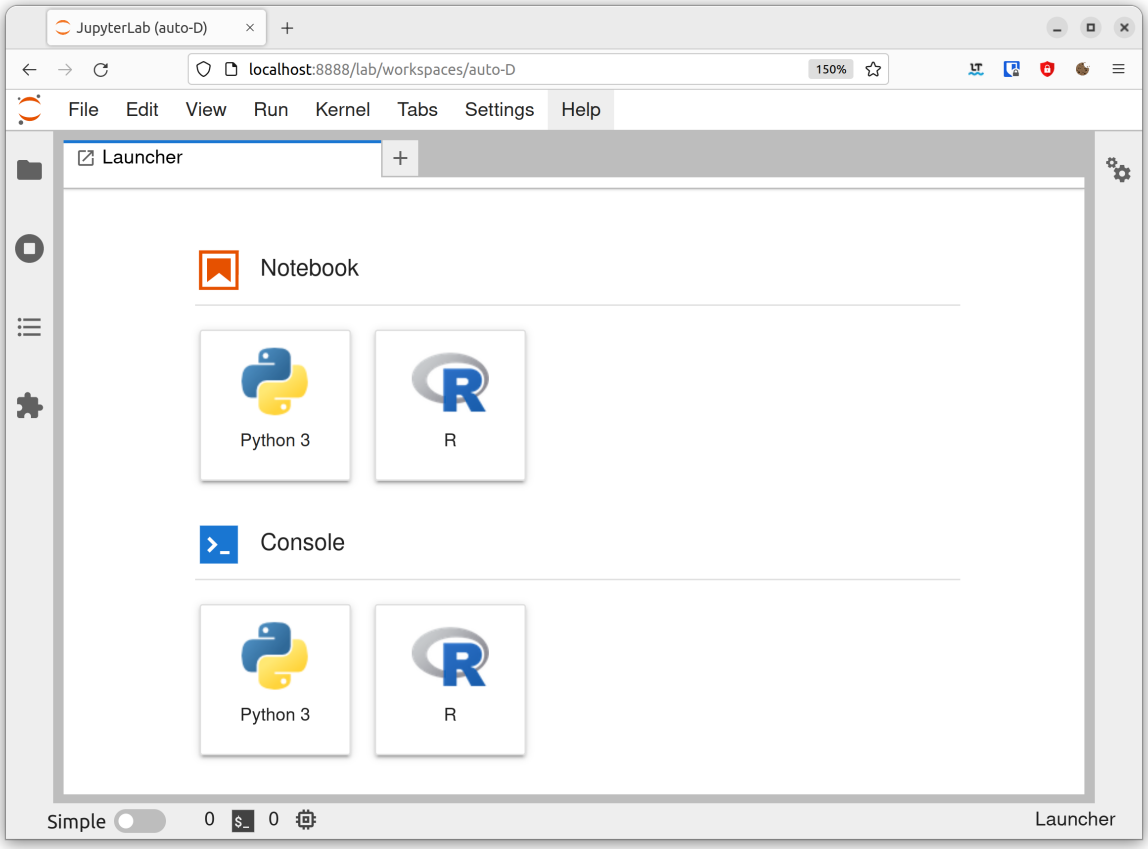


Abbildung 32.1.: Nach den oben beschriebnen Schritten sollte nun ein `R` Kernel zur Verfügung stehen

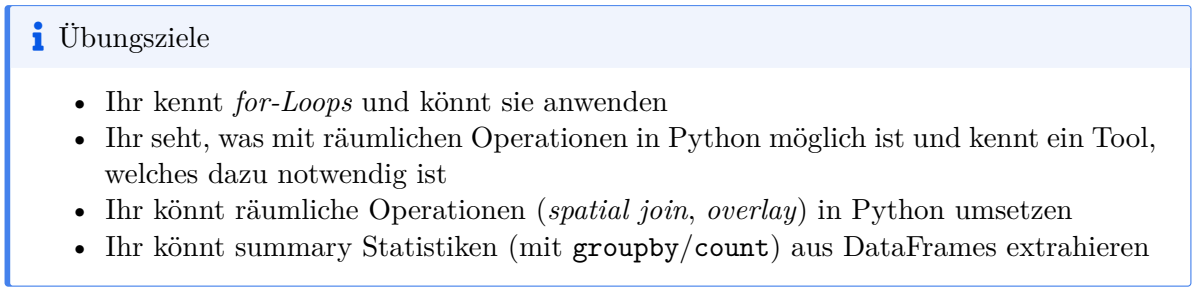
Teil IV.

Programmieren III

Heute widmen wir uns zwei grossen Themen bevor wir uns an die Umsetzung der Monte Carlo Simulation wenden.

Im ersten Thema geht es um *for Loops*, mit denen wir eine Aufgabe mehrfach ("Iterativ") durchzuführen können. Ein *for Loop* ist ein sehr wichtiges und hilfreiches Werkzeug das man unbedingt kennen sollte. Wir werden *for loops* brauchen um Zeckenstiche zu simulieren, indem wir die bestehenden Meldungen x-fach zufällig verschieben. Das Thema *for loops* behandeln wir in Kapitel 33, Kapitel 35 und Kapitel 37.

Beim zweiten Thema geht es um GIS. Wir haben bisher mit Daten gearbeitet die eine Rauminformation beinhalten, die Zeckenstiche haben x/y-Koordinaten. Diese Rauminformation haben wir aber nicht als solche behandelt, wir haben die Koordinaten als Zahlen eingelesen und uns nicht gross darum geschert, dass es sich dabei um spezifische Punkte in der Schweiz handelt. Heute werden wir dies ändern müssen da uns eine räumliche Abfrage bevorsteht: Wir wollen nämlich wissen, welche Zeckenstiche sich im Wald befinden. Dadurch lernen wir eine neue Erweiterung kennen und sehen, was für räumliche Operationen in Python möglich sind. Das Thema behandeln wir in Kapitel 39, Kapitel 40 und Kapitel 42.



Übungsziele

- Ihr kennt *for-Loops* und könnt sie anwenden
- Ihr seht, was mit räumlichen Operationen in Python möglich ist und kennt ein Tool, welches dazu notwendig ist
- Ihr könnt räumliche Operationen (*spatial join*, *overlay*) in Python umsetzen
- Ihr könnt summary Statistiken (mit `groupby`/`count`) aus DataFrames extrahieren

33. Einführung in *for loop*

33.1. Die Grundform

Nirgends ist der Aspekt der Automatisierung so sichtbar wie in *for loops*. Loops sind "Schleifen" wo eine Aufgabe beliebig lange wiederholt wird. Auch *for loops* sind im Grunde genommen simple. Auf den ersten Blick sieht eine *for loop* aus wie eine *Function* definition (siehe Kapitel 22 und Kapitel 24). Im folgenden Beispiel seht ihr ein minimales Beispiel einer *for loop*.

```
for platzhalter in [0,1,2]:
    print("Iteration",platzhalter)
```

```
Iteration 0
Iteration 1
Iteration 2
```

- `for` legt fest, dass eine For-Loop beginnt
- Nach `for` kommt eine Platzhalter-Variabel, die ihr beliebig benennen könnt. Im obigen Beispiel lautet diese `platzhalter`
- Nach dem Platzhalter kommt der Begriff `in`. Dieser Begriff kommt zwingend nach dem Platzhalter.
- Nach `in` wird der "Iterator" festgelegt, also worüber der For-Loop iterieren soll (hier: über eine `List` mit den Werten `[0,1,2]`).
- Danach kommt ein Doppelpunkt `:` der zeigt: "Nun legen wir gleich fest was im For-Loop passieren soll" (ähnlich wie in einer *Function*)
- Auf einer neuen Zeile wird eingerückt festgelegt, was in der *For-Loop* passieren soll. Dieser Teil kann beliebig lange sein, ein *for loop* ist dann fertig, wenn man nicht mehr eingerückt wird. In unserem Fall wird mit `print`[1] etwas in die Konsole ausgegeben.
- Achtung: `return()` gibt's in For-Loops nicht!

[1] Mit `print` können wir Variabeln in die Konsole "ausdrucken" lassen. Innerhalb von `print` können dazu verschiedene Variablen kommagetrennt aufgeführt werden, ohne sie mit `+` verbinden zu müssen wie damals in Kapitel 22.

33.2. Der Iterator

Im obigen Beispiel haben wir über eine *List* iteriert, wir haben also eine Liste als Iterator verwendet. Es gibt aber noch andere "Dinge", über die wir iterieren können. Angenommen wir wollen den gleichen *for loop* mit den Zahlen von 0 bis 100 oder 100 bis 1'000 durchführen. Es wäre ganz schön mühsam, alle Zahlen von 0 bis 100 manuell in einer Liste zu erfassen. Zu diesem Zweck können wir die Funktion `range` verwenden. Mit `range(3)` erstellen wir einen Iterator mit den Werten 0, 1 und 2. Mit `range(100,1001)` erhalten wir die Werte von 100 bis 1'000.

Der gleiche *loop* wie oben lautet mit `range` folgendermassen:

```
for platzhalter in range(3):
    print("Iteration",platzhalter)
```

```
Iteration 0
Iteration 1
Iteration 2
```

33.3. Der Platzhalter

Die Platzhaltervariabel liegt immer zwischen `for` und `in`, den Namen dieser Variabel könnt ihr frei wählen. Ich habe sie im obigen Beispiel `platzhalter` genannt. Speziell an dieser Variabel ist, dass sie während der Dauer des *Loops* ihren Wert verändert. Mehr dazu in Kapitel 35.

34. Übung 10

34.1. Übung 10.1

Kopiere den folgenden *for Loop* und lasse ihn bei dir laufen. Spiele mit den Werten rum, um ein Gefühl für *For Loops* zu bekommen: Ergänze die Liste mit weiteren Zahlen, verändere den Namen der Platzhaltervariabel und verändere den Text, der in `print` ausgegeben wird.

```
for platzhalter in [0,1,2,5,10]:
    print("Iteration",platzhalter)
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 5
Iteration 10
```

34.2. Übung 10.2

Konstruiere eine Liste bestehend aus 3 Namen und nenne diese Liste `namen`. Erstelle danach einen *for loop*, mit welcher jede Person in der Liste gegrüsst wird. Nutze dafür `print`.

```
namen = ["Il Buono", "Il Brutto", "Il Cattivo"]

for name in namen:
    print("Ciao ",name)
```

Der Output könnte etwa so aussehen:

```
for name in namen:
    print("Ciao ",name)
```

```
Ciao  Il Buono
Ciao  Il Brutto
Ciao  Il Cattivo
```

34.3. Übung 10.3

Kopiere den folgenden *For Loop* und spiele hier mit den Werten herum. Verändere den *For Loop* so, dass er über die Werte von -5 bis +5 iteriert.

```
for nonsense in range(3, 5):
    print("Iteration",nonsense)
```

```
Iteration 3
Iteration 4
```

```
# Iteriert von -5 bis +5
for platzhalter in range(-5,6):
    print("Iteration",platzhalter)
```

34.4. Übung 10.4

Bis jetzt haben unsere *Loops* nicht viel Arbeiten müssen. Erstelle nun einen *For Loop*, welcher für die Werte -5 bis +5 folgendes ausgibt:

```
Das Quadrat von -5 ist 25
Das Quadrat von -4 ist 16
...
```

```
for platzhalter in range(-5,6):
    print("Das Quadrat von",platzhalter, "ist",platzhalter*platzhalter)
```

34.5. Übung 10.5

Bisher haben wir die Platzhaltervariabel immer in unserem *Loop* wiederverwendet. Das müssen wir aber gar nicht, wir können den *for loop* einfach nutzen um etwas x mal zu wiederholen. Erstellen einen *for loop* der folgende beiden Sätze 5x wiederholt:

```
Who likes to party?
We like to party!
Who likes to party?
....
```

> **Hinweis**
>
> Nutze dafür zwei verschiedene `print` Befehle auf zwei Zeilen.

```
for i in range(5):
    print("Who likes to party?")
    print("We like to party!")
```

35. Basic *for loop*

Bis jetzt haben wir lediglich Sachen in die Konsole herausgeben lassen, doch wie schon bei Functions ist der Zweck einer *for loop* meist, dass nach Durchführung etwas davon zurückbleibt. Aber `return()` gibt es wie bereits erwähnt bei *for loops* nicht. Nehmen wir folgendes Beispiel:

```
for rolle in ["bitch","lover","child","mother","sinner","saint"]:
    liedzeile = "I'm a "+ rolle
    print(liedzeile)
```

```
I'm a bitch
I'm a lover
I'm a child
I'm a mother
I'm a sinner
I'm a saint
```

Der Output von dieser For-Loop sind zwar sechs Liederzeilen, wenn wir die Variabel `liedzeile` aber jetzt anschauen ist dort nur das Resultat aus der letzten Durchführung gespeichert. Das gleiche gilt auch für die Variabel `rolle`.

```
liedzeile
```

```
"I'm a saint"
```

```
rolle
```

```
'saint'
```

Das verrät uns etwas über die Funktionsweise des *for loops*: Bei jedem Durchgang werden die Variablen immer wieder überschrieben. Wenn wir also den Output des ganzen For-Loops abspeichern wollen, müssen wir dies etwas vorbereiten.

Dafür erstellen wir unmittelbar for dem *for loops* einen leeren Behälter, zum Beispiel eine leere Liste:

```
refrain = []
```

Nun können wir innerhalb des *Loops* `append()` nutzen, um den Output von einem Durchgang dieser Liste hinzuzufügen.

```
for rolle in ["bitch","lover","child","mother","sinner","saint"]:
    liedzeile = "I'm a "+ rolle
    refrain.append(liedzeile)
```

In unserer Liste `refrain` ist nun der Wert *jeder* Iteration gespeichert.

```
refrain
```

```
["I'm a bitch",
 "I'm a lover",
 "I'm a child",
 "I'm a mother",
 "I'm a sinner",
 "I'm a saint"]
```

36. Übung 11

36.1. Übung 11.1

Nehmen wir nochmals das Beispiel aus Kapitel 34.2. Erstelle nochmal ein Loop, wo drei Personen aus einer Liste gegrüsst werden. Diesmal sollen aber die drei Grüsse in einer Liste (z.B. `mylist`) gespeichert werden.

```
mylist = []

for name in ["Il Buono", "Il Brutto", "Il Cattivo"]:
    mylist.append("Ciao "+name)
```

```
# Das Resultat sieht dann so aus:
mylist
```

```
['Ciao Il Buono', 'Ciao Il Brutto', 'Ciao Il Cattivo']
```

36.2. Übung 11.2

Der im Beispiel verwendete Refrain aus dem Lied "Bitch" von Meredith Brooks besteht bis auf zwei Zeilen aus Wiederholungen. Versuche mit **zwei verschiedenen**, aneinander gereihten *for loops* den ganzen Refrain in einer Liste zu speichern. Die beiden Teile die vom Muster Abweichen ("*I do not feel ashamed*" und "*You know you wouldn't want it any other way*") kannst du auch ausserhalb der Loops in die Listen einfügen (`append`).

```
refrain = []

for rolle in ["bitch","lover","child","mother","sinner","saint"]:
    liedzeile = "I'm a "+ rolle
    refrain.append(liedzeile)
refrain.append("I do not feel ashamed")
for rolle in ["your hell","your dream","nothing in between"]:
    liedzeile = "I'm "+ rolle
```

```
    refrain.append(liedzeile)
refrain.append("You know you wouldn't want it any other way")
```

```
# Das Resultat sieht dann so aus:
refrain
```

```
["I'm a bitch",
 "I'm a lover",
 "I'm a child",
 "I'm a mother",
 "I'm a sinner",
 "I'm a saint",
 'I do not feel ashamed',
 "I'm your hell",
 "I'm your dream",
 "I'm nothing in between",
 "You know you wouldn't want it any other way"]
```

37. Advanced *for loops*

In diesem Kapitel kommen noch zwei Aspekte von *for loops*, die als "*Advanced*" eingestuft werden können aber in der Praxis sehr nützlich sind. Dabei geht es einerseits um verschachtelte *for loops* und zum andere um eine verkürzte Schreibweise von *for loops*.

37.1. Verschachtelte *for loops*

Wir können verschiedene *for loops* auch ineinander verschachteln (englisch: *nested loops*). Das ist vor allem dann nützlich, wenn alle Kombinationen aus zwei Datensätzen miteinander verrechnet werden müssen. Angenommen du willst die drei Mitglieder deiner Band (bestehend aus *Il Buono*, *Il Brutto*, *Il Cattivo*) deinen Eltern vorstellen und auch umgekehrt deine Eltern deiner Band vorstellen. Für so was eignen sich zwei verschachtelte *for Loops* hervorragend:

> **Hinweis**
>
> Als Platzhaltervariabel nutze ich wenn immer möglich das Singulär und für den Iterator das Plural von dem Objekt, über das ich iteriere. `for bandmitglied in band`, `for vogel in voegel` usw, dies hilft mir den Überblick im *loop* zu bewahren.

```
eltern = ["Papa", "Mama"]
band = ["Il Buono", "Il Brutto", "Il Cattivo"]

for bandmitglied in band:
    for elternteil in eltern:
        print(elternteil, "das ist",bandmitglied)
        print(bandmitglied, "das ist",elternteil)
        print("---")
```

```
Papa das ist Il Buono
Il Buono das ist Papa
---
Mama das ist Il Buono
Il Buono das ist Mama
---
```

```
Papa das ist Il Brutto
Il Brutto das ist Papa
---
Mama das ist Il Brutto
Il Brutto das ist Mama
---
Papa das ist Il Cattivo
Il Cattivo das ist Papa
---
Mama das ist Il Cattivo
Il Cattivo das ist Mama
---
```

> **Hinweis**
>
> Ein anderes Beispiel: In Kapitel 34.5 haben wir die beiden Zeilen `"Who likes to party?"`, "We like to party?" 5x wiederholt. Dabei werden die Wörter 'to party für jede Wiederholung zwei mal wiederholt:
>
> ```
> Who likes to party?
> We like to party!
> Who likes to party?
> We like to party!
> ...
> ```
>
> Dies kann man in zwei verschachtelte *For Loops* umschreiben:
>
> ```
> for i in range(5):
>     inner = ["Who likes", "We like"]
>     for j in inner:
>         print(j+" to party")
>
> Who likes to party
> We like to party
> Who likes to party
> We like to party
> ```

37.2. Verkürzte Schreibweise

Es ist äusserst häufig der Fall, dass wir den Output aus einem Loop in einer Liste abspeichern wollen. Wie das geht haben wir ja bereits in Kapitel 35 gelernt:

```
rollen = ["bitch","lover","child","mother","sinner","saint"]

refrain = []
for rolle in rollen:
    liedzeile = "I'm a "+ rolle
    refrain.append(liedzeile)
```

Nur ist das ein *bisschen* umständlich, weil wir dafür viele Zeilen Code brauchen, um etwas eigentlich ganz simples zu bewerkstelligen. Es gibt deshalb dafür auch eine verkürzte Schreibweise, welche ich in der letzten Woche bereits einmal verwendet habe (siehe Kapitel 29.6). Der obige Loop hat in der verkürzten Schreibweise die folgende Form:

```
refrain = ["I'm a "+ rolle for rolle in rollen]
```

Diese verkürzte Schreibweise heisst in Python *list comprehension* und sie ist äusserst praktisch, wenn man sie beherrscht. Das Beherrschen ist aber nicht zentral, es reicht schon wenn ihr eine solche Schreibweise wieder erkennt und richtig interpretieren könnt (im Sinne von "*Aha, hier wird also in einem Loop eine Liste erstellt*"). In der folgenden Darstellung seht ihr farblich, welche Elemente sich in der verkürzten Schreibweise wo wiederfinden und welche Elemente gar nicht wiederverwendet werden.

```
rollen = ["bitch","lover","child","mother","sinner","saint"]
```

Herkömmlicher For-Loop:

Verkürzte Schreibweise:

38. Übung 12

38.1. Übung 12.1

Erstelle zwei Listen bestehend aus 3 Hundenamen (`hunde`) und 3 Katzennamen (`katzen`). Erstelle einen verschachtelten *For Loop*, wo jeder Hund jede Katze beisst und jede Katze jeden Hund kratzt.

```
Bruno beisst Greta und Greta kratzt  Bruno
Berta beisst Greta und Greta kratzt  Berta
Helmi beisst Greta und Greta kratzt  Helmi
....
```

```
hunde = ["Bruno", "Berta","Helmi"]
katzen = ["Greta", "Xavier", "Zachy"]

for katze in katzen:
    for hund in hunde:
        print(hund, "beisst", katze+" und "+katze, "kratzt ", hund)
```

38.2. Übung 12.2

Erstelle einen verschachtelten Loop, wo alle Kombinationen von 0 bis 9 miteinander addiert werden.

```
addition = []

werte = range(10)

for i in werte:
    for j in werte:
        resultat = i+j
        addition.append(resultat)
```

38.3. Übung 12.3

Nutze die Funktion `offset_coordinate` (Lösung aus Übung 8.5) um einen Punkt in einem Koordinatensystem zu verschieben. Diesmal soll der Punkt aber nicht nur 1x, sondern 100x verschoben werden (100 Simulationen).

```
import random

def offset_coordinate(old, distance = 100):
    new = old + random.normalvariate(0, distance)
    return new

x_start = 0
y_start = 0

x_random = []
y_random = []
for i in range(100):
    x_new = offset_coordinate(x_start)
    y_new = offset_coordinate(y_start)

    x_random.append(x_new)
    y_random.append(y_new)
```

38.4. Übung 12.4 (fakultativ)

Versuche die Monte Carlo Simulation für die Annäherung an Pi (aus der ersten Übung "Datenqualität und Unsicherheit) mit einer Funktion und einem For Loop zu lösen.

Zur Erinnerung, die Vorgehensweise für die Annäherung an Pi geht folgendermassen:

1. Zufallskoordinaten (`x`, `y`) zwischen 0 und 1 erstellen
2. Distanz zum Ursprung (0) mit der Formel $\sqrt{(x^2 + y^2)}$ berechnen
3. Bestimmen ob sich der Punkt innerhalb des Kreisviertels befindet ($d < 1$)
4. Schritte 1 & 2 mehrfach wiederholen
5. Anteil der Punkte *innerhalb* des Kreisviertels mit 4 Multiplizieren

Tipps:

- Für die Erstellung der Zufallspunkte brauchst du die Funktion `random()` aus dem modul `random`

- Schritte 1 - 3 werden am sinnvollsten ein eine Funktion verpakt, welche keine Argumente benötigt
- Schritt 4 löst du am besten mit einer `For loop` mit `range(100)` (für 100 Wiederholungen)

```
import random

# erstelle eine Funktion, die zwei Zufallszahlen zwischen 0 und 1 generiert,
# die Distanz zum Ursprung (0,0) berechnet und True retourniert, wenn der Wert ausserhalb des
def get_pi():
    x = random.random()
    y = random.random()
    pythagoras = (x**2+y**2)**0.5
    ausserhalb = pythagoras > 1
    return ausserhalb

get_pi()

# die Funktion 100x wiederholen und die Anzahl Werte > 1 zählen
res = [get_pi() for x in range(100)]

# Anteil der Werte > 1 berechnen und mit 4 multiplizieren
(100-sum(res))/100*4
```

39. Input: GIS in Python

Manche mag das jetzt überraschen, andere haben es vielleicht schon gemerkt: Mit **GIS** hatte unseres bisheriges Wirken in Python eigentlich wenig zu tun. Unsere Zeckenstiche haben zwar x/y-Koordinaten, aber diese haben wir bisher gleich behandlet wie alle anderen Spalten.

Anders gesagt: *Wir* wissen ja, dass mit den Spalten `x` und `y` Koordinaten in der Schweiz gemeint sind, Python hingegen wusste das bisher (noch) nicht. Der konkrete Raumbezug fehlt also noch, und das wird irgendwann problematisch: Denn bald wollen wir für jeden simulierten Zeckenstich ermitteln, ob er sich im Wald befindet oder nicht. Das ist eine explizit räumliche Abfrage, welche nur mit explizit räumlichen Geodaten beantwortet werden kann.

Was wir später mit den simulierten Zeckenstiche machen wollen, spielen wir an dieser Stelle mit den original Zeckenstichmeldungen (`zeckenstiche.csv`, siehe Tabelle 6.1) durch.

```
# Unsere Zeckenstiche hatten wir auf folgende Weise importiert:

import pandas as pd
zeckenstiche = pd.read_csv("data/zeckenstiche.csv")
zeckenstiche
```

```
ModuleNotFoundError: No module named 'pandas'
```

39.1. *DataFrames > GeoDataFrames*

Glücklicherweise können wir unsere Zeckenstich-*Dataframe* mit nur einem Zusatzmodul und wenigen Zeilen code in eine **räumliche** *DataFrame* konvertieren. Mit dem Modul ***geo****pandas* erstellen wir aus unserer *pandas DataFrame* eine ***geo****pandas* ***Geo****DataFrame*. Mit dieser Erweiterung erhält die *DataFrame*:

1. **geometry**: eine Zusatzspalte `geometry` mit der Geometrie als räumliches Objekt und
2. **crs**: ein Attribut `crs` welches das Koordinatenbezugssystem der Geometriespalte enthält.

Beide Bestandteile müssen wir bei der Erstellung der *GeoDataFrame* festlegen. Nachstehend seht ihr, wie ihr dies bei den Zeckenstichen machen könnt, was diese Bestandteile genau bedeuten seht ihr in Kapitel 39.2 (**geometry**) und Kapitel 39.3 (**crs**).

```
import geopandas as gpd

geom = gpd.points_from_xy(
  zeckenstiche['x'],    #
  zeckenstiche['y']     # → die Geometrie Spalte
  )

zeckenstiche_gpd = gpd.GeoDataFrame(
    zeckenstiche,    # die DataFrame ("Attributtabelle")
    geometry = geom, #
    crs = 2056       # das Koordinatenbezugssystem (EPSG Code)
    )
```

```
ModuleNotFoundError: No module named 'geopandas'
```

Nun sehen wir, dass die neue `geometry` Spalte die Punkt-Geometrie enthält.

```
zeckenstiche_gpd
```

```
NameError: name 'zeckenstiche_gpd' is not defined
```

Zudem hat die *GeoDataFrame* nun ein Attribut `crs`, wo das Koordinatenbezugssystem gespeichert ist.

```
zeckenstiche_gpd.crs.name
```

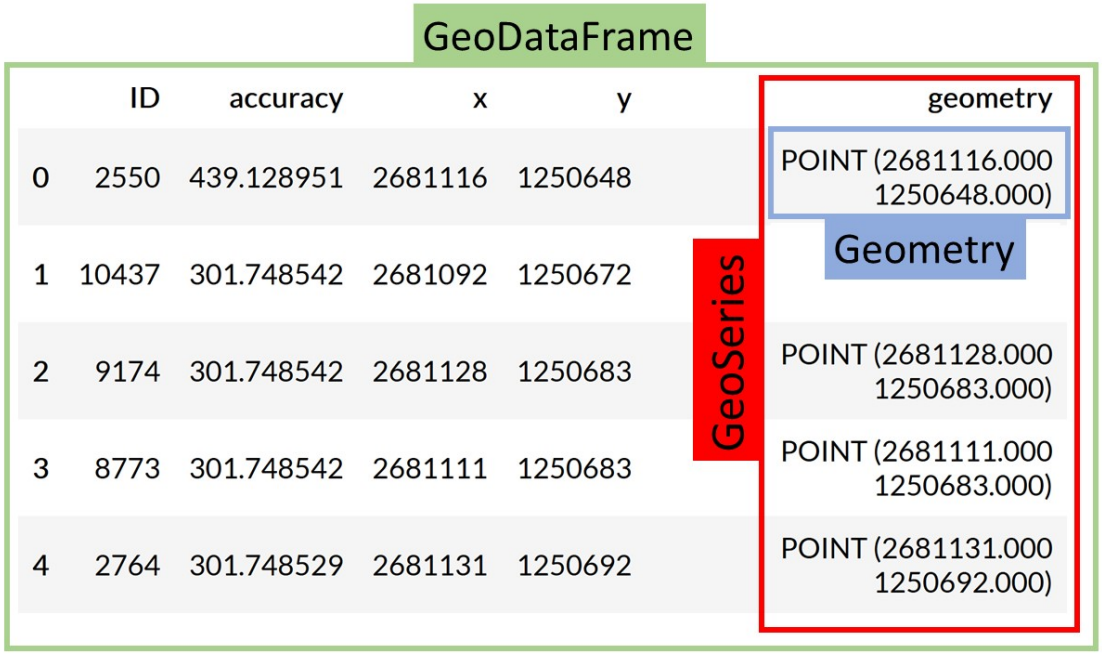
```
NameError: name 'zeckenstiche_gpd' is not defined
```

39.2. Aufbau von GeoDataFrames

Mit `geometry =` haben wir die Geometriespalte der *GeoDataFrame* festgelegt. Bei unseren Zeckenstichdaten ist die Geometrie sehr simpel: Sie besteht aus Punkt-Objekte die sich wiederum aus den x- und y-Koordinaten zusammensetzen. Um x/y-Koordinaten in Punktgeometrien zu konvertieren gibt es in Geopandas die Funktion `points_from_xy`. Diese verwende ich im obigen Code um die Punkt-Objekte zu erstellen.

An dieser Stelle ist es wichtig, die drei Hierarchiestufen von GeoDataFrames zu kären. Von "unten" nach "oben" erläutert gibt es folgende Stufen:

- **Geometry**: Einzelne Objekte [1] der folgenden Typen
  - Points / Multipoint [2]
  - Linestring / Multilinestring [3]
  - Polygon / Multipolygon [4]
- **GeoSeries**: Eine vielzahl an *Geometries*, typischerweise eine Spalte einer *GeoDataFrame* namens (*geometry*)
- **GeoDataFrame**: Eine Tabelle, welche über eine Geometrie-Spalte (*GeoSeries*) verfügt



[1] Die Geometrien in Geopandas sind eigentlich Objekte vom Modul *Shapely*. Shapely wiederum ist ein Python Modul, welches mit *Geopandas* mit-installiert und mit-importiert wird.

[2] Der Unterschied zwischen Point und *Multi*point, Linesstring und *Multi*linestring und Polygon *Multi*polyon ist folgendermassen: Typischerweise hat jedes Attribut *eine* Geometrie. Zum Beispiel hat jede Zeckenstichmeldung eine ID, eine Ungenauigkeitsangabe und eben *eine* Punkgeometrie. Somit handelt es sich bei diesem Datensatz um einen einfach *Point* Datensatz. Wenn pro Attribut mehrere Geometrien zugewiesen sind handelt es sich um ein *Multi*point Objekt.

[3] Der Unterschied zwischen Point und *Multi*point, Linesstring und *Multi*linestring und Polygon *Multi*polyon ist folgendermassen: Typischerweise hat jedes Attribut *eine* Geometrie. Zum Beispiel hat jede Zeckenstichmeldung eine ID, eine Ungenauigkeitsangabe und eben *eine* Punkgeometrie. Somit handelt es sich bei diesem Datensatz um einen einfach *Point* Datensatz. Wenn pro Attribut mehrere Geometrien zugewiesen sind handelt es sich um ein *Multi*point Objekt.

[4] Der Unterschied zwischen Point und *Multi*point, Linesstring und *Multi*linestring und Polygon *Multi*polyon ist folgendermassen: Typischerweise hat jedes Attribut *eine* Geometrie. Zum Beispiel hat jede Zeckenstichmeldung eine ID, eine Ungenauigkeitsangabe und eben *eine* Punkgeometrie. Somit handelt es sich bei diesem Datensatz um einen einfach *Point* Datensatz. Wenn pro Attribut mehrere Geometrien zugewiesen sind handelt es sich um ein *Multi*point Objekt.

Die drei Hierarchien in Geopandas: Eine *GeoDataFrame* verfügt immer über eine Geometrie-Spalte, welche sich eine *GeoSeries* nennt. Diese wiederum besteht aus Einzelgeometrien, sogenannten *Geometries*.

39.3. Koordinatenbezugssystem

Mit `crs =` wird das Koordinatenbezugssystem (engl. **C**oordinate **R**eference **S**ystem) unseres Datensatzes festgelegt. Das Koordinatenbezugssystem gibt unseren x/y-Zahlenwerten einen konkreten Raumbezug auf dem Planeten und macht aus ihnen Koordiaten *in der Schweiz*. Wie lautet aber das "Koordinatenbezugssystem" unserer Daten?

Im Prinzip weiss diese Information nur derjenige, der die Daten erstellt hat. Man man das Koordinatensytem aber auch anhand der Koordinaten erahnen: Es handelt sich in unserem Fall um Werte im Bereich von 2'600'000 auf der einen und 1'200'000 auf der anderen Achse. Da wir wissen das die Daten aus der Schweiz stammen kann man mit etwas Erfahrung sagen, dass es sich um Daten im neuen Schweizer Koordinatenbezugssystem CH1903+ / LV95 handeln muss. Der EPSG Code dieses Koordinatenbezugssystems lautet 2056 und diesen Code können wir in der Funktion `gpd.GeoDataFrame` nutzuen, um das Korrekte Koordinatenbezugssystem zu zuweisen.

```
# Das Attribut `crs` wurde aufgrund vom EPSG Code richtig erkannt:
zeckenstiche_gpd.crs.name
```

```
NameError: name 'zeckenstiche_gpd' is not defined
```

Diese CRS Information erlaubt uns auch, mit einer Zeile Code eine Webmap unserer Zeckenstiche zu machen. Versucht es aus!

```
zeckenstiche_gpd.explore()
```

```
NameError: name 'zeckenstiche_gpd' is not defined
```

39.4. Geodatenformate

Das Einlesen von CSV und die Konvertierung von *DataFrame* zu *GeoDataFrame* hat bei den Zeckenstichen zwar gut funktioniert, es ist aber auch etwas umständlich jedes Mal die `geometry` Spalte und das `crs` zu setzen. CSVs eignen sich nicht besonders gut für das speichern von Geodaten, insbesonders wenn die Daten komplexer sind (Linien, Polygone..)). Deshalb gibt

es auch spezifische Datenformate, in den Geodaten gespeichert werden können. Bei diesen Datenformaten werden `geometry` und `crs` automatisch abgespeichert.

Ihr seid im Studium mit solchen Datenformate bereits in Kontakt gekommen, sicher kennt ihr ESRI's *File Geodatabases/Feature Classes* sowie *Shapefiles*. Dies sind häufig genutzte Formate, aber leider nicht offen, und gerade Shapefiles haben viele Tücken (siehe switchfromshapefile.org). Es gibt dafür ganz tolle Alternativen, beispielsweise *Geopackages* (nicht zu verwechseln mit ArcGIS Pro Packages!). Mit nachstehendem Befehl können wir `zeckenstiche_gpd` als Geopackage abspeichern.

```
zeckenstiche_gpd.to_file("data/zeckenstiche.gpkg", driver = "GPKG")
# Wer UNBEDINGT ein shapefile abspeichern will,
# kann ".gpkg" mit ".shp" ersetzen
# (-_-)
```

Das File "zeckenstiche.gpkg" befindet sich nun in meiner *Working Directory* und ich kann sie mit `gpd.read_file("data/zeckenstiche.gpkg")` wieder einlesen. Im Unterschied zu vorher musst ich nun `geometry` und `crs` nicht mehr zuweisen, diese sind beim Schreiben des Geopackage abgespeichert worden. Das Geopackage kann ich nun auch mit ArcGIS / QGIS öffnen, wenn ich die Punkte interaktiv explorieren möchte.

Im Block "Datenqualität und Unsicherheit" habt ihr mit dem Wald Datensatz gearbeitet. Ich habe diesen als *Geopackage* exportiert (siehe Tabelle 6.1) und kann ihn wie im nachstehend Codeblock ersichtlich einlesen und visualisieren.

```
wald = gpd.read_file("data/wald.gpkg") # <- importiert das File "wald.gpkg"

wald
```

```
NameError: name 'gpd' is not defined
```

Es handelt sich also um einen Datensatz mit nur zwei Zeilen, aber mehreren Polygonen pro Zeile (das macht es zu einem ***Multi****polygon*[5]). Mit `plot` können wir diesen Datensatz einfach visualisieren und mit den Zeckenstichdaten überlagern.

```
base = wald.plot(color = ["Lightgrey","Green"])
zeckenstiche_gpd.plot(ax = base, color = "red")
```

[5]Der Unterschied zwischen Point und *Multi*point, Linesstring und *Multi*linestring und Polygon *Multi*polyon ist folgendermassen: Typischerweise hat jedes Attribut *eine* Geometrie. Zum Beispiel hat jede Zeckenstichmeldung eine ID, eine Ungenauigkeitsangabe und eben *eine* Punkgeometrie. Somit handelt es sich bei diesem Datensatz um einen einfach *Point* Datensatz. Wenn pro Attribut mehrere Geometrien zugewiesen sind handelt es sich um ein *Multi*point Objekt.

```
NameError: name 'wald' is not defined
```

Noch einfacher und sogar mit einer Basemap und Zoom möglichkeit, geht es mit der `explore()` Methode. Nun wird übrigens auch klar, dass es sich bei unseren 10 Zeckenstichen um sehr dicht beieinanderliegenden Punkte handelt.

```
base = wald.explore()
zeckenstiche_gpd.explore(m = base, color = "red")
```

```
NameError: name 'wald' is not defined
```

40. Räumliche Operationen

Was bringt uns diese *Geo* Erweiterung? Mit *GeoDataFrames* sind nun alle räumliche Operationen möglich, die wir bereits aus ArcGIS kennen aber mit einfachen *DataFrames* noch nicht möglich waren. Ich möchte dies an ein paar Beispielen Demonstrieren. Dazu müssen wir die Zeckenstiche in GeoDataFrame konvertiert und in ein Geopackage exportiert haben, wie in Kapitel 39 beschrieben.

```
import geopandas as gpd

zeckenstiche = gpd.read_file("data/zeckenstiche.gpkg")
```

```
ModuleNotFoundError: No module named 'geopandas'
```

Hinweis

- Die verschiedenen räumlichen Operationen in Geopandas erwarten unterschiedlichen Input, deshalb müssen wir manchmal zwischen *Geometrien*, *Geoseries* und *GeoDataFrames* hin- und her konvertieren (siehe Kapitel 39.2).
- Welcher Datentyp eure Operation *braucht* seht ihr in der Dokumentation. Welcher Datentyp ihr *habt* seht ihr mit `type()`.
- Um ein Objekt von einem Format in das andere zu konvertieren (angenommen das Objekt heisst `x`)

```
# von GeoDataFrame zu Geoseries:
zecken_geoseries = gpd.GeoSeries(zeckenstiche["geometry"])

# von GeoSeries zurück zu GeoDataFrame:
zecken_geodataframe = gpd.GeoDataFrame(geometry = zecken_geoseries)
```

```
NameError: name 'gpd' is not defined
```

40.1. Buffer

Eine typische GIS Operation ist das "Buffern" von Objekten. Der ArcGIS Befehl "Buffer" erreichen wir in Geopandas mit `.buffer()`. Folgender Code macht einen Buffer mit einer Distanz von 10m.

```
buffered = zeckenstiche.buffer(10)
```

```
NameError: name 'zeckenstiche' is not defined
```

Um Geopandas-Objekte zu plotten kann man einfach `.plot()` verwenden. Zudem kann man mit `boundary` die Umrisse eines Polygons extrahieren:

```
base = buffered.boundary.plot() # plottet die boundries

zeckenstiche.plot(ax = base, color = "black") # plottet die Punkte
```

```
NameError: name 'buffered' is not defined
```

40.2. Union

Mit `unary_union` können wir aus unserer *Point*-Geometrie ein *MultiPoint* erstellen (siehe Kapitel 39.2). Dieser Befehl lautet in ArcGIS Union.

```
zeckenstiche_union = zeckenstiche["geometry"].unary_union

type(zeckenstiche_union) # Es handelt sich nun um den Typ "MultiPoint"
```

```
NameError: name 'zeckenstiche' is not defined
```

Wenn wir uns `zeckenstiche_union` nun mit `print` anschauen sehen wir, dass sämtliche Koordinaten in einem Objekt zusammengepackt sind:

```
print(zeckenstiche_union)
```

```
NameError: name 'zeckenstiche_union' is not defined
```

40.3. Minimum Bounding Geometry

Über ein *MultiPoint* lassen sich jetzt wunderbar sogenannte (in ESRI Terminologie) Minimum Bounding Geometries rechnen. Mit den gleichnamigen Funktionen können wir nun eine `convex_hull` [1] sowie eine `envelope` [2] über alle Punkte rechnen.

```
my_convex_hull = zeckenstiche_union.convex_hull
my_envelope = zeckenstiche_union.envelope
```

```
NameError: name 'zeckenstiche_union' is not defined
```

Nun konvertiere ich beide Polygon-Geometrien in *GeoSeries*, damit sie einfacher zu visualisieren sind:

```
my_convex_hull = gpd.GeoSeries(my_convex_hull)
my_envelope = gpd.GeoSeries(my_envelope)
```

```
NameError: name 'gpd' is not defined
```

Um die beiden Objekte nebeneinander zu visualisieren importiere ich zuerst `pyplot` aus `matplotlib` (mit dem alias `plt`) und erstelle `subplots`

```
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2,sharex=True, sharey = True,figsize = (9, 9))

# Erstellt den linken Plot
my_convex_hull.plot(ax = ax1)
ax1.set_title("Convex Hull")
zeckenstiche.plot(ax = ax1, color = "black")

# Erstellt den rechten Plot
my_envelope.plot(ax = ax2)
ax2.set_title("Envelope")
zeckenstiche.plot(ax = ax2, color = "black")
```

```
ModuleNotFoundError: No module named 'matplotlib'
```

[1] Convex Hull stellt ein "Rahmen" um alle Punkte dar, wo alle Innenwinkel kleiner sind als 180° (*konvex*)
[2] Envelope stellt ebenfalls ein "Rahmen um alle Punkte dar, die aber quadratisch geformt und am Koordiatensystem ausgerichtet ist.

40.4. Overlay

Viele der Funktionen aus dem ESRI Toolset "Overlay" sind in der *Geopandas* Funktion `overlay` verpackt. Um sie zu demonstrieren nutze ich die Geometrien, die wir in weiter oben erstellt haben (`buffered` und `my_convex_hull`). Zuerst muss ich sie aber noch von *GeoSeries* in *GeoDataFrames* konverieren.

```
buffered_gdf = gpd.GeoDataFrame(geometry = buffered, crs = 2056)
my_convex_hull_gdf = gpd.GeoDataFrame(geometry = my_convex_hull, crs = 2056)
```

```
NameError: name 'gpd' is not defined
```

Nun kann ich zum beispielsweise die Overlay-Funktion `difference` ausführen:

```
my_difference = gpd.overlay(my_convex_hull_gdf,buffered_gdf, how='difference')
```

```
NameError: name 'gpd' is not defined
```

```
# Bereitet die drei Subplots vor #################
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, sharex=True, sharey = True, figsize = (9,9))
##################################################

# Plot links #####################################
my_convex_hull_gdf.plot(ax = ax1)               #
ax1.set_title("1. Das Minimum Convex Polygon")  #
ax1.set_axis_off()                              #
# Plot mitte #####################################
buffered_gdf.plot(ax = ax2)                     #
ax2.set_title("2. Die gebufferten Punkte")      #
ax2.set_axis_off()                              #
# Plot rechts ####################################
my_difference.plot(ax = ax3)                    #
ax3.set_title("Differenz aus 1. & 2.")          #
ax3.set_axis_off()                              #
##################################################
```

```
NameError: name 'plt' is not defined
```

41. Übung 14

Nun ist es Zeit, dass ihr selbst mit *GeoDataFrames* Hand anlegt. Achtet dabei immer auf die Datentypen eurer Daten (mit `type`) und konsultiert dazu Kapitel 39.2. Zudem ist Geopandas gut dokumentiert, es lohnt ich diese immer wieder zu konsultieren: geopandas.org

41.1. Übung 14.1

Importiere *Geopandas* und wandle `zeckenstiche` in eine *GeoDataFrame* um (`zeckenstiche`). Vergiss nicht, das Koordinatenbezugssystem festzulegen!

```
import pandas as pd
import geopandas as gpd

# Übung 14.1
zeckenstiche = pd.read_csv("data/zeckenstiche.csv")

geom = gpd.points_from_xy(
  zeckenstiche['x'],    #
  zeckenstiche['y']     # → die Geometrie Spalte
  )

zeckenstiche_gpd = gpd.GeoDataFrame(
    zeckenstiche,    # die DataFrame ("Attributtabelle")
    geometry = geom, #
    crs = 2056       # das Koordinatenbezugssystem (EPSG Code)
)
```

41.2. Übung 14.2

Buffere die Zeckenstiche um eine Distanz von 12 Meter und speichere den Output in der Variabel `zeckenstiche_buffer`. Visualisiere die gebufferten Punkte mit `.explore()`.

```
zeckenstiche_buffer = zeckenstiche_gpd.buffer(12)
zeckenstiche_buffer.explore()
```

```
NameError: name 'zeckenstiche_gpd' is not defined
```

41.3. Übung 14.3

Extrahiere die Umrisse von `zeckenstiche_buffer` und speichere diese in `zeckenstiche_buffer_outline`. Visualisiere anschliessend diese Umrisse.

```
zeckenstiche_buffer_outline = zeckenstiche_buffer.boundary

zeckenstiche_buffer_outline.explore()
```

```
NameError: name 'zeckenstiche_buffer' is not defined
```

41.4. Übung 14.4

Nutze nachstehenden Code um zwei Datensätze im gleichen Plot darzustellen.

```
# Nicht interaktiv:
from matplotlib import pyplot as plt
fig, ax = plt.subplots()

zeckenstiche_buffer_outline.plot(ax = ax, color = "green")
zeckenstiche_gpd.plot(ax = ax, color = "pink")
```

```
ModuleNotFoundError: No module named 'matplotlib'
```

```
# Interaktiv:
base = zeckenstiche_buffer_outline.explore(color = "green")
zeckenstiche_gpd.explore(m = base, color = "pink")
```

```
NameError: name 'zeckenstiche_buffer_outline' is not defined
```

41.5. Übung 14.5

Berechne das "Envelope" von `zeckenstiche_gpd` anhand der Beispiele in Kapitel 40. Speichere den Output als `zeckenstiche_envelope`.

> **Hinweis**
>
> Denk daran, dass du zuerst noch einen Union machen musst (siehe Kapitel 40)

```
zeckenstiche_envelope = zeckenstiche_gpd.unary_union.envelope

zeckenstiche_envelope
```

41.6. Übung 14.6

Exportiere `zeckenstiche_gpd` als "Geopackage" mit dem Namen "zeckenstiche.gpkg". Lese nochmal Kapitel 39.4 wenn du nicht mehr weisst, wie das geht. Versuche anschliessend, "zeckenstiche.gpkg" wieder einzulesen.

```
zeckenstiche.to_file("data/zeckenstiche.gpkg")
```

42. Spatial Joins

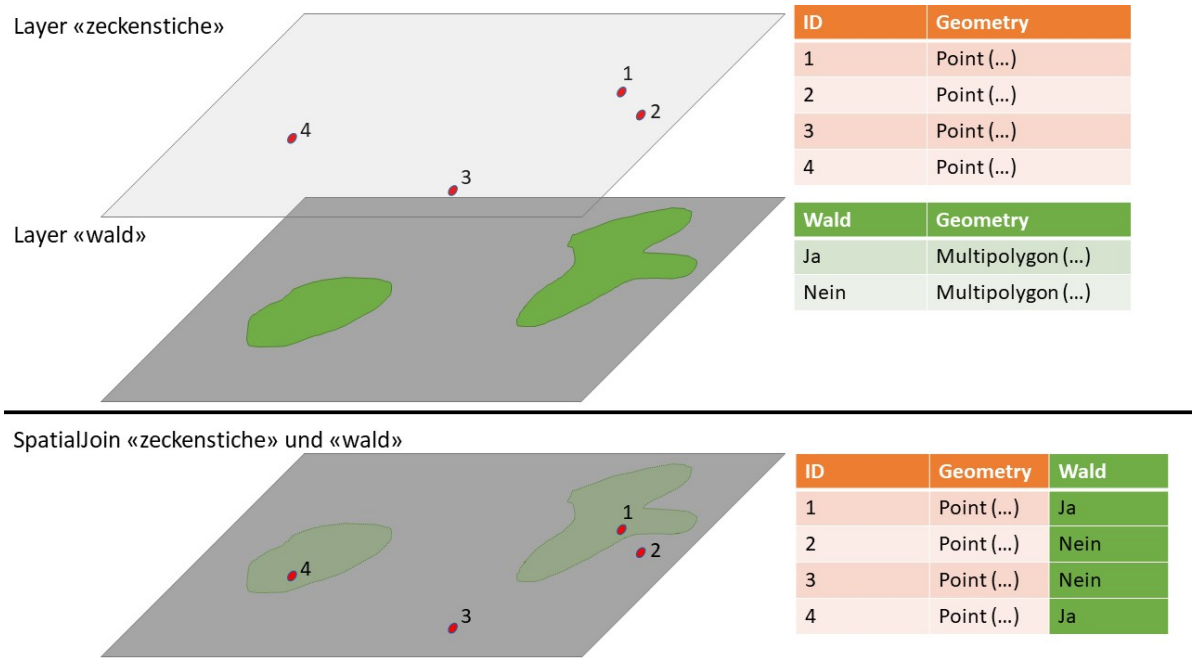
In dieser Aufgabe wollen wir für jeden Zeckenstich ermitteln, ob er sich im Wald befindet oder nicht. Den Wald Layer kennt ihr bereits aus dem Block "Datenqualität und Unsicherheit" und wir haben ihn in Kapitel 39.4 kurz angeschaut. Nutzen wir hier nochmal die Gelegenheit, um den Wald und die Zeckenstiche (siehe Tabelle 6.1) als Geodaten einzulesen und in einer grossem Plot zu visualisieren.

```
import geopandas as gpd

zeckenstiche_gpd = gpd.read_file("data/zeckenstiche.gpkg")
wald = gpd.read_file("data/wald.gpkg")

minx, miny, maxx, maxy = zeckenstiche_gpd.geometry.total_bounds  # holt die x und y min bzw.

from matplotlib.colors import ListedColormap

base = wald.plot(column = "Wald_text", legend = True, cmap = ListedColormap(["green","lightg
zeckenstiche_gpd.plot(color = "Red", ax = base)

base.set_xlim(minx - 10, maxx + 10)
base.set_ylim(miny - 5, maxy + 5)
```

```
ModuleNotFoundError: No module named 'geopandas'
```

42.1. Wie funktioniert ein *Spatial Join*?

In Kapitel 39 habt ihr euch mit den GIS-Funktionalitäten von `geopandas` vertraut gemacht. Eine ganz zentrale Funktion in GIS sind die sogenannten "Spatial Joins". Dabei werden Attribute von einem Geodatensatz auf einen anderen Geodatensatz aufgrund einer räumlichen Beziehung der beiden Datensätze übertragen. Konkret auf unsere Zeckenstiche bedeutet dies: Jedem Zeckenstich sollte die Eigenschaft "Wald: ja" / "Wald: nein" aus 'wald zugewiesen werden. Am einfachsten lässt sich dies in einer Darstellung erklären:



ID	Geometry
1	Point (...)
2	Point (...)
3	Point (...)
4	Point (...)

Wald	Geometry
Ja	Multipolygon (...)
Nein	Multipolygon (...)

ID	Geometry	Wald
1	Point (...)	Ja
2	Point (...)	Nein
3	Point (...)	Nein
4	Point (...)	Ja

Abbildung 42.1.: "Spatial Join" zwischen `zeckenstiche` und `wald`. In diesem Spatial Join wurde die Geometrie von `zeckenstiche` übernommen, das heisst das Resultat des Joins ist ein Punkt-Layer.

