CAS FHNW – Spatial Data Analytics

Infrastruktur der Datenanalyse

Dr. Urs-Jakob Rüetschi

Wir werden schwierige Gewässer befahren müssen:
Detail aus einem Globus von Mercator von ungefähr 1550,
digitalisiert von der Universität Lausanne.

Inhalt

- Grundlagen, Standards, Praktiken
- Isolation und Automation
- Von der Exploration zur Produktion
- Einbettung in bestehende Systeme

Urs-Jakob Rüetschi, Dr. sc. nat. UZH
Esri Schweiz AG
Dira GeoSystems AG

Infrastruktur der Datenanalyse

Skript zur Vorlesung im Rahmen des CAS Spatial Data Analytics FHNW

von Dr. Urs-Jakob Rüetschi 2022

Das vorliegende Skript begleitet den gleichnamigen Themenblock im CAS Spatial Data Analytics an der FHNW. Es gibt den vielfältigen Informationen eine Struktur, enthält wichtigen Begriffe und Konzepte und hilft Ihnen damit bei der Erstellung Ihrer eigenen Notizen, kann und will diese aber nicht ersetzen!

Inhaltsverzeichnis

1 Einleitung

Wenn Sie mit Daten arbeiten, arbeiten Sie immer auch mit Systemen und Werkzeugen, welche diese Daten speichern, lesen, bearbeiten, transformieren, darstellen und speichern, sowie mit der Umgebung, in der diese Systeme betrieben werden. Wir bezeichnen dies hier kollektiv als die Infrastruktur der Analyse. Bei der Präsentation der Ergebnisse einer Analyse oder sonstigen Arbeit ist die Infrastruktur kaum je ein Thema, allenfalls wird sie unter «Methoden» kurz erwähnt. Trotzdem ist sie unerlässlich, weil die Arbeit ohne diese darunter liegenden Systeme nicht möglich gewesen wäre, oder aber sehr viel aufwändiger. Das Thema von heute betrachtet diese «darunter liegenden Systeme» (Infrastruktur) und wie sie zu unserer Tätigkeit der (räumlichen) Datenanalyse stehen.

Mit «Spatial Data Analytics» bewegen wir uns im grossen aktuellen Trend der Data Science. Die Arbeit als «Data Scientist» wurde gar reisserisch als «the sexiest job of the 21st century» bezeichnet (Harvard Business Review, Oktober 2012; vergleiche dazu auch den Artikel über den Artikel vom Juli 2022, im Literaturverzeichnis aufgeführt). Die Tätigkeit mag einige Attraktivität haben, ein grosser Teil ist aber harte Arbeit, und dies nicht nur mit spannenden Daten, sondern auch mit anspruchsvoller Infrastruktur, mit Systemen, die unverhofft Probleme

verursachen, mit Komponenten, die notwendig sind aber nicht zueinander passen. Gefragt sind also auch Eigenschaften wie Durchhaltevermögen und Frustrationstoleranz!

Exploration und Produktion. Die Arbeit des Daten-Analysten (oder des Data Scientist) ist inhärent explorativ: die Daten liegen in irgendeiner Form vor, und daraus sollen durch Analysen neue Erkenntnisse gewonnen werden. Hier sind Fantasie, Erfahrung, Sorgfalt, und Experimente gefragt. Ausgang einer erfolgreichen Datenanalyse kann eine Kennzahl, eine Grafik, eine Karte, oder auch ein (Geschäfts-)Prozess, eine Optimierung, eine Anwendung, eine Präsentation sein, womit ein Übergang in die Produktion mit ihren erhöhten Anforderungen an Nutzbarkeit (*usability*), Verlässlichkeit (*reliability*), Reproduzierbarkeit (*replicability*) und Rechenschaft (*accountability*) einhergeht. Selbst wenn hier die Verantwortung des Datenanalysten irgendwann aufhört, zumindest am Übergang von Exploration zu Produktion wird er oder sie dabei sein.

Daten und Prozesse. Computing bedeutet Arbeit mit Daten und Prozessen. Prozesse sind der aktive Teil, der mit den Daten operiert. Sie brauchen Rechenleistung (Operationen pro Sekunde), während Daten Speicherleistung brauchen (Bytes). Das sind zwei Arten der Infrastruktur, die entweder lokal oder in der Cloud bereitgestellt werden müssen. Bei Cloud-Anbietern werden diese Dienste unter Begriffen wie «Storage» und «Compute» kategorisiert.

Abgrenzung. Wir werden hier einen sehr breiten Begriff von Infrastruktur pflegen; insbesondere werden wir in diesem Rahmen auch Daten und Dateiformate behandeln, was in der IT eher nicht mit Infrastruktur assoziiert wird. Dafür wird der ganze Bereich Big Data ausgeklammert – dem ist ein eigener Themenblock später gewidmet – und ebenso der Bereich der Hardware. Weiter konzentriert sich das Skript stark auf Python und geringfügiger auf Esri ArcGIS, andere Systeme mit Lösungen für die Datenanalyse (wie etwa QGIS oder R) werden weitgehend ausgeklammert. Dies bedeutet keine Wertung, sondern eine Fokussierung in einem begrenzten Zeitrahmen.

Übersicht. Das Skript ist wie folgt aufgebaut. Wir starten mit Überlegungen zu Infrastruktur im Allgemeinen und einer Ergänzung zum Begriff der «Spatial Data Infrastructure». Es folgt ein Abschnitt zu Computational Notebooks als einem in der Datenanalyse beliebten Paradigma für die Interaktion mit dem Computer. Wir wenden uns dann Daten, Datenbanken und dem Processing zu. Es folgen Ausführungen zu den wichtigsten Standards für die räumliche Datenanalyse. Die nächsten Abschnitte befassen sich mit der Isolation von Systemen, mit der Automatisierung von Aufgaben im Bereich der Infrastruktur, und mit dem Einbezug von Cloud-Diensten. Das Skript schliesst mit bewährten Prinzipien für eine robuste und replizierbare Infrastruktur.

2 Infrastruktur

Bei «Infrastruktur» denken wir zunächst an physische Einrichtungen wie etwa Autobahnen oder Stromleitungen, weiter an institutionelle und organisatorische Einrichtungen wie etwa ein Rechtssystem oder ein Gesundheitswesen. Ihnen allen ist gemeinsam, dass sie dem Zusammenleben einer Bevölkerung dienen. Der Begriff leitet sich ab aus lateinisch *infra* = unter und *structura* = Zusammenfügung. Infrastruktur impliziert also eine konzeptionelle Schichtung und bezeichnet darin tiefere Schichten.

In der Informatik versteht man unter Infrastruktur alle materiellen und immateriellen Güter, die den Betrieb von Anwendungssoftware ermöglichen (Wikipedia «IT-Infrastruktur», August 2022). Sowohl Hardware wie auch Software wird also zur Infrastruktur gezählt. Im Bereich der Hardware sind dies etwa Server, Speicheranlagen, Netzwerkkomponenten, aber auch Kühlanlagen, Gebäude, Stromversorgung. Die Grenze zur Anwendungssoftware ist dabei nicht eindeutig bestimmbar. Wir wollen hier Software so weit zur Infrastruktur zählen, als sie nicht unmittelbar mit der konkreten Datenanalyse in Verbindung steht. Wir verstehen also Infrastruktur als das technische Umfeld der Datenanalyse.

Wie schon der Begriff «Infrastruktur» eine Schichtung impliziert, sind **Schichten** (*layers*, *tiers*) eine beliebte Abstraktion in der Informatik. Aus grosser Distanz betrachtet stellt sich ein System zur Datenanalyse wie in **Abbildung 1** links dar. Hardware und Betriebssystem sind die Infrastruktur, welche die Bearbeitung von Daten und die Ausführung von Apps ermöglichen.

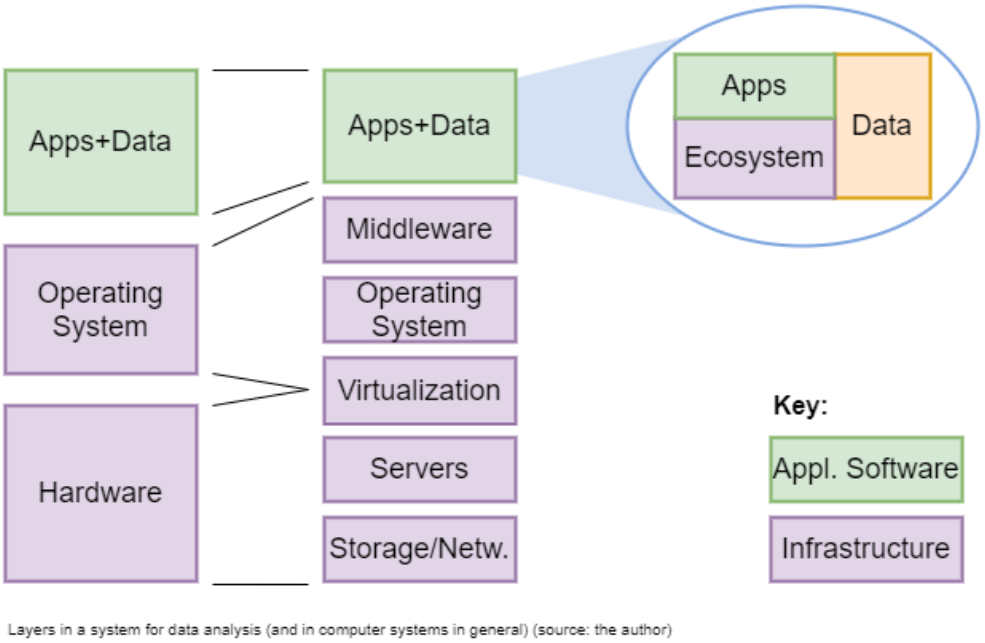


Abbildung 1. Schichten in einem generischen Lösungs-Stack

Bei näherer Betrachtung sieht das System eher aus wie in der Mitte von Abbildung 1. Der Anteil an Infrastruktur wurde grösser. Selbst wenn wir genauer in die Anwendungssoftware schauen, erkennen wir, dass diese oft eine Vielzahl von Bibliotheken/Paketen einbindet und externe Werkzeuge aufruft, was manchmal als ein «Ökosystem» bezeichnet wird. Typisches Beispiel: Python und seine unzähligen Pakete und Werkzeuge. Kommerzielle Softwarelösungen integrieren oft ein «Ökosystem» in einer Anwendung, was aus Anwendersicht einen wesentlichen Komfort darstellt.

Ein wesentlicher Teil der Infrastruktur ist das **Betriebssystem**, welches die Hardware-Ressourcen verwaltet, abstrahiert, und den Anwendungsprogrammen verfügbar macht. Der Markt für Betriebssysteme wird von dreien beherrscht: Windows, macOS, Linux. Es lässt sich beobachten, dass je länger je mehr Unix der gemeinsame Nenner aller drei ist: Linux wurde von Beginn weg nach Unix modelliert, macOS basiert seit 2001 auf einem Unix-Kern, und Windows verfügt seit 2016 auf Windows 10 und neuer mit dem *Windows Subsystem for Linux* (WSL)[1] über die Möglichkeit, Linux-Programme auszuführen; insbesondere ist es möglich, eine Linux-Distribution, z.B. Ubuntu, auf Windows zu installieren. Kenntnisse im Umgang mit der Unix Kommandozeile (die Shell und die gängigsten Tools) sind somit universell einsetzbar.