In Python wird ein *Spatial Join* zwischen `zeckenstiche` und `wald` wie folgt durchgeführt (wichtig ist dabei auch die Reihenfolge der Argumente: `left_df` bestimmt den Geometrietyp des Outputs):

```
gpd.sjoin(left_df = zeckenstiche_gpd, right_df = wald)
```

```
NameError: name 'gpd' is not defined
```

43. Übung 15

43.1. Übung 15.1

Lade das File "wald.gpkg" (siehe Tabelle 6.1) herunter und speichere es in deiner Workings Directory. Importiere den Datensatz und speichere ihn in der Variable `wald`.

Schau dir `wald` an (mit `type`, `.plot()` etc.)

```
import geopandas as gpd

wald = gpd.read_file("data/wald.gpkg")

wald
type(wald)
wald.plot()
```

43.2. Übung 15.2

Führe einen SpatialJoin zwischen `wald` und `zeckenstiche` durch. Vertausche die Reihenfolge (`left_df`, `right_df`) und schaue dir den Output an. Wodurch unterscheiden sich die beiden outputs? Was sagt dir das über die Funktionsweise von `sjoin`?

> **Wichtig!**
>
> Melde dich bei den Betreuern wenn der *SpatialJoin* bei dir nicht funktioniert und du eine Fehlermeldung bekommst.

```
zeckenstiche_gpd = gpd.read_file("data/zeckenstiche.gpkg")

gpd.sjoin(left_df = zeckenstiche_gpd, right_df = wald)
gpd.sjoin(left_df = wald, right_df = zeckenstiche_gpd)

# in beiden Fällen hat der Output geich viele Zeilen.
```

```
# In der ersten Variante ist die Geometrie
# des Outputs "Point", im zweiten Fall "Multipolygon"
```

43.3. Übung 15.3

Führe nun den SpatialJoin aus Kapitel 43.2 in der Reihenfolge aus, der dir am sinnvollsten erscheint. Weise den Output dieser Operation der Variabel `zeckenstiche_join` zu.

Nutze nun die dazugewonnene Spalte `Wald_text` um die Anzahl Zeckenstich-Meldungen *im Wald* zu zählen. Dazu musst du wie folgt vorgehen:

1. Entsprechende Spalte selektieren (siehe Kapitel 16.5)
2. Werte in der Spalte mit "ja" vergleichen
3. Anzahl `True` mit `sum()` zählen (`True` entspricht immer `1`, `False` entspricht immer `0`)

```
# die folgende Variante (mit dem Output "point") ist sinnvoller, da auch
# wir uns primär für die Punkte (Zeckenstiche) interessieren.
zeckenstiche_join = gpd.sjoin(left_df = zeckenstiche_gpd, right_df = wald)

sum(zeckenstiche_join["Wald_text"] == "ja")
```

43.4. Übung 15.4

Berechne nun *Anteil* der Zeckenstiche im Wald (gemessen an der Anzahl Zeckenstich-Meldungen total). Weise den Output der Variabel `anteil_wald` zu.

```
anteil_wald = sum(zeckenstiche_join["Wald_text"] == "ja")/len(zeckenstiche_join["Wald_text"])
```

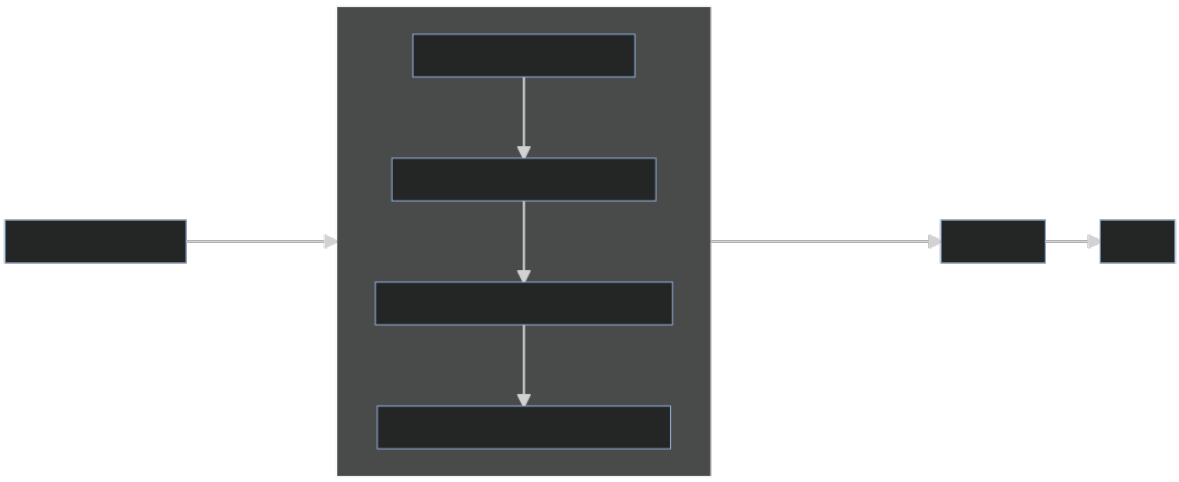
44. Leistungsnachweis

44.1. Ziel und Vorgehen

Für den Leistungsnachweis von *Programmieren* sollt ihr den Anteil der Zeckenstiche im Wald unter der Berücksichtigung der Unsicherheit mit einer Monte Carlo Simulation ermitteln.

Gehe dazu wie folgt vor: Erstelle einen `for loop` (siehe Kapitel 35), welcher pro Iteration folgende Schritte ausführt:

1. Schritt: Zeckenstiche importieren
2. Schritt: Zeckenstiche auf der Basis von gemeldeten Stiche simulieren (siehe Kapitel 30)
3. Schritt: Simulierte Zeckenstiche von pandas DataFrame in *geo*pandas *Geo*DataFrame umwandeln (siehe Kapitel 39.1)
4. Schritt: Spatial Join mit wald.gpkg (siehe Kapitel 43.2)
5. Schritt: Der Anteil der Zeckenstiche im Wald Berechnen (siehe Kapitel 43.3)
6. Schritt: Schritte 2 - 5 mindestens 50x Wiederholen (`for loop`)
7. Schritt: Die verschiedenen Waldanteile visualiseren (z.B. in einem Boxplot, siehe Kapitel 45.1)
8. Schritt: Schritte 6 und 7 für drei verschiedene Radien wiederholen
9. Schritt (Optional): Eine Karte mit den simulierten Zeckenstichen erstellen (siehe Kapitel 45.2)



44.2. Anforderungen

Führe die Analyse folgendermassen durch:

- mit dem **ganzen** Datensatz `zeckenstiche_full.csv` (1'076 Meldungen, siehe Tabelle 6.1)
- mit mindestens 50 Iterationen
- mit drei verschiedenen Distanz-Werten (`distance =` in Kapitel 29.5).

Visualisiere für jeden Distanzwert die Verteilung der Anteile (z.B: mittels einem Boxplot)

44.3. Struktur

Führe die Datenanalyse durch, indem du Python Code mit Markdown Text ergänzt.

Strukturiere den Bericht **mithilfe von Markdown** in folgende Unterkapitel

- Einleitung
- Material & Methoden
- Resultate
- Diskussion

Sorge dafür, dass der Bericht sauber und ordentlich daher kommt (vermeide zum Beispiel lange Python Messages im Bericht, entferne unnötigen Code). Führt die Datenanalyse durch, indem ihr den Python Code **mit Markdown Text** unterstützt. Beschreibt und begründet eure Analyseschritte und wie die Resultate zu interpretieren sind. Der Fokus soll auf der technischen Umsetzung liegen, nicht auf der Interpretation der Resultate.

44.4. Abgabeformat

Das Jupyter-Notebook File via Moodle. **Wichtig:** Zuerst alle Zellen ausführen (*Run > Restart Kernel and Run all cells…*). **Notebooks, bei denen dieser Schritt nicht ausgeführt wurden, werden nicht akzeptiert!**

45. Anhang

45.1. Anhang 1: Daten visualisieren

Es ist äusserst Zentral, Daten regelmässig und oft zu visualisieren. Die *de facto* standart Library hierfür in Python ist `matplotlib`. Diese Library kann man direkt ansteuern wie in diesem Tutorial beschrieben wird. Wir verwenden die library jedoch etwas anders: Dabei nutzen wir die Tatsache aus, dass sowohl `pandas` wie auch `geopandas` eingebaute Methoden (*methods*) haben um deren Inhalte zu visualisieren. Die Methode heisst in beiden Fällen `.plot()`, wie wir weiter unten noch sehen werden.

45.1.1. Histogramm aus `List`

`pandas` vereinfacht das Visualisieren von Daten sogar soweit, dass es sich jewils lohnt seine Listen, Dictionaries usw. zuerst in eine `Series` oder `DataFrame` zu überführen um sie zu visualisieren (wie ich zum Beispiel in Kapitel 28 jeweils gemacht habe).

```
import random
import pandas as pd

random_gamma = [random.gammavariate(1, 1) for x in range(1000)]

random_gamma = pd.Series(random_gamma)

random_gamma.plot(kind = "hist", bins = 50)
```

```
ModuleNotFoundError: No module named 'pandas'
```

Weitere Beispiele zu Histogrammen aus Listen findet ihr im Kapitel Kapitel 28 .

45.1.2. Boxplot aus `List`

Das Visualisieren als Boxplot ist sehr ähnlich, man ersetzt `"hist"` lediglich durch `"box"`. Eine komplette liste der möglichen Argumente für findet ihr hier: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.plot.html

```
random_gamma.plot(kind = "box")
```

```
NameError: name 'random_gamma' is not defined
```

45.1.3. Scatterplot aus DataFrame

Um einen Scatterplot zu erstellen, braucht es eine `DataFrame` (eine `Series` reicht hierfür nicht aus).

```
zeckenstiche = pd.read_csv("data/zeckenstiche.csv")
```

```
NameError: name 'pd' is not defined
```

```
zeckenstiche.plot("x", "y", kind = "scatter")
```

```
NameError: name 'zeckenstiche' is not defined
```

45.1.4. Statische Karte aus GeoDataFrame

Um aus *GeoDataFrames* Karten zu machen verwendet man ebenfalls die `.plot()` Methode. Um wirklich schöne Karten mit Python herzustellen braucht man viel Übung. Für eine schnelle Visualisierung reicht aber die `.plot()` Methode.

```
import geopandas as gpd

zeckenstiche = gpd.read_file("data/zeckenstiche.gpkg")
wald = gpd.read_file("data/wald.gpkg")
```

```
ModuleNotFoundError: No module named 'geopandas'
```

Einfache Plots ohne anpassung:

```
wald.plot()
```

```
NameError: name 'wald' is not defined
```

Anpassung der Plot Grösse:

```
wald.plot(figsize = (5,5))
```

```
NameError: name 'wald' is not defined
```

Choroplethenkarte Karte

```
from matplotlib.colors import ListedColormap

my_cmap = ListedColormap(["green","lightgrey"])
wald.plot(column = "Wald_text", legend = True, cmap = my_cmap)
```

```
ModuleNotFoundError: No module named 'matplotlib'
```

Mehrere Layers:

```
base = wald.plot(column = "Wald_text", legend = True, cmap = my_cmap)
zeckenstiche.plot(color = "Red", ax = base)
```

```
NameError: name 'wald' is not defined
```

45.1.5. Interaktive Karten aus GeoDataFrame

Weitere Informationen dazu findet ihr hier: https://geopandas.org/docs/user_guide/interactive_mapping.html

Vollautomatisch, ohne Anpassungen:

```
zeckenstiche.explore()
```

Zeckenstiche Rot eingefärbt:

```
zeckenstiche.explore(color = "red")
```

Zeckenstiche nach `"accuracy"` eingefärbt:

```
zeckenstiche.explore(column = "accuracy")
```

Wald nach `"Wald_text"` eingefärbt (beachte, dass ich `my_cmap` weiter oben erstellt habe!):

```
wald.explore(column = "Wald_text", cmap = my_cmap)
```

Zwei übereinander gelagerte Layers:

```
base = wald.explore(column = "Wald_text", cmap = my_cmap)
zeckenstiche.explore(m = base, color = "red")
```

```
NameError: name 'wald' is not defined
```

45.2. Anhang 2: Geodaten visualisieren

In folgenden Beispielen zeigen wir noch ein paar einfache Wege, wie ihr die Zeckenstichdaten visualisieren könnt.

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import descartes

wald = gpd.read_file("data/wald.gpkg")
zeckenstiche_full = pd.read_csv("data/zeckenstiche_full.csv")

geom = gpd.points_from_xy(zeckenstiche_full['x'], zeckenstiche_full['y'])

zeckenstiche_gpd = gpd.GeoDataFrame(
    zeckenstiche_full,
    geometry=geom,
    crs = 2056)
```

```
ModuleNotFoundError: No module named 'pandas'
```

45.2.1. Kernel Density Plot

Mit der Erweiterung `seaborn` lassen sich mit wenigen Zeilen Code eine *Kernel Density* rechnen und visualisieren. Siehe nachstehenden Code:

```
import seaborn as sns

f, ax = plt.subplots(1, figsize=(6, 6))
wald.boundary.plot(linewidth=0.1, ax=ax)
zeckenstiche_gpd.plot(color='red', linewidth=0.1, ax=ax)

sns.kdeplot(x = zeckenstiche_full.x, y= zeckenstiche_full.y,shade = False,n_levels = 10, cma

ax.set_axis_off()
plt.show()
```

```
ModuleNotFoundError: No module named 'seaborn'
```

45.2.2. Hintergrundkarte

Mit der Erweiterung `contextily` kann man sehr schnell und einfach Hintergrundkarten in den Plot einbinden. Dafür muss das *GeoDataFrame* vorher aber in WGS84 Koordinaten (EPSG 3857) konvertiert werden (mit `to_crs`, s.u.).

```
import contextily as ctx

f, ax = plt.subplots(1, figsize=(6, 6))
zeckenstiche_gpd.to_crs(epsg = 3857).plot(ax = ax)
ctx.add_basemap(ax)

ax.set_axis_off()
plt.show()
```

```
ModuleNotFoundError: No module named 'contextily'
```

45.2.3. KDE mit Hintergrundkarte

Kernel Denisty und Hintergrundkarte können auch kombiniert werden:

```
lat = zeckenstiche_gpd.to_crs(epsg = 3857).geometry.x
lng = zeckenstiche_gpd.to_crs(epsg = 3857).geometry.y

f, ax = plt.subplots(1, figsize=(6, 6))

sns.kdeplot(x = lat,y = lng,shade = False,n_levels = 25, cmap = "viridis", ax = ax)
ctx.add_basemap(ax)

ax.set_axis_off()
plt.show()
```

```
NameError: name 'zeckenstiche_gpd' is not defined
```

Teil V.

Datenbanken I

Ausgangslage

Die Gärten und Grünanlagen prägen das Bild des Campus Grüental. Eine grosse Vielfalt an verschiedenen einheimischen und exotischen Pflanzenarten lädt zum Verweilen ein. Mitarbeiter und Studierende nutzen den grünen Raum zur Erholung, zum Miteinander und zum Lernen. Nicht nur in den Sommermonaten ist das offene Areal des Campus ein Erlebnis für alle. Für die individuelle Pflege und Wartung der Gärten bedarf es sehr guter Planung. Als Datenbank Spezialist werden Sie beauftragt eine Datenbanklösung zur Verwaltung der Campus Pflanzenwelt zu entwickeln.

Übungsziele

- Sie lernen ein vorgegebenes Textmodell zu lesen, zu interpretieren und dazu die richtigen Fragen zu stellen, um fehlende Antworten zu bekommen.
- Sie können aus dem Textmodell relevante Objekte, Attribute und Beziehungen extrahieren.
- Sie können auf der Grundlage eines bestehenden Textmodells ein konzeptuelles Datenmodell mit Hilfe eines ER-Diagramms entwickeln.
- Sie können das konzeptuelle Datenmodell zu einem logischen Datenmodell weiter entwickeln, indem Sie Tabellen ableiten und Attribute, Schlüssel und Kardinalitäten identifizieren.

46. Übung

46.1. Anforderungen Pflanzendatenbank

Sie sind ein Datenbank-Spezialist und werden vom Garten-Team der ZHAW beauftragt eine Datenbanklösung zur Pflege und Verwaltung der Pflanzenwelt auf dem Areal des Campus Grüental zu entwickeln. Hierbei sollen alle Pflanzen des Campus katalogisiert und für verschiedene Nutzergruppen über differenzierte Zugriffsmöglichkeiten mit verschiedenen Informationen zugänglich gemacht werden.

Hierbei müssen folgende Anforderungen eingehalten werden:

Die Pflanzen müssen mit Familie, Gattung, Art und Sorte erfasst werden. Zusätzlich braucht es Informationen darüber, ob es sich um eine einheimische Pflanze handelt oder nicht. Als weitere Information muss es die Möglichkeit geben die Wuchsform, wie z.B. Staude, Strauch oder Baum zu erfassen. Zu einigen Pflanzen sollen Hinweise zur Pflege oder sonstige Bemerkungen eingegeben werden können.

Zu Lernzwecken sollen Pflanzen verschiedenen Pflanzenlernparcours zugeordnet werden können. Hierzu werden einzelne Pflanzen ausgewählt und einem Kurs zugeordnet. Jede Pflanze kann dabei zu verschiedenen Lernparcours gehören und ein Lernparcours kann unterschiedlich viele Pflanzen beinhalten.

Jede Pflanze auf dem Campus muss verortet werden. Dies bedeutet, dass jeder Pflanze auf dem Campus eine genaue Lage zugewiesen werden kann. Es reicht in dieser ersten Version aus die jeweilige Pflanze als Punkt darzustellen. Flächenartige Gewächse wie Stauden oder Gräser können als Punkt zusammengefasst werden. Linienartige Pflanzen wie Hecken können vernachlässigt werden und brauchen nicht aufgenommen zu werden. Für jede Pflanze ist zusätzlich zur genauen (Punkt-)Lage wichtig zu wissen auf welcher gärtnerischen Fläche (Beet) sich die Pflanze befindet. Zur besseren Planung müssen die Gärtner später abfragen können, welche Pflanzen auf welchem Beet stehen. Für die Flächen ist es ausreichend einen Namen und eine ID zu erfassen.

46.2. Aufgabenstellung

46.2.1. Schritt 1

Lesen Sie die Aufgabenstellung gewissenhaft. Skizzieren Sie ein erstes Textmodell (stichwortartig) bei welchem Sie bereits erste Objekte identifizieren und Attribute und Beziehungen ableiten. Überlegen Sie sich, ob es offene Fragen oder Unklarheiten gibt. Sie haben anschliessend die Gelegenheit den Auftraggeber über eventuell ungeklärte Anforderungen zu befragen, um ihr Textmodell zu vervollständigen.

46.2.2. Schritt 2 (optional)

Sind alle offenen Fragen geklärt und das Textmodell fertig, entwickeln Sie aus ihrem Textmodell ein konzeptuelles Modell. Identifizieren Sie alle notwendigen Objekte mit den zugehörigen Attributen und zeigen Sie die Kardinalitäten der Objekte untereinander auf. Nutzen Sie hierbei die ER-Modellierung, wie in den Vorlesungsfolien aufgezeigt.

46.2.3. Schritt 3

Zum Abschluss erstellen Sie auf der Grundlage ihres konzeptuellen Modells das erforderliche logische Modell. Leiten Sie aus den identifizierten Objekten, Attributen und Kardinalitäten die benötigten Tabellen und Schlüsselattribute ab. Achten Sie im Speziellen darauf wo es räumliche Tabellen sind und wo es sich um Sachdatentabellen handelt. Markieren Sie ausserdem auch alle Schlüssel-Attribute (Primärschlüssel und Fremdschlüssel).

Teil VI.

Datenbanken II

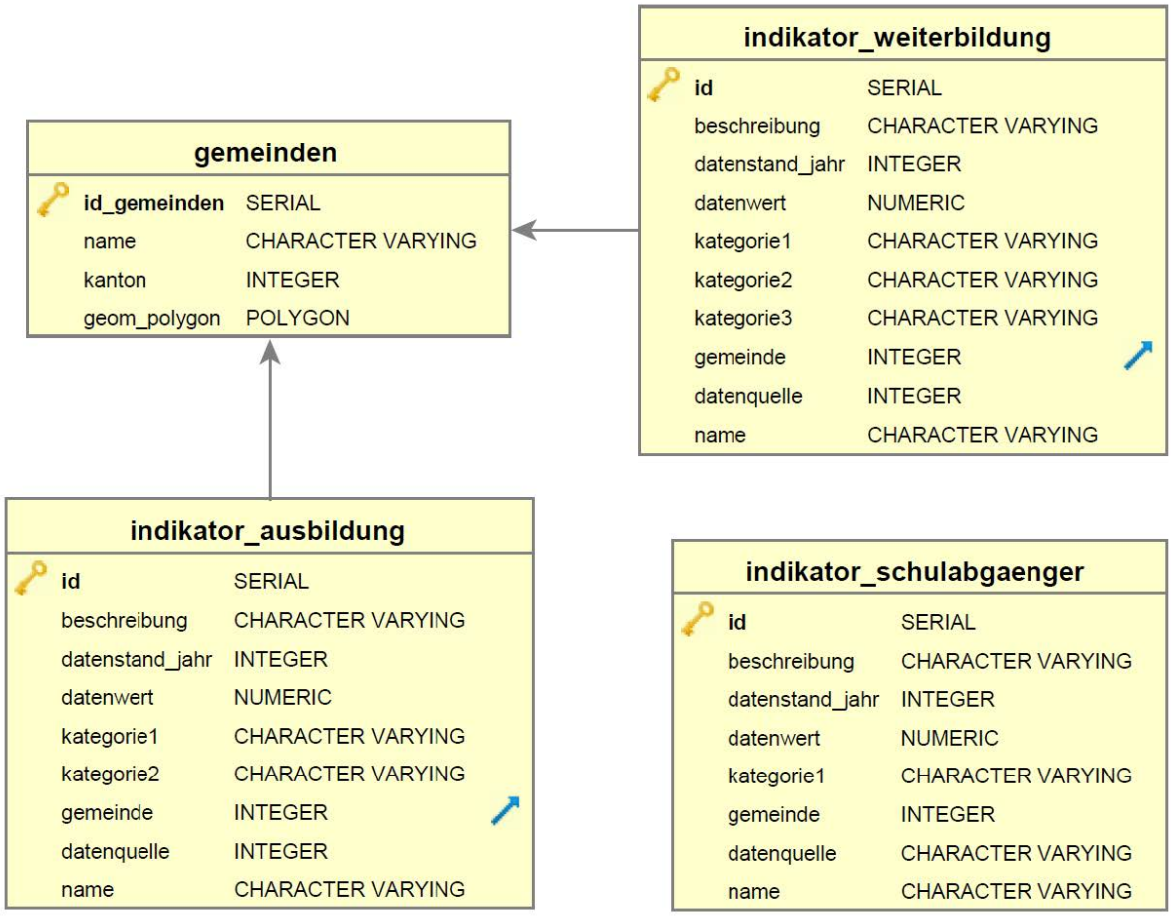
Übungsziele

- Sie können ein bestehendes logisches Modell auf Datenintegrität prüfen und Konsistenzfehler korrigieren.
- Sie können auf der Grundlage eines bestehenden Datenmodells mit Hilfe von pgAdmin und ArcGIS Pro auf eine Server Datenbank zugreifen.
- Sie können eine bestehende Server Datenbank mit Hilfe von pgAdmin und ArcGIS Pro mit Daten befüllen.
- Sie kennen die grundlegenden Mechanismen von Datenbank-Versionierung und können diese in ArcGIS Pro anwenden.

47. Datenintegrität

47.1. Übung 1: Datenintegrität und Konsistenz

Sie haben den Auftrag ein Datenmodell zur Speicherung von unterschiedlichen Sozialen Schlüssel-Indikatoren für schweizerische Gemeinden zu überprüfen. Hierbei werden jedes Jahr Datenwerte auf Basis einer Gemeinde erhoben. Diese sollen später in der Datenbank gespeichert und ausgewertet werden können. Wichtig hierbei ist, dass die Daten später in einer Karte visualisiert werden können. Deshalb ist es notwendig, dass Gemeinde-Geometrien ebenfalls in der Datenbank gespeichert werden sollen. Ebenfalls ist es sinnvoll bestimmte Indikatoren auch für den gesamten Kanton hochzurechnen, damit diese als Karte dargestellt werden können. Sie bekommen als erste Grundlage folgendes Datenbankmodell vom Auftraggeber gestellt. Leider gibt es in diesem Modell ein paar Konsistenzfehler.



Untersuchen Sie das Datenmodell auf Konsistenzfehler und korrigieren Sie diese. Lesen Sie den Aufgabentext aufmerksam und schauen sich die Abbildung an und entwickeln Sie daraus ein passendes logisches Datenmodell, welches alle Regeln der Datenintegrität einhält. Es muss nicht perfekt sein. Es geht in dieser Übung darum etwas Erfahrung im Lesen von gegebenen Modellen zu bekommen.

48. Arbeiten mit Server Datenbanken in ArcGIS Pro und pgAdmin

48.1. Übung 2: Datenbankverbindung zum Server in pgAdmin herstellen

In den nächsten Übungen wollen wir eine Beispiel Datenbank mit dem dort bereits implementierten Datenbankmodell abfragen. Hierzu müssen wir in einem ersten Schritt eine Verbindung zum PostgreSQL/PostGIS Datenbankserver herstellen. Hierfür nutzen wir das Datenbank-Administrations-Werkzeug pgAdmin.

1. Starten Sie pgAdmin.
2. Im Dashboard klicken Sie auf NEW Server und geben die folgenden Verbindungsparameter ein:
3. Register General: Name: svma-s-01323_waelder_studentNr. Verwenden Sie dabei die Nr. Ihres zugewiesenen Benutzers (siehe Abbildung 48.1)
4. Register Connection (siehe Abbildung 48.2):

    - Host name/address: svma-s-01323.zhaw.ch
    - Port: 5432
    - Maintenance database: waelder
    - Username: studentNr (Verwenden Sie hier Ihren zugewiesenen Anmeldenamen)
    - Passwort: (Verwenden Sie hier Ihr zugewiesenes Passwort)

5. Alle anderen Einstellungen können belassen werden. Speichern Sie mit Klick auf Save.
6. Lassen Sie sich nicht davon verunsichern, dass es bereits einige Einträge hat. Da auf dem Server bereits andere Datenbanken installiert sind, sehen Sie auch diese. Navigieren Sie zu unserer Datenbank “waelder” und verschaffen sich einen Überblick über die vorhandenen Tabellen.
7. Verwenden Sie für alle folgenden Übungen diese Datenbankverbindung, wenn nichts anderes angegeben ist (siehe Abbildung 48.2).

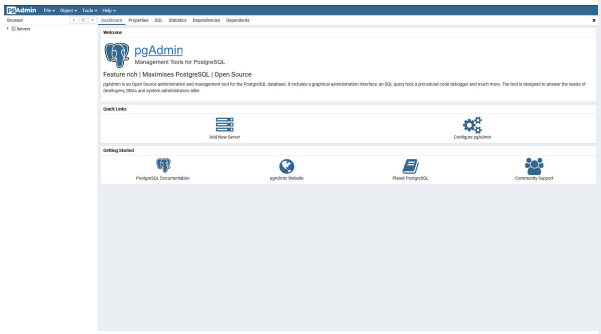


Abbildung 48.1.: PG Admin Oberfläche

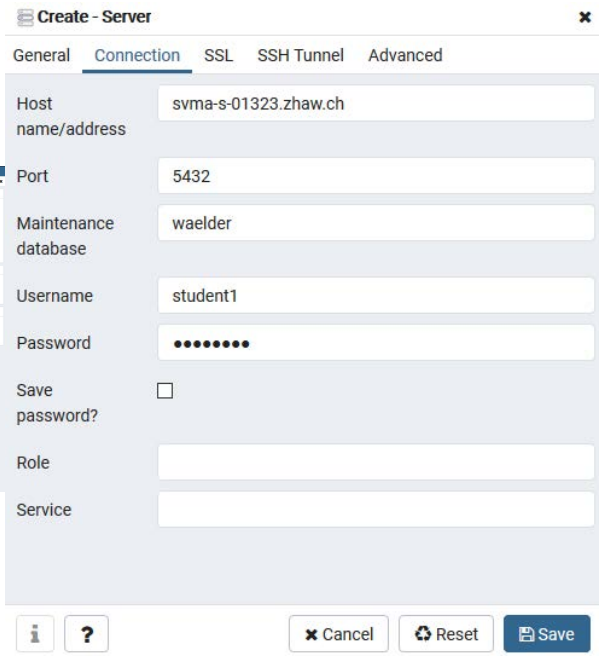


Abbildung 48.2.: PG Admin Connection

48.2. Übung 3: Verbindung zur PostgreSQL/PostGIS Datenbank mit ArcGIS Pro herstellen

Nachdem die Datenbank erstellt wurde und eine Verbindung über pgadmin eingerichtet wurde, benötigen wir auch eine Verbindung von ArcGIS Pro zu der Datenbank.

1. Starten Sie ArcGIS Pro und erstellen ein neues Projekt mit dem Namen “Datenbankzugriff”
2. Öffnen Sie das Catalog-Fenster oder das Register Catalog und wählen Sie “Databases” (siehe Abbildung 48.3).
3. Im Kontextmenü (rechte Maustaste) kann eine neue Datenbankverbindung zur Server Datenbank hergestellt werden.
   - Databases→Kontextmenü→New Database Connection
4. Geben Sie im Database Connection Fenster die notwendigen Verbindungsparameter ein (siehe Abbildung 48.4).
   - Database Platform: PostgreSQL
   - Instance: svma-s-01323.zhaw.ch
   - Authentication Type: Database authentication
   - User Name: studentNr (Verwenden Sie hier Ihren zugewiesenen Anmeldenamen)

- Password: (Verwenden Sie hier Ihr zugewiesenes Passwort)
- Database: waelder

5. Ein Klick auf OK stellt die Verbindung zur Datenbank her.
6. Sie können der Datenbankverbindung noch einen sinnvollen Namen geben. Es empfiehlt sich eine Kombination aus dem Servernamen, der verbundenen Datenbank und dem Benutzer. Z.B. “svma-s-01323_waelder_student1”
7. Unter dem Eintrag “Databases” im Catalog gibt es jetzt eine neue Datenbankverbindung zur Datenbank (siehe Abbildung 48.5).

48.3. Übung 4: Datenbank Schemata für ArcGIS Pro Benutzer

Nachdem die Datenbank erstellt wurde und alle Verbindungen eingerichtet sind, ist die Datenbank startklar. ArcGIS pro nutzt für die Daten jeweils ein eigenes Schema. Das Schema hat dabei den Namen des Datenbank Admin Users.

1. Nutzen Sie wieder die Verbindung mit der Datenbank “waelder” in pgAdmin und öffnen Sie die Inhalte/Eigenschaften mit Klick auf den kleinen Pfeil.
2. Navigieren Sie zum Eintrag Schemas. Dort sind 4 Einträge zu sehen. Ein Schema ist eine Art abgeschlossener Namensraum für die Datenhaltung. Es empfiehlt sich thematisch zusammenpassende Tabellen in einem gemeinsamen Schema zu speichern. Für die meisten Datenbank-Anwendungen genügt es aber alle Tabellen im default Schema “public” zu erstellen. Für den Zugriff über ArcGIS wurde aber durch das Werkzeug “Create Database User” ein spezielles Schema mit dem Namen des Database Users erstellt. Dies ist notwendig damit auch ArcGIS Pro auf die Daten zugreifen und diese verwalten kann. Das entsprechende Schema bekommt dabei immer den Namen des Admin Datenbankbenutzers. In diesem Fall “arcgispro_editor”. Sie können dort z.B. auch ein Schema mit Namen “sde” vom Geodatenbank-Administrator sehen (siehe Abbildung 48.6).
3. Öffnen Sie das public Schema. Hier gibt es bereits Einträge. Hier werden die zugehörigen Informationen der Datenbank, wie Sichten, Funktionen usw. zugänglich gemacht. Interessant ist zunächst aber nur der Eintrag “Tables (3)”. In Klammern ist immer die Anzahl der vorhandenen Datenbanktabellen angegeben. Wie wir sehen, gibt es default-Tabellen, welche bereits erstellt wurden. Die Tabelle “spatial_ref_sys” enthält z.B. alle Koordinatensysteme. Die Tabelle “sde_spatial_references” ist eine Hilfstabelle für Koordinatensysteme von ArcGIS Pro. Löschen Sie diese beiden Tabellen NIEMALS. Ohne diese Tabellen sind bestimmte Funktionen nicht mehr ausführbar (siehe Abbildung 48.7)

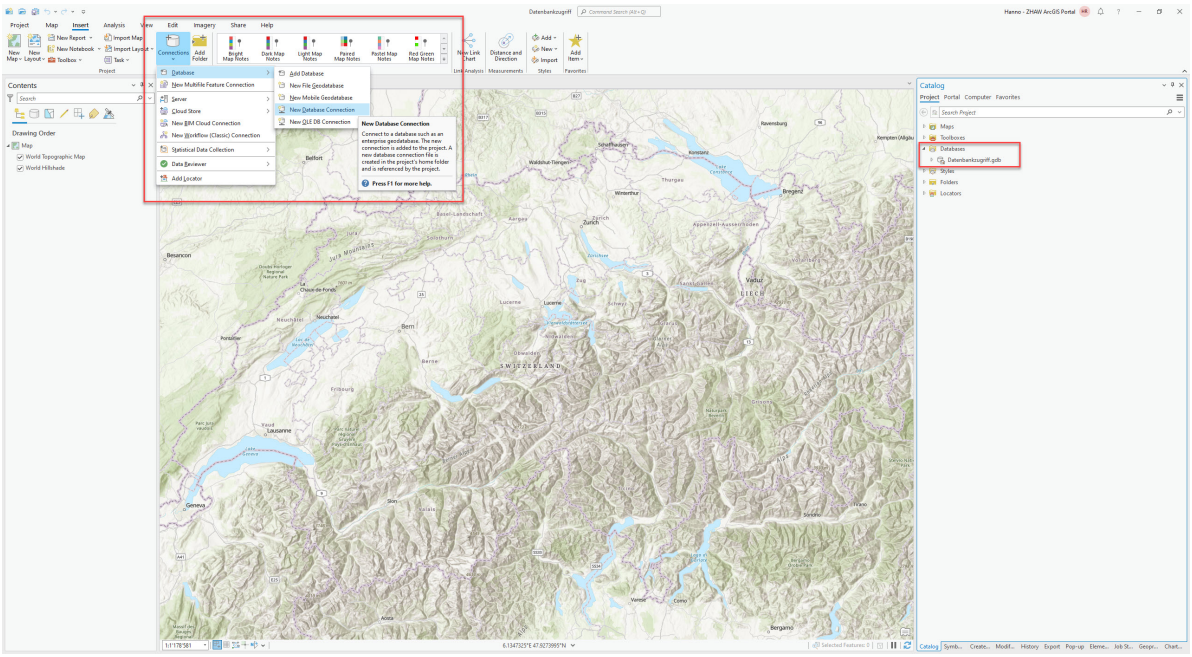


Abbildung 48.3.: Neue Datenbankverbindung in ArcGIS Pro

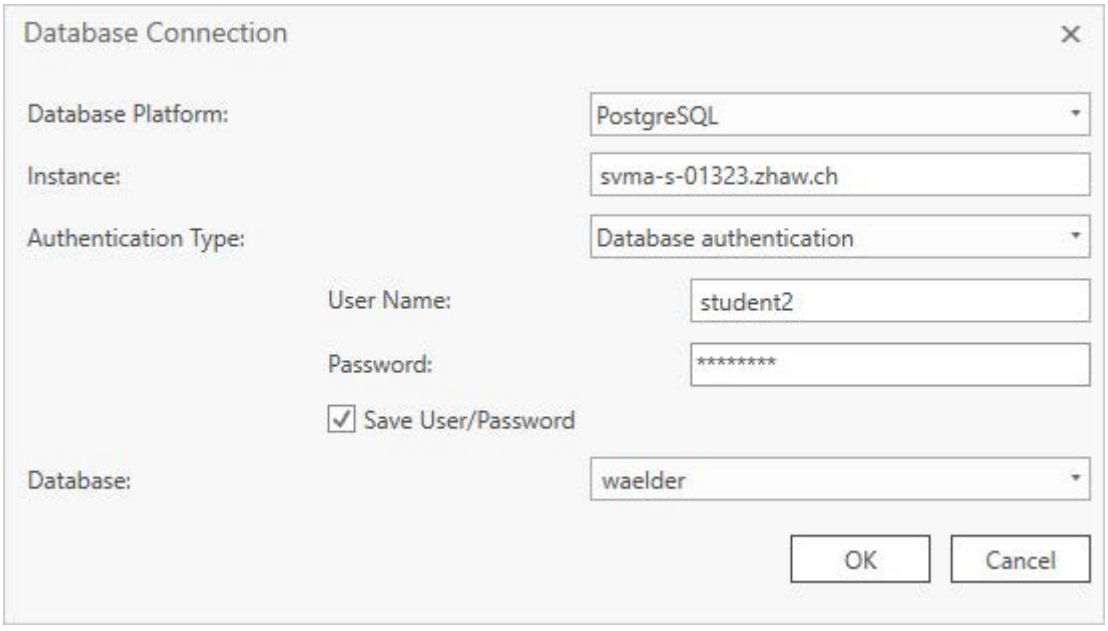
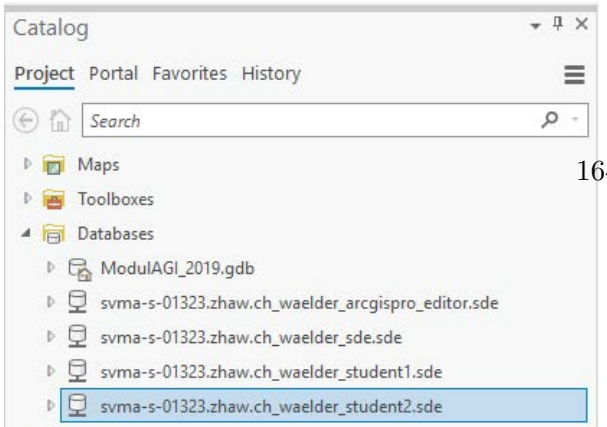


Abbildung 48.4.: Notwendige Verbindungsparameter eingeben

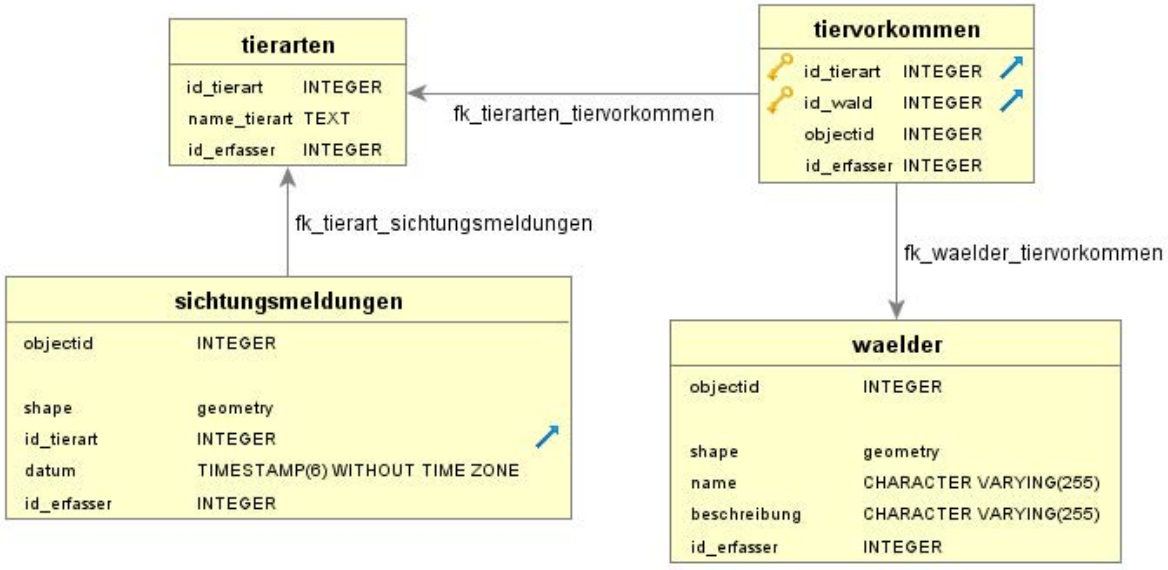


> **Tipp**
>
> Der Name des Schemas muss in zukünftigen Abfragen immer dem Tabellennamen vorangestellt werden, damit klar ist aus welchem Schema abgefragt werden soll.

> **Tipp**
>
> Falls Sie einmal eigene Tabellen erstellen sollten: Es empfiehlt sich alle räumlichen Tabellen direkt in ArcGIS Pro zu erstellen. Dadurch entscheidet ArcGIS Pro selbstständig über die korrekten Datentypen und Geometrietypen. Alle anderen Tabellen sowie Fremdschlüssel usw. können auch in pgadmin erstellt werden. Alle Tabellen wurden im ArcGIS Pro Schema "arcgispro_editor" erstellt.

48.4. Übung 5: Datenbankschema prüfen

Für die kommenden Übungen nutzen wir die Datenbank "waelder" auf unserem Server. Den Zugriff haben wir in den vorigen Übungen eingerichtet. Um einen Überblick zu bekommen, lesen Sie das folgende Datenbankschema und machen sich mit den Tabellen, Attributen und Abhängigkeiten vertraut. Da Sie jetzt alle gemeinsam dieselben Tabellen bearbeiten, gibt es in jeder Tabelle jeweils ein Feld "id_erfasser". Sobald Sie Tabellen mit Daten befüllen, schreiben Sie bitte immer jeweils Ihre studentNr in das Feld "id_erfasser", damit die Einträge später unterscheidbar sind.



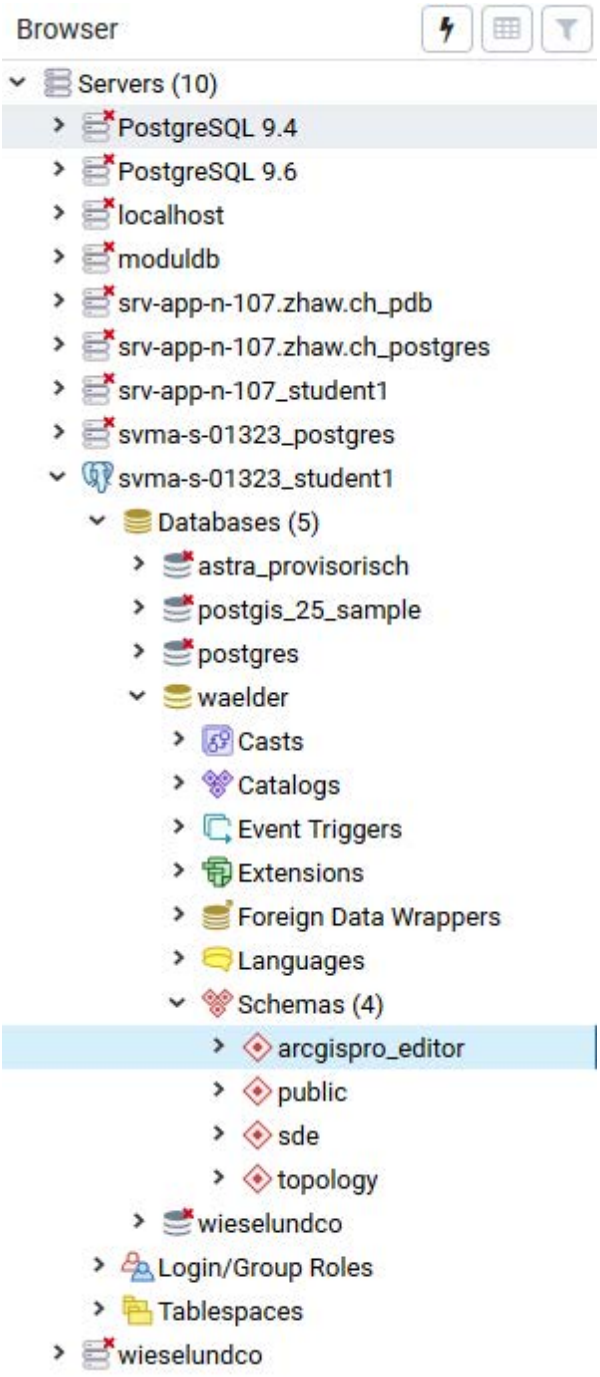


Abbildung 48.6.: PG Admin Schemas

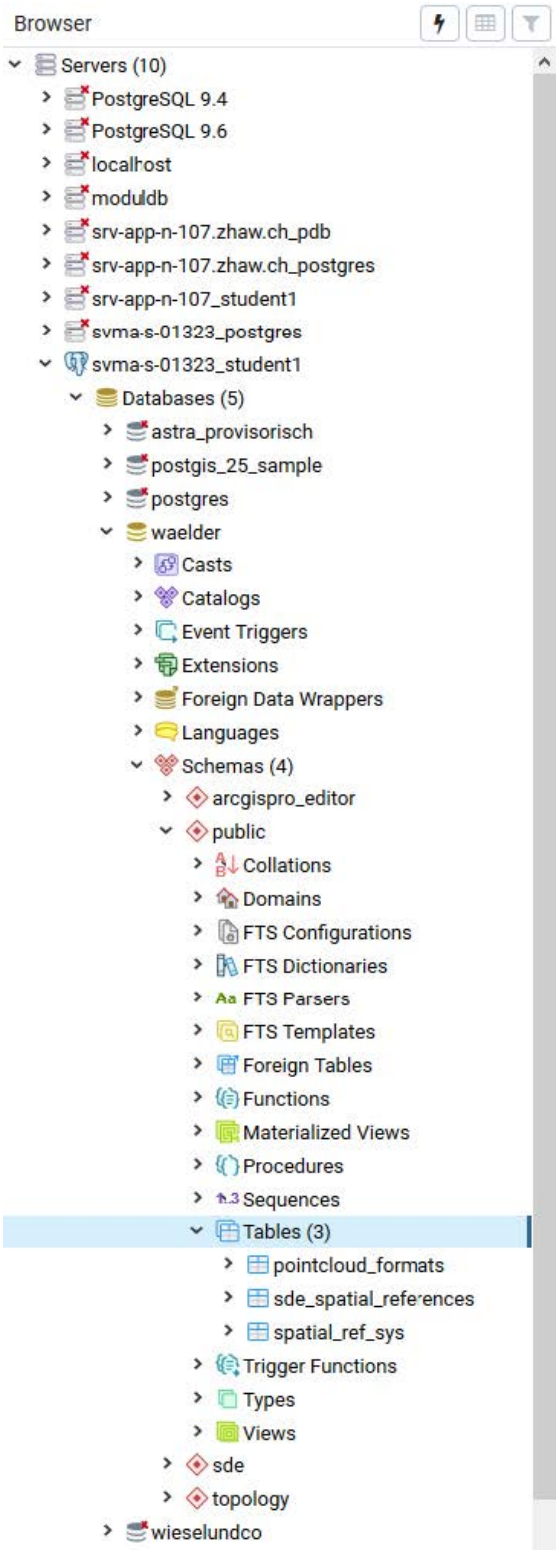


Abbildung 48.7.: PG Admin Tables

48.5. Übung 6: Datenimport in die Datenbank über pgAdmin

Da jetzt alle Tabellen vorliegen sollen diese noch mit ein paar Beispiel-Daten gefüllt werden. Auch hier bietet es sich an räumliche Objekte (Geometrien) mit Hilfe von ArcGIS Pro zu erfassen. Nicht-räumliche Objekte können meist sogar schneller über SQL importiert werden.

1. Suchen Sie in der Datenbank "waelder" das schema "arcgispro_editor".
2. Öffnen Sie anschliessend den SQL Editor von pgadmin. Sie finden das so genannte Query Tool im Kontextmenü des Schemas oder in der Menüleiste oben unter Tools→Query Tool
3. Importieren Sie mit Hilfe eines SQL Statements eine Tierart in die Tabelle "tierarten".
4. Da Sie jetzt alle gemeinsam und gleichzeitig Daten in dieselbe Tabelle einfüllen und Sie die Einträge noch unterscheiden wollen, fügen Sie unter id_erfasser jeweils ihre zugewiesene studentNr (nur die Zahl) ein.
5. Benutzen Sie dazu das INSERT Statement und tippen es in das Query Tool (siehe Abbildung 48.8)

    - ```
      INSERT INTO arcgispro_editor.tierarten (id_tierart, name_tierart, id_erfasser)
      Values (DEFAULT, 'Bär', studentNr);
      ```

    - Schliessen Sie die Eingabe mit Klick auf Execute ab.

> **Tipp**
>
> DEFAULT ist ein SQL Befehlswort, welches den Zähler für die id_tierart automatisch auf den nächsten freien Wert setzt. So kann die id_tierart automatisch fortlaufend hochnummeriert werden. Sie dürfen gern ein eigenes Tier in die Datenbank einfüllen.
> Achten Sie darauf, dass Text immer in einfache Anführungszeichen (Das Zeichen auf der ?- Taste) gesetzt werden muss. Bei Copy & Paste aus Dateien wird oft das Anführungszeichen verändert.Beachten Sie ausserdem die Punktschreibweise, dabei muss der Name des Schemas "arcgispro_editor" immer mit Punkt verbunden vor dem Namen der Tabelle angegeben werden. Jedes Statement wird mit Semikolon abgeschlossen.

6. Lassen Sie sich den Inhalt der Tabelle ausgeben. Dies können Sie über ein SELECT SQL Statement ausführen (siehe Abbildung 48.9).

    - ```
      SELECT * FROM arcgispro_editor.tierarten;
      ```

7. Sie können erkennen, dass ihr Tier mit Ihrer studentNr als id_erfasser in die Tabelle eingetragen wurde. Vermutlich sehen Sie auch bereits Einträge Ihrer Mit-Studierenden.
8. Navigieren Sie im Object Explorer von pgAdmin zur Tabelle "tierarten".
9. Über das Kontextmenü können ebenfalls die Inhalte der Tabelle angezeigt werden (siehe Abbildung 48.10).

    a. Tabelle "tierarten"→Kontextmenü→View/Edit Data→All Rows

10. Es öffnet sich automatisch das Query Tool und die Inhalte der Tabelle werden angezeigt.

> **Tipp**
>
> Auf diese Weise kann immer nur die komplette Tabelle angezeigt werden. Filter oder sonstige Einschränkungen müssen als SQL Statement abgesetzt werden.

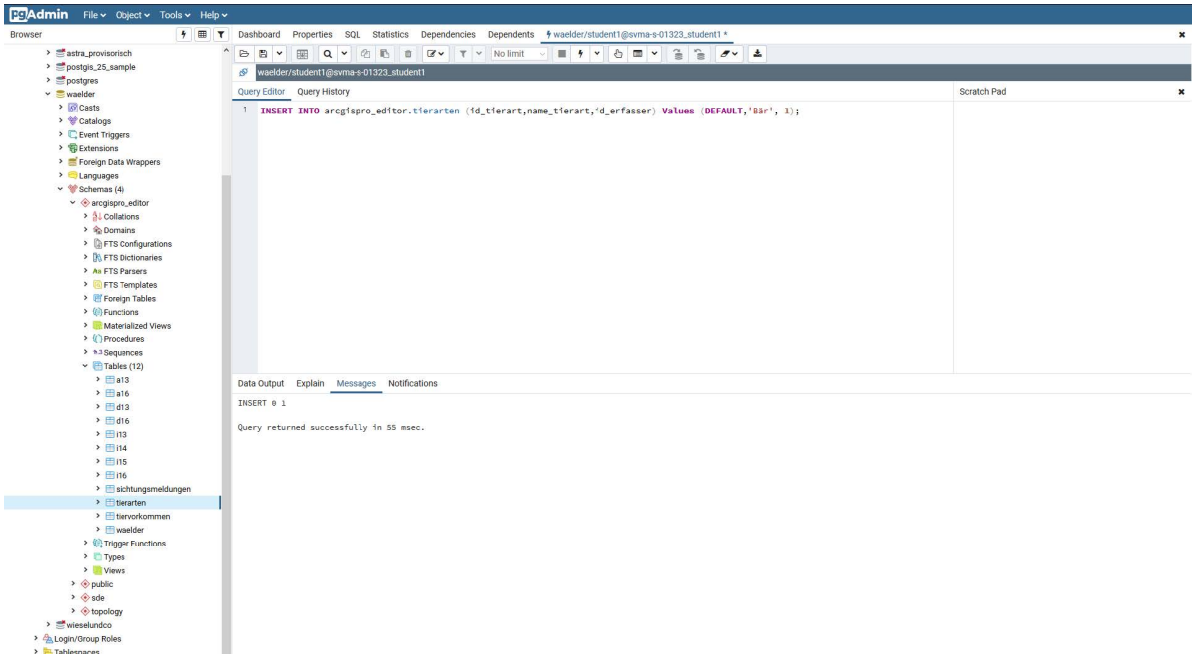


Abbildung 48.8.: Insert Statement

48.6. Übung 7: Datenimport in die Datenbank mit einer SQL-Datei

Jedes SQL Statement kann in einer Datei abgespeichert werden. Dadurch lassen sich bequem z.B. gleich mehrere Daten in die Datenbank einlesen.

1. Bleiben Sie in der Datenbank “waelder” im schema “arcgispro_editor”.
2. Öffnen Sie das Query Tool.
3. Laden Sie die Datei “tierarten.sql” aus dem Moodle und speichern Sie in Ihrem Ordner.
4. Öffnen Sie die Datei in einem beliebigen Texteditor und ändern das Wort “studentNr” in Ihre studentNr ab. Denken Sie sich ausserdem ein paar eigene/andere Tierarten aus und schreiben diese ebenfalls in die Datei (siehe Abbildung 48.11).
5. Über das kleine Ordner Symbol (Im Menü des Query Tools links oben) kann eine SQL-Datei geladen werden. Laden Sie die geänderte “tierarten.sql” Datei in pgAdmin.

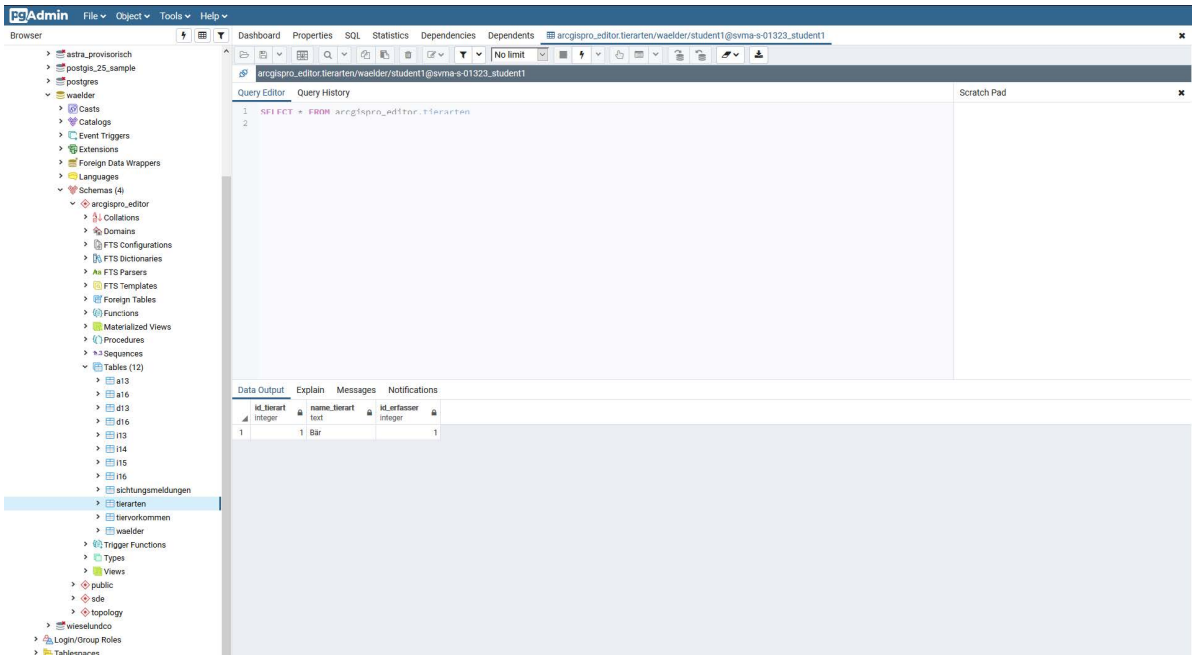


Abbildung 48.9.: Select Statement

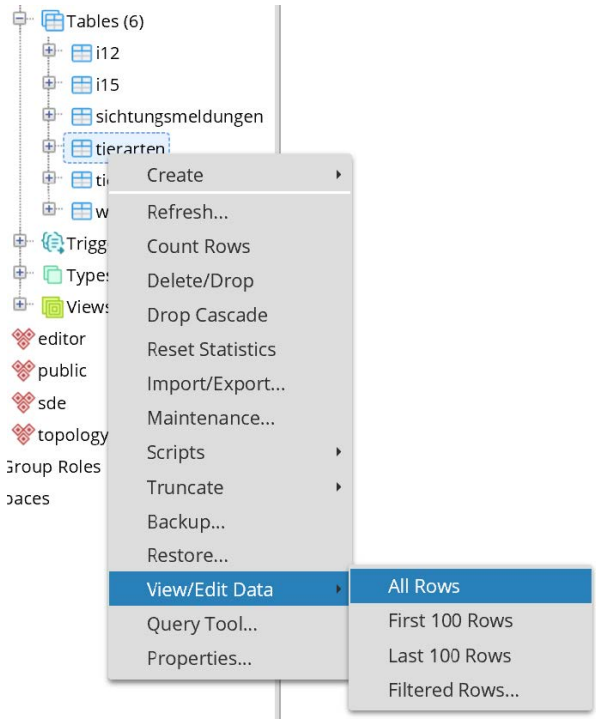


Abbildung 48.10.: Überprüfen der Tabelleninhalte

6. Die Auswahl lädt den Inhalt der Datei ins Query Tool (siehe Abbildung 48.12).
7. Schliessen Sie die Eingabe mit Klick auf Execute ab.
8. Schauen Sie sich erneut den Inhalt der Tabelle “tierarten” an. Die Tabelle sollte jetzt die eigenen Tierarten plus ein paar Einträge Ihrer Mit-Studierenden enthalten.

> **Tipp**
>
> Da der Primärschlüssel der Tabelle nur auf id_tierart gesetzt ist und wir alle zusammen auf derselben Tabelle arbeiten, gibt es einige Tierarten bestimmt doppelt. Das ist normal.