Nach einer verständlichen Euphorie für graphische Benutzerschnittstellen hat sich gezeigt, dass eine durchdachte **Kommandozeile** nach dem Vorbild jener der Unix-Systeme ein sehr effizientes Mittel für die Interaktion mit einem Computer ist und vor allem für die Reproduzierbarkeit und die Automatisierung einer graphischen Schnittstelle weit überlegen ist. Das mag auch der Grund sein, weshalb heute viele Systeme neben einem API auch über eine CLI (Command Line Interface) verfügen (dazu mehr in späteren Abschnitten).

Höhere Schichten stellen Anforderungen an tiefere Schichten, oft als **System Requirements** bezeichnet. Diese sollten bekannt und dokumentiert sein. Eine Internet-Suche nach «system requirements ProductName» sollte diese finden, zumindest für kommerzielle Produkte. Im Open Source Bereich sind die Anforderungen meistens weniger formell oder gar nicht beschrieben; hier sind Erfahrung und Experimente gefragt (was wegen der fehlenden Lizenzgebühr etwas einfacher möglich ist).

In der Software-Entwicklung und -Architektur spricht man oft von einem **Software-Stack** und meint damit wiederum eine Schichtung von Werkzeugen und Bibliotheken, welche in ihrer Gesamtheit ein Softwaresystem ausmachen. Ein bekanntes Beispiel für einen Software-Stack ist «LAMP» und steht für Linux-Apache-MySQL-PHP (Betriebssystem-Webserver-Datenbank-Skriptsprache), auf welcher Basis viele Web-Anwendungen entwickelt wurden.[2] Zwei bekannte Stacks für die Software-Entwicklung sind Java und .NET Core.

Wenn man sich mit der Infrastruktur der (räumlichen) Datenanalyse befasst, besteht eine der ersten Aufgaben darin, einen geeigneten Stack zusammenzustellen, indem eine spezifische Auswahl an Werkzeugen, Diensten und Bibliotheken getroffen wird. Dies kann ad-hoc geschehen, wird sich aber an der praktischen Arbeit messen müssen und sich gestützt auf Erfahrungswerte bei einem «Hausstack» einpendeln und von da mit dem Fortschritt der Technik weiterentwickeln. Einher mit dem Stack gehen Lernen, Dokumentation und Erfahrung, so dass auch ein auf Open Source gestützter Stack eine Investition darstellt, die nicht mehr beliebig änderbar ist. In der Praxis sind mehrere Stacks möglich und üblich, erfordern aber gegenseitige Isolation, v.a. bei ähnlichen Stacks (Java und Python werden sich kaum in die Quere kommen, bei einem Python-Stack mit Jupyter Notebooks und einem Python-Stack für die Entwicklung von Processing Tools sind Isolations-Massnahmen unumgänglich). Idealerweise laufen die oberen Teile des Stacks auf allen gängigen Betriebssystemen; im Sinne der Reproduzierbarkeit der Analysen ist das ein erstrebenswerter Anspruch, aber auch ein hoher Anspruch, der sich nicht immer erreichen lässt.

[1] Dokumentation zu Window Subsystem for Linux: docs.microsoft.com/en-us/windows/wsl

[2] LAMP ist hier nur als bekanntes Beispiel erwähnt und nicht als Empfehlung zu verstehen. Es gibt viele alternative Stacks, z.B. WIMP (Windows-IIS-MySQL/MariaDB-PHP/Perl/Python). Der geeignete Stack ist im Einzelfall zu ermitteln. Ein Software-Stack wird auch als *Solution-Stack* bezeichnet.

Je nach Grösse und Organisation Ihrer Institution steht Ihnen eine IT-Abteilung zur Verfügung, welche für mindestens Teile der Infrastruktur und des Software-Stacks zuständig ist. Das kann eine Annehmlichkeit ebenso wie eine Limitation darstellen. Vielleicht sind Sie auch auf sich selbst angewiesen und neben der Datenanalyse für Ihre eigene Infrastruktur zuständig. In letzterem Fall werden fertig zusammengestellte und erprobte Stacks willkommen sein, sei es als freie Software oder als kommerzielle Plattformen (oder eine Kombination), und Dienste in der Cloud werden sich als hilfreich erweisen.

Die aktuell dominante **Plattform** für geographische Daten und Analysen ist wohl ArcGIS von Esri. Mit einem pay-per-use-Modell stehen einem eine Vielzahl von Analysemöglichkeiten, aber auch Werkzeuge zum Datenmanagement, zur Datenmodellierung, und zur Präsentation zur Verfügung. Die Plattform ist offen und dokumentiert, ein Datenaustausch und sogar Interoperabilität mit anderen Systemen ist gewährleistet.

Spatial Data Infrastructure

In der Wortfügung «Spatial Data Infrastructure» erhalten wir eine anders gelagerte, aber ebenfalls sehr wichtige Bedeutung des Begriffes Infrastruktur für die Arbeit in der Datenanalyse: eine Spatial Data Infrastructure (SDI) ist eine geordnete und kuratierte Bereitstellung von Geodaten. Solche Infrastrukturen werden typischerweise von der öffentlichen Verwaltung aufgebaut und angeboten, meist mit einem gesetzlichen Auftrag (Schweiz: Bundesgesetz über Geoinformation). Die wichtigsten Komponenten einer SDI sind ein Geoportal im Internet, Metadaten und eine gute Suchfunktion (Hu 2017); natürlich braucht es auch geeignete Hard- und Software, eine Organisation und Personal, sowie eine finanzielle Ausstattung. Die Daten selbst liegen im Allgemeinen bei deren Produzenten. Damit ergibt sich ein Muster, das als publish-find-bind bekannt ist (Rose 2004): der Produzent publiziert seine Daten auf dem Geoportal der SDI (publish), der Anwender sucht und findet passende Daten im Geoportal (find), konsumiert diese dann aber direkt vom Produzenten (bind).

Bemühungen für den Aufbau einer nationalen SDI begannen in den USA 1993. Hintergrund war das oft schwierige Auffinden von Daten (*resource discovery*), die über diverse Ämter verteilt erhoben und gepflegt werden. Ein zweiter Aspekt war die Vermeidung von Redundanzen dadurch, dass mehrere Ämter ihre Daten auf einer Plattform teilen (oder zumindest sichtbar machen). Die Benennung «National Spatial Data Infrastructure» war wohl bewusst gewählt, um die Bemühung in die Nähe von bekannten Infrastrukturen wie das Strassennetz oder Stromleitungen zu rücken.

Das Pendant in der Schweiz heisst Bundes-Geodaten-Infrastruktur (BGDI) und basiert auf dem Bundesgesetz über Geoinformation (GeoIG) von 2008.

> *Dieses Gesetz bezweckt, dass Geodaten über das Gebiet der Schweizerischen Eidgenossenschaft den Behörden von Bund, Kantonen und Gemeinden sowie der Wirtschaft, der Gesellschaft und der Wissenschaft für eine breite Nutzung, nachhaltig, aktuell, rasch, einfach, in der erforderlichen Qualität und zu angemessenen Kosten zur Verfügung stehen. (Art. 1 GeoIG)*

Beispiele für (Spatial) Data Infrastructures mit Fokus Schweiz:

- geo.admin.ch – das Geoportal des Bundes; auf dieser Seite findet sich auch die nette Broschüre «Geoinformation für alle – geo.admin.ch» zum Download.
- geocat.ch – Katalog der Geodaten der Schweiz (Bund, Kantone, Gemeinden, Private)

- opendata.swiss – Portal für Open Government Data schweizweit (keine *Spatial* DI, enthält aber auch viele räumliche Daten)
- geolion.zh.ch – Geodatenkatalog Kanton Zürich
- www.stadt-zuerich.ch/geodaten – Geodaten der Stadt Zürich
- INSPIRE – Geodateninfrastruktur in der EU, eine Richtlinie, welche die Mitgliedsstaaten zur Bereitstellung von Geobasis- und -fachdaten verpflichtet (über Netzdienste). INfrastructure for SPatial InfoRmation in Europe.
- data.gov – U.S. Government Open Data
- Keine SDI, aber eine nützliche Sammlung: en.wikipedia.org/wiki/List_of_GIS_data_sources

Etwas anders gelagert ist der *Living Atlas of the World* von Esri (livingatlas.arcgis.com)[3]. Hier handelt es sich um eine grosse Kompilation von weltweiten Geodaten, d.h., die Daten werden nicht referenziert (wie bei einer SDI die nach dem publish-find-bind Muster funktioniert), sondern sie werden (oft) von autoritativen Quellen erworben/lizenziert, eingepflegt, aufdatiert, auf der eigenen Daten- und Softwareplattform abgelegt, dort kuratiert und präsentiert. Die Einbindung in Esri's eigene Software gestaltet sich sehr einfach.

Seit 1. März 2021 stehen die digitalen Standardprodukte (digitale Karten, Luftbilder, Landschaftsmodelle) von swisstopo, der Schweizerischen Landestopografie, online kostenlos zur Verfügung. Dies erfolgte im Rahmen der Open Government Data (OGD) Strategie und entspricht einer allgemeinen Tendenz. Vorreiter sind auch hier die USA, welche schon lange mit Steuergeldern erhobene Daten frei zur Verfügung stellen. Studien im Vereinigten Königreich und in Österreich kamen schon früh zum Schluss, dass die zu erwartenden zusätzlichen Mehrwertsteuer-Einnahmen die wegfallenden Gebühren bei der kostenlosen Datenabgabe mehr als kompensieren (Frank 2003).

[3] Produktbeschreibung zum Living Atlas: https://www.esri.com/en-us/arcgis/products/living-atlas

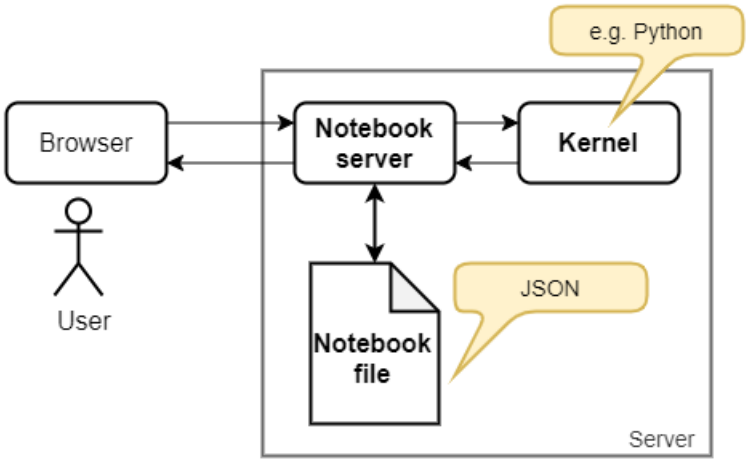
3 Notebooks

Interaktive «Computational Notebooks» sind ein Paradigma der Benutzer-Interaktion mit dem Computer, das sich anlehnt an das papierene Notizbuch, etwa in Form eines Labor-Journals. Die Idee besteht darin, dass erläuternder Text und ausführbarer Code gemischt in einem Dokument gespeichert werden kann, dass der Code auch tatsächlich ausgeführt werden kann, und dass der Output des Codes wiederum im Notebook gespeichert wird. Das Konzept ist sehr allgemein und mit beliebigen Programmiersprachen möglich; hier sei aber ausschliesslich Python betrachtet.