```
INSERT INTO arcgispro_editor.tierarten (id_tierart,name_tierart,id_erfasser) Values
(DEFAULT,'Tierart1',studentNr),
(DEFAULT,'Tierart2',studentNr),
(DEFAULT,'Tierart3',studentNr),
(DEFAULT,'Tierart4',studentNr),
(DEFAULT,'Tierart5',studentNr);
```

```
INSERT INTO arcgispro_editor.tierarten (id_tierart,name_tierart,id_erfasser) Values
(DEFAULT,'Reh',1),
(DEFAULT,'Schwein',1),
(DEFAULT,'Fuchs',1),
(DEFAULT,'Dachs',1),
(DEFAULT,'Eichhörnchen',1);
```

Abbildung 48.11.: Insert statments

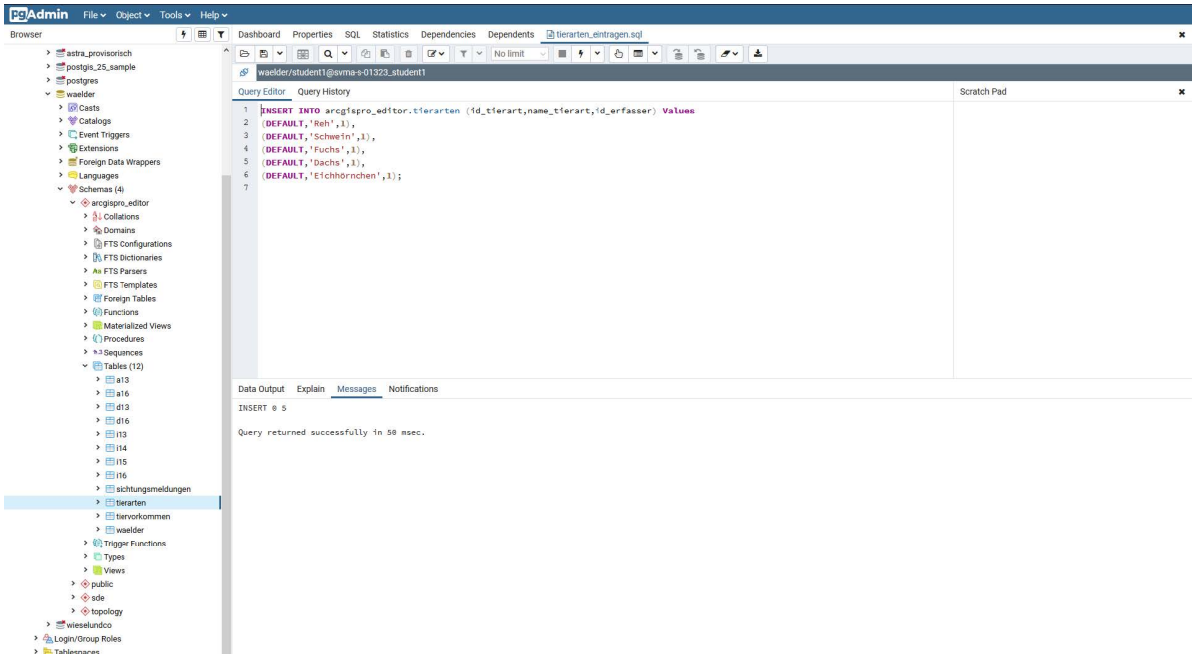


Abbildung 48.12.: Query tool

48.7. Übung 8: Geometrieimport in die Datenbank

Herkömmliche GIS Methoden zum Erstellen und Importieren von Geometrien ins GIS sollten Ihnen aus dem GIS Basic Modul bereits bekannt sein. Auch in ArcGIS Pro können Geometrien z.B. über ein Shapefile in eine Geodatenbank importiert werden. Ausserdem können Geometrien z.B. aus einem Luftbild digitalisiert und direkt in der Geodatenbank gespeichert werden.

1. Laden Sie die Datei "waelder.zip" aus dem Moodle und entpacken Sie die Dateien. Es handelt sich um ein Shapefile mit einigen Wäldern.
2. Starten Sie ArcGIS Pro und laden das Projekt "Datenbankzugriff".
3. Fügen Sie das Shapefile ein und erfassen Sie mit dem Digitalisierungsmöglichkeiten von ArcGIS Pro zusätzlich ein paar eigene fehlende Wälder.
4. Öffnen Sie die Attributtabelle.
5. Erfassen Sie im Feld "id_erfasser" wiederum Ihre zugewiesene studentNr (Nur die Zahl). Alle weiteren Attribute können Sie beliebig erfassen. Denken Sie sich einfach etwas aus.
6. Importieren Sie die Daten anschliessend folgendermassen in die Feature Class "waelder" in der Datenbank auf dem Server.
7. Kontextmenü der Feature Class "waelder"→Load Data. Es öffnet sich das Werkzeug "Append" (siehe Abbildung 48.13)
   - Input Dataset: Shapefile "waelder"
   - Target Dataset: waelder.arcgispro_editor.waelder
   - Field Matching Type: Use the Field Map to reconcile schema differences. Shapefiles können nur Spaltenbezeichnungen mit 10 Zeichen speichern. Alles andere wird abgeschnitten. Das Attribut "beschreibung" ist deshalb in der Datenbank als "beschreibung" gespeichert, im Shapefile aber nur als "beschreibu". Dies hat zur Folge, dass das Schema beider Feature Classes nicht gleich ist. Aus diesem Grund muss hier explizit die Übereinstimmung der Spalten bestimmt werden. Das gleiche gilt für das Feld "id_erfasser" bzw. "id_erfasse" (siehe Abbildung 48.14).
8. Erstellen Sie eine neue Karte in ArcGIS Pro und laden Sie die Feature Class "waelder" aus der Server Datenbank.
9. Sie haben jetzt die Wälder in die Datenbank importiert.
10. Schauen Sie sich die Wälder auch in pgAdmin an. Machen Sie hierzu eine Select Abfrage im Query Tool. In der Ergebnis Tabelle sehen Sie in der Geometry-Spalte ein kleines Karten-Symbol (View all Geometries in this column). Dies öffnet den "Geometry Viewer". Sie sollten auch dort die Geometrien der importierten Wälder sehen (siehe Abbildung 48.15).

Tipp

Da wir jetzt alle unsere Wälder in dieselbe Datenbank laden, sind natürlich einige Wälder doppelt vorhanden. Dies ist für unser Vorhaben aber nicht weiter schlimm.

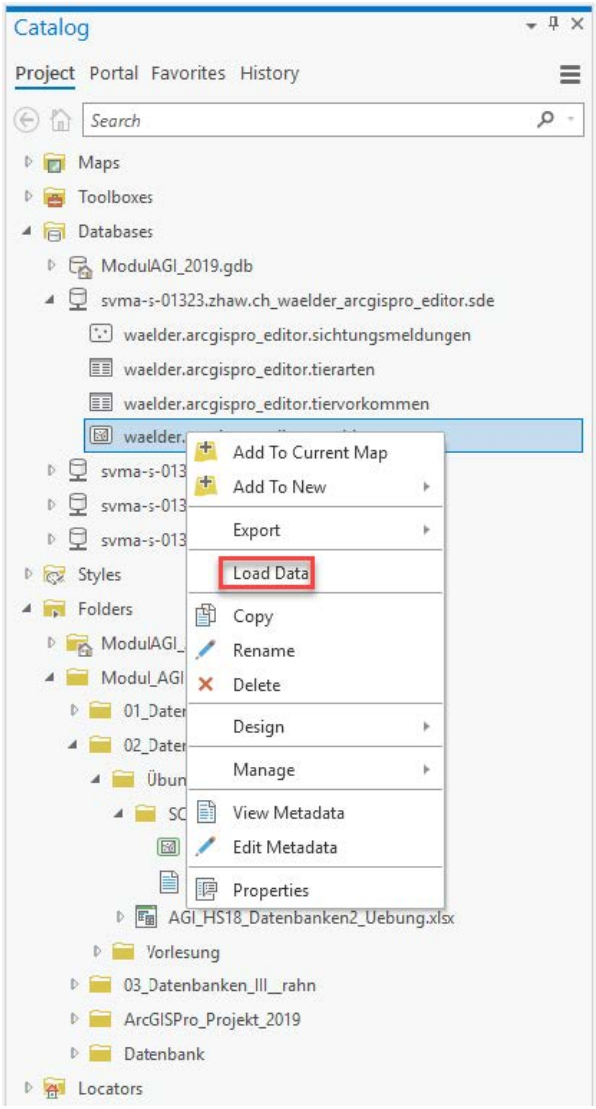


Abbildung 48.13.: Load data

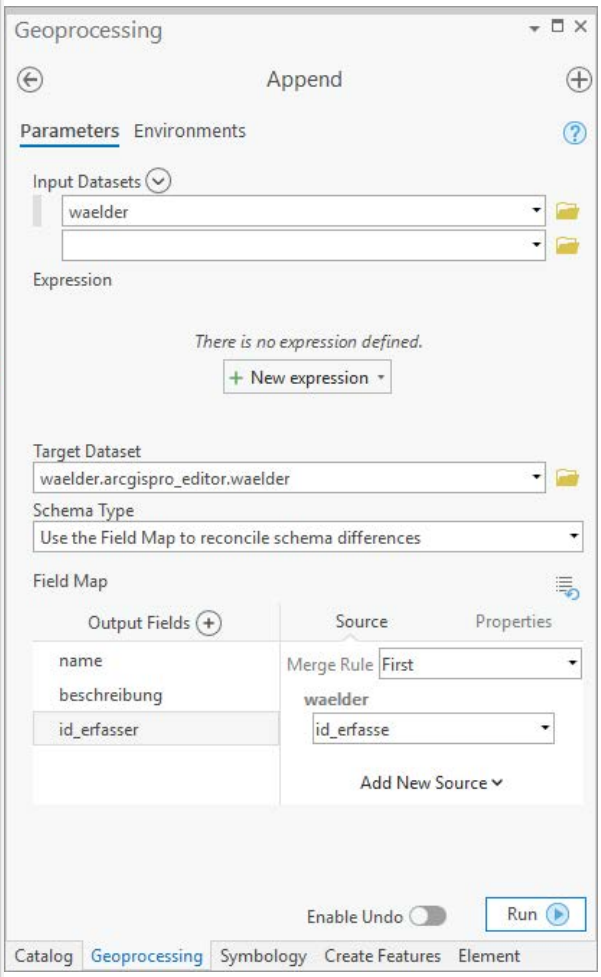


Abbildung 48.14.: Geoprocessing Append

Abbildung 48.15.: GIS GUI mit Wälder

48.8. Übung 9: Geometrieimport in die Datenbank mit SQL-Datei

Auch Geometrien können ebenso wie Sachdaten über eine SQL-Datei in die Datenbank kopiert werden.

1. Laden Sie die Datei “sichtungsmeldungen.sql” aus dem Moodle. Es handelt sich um eine SQL-Datei mit Sichtungsmeldungen von Tieren.
2. Öffnen Sie die Datei in einem beliebigen Texteditor. Was sehen Sie? Was fällt auf?
3. Sie werden bemerken, dass die Geometrie über die Funktion ST_GeomFromText lesbar dargestellt werden kann. Der Geometrietyp von ArcGIS Pro wird dann über diese Funktion automatisch umgewandelt. I.d.R. ist es aber sehr viel einfacher Geometrien über ArcGIS Pro zu importieren oder zu erstellen. Sie sollen hier aber einmal sehen, dass es auch direkt über SQL geht.
4. Die Attribute (Spalten in der DB) in der Datei sind Kommagetrennt in Klammern nach dem Wert VALUE. Die Reihenfolge ist diesselbe wie in der Klammer vor dem VALUE.
5. Passen Sie die Datei an, indem Sie als objectid (erstes Attribut) ihre studentNr gefolgt von einer 0 eintragen und als id_erfasser (letztes Attribut) jeweils Ihre studentNr (Nur die Zahl ohne 0) eintragen. Denken Sie sich auch eine sinnvolle Schweizer-Koordinate in der Nähe der anderen Objekte aus, damit Sie nicht alle denselben Punkt importieren. Die Punkte dürfen, müssen aber nicht im Wald liegen. Variieren Sie etwas. Sie können eine Koordinate einfach im Map-Fenster von ArcGIS Pro ablesen und ebenfalls in der Datei anpassen. Es kommt jetzt hier nicht auf Genauigkeit der Lage des Punktes an. Über die id_tierart können Sie sich ein beliebiges Tier aussuchen. Tragen Sie eine vorhandene id_tierart aus Ihrer Tierarten Tabelle aus Übung 6 ein. Passen Sie zum Schluss auch noch das Datum an (siehe Abbildung 48.16).
6. Starten Sie pgAdmin und laden die Datenbank “waelder” im Schema “arcgispro_editor”
7. Importieren Sie die Sichtungsmeldung mit Hilfe der SQL-Datei in die Datenbank.
8. Schauen Sie sich die Daten in pgadmin und in ArcGIS Pro an. Sie sollten einige Punkte in der Karte sehen (siehe Abbildung 48.17 und Abbildung 48.18).

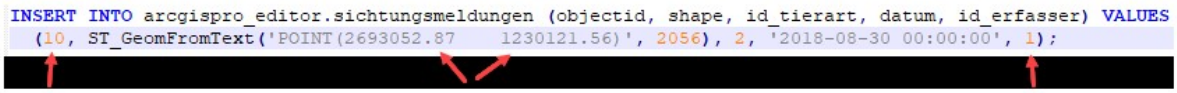


Abbildung 48.16.: Anpassungen INSERT Statement

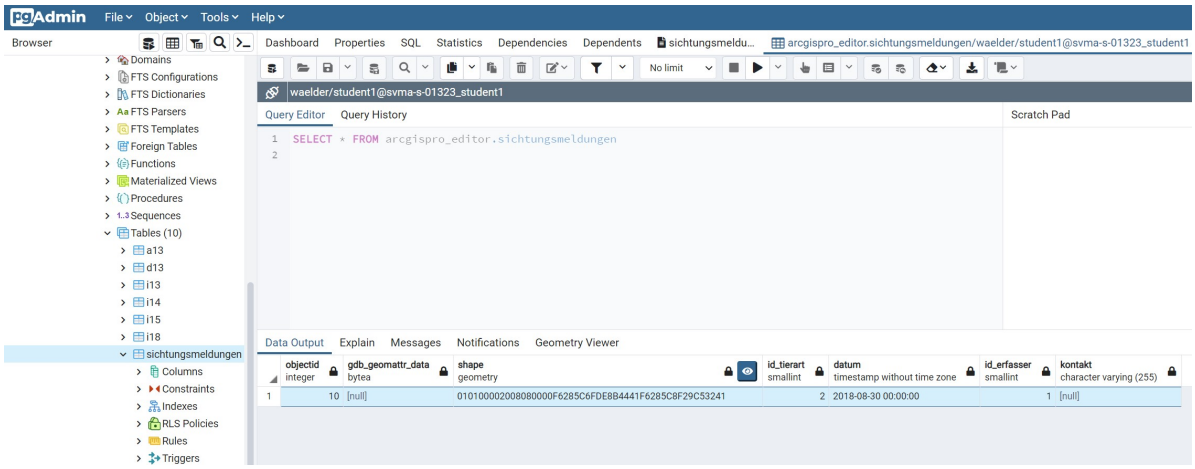


Abbildung 48.17.: pgAdmin Select Statement

Abbildung 48.18.: GIS GUI mit Wälder und Sichtungen

49. Versionierung und Mehrbenutzerbetrieb in ArcGIS Pro

49.1. Übung 10: Version erstellen in ArcGIS Pro

Die Nutzung von Enterprise Server Geodatenbanken wie z.B. PostgreSQL erlaubt einen Mehrbenutzerbetrieb derselben Datenbank. In den vorigen Übungen haben wir jeweils alle Einträge in die Datenbank mehr oder weniger gleichzeitig durchgeführt, ohne auf eine Version zu achten. In solchen Fällen gilt für die Datenbank, dass der letzte Eintrag einen vorangehenden Eintrag überschreiben würde. Macht ein Benutzer Änderungen an einem Objekt, z.B. einem Polygon und gleichzeitig ein anderer Benutzer ebenfalls Änderungen am selben Polygon, so gilt jeweils die letzte Eintragung als aktuell. Alle vorigen Anpassungen werden überschrieben. Um dies zu verhindern, braucht es einen Mechanismus, welcher diese Mehrfachänderungen im Mehrbenutzerbetrieb managen kann. Hierzu wird Versionierung verwendet.

1. Kehren Sie in das Projekt “Datenbankzugriff” in ArcGIS Pro zurück und verbinden zur Datenbank “waelder” mit Ihrem zugewiesenen Benutzer (studentNr).
2. Zeigen Sie die Sichtungsmeldungen aus der Datenbank in der Karte an.
3. Starten Sie jetzt den Editor und erfassen ein paar beliebige Sichtungsmeldungen. Erfassen Sie auch die Attribute über die Attributtabelle. Speichern Sie anschliessend Ihre neuen Punkte.
4. Wechseln Sie anschliessend die Ansicht im Contents-Fenster von “List By Drawing Order” auf “List By Data Source” (siehe Abbildung 49.1).
5. Schauen Sie sich die Verbindung zum Layer im Contents-Fenster an. Sie sehen dort zu Anfang “sde.DEFAULT”. Dies ist wie der Name vermuten lässt die Standard-Version in welche automatisch alle Änderungen usw. eingespielt werden, sofern keine weiteren Versionen vorhanden sind. “student1@...” zeigt hier auch noch den verbundenen Benutzer. (siehe Abbildung 49.2).
6. Selektieren Sie die Verbindung und achten auf die Registerkarten im ArcGIS Pro Hauptmenü (ganz oben). Es wird ein neues Register “Versioning” angezeigt.
7. Hier sehen Sie alle Werkzeuge zur Verwaltung der Versionen. Erstellen Sie eine neue eigene Version durch Klick auf “New Version”. Achten Sie darauf das Häkchen bei “Change to this new version” zu setzen, damit die neue Version aktiviert wird (siehe Abbildung 49.3).

a. Name: studentNr_name
b. Description: beliebiger Text
c. Access Permission: Protected
    - Private: Alle Änderungen in dieser Version sind privat, d.h. kein anderer Benutzer kann diese Version sehen und/oder benutzen.
    - Protected: Alle Änderungen sind privat und niemand kann in dieser Version Daten bearbeiten. Andere Benutzer können aber die Version anzeigen und Änderungen sehen.
    - Public: Öffentliche Nutzung, d.h. auch andere Benutzer können zu dieser Version wechseln und Daten bearbeiten

8. Erstellen Sie die neue Version mit "OK".
9. Schauen Sie wiederum die Datenbankverbindung im Contents-Fenster an. Die neue Version student1 ist jetzt aktiv. Ab sofort werden alle Änderungen innerhalb dieser Version gemacht und nicht mehr in der sde.DEFAULT Version (siehe Abbildung 49.4).
10. Starten Sie den Editor und löschen einige der bereits von Ihnen erfassten Sichtungsmeldungen und erfassen auch ein paar neue Punkte. Speichern Sie die Änderungen im Editor.
11. Merken Sie sich ungefähr die Lage der von Ihnen jetzt digitalisierten Punkte (siehe Abbildung 49.5).
12. Wechseln Sie jetzt zurück auf die DEFAULT Version. Dies können Sie über das Kontextmenü des Eintrags im Contents-Fenster→Change Version. Alternativ kann die Version auch in der Registerkarte "Versioning" gewechselt werden.
13. Wechseln Sie die Version zurück auf die sde.DEFAULT Version.
14. Schauen Sie sich das Kartenbild an. Was fällt auf?
15. Alle Ihre Anpassungen/Änderungen, welche Sie in der Version studentNr gemacht haben, sind in der sde.DEFAULT nicht mehr vorhanden. Dies ist normal, da die Änderungen eben in einer anderen Version gemacht wurden (siehe Abbildung 49.6 und Abbildung 49.7).
16. ACHTUNG: Es kann natürlich sein, dass Ihre Mit-Studierenden etwas schneller waren und bereits Änderungen auch in die sde.DEFAULT eingespielt haben und Ihr Kartenbild jetzt deshalb trotzdem anders aussieht als vorher.
17. Speichern Sie Ihr ArcGIS Pro Projekt.

49.2. Übung 11: Versionen zusammenführen in ArcGIS Pro

In vielen Anwendungsfällen bearbeiten viele Mitarbeiter die Daten in derselben Datenbank. Z.B. können unterschiedliche Mitarbeiter jeweils einen Teil eines Untersuchungsgebiets kartieren. Durch die Versionierung können alle Arbeiten unabhängig voneinander auf derselben Datenbank durchgeführt werden. Zu einem gewissen Zeitpunkt, spätestens nach Abschluss der Arbeiten sollten alle Arbeiten aber wieder in einer gemeinsamen Version zusammengeführt werden.

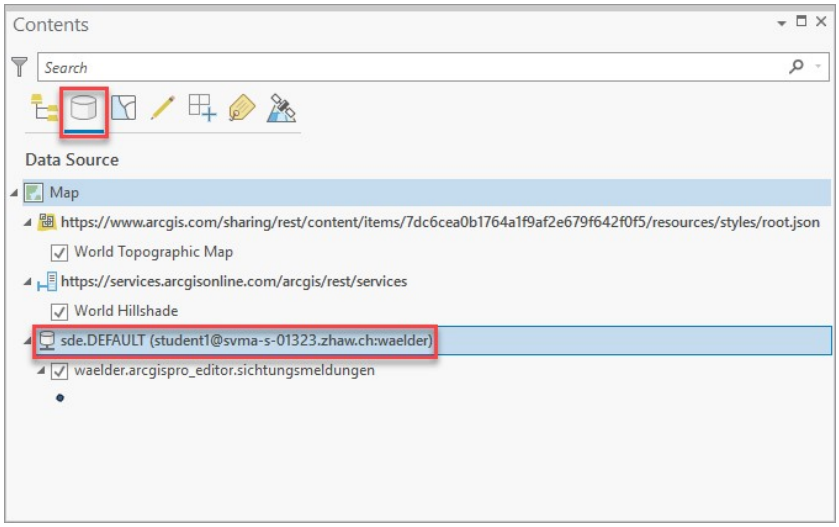


Abbildung 49.1.: Data Source in Contents-Fenster

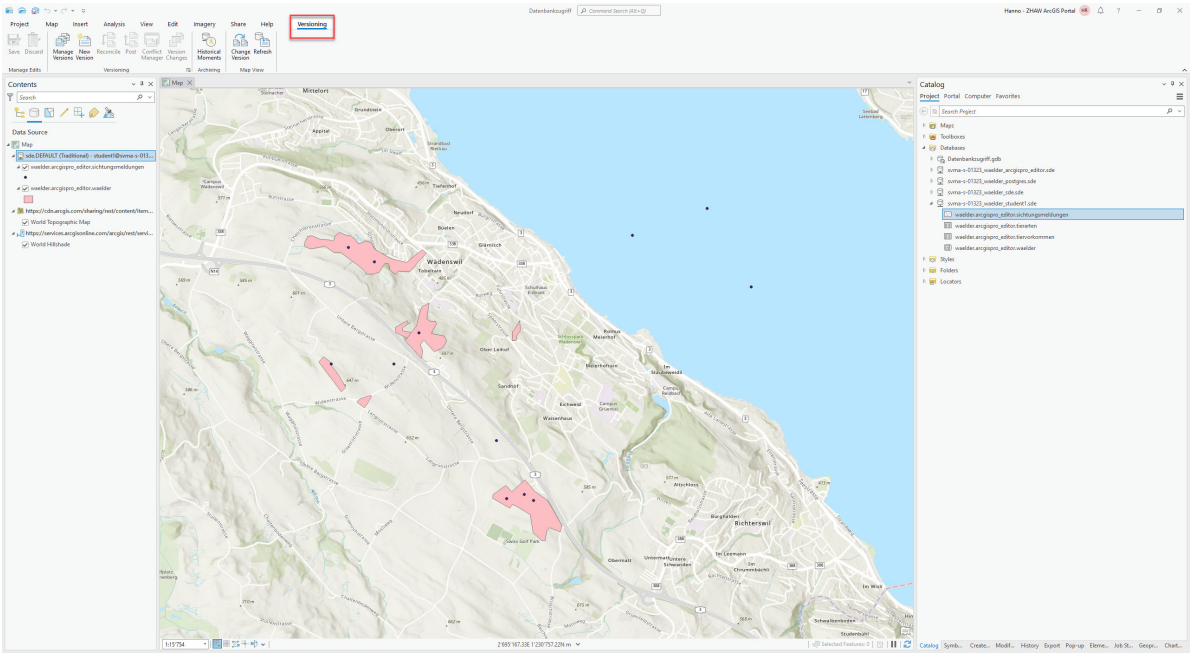


Abbildung 49.2.: ArcGIS Pro Versionierung

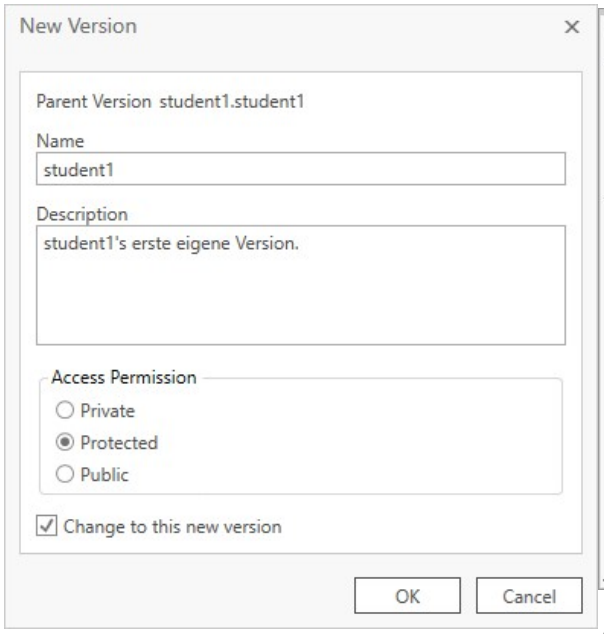


Abbildung 49.3.: ArcGIS neue Version erstellen

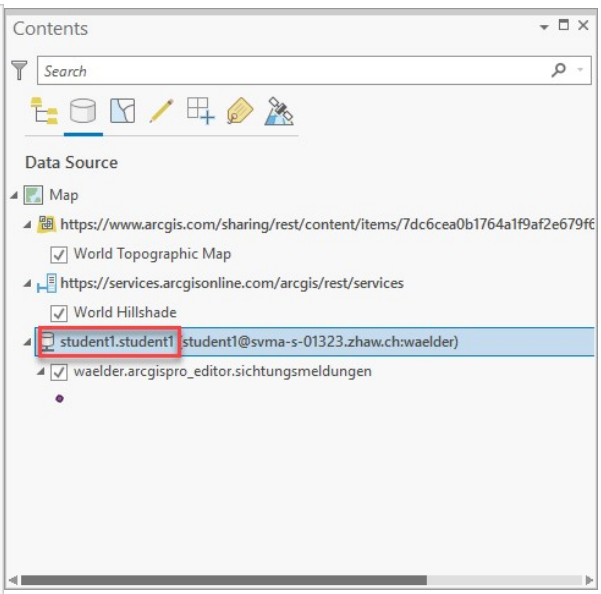


Abbildung 49.4.: Neue Version in Data Source in Contents-Fenster

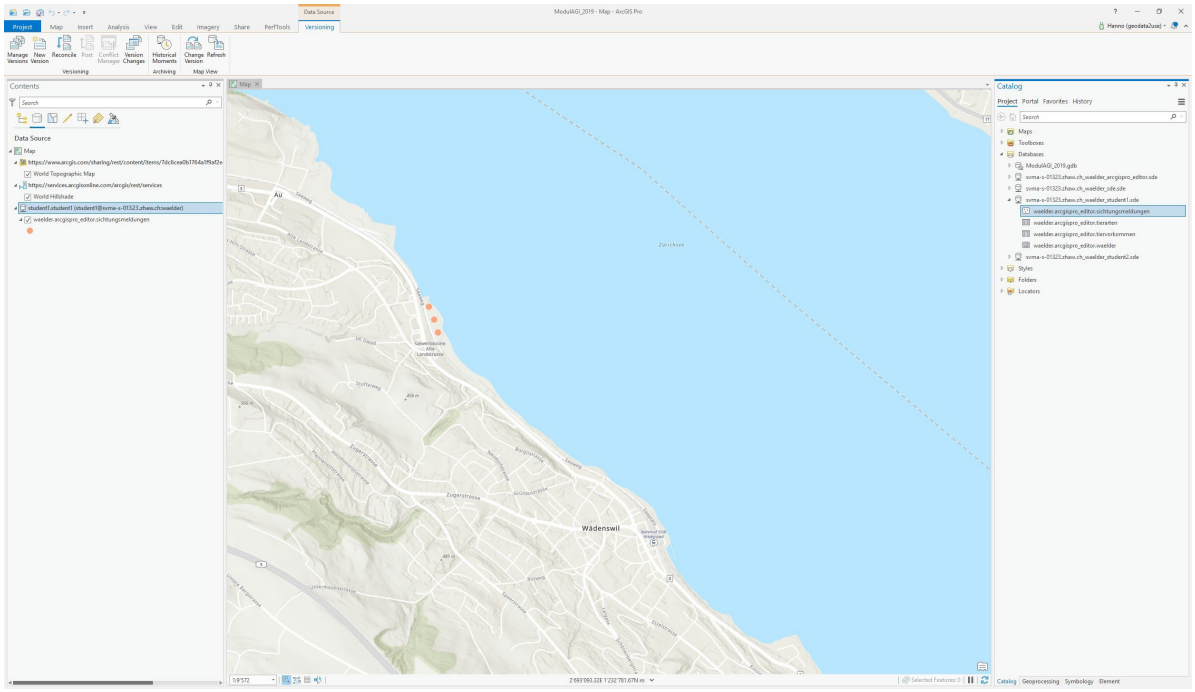


Abbildung 49.5.: Neue Punkte in ArcGIS GUI

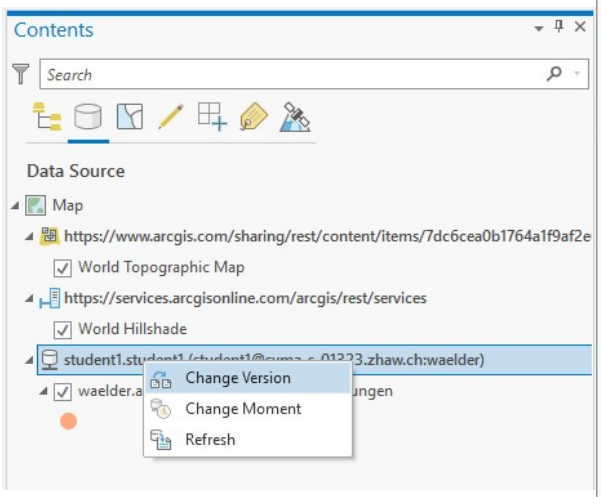


Abbildung 49.6.: Ändere Version in Contents-Fenster

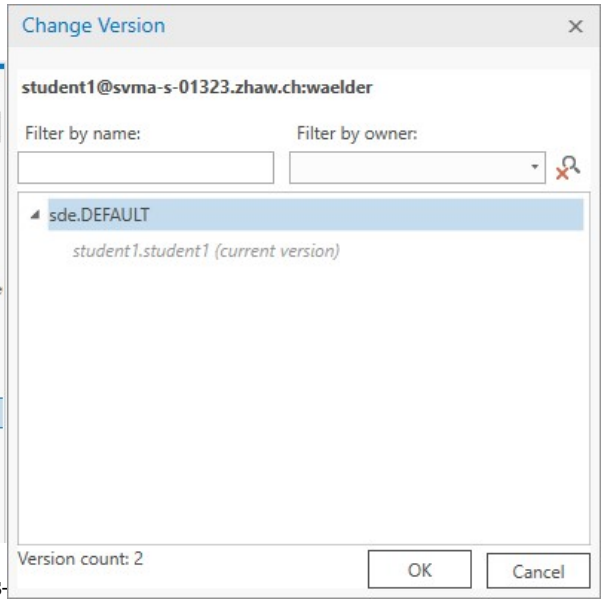


Abbildung 49.7.: Ändere Version in Contents-Fenster

1. Bleiben Sie im Projekt “Datenbankzugriff” in ArcGIS Pro und in der Datenbank “waelder” mit Ihrem zugewiesenen Benutzer (studentNr).
2. Wechseln Sie wiederum auf die “List By Data Source” Ansicht im Contents-Fenster.
3. Schauen Sie sich den Versionsmanager an. Sie öffnen diesen über die Registerkarte “Data Source Versioning” über die Schaltfläche “Manage Versions” (siehe Abbildung 49.8).
4. Hier können Sie alle vorhandenen (sofern berechtigten) Versionen einsehen und verwalten.
5. Da Sie Ihre jeweilige Version als “protected” erstellt haben, sollten auch die Versionen Ihrer MitStudierenden sichtbar sein (siehe Abbildung 49.9).
6. Schliessen Sie den Versionsmanager wieder.
7. Wechseln Sie anschliessend für den Sichtungsmeldungen Layer die Version. Wählen Sie diesmal die Version eines anderen Teilnehmers z.B. Ihres Tischnachbarn (siehe Abbildung 49.10).
8. Sie sollten im Kartenbild wieder eine andere Version mit anderen Punkten sehen.
9. Starten Sie den Editor und versuchen ein paar neue Punkte zu erfassen.
10. Sie bekommen eine Fehlermeldung (siehe Abbildung 49.11).
11. Wie bereits erwähnt können “fremde” Versionen nicht bearbeitet werden, wenn diese nicht öffentlich sind.
12. Wechseln Sie wieder zurück zu Ihrer eigenen Version “studentNr”.
13. Zeigen Sie die Registerkarte “Versioning” an.
14. Bevor wir unsere eigenen Änderungen zurück in die sde.DEFAULT Version einspielen

können, müssen wir auf Fehler bzw. Versionierungsprobleme prüfen, um sicher zu stellen, dass keine Inkonsistenzen entstehen.

15. Klicken Sie dazu im Menü auf “Manage Versions”. Selektieren Sie Ihre Version und klicken “Reconcile/Post”. Wählen Sie die Zielversion so wie Einstellungen für das Fehlerhandling. Wählen Sie als Zielversion die sde.DEFAULT Datenbank und als Edit versions Ihre zuvor erstellte Version. Setzen Sie noch den Haken bei “Post versions after reconcile”. Lassen Sie ansonsten die Voreinstellungen (siehe Abbildung 49.12).
16. Sollte der Prozess ohne Fehler durchgeführt werden, so können im Anschluss die Änderungen in die sde.DEFAULT Datenbank geschrieben werden.
17. Klicken Sie auf “Ok”. Alle Ihre Änderungen sollten jetzt auch in die sde.DEFAULT Version importiert worden sein.
18. Prüfen Sie dies, indem Sie zurück auf die sde.DEFAULT Version wechseln. Diese sollte jetzt identisch mit Ihrer eigenen Version sein

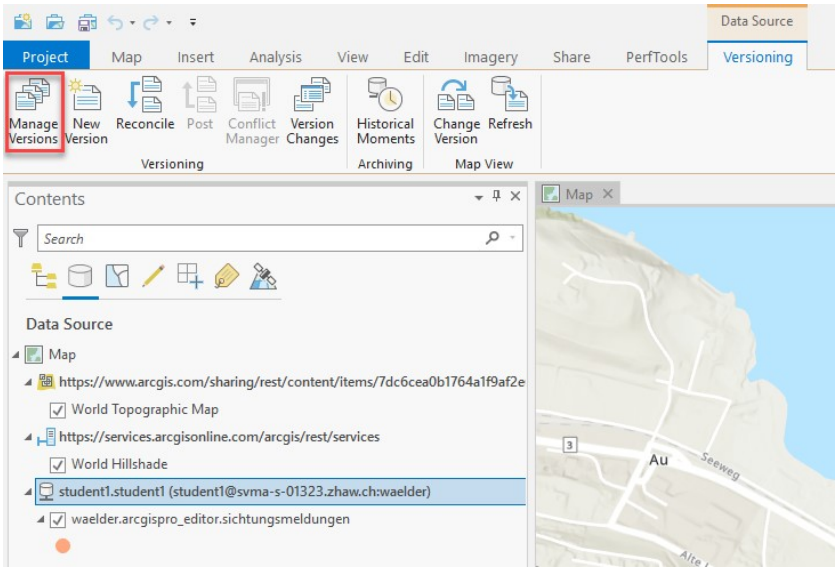


Abbildung 49.8.: “Manage Versions” in Data Source [ArcGIS Pro]

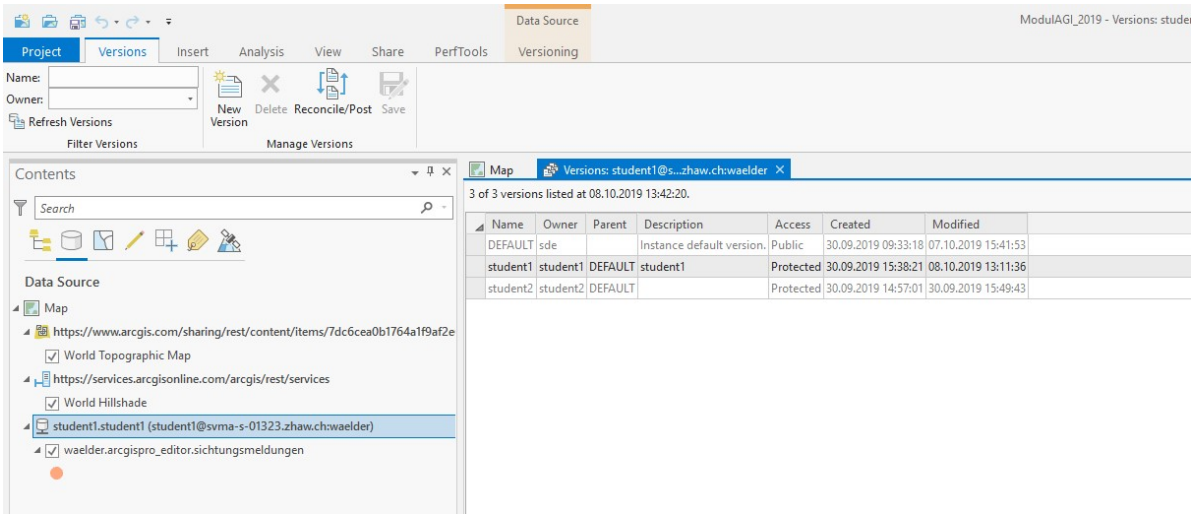


Abbildung 49.9.: Verschiedene Versionen sind verfügbar

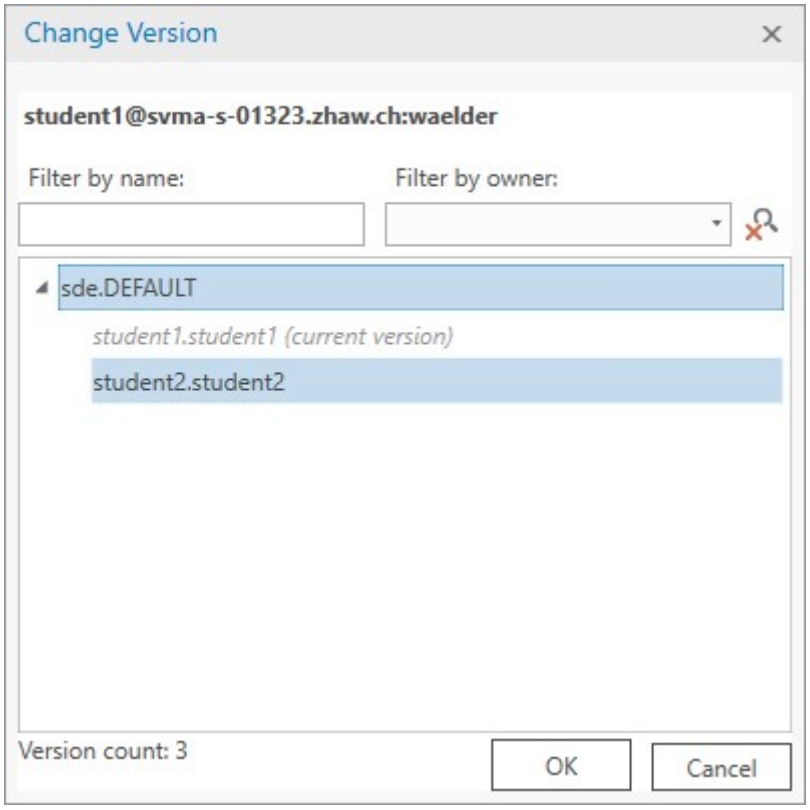


Abbildung 49.10.: Änderung der Version

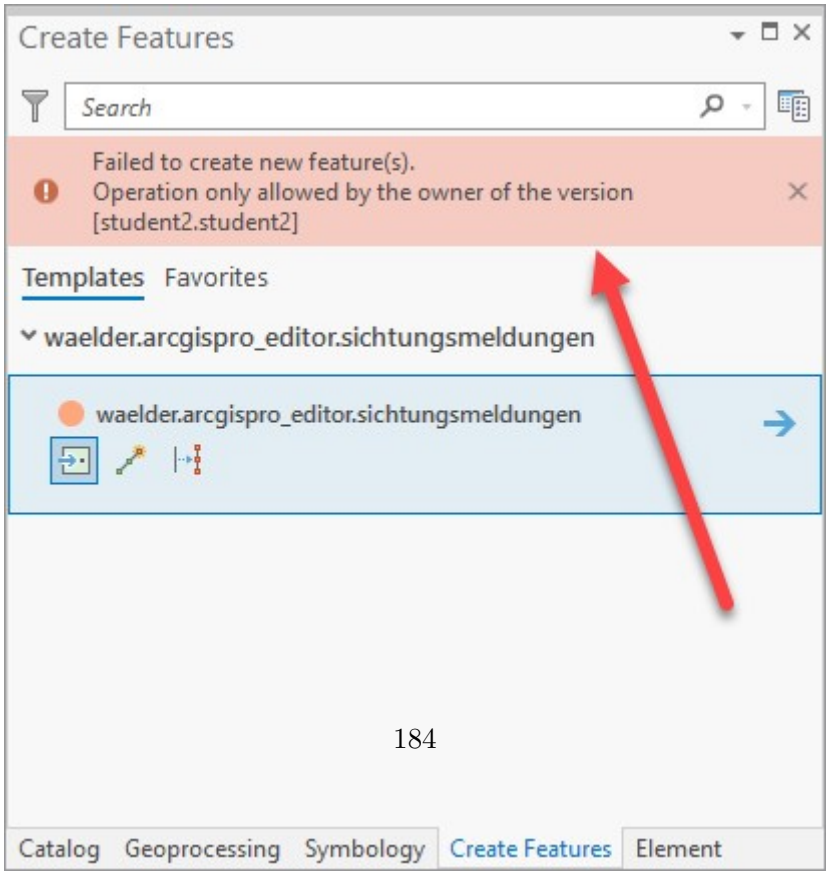


Abbildung 49.11.: Fehlermeldung in ArcGIS Pro

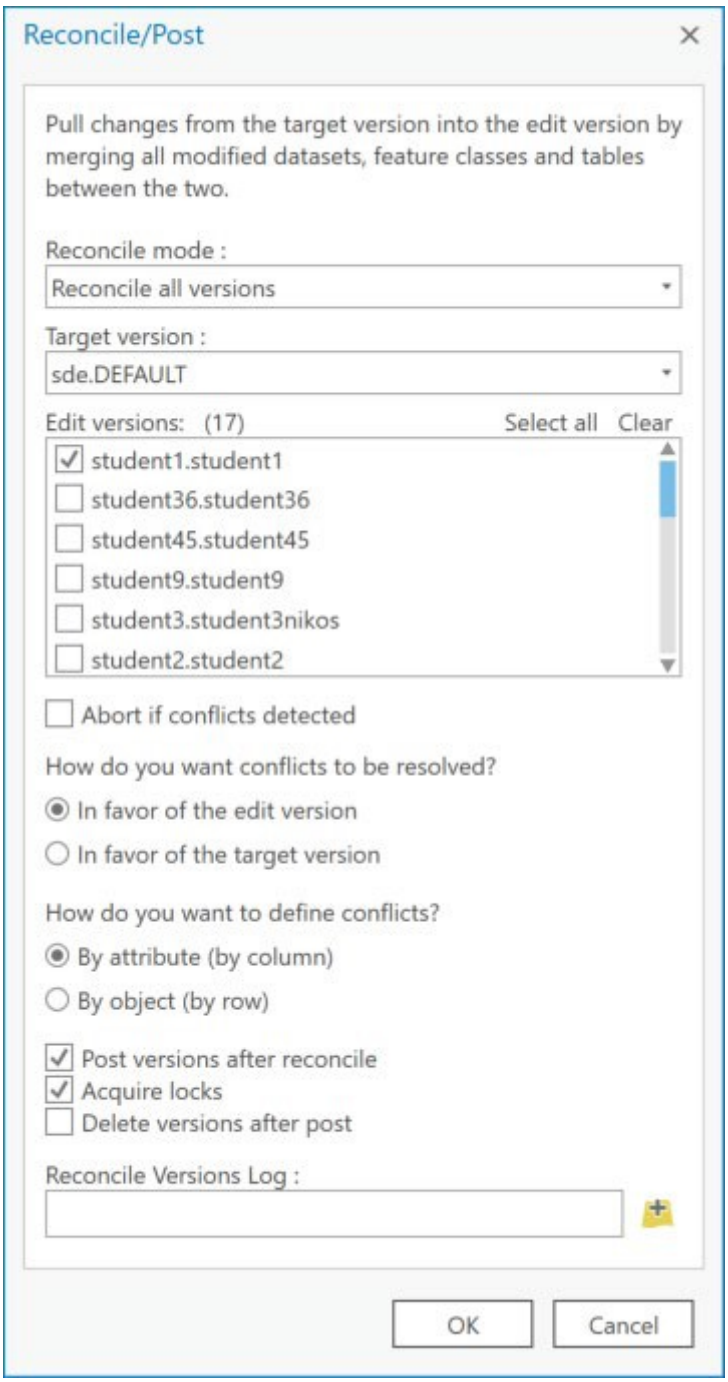


Abbildung 49.12.: Fehlerbehebung mit Reconcile/Post

50. Abfragen in SQL mit pgAdmin und ArcGIS Pro

50.1. (Optional) Übung 12: Abfragen in SQL

Wir haben in der Vorlesung ein paar SQL-Abfragen über die "waelder" Datenbank definiert. Da wir jetzt über die vorigen Übungen ein paar Beispiel Daten in die Datenbank importiert haben, können wir diese Datenbank nutzen, um noch ein paar Abfragen zu üben. Es ist nicht Ziel alle Abfragen gemacht zu haben. Machen Sie einfach so viele Abfragen wie Sie können. Beachten Sie auch, dass es oft mehr als einen Lösungsweg gibt.

1. Starten Sie pgAdmin und verbinden mit der Datenbank "waelder" mit Ihrem zugewiesenen Login (studentNr).
2. Formulieren Sie die folgenden SQL-Abfragen und führen diese über das Query Tool von pgAdmin aus. Schauen Sie sich anschliessend das Ergebnis in der Attributtabelle und im Geometry Viewer an.

- Nicht räumliche Abfragen:

  a. Welche Sichtungsmeldungen wurden von "id_erfasser_x" erfasst?
  b. Wurden Löwen gesichtet? Wenn ja, wie viele?
  c. Wie viele Tiere von jeder Art wurden jeweils gesichtet. Geben Sie eine Liste mit den Namen der Tierarten und der Anzahl der Sichtungen aus.
  d. Wie viele Hasen wurden gesichtet? Geben Sie eine Zahl aus.
  e. Welcher Erfasser hat am meisten Meldungen erfasst? Geben Sie die id_erfasser und die Anzahl der erfassten Tiere aus.
  f. Welcher Erfasser hat am meisten unterschiedliche Tiere erfasst? Geben Sie die id_erfasser und die Anzahl der unterschiedlichen Tiere aus.
  g. In welchen Wäldern kommen Rehe vor? Geben Sie die Namen und die Beschreibung der betroffenen Wälder und die Tierart aus.
  h. Wie viele unterschiedliche Tierarten wurden gesichtet und welche sind das? Zeigen Sie jede Tierart mit dem Datum der letzten Sichtung. SQL→MAX(datum).
  i. Welche Tierarten wurden NICHT gesichtet?
  j. In welchen Monaten wurden die meisten Tiere gesichtet? Geben Sie die drei "besten" Monate für Tiersichtungen aus.

- Räumliche Abfragen:

k. Wie gross sind die erfassten Wälder? Geben Sie eine Liste mit den Waldflächen in qm aus.
   - Funktion: ST_AREA

l. Wie viel Fläche in qm haben alle Wälder zusammen?
   - Funktion: ST_AREA und SUM

m. Welche Sichtungsmeldungen liegen innerhalb eines Waldes?
   - Funktion: ST_CONTAINS oder ST_WITHIN

n. Welche Sichtungsmeldungen liegen ausserhalb des Waldes?
   - Funktion: ST_UNION und ST_CONTAINS

o. In welchem Wald gibt es die meisten Sichtungen?
   - Funktion: ST_CONTAINS und SUM

p. Wie weit sind die ausserhalb liegenden Sichtungsmeldungen vom Wald entfernt?
   - Funktion: ST_DISTANCE, ST_UNION und ST_CONTAINS

q. Welche Tiere wurden ausserhalb vom Wald gesichtet?
   - Funktion: ST_UNION, ST_CONTAINS + JOIN

- Weitere Abfragen:

r. Gibt es Tierarten, welche nur im Wald gesichtet wurden? Welche sind das?
s. Gibt es Tierarten, welche nie im Wald gesichtet wurden? Welche sind das?
t. In welchen Wäldern wurden Füchse gesichtet?
u. In welchen Wäldern kommen Rehe und Füchse gemeinsam vor?
v. Wo gibt es Hermeline aber keine Mauswiesel?

Teil VII.

Datenbanken III

Übungsziele

- Sie lernen die Abfragesprache SQL kennen.
- Sie können Abfragen in SQL formulieren und dabei auch räumliche PostGIS Funktionen nutzen.
- Sie lernen SQL-Abfragen mit ArcGIS Pro Werkzeugen zu visualisieren.

51. Leistungsnachweis

Die Ergebnisse dieser folgenden Übungen sollen als Leistungsnachweis aufbereitet und eingereicht werden. In diesem Fall muss zu jeder Übung im Abschnitt 4 “Geoverarbeitung in der Datenbank mit ArcGIS Pro und SQL” (Übung 7-12), ausgenommen optionale und Wiederholungsübungen, eine kurze schriftliche Dokumentation der Lösung abgegeben werden. Die Aufgabe gilt als Einzelarbeit. Eine gemeinsame Abgabe in Gruppen ist nicht erlaubt. Abgeschriebenes gilt sofort als nicht bestanden.

Die Dokumentation muss Folgendes beinhalten:

- Name und Vorname der Autoren oder Autorinnen.
- Kurze Beschreibung des Lösungsweges inkl. der formulierten SQL-Statements.
- Beschreiben Sie sowohl das Vorgehen in ArcGIS Pro als auch das Vorgehen in pgAdmin. Betrachten Sie dabei insbesondere die Geometrien und die Sachdaten und gehen Sie auf Unterschiede in den Vorgehensweisen und Ergebnissen ein.
- Stellen Sie zum Schluss ArcGIS Pro und SQL/PostGIS gegenüber. Überlegen Sie sich Argumente für und gegen den Einsatz von ArcGIS Pro bzw. von SQL/PostGIS. Wann würden Sie ArcGIS Pro nutzen? Wann eher SQL/PostGIS? Befragen Sie dazu auch einschlägige Literatur und starten Sie eine Web-Recherche. Dokumentieren Sie Ihre Erkenntnisse in ein paar kurzen Sätzen.

51.0.0.1. Abgabeform und -termin:

Reichen Sie Ihre vollständige Dokumentation spätestens bis Freitag 17. November 2023 (2 Wochen nach letztem Input) als PDF über Moodle ein. Ihr findet dies unter der Aufgabenstellung im Abschnitt Geodatenbanken 3.

Abgabe Leistungsnachweis

52. Datenbankverbindung zum Server aufbauen

In den nächsten Übungen wollen wir mit Daten unserer Pflanzendatenbank aus der ersten Datenbank-Lektion vor 2 Wochen umgehen. Dazu versuchen Sie die Daten aus den einzelnen Tabellen zu verknüpfen und sinnvoll aufzubereiten, um Sie anschliessend im ArcGIS Pro einzubinden und zu visualisieren. Unsere Pflanzendatenbank hat hierbei immer den Namen "modulagi_pdb". Verwenden Sie, wenn nicht anders angegeben diese Datenbank.

OPTIONAL: Es ist Ihnen selbst überlassen, ob Sie in den kommenden Übungen die bestehende Server Datenbank nutzen möchten oder Ihre eigene lokale Pflanzendatenbank. Die Übungen beziehen sich auf die Server Variante. Bei eigener Datenbank passen Sie die Datanbank Verbindungsparameter entsprechend an. Wie Sie Ihre eigene lokale Pflanzendatenbank aufsetzen ist in den Vorbereitungsübungen auf Moodle beschrieben.

Unterlagen auf Moodle

52.1. Übung 1: (Wiederholung) Datenbankverbindung zum Server in pgAdmin herstellen

In den folgenden Übungen wollen wir mit einer Datenbank auf einem ZHAW-Server arbeiten. Hierzu erstellen wir zunächst eine Verbindung zu dieser Datenbank über pgAdmin. ACHTUNG: funktioniert nur im ZHAW Netz (VPN).

1. Starten Sie pgAdmin.
2. Im Dashboard klicken Sie auf NEW Server und geben die folgenden Verbindungsparameter ein:
3. Register General: Name: svma-s-01323_modulagi_pdb_studentNr. Verwenden Sie dabei die Nr. Ihres zugewiesenen Benutzers.
4. Register Connection (siehe Abbildung 52.1):
    - Host name/address: svma-s-01323.zhaw.ch
    - Port: 5432
    - Maintenance database: modulagi_pdb
    - studentNr (Verwenden Sie hier Ihren zugewiesenen Anmeldenamen)
    - Passwort: (Verwenden Sie hier Ihr zugewiesenes Passwort)

5. Alle anderen Einstellungen können belassen werden. Speichern Sie mit Klick auf Save.
6. Lassen Sie sich nicht davon verunsichern, dass es bereits einige Einträge hat. Da auf dem Server bereits andere Datenbanken installiert sind, sehen Sie auch diese. Navigieren Sie zur Datenbank “modulagi_pdb” und verschaffen sich einen Überblick über die vorhandenen Tabellen usw..
7. Verwenden Sie für alle folgenden Übungen diese Datenbankverbindung, wenn nichts anderes angegeben.

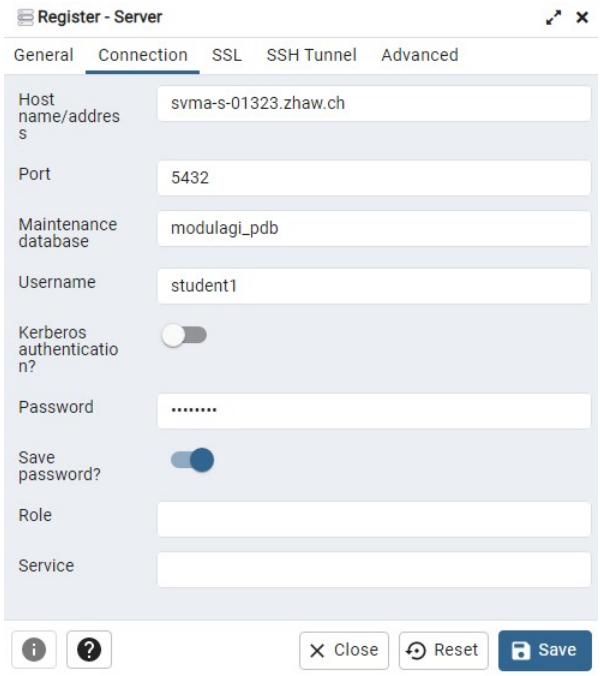


Abbildung 52.1.: Register Connection pgAdmin

52.2. Übung 2: (Wiederholung) Datenbankverbindung zum Server in ArcGIS Pro herstellen

In den folgenden Übungen wollen wir mit einer Datenbank auf einem ZHAW-Server arbeiten. Hierzu erstellen wir zunächst eine Verbindung zu dieser Datenbank über ArcGIS Pro.

1. Starten Sie ArcGIS Pro und erstellen Sie ein neues Projekt “Pflanzendatenbank_III”.
2. Sie können alle folgenden Übungen in diesem Projekt durchführen.
3. Navigieren Sie im Catalog Fenster zum Eintrag Databases. Über das Kontextmenü (Rechtsklick) können Sie eine neue Datenbankverbindung herstellen.
4. Verwenden Sie folgende Parameter (siehe Abbildung 52.2):

- Database Platform: PostgreSQL
- Instance: svma-s-01323.zhaw.ch
- Authentication Type: Database authentication
- User Name: studentNr (Verwenden Sie Ihren zugewiesenen Benutzer)
- Password: (Verwenden Sie ihr zugewiesenes Passwort)
- Database: modulagi_pdb (Achtung, auch hier sehen sie eine Reihe von anderen Datenbanken, Achten Sie darauf die korrekte Datenbank zu wählen)

5. Speichern Sie die Einstellungen mit Klick auf OK und geben der neuen Datenbankverbindung einen sinnvollen Namen.
6. Verwenden Sie für alle folgenden Übungen diese Datenbankverbindung, wenn nichts anderes angegeben.

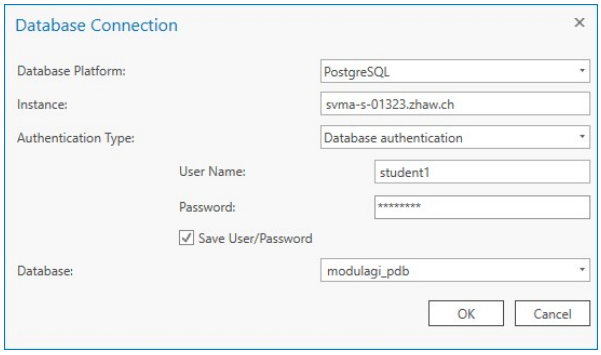


Abbildung 52.2.: Register Connection ArcGIS Pro

53. Abfragen aus der Datenbank in ArcGIS Pro einbinden

In den nächsten Übungen wollen wir mit den Daten in unserer Beispiel-Datenbank umgehen (siehe Abbildung 53.1). Dazu versuchen wir die Daten aus den einzelnen Tabellen zu verknüpfen und sinnvoll aufzubereiten, um Sie anschliessend im ArcGIS Pro einzubinden und zu visualisieren. Unsere Pflanzendatenbank hat hierbei immer den Namen "modulagi_pdb". Verwenden Sie, wenn nicht anders angegeben diese Datenbank. Erstellen Sie zunächst noch einen beliebigen Ordner auf Ihrem Rechner in welchem wir die jeweiligen Ergebnisse speichern können. Sie finden ein Datenbankschema der Pflanzendatenbank auf dem Moodle. Dies hilft Ihnen beim Verständnis der Datenbank und seinen Beziehungen. Nutzen Sie auch die PostGIS Referenz, um die PostGIS Funktionen zu verstehen.

- Datenbankschema
- PostGIS Referenz

Wenn Sie sich bei den SQL-Befehlen nicht ganz sicher sind, so versuchen Sie schrittweise zum Ziel zu kommen. Formulieren Sie zunächst einen einfacheren Teil der Abfrage, z.B. nur Attribute aus einer Tabelle und fügen Sie dann die weiteren Attribute und/oder Tabellen schrittweise hinzu. Probieren Sie bei Bedarf nach jedem Zwischenschritt die Abfrage aus und schauen sich die Ergebnisse an.

53.1. Übung 3: Werkzeug "Make Query Table" anwenden

Das Werkzeug erstellt aus einer beliebigen Datenbankabfrage eine von ArcGIS Pro lesbare temporäre Tabelle. Dies ist immer dann sinnvoll, wenn z.B. nur eine Teilmenge aller Objekte gebraucht wird. Diese Teilmenge wird dabei in eine neue Tabelle geschrieben.

> **Tipp**
>
> Die Tabelle wird nur temporär in dem Projekt abgelegt, d.h. es wird keine konkrete Datei geschrieben und auch keine neue Tabelle in der Datenbank erstellt, sondern nur die Abfrage im Projekt gespeichert.

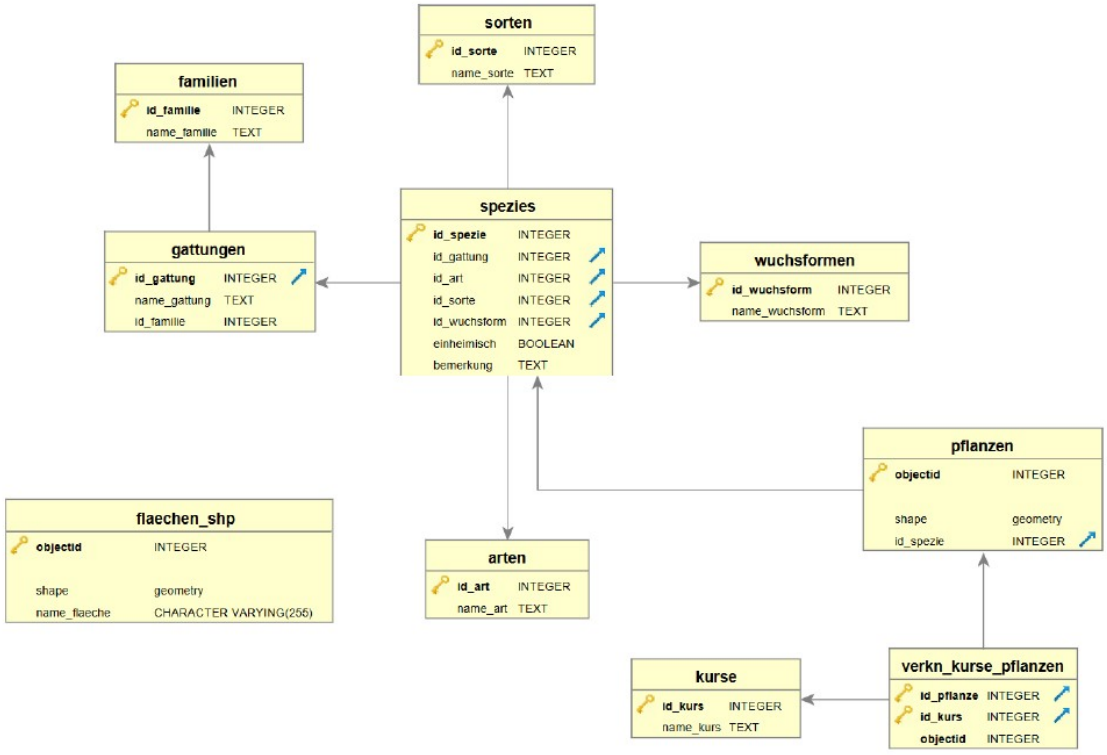


Abbildung 53.1.: Datenbankschema: Pflanzendatenbank (die gelben Schlüssel sind Primärschlüssel, die blauen Pfeile sind Fremdschlüssel).

1. Starten Sie pgAdmin und verbinden Sie mit der Pflanzendatenbank “modulagi_pdb” mit Ihrem Benutzer studentNr.
2. Probieren Sie folgende SQL-Abfrage zu formulieren.
3. Suchen Sie alle Gattungen der Familie “Asteraceae”. Zeigen Sie den Namen der Familie und den Namen der Gattung an (siehe Abbildung 53.2).
4. Starten Sie ArcGIS Pro und öffnen das Projekt “Pflanzendatenbank_III”.
5. Erstellen Sie eine neue Karte.
6. Öffnen Sie das Werkzeug “Make Query Table”
7. Wählen Sie die beteiligten Tabellen familien und gattungen (über die Datenbankverbindung)
8. Wählen Sie die anzuzeigenden Felder name_familie und name_gattung
9. Filtern Sie die Abfrage über die SQL-Expression nach ‘Asteraceae’.

> **Tipp**
>
> Sie brauchen im Werkzeug nur den WHERE Teil der Abfrage ohne das Schlüsselwort WHERE eingeben. Achten Sie auch auf die verknüpften Schlüssel (id_familie) in beiden Tabellen. Sie können den WHERE Teil natürlich aus pgAdmin kopieren. Achten Sie in dem Fall darauf im Werkzeug auf die SQL-Ansicht umzustellen.

10. Geben Sie der neuen Tabelle einen Namen.
11. Prüfen Sie die Inhalte der neuen Tabelle. Vergleichen Sie mit dem Ergebnis aus pgAdmin.

> **Tipp**
>
> “Make Query Table” erlaubt nur NICHT-räumliche Abfragen. Es können zwar Felder aus räumlichen Tabellen abgefragt werden, das Ergebnis ist aber immer eine einfache Tabelle und keine Feature Class.

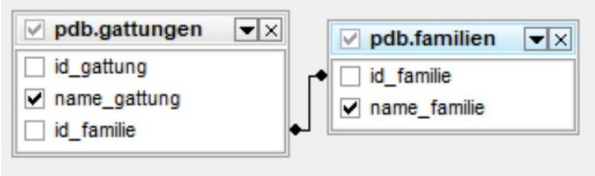


Abbildung 53.2.: Ausschnitt Datenbankschema modulagi_pdb

53.2. Übung 4: Werkzeug “Make Query Layer” anwenden

Das Werkzeug erstellt aus einer beliebigen Datenbankabfrage eine von ArcGIS Pro lesbare temporäre Feature Class.

> **Tipp**
>
> Die Feature Class wird nur temporär in dem Projekt abgelegt, d.h. es wird keine konkrete Datei geschrieben und auch keine neue Tabelle in der Datenbank erstellt, sondern nur die Abfrage im Projekt gespeichert.

1. Probieren Sie folgende SQL-Abfrage in pgAdmin zu formulieren.
2. Suchen Sie den Standort aller Eichen (Gattung = Quercus). Zeigen Sie Gattung, Art und den Standort mit der objectid (Attribut shape + objectid) (siehe Abbildung 53.3).
3. Wechseln Sie zu ArcGIS Pro.
4. Öffnen Sie das Werkzeug "Make Query Layer".
5. Schreiben Sie das SQL-Query.

> **Tipp**
>
> Sie können die Abfrage aus pgAdmin (Schritt 3) kopieren und hier wieder einfügen. Klicken Sie danach auf eine leere graue Fläche irgendwo im Werkzeug-Fenster damit das Kopierte angenommen wird.

6. Die folgenden Einstellungen sollten dadurch dann automatisch gefunden werden.
7. Achten Sie auf den korrekten Geometrietyp (Shape Type) und das Koordinatensystem, die SRID Nummer für das Schweizer Koordinatensystem CH1903+_LV95 lautet 2056.

> **Tipp**
>
> Eine Abfrage mit dem Werkzeug "Make Query Layer" benötigt immer ein eindeutiges Identifizierungsfeld (Unique identifier Field(s)). In diesem Fall nutzen wir das Attribut "objectid". Achten Sie auch in den kommenden Abfragen auf ein solches ID-Feld.

9. Schauen Sie sich den neuen Layer in der Karte an und werfen Sie auch einen Blick auf die Attributtabelle.
10. Vergleichen Sie wieder mit dem Ergebnis aus pgAdmin.

> **Tipp**
>
> Über das Kontextmenü des neuen Layers im Contents Fenster (Data→Export Features) können Sie den Layer auch dauerhaft z.B. als neue Feature Class in eine eigene Datenbank oder als neues Shapefile speichern.

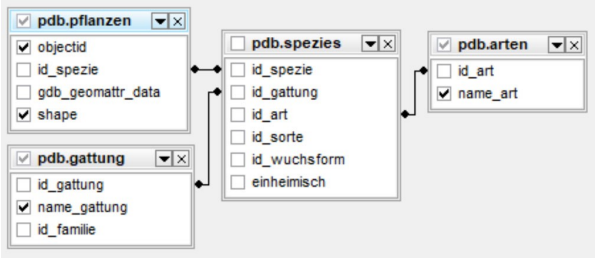


Abbildung 53.3.: Ausschnitt Datenbankschema modulagi_pdb

53.3. Übung 5: Werkzeug "Create Database View" anwenden

"Create Database View" kann dazu benutzt werden, eine Sicht auf eine Tabelle zu erzeugen bzw. eine Abfrage in der Datenbank zu speichern. Hierbei wird dann nur eine Auswahl von Attributspalten und/oder Objekten zurückgegeben. Eine Sicht ist oft auch nützlich, um komplizierte Abfragen aus mehreren Tabellen in eine Tabelle "zusammen zu fassen", um dann einfacher darauf zugreifen zu können.

> **Tipp**
>
> Die Sicht wird bei diesem Werkzeug, anders als in den beiden vorigen Werkzeugen, in der Datenbank gespeichert. D.h. es wird eine neue Tabelle bzw. View in der Datenbank erstellt.

1. Probieren Sie folgende SQL-Abfrage in pgAdmin zu formulieren.
2. Machen Sie eine Liste mit allen Pflanzen des Kurses "UÖ Stauden Gärtnerisch". Die Liste soll dabei die ID der Spezies, die Gattung, Art und Sorte der Pflanze sowie den Namen des Kurses und den Standort (Attribut shape) enthalten. Speichern Sie diese Abfrage als View mit dem Namen "pdb.view_kursliste_stauden_gaertnerisch_studentNr" in der Datenbank. (CREATE VIEW … AS SELECT…) (siehe Abbildung 53.4).

> **Tipp**
>
> Mit zwei Minus Zeichen vor einer Zeile können Sie die Zeile auskommentieren.

4. Wechseln Sie zu ArcGIS Pro.
5. Erneuern Sie die Datenbankverbindung (Refresh / F5), damit die neue View sichtbar ist.
6. Eine View verhält sich wie eine neue Tabelle in ArcGIS Pro. Aus diesem Grund müsste die Tabelle eigentlich mit der Datenbank registriert werden. Sie erkennen dies auch in der Catalog Ansicht an dem nicht ausgefüllten Rechteck (siehe Abbildung 53.5).

7. Eine Registrierung bewirkt das Freischalten der Geodatenbank-Funktionen für diese Tabelle, damit auch neue Daten erfasst und bearbeitet werden können. Da Ihr Benutzer studentNr keine Berechtigungen zum Schreiben im Datenbankschema auf der Datenbank hat, kann die Tabelle jetzt nicht registriert werden. Da wir aber die Tabelle nur abfragen wollen ist dies nicht weiter schlimm.
8. Öffnen Sie das Werkzeug “Create Database View”.
9. Speichern Sie mit dem Werkzeug eine identische Liste für den Kurs “UÖ Gehölze”. Geben Sie der View den Namen “pdb.view_kursliste_gehoelze_studentNr”. Achten Sie auf das “pdb.” vor dem eigentlichen Namen. Dies gibt das Schema an indem die View gespeichert werden soll. Sie können hierzu wieder das SQL-Select-Statement von oben kopieren (ohne den CREATE VIEW Teil) und den Kursnamen entsprechend anpassen.
10. Schauen Sie sich das Ergebnis in der Attributtabelle und in der Karte an.
11. Sie sehen an dieser Übung, dass nur der Datenbankbesitzer neue Tabellen erstellen und hinzufügen kann. Andere Benutzer wie der student1 können nur vorhandene Tabellen und Daten bearbeiten und ändern. Neue Tabellen können nur ohne Schreibberechtigung erstellt werden.

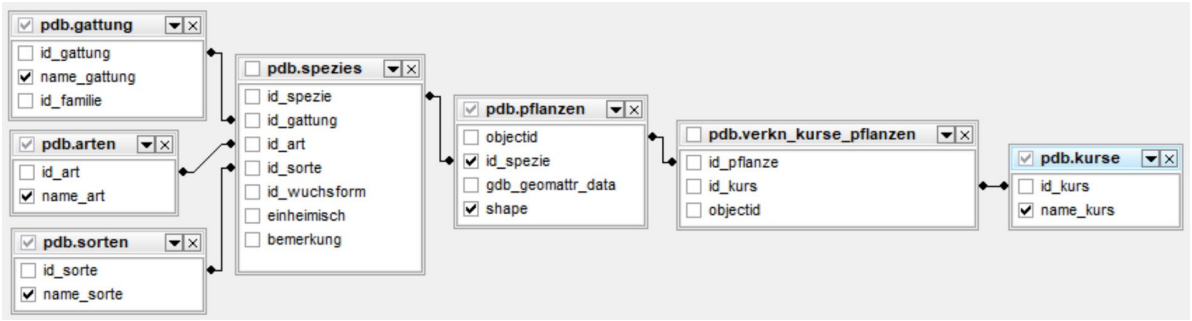


Abbildung 53.4.: Ausschnitt Datenbankschema modulagi_pdb

53.4. Übung 6: (optional) Werkzeug “Make Table View”

“Make Table View” kann dazu benutzt werden, um eine reduzierte Sicht auf eine Tabelle zu erzeugen. Hierbei wird dann nur eine Auswahl von Attributspalten (ohne Geometrie-Attribut) angezeigt.

> **Tipp**
>
> Die Sicht wird nur temporär in dem Projekt abgelegt, d.h. es wird keine konkrete Datei geschrieben und auch keine neue Tabelle in der Datenbank erstellt, sondern nur die Abfrage im Projekt gespeichert.

1. Starten Sie ArcGIS Pro und laden das Projekt mit dem Namen “Pflanzendatenbank_III”

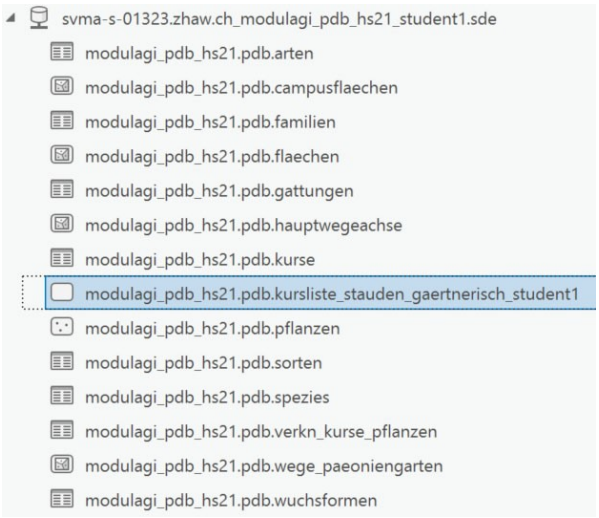


Abbildung 53.5.: Catalog Ansicht in ArcGIS Pro

2. Öffnen Sie das Werkzeug “Make Table View”.
3. Erstellen Sie eine Abfrage wie in Übung 5 und filtern nach der Art “japonicum”.
4. Wählen Sie als Input Table die zuvor erstellte View, lassen Sie aber das Geometriefeld (shape) in der Abfrage weg und filtern über eine “new Expression” nach name_art = “japonicum”.
5. Lassen Sie sich die Attributtabelle der neuen Tabelle anzeigen.

54. Geoverarbeitung in der Datenbank mit ArcGIS Pro und SQL

In diesem Kapitel sollen konkrete Geoverarbeitungsoperationen in ArcGIS Pro und PostGIS/SQL gegenübergestellt werden. Dafür bearbeiten wir die entsprechende Fragestellung zunächst in ArcGIS Pro und stellen anschliessend die gleiche Funktion in PostGIS nach. Zur Visualisierung des Ergebnisses der SQL-Abfrage nutzen wir das Werkzeug "Make Query Layer" in ArcGIS Pro aus Übung 4.

54.1. Übung 7: Geometrien zusammenführen mit "Dissolve"

Die importierten Flächen sind alle einem bestimmten Bereich zugeordnet. Um einen Überblick über die Grösse und Ausdehnung der Bereiche zu bekommen, möchten wir alle Flächen (Tabelle pdb.flaechen_shp) mit identischem Bereich (Attribut "name_bereich") zusammenführen (Abbildung 54.1).

1. Führen Sie die Geometrien mit identischem Bereichsnamen (Attribut "name_bereich") zusammen. Verwenden Sie wieder die Verbindung mit dem Benutzer studentNr.
    - Verwenden Sie das Werkzeug "Dissolve" in ArcGIS Pro. Nutzen Sie dabei Multipart Features. Speichern Sie das Ergebnis als neuen Layer.
    - Formulieren Sie jetzt die SQL Anfrage in pgAdmin (Verbindung wieder mit Benutzer studentNr). Verwenden Sie dabei die Funktion ST_UNION. Um nur gewünschte Flächen zu erhalten, sollten Sie das Ergebnis im SQL-Befehl nach dem Namen des Bereichs gruppieren (GROUP BY).
    - Optional: Was passiert wenn das GROUP BY weggelassen würde? Probieren Sie es aus.
    - Lassen Sie das Attribute name_bereich für die Flächen anzeigen.
    - Zeigen Sie das Ergebnis mit "Make Query Layer" an und vergleichen mit dem ArcGIS Pro Resultat.

> **Tipp**
>
> Da Ihr Datenbankbenutzer studentNr keine Berechtigungen zum Speichern in der Server Datenbank hat, können Sie das Ergebnis lokal auf Ihrem Rechner in Ihrem Ordner oder

einer eigenen Geodatenbank ablegen.

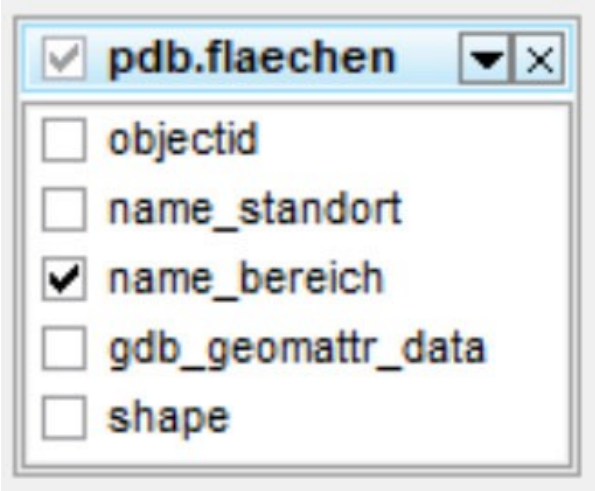


Abbildung 54.1.: Auszug aus Datenbankschema

54.2. Übung 8: Puffer um Geometrie erstellen