> À propos Python: In der Literatur zu Data Science wird gerne noch Python 2.x verwendet. Aktuell (2022) ist aber unbedingt Python 3.x zu empfehlen, weil Python 2.x seit 2020 nicht mehr weiter gepflegt wird.

Notebooks mit Python sind das vermutlich am weitest verbreitete Tooling für Data Science. Der Grund liegt wahrscheinlich darin, dass das Paradigma der Computational Notebooks das explorative Arbeiten mit Daten sehr gut unterstützt, und dass das Python Ökosystem über eine sehr grosse Zahl relevanter Pakete verfügt, die zudem als Open Source und gratis zur Verfügung stehen.

Die **Architektur** eines Notebook-Systems ist in Abbildung 2 gezeigt. Das Notebook selbst ist eine Datei, bestehend aus einer Sequenz von Zellen, jede Zelle ist entweder beschreibender Text, oder ausführbarer Code, oder das Ergebnis der Code-Ausführung. Der Anwender interagiert über seinen Webbrowser mit dem Notebook Server, dieser wiederum bearbeitet die Notebook-Datei und reicht eingebetteten Code an den Kernel weiter zur Ausführung.



Source: author's drawing after the similar illustration on jupyter.org

Abbildung 2. Architektur eines Notebook-Systems

Die führende offene Implementation eines Notebook-Systems ist **Jupyter Notebook** vom Jupyter-Projekt. Dieses Projekt entstand 2014 aus dem IPython Projekt, welches wiederum eine angereicherte interaktive Shell für die interaktive Arbeit mit Python bereitstellt, jedoch noch keine Notebooks. Als User Interface empfiehlt sich das neuere Projekt JupyterLab, welches neben den eigentlichen Jupyter Notebooks noch einen Dateibrowser integriert, mehrere offene Notebooks unterstützt, und damit ein Integrated Development Environment (IDE) für Notebooks darstellt.

Anaconda (www.anaconda.com) ist eine Python-Distribution, welche neben dem eigentlichen Python-Interpreter (*python.exe*) eine Vielzahl von gängigen Paketen, einen Paketmanager (conda), und den «Anaconda Navigator» als bequemen Einstiegspunkt bietet. Insbesondere beinhaltet die Anaconda-Distribution Jupyter (jupyter.org), NumPy (numpy.org), Matplotlib (matplotlib.org) und Pandas (pandas.pydata.org), nicht aber Geopandas.

Pakete enthalten Python-Code für eine bestimmte Funktionalität; zum Beispiel enthält das Paket matplotlib Funktionen, um Plots zu zeichnen. Pakete sind abhängig von anderen Paketen, wenn sie deren Funktionen benötigen. Ein Paket funktioniert nur dann korrekt, wenn auch alle jene Pakete installiert sind, von denen es abhängt. Ein Paket-Manager kann diese Abhängigkeiten erkennen und mit installieren. Der Paketmanager der Anaconda-Distribution heisst **conda**. Mit **mamba** steht eine schnellere Alternative zur Verfügung. Paket Manager arbeiten mit einem oder mehreren Repositories, in welchen die Pakete liegen. Bei conda werden diese als **Channel** bezeichnet. Der Standard-Channel von Anaconda heisst «defaults» und wird von der Firma hinter Anaconda (Anaconda Inc., vormals Continuum) gepflegt. Ein bekannter alternativer Channel heisst «conda-forge» und wird von einer freiwilligen Community gepflegt. Die Menge der installierten Pakete bilden die Umgebung (*environment*) von Python. Bei der Installation von Paketen in eine Umgebung kann es zu Konflikten kommen, nämlich wenn von einem Paket A andere Pakete B und C abhängen, diese aber das Paket A in unterschiedlichen Versionen fordern. Eine gängige Lösung besteht darin, mehrere Umgebungen zu verwenden und jeweils nur die nötigen Pakete zu installieren, was das Potential zu Konflikten minimiert. Pakete können einzeln installiert werden, es besteht aber auch die Möglichkeit, Pakete in einer Beschreibungsdatei aufzulisten und dann alle aufs Mal zu installieren. Das hat den zusätzlichen Vorteil einer ausführbaren Dokumentation (und wird in den Abschnitten zur Isolation und zur Automation wieder aufgenommen).

> **Best Practice:** install all packages from the same conda channel (if possible)

> **Best Practice:** do not mix conda and pip for installations (if possible)

Nachfolgend eine kurze Liste gängiger Befehle an den Paket Manager (conda oder mamba; mamba hat die gleiche Syntax wie conda):

```
conda --version          # Version von conda anzeigen
conda --help             # kurze Anleitung
conda info               # zeigt u.a. das aktive Environment
conda list               # listet Pakete im aktiven Environment
conda env list           # listet erkannte Conda Environments
conda install PACKAGE
conda install -n ENV -c CHAN PACKAGE
```

Verweise auf Literatur und Online-Ressourcen:

- G Boeing, D Arribas-Bel, “GIS and Computational Notebooks.” *The Geographic Information Science & Technology Body of Knowledge*, 2021, doi.org/10.22224/gistbok/2021.1.2 (sehr empfohlen)
- S J Rey, “Python for GIS.” *The Geographic Information Science & Technology Body of Knowledge*, 2017, gistbok.ucgis.org/bok-topics/python-gis (empfohlen)
- H Tenkanen, V Heikinheimo, D Whipp, *Introduction to Python for Geographic Data Analysis*, CRC Press, upcoming, online at pythongis.org (optional)
- List and graphic depiction of packages in the Python ecosystem that are relevant for geographic information and earth observation: ecosystem.pythongis.org

- Ein Vergleich von conda und pip (Pythons eigenem Paketmanager): www.anaconda.com/blog/understanding-conda-and-pip

Mit Esri hat auch der grosse GIS-Lieferant Jupyter Notebooks in ihre ArcGIS-Plattform eingebunden, sowohl in die Desktop-Anwendung ArcGIS Pro als auch in die Cloud Plattform ArcGIS Online und deren On-Premise-Variante ArcGIS Enterprise. Auf der Cloud Plattform müssen die Notebooks pro Benutzer freigeschaltet werden, weil mit der serverseitigen Berechnung effektive Kosten verbunden sind. Wie bei der Anaconda-Distribution stehen auch in ArcGIS Notebooks sehr viele Python-Pakete zur Verfügung und können nachgeladen werden (ArcGIS verwendet die Anaconda-Distribution).

Computational Notebooks mit Jupyter und Python sind aktuell ein grosser Trend, stehen aber weder allein da noch sind sie eine neue Erfindung. Das mathematisch-naturwissenschaftliche Programmpaket Mathematica (wolfram.com/mathematica) verwendet seit Ende der 1980er Jahren Notebooks zur Interaktion mit dem Anwender. Zudem verfügt Mathematica über viel Funktionalität für die Datenanalyse und auch für Geodaten. Mit Mathics (mathics.org) steht eine freie Alternative zur Verfügung, die aber bei weitem nicht an das kommerzielle Vorbild heranreicht. Eine neuere Alternative zu Jupyter Notebooks ist Observable (observablehq.com), eine kommerzielle Plattform die aus der Open Source Visualisierungs-Software D3.js (d3js.org) entstanden ist und den Fokus auf JavaScript und die gemeinsame kollaborative Arbeit legt.

Nach ersten Erfahrungen in der Arbeit mit Computational Notebooks sind Überlegungen angezeigt zu folgenden Fragen:

- Inwiefern sind Notebooks ein neues Paradigma in der Nutzerinteraktion mit Computern?
- Ein Wechsel weg von «point and click» Desktop-Anwendungen zu Code-basierten (*code centric*) Notebooks hat Vor- und Nachteile. Welche?
- Die folgenden Aussagen sind oft zu vernehmen. Stimmen Sie zu? Warum (nicht)?
    - Notebooks sind einfach mit anderen zu teilen
    - Notebooks begünstigen eine offene Wissenschaft (*open science*)
    - Notebooks sind selbst-dokumentierend
    - Notebooks sind wiederholbar/reproduzierbar
- Woher kommen die eingebundenen Python-Pakete?
- Eignen sich Notebooks für den Übergang in die Produktion?

Notebooks werden in späteren Abschnitten wieder auftauchen.

4 Daten, Dateien, Dienste

Datenanalyse bedingt das Vorhandensein von Daten. Diese können selbst erhoben werden (was hier nicht thematisiert wird), oder aber sie liegen schon vor, vielleicht noch ohne unser Wissen (siehe dazu aber den Abschnitt über Spatial Data Infrastructure).

Daten sind das Gegebene (lat. *datum* = gegeben) und bilden die Grundlage für Informationen, und diese Wiederum unterliegen dem Wissen und der Entscheidungsfindung. Dieser Zusammenhang wird gern als Pyramide dargestellt, wie in Abbildung 3 gezeigt.

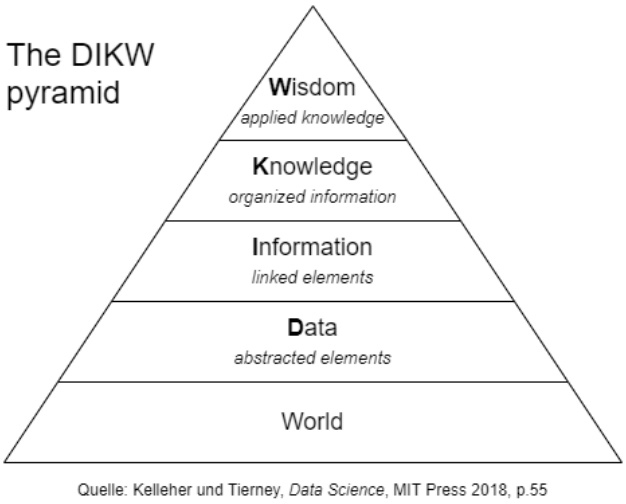
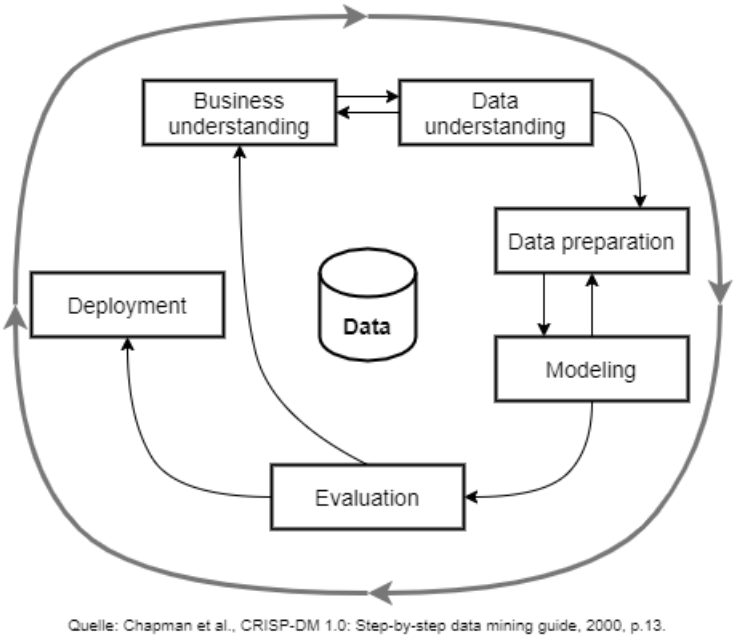