Auf dem Campus sollen die Wege im Päoniengarten auf jeder Seite um 25cm verbreitert werden. Um die neuen Wege darzustellen, erstellen wir einen Puffer. Die Wegflächen finden Sie in der Datenbank in der Feature Class "wege_paeoniengarten" (siehe Abbildung 54.2).

1. Legen Sie jeweils einen Puffer von beidseitig 25cm um das Wegenetz.
    - Nutzen Sie dazu einmal das Werkzeug "Buffer" in ArcGIS Pro. Speichern Sie hierbei das Ergebnis.
    - Nutzen Sie jetzt die POSTGIS Funktion ST_BUFFER in pgAdmin. Geben Sie auch die id der Flächen mit aus.

> **Tipp**
>
> Achten Sie auf die Einheiten. Welche Einheit nimmt ST_BUFFER? Fragen Sie die Online-Hilfe.

2. Skizzieren Sie ihr Vorgehen in ArcGIS Pro und notieren ihren SQL-Befehl.
3. Um das Ergebnis der PostGIS Funktion zu visualisieren, nutzen Sie das Werkzeug "Make Query Layer" mit Ihrer SQL Abfrage.
4. Vergleichen Sie die Ergebnisse. Schauen Sie sich auch die Attribute an. Was fällt auf?

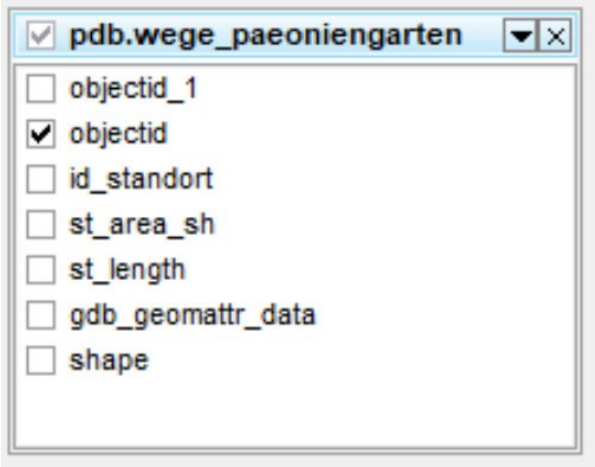


Abbildung 54.2.: Auszug aus Datenbankschema

54.3. Übung 9: Geometrien überschneiden mit “Intersect”

An einigen Stellen muss für die Verbreiterung des Weges die Fläche einiger Beete weichen. Ermitteln Sie die betroffenen Beete, welche Fläche für die neuen Wege abgeben müssen (siehe Abbildung 54.3).

1. Bilden Sie die Schnittmenge (Intersection) der gepufferten Wegfläche und der restlichen Flächen aus der Tabelle “pdb.flaechen”.
    - In ArcGIS Pro verwenden Sie dazu das Werkzeug “Pairwise Intersect”. Sie können als Input entweder Ihr Ergebnis aus der vorigen Übung benutzen oder Sie verwenden den Datensatz “wege_paeoniengarten_buffer” aus der Datenbank. Speichern Sie das Ergebnis.
    - Stellen Sie den SQL-Befehl mit Hilfe der Funktion “ST_INTERSECTION” für diese Abfrage in pgAdmin auf und visualisieren das Ergebnis wieder mit “Make Query Layer”. Sie können als Input entweder den gespeicherten Puffer aus der vorigen Übung verwenden oder Sie verwenden den Datensatz “wege_paeoniengarten_buffer” aus der Datenbank. Probieren Sie ebenfalls einmal aus durch eine verschachtelte Abfrage den Puffer mit ST_BUFFER direkt einzubeziehen. Geben Sie in Ihrem SQL-Befehl auch die ID der Fläche mit aus.
2. Skizzieren Sie ihr Vorgehen in ArcGIS Pro und notieren ihren SQL-Befehl.
3. Vergleichen Sie die Ergebnisse aus ArcGIS Pro und SQL.

Tipp

Die SQL Funktion ST_INTERSECTION erzeugt immer eine so genannte Geometry Collection, auch wenn zwei Geometrien keine Schnittmenge aufweisen, gibt es einen dann leeren Eintrag. Aus diesem Grund zeigt das Ergebnis immer alle Geometrien an. Haben die Geometrien keine Schnittmenge, so ist die Geometrie leer. Um leere Geometrien auszuschliessen, sollte im WHERE-Abschnitt z.B. mit der Funktion ST_OVERLAPS oder der Funktion ST_INTERSECTS geprüft werden, ob die zwei Eingabe-Geometrien (Gepufferte Wege und sonstige Flächen) überhaupt irgendwo überlappen. Im ArcGIS Pro werden leere Geometrien automatisch vom Ergebnis abgeschnitten.

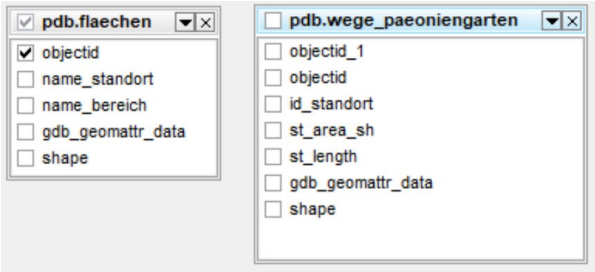


Abbildung 54.3.: Auszug aus Datenbankschema

54.4. Übung 10: Flächen berechnen in ArcGIS Pro und SQL

In der Übung 10 wurden Flächen ausgeschieden, welche von den betroffenen Beeten abgegeben werden müssen. Die genauen Flächengrössen sollen jetzt berechnet werden (Abbildung 54.4).

1. Welche Beete müssen wie viel Fläche abgeben? Berechnen Sie die Fläche pro Beet in qm.
    - Legen Sie dazu in ArcGIS Pro ein neues Feld in der Attributtabelle ihres gespeicherten Layers an und berechnen dort die jeweiligen Einzelflächen in qm (Calculate Geometry). Haben Sie das Ergebnis in einer Geodatenbank gespeichert gibt es bereits das Attribut shape_area mit der Fläche in qm.
    - Erstellen Sie in pgAdmin eine Abfrage in SQL, welche ebenfalls die jeweiligen Einzelflächen in qm ausgibt. Ergänzen Sie dazu die SQL-Abfrage aus Übung 9 mit einem neuen Feld, welches die Fläche ausgibt. Wenden Sie dazu die Funktion ST_AREA auf das Geometriefeld an.
    - Visualisieren Sie das Ergebnis erneut mit “Make Query Layer”.

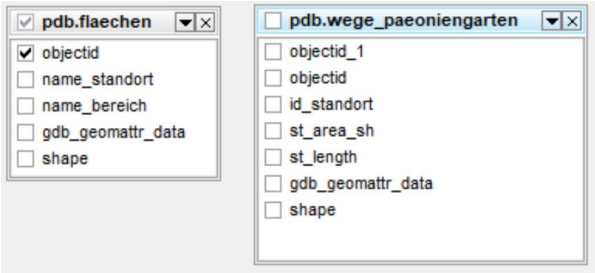


Abbildung 54.4.: Auszug aus Datenbankschema

54.5. Übung 11: Geometrien ausschneiden mit “Erase”

Als nächstes wollen wir die grosse Hauptwege-Achse zu unseren Flächen hinzufügen. Hierzu haben wir eine weitere Feature Class in der Datenbank mit der Hauptwegeachse zur Verfügung. Leider enthält die Feature Class nur die Aussengrenzen des Weges. Da es auf dem Weg auch Beete gibt, wollen wir diese Beetflächen aus der Wegfläche ausschneiden (siehe Abbildung 54.5).

1. Schneiden Sie die Beetflächen aus der Wegeachse aus. Nutzen Sie dafür:
    - In ArcGIS Pro das Werkzeug “Erase” und speichern das Ergebnis als neue Feature Class oder als Shapefile in Ihrem Ordner.
    - Formulieren Sie in pgAdmin den SQL-Befehl mit der Funktion ST_DIFFERENCE und zeigen Sie das Resultat mit “Make Query Layer” auf der Karte. Geben Sie dabei zusätzlich die id der Fläche sowie die Attribute name_standort und name_bereich mit aus.
2. Vergleichen Sie die Ergebnisse.

> **Tipp**
>
> Diese Anfrage funktioniert in zwei Schritten mit einer verschachtelten Abfrage (aber trotzdem in einem SQL-Befehl). Um Polygone auszuschneiden, müssen wir zunächst alle Wegeflächen und alle sonstigen Flächen aus der Tabelle “flaechen_shp” zu jeweils einem Polygon zusammenfügen (ST_Union), damit wir nur noch zwei Flächen miteinander vergleichen müssen. Anschliessend werden die beiden Flächen dann mit dem Werkzeug ST_Difference untersucht.
> Die Funktion ST_Difference berechnet für jedes eingeschlossene Polygon einmal die Differenzmenge aus der Wegefläche und allen eingeschlossenen Polygonen. Dadurch hat das Ergebnis eigentlich zu viele Einträge, für jedes eingeschlossene Polygon ein Eintrag, welche aber alle dieselbe Geometrie aufweisen. Der Parameter LIMIT 1 limitiert die Ausgabe auf ein Feature.

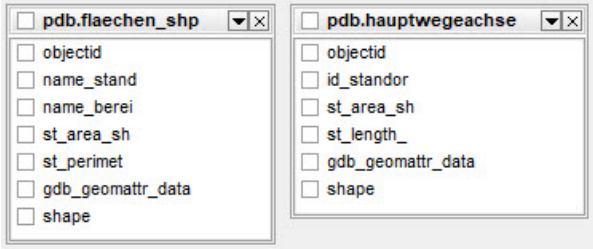


Abbildung 54.5.: Auszug aus Datenbankschema

54.6. Übung 12: Geometrien ausschneiden mit “Clip”

Als nächstes möchten wir alle Pflanzen suchen, welche auf bereits kartierten Flächen stehen. Hierzu nehmen wir den Punktdatensatz der Pflanzen und verschneiden diesen mit den bereits vorhandenen Beet-Flächen (siehe Abbildung 54.6).

1. Suchen Sie alle Standorte (Punkte) der Pflanzen, welche auf kartierten Flächen stehen.
   - In ArcGIS Pro verwenden Sie das Werkzeug “Clip” und schneiden damit überflüssige Punkte weg. Speichern Sie das Resultat in Ihrem Ordner. Alternativ würde auch ein “Select by Location” gehen.
   - Formulieren Sie auch hierzu die SQL-Anfrage in pgAdmin, welche alle Punkte auf den Flächen zurückgibt. Sie können dazu wieder die Funktion ST_INTERSECTION nutzen. Achten Sie darauf, dass wir dabei wieder nur diejenigen Punkte haben wollen, welche auch wirklich in Beziehung zu den Flächen stehen. Diesen Filter müssen Sie im WHERE-Abschnitt bereitstellen. Verwenden Sie dafür diesmal die Funktion ST_INTERSECTS. Binden Sie ausserdem noch die Attribute id_pflanze (objectid) und id_spezies in die Ausgabe ein und sortieren die Liste danach anhand der id_pflanze.

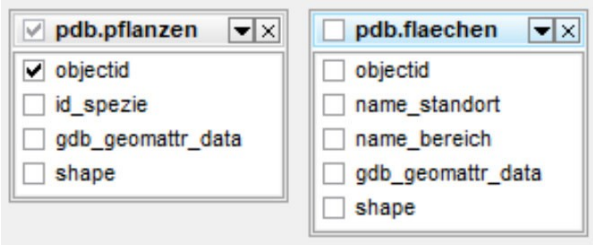


Abbildung 54.6.: Auszug aus Datenbankschema

Teil VIII.

Netzwerkanalyse I

Die Lerneinheit vermittelt die Grundlagen bei der Verwendung von Netzwerk-Geodaten. Die Anwendungsfelder reichen von Streckenfindung und Reisezeitberechnungen zu Energienetzwerk-Systeme bis hin zu Gewässernetzen in der Hydrologie. Diese Lerneinheit vermittelt die dafür relevanten Grundlagen aus der Graphentheorie und knüpft dabei an die bekannten Grundlagen der Vektordatenmodellierung (Definitionen, Elemente von Graphen, Eigenschaften von Graphen, Bäume) an. Zudem werden im theoretischen Teil Netzwerkmasse, insbesondere Zentralitätsmasse nähergebracht. Der praktische Teil basiert auf der Einführung und Verwendung von QGIS als Alternative zu den ESRI-Produkten.

Übungsziele

- Einfache GIS-Operationen mit QGIS durchführen können (Clip, Reproject, Abfrage der Attributtabelle, Symbolisierung, Export als jpg)
- Schnittstelle zwischen QGIS und anderen GIS-Software am Beispiel GRASS kennenlernen
- Erste Netzwerkoperationen mit QGIS/GRASS GIS durchführen
- Netzwerk-Zentralitätsmasse verstehen und für einfache Netzwerkdaten berechnen können

55. Vorbereitung

Damit wir im ersten Teil von Netzwerkanalyse direkt loslegen können solltet ihr euch vorgängig mit QGIS vertraut machen. Dafür haben wir ein paar Übungen zusammengesetellt, die ihr als Vorbereitung auf Netzwerkanalyse I erledigen solltet.

Übungsziele

- QGIS herunterladen und installieren
- QGIS starten und kennenlernen
- Modularer Aufbau von QGIS verstehen
- Plugins installieren und anwenden können
- Daten für Übungseinheit Netzwerkanalyse I vorbereiten

55.1. Datensätze

Laden Sie das File netzwerkanalyse.gpkg von Moodle herunter. Hier befinden sich alle Datensätze, die im Laufe des Blocks "Netzwerkanalyse" benötigt werden:

Tabelle 55.1.: Datensätze im File *netzwerkanalyse.gpkg* für den Block "Netzwerkanalyse"

Layer	Koord. System	Beschreibung
Gemeinde_Waedenswil	EPSG 2056	Die Gemeindegrenze von Wädenswil. Dieser Datensatz basiert auf swissBOUNDARIES3D von Swisstopo.
osm_highway	EPSG 4326	Alle "Highway" Linien aus dem OpenStreetmaps (Stand nach* Kapitel 55.6)
osm_highway_prepared	EPSG 2056	Alle "Highway" Linien transformiert und neu projiziert (Output aus Kapitel 56.2)

Layer	Koord. System	Beschreibung
shops_waedenswil	EPSG 2056	Alle Läden in Wädenswil, für Kapitel 59.2
buildings_waedenswil_polygons	EPSG 2056	Alle Gebäudestandorte (Polygon Daten) in Wädenswil, für Kapitel 59.3
abfallentsorgung_waedenswil	EPSG 2056	Entsorgungsstellen Wädenswil
waedenswil_centrality	EPSG 2056	Zentralitätsmasse für Wädenswil (aus Kapitel 57.3)
Haltestellen_waedenswil	EPSG 2056	Haltestellen in Wädenswil

55.2. Übung 1: QGIS installieren

Auf der QGIS Homepage qgis.org finden Sie die aktuellen QGIS Versionen für Windows, Linux, Mac und weitere Betriebssysteme in 32 und 64 bit. Laden Sie den passenden Standalone Installer des "Longterm release (most stable)" herunter (aktuell **3.28**). Installieren Sie QGIS nach dem Download.

55.3. Übung 2: Tutorials anschauen

Schauen Sie sich danach zum Einstieg einzelne Videos vom Youtube Nutzer Marshal Mappers an. Schauen Sie die Videos bis Sie die grundlegendsten Arbeitsschritte von QGIS (Daten importieren, Werkzeuge finden, Karte zu pdf exportieren) verstanden haben. Erstellen Sie dann ein neues Projekt (Projekt→New) und speichern Sie dieses direkt ab (Projekt→Save As..).

55.4. Übung 3: Daten importieren

Während Shapefiles im GIS-Unterricht bisher oft zur Speicherung von Vectordaten verwendet wurden, werden in QGIS vor allem Geopackage Dateien verwendet (.gpkg). QGIS kann Shapefiles durchaus lesen und schreiben, wir werden in den Übungen aber vor allem mit Geopackage Daten arbeiten. Geopackages sind eine alternative Methode, Vektordaten abzuspeichern. Sie beheben einige Defizite, die Shapefiles mit sich bringen. Siehe dazu auch die Website "Shapefiles must die".

1. Neues QGIS Projekt starten und Projektdatei speichern (Beachten Sie dazu die Empfehlungen am Anfang)

2. CRS auf EPSG 2056 setzen
3. Aus dem File *netzwerkanalyse.gpgk* den Layer *Gemeinde_Waedenswil* (siehe Tabelle 55.1) in QGIS Importieren
4. Symbologie folgendermassen ändern: Fläche transparent, Stadtgrenze schwarz

55.5. Übung 4: Plugin installieren

QGIS wird von zahlreichen Einzelpersonen und Gruppen entwickelt. Aus diesem Grund ist die Software modular aufgebaut, und nur ein Teil wird mit der Standard-Installation mitgeliefert. Für einige Funktionen müssen zusätzliche Erweiterungen (sogenannte "Plugins") installiert werden. Installieren Sie das Plugin "QuickOSM" um OpenStreetMap (OSM) Vektordaten rasch und einfach lokal abspeichern zu können.

1. Reiter Plugins→Manage and Install Plugins
2. Im Suchfenster "QuickOSM" suchen
3. Plugin anwählen und auf "install" klicken

Die wichtigsten Metadaten zu allen Plugins werden auf plugins.qgis.org festgehalten. Dort findet man auch Links zur Projektseite, weiteren Dokumentation und ggf. Tutorials: Zu QuickOSM sind die Metadaten hier abrufbar: https://plugins.qgis.org/plugins/QuickOSM/

55.6. Übung 5: OpenStreetMap Vektordaten herunterladen

Mit dem neuen Plugin "QuickOSM" laden Sie nun den Strassendatensatz der Gemeinde Wädenswil herunter. Dies geschieht folgendermassen:

1. Rechtsklick auf den Layer *Gemeinde_Waedenswil* → Zoom to layer
2. Reiter Vektor → Quick OSM → Quick OSM
3. Wählen Sie bei der Option key "highway" und lassen value leer
4. Wählen Sie Option "Canvas Extent"
5. Klicken Sie anschliessend auf "Run query"

Das Query lädt nebst den Liniendaten auch noch Punkt- und Polygon-Daten herunter. Diese interessieren uns nicht und können entfernt werden (Rechtsklick→remove).

55.7. Übung 6: Temporäre Datei abspeichern

Outputs werden in QGIS standardmässig in einem Temp-Folder abgelegt. Diese Dateien werden nach Beendigung von QGIS gelöscht. Um die Daten auch zu einem späteren Zeitpunkt verwenden zu können, müssen sie an einem geeigneten Ort abgespeichert werden. Führen Sie

diesen Schritt mit den eben beschafften OSM Strassendaten (nur Linien) aus. Im gleichen Schritt können Sie alle unnötigen Spalten aus der grossen Attributtabelle löschen.

1. Rechtsklick auf den temporären Linien-Layer → Export → Save Features As…
2. Unter "Select fields to export and their export options" nur Spalte *highway* auswählen
3. Format: Geopackage
4. Filename: Geeigneter Speicherort[1] aufsuchen und Datei abspeichern als "osm_highway.gpkg" (Erweiterung muss nochmals angegeben werden)

Merken Sie sich:

- Es lohnt sich, vor jedem Projekt eine sinnvolle Ordnerstruktur aufzubauen
- Neben Shapefiles gibt es weitere (bessere?) Wege, wie Vektordaten abgespeichert werden können. Eine gute Variante ist Geopackage.
- QGIS ist mittlerweile ein mächtiges Werkzeug, welches für die Bearbeitung vieler klassischer GIS Fragestellungen geeignet ist.
- QGIS ist modular aufgebaut, wichtige Funktionen sind über Plugins verfügbar. Dadurch kommt QGIS nicht aus "einem Guss" daher.
- Outputs werden in temporären Dateien abgespeichert, die bei der Schliessung von QGIS gelöscht werden. Sollen Geodaten permanent verfügbar sein, müssen sie entsprechend explizit abgespeichert werden.

[1] Merken Sie sich den Speicherort, Sie werden das File in der kommenden Übung brauchen.

56. Aufgabe 1: Operationen mit QGIS

Starten Sie QGIS, erstellen Sie ein neues QGIS Projekt und speichere dieses an einem geeigneten Ort (siehe Vorbereitungsübungen) ab. Weisen Sie dem Projekt das Koordinatensystem EPSG 2056 zu und importiere die Gemeindegrenze sowie die OSM Strassendaten aus der Vorbereitungsübung (siehe Tabelle 55.1).

56.1. Übung 1.1: Daten Transformieren

Die OSM Daten sind aktuell noch im Koordinatensystem WGS84 (EPSG 4326). Die Gemeindegrenze hingegen ist mit den neuen Schweizer Landeskoordinaten CH1903+ LV95 (EPSG 2056) abgespeichert. Wir wollen in unserer Analyse mit CH1903+ LV95 (EPSG 2056) arbeiten. Transformieren Sie dazu den Strassendatensatz in das Koordinatensystem CH1903+ LV95 (EPSG 2056). Nutzen Sie dazu das Tool `Reproject Layer`. **Wichtig**: Speichern Sie den Transformierten Strassendatensatz *in einer neuen Datei* (siehe Abbildung 56.2)

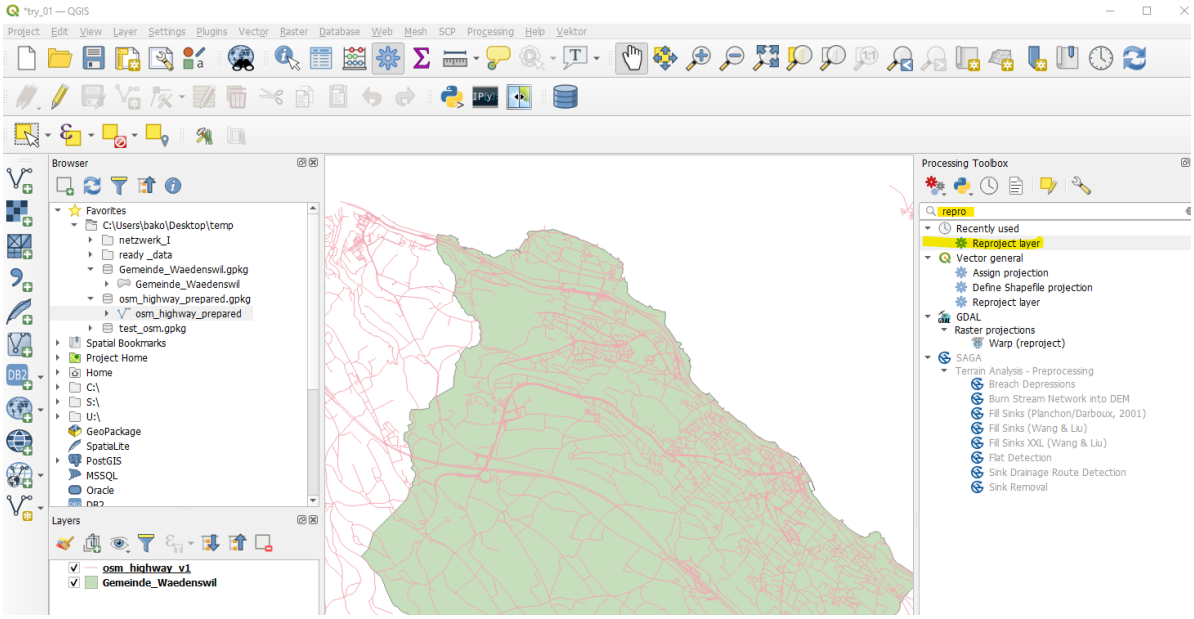


Abbildung 56.1.

Viele wichtige Tools lassen sich über die Menü Bar aufrufen (v.a. “Vector” und “Raster”). Die Tools lassen sich auch relativ rasch mit der Suchfunktion in “Processing Toolbox” finden.

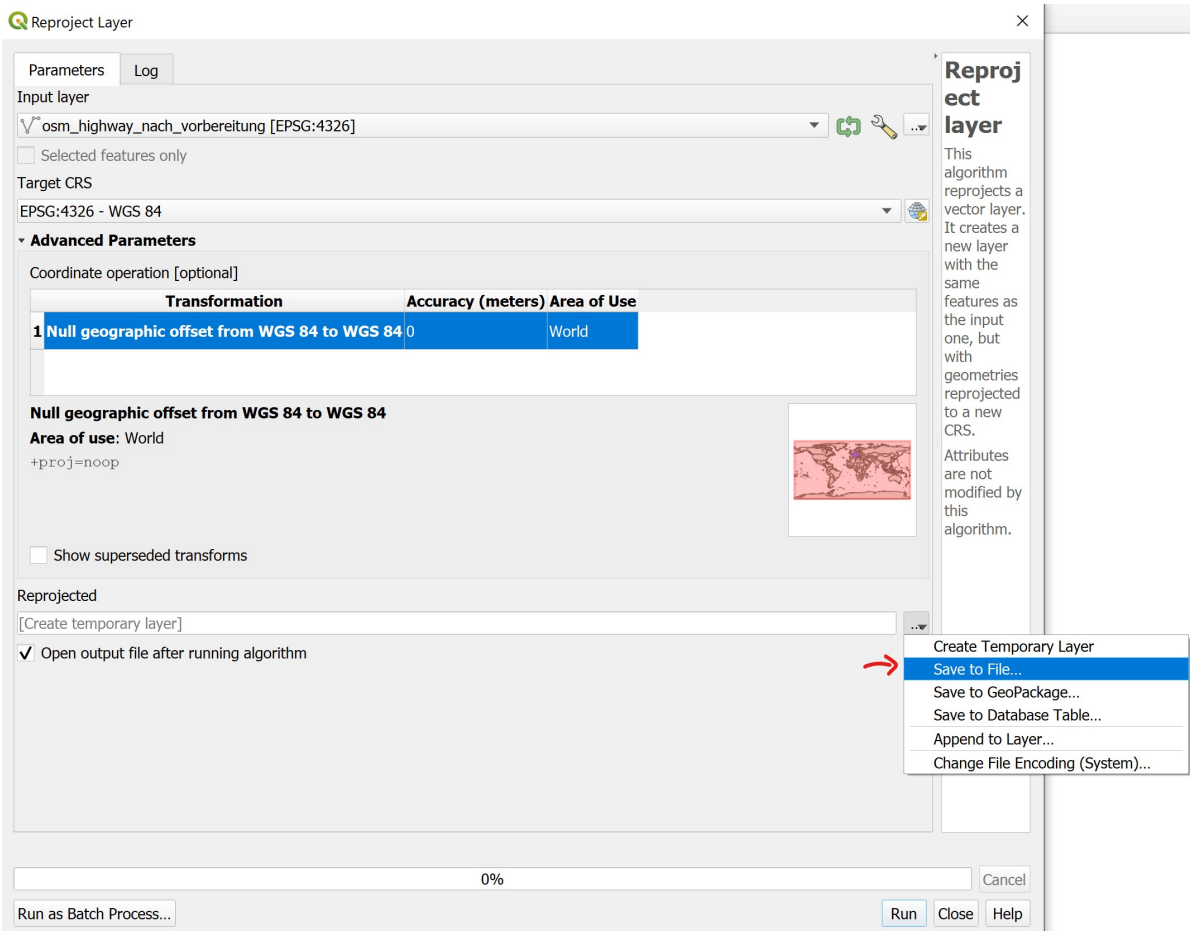


Abbildung 56.2.: Tool Reproject Layer

Speichern Sie den neuen Strassendatensatz in einer *neuen* Datei, nicht als temporärer Layer. Normalerweise spielt dies keine Rolle, doch im Falle von “reproject” entsteht beim Speichern als temporärer Layer ein Fehler.

56.2. Übung 1.2: Daten Clippen

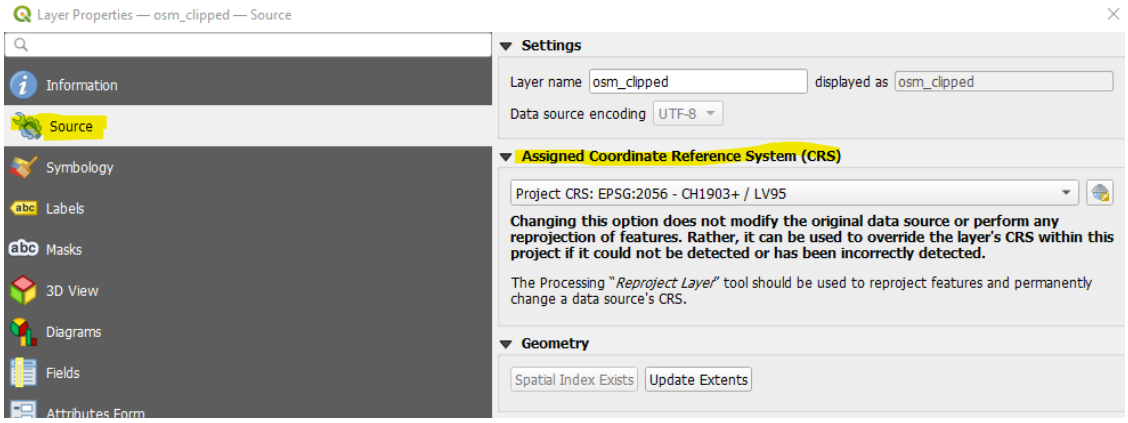
Zoomen sie auf die Gemeindegrenze (Layers Panel→Rechtsklick auf Datensatz → *Zoom to layer*). Sie stellen fest, dass die Strassendaten über die Gemeindegrenze hinaus verlaufen. Wir möchten für die kommenden Übungen nur die Strassen, die *innerhalb* der Gemeinde Wädenswil liegen. Dazu müssen wir das Strassennetz mit der Gemeindegrenze verschneiden

(“clip”). Führen Sie das gleichnamige Werkzeug mit dem Input Layer “OSM_highway” und dem Clip Layer *Gemeinde_Waedenswil* aus. Speichern Sie die Ausgabe in einer neuen Datei.

Es gibt eine ganze Reihe Werkzeuge zum Begriff “clip”. Entscheiden Sie selbst, welches für diese Fragestellung geeignet ist.

Hinweis

- Achten Sie darauf, das dem neuen (“clipped”) Datensatz das richtige Koordinatensystem zugewiesen wurde. Wenn nicht, gehe zu: Properties > Source > Assigned Coordinate Reference System (CRS), wähle the EPSG:2056 - CH1903+ / LV95 und klicke auf Apply



57. Aufgabe 2: Zentralitätsmasse berechnen

Die mächtigsten (aber nicht die einzigen[1]) Netzwerkanalyse Werkzeuge in QGIS stammen aus dem eigenständigen GIS "GRASS GIS", welches bei der Installation von QGIS mitinstalliert wird. Diese können innerhalb von QGIS verwendet werden.

57.1. Übung 2.1: Topologie bereinigen

Das OSM Strassennetz "osm_highway" ist topologisch nicht perfekt für unsere Zwecke vorbereitet. An Kreuzungen fehlen teilweise Knoten, welche ein "Abbiegen" auf der Kreuzung ermöglichen. Um diesen Umstand zu beheben, nutzen wir das Tool `v.clean` und führen damit die Operation "break" durch. Dadurch werden Linien an Kreuzungen unterbrochen. Diese Operation löst einige topologische Fehler, führt z.B. bei Brücken und Tunnels aber zu neuen Unstimmigkeiten, die Sie an dieser Stelle aber getrost ignorieren können.

1. Werkzeug `v.clean` auswählen
2. Parameter:
    - *Layer to clean* : Transformierter und geclippter OSM Strassendatensatz
    - *Cleaning tool* : break
    - *v.out.ogr output type:* auto
    - *cleaned (output)*: Save to File…
3. Mit "Run" ausführen

57.2. Übung 2.2: Losgelöste (getrennte) Elemente entfernen

Aufmerksamen Anwendern könnte nun auffallen, dass gewisse Bestandteile des Netzwerks nicht mit dem Hauptnetz verbunden sind. Diese getrennten Elemente können mit dem Werkzeug `v.net.components` identifiziert werden. Das Werkzeug prüft, welche Bestandteile des Netzwerkes miteinander verbunden sind und gruppiert diese mit Nummern. Diese Nummern werden in der Spalte "comp" abgespeichert.

[1] Da QGIS wie bereits erwähnt von verschiedenen Personen und Gruppen entwickelt wird, gibt es auch Doppelspurigkeiten, die man so in einer kommerziellen Software wie ArcGIS weniger vorfindet. In dieser Hinsicht ist QGIS sehr ähnlich wie die Programmiersprachen R und Python.

Im Idealfall sollte unser Netzwerk aus einer Gruppe bestehen; so wäre jeder Knotenpunkt mit jedem andern Knotenpunkt verbunden: Bei uns ist dies jedoch nicht der Fall. Damit wir in den nachstehenden Übungen mit einem sauberen Datensatz arbeiten können, bereinigen wir dieses Problem an dieser Stelle:

1. Führen Sie das Werkzeug `v.net.components` aus
   - *Type of components* : "strong"
   - *V.out.ogr output type:* auto
   - *Network_Components_Line:* Save to File...
   - *Network_Components_Point:* Save to File...
2. Der neue Linien Datensatz verfügt (wie oben beschrieben) über eine Spalte "comp". Überprüfen Sie dies indem Sie die Attributtabelle anschauen.
3. Färben Sie die Linien nach der Spalte ("comp") ein. So, dass jede Gruppe eine Farbe erhält.
4. Ermitteln Sie die Nummer der Hauptkategorie (z.B. mit dem Werkzeug "Identify Features")
5. Öffnen Sie die Attributtabelle und machen sie eine "Selektion anhand einer Abfrage" ("Select Features using an expression" [1])
6. Selektieren Sie die Hauptkategorie mittels einer korrekten Abfrage (z.B. comp = 99). QGIS bietet ihnen hierfür leider wenig Hilfestellung (im Gegensatz zum Query Builder von ArcGIS)
7. Um die anderen Kategorien zu entfernen, haben Sie nun zwei Möglichkeiten:
   - Eine Editiersession starten [2], die Selektion invertieren [3] und die selektierten Daten löschen [4]. **ODER**
   - Selektierte Daten (ohne Invertierung) in ein neues File exportieren mittels Rechtsklick auf den Layer→"Save as"→Häkchen bei "Save only selected features"
8. Speichern Sie den Layer anschliessend mit folgendem Namen ab: "osm_highway_prepared.gpkg"

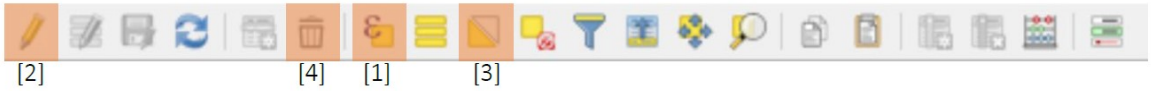


Abbildung 57.1.

57.3. Übung 2.3: Zentralitätsmasse berechnen

Nun kann mit der eigentlichen Netzwerkanalyse begonnen werden. Wir rechnen für unser bereinigtes Strassennetz verschiedene Zentralitätsmasse.

1. `v.net.centrality` starten

2. Parameter
    - *Input vector line layer* : Bereinigter Output aus letzter Übung
    - *v.out.ogr output type* : auto
    - *Network Centrality*: Save to File…
3. Mit “run” ausführen
4. Speichern Sie die resultierende Datei in ihrem Ordner ab (“waedenswil_centrality.gpkg”)

57.4. Übung 2.4: Zentralitätsmasse Visualisieren

Visualisieren Sie die Ausprägung “Closeness” der berechneten Zentralitätswerte über die Symbologie (Layer Properties→Style→Graduated). Wählen Sie eine geeignete Methode und passen Sie den Stil an, bis er Ihnen gefällt. Spielen Sie dabei mit der Klassifikationsmethode (Mode: Equal interval, Quantile, Natural Breaks..) sowie der Color ramp rum. Achten Sie darauf, dass Sie immer auf “Classify” klicken müssen, um Änderungen anzuwenden.

57.5. Übung 2.5: Zentralitätsmasse vergleichen

Installieren Sie das Plugin “QuickMapServices” um eine OSM Hintergrundkarte einzubinden (e.g. Web → QuickMapServices → OSM → OSM Standard). Um direkt in QGIS die Zentralitätsmasse zu vergleichen, duplizieren sie den entsprechenden Layer noch zweimal (rechtsklick > Duplicate) – so können Sie jede der Zentralitäten separat symbolisieren und vergleichen. Sie können auch für jedes der Zentralitätsmasse eine Karte exportieren via *Project > Import / Export Export Map to Image*.

Vergleichen Sie die drei Zentralitätsmasse und setzen Sie sie in den Kontext der Theorie.

Teil IX.

Netzwerkanalyse II

Übungsziele

- In einem gegebenen Netzwerk kürzeste Pfade mit QGIS berechnen können
- Weitere GRASS GIS Werkezeuge innerhalb von QGIS ansteuern können
- Verschiedene QGIS / GRASS GIS Tools kennenlernen und deren Outputs interpretieren können
- Eigene einfache Vektor-Datensätze erstellen können, resp. Punkte von Hand digitalisieren können
- Alternative, professionelle Routing Services kennenlernen und ausserhalb herkömmlicher Services (z.B. Google) nutzen

58. Aufgabe 3: Kürzeste Pfade (shortest path)

Hinweis

Standartmässig werden Dateien als temporäre Files abgespeichert, die nach dem Schliessen von QGIS gelöscht werden. Um einen Output an einem festgelegten Ort abzuspeichern muss der Output definiert werden. Dazu klickt man neben [Save to temporary file] auf die drei Punkte und wählt "Save to File" aus.

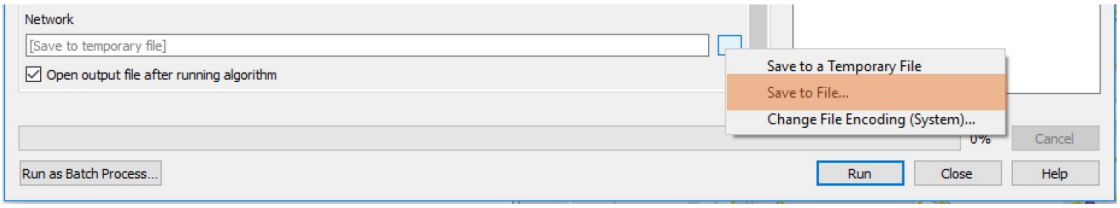


Abbildung 58.1.: Save to File

58.1. Übung 3.1: Projekt vorbereiten

Starten Sie QGIS und beginnen Sie ein neues Projekt mit dem CRS CH1903+ LV95 (EPSG 2056). Lesen Sie den Datensatz "osm_highway_prepared.gpkg" von letzter Woche ein. Wer den Datensatz von letzter Woche nicht auffinden kann, findet die Datei in Tabelle 55.1. Prüfen Sie, ob das CRS richtig erkannt wurde (Rechtsklick → Properties → Reiter Source → *Set source coordinate reference system* → Hier sollte EPSG 2056 stehen).

Wir brauchen zudem eine Hintergrundkarte zur Orientierung. Blenden Sie mit dem Plugin "QuickMapServices" die Openstreetmap Hintergrundkarte ein (Web → QuickMapServices → OSM → OSM Standard) ein. Falls Sie diese Option nicht finden, müssen Sie das enstprechede Plugin "QuickMapServices" installieren (siehe dazu Kapitel 55.5).

Die Hintergrundkarte dient lediglich zur Orientierung, die Farben lenken uns jedoch vom Netzwerk ab. Wechseln Sie deshalb den Darstellungsmodus auf Graustufen mittels Rechtklick auf den Layer "OSM Standard" → Properties → Symbology → Grayscale Auswahl: "By lightness".

58.2. Übung 3.2: Kürzester Pfad berechnen

Nun können wir mittels "*Shortest path (point to point)*" (aus dem Toolset "Network analyses" den kürzesten Pfad zwischen zwei Knotenpunkten auf dem Netzwerk berechnen. Starten sie das Tool und wählen sie als Input Datensatz ("Vector Layer representing network") *osm_highway_prepared* aus.

Die Start- und Endpunkte können Sie interaktiv in der Karte setzen. Klicken Sie dazu auf das Symbol neben den entsprechenden Feldern ("Start point" bzw. "End point") und klicken Sie in der Karte an die gewünschten Stellen. Führen Sie das Tool mit "Run" aus.

Visualisieren Sie nun den neuen Layer "Shortest Path" so, dass er gut ersichtlich ist.

Hinweis: Auch GRASS GIS bietet einen Shortest Path Algorithmus an (v.net.path). Dieser ist darauf ausgelegt, viele kürzeste Pfade für viele Punkte zu berechnen, und nimmt als Input deshalb ein Textfile.

58.3. Übung 3.3: Mit ORS Routing vergleichen

Nun wollen wir diese Route mit derjenigen eines professionellen Routing Services vergleichen. https://maps.openrouteservice.org/ bietet ihre Dienste bis zu einem bestimmten Kontingent kostenlos an. Installieren Sie das Plug-In "ORS Tools" um diesen Service zu nutzen.

Führen Sie das Tool nach der Installation via Web → ORS Tools → ORS Tools aus. Fügen Sie bei Settings ( ) → "API Key" den Schlüssel ein. Den Schlüssel findet ihr auf Moodle.

Über diesen Schlüssel wird sichergestellt, dass die Anzahl Abfragen pro Minute und Tag ein gewisses Maximum nicht überschreiten.

Geben Sie Start und Endpunkt mit der Maus ein (klick auf das +) und orientieren sich dabei an dem Layer "Shortest_Path" (aus der vorherigen Übung). Allenfalls verschwindet das "ORS Tools" Fenster, sie können es aber über die Toolbar (siehe Abbildung 58.2) wieder aufrufen.

Abbildung 58.2.: ORS Routing

Mit einem Klick auf dieses Symbol erscheint das ORS Routing wieder.

Führen Sie die Berechnung mit "Apply" aus und vergleichen Sie den resultierenden Pfad mit dem "Shortest Path" aus Kapitel 58.2. Führen Sie die gleiche Berechnung mit verschiedenen Einstellungen durch (kürzeste Route, schnellste Route, Fahrrad, zu Fuss). Vergleichen Sie die unterschiedlichen Routen mit unserer eigenen Berechnung und visualisieren Sie diese in einer Karte.

Berechnen Sie nun mit OSM Routing den kürzesten Pfad zwischen dem Campus Grüental und dem Campus Reidbach, auch wieder je einmal mit der Verkehrsmodalität Auto, Fahrrad und Fussweg. Vergleichen Sie die drei Resultate.

> **Hinweis**
>
> Das `ORS Tools Plugin` bietet keine Möglichkeit, die blauen Linien zu entfernen, die die ausgewählten Punkte auf der Karte verbinden (https://github.com/GIScience/orstools-qgis-plugin/issues/120). Diese Linien sind temporär, d.h. sie werden beim nächsten Öffnen des QGIS-Projekts wieder verschwinden.

59. Aufgabe 4: Traveling Salesperson

Das Problem des Handlungsreisenden (engl. Traveling Salesperson Problem) ist ein kombinatorisches Optimierungsproblem, in dem die Aufgabe darin besteht, eine Reihenfolge für den Besuch mehrerer Orte so zu wählen, dass keine Station ausser der ersten mehr als einmal besucht wird, die gesamte Reisestrecke des Handlungsreisenden möglichst kurz und die erste Station gleich wie letzten Station ist.

59.1. Übung 4.1: Traveling Salesperson für Campus Standorte

Angenommen Sie sind ein Kurrierdienst und müssen ausgehend von der Halbinsel Au aus alle Campus Standorte der ZHAW Wädenswil besuchen. Sie wollen die Route so optimieren, dass Sie die kürzeste Route Sie das gleichnamige Tool (`v.net.salesmen`) um genau dieses Problem zu lösen.

1. Erstellen Sie dazu als erstes eine neue Geopackage Datei (Layer→Create Layer→New Geopackage Layer) um die Campus Standorte zu erfassen.

   - Mit dem Feld "Database" ist der Pfad inkl. Dateiname der zu erstellenden Datei gemeint. Wählen Sie hier an einem geeigneten Speicherort den Dateiname "campus_waedenswil.gpkg"
   - "Table name" ist der Name des Layers innerhalb der Geopackage Datei (im Gegensatz zu einem Shapefile können innerhalb eines Geopackage mehrere Layers von unterschiedlichen Datentypen koexistieren)

   - Wählen Sie bei "Geometry Type" "Point" aus und bei CRS EPSG 2056
   - Fügen Sie eine Spalte in der Attributtabelle hinzu indem Sie unter "New field"→"Name" den Wert "Standort_name" eingeben (Hier wollen wir "Grüental", "Reidbach".. erfassen) Wählen Sie einen geeigneten Datentyp sowie eine geeignete Maximallänge
   - Bestätigen Sie mittels "Add field to list"
   - Erstellen Sie das Geopackage mit "OK"

2. Starten Sie mit einem Klick auf **den Stift** die Editiersession und fügen Sie mit dem "Add Feature" Werkzeug Features hinzu. Digitalisieren Sie so die Campus Standorte (Grüental,

Reidbach, Seifenstreuli, Schloss) sowie den Ausgangspunkt (Halbinsel Au). Wählen Sie pro Standort einen für Sie geeigneten Punkt, möglichst auf dem OSM Strassennetz.
3. Speichern Sie die erfassten Punkte mit einem Klick auf "Save Edits".
4. Starten Sie nun das Tool `v.net.salesman` (über die Processing Toolbox) und wählen Sie als Input Layer den OSM Strassendatensatz und als "Center Point Layer" die eben digitalisierten Standorte.
5. v.out.ogr output type: auto.
6. Betrachten Sie die Outputdaten.

59.2. Übung 4.2: Traveling Salesperson für mehr Standorte

Der Traveling-Salesperson-Pfad für fünf Punkte zu berechnen ist relativ trivial und könnte "von Hand" gerechnet werden. Anspruchsvoller wird es jedoch, wenn sich die Anzahl der Standorte erhöht. Nehmen wir an, Sie wollen eine Einkaufstour durch alle Läden in Wädenswil machen: Nutzen Sie die OSM Daten und `v.net.salesman` um eine sinnvolle Route zu berechnen.

1. OSM Daten der Läden laden: Vector→QuickOSM→QuickOSM
2. key "shop"→Run query sowie Gebiet wählen (siehe dazu Kapitel 55.6)
3. Punkt-Daten der Shops in CRS 2056 konvertieren (reproject, siehe Kapitel 56.1). Clippen ist fakultativ (nicht erreichbare Knotenpunkte werden schlicht ignoriert)
4. `v.net.salesman` mit diesen Standorten durchführen

59.3. Übung 4.3 (fakultativ, Guezli-Challenge): Traveling Salesperson für noch mehr Standorte

Um unsere Rechenmaschine richtig herauszufordern, können wir den Traveling Salesperson Pfad für alle Gebäudestandorte in Wädenswil berechnen. Nutzen Sie hierzu QuickOSM um "building" herunterzuladen. Reprojizieren Sie die Polygon-Daten in CRS 2056 und konvertieren Sie diese in Punkte, indem Sie das Centroid pro Polygon berechnen (Tool "Polygon Centroids"). Berechnen Sie anschliessend den Traveling Salesperson. Ermitteln Sie die Gesamtdistanz dieses Pfades, indem Sie mit dem Field Calculator die Länge pro Segment rechnen (length) und anschliessend die Summe aller Längen ermitteln (View→Statistical Summary). **Wer zuerst die korrekte Distanz in den Chat schreibt, wird mit Ruhm und Ehre belohnt und zur/zum "AGI Studentin/Student des Tages" erkürt!**

- Für viele klassische Fragestellungen (z.B. shortest path, traveling salesmen) bietet QGIS / GRASS einen passenden Algorithmus
- Die Tools werden teilweise sehr unterschiedlich angesprochen (shortest path braucht Textfiles, Traveling salesmen braucht Punkt-Features) und liefern unterschiedliche Outputs

Teil X.

Netzwerkanalyse III

Angenommen Sie sind auf Wohnungssuche in Wädenswil. Dabei gilt es nebst dem Budget viele wichtige raumgebundene Variablen zu berücksichtigen, dazu verwenden Sie natürlich QGIS. Sie wollen drei Kriterien untersuchen:

1. Laufdistanz zur nächsten Entsorgungsstelle
2. Erschliessung an die öffentlichen Verkehrsmittel (unter Berücksichtigung des Fahrplans)
3. Distanz zur Durchfahrtsstrasse

Wir werden für jeden dieser drei Kriterien einen Rasterdatensatz kreieren, den wir zum Schluss miteinander verrechnen können. So finden wir den optimalen Standort unter der Berücksichtigung aller drei Kriterien. Starten Sie dazu QGIS und laden Sie folgende Daten in das Projekt:

1. Strassennetz Wädenswil ("osm_highway_prepared.gpkg", siehe Tabelle 55.1)
2. Entsorgungsstellen Wädenswil ("abfallentsorgung_waedenswil.gpkg", siehe Tabelle 55.1)
3. Gemeindegrenze Wädenswil ("Gemeinde_Waedenswil.gpkg", siehe Tabelle 55.1)
4. Optional: OSM Hintergrundkarte grau eingefärbt (siehe Kapitel 58.1)

Übungsziele

- Sie sind in der Lage, einfache Reisezeitberechnungen in QGIS selber durchzuführen.
- Sie erweitern Ihr Skillset zur Umwandlung von verschiedenen Geodatentypen (Vektor zu Raster, Raster zu Vektor, Linien zu Punkte, usw.)
- Sie können eine einfache Multi-Kriterien-Analyse (MCA) mit Raster-Daten in QGIS selbständig durchführen.
- Sie können einfache Map Overlay-Operationen in QGIS selber ausführen.

60. Aufgabe 5: Entsorgungsstelle

60.1. Übung 5.1: Linien in Segmente unterteilen

Da sie gerne hopfenhaltige Getränke in Dosen und Glasflaschen konsumieren, ist Ihnen bei der Standortwahl die Distanz zur nächsten Entsorgungsstelle sehr wichtig. Je näher die Wohnung an einer solchen Entsorgungsstelle wäre, desto weniger weit müssten Sie das Recycling Material mit Ihrem Wägelchen durch die Gegend fahren. Wir berechnen deshalb die Isochronen Linien zu den Entsorgungsstellen auf dem Strassennetz.

- Starten Sie das Tool `v.net.iso` um die Isochronen-Linien auf dem Strassennetz zu berechnen.
- Wählen Sie als Input Vector Line das Strassennetz
- Als "Center Points layer" wählen Sie die Entsorgungsstellen
- Im Feld "Cost for Isolines" können Sie die Schwellenwerte festlegen, bei denen die Isochronenlinien gezogen werden sollen. Die Einheit entspricht den "Map units" (Meter) und werden kommagetrennt eingegeben. Um den vollen Effekt des Tools auszukosten verwenden wir eine grosse Anzahl von Schwellenwerten (z.B. alle 200m von 0 bis 3 km):

  ```
  200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800, 3000
  ```

- Wählen Sie wie immer den v.out.ogr output type: "auto"
- Führen Sie das Werkzeug anschliessend mit "Run" aus.

Visualisieren Sie das Resultat in dem Sie die Symbolisierung des neu entstanden Linienlayers entsprechend anpassen (Spalte "cat" einfärben). Sind das nun bereits Isolinien?

60.2. Übung 5.2: Linien in Punkte Umwandeln

Um aus den Linien eine Oberfläche zu erhalten müssen wir die Kategorien aus dem Output von `v.net.iso` interpolieren. Dies funktioniert besser mit Punkten als Linien. In einem ersten Schritt müssen wir also den Output aus `v.net.iso` in Punkte umwandeln. Verwenden Sie hierfür das Werkzeug "Convert Lines to Points" (SAGA). Belassen sie das Häkchen bei "Insert

additional Points" und setzten Sie die Distanz ("Insertion Distance") auf 100 Meter. Führen Sie das Werkzeug mit "run" aus.

60.3. Übung 5.3: Punkte interpolieren

Nun haben wir Punkte, die mit einer Interpolation in eine Oberfläche überführt werden können. Verwenden Sie dazu das Tool "Inverse distance weighted" (SAGA). Überlegen sie, was Sie interpolieren möchten und legen sie das entsprechende Feld in der Option "Attribute" fest. Zusätzlich können Sie folgende Parameter wählen:

- Cellsize: `25`
- Search Range: `[0] local`
- Maximum Search Distance: `500`
- Weighting Function: `[1] Inverse Distance to a power`

Clippen sie den Output anschliessen auf die Gemeindegrenze mit dem Werkzeug "Clip Raster by mask layer". Speichern Sie den geclippten Layer mit dem Namen abfall_raster.tif in ihrem Projektordner.

Um den Datensatz zu visualisieren können sie im Reiter "Symbology" der Layereigenschaften den Rendertype "Singleband Pseudocolor" auswählen. Wählen Sie einen geeigneten Farbverlauf und klicken sie auf "classify" und anschliessend auf "ok".

> **Hinweis**
>
> Gruppieren Sie alle Layers im Zusammenhang mit den Entsorungsstellen mittels *Selektion > Rechtsklick > Group selected* um den Überblick zu behalten.

60.4. Übung 5.4: Isolinien berechnen

Dieser nächste Schritt dient nur zu Illustrationszwecken: Wir möchten aus der generierten Oberfläche Isolinien berechnen. Nutzen sie hierfür das Tool *Raster→Extraktion→Contour…*. Spielen sie mit verschiedenen Intervallen (Interval between..) rum bis Ihnen eine Darstellung gefällt. Überlagern Sie den interpolierten Raster, die Isolinien sowie die Entsorgungsstellen und freuen sie sich ab diesem Ergebnis!

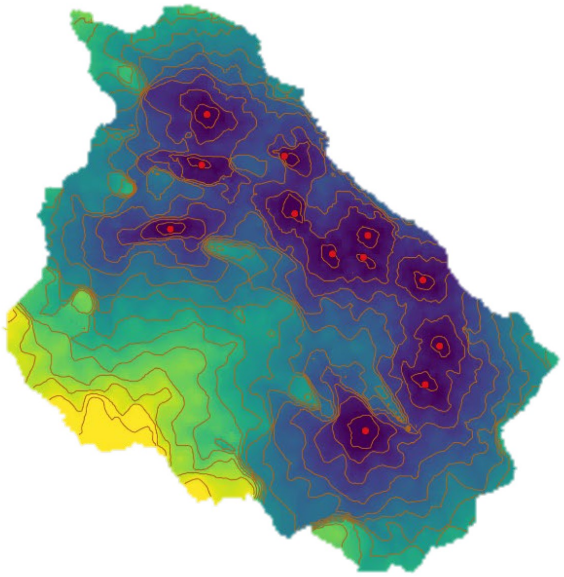


Abbildung 60.1.: Isolinien berechnen

61. Aufgabe 6: ÖV-Güteklassen

61.1. Übung 6.1: Datensatz Clippen und betrachten

Als weiteres Kriterium wollen Sie eine gute Erschliessung mit dem öffentlichen Verkehr. Dabei geht um den ÖV-Anschluss unter Berücksichtigung des elektronischen Fahrplans. Dafür hat das ARE einen Datensatz herausgeben, den Sie hier auffinden können: https://s.geo.admin.ch/7e80a8bd28

Laden Sie den Datensatz "ÖV-Güteklassen ARE" herunter und importieren Sie ihn in Ihr Projekt.

Tipp zum Herunterladen!

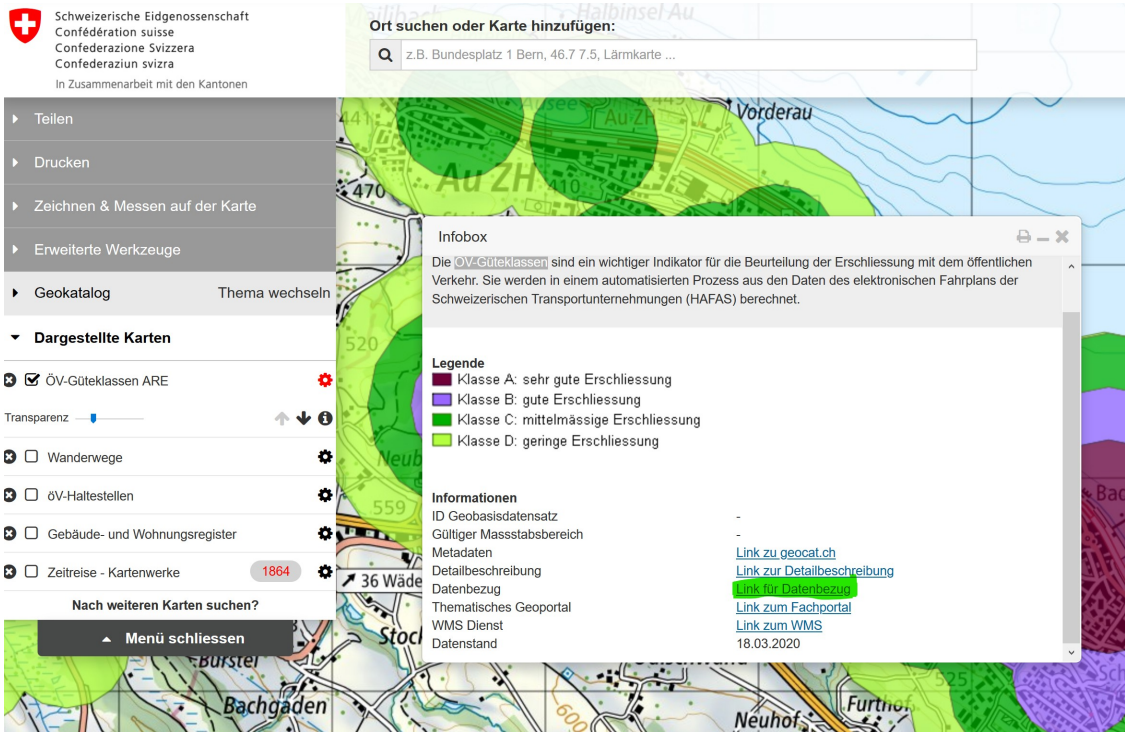


Abbildung 61.1.: ÖV-Güteklassen

Clippen Sie den Datensatz mit der Gemeindegrenze von Wädenswil. Symbolisieren Sie den Datensatz anhand des Feldes “KLASSE” und betrachten Sie anschliessend die Attributtabelle.

61.2. Übung 6.2: Buchstaben in nummerische Kategorien überführen

Da es sich bei den Klassen um Buchstaben handelt, können wir sie für unsere Analyse in dieser Form nicht verwenden. Wir müssen die Buchstaben noch in Zahlen konvertieren (A zu 1, B zu 2 usw.). Verwenden Sie hierfür den Field Calculator der Attributtabelle. Versuchen Sie die Konversion Buchstaben in Zahlen mit einer if() Funktion um zu setzen. Falls sie nicht weiter kommen finden Sie weiter unten einen Lösungsvorschlag.

Hinweise

- Bevor Sie die Funktion anwenden können müssen Sie noch den “Output field name” definieren.
- Sobald sie den Field Calculator ausführen wechselt QGIS in den Editiermodus. Änderungen werden nur gespeichert, wenn die Editiersession beendet wird.
- Wenn sie im mittleren Bereich auf unter “Fields and Values” auf ein Feld klicken haben sie rechts die Möglichkeit, die Werte dieser Felder zu betrachten und ebenfalls mit Doppelklick in ihre Formel zu übernehmen.

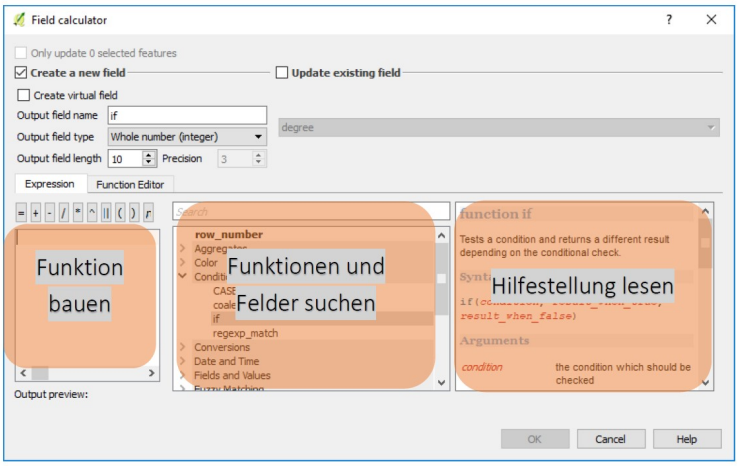


Abbildung 61.2.: Field Calculator

Zur Erinnerung: Der Field Calculator besteht aus drei Bereichen: Im mittleren Bereich können Sie Funktionen sowie Felder der Attributtabelle suchen und mittels einfachem anklicken Hilfestellung erhalten (rechts). Mittels Doppelklick wird die Funktion oder die Feldbezeichnung in den “Function Editor” (links) übertragen. Hier wird die eigentliche Funktion zusammengebaut die ausgeführt werden soll.

Hinweis

```
if("KLASSE" = 'A' ,1,
  if("KLASSE" = 'B' ,2,
    if("KLASSE" = 'C' ,3,
      if("KLASSE" = 'D' ,4,0)
    )
```