Abbildung 3. Die DIKW-Pyramide

Das Ziel einer Datenanalyse ist es, von der untersten Datenebene in die höheren Ebenen der Information und des Wissens zu gelangen. In der Industrie wird das Vorgehen in solchen Datenprojekten durch den «Cross-Industry Standard Process for Data Mining» (CRISP-DM, Chapman et al. 2000) beschrieben. Es zeigt dies ein datenzentriertes Vorgehen, beginnend beim Geschäftsverständnis und in einer oder mehreren Iterationen über sechs Phasen zu einem Einsatz der gewonnenen Erkenntnisse gelangend (Abbildung 4). Nicht gezeigt, aber nicht weniger wichtig ist neben den Daten eine Infrastruktur, welche den Prozess mitträgt. Hinter jeder Phase steht eine Reihe von Aufgaben, etwa eine Situationsbeurteilung im Geschäftsprozess, eine Qualitätsprüfung der Daten, die Auswahl einer Modellierungstechnik, etc. Wichtig sind auch die kurzen Rückkopplungen, welche auf das explorative Vorgehen in diesen Phasen hinweist. Erst beim Deployment, dem Einbringen der Erkenntnisse in die Geschäftsprozesse, findet ein Übergang in die Produktion statt und ändern sich die Anforderungen an Vorgehen und Infrastruktur ganz wesentlich.



Quelle: Chapman et al., CRISP-DM 1.0: Step-by-step data mining guide, 2000, p.13.

Abbildung 4. Das CRISP-DM Modell

Zurück zu den Daten: Diese lassen sich sammeln, oft akkumulieren sie sogar von selbst, und daraus entstanden neue Fachrichtungen wie das Data Mining und die Data Science, und neue Begriffe wie das VGI (siehe unten). Daten selbst sind aber nur Mittel zum Zweck und kein Ziel in sich, wie die folgenden Zitate zeigen:

- “Knowledge is ordered access to information” (Guerino Mazzola)
- «Wissen kann man mitteilen, Weisheit aber nicht» (aus Hesses Siddharta)
- “Where is the wisdom we have lost in knowledge? Where is the knowledge we have lost in information?” (T.S. Eliot)
- “What information consumes is attention. Hence a wealth of information creates a poverty of attention.” (H. Simon)

Nach dieser kurzen Besinnung darüber, wo wir uns bewegen, wenn wir mit Daten arbeiten, erkennen wir aber auch, dass Daten eine tiefe Schicht sind und somit ein Teil der Infrastruktur der Datenanalyse. Und in diesem Kontext sind die folgenden Betrachtungen zu Daten wichtig (nach Kelleher und Tierney, 2018):

- Daten sind *strukturiert* oder *unstrukturiert*. Strukturierte Daten liegen in Tabellenform vor, haben also ein Schema und sind oft in einer (relationalen) Datenbank gespeichert. Unstrukturierte Daten sind Dokumente, z.B. Webseiten, Bilder oder Tonaufnahmen, aus denen Daten vor einer Analyse erst extrahiert werden müssen.
- Daten können als *Rohdaten* (z.B. Messwerte) verwendet werden, oder aber es werden aus ihnen andere Daten *abgeleitet* (z.B. eine Bevölkerungsdichte aus einer Volkszählung und einer Landvermessung).
- Daten können *bewusst erfasst* worden sein (durch Messung oder Beobachtung), oder aber sie sind *beiläufig angefallen* (z.B. Logdateien, Kameraüberwachung, Sensoren im persönlichen Mobiltelefon) oder stellen ein Nebenprodukt dar (z.B. Metadaten).

Bei räumlichen Daten wurde der Begriff der *Volunteered Geographic Information* (VGI, Goodchild 2007) geprägt, dies im Unterschied zu den Professional Geographic Data, wie sie in einem institutionellen oder kommerziellen Rahmen erfasst werden. Klassisches Beispiel für VGI

ist Open Street Map, wo unbezahlte Individuen ohne institutionellen Rahmen einen grossen und wachsenden Datensatz zusammengetragen haben.

Geographische Phänomene werden gerne abstrahiert als *Objekte*, *Felder* oder *Netzwerke*. Dem entsprechen die datentechnischen Repräsentationen Features, Raster und Graphen. Umwandlungen zwischen den Repräsentationen sind möglich, aber mit Verlusten behaftet. Man spricht von «Rasterisierung» und «Vektorisierung».

Abstraktion	gängige Repräsentation	Beispiel
Objekte	Features (Vektorgeometrie)	Administrative Grenzen
Felder	Raster, TIN	Niederschlagsmenge
Netzwerke	Graphen	Verkehrsnetz

Features repräsentieren diskrete Einheiten in der realen Welt mit ihrer geometrischen Ausprägung (Punkt, Linien, Fläche) und Sachattributen. Für die Geometrie eines Punktfeatures genügt ein Koordinatenpaar (x, y), für Linienfeatures ist es eine Sequenz von Koordinatenpaaren, ebenso für Flächenfeatures, wobei bei letzteren angenommen wird, dass der der Linienzug geschlossen ist. Ferner können mehrere Punkte als ein Multipunkt gespeichert werden, mehrere Linienzüge als eine Polylinie, und mehrere einfache Flächen als Polygone (für disjunkte oder «löchrige» Flächen, etwa ein See mit Inseln).

Raster bestehen aus einem zweidimensionalen Array von Zellen, analog zu einem Rasterpapier. Jede Rasterzelle wird Pixel (*picture element*) genannt und speichert den Wert des Rasters an jenem Ort. Für die Farbbilder einer typischen Kamera werden zu jeder Rasterzelle drei Farbwerte erfasst, Rot, Grün und Blau. Die Werte der Ausprägung «Rot» über alle Rasterzellen werden als Band oder Kanal bezeichnet; analog für die anderen Ausprägungen. Bei Rasterdaten aus der Fernerkundung entsprechen die Bänder eines Rasters meist bestimmten Ausschnitten aus dem Frequenzspektrum, etwa dem nahen Infrarot (Wärmestrahlung), einem Ausschnitt aus dem sichtbaren Bereich, oder einer Radar-Frequenz. Die Genauigkeit, mit der die Pixelwerte gespeichert werden, wird als bit depth oder pixel depth bezeichnet und ist die Anzahl Bit pro Pixel (oder pro Pixel und Kanal), oft 1 (schwarz/weiss) oder 8 (256 Abstufungen) pro Kanal.

Graphen zur Repräsentation von Netzwerken bestehen aus Knoten (*nodes*) und Kanten (*edges*), die über Sachattribute und/oder eine geometrische Verortung verfügen können. Kanten können je nach Anwendungsfall gerichtet oder ungerichtet (bidirektional) sein. Wesensmerkmal ist in jedem Fall die topologische Beziehung: die zu einem Knoten inzidenten Kanten und die über eine Kante adjazenten Knoten.

Bei allen räumlichen Daten wesentlich ist der Bezug der darin implizit (Raster) oder explizit (Vektoren) vorhandenen Koordinatenwerte zur Lage in der realen Welt. Dies wird durch ein **räumliches Bezugssystem** (*spatial reference system*, SRS) erreicht. Diese Referenzsysteme werden unterteilt in Geographische Koordinatensysteme (sphärische Koordinaten im Gradnetz der Erde) und projizierte oder geodätische Koordinatensysteme (kartesische x/y-Koordinaten in der Ebene). Wichtig ist, dass allen räumlichen Daten diese Information mitgegeben wird. Mehr zu den Bezugssystemen im Abschnitt über Standards.

Für die konkrete Speicherung in Dateien gibt es eine Vielzahl von **Dateiformaten**. Hier ist es zweckmässig, diese zu unterteilen in Formate für die Quelldaten (Eingabe für die Analyse) und Formate für die Präsentation (Ausgabe der Analyse). Nur eine kleine Auswahl kann hier präsentiert werden.

Häufige Dateiformate für die Quelldaten von Analysen:

Plain Text (*.txt*, manchmal auch .dat) – keine strukturellen Einschränkungen. Die Extraktion der Information kann einfach oder schwierig sein, je nach den angewendeten Konventionen. Früher war oft die Kolonne bedeutsam (z.B. Temperatur als zwei Ziffern in Kolonnen 7 und 8), heute werden eher Trennzeichen, was uns dann zu CSV und TSV führt.

CSV (*.csv*) und **TSV** (*.tsv*) – Comma/Tab Separated Values, ein Plain Text Format, bei dem jede Zeile einer Beobachtung (*observation*, *record*, *feature*) entspricht, und darin die Variablen (*values*, *fields*) durch Komma oder Tab getrennt sind. Achtung: kein standardisiertes Format! Es gibt Varianten bezüglich der Trennzeichen (zwischen den Feldern und zwischen den Zeilen), sowie unterschiedliche Quoting- und Escape-Konventionen (womit zwischen Beobachtung und Zeile nicht mehr eine 1:1 Beziehung bestehen muss). Fazit: `for line in textfile: fields=line.split()` (also zeilenweises Lesen mit anschliessender String-Split-Operation, wie es oft in Beispiel-Code gesehen wird) mag oft funktionieren, ist aber nicht korrekt. Beim Schreiben ist auf die zu wählende Variante zu achten (Hinweis: Semikolon als Feldtrenner funktioniert gut mit Excel). Python verfügt über das Standard-Modul `csv`[4] zum Lesen und Schreiben verschiedener Varianten von CSV/TSV, `pandas` und andere Pakete aus dem Data Science Umfeld verfügen ebenfalls über diese Funktionalität.

Excel (*.xls*/*.xlsx*) – das Dateiformat von Microsofts Tabellenkalkulation kann auch von einigen fremden Software-Paketen gelesen und geschrieben werden. Wegen der grossen Popularität von Excel liegen viele Daten in diesem Format vor.[5] Der Raumbezug ist, wie auch bei CSV/TSV, entweder über enthaltene Adress-/Ortsangaben oder über explizit abgelegte Koordinaten. Das `pandas` Paket kann das Excel-Format lesen und schreiben, es können dabei auch einzelne Worksheets adressiert werden.