```
  )
)
```

61.3. Übung 6.3: Polygon in Raster konvertieren

Konvertieren Sie das Polygon nun in ein Raster mit Werkzeug “Rasterize (Vector To Raster)”. Führen Sie das Tool mit den folgenden Einstellungen aus:

- Field to use for a burn-in value: `KLASSE (numerisch)`
- Output raster size units: `Georeferenced Units`
- Width/Horizontal resolution: `25`
- Height/Vertical resolution: `25`

Beachten Sie, dass nun alle Bereiche in Wädenswil, die keiner Kategorie zugewiesen waren nun keinen Wert erhalten haben. Für unsere Abstufung bräuchten wir hier aber den Wert “5” (1 – 4 haben wir bereits zugewiesen). Benutzen Sie das Tool “r.null” um alle Null Werte in 5 zu konvertieren (“the value to replace the null value by”).

Speichern Sie den Output mit dem Namen **oev_raster.tif** in ihrem Projektordner und gruppieren Sie alle Layers im Zusammenhang mit den ÖV-Güteklassen.

62. Aufgabe 7: Zentralitätsmasse

62.1. Übung 7.1: Zentralitätsmasse betrachten und auswählen

Da Sie aber auf keinen Fall an einer Durchfahrtsstrasse wohnen möchten, müssen Sie dies in der Wohnungssuche ebenfalls berücksichtigen. Laden Sie deshalb den Datensatz "waedenswil_centrality.gpkg" aus Kapitel 57.3. Falls Sie diesen nicht mehr haben, können Sie die Zentralitätsmasse neu berechnen (`v.net` sowie `v.net.centrality`) oder Sie verwenden den Datensatz "waedenswil_centrality.gpkg" in Tabelle 55.1. Prüfen Sie anhand der Symbolisierung, welches Mass "Durchfahrtsstrassen" am besten abbildet.

62.2. Übung 7.2: Zentralitätsmasse in Fläche überführen

Das gewählte Zentralitätsmass können wir nun ebenfalls mit dem Tool "Inverse distance weighted" (SAGA) in eine Oberfläche überführen. Führen sie das Tool analog Kapitel 60.3, mit dem gewählten Zenralitätsmass aus. Sie können die Parameter des Tools nach eigenem Gutdünken auch anpassen (wir empfehlen auf jedenfall die gleiche Output Cellsize von 25m zu verwenden um im Anschluss die Flächen miteinander verrechnen zu können).

Clippen sie anschliessend den Output mit dem Werkezeug "Clip raster by Mask Layer" (GDAL). Speichern Sie den Output unter **centrality_raster.tif** in ihrem Projektordner und gruppieren Sie alle Layers im Zusammenhang mit den Zentralitätsmassen.

63. Aufgabe 8: Flächen verrechnen

In dieser Übung verschneiden wir die Informationen aus Entsorgungsstellen (abfall_raster.tif), ÖV-Güteklassen (oev_raster.tif), und Zentralitätsmass (centrality_raster.tif), die wir in den vorherigen Übungen berechnet und in Raster überführt haben. Durch die Verschneidung können wir den optimalen Wohnort eruieren. Diese Methode ist auch als Multikriterienevaluation bekannt und kennt ihr bereits vom GIS "Basic" Modul. In einem ersten Schritt müssen wir alle drei Rasterdatensätze auf eine einheitliche Skala (z.B. 0 – 100) bringen um sie anschliessend miteinander verrechnen zu können. Dafür brauchen wir die Minimum- und Maximumwerte der drei Raster Datensätzen oev_raster.tif, centrality_raster.tif, abfall_raster.tif. Gehen Sie dafür in die Properties→Information von jedem Layer und notieren Sie sich die minimalen und maximalen Zellenwerte.

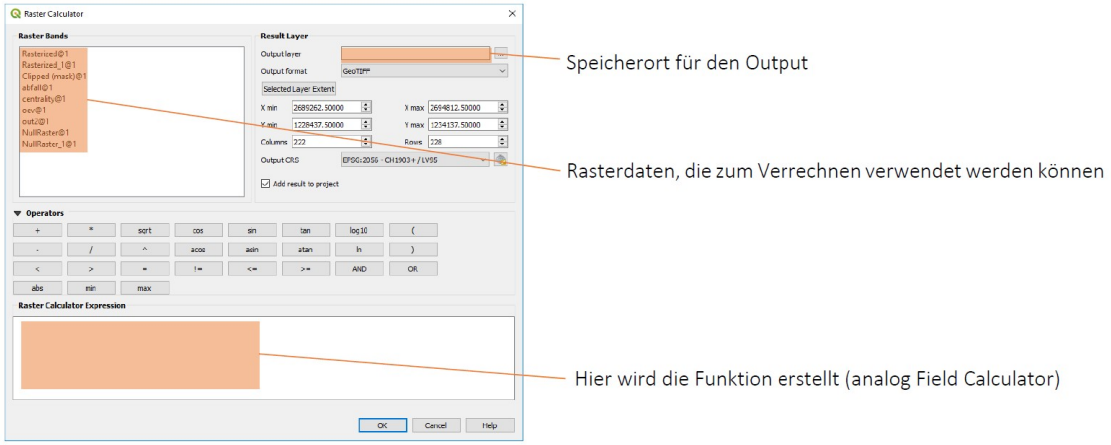
63.1. Übung 8.1: Rasterdatensätze skalieren

Öffnen Sie anschliessend das Tool `Rescale Raster` und skalieren Sie jeden der drei Rasterdatensätze jeweils so, dass Sie Werte von 0 bis 100 erhalten. Geben Sie dabei bei "New minimum value" 0 und bei "New maximum value" 100 ein.

Speichern Sie die Ergebnisse unter folgenden Namen ab: oev_scaled.tif, abfall_scaled.tif, centrality_scaled.tif.

63.2. Übung 8.2: Zusammenführen mit Raster Calculator

Nutzen Sie nun den Raster Calculator (Raster→Raster Calculator) um die drei Rasterdatensätze miteinander zu verrechnen.



Die einfachste Variante ist die Berechnung des Mittelwertes der drei Rasterdatensätze. Dafür muss man alle drei summieren und durch drei dividieren:

$$raster_{neu} = \frac{oev_{scaled} + abfall_{scaled} + centrality_{scaled}}{3}$$

Optional kann man auch die drei Rasterdatensätze unterschiedlich gewichten, wie beispielweise nachstehende Berechnung zeigt. Beachten Sie, dass in diesem Fall nicht mehr durch 3, sondern durch die Summe der Gewichte dividiert wird. Visualisieren und interpretieren Sie anschliessend das Resultat.

$$raster_{neu} = \frac{oev_{scaled} \times 1 + abfall_{scaled} \times 10 + centrality_{scaled} \times 5}{16}$$

Zum Abschluss clippen Sie den Output mit dem Werkezeug `Clip raster by Mask Layer (GDAL)` auf die Gemeindegrenze von Wädenswil. Speichern Sie den Output unter final_result.tif in ihrem Projektordner ab. Passen Sie die Symbolisierung entsprechend an.

64. Leistungsnachweis

Um den Leistungsnachweis für Netzwerkanalyse zu erbringen, müsst ihr die Übungen aus Kapitel 60, Kapitel 61, Kapitel 62 und Kapitel 63, ausführen und dokumentieren (maximal zwei A4-Seiten).

- Abgeben müsst Ihr ein PDF Dokument mit Eurem Lösungsansatz der Multikriterien-Analyse (Kapitel 63) inkl. einer kurzen Beschreibung (ca. 400 - 600 Wörter), wie Ihr hier vorgegangen seid.
- Beschreibt vor allem auch, was für ein Zentralitätsmass Ihr für die Teilfrage "Durchfahrtsstrasse" gewählt habt. Begründet Eure Wahl durch einen konzeptionellen Vergleich mit den beiden anderen Zentralitätsmassen.
- Für den Leistungsnachweis erwarten wir, dass Ihr die Kriterien (ÖV, Abfall, Durchfahrtsstrasse) unterschiedlich gewichtet und die Wahl der Gewichte erklärt.
- Achtet darauf, dass Eure Karte die nötigen Elemente enthält (Datenherkunft, Datum, Legende usw.).
- Abgabe erfolgt via Moodle

Teil XI.

WebGIS I

Übungsziele

- Du bist in der Lage auf dem ArcGIS Online Organisationskonto Deine Inhalte zu organisieren und die Freigabeeigenschaften von Inhalten zu ändern.
- Du kannst in ArcGIS Pro eine Karte vorbereiten, eine Zeitanimation aktivieren und ArcGIS Pro Karten auf einem ArcGIS Online Organisationskonto als Web Maps veröffentlichen.
- Du kannst eine Web Map zielgerichtet gestalten und basierend auf bestehenden Web Maps neue Web Map Applikationen erstellen.
- Du weisst, wie Du eine Story Map erstellen und diese mit Inhalten füllen kannst.

65. Übung Web Map

65.1. Übung 1: Erste Schritte mit ArcGIS Online (Online Tutorial)

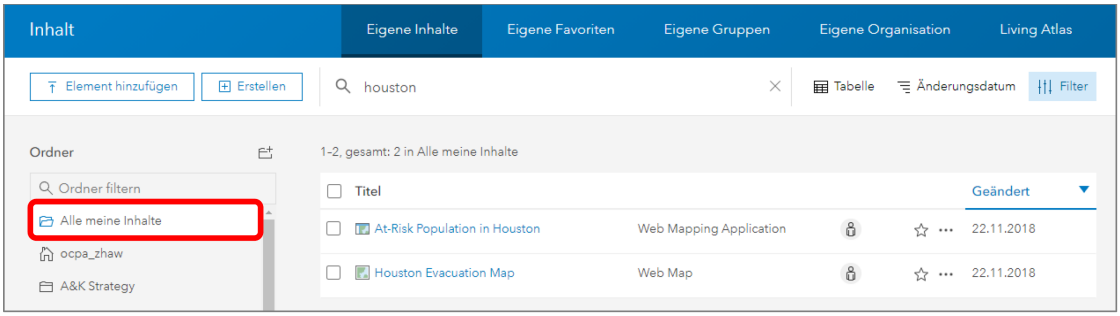
Führe das Online Tutorial «Erste Schritte mit ArcGIS Online» durch. Damit erlernst Du die grundlegenden Funktionalitäten der Kartenerstellung in ArcGIS Online.

Link zum Tutorial

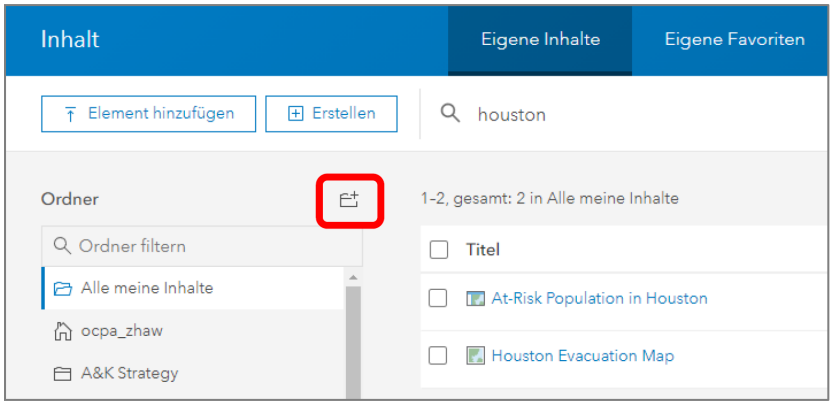
65.2. Übung 2: ArcGIS Online: Inhaltsseite organisieren

In ArcGIS Online kann die eigene Inhaltsseite personalisiert werden. Es können Ordner erstellt werden in denen Inhalte verwaltet und organisiert werden können.

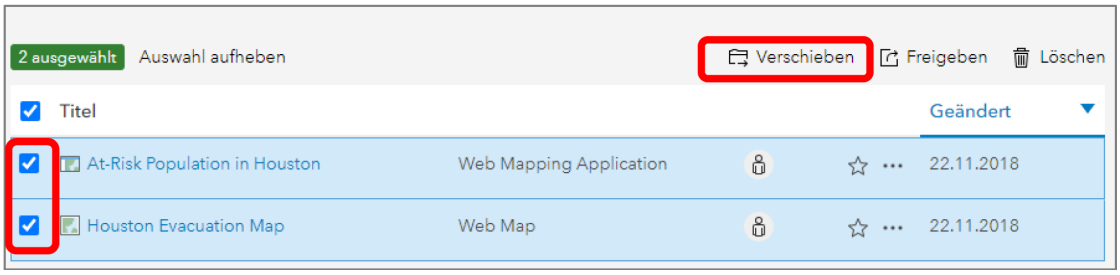
1. Melde Dich mit Deinen persönlichen Nutzerdaten auf dem ArcGIS Online Organisationskonto der ZHAW an: ZHAW ArcGIS Maps
2. Wechsle von der Startseite in das Register Inhalt. Im Ordner «Alle eigenen Inhalte» sollten die beiden Inhalte «At-Risk Population in Houston» (Instant App) und «Houston Evacuation Map» (Web Map) welche Du im Rahmen der ersten Übung (Online Tutorial) erstellt hast, aufgeführt sein.



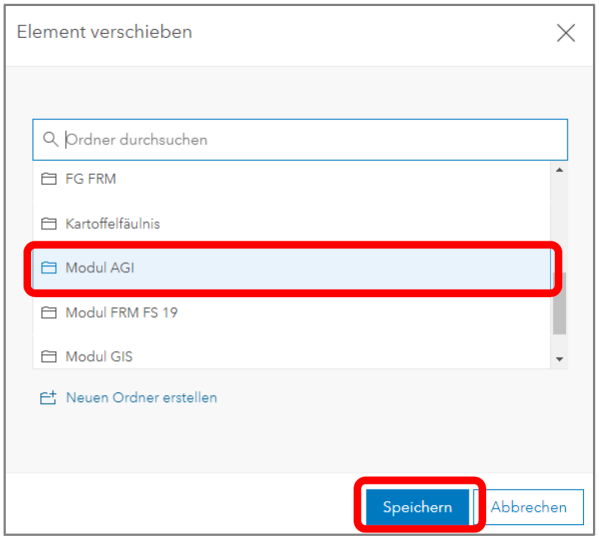
3. Erstelle einen neuen Ordner mit dem Namen «Modul AGI» in dem Du alle Inhalte ablegen kannst, welche Du im Rahmen dieses Moduls auf ArcGIS Online hochlädst. Klicke hierfür auf «Neu», und gib den Namen für den neu zu erstellenden Ordner ein.



4. Verschiebe nun die beiden Inhalte «At-Risk Population in Houston» (Web Mapping Application) und «Houston Evacuation Map» (Web Map) in den soeben neu erstellten Ordner. Gehe hierfür zurück auf den Ordner «Alle meine Inhalte». Selektiere die beiden Inhalte und klicke auf die Option «Verschieben».



Wähle im sich öffnenden Fenster «Element verschieben» den Ordner «Modul AGI» und verschiebe diese Inhalte in den neuen Ordner

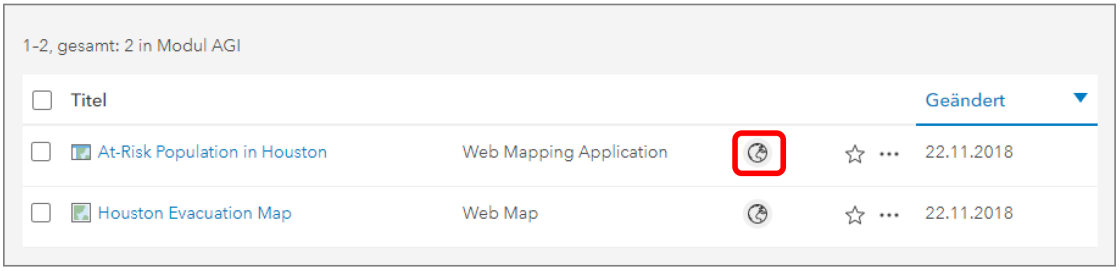


5. Lege alle Deine Inhalte, welche Du im Rahmen dieses Moduls auf ArcGIS Online hochlädst im Ordner «Modul AGI» ab.
6. Du kannst jederzeit neue Ordner erstellen und so Deine Inhalte auf ArcGIS Online organisieren.

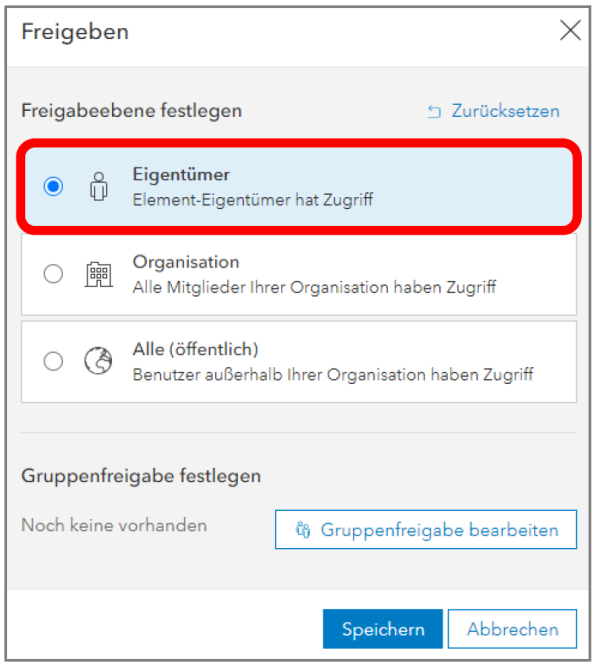
65.3. Übung 3: ArcGIS Online: Freigabeeigenschaften bei bestehenden Inhalten ändern

Bei der Durchführung des Esri Online Tutorials «Erste Schritte mit ArcGIS Online» hast Du sowohl die Web Map als auch die Web Mapping Application für «Alle (öffentlich)» freigegeben. Ändere nun die Freigabe dieser beiden Inhalte, damit diese nur noch von Dir aufgerufen werden können.

1. Zeige die Inhalte im Ordner «Modul AGI».
2. Klicke bei der Web Mapping Application «At-Risk Population in Houston» auf das Globus-Symbol.



3. Ändere im Freigabefenster nun die Freigabeeigenschaften für diesen Inhalt. Diese Anwendung soll nun nicht mehr für die Öffentlichkeit freigegeben sein, sondern nur noch für Dich (Du kannst die Anwendung öffnen, andere haben jedoch keinen Zugriff mehr darauf). Wähle hierfür die Option «Eigentümer».



4. Führe dasselbe für die Web Map «Houston Evacuation Map» durch.

65.4. Übung 4: ArcGIS Pro: Karte einrichten

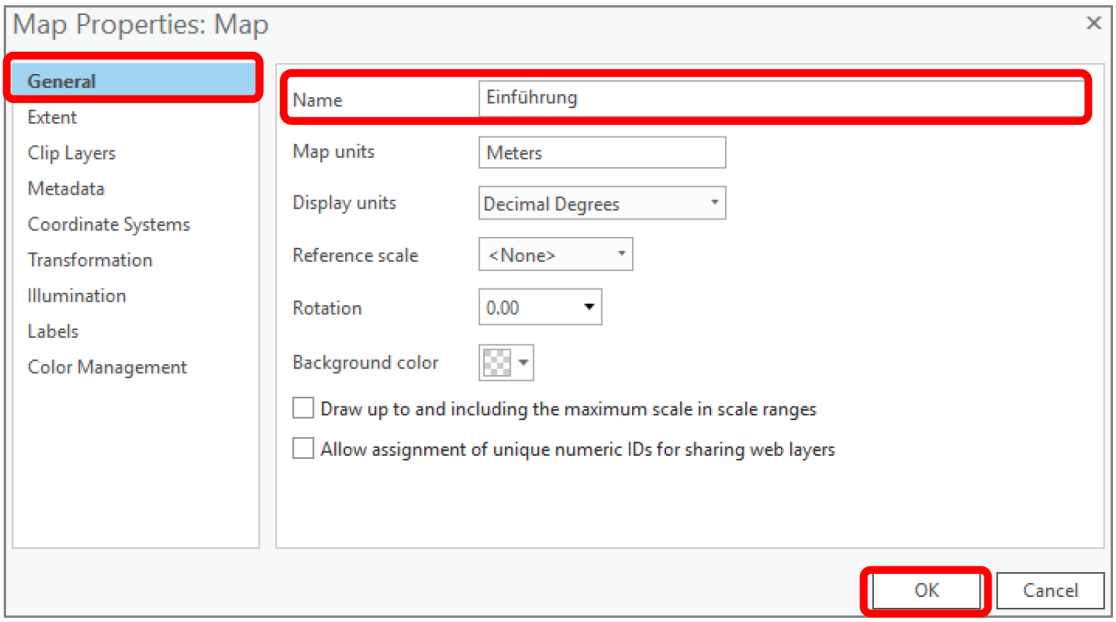
Lade die Daten von Moodle herunter (ZIP-File: Daten Einführungsübung ArcGIS Online), extrahiere die Daten unter einem sinnvollen Ordnerpfad und erstelle ein neues ArcGIS Pro Projekt mit dem Namen E_ArcGIS_Online im Ordner, in dem Du die Übungsdaten extrahiert hast.

> **Tipp**
>
> Übrigens: Du kannst alle Deine Karten (Maps) welche Du im Laufe dieses Unterrichtsblocks erstellst in diesem Projekt speichern

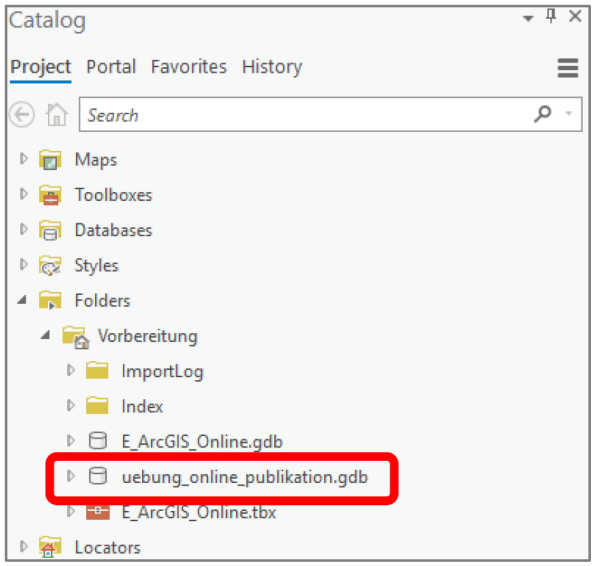
1. Starte ArcGIS Pro und erstelle ein neues ArcGIS Pro Projekt mit dem Namen «E_ArcGIS_Online».

2. Melde Dich mit Deinen persönlichen Benutzerdaten (kürzel_student / PW) in ArcGIS Pro beim ArcGIS Online Portal der ZHAW an (sofern Du noch nicht angemeldet bist).

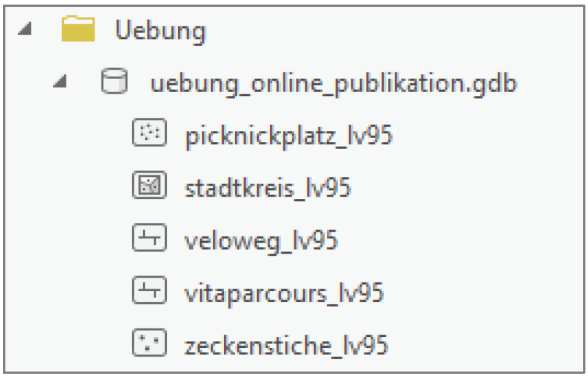
3. Benenne die neue Map als «Einführung» (Contents Pane > Kontextmenü Map > Properties > Map Properties > General Tab > bei Name Eingabe vornehmen > mit OK bestätigen).



4. Gehe in das Catalog Window und überprüfe die Inhalte im Projektordner. Zusätzlich zur Projekt-GDB sollte die Geodatenbank «uebung_online_publikation.gdb» aufgeführt sein. Falls nicht, erstelle im Catalog Pane eine Verbindung zum Ordner her, in dem Du die heutigen Übungsdaten entpackt hast (Catalog Pane > Project > Folders > Kontextmenü Add Folder Connection).



5. Die Geodatenbank «uebung_online_publikation.gdb» enthält folgende Feature Classes:



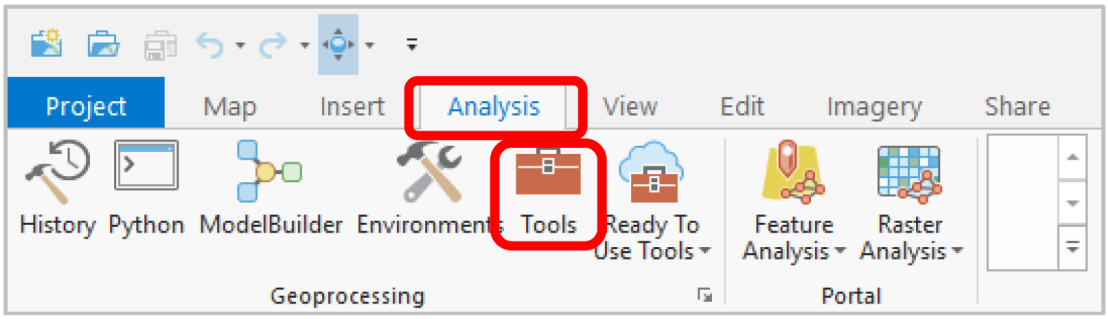
65.5. Übung 5: ArcGIS Pro: Koordinatensystem anpassen

Damit die Daten später auf ArcGIS Online veröffentlicht werden können, muss den Daten das Koordinatensystem «WGS 1984 Web Mercator Auxiliary Sphere» zugewiesen werden. Momentan ist den fünf Layern noch das Schweizerische Koordinatensystem zugewiesen. Führe für jeden einzelnen Layer eine Koordinatentransformation durch.

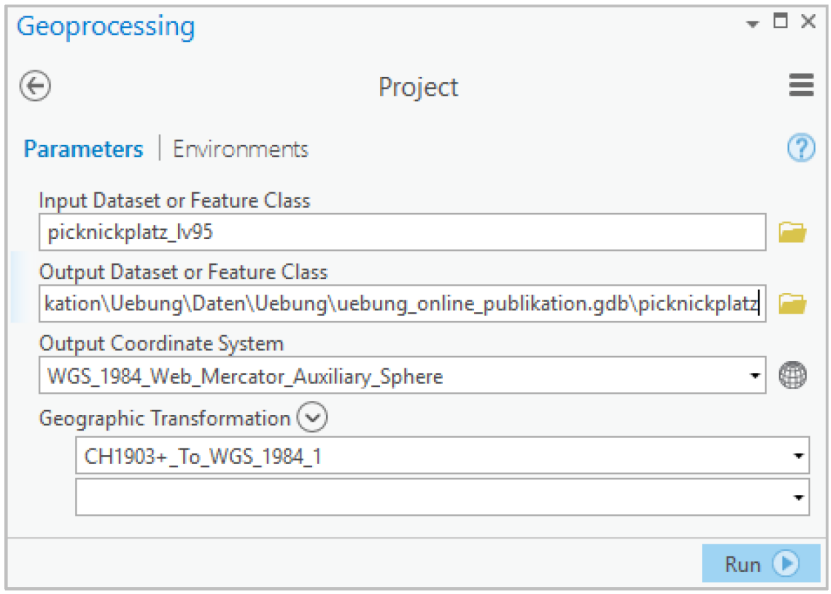
> **Tipp**
>
> Um Probleme bei der Veröffentlichung auf ArcGIS Online zu vermeiden empfiehlt es sich, immer mit dem Koordinatensystem «WGS 1984 Web Mercator Auxiliary Sphere» zu

arbeiten.

1. Öffne das Geoprocessing Window (Menü Analysis > Tools)



2. Suche nach «Project» und öffne das Tool «Project (Data Management Tools). Transformiere die fünf Layers vom Schweizerischen Koordinatensystem in das projizierte weltweite Koordinatensystem. Definiere hierfür die nötigen Werkzeug-Parameter. Speichere die Output Layers in der gleichen Geodatenbank (uebung_online_publikation.gdb), jeweils ohne «_lv95» (bsp. picknickplatz). Das Koordinatensystem WGS 1984 Web Mercator Auxiliary Sphere findest Du unter folgendem Pfad: Projected coordinate system > World > WGS 1984 Web Mercator (auxiliary sphere). Füge es doch zu Deinen Favoriten hinzu.

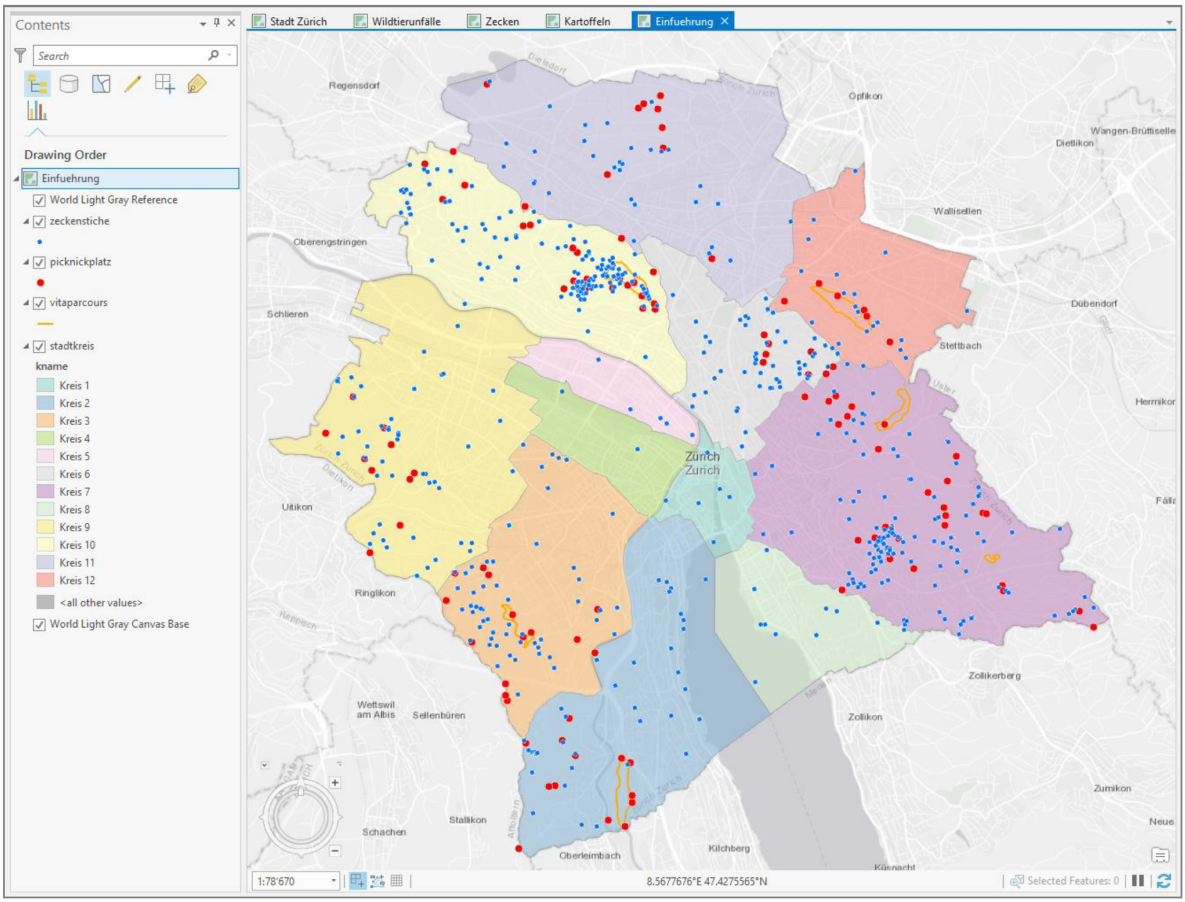


Nach Ausführung des Werkzeugs «Project» werden die Layers automatisch der Karte «Einfuehrung» hinzugefügt.

Wichtig: Entfernen Sie den Layer «veloweg» aus Ihrer Karte.

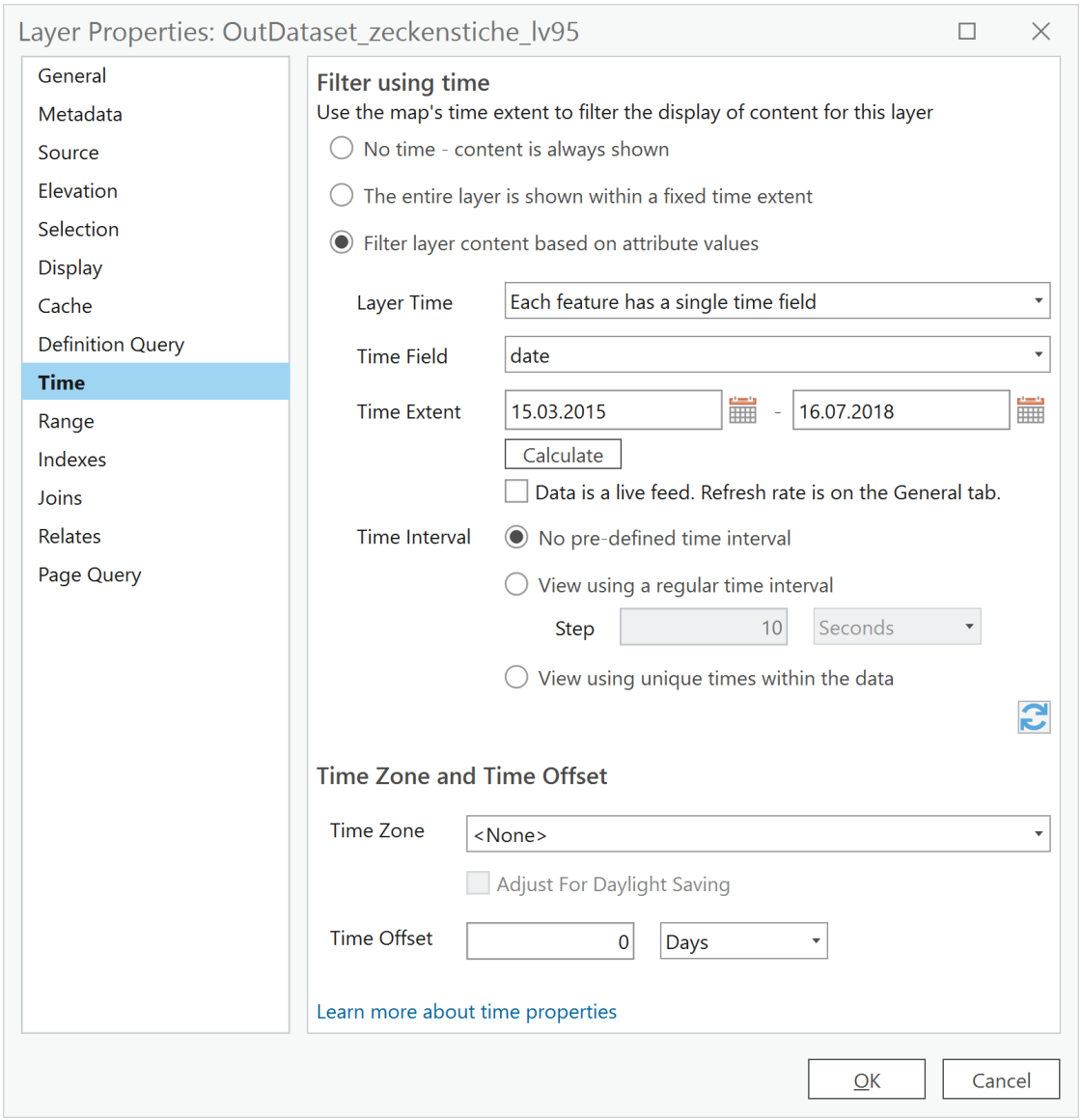
65.6. Übung 6: ArcGIS Pro: Vorbereitung der Karte

Symbolisiere die vier Layer in einer Art und Weise, dass sie gut voneinander unterschieden werden können. Weise dem Layer «stadtkreis» eine Einzelwert-Legende basierend auf dem Feld «kname» zu und versehe diesen Layer mit einer Transparenz. Ändere zudem die Basemap auf «Light Gray Canvas».



Aktiviere nun für den Layer «zeckenstiche» die Zeiteigenschaften:

1. Contents Window > Kontextmenü Layer «zeckenstiche» > Properties.
2. Wähle unter Time folgende Einstellungen und bestätige sie mit OK:
    - Layer Time: Each feature has a single time field
    - Time Field: date



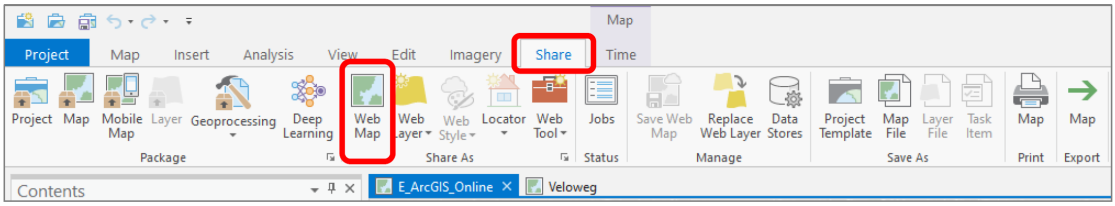
Für den Layer «zeckenstiche» hast Du die Zeiteigenschaften aktiviert und Du kannst nun im Menüband «Time» die Eigenschaften der Zeitanimation definieren und anpassen.

65.7. Übung 7: ArcGIS Pro: Veröffentlichen (Publishing) einer Karte

Im Folgenden wird diese Karte mitsamt all ihren Inhalten, Symbolisierung und Funktionalitäten als Web Map auf ArcGIS Online veröffentlicht (direkt via ArcGIS Pro). Vergewissere Dich,

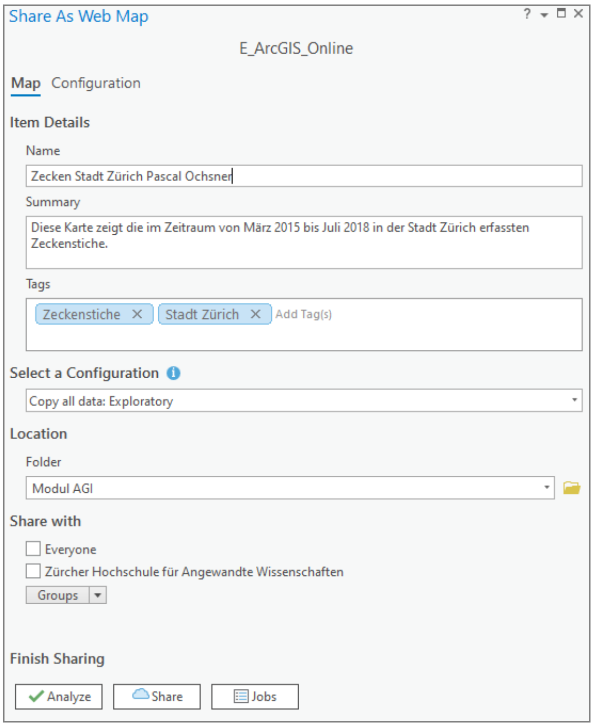
dass Du mit dem ZHAW ArcGIS Online Organisationskonto verbunden bist (Portal Connection) und speichere das ArcGIS Pro Projekt.

1. Wechsle in das Menüband «Share»
2. Wähle anschliessend «Web Map»



3. Gebe der Karte einen Namen (Zecken Stadt Zürich Vorname Name), schreibe eine kurze Zusammen-fassung und definiere Tags. Wähle folgende Einstellungen:

   - Select a Configuration: Copy all data Exploratory
   - Location Folder: Modul AGI (Die Inhalte werden in ArcGIS Online in diesem Ordner abgelegt)
   - Share With: Groups (wähle «Modul Angewandte Geoinformatik HS 2021»; die Karte und die Feature Layers werden nur für diese Gruppe freigegeben).

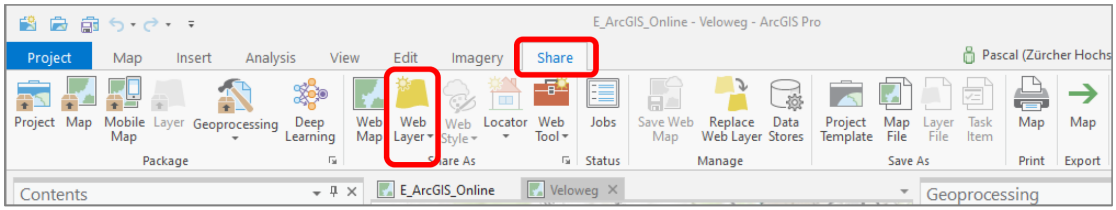


4. Klicke auf «Analyze» und löse bei Bedarf die angezeigten Probleme. Nutze zur Lösung der etwaigen Probleme die ArcGIS Webhelp. Sofern die Warnung «Layer does not have a feature template set» erscheint, kann diese vernachlässigt werden.
5. Die Meldung «00374 Unique numeric IDs are not assigned» kann über das Kontextmenü gelöst werden. (Kontextmenü -> «Auto-Assign IDs Sequentially»)
6. Veröffentliche anschliessend die Web Map mit «Share». Dieser Vorgang kann einige Zeit in Anspruch nehmen.

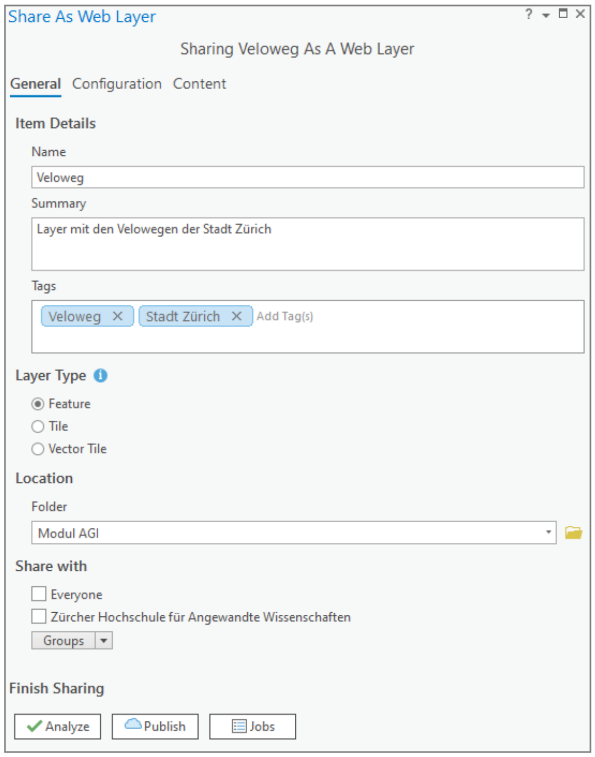
65.8. Übung 8: ArcGIS Pro: Veröffentlichen (Publishing) eines Layers

In der vorangehenden Übung 7 hast Du eine gesamte Karte als Web Map veröffentlicht. Manchmal müssen oder sollen jedoch nur einzelne (oder mehrere) Layers als Web Layers veröffentlicht werden ohne gleichzeitige Erstellung einer Web Map. Erstelle nun in Deinem ArcGIS Pro Projekt eine neue Karte und lade den Layer «veloweg» hinzu. Anschliessend kann dieser Layer als einzelner Feature Layer auf ArcGIS Online veröffentlicht werden.

1. Gehe wiederum ins Menü «Share» und wähle «Web Layer > Publish Web Layer».



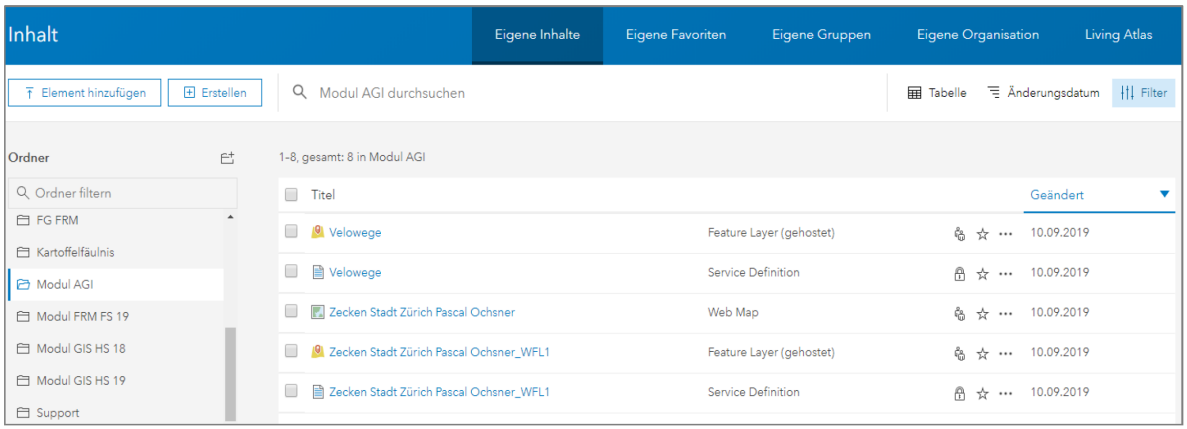
2. Gebe dem Layer den Namen «Velowege Vorname Name», schreibe eine kurze Zusammenfassung und definiere Tags. Wähle folgende Einstellungen:
   - Layer Type: Feature
   - Location Folder: Modul AGI (Der Layer wird in ArcGIS Online in diesem Ordner abgelegt)
   - Share With: Groups (wähle «Modul Angewandte Geoinformatik HS 2022»; der Feature Layer wird nur für diese Gruppe freigegeben).



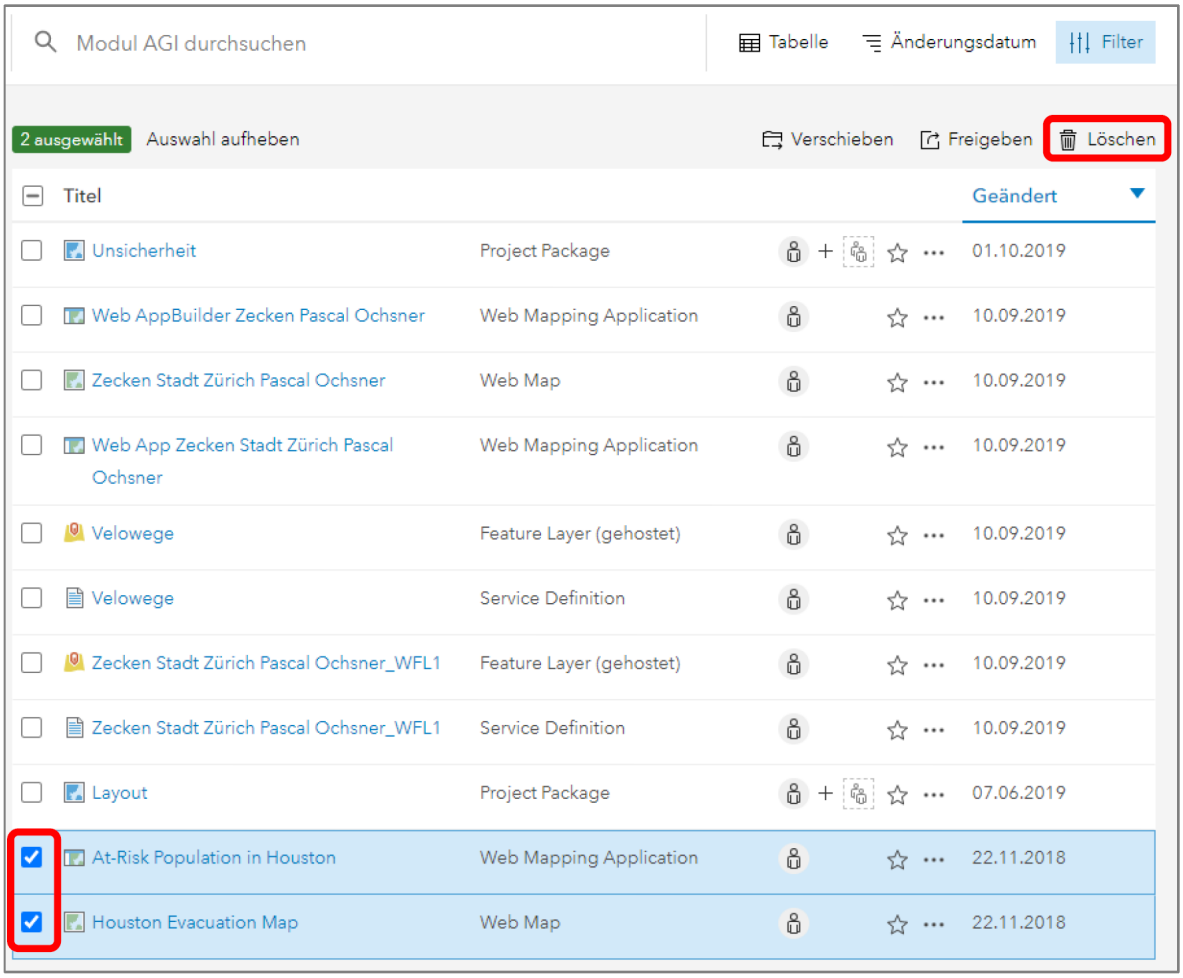
3. Klicke auf «Analyze» und löse angezeigte Probleme.
4. Die Meldung «00374 Unique numeric IDs are not assigned» kann über das Kontextmenü gelöst werden. (Kontextmenü -> «Auto-Assign IDs Sequentially»)
5. Veröffentliche anschliessend den Web Layer «Velowege Vorname Name» mit «Share». Dieser Vorgang kann einige Zeit in Anspruch nehmen.
6. Speichere das ArcGIS Pro Projekt.

65.9. Übung 9: ArcGIS Online: Web Map anpassen

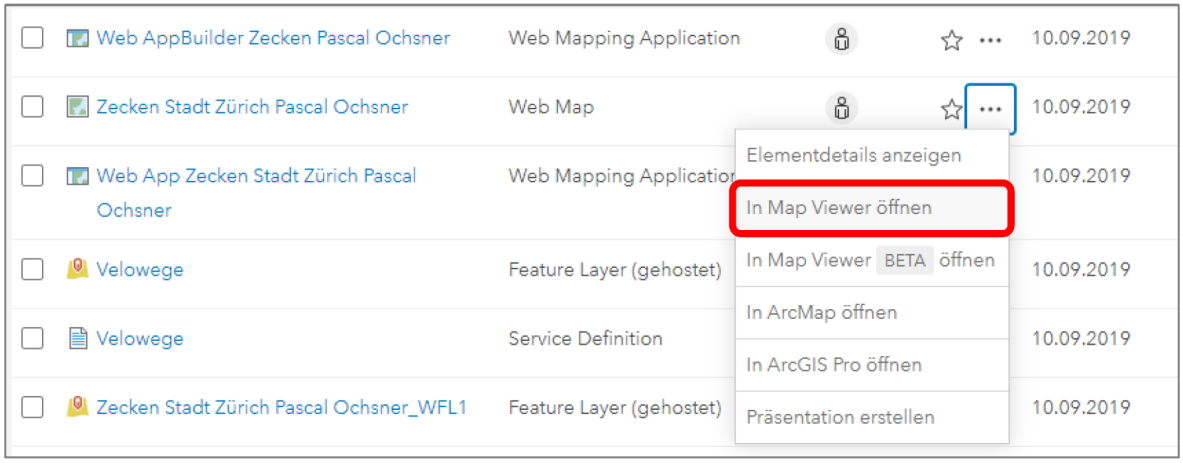
Wechsle in einen Internet Browser und melde Dich beim ZHAW ArcGIS Online Organisationskonto an. Wechsle anschliessend in das Menü «Inhalt» und betrachte die Inhalte im Ordner «Modul AGI». Momentan sollten zwei Feature Layer inkl. Service Definition, zwei Web Maps und eine Web Mapping Application (Instant App) als Inhalte sichtbar sein.



Inhalte können auch gelöscht werden, beispielsweise benötigst Du die Web Mapping Application «At-Risk Population in Houston» und die Web Map «Houston Evacuation Map» nicht mehr. Selektiere die beiden Inhalte und durch Klicken auf «Löschen» kannst Du diese Inhalte unwiderruflich löschen. **Merke Dir:** Sofern Du nur die Web Map löschst, funktioniert die Web Mapping Application nicht mehr. Jede Web Mapping Application basiert immer auf einer Web Map!

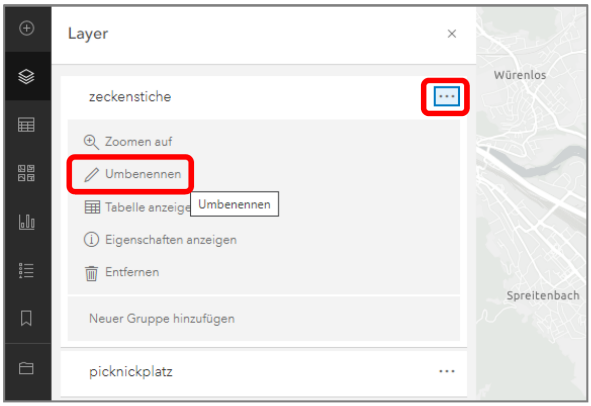


Öffne nun die Web Map «Zecken Stadt Zürich Vorname Name». Klicke auf «…» und wähle «In Map Viewer öffnen» (evtl. in neuem Browser-Fenster öffnen).



Die Web Map enthält die vier Feature Layer «zeckenstiche», «picknickplatz», «vitaparcours» und «stadtkreis». Ändere nun die Namen der Layer (bsp. Zeckenstiche Zeit-animiert statt zeckenstiche):

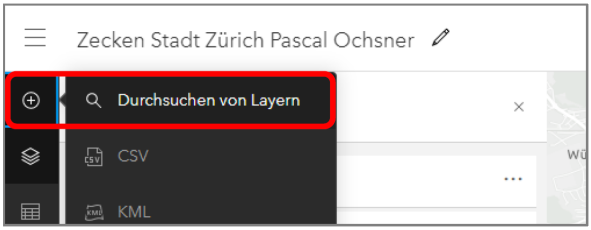
1. Gehe auf «Inhalt».
2. Öffne bei Layer «zeckenstiche» das Kontextmenü durch Klicken auf «...»
3. Wähle «Umbenennen» und ändere den Layernamen auf «Zeckenstiche Zeit-animiert»



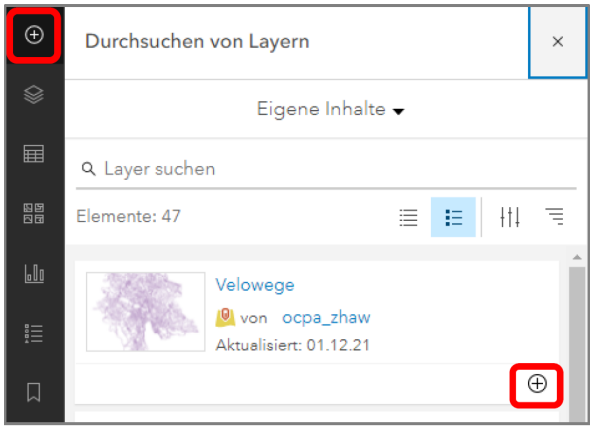
4. Passe die Namen der anderen Layers an (bsp. Grossschreibung)

Füge nun das Veloweg-Netz der Stadt Zürich als zusätzlichen Feature Layer zur Karte.

5. Wähle «Hinzufügen» und dann «Durchsuchen von Layern» (beende bei Bedarf die Zeitanimation).



6. Suche unter «Eigene Inhalte». Gib im Suchfeld «Veloweg» ein. Der Layer sollte dann angezeigt werden. Füge den Layer durch Klicken auf «+» der Karte hinzu. Der Layer wird der Karte hinzugefügt.



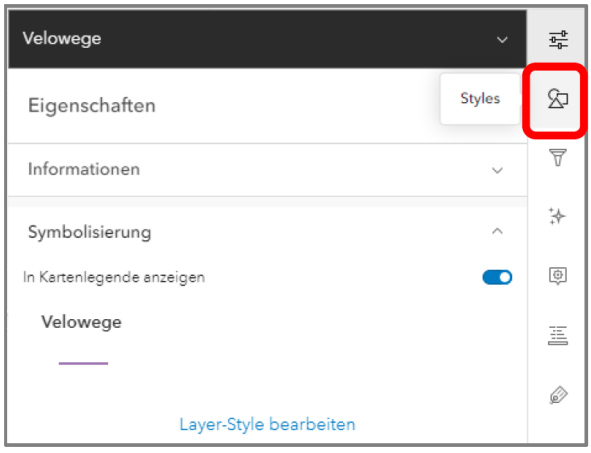
Verändere die Layer-Reihenfolge. Verschiebe den Layer «Velowege» unterhalb von «Vita Parcours».

7. Gehe in den Reiter «Layer».
8. Verschiebe mittels drag and drop den Layer «Velowege» an die gewünschte Position und ändere den Namen des Layers.

   Alternative: Klicke unter Inhalt beim Layer «Velowege» auf die drei vertikalen Punkte und verschiebe den Layer an den gewünschten Ort.

Ändere das Symbol für den Layer «Velowege» (bsp. ein dezentes dunkelgrau).

9. Selektiere den Layer «Velowege» in der Layer-Ansicht.
10. Öffne das Menü «Styles» (vertikales Menü rechts).

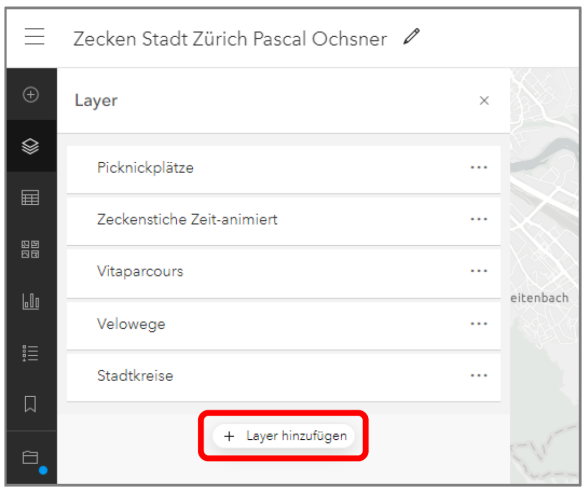


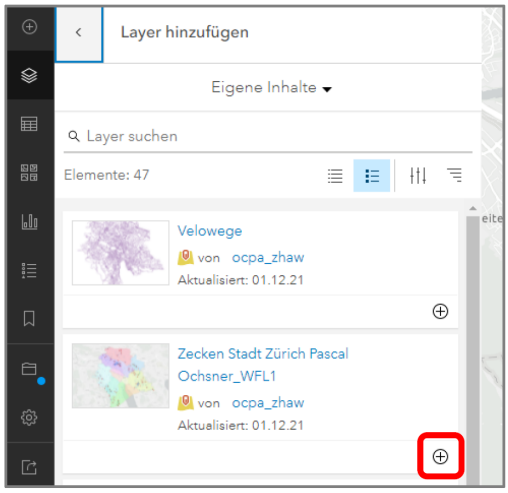
11. Gehe auf «Style-Optionen»
12. Klicke auf das Stift-Symbol und ändere die Linienfarbe und -breite.

13. Bestätige die Eingaben mit “Fertig” > “Fertig”

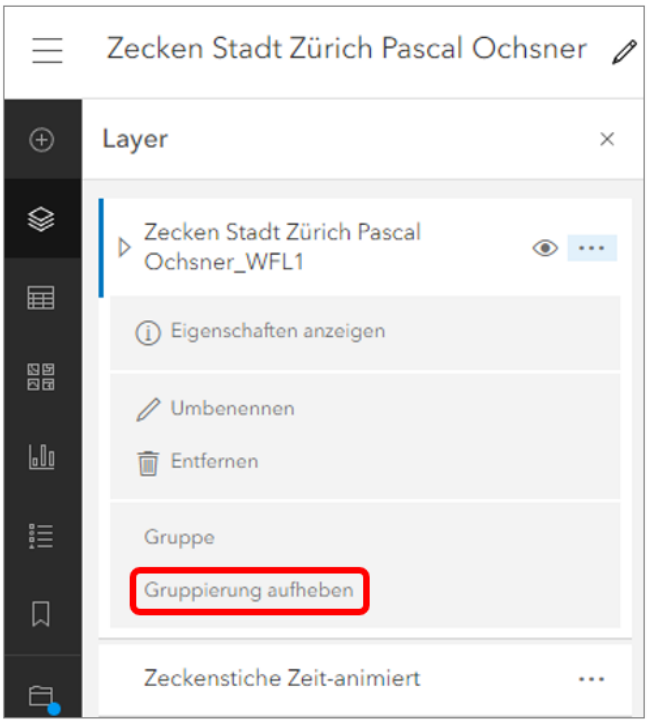
Lade nun den Layer «Zeckenstiche Zeit-animiert» ein zweites Mal zur Webmap. Hierfür musst Du die Web-Feature-Layers «Zecken Stadt Zürich Vorname Name» nochmals zur Karte hinzufügen und anschliessend die Layers, welche nicht doppelt vorhanden sein sollen wieder entfernen. Ändere anschliessend den Style und deaktiviere die Zeitanimation.

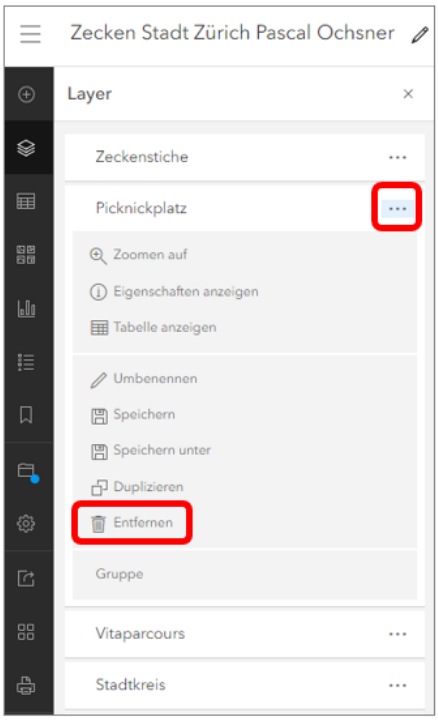
14. Wähle «Hinzufügen». Füge den Web-Layer mit dem Namen «Zecken Stadt Zürich Vorname Name_WFL1» mittels «+» Button zur Karte hinzu.





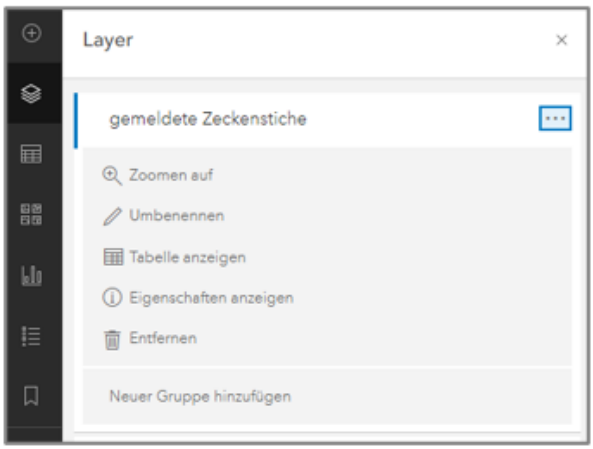
15. Hebe zuerst mittels «Gruppierung aufheben) die Gruppierung auf (dieser Feature Layer enthält vier verschiedene Layers). Entferne anschliessend die Layers «Picknickplatz», «Vitaparcours» und «Stadtkreis» aus der Webmap, denn diese Layers sollen nicht doppelt in der Webmap enthalten sein. Führe dies für jeden Layer einzeln durch.

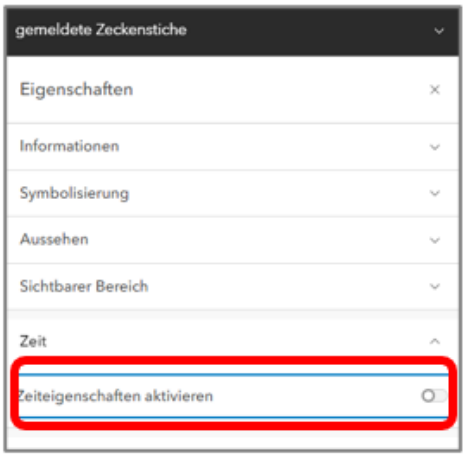




16. Ändere anschliessend beim neu hinzugefügten Zeckenstich-Layer den Namen zu «Gemeldete Zeckenstiche».

17. Deaktiviere nun die Option Zeitanimation im Layer «Gemeldete Zeckenstiche». Gehe in diesem Layer auf «Eigenschaften anzeigen» und deaktiviere die Zeiteigenschaften im entsprechenden Abschnitt.

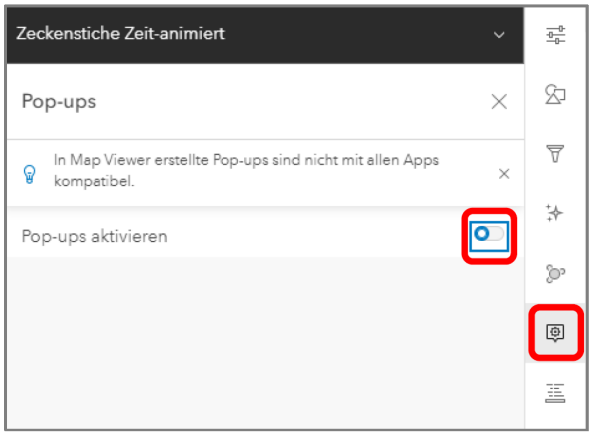




18. Verschiebe danach diesen Layer unterhalb des Layers «Zeckenstiche Zeit-animiert».

19. Verändere beim Layer «Zeckenstiche Zeit-animiert» den Style, so dass sich die animierten Zeckenstiche deutlich von den gemeldeten Zeckenstichen unterscheiden lassen. Wähle beispielsweise ein gelbes Punktsymbol inklusive Blooming-Effekt. Symboleffekte können via Reiter «Effekte» definiert werden.

20. Entferne beim Layer «Zeckenstiche Zeit-animiert» das Pop-up via «Pop-up konfigurieren». Somit wird verhindert, dass das Pop-up beim Anklicken eines Zeckenstich-Standortes zwei Mal aufgerufen wird.



21. Pop-ups können übrigens über das Menü «Pop-ups» benutzerdefiniert angepasst werden. Probiere diverse Optionen für andere Layer aus.

22. Die Zeit-Animation kann im linken Bereich über das Menü «Karteneigenschaften» und dann «Optionen des Zeitschiebereglers» angepasst werden (Intervall, Geschwindigkeit etc.). Probiere auch hier unterschiedliche Optionen aus.

Erstelle nun noch zwei Bookmarks (Lesezeichen) für die Stadtkreise 7 und 10. Zoome hierfür in den Bereich des Stadtkreises 7. Gehe dann auf «Lesezeichen» > «Lesezeichen hinzufügen», gib «Stadtkreis 7» ein und bestätige die Eingabe mit der Enter-Taste. Gehe gleich vor für den Stadtkreis 10. Zoome nachfolgend wieder auf das gesamte Stadtgebiet.

Die Karte ist nun soweit vorbereitet, damit basierend auf dieser Web Map eine Web Mapping Applikation erstellt werden kann (folgende Übungen). Speichere die Karte via «Speichern und öffnen > Speichern».

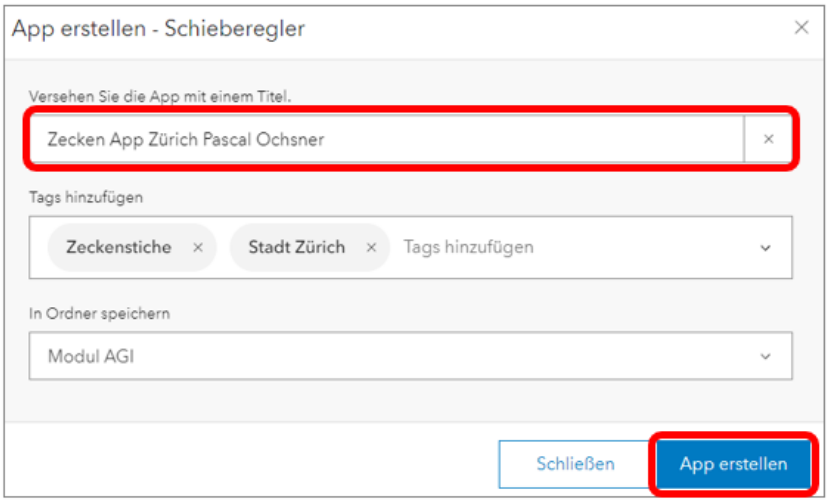
Erst nachdem die Layers einer Karte fertig symbolisiert sind, die Pop-ups konfiguriert und die Layer-Funktionalitäten etc. definiert sind kann eine Web Mapping Applikation erstellt werden. Die Web Mapping Applikation übernimmt die grundlegenden Einstellungen, welche in der Web Map definiert sind.

> **Tipp**
>
> In der Regel müssen sowohl die Web Map als auch die Web Mapping Applikation auf dieselbe Art und Weise freigegeben werden. Dies gilt natürlich auch für alle Web-Layers welche in einer Karte enthalten sind.

65.10. Übung 10: ArcGIS Online: Web App erstellen via "Instant Maps"

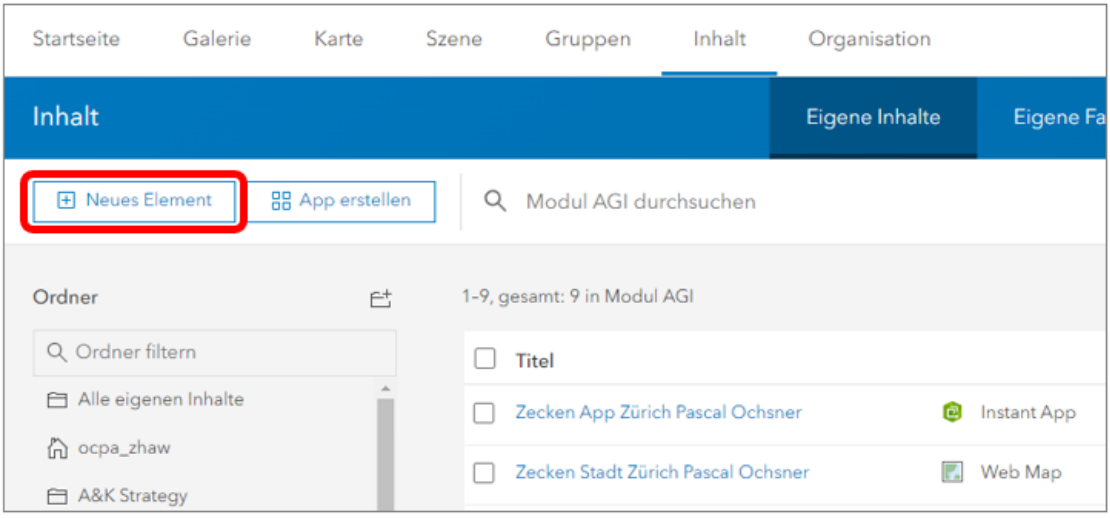
Erstelle nun eine Web Mapping Applikation basierend auf der vorher konfigurierten Web Map. In dieser Übung arbeiten wir mit «Instant Maps». Hierfür muss die Karte zuerst im «Map Viewer» geöffnet werden. Anschliessend generierst Du die Web Mapping Applikation über die Option «Instant Apps» (analog Übung 1 – Online Tutorial).

1. Wähle die Option «Schieberegler» und gehe auf «Vorschau». Gehe Zurück und erstelle die App mit «Auswählen».
2. Benenne die App als «Zecken App Zürich Vorname Name», lege sie im Ordner Modul AGI ab und wähle «App erstellen»



3. Gehe durch die 5 Schritte, aktiviere die Pop-Ups, die Lesezeichen und die Layer Liste. Veröffentliche anschliessend die Web App.
4. Ergänze die Web App nun noch mit einem zusätzlichen WMS-Layer. Gehe hierfür zuerst auf www.geolion.zh.ch und suche nach «Klimamodell». Gehe auf «Klimamodell ZH: Analysekarten (Raster)», öffne das Register «Datenbezug» und wähle den Link unter Datenbezug -> «WMS».

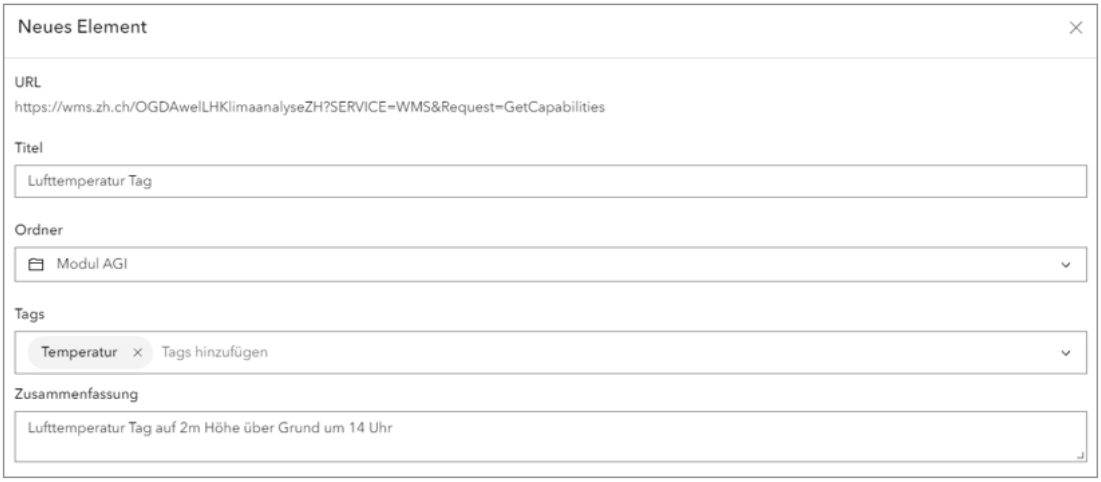
5. Wechsle in ArcGIS Online auf «Inhalt» und füge den WMS-Klimamodell via « + Neues Element» hinzu. Wähle die Option «URL».



Kopiere nun die WMS-URL in geolion und füge sie in ArcGIS Online im entsprechenden Feld ein. Füge nach http ein s ein (https://wms.zh.ch.....) Wähle die Option WMS (OGC) und klicke auf «Weiter». Wähle in der Liste den WMS-Layer «Lufttemperatur Tag».

Definiere folgende Parameter:

- Titel = Lufttemperatur Tag
- Ordner = Modul AGI
- Tags = Temperatur
- Zusammenfassung = Lufttemperatur Tag auf 2m Höhe über Grund um 14 Uhr

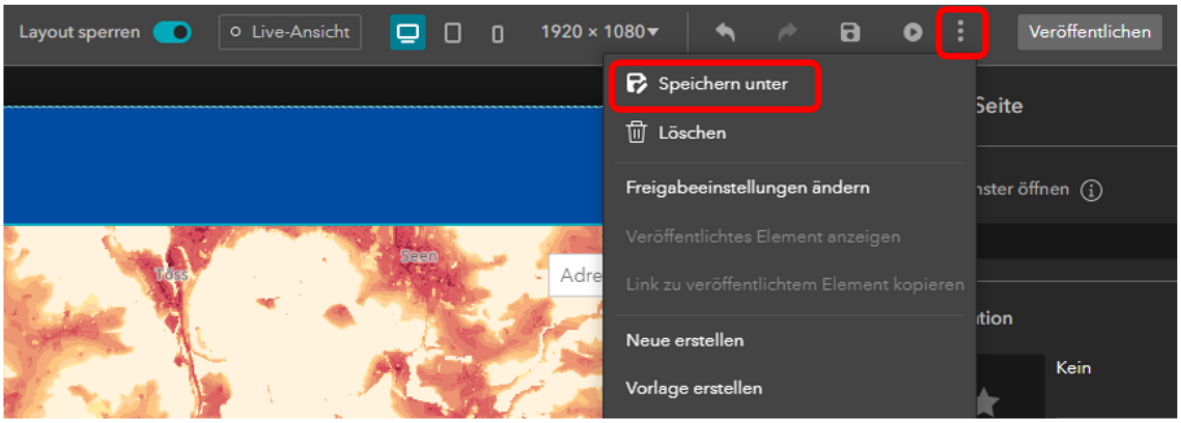


6. Öffne in ArcGIS Online die Web Map «Zecken Stadt Zürich Vorname Name». Gehe auf «Hinzufügen > Layer durchsuchen» und füge den Layer «Lufttemperatur Tag» zur Web Map hinzu. Verändere die Layerhierarchie und platziere die Temperaturkarte zuunterst.
7. Speichere die Änderungen.
8. Öffne nun die Web App. Der Layer «Lufttemperatur Tag» ist nun auch in der Web App enthalten und kann je nach Bedarf ein- und ausgeblendet werden (so wie alle anderen Layer auch).

65.11. Übung 11: ArcGIS Online: Web App erstellen via "Experience Builder"

Nun wirst Du eine zweite Web App erstellen, dieses Mal mit Unterstützung des Experience Builders. Der Experience Builder erlaubt eine viel flexiblere Gestaltung der Web App, insbesondere können verschiedenste Widgets in die Applikation integriert werden.

1. Öffne wiederum die Web Map «Zecken Stadt Zürich Vorname Name» und wähle im Menü links die Option «App erstellen > Experience Builder»
2. Wähle die Vorlage «Foldable». Du kannst eine Vorschau betrachten und dann via «Erstellen» in die Design-Oberfläche wechseln.
3. Nun kannst Du die Web App nach Belieben konfigurieren, mit Widgets ergänzen und Funktionalitäten einbinden.
4. Klicke nach Abschluss der Konfiguration auf «Speichern unter» und weise einen sinnvollen Namen zu. Veröffentliche anschliessend die Anwendung.



66. Übung Story Map

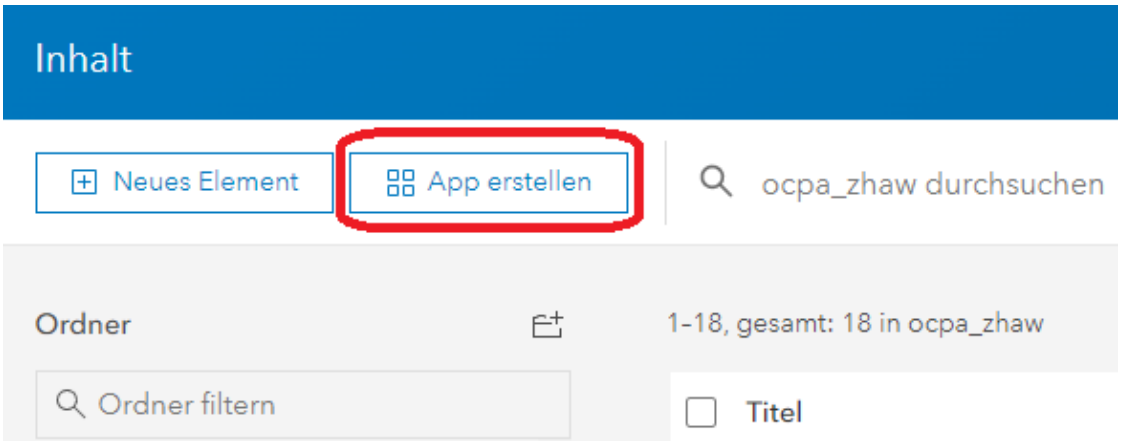
Story Maps vermitteln in interaktiver Form Informationen zu einem Ort, Ereignis, Problem, Trend oder Muster in einem geographischen Kontext. Textliche Inhalte können mit Fotos, Videos, Audiodateien und interaktiven Karten kombiniert werden.

Für diese Übung werden keine Daten zur Verfügung gestellt. Du kannst eine Story Map basierend auf einem in diesem Modul behandelten Themenbereich und deren vorhandenen Daten aufbauen – oder alternativ – die Grundlage für die Story Map (eine mögliche Dokumentationsart) Deines Semesterprojekts entwerfen. Die Story Map erstellst Du auf dem ZHAW ArcGIS Online Portal.

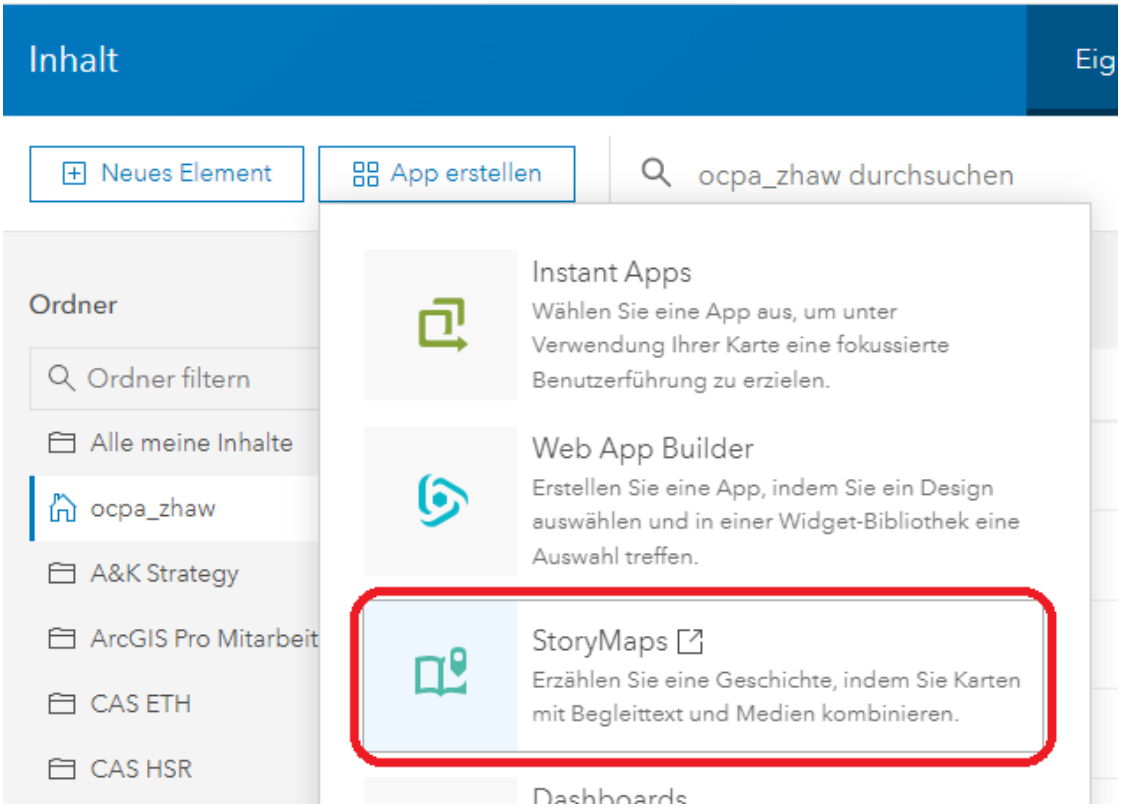
66.1. Übung 1: Story Map im ZHAW ArcGIS Online Portal erstellen

Der einfachste Weg eine Story Map zu erstellen, geht am schnellsten via ZHAW ArcGIS Online Oberfläche im Menü «Inhalt». Am besten erstellst Du vorgängig einen eigenen Ordner, in dem alle Inhalte der Story Map abgelegt sind.

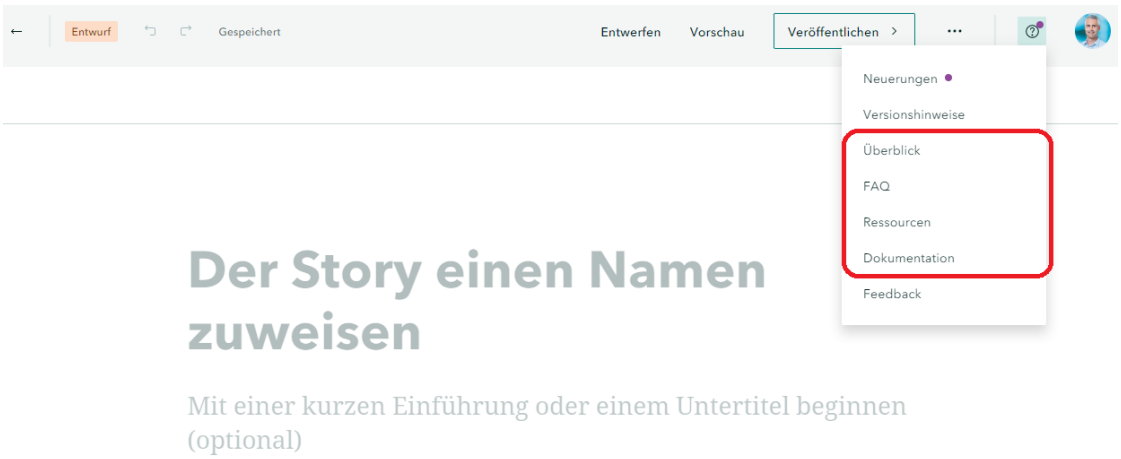
1. Melde Dich mit Deinen persönlichen Nutzerdaten auf dem ArcGIS Online Organisationskonto der ZHAW an: https://zhaw.maps.arcgis.com
2. Wechsle von der Startseite in das Register Inhalt. Erstelle bei Bedarf einen neuen Ordner für die Story Map (via Symbol «Neuen Ordner erstellen»).
3. Gehe auf «App Erstellen».



Wähle die Option «StoryMaps»



4. Über das Hilfemenü (oben rechts) hast Du Zugriff auf verschieden Links, unter denen Du Hilfe zur Erstellung einer Story Map erhältst.

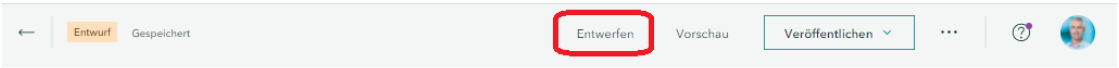


Öffne die Links jeweils in einem neuen Browser-Tab.

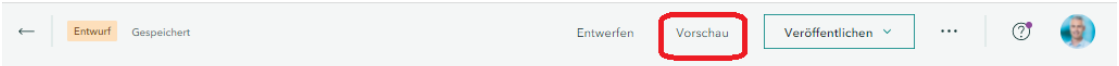
- Überblick: Allgemein Information zu Story Maps.
- Dokumentation: Lese was eine Story Map ist und informiere Dich, wie Du eine erste Story erstellen und veröffentlichen kannst.

- Ressourcen: Hier findest Du Tipps, was Du bei der Erstellung von Story Maps beachten musst damit diese aussagekräftig und fesselnd sind. Beachte hier die Story «Nine steps to great storytelling» (https://storymaps.arcgis.com/stories/429bc4eed5f145109e603c971
  Über folgendes Tutorial wird Dir vermittelt, wie Du deine Story erstellen kannst: Tutorial
- FAQ: Hier findest Du Antworten zu häufig gestellten Fragen.

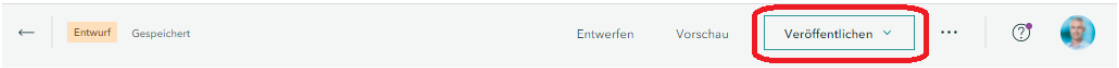
5. Erstelle nun den ersten Entwurf einer Story Map. Über die Option «Entwerfen» kannst Du das Erscheinungsbild Deiner Story anpassen.



Über die Option «Vorschau» kannst Du dir eine Vorschau der erarbeiteten Story anzeigen lassen (am besten in einem neuen Browser-Tab öffnen).



6. Die Option «Veröffentlichen» ermöglicht Dir, die Story zu veröffentlichen und die Freigabeeinstellungen anzupassen.



7. Erstelle nun Deine Story Map, fülle die Story mit Inhalten (Text, Bilder, Grafiken, Web Maps, Videos, Links etc.). Nutze selbständig die zur Verfügung stehenden Hilfe-Optionen (vorangehende Seite). Beachte: Bevor Du eine Web Map in eine Story Map integrieren kannst, muss diese vorgängig erstellt werden. Dabei müssen alle Inhalte auf die gleiche Art und Weise freigegeben werden.
8. Arbeite selbständig und sei kreativ. Probiere verschiedene Dinge aus. Beachte jeweils die Freigabeeigenschaften.

67. Leistungsnachweis

Es muss eine voll funktionsfähige Story Map erstellt werden, welche eine integrierte Web Map aus der vorherigen Übung enthält. Die Story Map soll in einer eigenen Gruppe veröffentlicht werden.

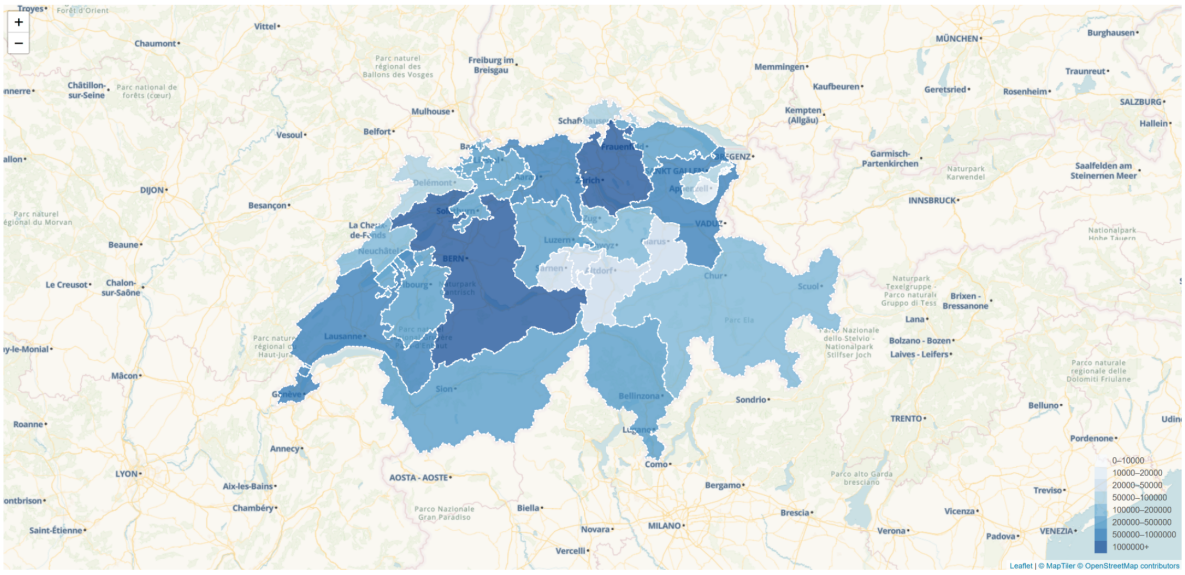
Gehe folgendermassen vor:

- Erstelle in ArcGIS Online eine neue eigene Gruppe.
- Benenne die Gruppe als «Leistungsnachweis Vorname Name».
- Definiere folgende Gruppeneigenschaften:
  - Wer kann diese Gruppe sehen? -> Nur Gruppenmitglieder
  - Wer kann Mitglied dieser Gruppe sein? -> Nur Mitglieder der eigenen Organisation
  - Wie können Personen dieser Gruppe beitreten? -> Nach Einladung
  - Wer kann Inhalte beitragen? -> Alle Gruppenmitglieder
- Füge Hanno Rahn (rahn_zhaw) und Dominic Lüönd (luoe_zhaw) deiner eigenen Gruppe «Leistungsnachweis Vorname Name» als Mitglied hinzu (via Benutzer einladen) ohne eine Bestätigung anzufordern.
- Veröffentliche deine Story Map für alle Mitglieder Deiner Gruppe «Leistungsnachweis Vorname Name».

Gib deine Lösung bis spätestens Freitag **17. Dezember 2023** auf Moodle ab. Lade hierzu den Link zu deiner Story Map hoch. Achte auf die korrekten Berechtigungen von Story Map und Web Map. Beide müssen diesselben Berechtigungen haben.

Teil XII.

WebGIS II



Übungsziele

- Du kannst eine Map mit mehreren Layern und einer Legende in Python erstellen.
- Du kannst dein Jupyter Notebook mittels Chunk Options und weiteren Tools verschönern.
- Du kennst Github und kannst ein Repository erstellen, Files hochladen und diese publizieren.
- Optional: Du kennst die grundlegenden Webtechnologien wie HyperText Markup Language (HTML), JavaScript und Cascading Style Sheets (CSS).
- Optional: Du hast dich mit Leaflet, der beliebtesten Open-Source-JavaScript-Bibliothek, vertraut gemacht.
- Optional: Du kannst das GeoJson-Dateiformat verwenden.

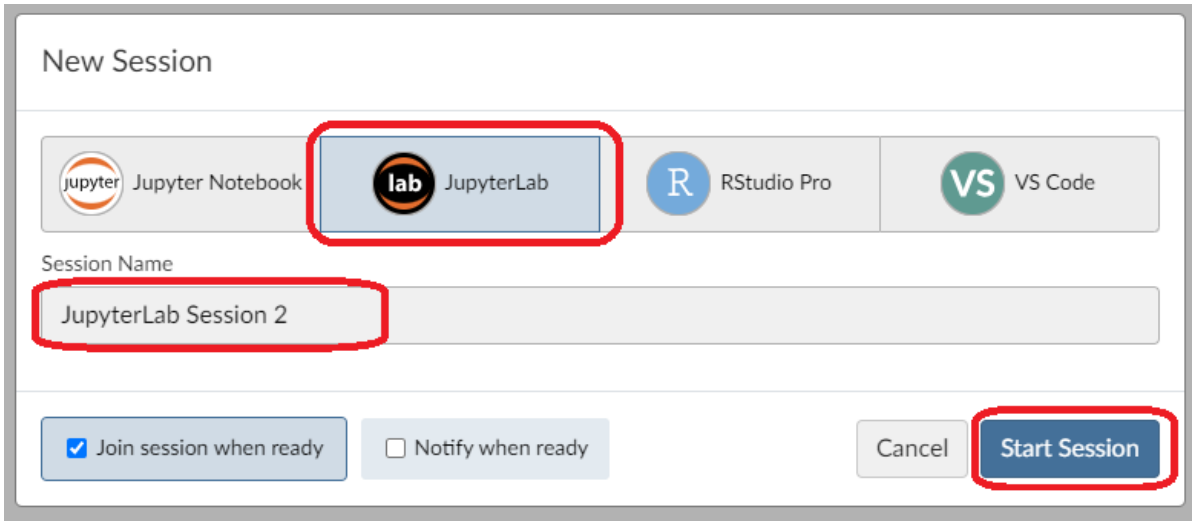
68. Übung Visualisierung Zeckenstiche

In dieser Übung wollen wir mit unseren originalen und simulierten Zeckenstichen aus dem Block Programmieren eine interaktive Karte erstellen. Zudem werdet ihr weitere Tools und Tipps erhalten, wie ihr euer Jupyter Notebook aufwerten und zum Schluss Online über GitHub publizieren könnt.

68.1. Übung 1: Vorbereitung

1. Bitte ladet zuerst die Übungsdaten hier runter und entpackt diese.
2. Um technische Probleme zu vermeiden, haben wir heute eine ZHAW-eigene Cloud Lösung vorbereitet. Es handelt sich dabei um eine *gehostete* Version von Jupyter Lab, auf welche ihr euch mit eurem ZHAW Benutzername und Passwort einloggen könnt: statistik-ide.zhaw.ch. Klickt auf `+ New Session` und wählt `JupyterLab` und gebt einen Session Namen ein.

Wer allerdings lieber lokal arbeiten möchte kann dies gerne tun, mit den Kentnissen aus Programmieren I - III sollte das gut möglich sein. Jedoch müssen diese noch das Programm Quarto installieren, sowie das fehlende Package Folium via conda installieren.



3. Wenn die Session geladen ist, könnt ihr eure Übungsdaten über den `Upload` Button hochladen. Öffnet anschliessend ein neues Jupyter Notebook und speichert es unter dem Namen `visualisierung_zeckenstiche.ipynb`.
4. Nun müssen wir noch die Packages installieren, die wir in dieser Session haben wollen. Statt `conda` verwenden wir jetzt allerdings `pip`, da die Serverlösung mit diesem Packagemanager arbeitet.