Shapefile (*.shp*) – das ursprüngliche Format von Esri ArcView GIS für die Speicherung von Vektordaten. Wegen der Publikation seiner Spezifikation als White Paper[6] wurde das Format zu einem de facto Standard und wird breit unterstützt. Das Format besteht aus mehreren Dateien in einem Ordner: *.shp* speichert die Geometrie, *.dbf* speichert die zugehörigen Attributdaten, *.shx* ist ein Index für den schnellen Zugriff auf Geometrien in *.shp*; optional können weitere Dateien vorhanden sein, etwa *.prj* mit Informationen zum verwendeten Koordinatensystem. Shapefiles speichern jede Geometrie individuell, topologische Informationen (wie Nachbarschaftsbeziehungen oder gemeinsame Grenzen) zwischen den Geometrien sind nicht gespeichert und müssen bei Bedarf von Programmen aus den Koordinaten rekonstruiert werden. In einem Shapefile können nur Geometrien eines Geometrietyps (Punkte, Linien, Polygone) gespeichert werden.

GeoJSON (*.geojson*, *.json*) – kann räumliche Daten in einem JSON-Format speichern, sowohl einzelne Geometrien als auch Features, d.h. Geometrien mit Sachattributen. Der Umfang der Geometrien entspricht den OGC Simple Features (siehe Abschnitt zu Standards). Jede Geometrie speichert explizit alle ihre Koordinaten und zwischen den Stützpunkten wird linear interpoliert. GeoJSON wurde etwa 2008 initiiert und ist seit 2016 spezifiziert in RFC 7946.[7] **TopoJSON** ist eine interessante Erweiterung von GeoJSON, welche die Geometrien redundanzfrei und quantifiziert abspeichert. TopoJSON-Dateien sind kleiner und lassen sich

[4] Dokumentation in docs.python.org/3/library/csv.html und Hintergründe in peps.python.org/pep-0305/
[5] Die erste Tabellenkalkulation für den Computer, VisiCalc 1979, eröffnete ein neues Paradigma für die Interaktion mit dem Computer, ähnlich wie es die schon erwähnten Computational Notebooks tun.
[6] Esri Shapefile Spezifikation: www.esri.com/library/whitepapers/pdfs/shapefile.pdf
[7] Spezifikation von GeoJSON in RFC 7946: www.rfc-editor.org/rfc/rfc7946

besser komprimieren. Je nach Anwendungsfall kann aus der reincodierten topologischen Information Nutzen gezogen werden. GeoJSON und TopoJSON können ineinander umgewandelt werden. TopoJSON wird oft im Bereich der kartografischen Visualisierung verwendet, vor allem im Dunstfeld der D3.js Grafikbibliothek.

GeoPackage (*.gpkg*) – ein eher neues Format für räumliche Daten, Vektoren wie Raster, welches zwar in einer Datei gespeichert wird, jedoch eine ganze Datenbank mit beliebig vielen Tabellen darstellt. Basiert auf der freien Datenbank SQLite. Wurde entwickelt als Ersatz für Shapefiles und in Nachahmung der File Geodatabase von Esri (mehr dazu im Abschnitt zu Datenbanken). Seit 2014 ein OGC Standard. Siehe www.geopackage.org

GML (*.gml*) – ein OGC Standard, basiert auf XML, kann sowohl Vektordaten als auch Coverages und Sensordaten speichern. Es gibt diverse Erweiterungen von GML, die auf spezifische Anwendungen fokussiert sind, z.B. CityGML für 3D Städtemodelle.

GeoTIFF (*.tif*/*.tiff*) – eine GeoTIFF-Datei ist eine reguläre TIFF (Tag Image File Format) Datei mit eingebetteten Informationen zur Georeferenzierung (d.h., das Bild kann in Bezug zu einem geographischen Koordinatensystem gesetzt werden). TIFF ist ein gängiges Rasterformat, welches verschiedene Farbmodi und -tiefen sowie unterschiedliche Kompressionen unterstützt. Eine Alternative zu GeoTIFF ist ein reguläres TIFF mit einem zugehörigen World File (*.tfw*).

NetCDF (*.nc4*) – Network Common Data Form – ein Rasterdateiformat welches oft für mehrdimensionale wissenschaftliche Daten verwendet wird.

World File (*.tfw*, *.jgw*, *.pgw*) – speichert die sechs Koeffizienten einer affinen Transformation, die notwendig ist, um ein Raster zu georeferenzieren (also in die rechte Lage zu rücken). Plain Text, auf jeder Zeile eine Zahl in der üblichen Dezimalnotation. Wenn die Zahlen auf den Zeilen A, B, C, D, E, F sind, dann wird dadurch die affine Transformation $\begin{pmatrix} x\prime \\ y\prime \end{pmatrix} = \begin{pmatrix} A & C \\ B & D \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} E \\ F \end{pmatrix}$ ausgedrückt.[8] Falls keine Rotation vorliegt, sind $B = C = 0$ und A und D sind dann eine Skalierung; dabei ist D oft negativ, weil die Pixelkoordinaten oben links beginnen, die geodätischen aber unten links. Die Dateiendung richtet sich nach derjenigen der Datei, zu der das World File gehört: *.tif* → *.tfw*, *.jpg* → *.jgw*, *.png* → *.pgw*, allgemein *.abc* → *.acw*.

Graphen für Netzwerke werden oft im Rahmen der Analyse aus Vektordaten (z.B. aus einem Linien-Shapefile) rekonstruiert.

Dateiformate für die Präsentation von Analysen: die Dateiformate für die bekannten Office-Software-Pakete (Word, PowerPoint, etc. und ihre freien Alternativen) werden als bekannt vorausgesetzt. Bevor ein Analyse-Ergebnis in ein solches Dokument Eingang findet, wird es wahrscheinlich zunächst in einem der folgenden Formate vorliegen:

PNG (*.png*) – Portable Network Graphic, ein Rasterformat mit verlustfreier Kompression, unterstützt verschiedene Farbmodi (Graustufen, Mehrkanal, Farbpalette) und Farbtiefen, Transparenz, und wurde explizit als von Patenten nicht berührtes neues Format entwickelt.

JPEG (*.jpg*, *.jpeg*) – Joint Photographic Expert Group, bezeichnet eigentlich das effiziente, aber verlustbehaftete Kompressionsverfahren, das Dateiformat selbst ist als JFIF oder JPEG File Interchange Format bekannt, wird in der Praxis aber fast immer als «JPEG» bezeichnet. Gut

[8] Siehe auch en.wikipedia.org/wiki/World_file

geeignet für Bilddaten, wegen der verlustbehafteten Kompression aber ungeeignet für kategoriale Daten.

PDF (*.pdf*) – Portable Document Format, kann von einzelnen Graphiken (Vektor wie Raster) bis zu ganzen Büchern repräsentieren und ist gut für den Druck geeignet. Früher war zur Erzeugung Spezialsoftware nötig, heute wird PDF meist direkt als Output-Format unterstützt. Die Details sind komplex und verlangen eine vertiefte Auseinandersetzung, in den meisten Fällen kommt man aber auch ohne dieses Verständnis zurecht.

EPS (*.eps*) – Encapsulated PostScript, wird oft verwendet zur Speicherung von Vektorgraphiken, die zum Import in andere Dokumente gedacht sind. War vor allem vor der breiten Unterstützung von PDF als Ausgabeformat oft verwendet, denn als Text-Format konnte es relativ technisch einfach generiert werden, allerdings ist das zugrundeliegende PostScript komplex und verlangt ein gutes Verständnis.

SVG (*.svg*) – Scalable Vector Graphics, ein vom W3C (siehe Abschnitt über Standards) entwickeltes Format für Vektorgraphik. Syntaktisch handelt es sich um XML, es ist also ein Text-Format und kann daher relativ einfach generiert werden, was aber selten notwendig ist, weil es auch als Ausgabe-Format breit unterstützt wird. Web-Browser können SVG direkt darstellen, was SVG attraktiv macht für die Weiterverwendung im Web.

Markdown (*.md*) – das ganz einfache Textformat für die beschreibenden Teile in Jupyter Notebooks wird auch ausserhalb von Jupyter immer weiter unterstützt, vor allem im Umfeld von Werkzeugen für die Software-Entwicklung. Das macht Markdown zu einem sehr effizienten Format, um einfach strukturierte Texte zu verfassen.[9]

Es gibt viele weitere Formate. Zu allen lassen sich im Internet problemlos Informationen finden; wie meistens ist Wikipedia ein guter Einstieg.

Dateien sind eine Abstraktion über einem Speichermedium, die vom Betriebssystem angeboten wird. In allen heute gängigen Betriebssystemen hat eine Datei einen Namen und besteht aus einer endlichen Sequenz von Null oder mehr Bytes. Diese Bytes können adressiert werden: das erste, das zweite, und so weiter, in beliebiger Reihenfolge.[10] Auf dieser, aus menschlicher Sicht sehr abstrakten Basis bauen die Dateiformate auf. Weil ein Byte 8 Bits umfasst, kann ein Byte $2^8 = 256$ verschiedene Zustände darstellen, etwa die Zahlen zwischen 0 und 255. Eine grössere Zahl braucht zur Speicherung mehr als ein Byte und es stellt sich die Frage, wie die Zahl auf die Bytes aufgeteilt werden soll. An dieser Stelle soll nicht weiter darauf eingegangen werden, einige Aspekte werden aber später wieder durchschimmern.

Metadaten sind Daten über Daten (zu gr. *μετά* = nach, hinter). Metadaten beschreiben die eigentlichen Daten und sind damit eine ganz wesentliche Hilfe für die Datenanalyse. Was möchten Sie über einen Datensatz wissen, bevor Sie ernsthaft mit ihm arbeiten?

Daten werden oft nicht (nur) als Dateien zum Download zur Verfügung gestellt, sondern (auch) als **Dienste**, über welche die Daten massgeschneidert konsumiert werden können. Solche Dienste bieten immer öfter ein **API** (Application Programming Interface) an, so dass die Daten auch programmatisch und somit automatisiert bezogen werden können. (Natürlich kann auch

[9] Ein Blick in www.markdownguide.org wird genügen, um damit zu starten; von der Extended Syntax ist abzusehen, weil die nicht ganz so breit unterstützt wird.
[10] Das gilt nicht für Ströme (*streams*), die ebenfalls eine Sequenz von Bytes sind, die aber nur in geordneter Folge angesprochen werden können.

der Download einer Datei programmatisch und automatisiert erfolgen, dies setzt aber eine stabile URL voraus.) Mehr zu dieser aktuellen Thematik findet sich in der Literatur, z.B. hier:

- E Chow, Y Yuan, "GIS APIs." *The Geographic Information Science & Technology Body of Knowledge*, John P. Wilson (ed.), 2019. DOI: 10.22224/gistbok/2019.2.15.

Wie der zitierte Artikel zeigt, eröffnen APIs dem Datenanalysten einen Zugang zu diversen Systemen und Plattformen, insbesondere zu den Social Networks, die über interessante und hochaktuelle Daten verfügen. Weiter eröffnen APIs den Zugang zu Diensten für die Prozessierung, die Analyse und die Darstellung (mit dem Klassiker Google Maps). Auf der anderen Seite ist zu erwähnen, dass die Nutzung von APIs zwingend Programmierkenntnisse voraussetzt und oft auch mit Kosten verbunden sind.

Zwei konkrete Beispiele für ein API zum Datenbezug:

- opendata.swiss (siehe Abschnitt zu Spatial Data Infrastructure) verfügt über ein API für die Suche nach Open Government Data; Dokumentation dazu findet sich direkt auf der Webseite
- Esri's Python API (developers.arcgis.com/python) erlaubt ebenfalls die Suche nach Datensätzen (und weiteren Ressourcen) in ArcGIS Online (Cloud) oder in einer On-Premise Installation von ArcGIS

Die Klassiker unter den Diensten im Geobereich jedoch sind die OGC Standards WFS, WCS, WMS, welche in einem eigenen Abschnitt behandelt werden.

5 Datenbanken

Datenbanken sind ein wichtiger Teil der Infrastruktur für die Datenanalyse. Für vertiefte Betrachtungen zu Datenbanken sei auf die Literatur verwiesen (insbesondere das Standardwerk *Fundamentals of Database Systems* von R. Elmasri und S.B. Navathe in der aktuellen Ausgabe). Hier muss eine ein kurzer Überblick zu Relationalen Datenbanksystemen genügen, bevor wir uns den räumlichen Datenbanken zuwenden.

1970 wurde der Artikel "A Relational Model of Data for Large Shared Data Banks" von E.F. Codd publiziert und markiert den Beginn der Relationalen Datenbanken, wie sie auch heute noch im Einsatz sind. Die «Relation» ist das mathematische Modell dessen, was umgangssprachlich als Tabelle bezeichnet wird. Eine relationale Datenbank verwaltet Tabellen, kann darin Informationen speichern und ändern, kann die Tabellen miteinander in Beziehung setzen und vielfältige Abfragen tätigen. Das mathematische Modell hinter diesen Systemen war ein wesentlicher Faktor für den grossen und langanhaltenden Erfolg; es stellt ein geschlossenes System dar in dem Sinne, dass die Operationen auf Relationen (Tabellen) als Ergebnis wiederum eine Relation haben. Das ermöglicht beliebig komplexe Abfragen und Analysen direkt in der Datenbank. Die technische Umsetzung des mathematischen Modells ist die Aufgabe des Relational Database Management Systems (RDBMS). Der langjährige Platzhirsch in diesem Feld heisst Oracle, es gibt aber diverse Alternativen.

Die Operationen mit einer relationalen Datenbank fallen in zwei Kategorien: Manipulationen an der Struktur, auch als Schemaoperationen bekannt, und Manipulationen an den Daten in diesem Schema (also in den Tabellen). Schemaoperationen werden in einer Data Definition Language (DDL) ausgedrückt; es geht dabei im Wesentlichen darum, Tabellen zu erstellen und zu löschen, sowie ihre Struktur zu verändern. Die Struktur einer Tabelle besteht aus ihren Spalten (*columns*) und deren Datentypen (z.B. Ganzzahl, Fliesskommazahl, Text, Zeitstempel). Datenoperationen beziehen sich auf die Zeilen (*rows*) in den Tabellen. Es gibt deren vier: Create, Read, Update, Delete (CRUD), also Zeilen einfügen, abfragen, verändern und löschen.

Operationen auf einer relationalen Datenbank haben vier wichtige Eigenschaften, welche die praktische Arbeit angenehm und zuverlässig machen: jede Operation ist *atomar* (wird vollständig ausgeführt oder gar nicht), die Datenbank ist vor- und nachher *konsistent*, jede Operation läuft *isoliert* von allen anderen, und das Ergebnis ist *dauerhaft*. Das wird im Kürzel «ACID» (atomicity, consistency, isolation, durability) zusammengefasst.[11]

Mit SQL existiert eine standardisierte und weit verbreitete Sprache zur Beschreibung aller CRUD Operationen, wie auch für die Schemaoperationen (SQL ist auch eine DDL). Ein praktisches Grundverständnis von SQL ist für jeden Datenanalysten unbedingt zu empfehlen und über die Literatur und Artikel im Internet leicht zu erwerben. Fallstrick für Programmierer in Java, C# und ähnlichen Sprachen: in SQL ist `'foo'` ein String, `"foo"` aber der Name eines Feldes.

Eine konkrete Sammlung von Tabellen und Beziehungen in einem RDBMS wird als **Datenmodell** bezeichnet. Dieses kann auf drei Abstraktionsebenen betrachtet werden: konzeptionell, logisch und physisch. Details in der Literatur, etwa der sehr gute Artikel zu Datenbanken im Informatik-Handbuch (Rechenberg und Pomberger 2006).

Vor dem Siegeszug der RDBMS und wiederum in neuester Zeit unter dem Begriff «NoSQL» (Not only SQL) waren und sind alternative Konzepte zum Relationalen Modell im Einsatz. Diese

[11] Der Wikipedia-Artikel zu ACID sei empfohlen: en.wikipedia.org/wiki/ACID

alternativen Konzepte opfern die formale Strenge der relationalen Datenbanken und sind dafür einfacher und schneller, und somit je nach Anwendungsgebiet einem RDBMS überlegen. Ein Blick in das «DB-Engines Ranking» zeigt jedoch, dass die meistverbreiteten Systeme immer noch dem relationalen Modell folgen:[12]

1. Oracle (*relational*)
2. MySQL (*relational*)
3. SQL-Server (*relational*)
4. PostgreSQL (*relational*)
5. MongoDB (*document*)
6. Redis (*key-value*)

Räumliche Datenbanken (*spatial database*) sind eine Erweiterung einer relationalen Datenbank, indem sie diese um folgende Elemente ergänzen:

- Räumliche Datentypen zur Speicherung von Punkten, Linien, Polygonen
- Funktionen für die Konstruktion, Abfrage und Relation der räumlichen Typen
- Räumliche Indizierung zur performanten Ausführung von räumlichen Operationen

Im Esri-Umfeld werden diese drei Elemente als «**Spatial Type**» bezeichnet. Esri unterstützt je nach Datenbank unterschiedliche Spatial Types.[13] Esri selbst hat den Spatial Type ST_Geometry entwickelt, welcher eine Oracle- oder PostgreSQL-Datenbank zu einer räumlichen Datenbank erweitert.[14] Dieser Spatial Type hat den Ruf, äusserst performant zu sein. Oracle verfügt über einen eigenen Spatial Type, genannt SDO_Geometry. PostgreSQL kann mit PostGIS um zwei Spatial Types erweitert werden, geometry und geography. Microsofts SQL-Server verfügt ebenfalls über zwei Spatial Types geometry und geography. PostGIS wird im Folgenden genauer betrachtet.

PostGIS erweitert PostgreSQL zur räumlichen Datenbank und ist konform zum OGC Standard *Simple Features for SQL* (dazu mehr im Abschnitt über Standards). PostGIS muss für jede Datenbank, in der es verfügbar sein soll, aktiviert werden. Wie erwähnt bringt PostGIS zwei Spatial Types: geometry für kartesische Koordinaten und geography für geographische (sphärische) Koordinaten. Ebenfalls bringt PostGIS eine Vielzahl von Funktionen für räumliche Operationen, deren Namen alle mit dem Präfix ST_ beginnen. Hier ein Beispiel:

```
CREATE EXTENSION postgis;  -- PostGIS in der aktuellen DB einrichten

-- Anzahl und Länge aller Strassen im Borough Manhattan:
SELECT count(*), sum(ST_Length(s.geom))
  FROM nyc_streets AS s
  JOIN nyc_neighborhoods AS n
    ON ST_Intersects(s.geom, n.geom)
 WHERE n.boroname = 'Manhattan';
```

Das Beispiel richtet zunächst die PostGIS-Erweiterung in der aktuellen Datenbank ein, installiert also die beiden Spatial Types und die zugehörigen Funktionen. Danach folgt eine Abfrage aller Strassen im Borough Manhattan von New York, dies unter der Annahme, dass

[12] Quelle: DB-Engines Ranking, db-engines.com/de/ranking (letzter Zugriff am 10.9.2022)
[13] Vollständige Übersicht: https://pro.arcgis.com/en/pro-app/latest/help/data/databases/overview-database-spatial-types.htm (letzter Zugriff am 10.9.2022)
[14] Siehe pro.arcgis.com/en/pro-app/latest/help/data/databases/databases-and-st-geometry.htm (letzter Zugriff am 12.9.2022)

eine Tabelle `nyc_streets` mit allen Strassen von New York City als Linien existiert, und eine Tabelle `nyc_neighborhoods` mit allen Neighborhoods von New York als Polygone. Unter diesen Umständen ist eine räumliche Verknüpfung (*spatial join*) der beiden Tabellen über das Prädikat «Strasse schneidet Neighborhood» notwendig, was durch die Funktion `ST_Intersects` implementiert wird. Die Funktion `ST_Length` ermittelt die Länge der übergebenen Geometrie; diese Funktion ist überladen: gegeben eine geometry wird die Länge in der Ebene ermittelt, gegeben eine geography wird die Länge sphärisch (entlang von Grosskreisen) ermittelt.[15]

> **Konvention** bei der Verwendung von PostGIS: Spalten vom Typ geometry heissen `geom`, Spalten vom Type geography heissen `geog`.

Für praktische Arbeiten und Experimente mit PostgreSQL und PostGIS sei das graphische Administrations- und Entwicklungswerkzeug pgAdmin4 empfohlen (Links siehe unten). Darin lassen sich schön die installierte Erweiterung PostGIS und die damit verbundenen Datentypen, Funktionen, Tabellen, Views sehen. Für ein typisches Open Source Spatial Database Tooling sind folgende Online-Quellen zu notwendig (und oft auch hinreichend):

www.postgresql.org • postgis.net • www.pgadmin.org

Auf der Webseite von PostGIS findet sich ein hervorragendes Tutorial zusammen mit Public Domain Daten, welches Ihrer Aufmerksamkeit wärmstens empfohlen ist.

Esri's **Geodatabase** erweitert die zugrunde liegende relationale Datenbank nicht nur um einen Spatial Type, sondern zusätzlich um «Verhalten» wie Domänen (*Coded Value* und *Range*), Relationship-Klassen, Feature-Datasets, Topologie, Versionierung, und weiteres, welches bei der Datenmodellierung nützlich ist und im User Interface widerspiegelt wird. Diese Verhaltens-Erweiterungen sind mit Standardmitteln einer relationalen Datenbank realisiert, indem Metadaten in von ArcGIS verwalteten Tabellen abgelegt werden. Die Geodatabase unterscheidet daher zwischen Anwender Tabellen und System-Tabellen. Die Namen der Systemtabellen beginnen mit dem Präfix SDE_. Details dazu finden sich in der Dokumentation mit der Einstiegsseite unter pro.arcgis.com/en/pro-app/latest/help/data/geodatabases und in spezifisch zu den Systemtabellen in desktop.arcgis.com/en/arcmap/latest/manage-data/using-sql-with-gdbs/overview-geodatabase-system-tables.htm

Eine spezielle Ausprägung der Geodatabase ist die **File Geodatabase**, welche ohne DBMS auskommt und in Form von Dateien in einem Ordner mit der Endung *.gdb* gespeichert wird. Diese verfügen über eine äusserst gute Performance, erlauben aber naturgemäss keinen Multiuser-Zugriff. Ähnlich gelagert ist das Dateiformat der GeoPackage (*.gpkg*), wie weiter oben schon erwähnt.