```
pip install geopandas folium matplotlib mapclassify
```

Startet nun das Kernel neu über Kernel -> Restart.

Importiert nun als erstes die zwei Packages `geopandas` und `folium`.

```
ModuleNotFoundError: No module named 'geopandas'
```

Ladet zudem die drei Datensätze `wald.gpkg`, `zeckenstiche_original.gpkg` und `zeckenstiche_simuliert.gpkg` mit der `read_file` Funktion von Geopandas in euer Jupyter Notebook.

```
NameError: name 'gpd' is not defined
```

68.2. Übung 2: Base Map

Nun können wir unsere Karte bauen. Zuerst müssen wir eine Basiskarte erstellen, zu welcher wir dann die weiteren Layer hinzufügen können. Dazu nutzen wir nun Folium. Als Mittelpunkt der Karte wählen wir die WGS84 Koordinaten der Stadt Zürich (Zurich latitude: 47.36667, longitude: 8.55) aus.

```
# Zurich latitude longitude: 47.36667, 8.55
map = folium.Map(location=[47.36667, 8.55], zoom_start = 13, tiles = "cartodbpositron")
```

```
NameError: name 'folium' is not defined
```

Ihr könnt mit den Tiles und dem Zoom Level experimentieren. Information zu den Tiles findet ihr hier.

68.3. Übung 3: Wald Layer hinzufügen

Nun wollen wir die Waldflächen als unseren ersten Layer hinzufügen. Ihr habt die .explore() Funktion bereits kennengelernt. Sie verfügt über diverse interessante Attribute, welche wir nun benutzen wollen.

Da unser Wald Datensatz aus zwei Zeilen mit Wald und nicht Wald besteht, müssen wir die Waldflächen zuerst noch rausfiltern.

Folium kann nur das weltweite CRS WGS84 (EPSG:4326) verarbeiten und da der Geodataframe Wald das Schweizer CRS CH1903+ / LV95 (EPSG:2056) verwendet, müssen wir dieses mittels `to_crs()` transformieren, bevor wir ihn visualisieren können.

Tipp: Wenn ihr am Schluss des Commands ein ; hinzufügt, wird kein Output generiert. Somit wird nicht für jeden Layer den wir hinzufügen, eine Karte erstellt.

```
# Wald zur Karte hinzufügen
is_wald = wald[wald["Wald_text"] == "ja"] # Selektieren nur den Wald
is_wald.to_crs(4326).explore(
    m = map,           # Fügt diesen Layer der Map map hinzu
    color = "green",   # Der Layer wird grün dargestellt
    tooltip = False,   # Es werden keine Daten angezeigt, wenn man über ein Objekt hovered.
    highlight = False, # Das Objekt wird nicht hervorgehoben, wenn man darüber hovered.
    name = "Wald"      # Der Name des Layers
);
```

68.4. Übung 4: Original und simulierter Zeckenstich Layer hinzufügen

Nun wollen wir neben dem Wald Layer (Polygon) den originalen Zeckenstiche Layer (Punkte) hinzufügen. Kopiere dazu den unteren Code.

```
# Original Zeckenstiche zur Karte hinzufügen
zeckenstiche_original_gpd.to_crs(4326).explore(
    m = map,                                    # Fügt diesen Layer der Map map hinzu
    color = "red",                              # Der Layer wird rot dargestellt
    marker_kwds = dict(radius = 1, fill = True), # Optionen für das Aussehen der Punkte
    tooltip = "ID",                             # Beim Hovern über das Objekt wird die ID da
    name = "Original Zeckenstiche"              # Der Name des Layers
);
```

Nun wollen wir die simulierten Zeckenstiche darstellen. Als Vorlage könnt ihr den Code von oben kopieren. Erweitere die Funktion `explore` um folgende Logik:

- Gerne würden wir die Spalte "Radius" mit der Farbvariation viridis visualisieren. Nutze hierfür die Attribute `column` und `cmap`.
- Zudem soll die Legende des Layers dargestellt werden. Zudem wollen wir keine Colorbar als Legende haben. Nutze dafür die Optionen der Legende `legend_kwds = dict(colorbar = False)`.
- In den Tooltips sollen neben der ID noch der Radius und der Run dargestellt werden.
- Nenne den Layer "Simulation Zeckenstiche".

Weitere Informationen zur `explore` Funktion findet ihr hier.

68.5. Übung 5: Layer Control hinzufügen

Zum Abschluss wollen wir noch ein `Layer Control` hinzufügen, damit wir die verschiedenen Layer ein- und ausblenden können. Schaut dafür im User Guide von Folium und sucht nach der Layer Control Funktion und versucht diese eurer `map` hinzuzufügen. Zum Schluss kannst du die `map` aufrufen, damit diese Dargestellt wird. Das finale Produkt sollte wie folgt aussehen:

```
NameError: name 'folium' is not defined
```

```
map
```

Nun ist unsere Map fertig. Inspiziert diese und spielt mit den `Radien` und `Runs` herum und versucht auch die Grössen und Farben der Points anzupassen, bis Ihr mit dem Resultat zufrieden seid.

68.6. Übung 6: Notebook verschönern

68.6.1. Input

Um unser Notebook zu verschönern werden wir das hilfreiche Tool Quarto verwenden. Dieses erlaubt es uns dynamischen Output zu generieren und ist eine Erweiterung zum klassischen Markdown. Mithilfe von Quarto kann euer Code in diverse Formate (PDF, HTML, ePub, und weitere) transformiert werden.

68.6.1.1. Vorbereiten von Quarto

Fügt ganz oben in eurem Notebook eine neue *raw* Cell ein und kopiert folgende Metadaten für Quarto. Dies bewirkt, dass euer Quarto Output ein Titel und Autor hat. Zudem sagen wir Quarto das wir mit Python3 arbeiten. Für unseren Ausgabeformat, welches HTML sein soll, definieren wir folgende global Optionen:

- `embed-resources: true`: Generiert ein einzelnes HTML File.
- `code-fold: true`: Alle Code Cells werden eingeklappt.

```
---
title: Visualisierung Zeckenstiche
author: Vorname Nachname
jupyter: python3
format:
    html:
        embed-resources: true
        code-fold: true
---
```

Quarto gibt uns zudem die Möglichkeit unser File als Vorschau darzustellen. Öffne hierfür eine neue Konsole, indem du auf das grosse `+` in Jupyter Lab klickst. Wähle unter Other das Terminal aus. Mit dem Command `quarto preview visualisierung_zeckenstiche.ipynb` erhaltet ihr ein Preview im Internet Explorer.

68.6.1.2. Cell Options

Cell Options können dazu benutzt werden, um das Verhalten einer Cell zu verändern. Dabei werden diese Optionen mittels des Vorschubs `#|` am Anfang der Cell eingefügt. Wenn ihr eine Option für alle Cells haben wollt, könnt ihr diese als globale Optionen in den Metadaten von Quarto setzen (analog zu `code-fold: true`).

```
#| echo: false
#| code-summary: This is a Title for a collapsed Code Block

Python Code...
```

Die häufigsten Options sind:

- `eval`: Soll die Code Cell ausgeführt werden.
- `echo`: Soll der Code der Cell angezeigt werden.
- `output`: Soll der Code Output dargestellt werden.

- `code-fold`: Soll die Code Cell eingeklappt werden.
- `code-summary`: Wenn `code-fold` auf `true` gesetzt ist kann ein Titel für die Code Cell gesetzt werden.

Es gibt aber noch viele weitere Möglichkeiten (siehe hier).

68.6.1.3. Page Layout

Mit Quarto könnt ihr euer Webseiten Layout relativ einfach anpassen. Ihr könnt zum Beispiel mit folgendem Code ein Grid erstellen, welcher aus zwei Spalten besteht. Eine Spalte ist 9 und die Zweite 3 breit. Die Gesamtbreite ist immer 12.

```
::: {.grid}

::: {.g-col-9}
Text 1
:::

::: {.g-col-3}
Text 2
:::

:::
```

Weitere Möglichkeiten findet ihr hier.

68.6.1.4. Tabellen

Eine Tabelle muss nach folgender Markdown Syntax erstellt werden (siehe Quarto Tables):

```
|Name |Gender|Age  |Origin |
|-----|:-----|:---:|:-----:|
|Jack |Male  |23   |USA    |
|Susan|Female|22   |Canada |

: Name der Tabelle {#tbl-names}
```

- Die erste Zeile ist die Tabellenüberschrift
- Die zweite Zeile beinhaltet Bindestriche "-" und optionalen Doppelpunkten ":", um den Text in den Spalten auszurichten.
  - `:---` für die linke Ausrichtung

  - `:---:` für die mittlere Ausrichtung
  - `---:` für die rechte Ausrichtung
- Die weiteren Zeilen beinhalten der Tabelleninhalt.
- Mit dem : unter der Tabelle kannst du den Namen der Tabelle definieren. Mit `{#tbl-???}` kannst du den Namen für einen Querverweise im Text erstellen. Dieser kann dann mittels `@tbl-???` im Text aufrufen.

Eine praktische Hilfe für das Erstellen einer Tabelle ist der Tablesgenerator.

68.6.1.5. Bilder

Ein Bild kann ebenfalls via Markdown Syntax hinzugefügt werden. Zuerst folgt die Caption des Bildes, gefolgt vom Bild selbst. Mit `{#fig-???}` kannst du einen Querverweis für das Bild erstellen. Gleich wie bei der Tabelle kannst du ihn im Text mittels `@fig-???` aufrufen.