Für die praktische Arbeit in der Datenanalyse stellt sich noch die Frage, wie programmatisch auf eine Datenbank zugegriffen wird. Dafür gibt es Bibliotheken (im Falle von Python Module und Pakete). Ein gängiges Muster für die Verwendung einer solchen Bibliothek ist wie folgt: erst wird eine Verbindung zur Datenbank aufgebaut, indem Informationen wie Servername (oder IP-Adresse), Datenbankname, Benutzername und Kennwort bekannt gegeben werden. Danach werden SQL-Statements, oft in Form von einfachen Strings, an die Verbindung gegeben. Die Antwort ist ein Cursor über die Ergebnisse. Auf höherer Abstraktionsebene kann auch direkt eine Abbildung von Objekten der verwendeten Programmiersprache auf die Daten

[15] In Systemen mit nur einem Spatial Type, z.B. ST_Geometry oder SDO_Geometry in Oracle, wird anhand der SRID entschieden, ob kartesische oder sphärische Berechnungen vorgenommen werden.

in den Tabellen der Datenbank erfolgen; man spricht dann von *Object Relational Mapping* (ORM).

Für den Zugriff auf PostGIS aus Jupyter Notebooks mit Python empfiehlt sich das Paket `geopandas` (geopandas.org) in Kombination mit `sqlalchemy` (www.sqlalchemy.org).

```
from sqlalchemy import create_engine
url = "postgresql://username:password@dbhost:5432/dbname"
con = create_engine(url)
sql = "SELECT geom, highway FROM roads"
df = geopandas.read_postgis(sql, con)
```

6 Processing

Mit «Processing» sei hier die automatische Verarbeitung von Daten gemeint. Im Rahmen der Datenaufbereitung (*data preparation* in CRISP-DM) wird man die Daten für eine Analyse so weit vorbereiten, dass sie für die Analyse geeignet sind. Dazu gehören Aufgaben wie: Qualitätsprüfung, Formatanpassungen, Entfernung nicht benötigter Informationen, Ergänzung (Schätzung) fehlender Informationen, Harmonisierung, Projektion und geodätische Transformation, allenfalls Speicherung in einer räumlichen Datenbank. Natürlich kann auch die Analyse selbst und die Aufbereitung der Resultate als Teil des Processing verstanden werden.

Die Essenz im Processing ist die Möglichkeit, einzelne Schritte zu verketten und die Kette der Schritte als Ganzes zu betrachten. Im einfachsten Fall geschieht dies über die Kommandozeile, wo die einzelnen Schritte der Ausführung eines Programms entsprechen. Beim frühen GIS GRASS war dies das Paradigma der Anwendung und die mächtigen Fähigkeiten der Unix Shell wurden gekonnt genutzt. Grundsätzlich erlaubt die Kette eine parallele Ausführung durch Streaming der Daten durch die Kette, in der aktuellen GIS-Praxis ist aber eher die strikt schrittweise Ausführung mit Speicherung der Zwischenergebnisse anzutreffen. Auch wenn einzelne Schritte sehr performant sind, ist die Kette nie schneller als das langsamste Glied.

Moderne GIS bieten dafür spezifische Funktionalität: bei Esri ArcGIS wird diese als Geoprocessing[16] (GP) bezeichnet, das Pendent von QGIS heisst schlicht Processing[17]. Beiden ist gemeinsam, dass einzelne Prozess-Schritte zu Ketten verbunden werden können, die sequenziell durchlaufen werden. Für komplexere Fälle sind auch Verzweigungen und Iterationen machbar. Beide Systeme bieten Hunderte von Tools (ArcGIS: *GP Tools*, QGIS: *Algorithmen*), die für die einzelnen Prozessschritte verwendet werden können. Sollte für eine Funktionalität kein Tool vorhanden sein, ist es möglich, einzelne Prozessschritte mit Python selbst zu erstellen. Schliesslich bieten beide Systeme eine graphische Oberfläche zur Erstellung der Workflows (ArcGIS: *Model Builder*, QGIS: *Processing modeler*).

Weil immer mehr Dienste über API angeboten werden, ist es auch denkbar, in einzelnen Schritten solche API zu konsumieren. Wenn man sich dabei innerhalb einer Plattform (z.B. Esri ArcGIS) bewegt, ist das einfach möglich, ansonsten ist mit (Programmier-)Aufwänden für die Einbindung zu rechnen. Eine Reihe aktueller GIS APIs sind im Artikel von Chow und Yuan (2019) beschrieben.

Ein Spezialfall des Processing wird als **ETL** bezeichnet, kurz für Extract-Transform-Load. Es sind dies die typischen Schritte der Datenvorbereitung: Daten aus ihren jeweiligen Quellen extrahieren, so transformieren, dass sie für die ihnen zugedachte Aufgabe/Analyse geeignet sind, und sie dann speichern (in eine Datenbank oder ein Data Warehouse laden). Der Platzhirsch auf dem Feld der räumlichen ETL ist die kommerzielle Software FME (www.safe.com/fme), deren besondere Stärke die Kenntnis von unzähligen Daten- und Dateiformaten ist. Ein Open Source Spatial ETL Tool ist GeoKettle, dieses scheint aber gerade nicht mehr unterhalten zu sein.

Die Erstellung eines Tools für Geoprocessing/Processing/FME/etc. ist eine attraktive Möglichkeit, um eine Analyse in die «Produktion» zu überführen, sie also einfach verwendbar zu machen. Voraussetzung sind allerdings Programmierkenntnisse, die möglicherweise über

[16] Geoprocessing in ArcGIS: pro.arcgis.com/en/pro-app/latest/help/analysis/geoprocessing
[17] QGIS Processing Framework: docs.qgis.org/3.22/en/docs/user_manual/processing

die für die Analyse notwendigen hinausgehen, Kenntnis der jeweiligen Frameworks und deren APIs und SDKs.

7 Standards

«Das Schöne an Standards ist, dass es so viele zur Auswahl gibt», so ein Bonmot, dessen Urheber mir unbekannt ist. Stand heute (2022) ist jedoch anzumerken, dass es im Bereich der räumlichen Daten gar nicht so viele Standards gibt und somit jene, die es gibt, einen echten Wert darstellen: Standards helfen beim Austausch von Daten (wenn diese *kompatibel* sind) und bei der Nutzung von Diensten (wenn diese *interoperabel* sind).

Hinter Standards stehen Organisationen, die diese Standards entwickeln und publizieren, allenfalls gegen Gebühr, und in deren Mitglieder ein Interesse an den Standards haben. Am bekanntesten ist wohl die ISO, die International Organization for Standardization, für unseren Bereich am bedeutsamsten sind jedoch:

- Open Geospatial Consortium (OGC), entwickelt GIS Standards
- World Wide Web Consortium (W3C), entwickelt die Standards des WWW
- Internet Engineering Task Force (IETF), entwickelt die Internet Standards

Erfolgreich ist jedoch derjenige Standard, der in der Breite Anwendung findet. Das hat oft weniger mit Standardisierungs-Gremien und -Organisationen zu tun als mit Marktmacht und mit glücklichem Timing. Esri's **Shapefile** Format ist kein formeller Standard, aber seit Jahren ein *de facto* Standard, gegen den neuere (und bessere) Alternativen (z.B. GeoPackage) einen schweren Stand haben.

Die beiden Standards mit der wohl grössten praktischen Bedeutung für die räumliche Datenanalyse sind die Simple Features Spezifikation des OGC und die Katalogisierung räumlicher Bezugssysteme der EPSG. Letzteres ist nicht ein Standard im eigentlichen Sinne, die Zuordnung von numerischen Codes zu Koordinatensystemen ist aber über die ganze GIS-Branche und darüber hinaus von grösstem Nutzen. Nicht übersehen werden sollten Basis-Standards im Web-Bereich wie HTTP und UTF-8, sowie die Serialisierungs-Standards XML und JSON.

EPSG Geodetic Parameter Dataset

Ein Katalog räumlicher Bezugssysteme wurde 1985 von der der European Petroleum Survey Group (**EPSG**) erstellt, wurde 1993 veröffentlicht, und wird gegenwärtig vom Geomatics Committee der International Association of Oil & Gas Producers (IOGP) gepflegt[18]. Neben Koordinatensystemen finden sich da ebenfalls Ellipsoide, geodätischen Daten, und Masseinheiten, so dass der Datensatz als "EPSG Geodetic Parameter Dataset" bezeichnet wird. Die den verwalteten Entitäten zugewiesenen Codes werden in der ganzen GIS-Industrie und darüber hinaus verwendet. Die Tabelle zeigt vier in der Schweiz häufig verwendete Koordinatensysteme mit den ihnen zugewiesenen Nummern (SRID = Spatial Reference ID). Die offizielle Web-Präsenz des EPSG Katalogs findet sich unter epsg.org, wo auch eine Suche über den ganzen Datensatz und Dokumentation angeboten wird. Ein praktisches Online-Nachschlagewerk über den EPSG Katalog findet sich unter der URL epsg.io; direkt zu den Informationen zu einer SRID, z.B. 4326, kommt man mit der URL epsg.io/4326. Eine weitere Seite mit Informationen zu räumlichen Referenzsystemen ist spatialreference.org – hier können die Koordinatensystem-Definitionen in verschiedenen Formaten (OGC WKT, Proj4, *.prj*-Datei, etc.) kopiert oder heruntergeladen werden.

[18] Quelle: en.wikipedia.org/wiki/EPSG_Geodetic_Parameter_Dataset (letzter Zugriff am 10.9.2022)

SRID	Bezeichnung	Anwendungsgebiet
4326	WGS 84	World Geodetic System 1984: geographische Koordinaten, von GPS verwendet
21781	CH1903 / LV03	Schweizerische Landesvermessung 1903: die traditionellen projizierte Koordinaten der Schweiz (Bern bei 600'000/200'000)
2056	CH1903+ / LV95	Schweizerische Landesvermessung 1905: die neuen projizierten Koordinaten der Schweiz (Bern bei 2'600'000/1'200'000)
3857	WGS 84 / Pseudo-Mercator	Web Mercator: Google Maps, Open Street Map und andere; von echten Geodäten nicht anerkannt, weil sphärische Behandlung von ellipsoidischen (WGS84) Koordinaten

Open Geospatial Consortium

Das *Open Geospatial Consortium* (OGC) wurde 1994 unter dem Namen *Open GIS Consortium* als gemeinnützige Organisation gegründet mit dem Ziel, Standards zu entwickelt für die Interoperabilität in der raumbezogenen Informationsverarbeitung. Das Konsortium hat rund 500 Mitglieder aus Regierungsorganisationen, Universitäten (z.B. ETH Zürich) und privaten Unternehmen (z.B. Esri, Google, Microsoft). Die entwickelten Standards sind frei verfügbar und über die Webseite der OGC einfach zugänglich. Zu den wichtigsten Standards des OGC gehören sicher der Simple Features Access (siehe unten) und die OpenGIS Web Service Standards.