```
![Caption](../images/test.png){#fig-???}
```

Weitere Information findet ihr auf Quarto.

68.6.2. Aufgabe

Seht euch die Webseite an. Versucht nun die Struktur dieser Webseite so gut wie möglich in eurem Jupyter Notebook nachzubauen. Nutzt dafür die oben beschriebenen Methoden. Das Bild findet ihr in den Übungsdaten.

Benutze zudem die Cell Options um euren Code sauber darzustellen. Nutzt dazu `echo` und `output` um unnötige Cell Codes und Outputs nicht auszugeben. Benenne zudem alle eingeklappten Code Cells sinnvoll mittels `code-summary`.

68.7. Übung 7: Notebook in HTML konvertieren

Damit wir nun unser Jupyter Notebook publizieren können, müssen wir davon einen HTML Output generieren.

Dazu nützen wir wieder Quarto in der Konsole. Aber anstelle des `preview` Commands nutzen wir den `render` Command, sprich `quarto render visualisierung_zeckenstiche.ipynb`. Da wir in den Quarto Metadaten als format HTML angegeben haben, wird Quarto ein HTML File generieren.

Ändere zum Schluss den Namen des HTML Files auf `index.html`, damit später GitHub Pages das File automatisch als Startseite erkennt. Mit Rechtsklick auf das File kann über `Download` das HTML File heruntergeladen werden.

68.8. Übung 8: GitHub und Publizieren

> **Hinweis**
>
> GitHub ist eine webbasierte Plattform, die Hosting für Git-Repositories bereitstellt. Es bietet Tools für die Zusammenarbeit, Codeüberprüfung und Projektverwaltung. GitHub wird von Entwicklern weit verbreitet genutzt, um Open-Source-Projekte zu teilen, dazu beizutragen und zusammenzuarbeiten. Es bietet auch Funktionen wie die Verfolgung von Problemen (Issues) und Pull-Anfragen (Pull Requests).

1. Registriert euch mit eurer privaten Email Adresse auf GitHub.
2. Nun da ihr eingeloggt seid, könnt ihr euer Repository erstellen. Drücke hierfür oben rechts auf euren Avatar und wähle dann "Your repositories". Klicke anschliessend auf das grüne Symbol "New". Im neuen Fenster kannst du nun einen Namen für dein Repository definieren. Zudem kannst du die Sichtbarkeit des Repository anpassen, aber wir belassen es bei Public, damit wir dann auch das HTML File publizieren können. Schliesse das Ganze mit "Create repository" ab.

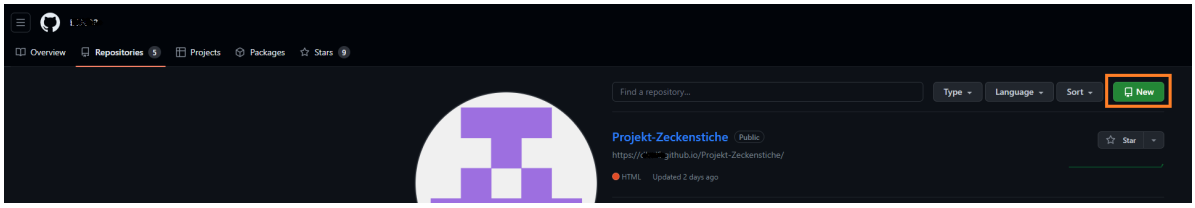


Abbildung 68.1.: Repository erstellen Schritt 1

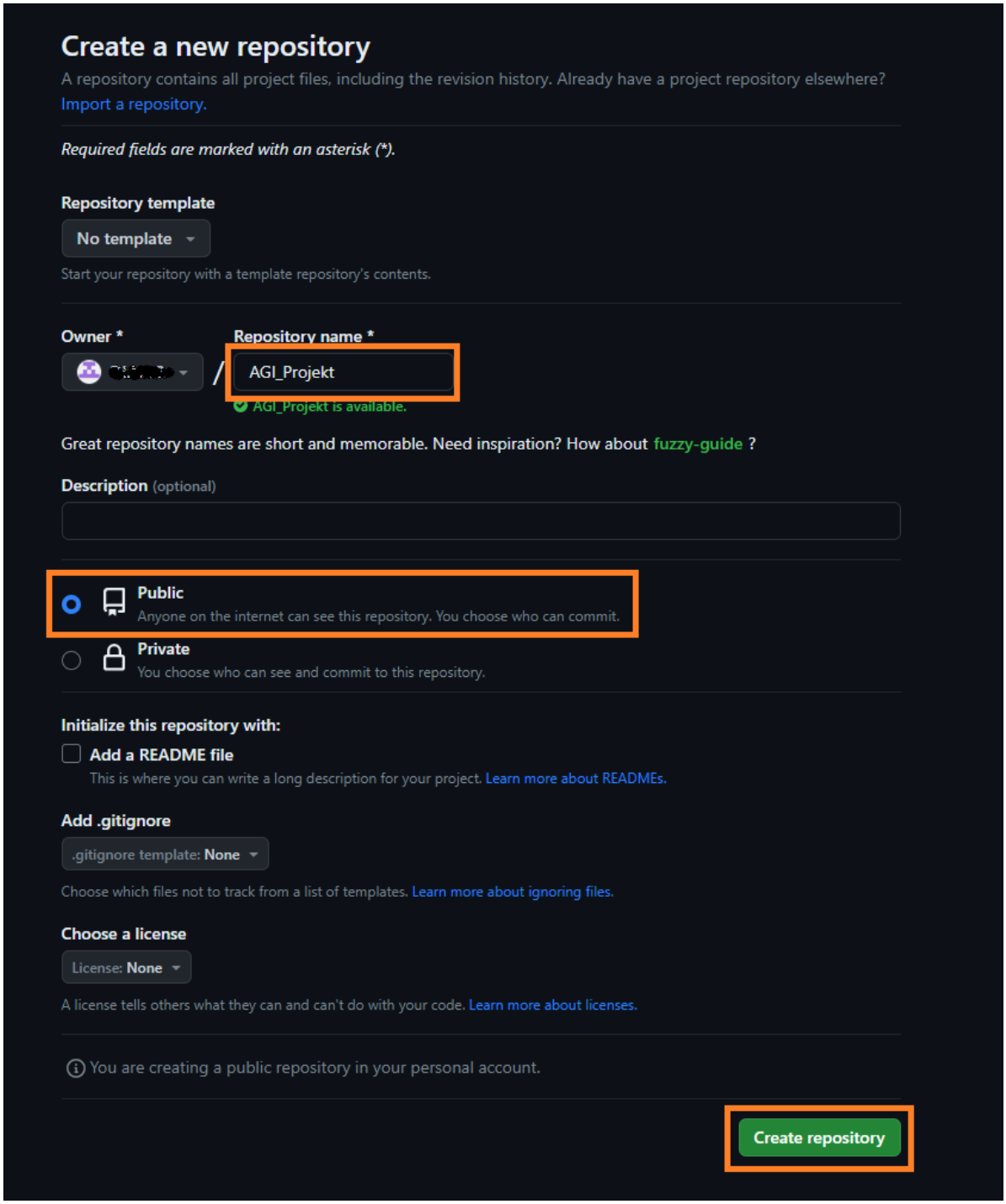
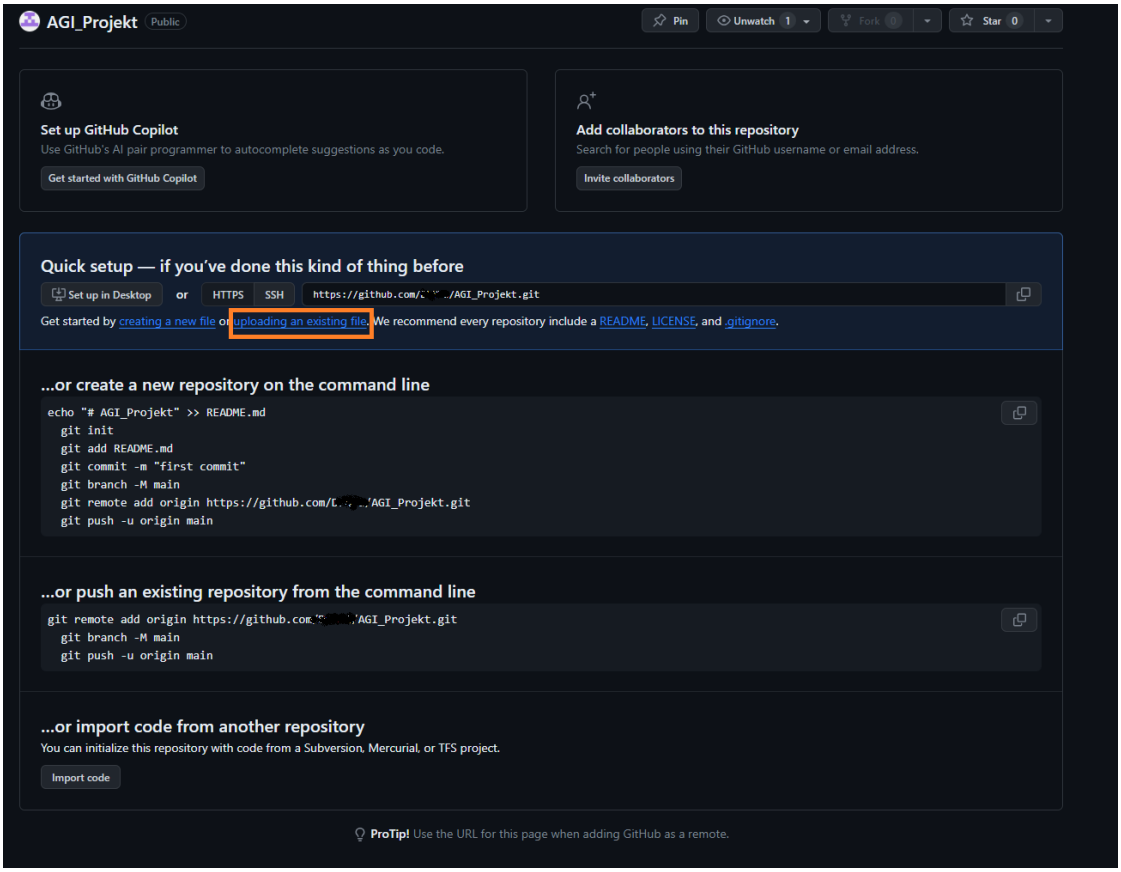
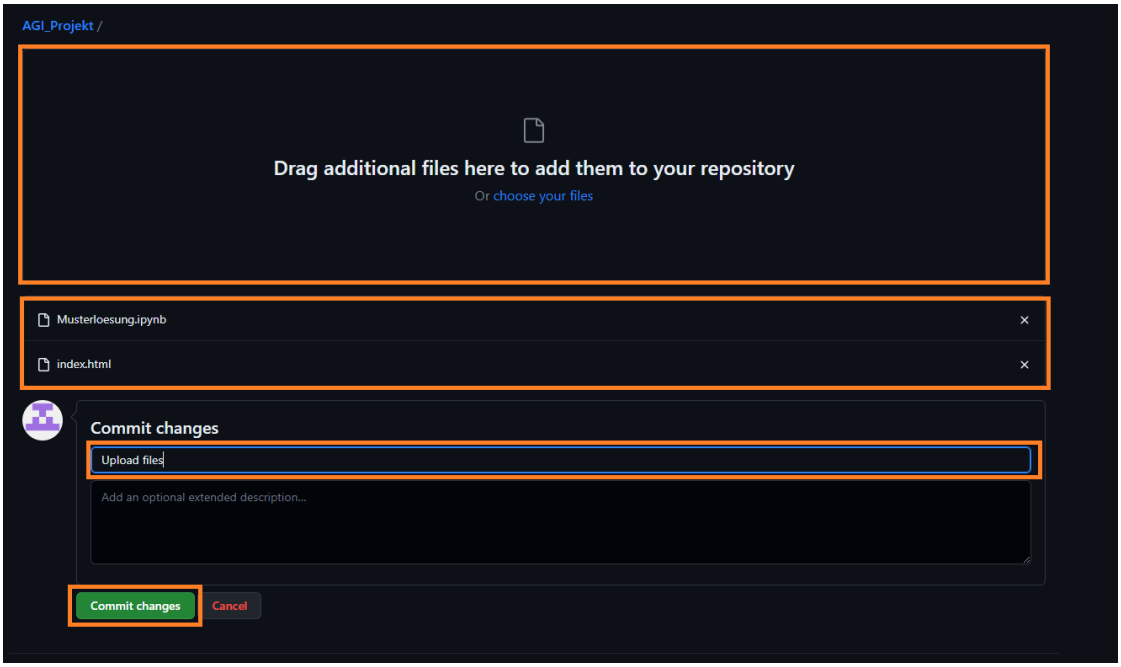


Abbildung 68.2.: Repository erstellen Schritt 2

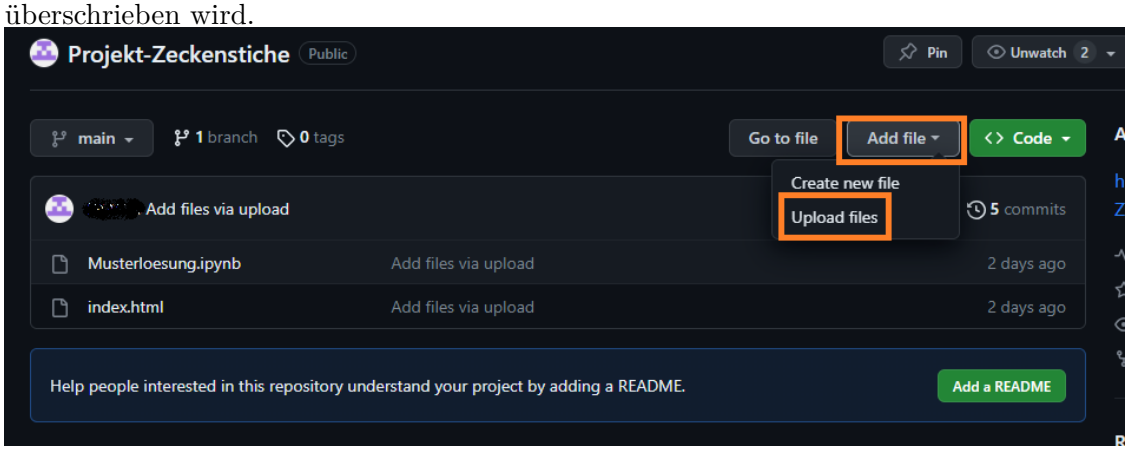
3. Nun hast du ein leeres Repository. Es gibt verschiedene Möglichkeiten nun Daten in dieses Repository zu kriegen. Am einfachsten ist aber das direkte Hochladen von Files. Klicke hierfür auf “uploading an existing file” und ziehe anschliessend das Jupyter Notebook und dein daraus generiertes HTML File (`index.html`) in den Explorer. Gib eine kurze Commit Beschreibung und klicke auf “Commit changes”.



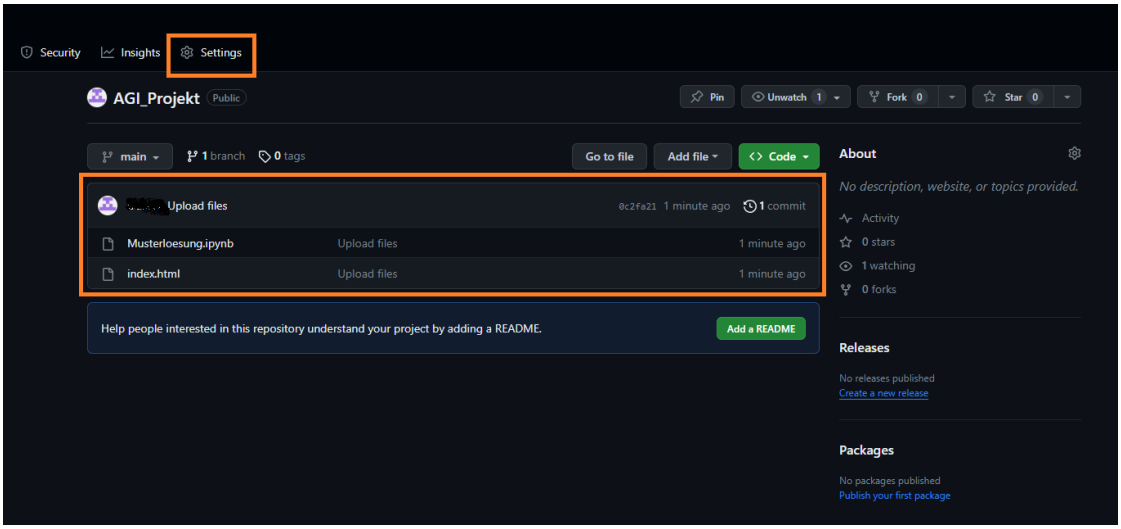


Update existierende Files

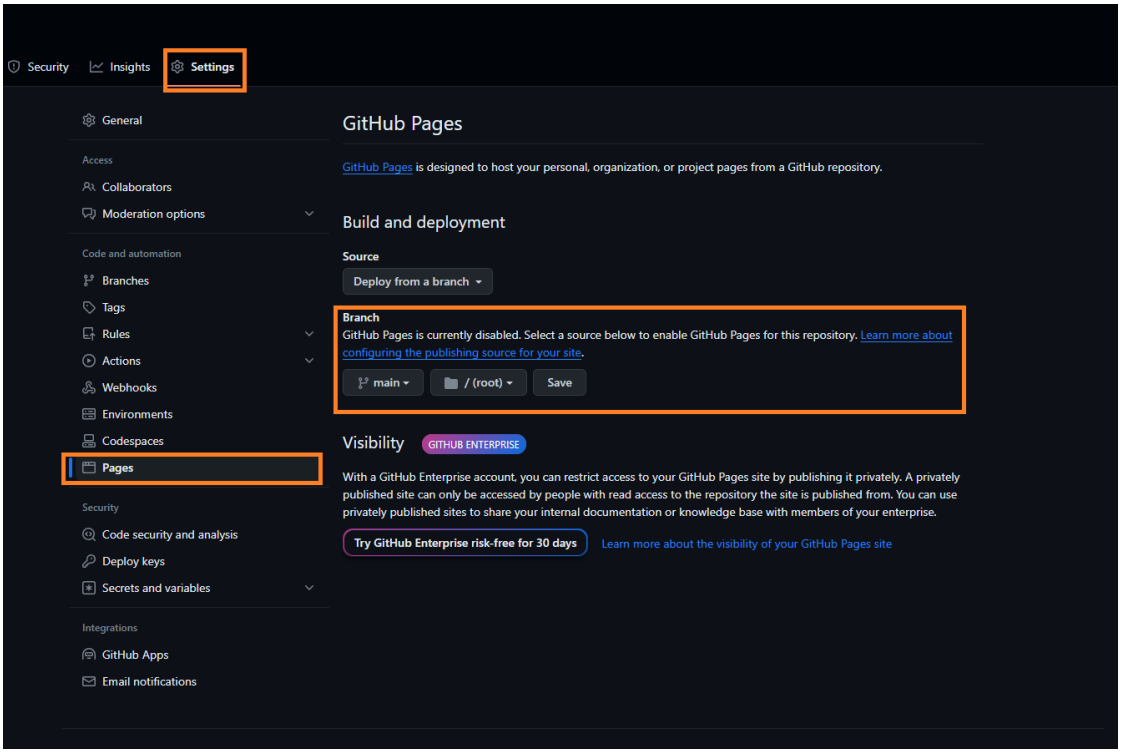
Wenn ihr nochmals Änderungen an den Files vornehmt, könnte ihr diese auf der Repository Seite über `Add File` vornehmen. Achte darauf das die Files den gleichen Namen haben, damit wird sichergestellt, dass das auf GitHub existierende File überschrieben wird.



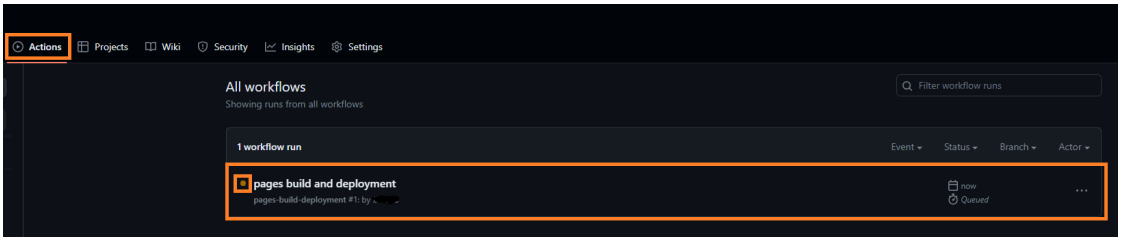
4. Ihr seht nun eure Files im Repository. Geht nun auf Einstellungen um eure Webseite zu Publizieren.



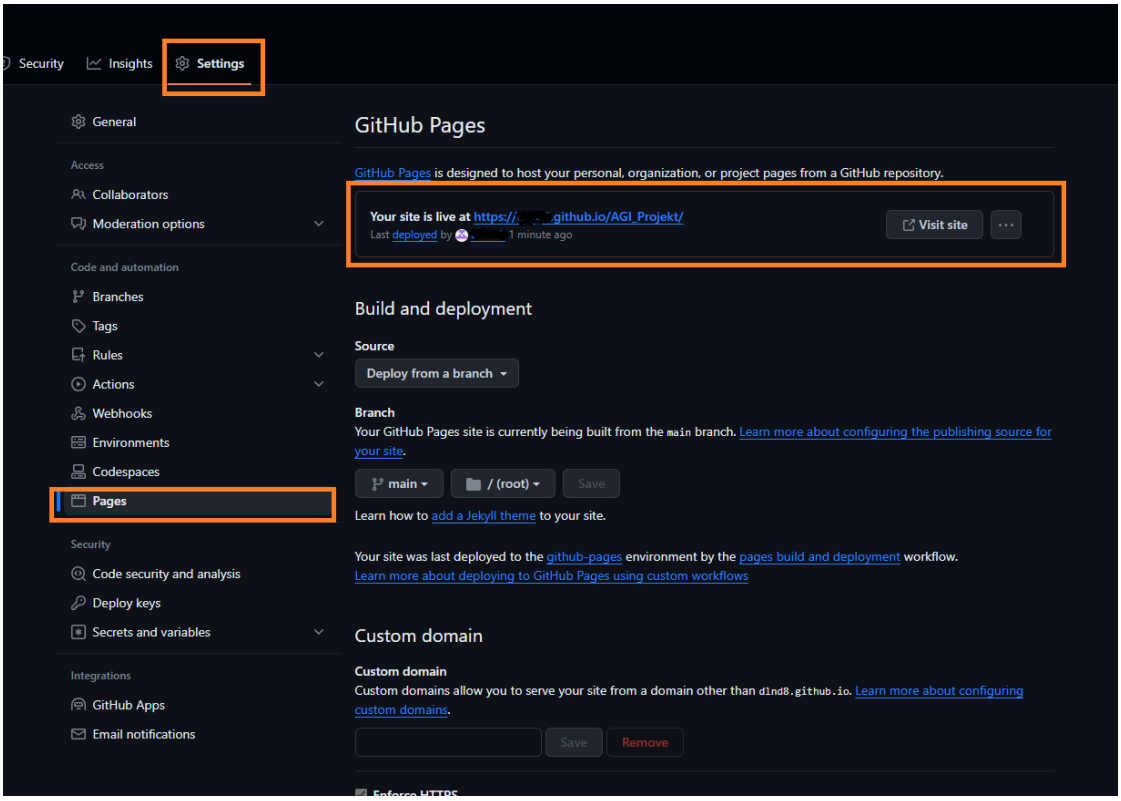
Geht auf Pages und setzt bei Branch die Einstellungen auf `main` und `/root` und klickt auf Save. Dies bewirkt, dass GitHub nun nach einem index.html in deinem main Branch sucht und aus diesem eine Webseite macht.



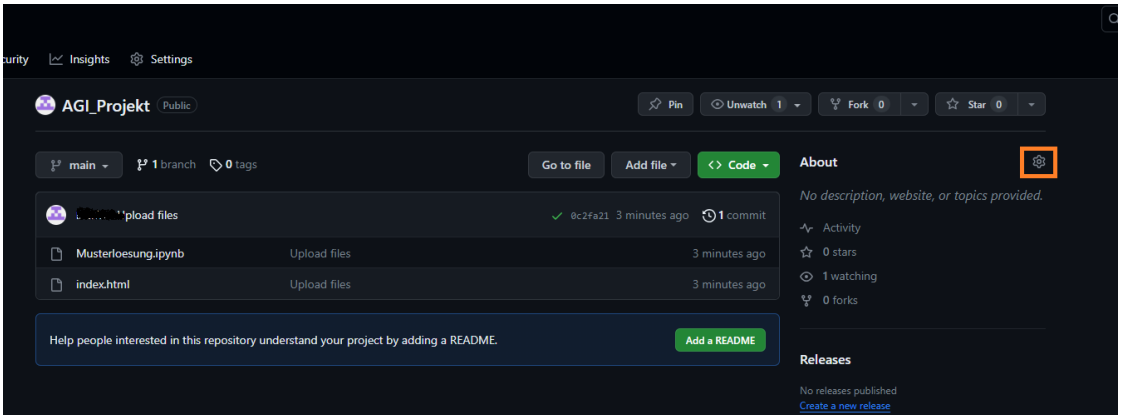
Diesen Vorgang könnt ihr unter Actions finden. Sobald der orange Punkt zu einem grünen Häkchen wird, ist die Webseite fertig erstellt.



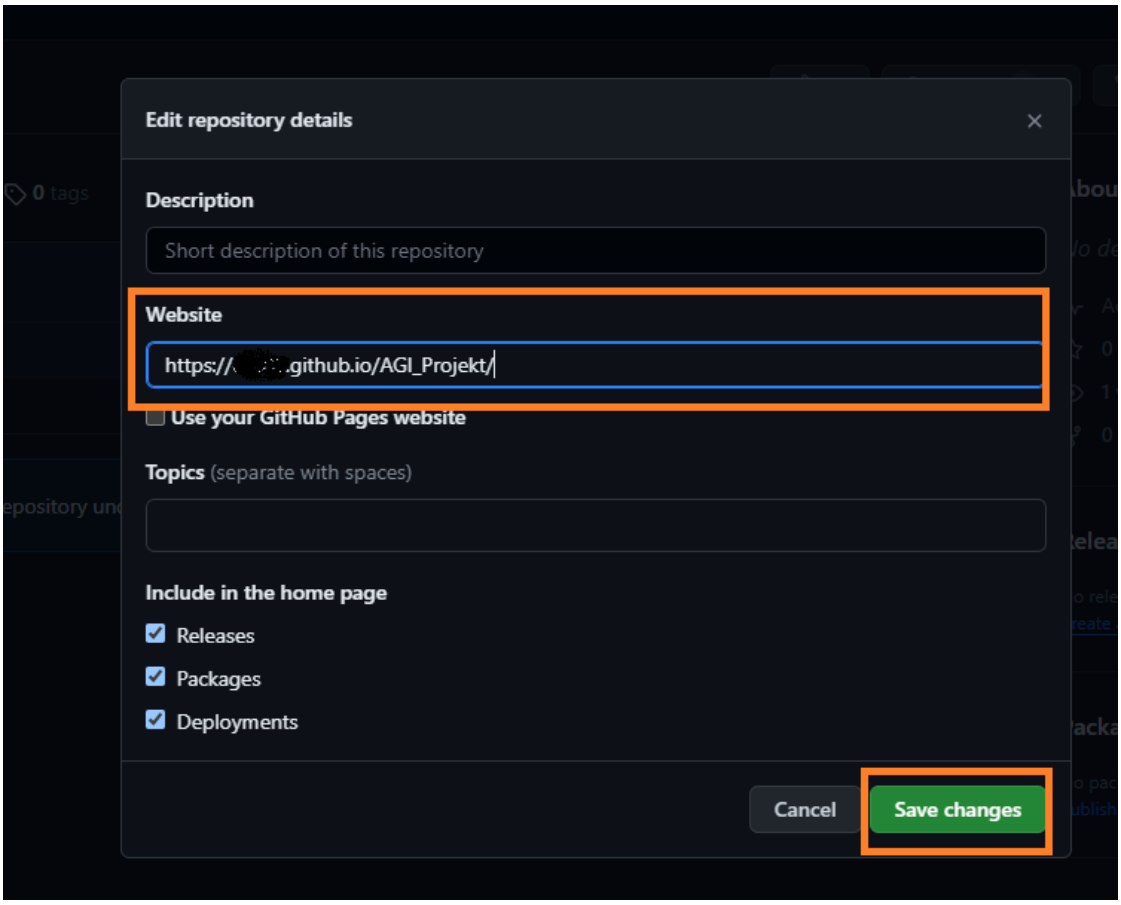
Kehr nun zurück zu Pages in den Settings. Ihr sollten nun eine URL sehen, auf welcher eure Webseite läuft.



5. Zum Abschluss könnt ihr diese URL im About des Repositorys hinzufügen. Dies macht es euch einfacher auf die Webseite zuzugreifen, da ihr nicht immer den Link in den Settings suchen müsst, sondern direkt in eurem Repository angezeigt wird.



Kopiere hierfür die URL und gehe auf den Code Tab und wählt dort das Rädchen neben dem About. Im Feld Webseite könnt ihr nun die URL reinkopieren. Speichert das mit "Save Changes". Nun sieht ihr unter dem About eure Webseite.



69. (Optional) Übung Leaflet

69.1. Übung 1: Erste Schritte mit HTML

HTML Tutorial

Die HyperText Markup Language ***(HTML)*** ist die Standard Markup Language für Inhalte, die in einem Webbrowser angezeigt werden sollen. Einfach ausgedrückt ist HTML die Sprache um den Inhalt einer Webseite zu definieren. Praktisch alle Webseiten basieren auf HTML.

Lass uns mit einem einfachen Editor eurer Wahl ein erstes einfaches HTML-Dokument erstellen.

Vorgeschlagene Editoren:

- Sublimetext
- Notepad++
- Atom

Öffnet den Editor Eurer Wahl und erstellt eine grundlegende HTML-Struktur, indem ihr das folgende Codefragment einfügt.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page title</title>
    </head>
    <body>
        <h1>My first heading</h1>
        <p>The first paragraph.</p>
    </body>
</html>
```

Speichert das Dokument als **index.html** in Eurem Arbeitsverzeichnis ab und öffnet es in einem von euch bevorzugten Browser. Ihr solltet etwas ähnliches wie die folgende Abbildung sehen (Abbildung 69.1).

Alle HTML-Dokumente müssen mit einer Dokumenttyp-Deklaration beginnen: *<!DOCTYPE html>*. Das HTML-Dokument selbst beginnt mit *<html>* und endet mit *</html>*. Der sichtbare

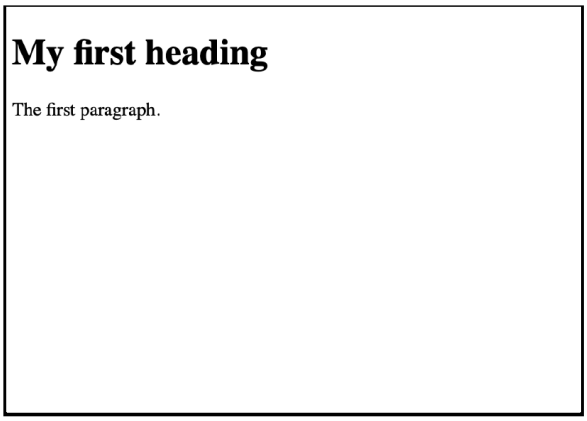


Abbildung 69.1.: Ihre Erste HTML-Seite

Teil des HTML-Dokuments liegt zwischen `<body>` und `</body>`. In diesem Abschnitt haben wir unsere erste Überschrift `<h1></h1>` positioniert, sowie das HTML-Element `<p></p>`, welches einen Absatz innerhalb des Body-Abschnitts definiert. Mehr zum Thema fHTML-Elemente findet Ihr unter folgendem Link HTML Elemente.

69.2. Übung 2: Erste Schritte mit JavaScript und der Leaflet-Bibliothek

JavaScript Tutorial

In der Welt der Webentwicklung ist HTML die Basis von allem, da es uns die Möglichkeit gibt, unsere Inhalte im Web darzustellen. Das, was die Dinge wirklich interessant und interaktiv macht, ist jedoch **JavaScript**. JavaScript ist im Moment die beliebteste Programmiersprache der Welt. Wenn HTML den Inhalt der Webseite definiert, definiert JavaScript dessen Verhalten. In HTML wird der JavaScript-Code zwischen den Tags `<script>` und `</script>` in einem beliebigen Teil unseres HTML-Codes (Head oder Body) eingefügt.

In dieser Übung verwenden wir zur Entwicklung einer einfachen Webkarte die Library **Leaflet**. Leaflet ist die beliebteste Open-Source-JavaScript-Library für die Entwicklung interaktiver Webkarten. Ihr Hauptvorteil besteht darin, dass sie effizient auf allen wichtigen Desktop- und Mobilplattformen läuft und auf einer einfach zu verwendenden und gut dokumentierten Anwendungsprogrammierschnittstelle (API) basiert.

- Geht zurück in Euer HTML-Dokument, das Ihr in Übung 1 erstellt habt und löscht die Elemente `<h1>` und `<p>`, sodass nur noch der leere `<body>` übrig ist.

Als nächsten Schritt werden wir ein `<div>`-Element für die Definition unserer Karte erstellen. Ein `<div>` HTML-Element ist ein allgemeiner Container, der keine Auswirkung auf den Inhalt

oder das Layout hat, bis er in irgendeiner Art und Weise mit Hilfe einer Styling-Sprache angesteuert wird. Hier kommt der dritte wichtige Akteur der Webentwicklung ins Spiel. Nämlich die Sprache Cascading Style Sheets kurz **CSS**. CSS ist die Sprache, die verwendet wird, um ein HTML-Dokument zu gestalten und um zu beschreiben, wie unsere Inhalte im Web dargestellt werden sollen. In HTML wird der CSS-Code zwischen den Tags *<style>* und *</style>* innerhalb der *<head>*-Tags eingefügt.

CSS Tutorial

- Erstellt ein div-Element, um die Karten-ID festzulegen, damit wir unsere Karte gestalten können.

```
<div id = "map"></div>
```

- Verwendet den # CSS-Selektor CSS-Selektoren innerhalb der *<style></style>*-Tags, um unsere Karte auf Vollbild zu setzen, indem Ihr die absolute Positionierung in die Ecken setzen.

```
<style>
    #map {position: absolute; top: 0; bottom: 0; left: 0; right: 0;}
</style>
```

- Nun werden wir die Leaflet-Bibliothek in unser File einbinden. Geht dazu auf Leaflet. Kopiert die folgenden zwei Codezeilen, welche die Leaflet CSS- und JavaScript-Dateien enthalten, in den Head-Bereich Eures HTML-Dokuments.

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css" integrity
<script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js" integrity="sha256-WBkoXOw
```

- Als nächstes erstellen wir eine neue Variable (innerhalb von *<script></script>*), welcher wir unsere bereits definierte Karte zuweisen. Im Folgenden definieren wir das Zentrum unserer Karte sowie die gewünschte Schärfeebene.

```
var map = L.map('map', {center: [46.944, 8.028], zoom: 8}); // Focus on Switzerland
```

- Um einen geigneten Tile Layer zu finden, könnt Ihr die Website Maptiler durchforsten. Wählt eine Map die Euch gefällt aus, scrollt nach unten zu den Tiles und kopiert den Link (siehe Abbildung 69.2).
  - Fügt den Link mit der Methode L.tileLayer() hinzu. Diese Methode von Leaflet erlaubt es uns den Tile Layer zu hosten (Leaflet-Erweiterungsmethoden). Aus Referenzierungsgründen ist es wichtig das Wir auch die Attribution hinzufügen (siehe Abbildung 69.3), welche am Ende der gleichen Webseite zu finden ist (Tipp: Ihr könnt diesen Schritt aus dem Code unten kopieren).
  - Füge schliesslich alles mit der Methode addTo(map) zu unserer Karte hinzu.

Abbildung 69.2.: Tile kopieren

Abbildung 69.3.: Kopieren der Attribution für die Nennung der Autoren

> **Tipp**
>
> Um die Daten zu erhalten, müsst Ihr ein kostenloses Konto erstellen. (Ihr könnt dazu eure ZHAW-E-Mail-Konto verwenden).

Nachdem Ihr die oben genannten Schritte durchgeführt habt, sollte Euer Code wie folgt aussehen:

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css" integ
        <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js" integrity="sha256-WBko
        <style>
            #map {position: absolute; top: 0; bottom: 0; left: 0; right: 0;}
        </style>
    </head>
    <body>
        <div id = "map"></div>
        <script>
            var map = L.map('map', {center: [46.944, 8.028],zoom: 8});
            L.tileLayer('https://api.maptiler.com/maps/streets-v2/256/{z}/{x}/{y}.png?key=4X
        </script>
    </body>
```

```
</html>
```

- Speichert das Dokument (**index.html**) und öffnet es in Eurem bevorzugten Browser. Das Ergebnis sollte ähnlich aussehen wie Abbildung 69.4.

Abbildung 69.4.: Interaktive Webkarte mit Leaflet JavaScript-Bibliothek

69.3. Übung 3: GeoJson-Datei zur Webkarte hinzufügen

GeoJson ist ein sehr verbreitetes Dateiformat zur Kodierung einer Vielzahl von geografischen Datenstrukturen. In der Webkartenentwicklung können GeoJson Variablen innerhalb einer JavaScript-Datei zugewiesen werden.

- Ladet von Moodle die Übungsdateien herunter und speichert alles im gleichen Ordner, in dem Euer Dokument ***index.html*** liegt. Einer der Hauptvorteile von JavaScript ist, dass es sich gut in ein HTML-Dokument integrieren lässt. Mit den Tags *'<script></script>* können wir beliebig viele JavaScript-Dateien mit dem Haupt-HTML-Dokument verknüpfen. Dies werden wir mit der Datei ***kantons.js*** tun.
- Verknüpft die Datei ***kantons.js*** im Head Eueres Dokuments (nach dem Leaflet Script) wie folgt:

```
<script src="kantons.js"></script>
```

- Öffnet nun die Datei ***kantons.js*** und kopiert den Namen der Variable, in der die Geojson-Daten gespeichert sind. Fügt diese anschliessend wie folgt zu Eurer Karte hinzu:

```
L.geoJSON(nameOfTheVariable).addTo(map);
```

Nach dem Speichern und Aktualisieren Eures index.html-Dokuments sollte Eure Karte wie Abbildung 69.5 aussehen.

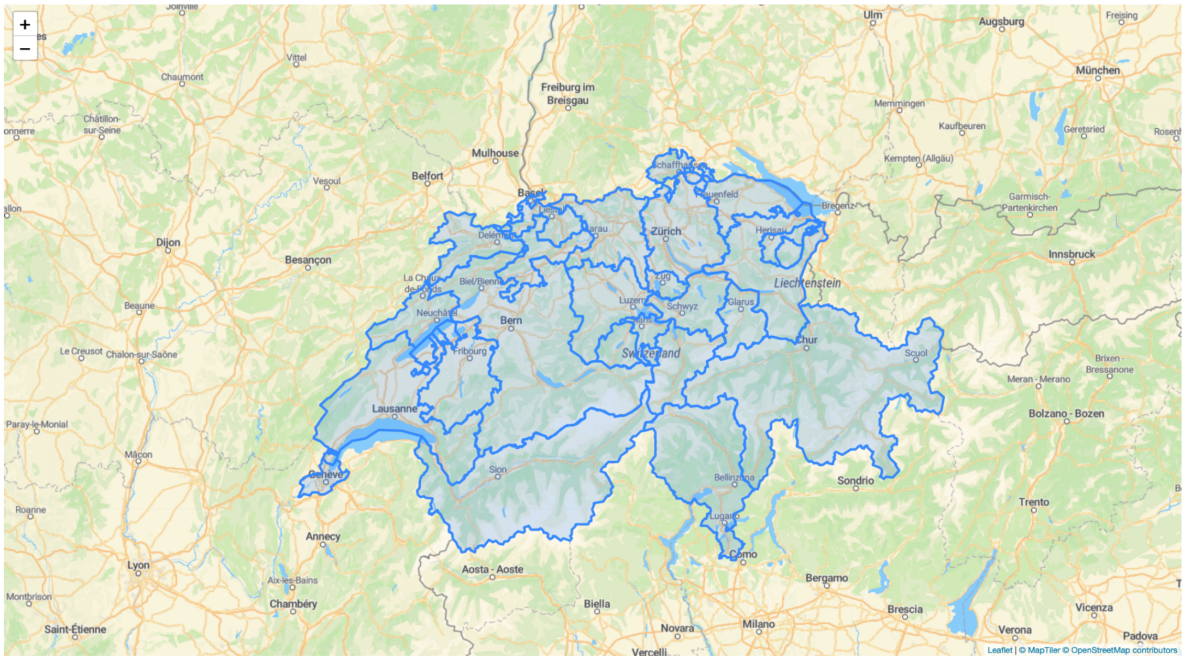


Abbildung 69.5.: Interaktive Webkarte mit den Kantonen der Schweiz

69.4. Übung 4: Farbe in die Webkarte einbringen

Das Ziel dieser Übung ist die Entwicklung einer interaktiven Choroplethenkarte. Jeder der räumlichen Einheiten (Kanton der Schweiz), wird eine Farbe zugewiesen. Jede Farbe basiert dabei auf einem spezifischen Geojson-Attribut, in unserem Fall die Bevölkerungszahl jedes Kantons.

- Öffnet erneut die Datei **kantons.js** und untersuchet die Geojson-Datei. Suchet nach dem Attribut welches die Bevölkerungszahl der Kantone representiert (siehe Abbildung 69.6).

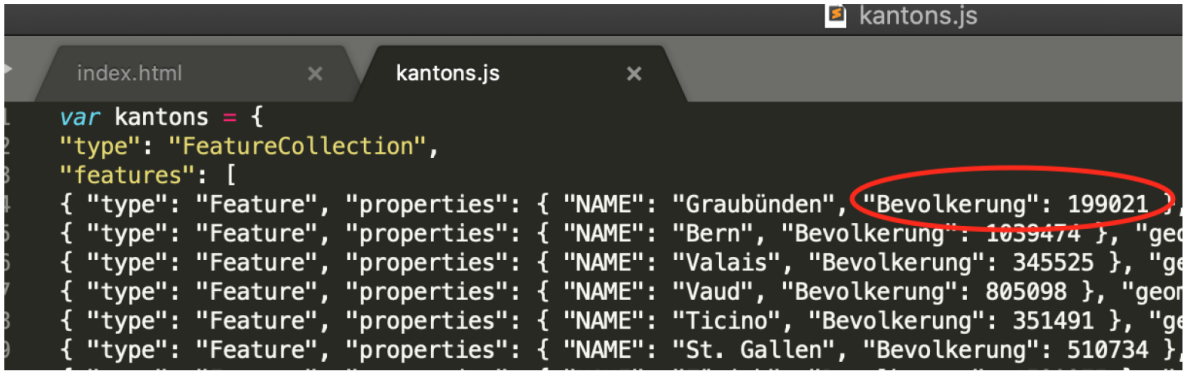


Abbildung 69.6.: JavaScript-Datei mit einer Sammlung von geografischen Datenstrukturen im Geojson-Format

- Als Nächstes werden wir die folgenden beiden Funktionen (***getColor(d)***, ***style(feature)***) verwenden, um jedem Kanton eine Farbe auf der Grundlage seiner Bevölkerungszahl zuzuweisen. Kopiert die folgenden Funktionen und fügt diese nach der ***map-Variablen*** in Euer Dokument ein.

```
function getColor(d) {
    return d > 1000000 ? '#084594' :
    d > 500000 ? '#2171b5' :
    d > 200000 ? '#4292c6' :
    d > 100000 ? '#6baed6' :
    d > 50000 ? '#9ecae1' :
    d > 20000 ? '#c6dbef' :
    d > 10000 ? '#deebf7' :
    '#f7fbff';
}

function style(feature) {
    return {
        // using the population property as argument
        fillColor: getColor(feature.properties.Bevolkerung),
        weight: 2,
        opacity: 1,
        color: 'white',
        dashArray: '3',
        fillOpacity: 0.7
    };
}
```

Die erste Funktion (***getColor(d)***) gibt das Argument **d** als Farbcode (z.B. `#9ecae1`)

zurück, nachdem dessen Wert überprüft wurde. Wenn d zum Beispiel grösser als 1'000'000 ist, erhalten wir die Farbe `#084594`.

Die Funktion (***getColor(d)***) wird dann innerhalb der zweiten Funktion (***style(feature)***) aufgerufen und nimmt als Argument das Attribut Bevolkerung der Variable kantons, die in der Datei ***kantons.js*** gespeichert ist. Schliesslich weist die Funktion ***style(feature)*** als Füllfarbe (fillColor) das Ergebnis der Funktion ***getColor(d)*** zu.

- Wir fügen nun als Style-Eigenschaft unseres **L.geoJson**-Objekts die Funktion ***style(feature)*** wie folgt hinzu:

```
L.geoJSON(kantons, {style: style}).addTo(map);
```

Nachdem Euer Dokument gespeichert und Euer Browser aktualisiert ist, sollte Eure Karte wie Abbildung 69.7 aussehen.

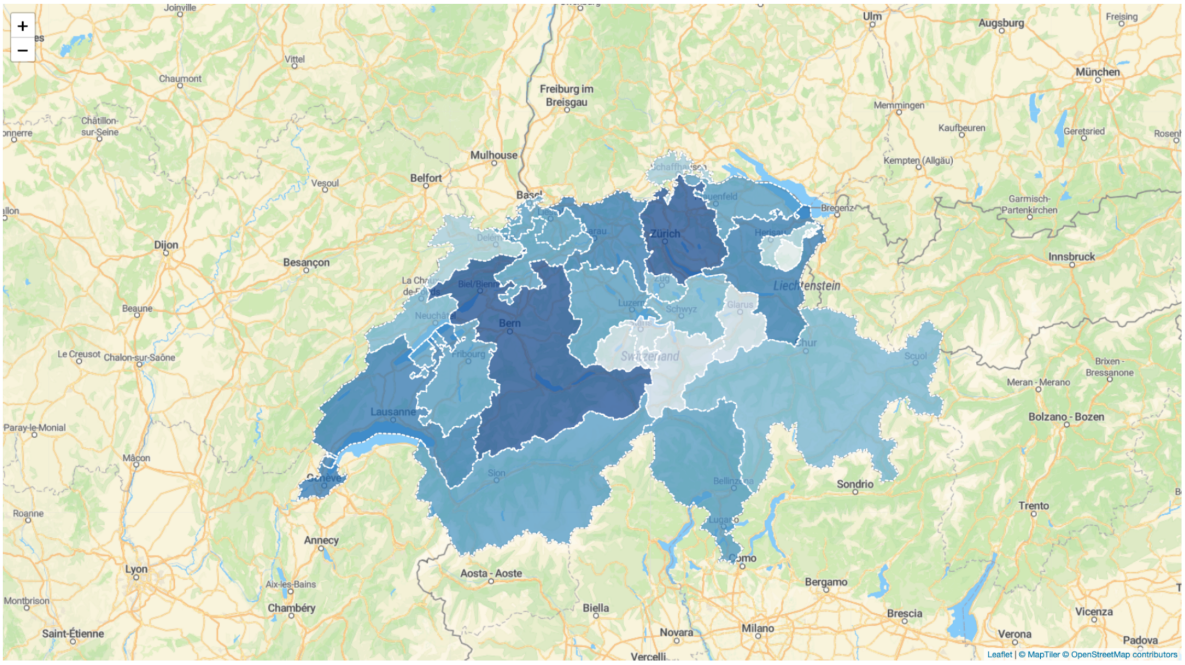


Abbildung 69.7.: Web Interaktive Choroplethenkarte mit den Kantonen der Schweiz

69.5. Übung 5: Hinzufügen von Interaktion zur Webkarte

Nachdem wir die Choroplethenkarte entwickelt haben, wäre es schön, wenn wir ihr etwas Interaktivität geben könnten. Dazu verwenden wir eine der wichtigsten Funktionen von JavaScript, den Event Listener. Kurz gesagt definieren Event Listener die Art und Weise,

wie die Endbenutzer mit dem Webinhalt interagieren können. Der wohl bekannteste ist der Onclick-Listener. Onclick-Listener definieren, was passiert, wenn der Benutzer irgendwo auf der Seite klickt. In unserem Fall ist es unser Ziel, dass wir auf jede räumlichen Einheiten (Kantone) in unserer Karte klicken können und wir im Gegenzug möglichst viele Informationen zurückerhalten. Genauer gesagt wollen wir, dass bei jedem Klick auf einen Kanton der Name und die Bevölkerungszahl auf dem Bildschirm angezeigt wird.

In einem ersten Schritt müssen wir sicherstellen, dass unsere GeoJSON-Ebene über eine Variable zugänglich ist, indem wir sie vor unseren Listenern definieren.

- Erstellt eine Variable mit dem Namen geojson, direkt nach der Variable map wie folgt:

```
var map = L.map("map", {center: [46.984, 8.950], zoom:7}); // focus on Switzerland
var geojson;
```

- Zuerst definieren wir, was bei einem Klick passieren soll. In userem Fall soll ein Popup mit dem Namen und der Bevölkerungszahl des Kontons erscheinen. Die Funktion sieht wie folgt aus:

```
function showInfo(e) {
    geojson.bindPopup("Kanton: " + " " + e.target.feature.properties.NAME +
    '</b><br />' +
    "Bevolkerung: " + e.target.feature.properties.Bevolkerung);
}
```

  In der obigen Funktion (***showInfo(e)***) verknüpfen wir mit unserer Geojson-Variable ein ***Popup***-Objekt (Leaflet popup), in das wir einen Text sowie die Attribute **NAME** und **Bevolkerung** jedes der in unserem ***e.target***-Objekt enthaltenen räumlichen Merkmale einbinden. In unserem Fall ist das ***e.target***-Objekt die Variable ***kantons*** (gespeichert in der Datei ***kantons.js***). (Mehr über e.target)

- Als nächsten Schritt verwenden wir die Funktion ***onEachFeature***, um die Funktion ***showInfo(e)*** mit dem onclick-Listener zu verknüpfen. Dies ist die Standardfunktion zur Verknüpfung der JavaScript-Listener mit den jeweiligen Funktionen, die für die Ausführung einer bestimmten Aufgabe erstellt wurden. Wir hätten zum Beispiel eine andere Funktion definieren können, die jedes Mal aufgerufen wird, wenn wir den Mauszeiger über ein bestimmtes Element unserer HTML-Struktur bewegen. In unserem Fall definieren wir den ***onclick listener*** wie folgt:

```
function onEachFeature(feature, layer) {
    layer.on({
        click: showInfo
    });
}
```

- Der letzte Schritt besteht darin, unsere GeoJSON-Ebene anzupassen, indem wir ihr die Option onEachFeature hinzufügen (L.geoJson documentation).

```
geojson = L.geoJson(kantons, {
    style: style,
    onEachFeature: onEachFeature
}).addTo(map);
```

Nachdem Ihr Eure HTML-Struktur aktualisiert habt, speichert Eure Datei ***index.html*** und aktualisiert den Browser. Versucht nun auf der Karte herumzuklicken und schaut was passiert. Ihr solltet etwas ähnliches sehen wie Abbildung 69.8.

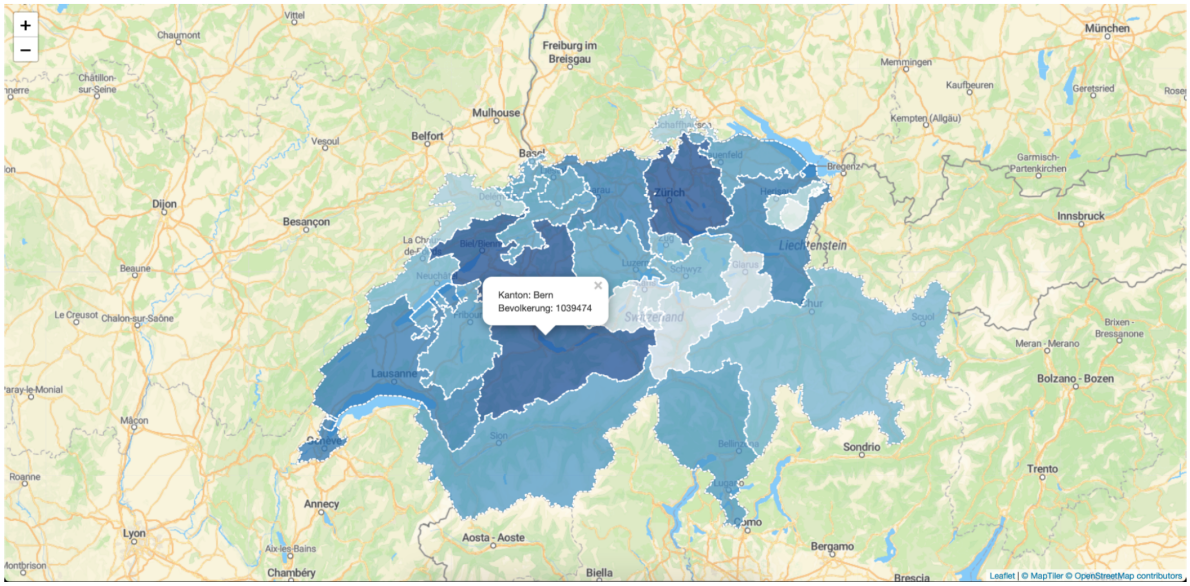


Abbildung 69.8.: Web Interaktive Choroplethenkarte mit den Kantonen der Schweiz (mit Interaktivität)

69.6. Übung 6: Der Webkarte eine Legende hinzufügen

Um eine Legende zu unserer Webkarte hinzuzufügen, werden wir JavaScript und CSS verwenden.

- Verwendet die beiden verbleibenden Dateien ***control.js*** und ***control.css*** (zu finden im Ordner data). Speichert diese beiden Dateien in Eurem Arbeitsverzeichnis (der Pfad, in dem sich auch index.html und kantons.js befinden)
- Verknüpft diese beiden Dateien mit Eurem Haupt-HTML-Dokument am Ende des Body-Abschnitts wie folgt.

```
<!-- Linking the js and css code for the legend -->
<script src="control.js"></script>
<link rel="stylesheet" type="text/css" href="control.css"/>
```

- Speichert die Datei ***index.html*** und aktualisieren die Browserseite. Die Webseite sollte nun wie Abbildung 69.9 aussehen.

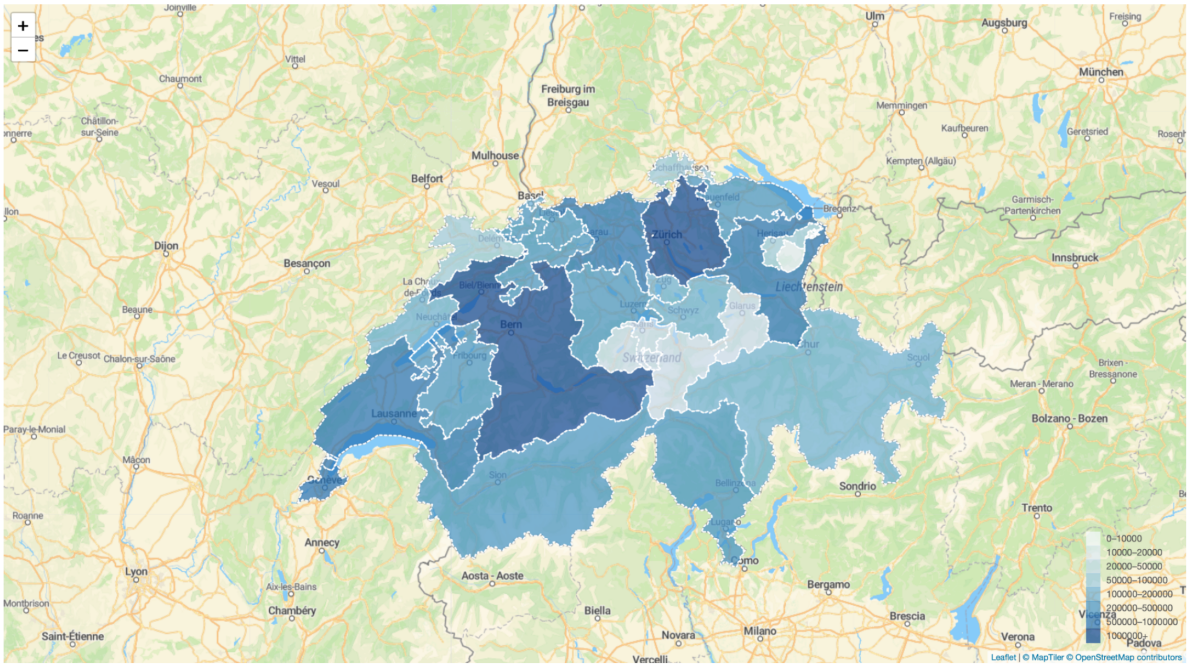


Abbildung 69.9.: Web Interaktive Choroplethenkarte mit den Kantonen der Schweiz (Endgültige Version)

Öffnet die beiden Dateien, die Ihr zuletzt zur Karte hinzugefügt habt, und versucht, den darin enthaltenen Code zu interpretieren. Versucht, etwas zu ändern und seht anschliessend, wie sich diese Aktion auf Eure Karte auswirkt.

Glückwunsch!! Ihr habt soeben Eure erste voll funktionsfähige interaktive Webkarte erstellt und dabei nur die grundlegenden Webtechnologien verwendet. Man könnte dieses HTML nun zum Beispiel über Netlify Drop Online zur Verfügung stellen.

69.7. Endgültige Lösung

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css" integ
        <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js" integrity="sha256-WBko
        <script src="kantons.js"></script>
        <style>
            #map {position: absolute; top: 0; bottom: 0; left: 0; right: 0;}
        </style>
    </head>
    <body>
        <div id = "map"></div>
        <script>
            var map = L.map('map', {center: [46.944, 8.028],zoom: 8});
            var geojson;
            // function for coloring the cantons based on their population property

            function getColor(d) {
                return d > 1000000 ? '#084594' :
                d > 500000 ? '#2171b5' :
                d > 200000 ? '#4292c6' :
                d > 100000 ? '#6baed6' :
                d > 50000 ? '#9ecae1' :
                d > 20000 ? '#c6dbef' :
                d > 10000 ? '#deebf7' :
                '#f7fbff';
            }

            function style(feature) {
                return {
                    fillColor: getColor(feature.properties.Bevolkerung), // using the popula
                    weight: 2,
                    opacity: 1,
                    color: 'white',
                    dashArray: '3',
                    fillOpacity: 0.7
                };
            }

            function showInfo(e) {
                geojson.bindPopup("Kanton: " + " " + e.target.feature.properties.NAME + '</b
            }
```

```
            function onEachFeature(feature, layer) {
                layer.on({
                    click: showInfo
                });
            }

            L.tileLayer('https://api.maptiler.com/maps/streets-v2/256/{z}/{x}/{y}.png?key=4X8

            geojson = L.geoJson(kantons, {
                style: style,
                onEachFeature: onEachFeature
            }).addTo(map);
        </script>

        <!-- Linking the js and css code for the legend -->
        <script src="control.js"></script>
        <link rel="stylesheet" type="text/css" href="control.css"/>
    </body>
</html>
```