Simple Features

Ein Standard, entwickelt von der OGC, über die Repräsentation von Geometrien als ein Objektmodell (Part 1 = SFA-CA Common Architecture[19]) und eine Umsetzung in SQL (Part 2 = SFA-SQL[20]). Seit 2004 auch ISO Standard 19125. Der Simple Features Standard umfasst im Wesentlichen die folgenden vier Bereiche:

- Ein **Objektmodell:** dieses definiert in objektorientierter Manier den Typen Geometry mit Subtypen Point, LineString, Polygon, GeometryCollection und weiteren (Abbildung 5); sowie eine Menge von Methoden (Funktionen) für geometrische Operationen mit diesen Typen. Es sind dies die Funktionen, die PostGIS unter den Namen `ST_Foo()` zur Verfügung stellt, denn PostGIS ist konform zum Simple Features Standard, und analog bei Esri's ST_Geometry.
- Ein **Geometriemodell:** Simple Features sind 2-dimensional, zwischen den Stützpunkten wird linear interpoliert, und topologische Beziehungen (*intersects*, *touches*, *overlaps*, etc.) sind im Sinne einer Point Set Topology strikt definiert. Dieser letzte Aspekt ist im Standard im Abschnitt 6.1.15 der *Common Architecture* erläutert und basiert auf der einschlägigen Forschung zum Thema (siehe Egenhofer & Franzosa 1991).
- Mit **WKT** (Well Known Text) und **WKB** (Well Known Binary) definiert der Standard zwei Serialisierungsformate für den Austausch, das eine menschenlesbar, das andere binär. PostGIS erweitert WKT und WKB um auch 3- und 4-dimensionale Geometrien (mit Z- und/oder W-Koordinate) repräsentieren zu können.
- SFSQL verlangt die beiden Tabellen `SPATIAL_REF_SYS` und `GEOMETRY_COLUMNS` (PostGIS ergänzt noch die Tabelle `GEOGRAPHY_COLUMNS`) mit Informationen zu den

[19] Simple Feature Access, Part 1: Common Architecture: www.ogc.org/standards/sfa
[20] Simple Feature Access, Part 2: SQL Option: www.ogc.org/standards/sfs

räumlichen Bezugssystemen (Primärschlüssel SRID) und zu den Geometrie-Spalten in allen Tabellen des Schemas (siehe Abbildung 6).

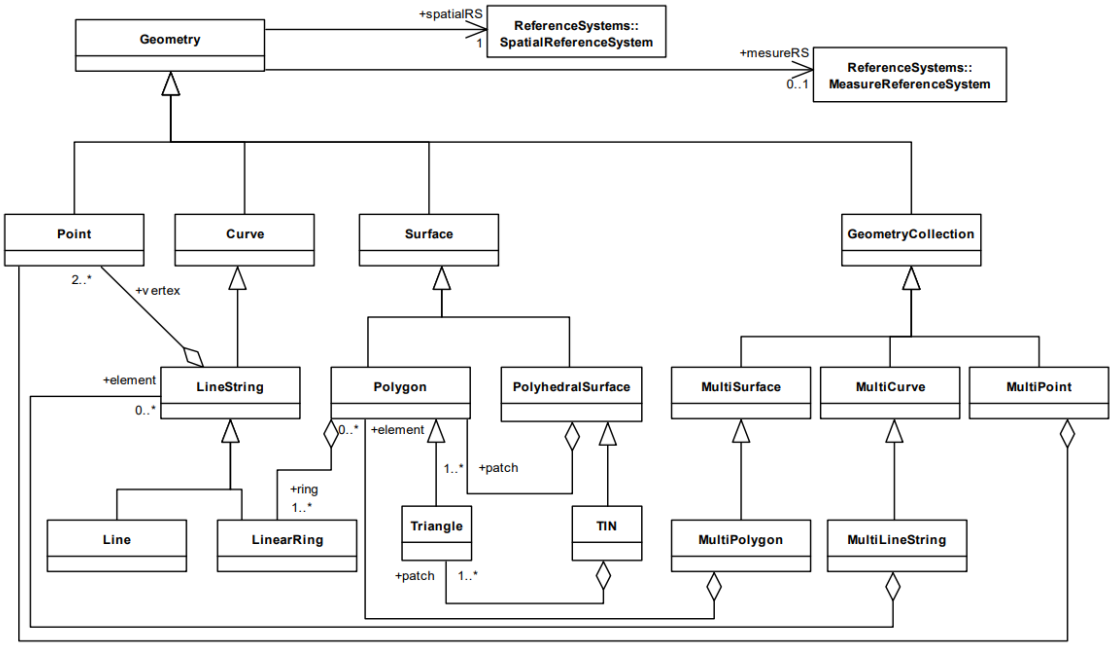
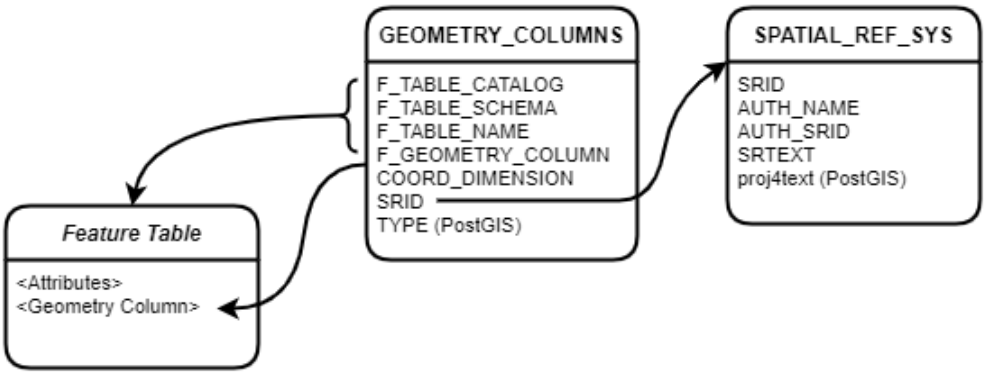


Abbildung 5. Das Simple Features Objektmodell



Quelle: nach Abbildung 3 im Abschnitt 6.2.1 in: *OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*, Version 1.2.1, Open Geospatial Consortium, Inc., 2010.

Abbildung 6. Metatabellen nach Simple Features SQL-Option

Das Geometriemodell ist wichtig, um z.B. die folgende Frage beantworten zu können: gegeben zwei Linien, A = LINESTRING (0 0, 2 0) und B = LINESTRING (0 0, 1 0, 2 0), was ist das Ergebnis von `ST_Equals(A,B)`? Antwort: True, weil beide WKT-Repräsentationen das gleich Point Set in der Ebene repräsentieren. PostGIS verfügt über das Prädikat `ST_OrderingEqual(A,B)`, welches im vorliegenden Fall `False` ist, weil sich die beiden Geometrien in ihren Stützpunkten unterscheiden.

Notiz am Rande: Die Validierungsregeln von OGC SFS und jene, die Esri-Software intern verwendet,[21] unterscheiden sich: Ringe, die sich berühren (*touch*), sind für OGC SFS gültig, für

[21] Siehe «Geometry Validation» in der ArcMap Hilfe: desktop.arcgis.com/en/arcmap/latest/manage-data/using-sql-with-gdbs/geometry-validation.htm

Esri nicht (**Abbildung 7**). Entsprechend kann PostGIS `ST_MakeValid()` Geometrien erzeugen, die für Esri ungültig sind (und umgekehrt).[22] Eine Linie (Polyline), die sich selbst schneidet, ist gültig. Das ist gut so, denkt man etwa an den Kreisviadukt von Brusio (**Abbildung 8**) und ähnliche Ingenieurskünste.

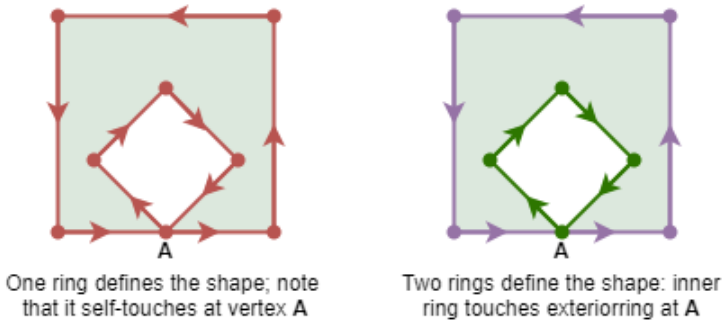


Abbildung 7. Zwei Arten das «Gipfeli-Polygon» (boundary loop) zu repräsentieren; keine ist besser als die andere, je nach Validierungsregeln ist aber nur die eine oder die andere korrekt.

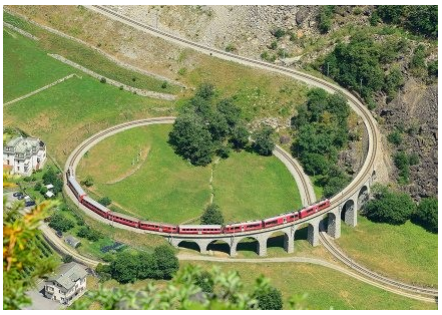


Abbildung 8. Kreisviadukt von Brusio

WMS, WMTS, WFS WCS

Die OGC Web Standards werden kollektiv als OWS (OGC Web Services) bezeichnet und spezifizieren die Schnittstelle von Diensten für den Zugriff auf drei klassische Repräsentationen geographischer Information: die gerenderte Karte (WMS, WMTS), diskrete Features (WFS), und Felder (coverages, WCF). Alle OWS unterstützen einen GetCapabilities-Request zur Abfrage von Metadaten über den Dienst.

Ein **Web Map Service** (WMS) liefert ein fertig gerendertes Kartenbild. Parameter sind: Layer (ein oder mehrere), Koordinatensystem, Bounding Box (räumliche Begrenzung), Styling-Informationen, die gewünschte Bildgrösse (Breite und Höhe) und das Bildformat (z.B. PNG). Beispiel für einen Request:

https://ows.terrestris.de/osm/service?REQUEST=GetMap&SERVICE=WMS&VERSION=1.3.0&LAYERS=OSM-WMS&STYLES=&CRS=EPSG:4326&BBOX=51.49451,-0.11377,51.53267,-0.06971&WIDTH=400&HEIGHT=300&FORMAT=image/png&TRANSPARENT=TRUE

Ein **Web Map Tile Service** (WMTS) ist eine Spielart eines WMS: er tauscht die Flexibilität eines WMS gegen Skalierbarkeit und gute Performance, indem nur vorbestimmte Kartenausschnitte, genannt Kacheln (*tiles*), abgefragt werden können (und nur *ein* Layer). Diese Kacheln können einfach vorgerechnet und gespeichert (*cached*) werden. Die Kacheln bilden eine Art Pyramide:

[22] Quelle: https://postgis.net/workshops/postgis-intro/validity.html (letzter Zugriff am 10.9.2022)

zuunterst repräsentieren viele Kacheln die Karte in einem grossen Massstab, weiter oben repräsentieren immer weniger Kacheln die gleiche Karte in immer kleinerem Massstab. Im WMTS-Standard wird von *tile matrix set* (nicht von Pyramide) gesprochen.

In Ermangelung eines offenen WMTS folgt als Beispiel die Abfrage einer Kachel von Open Street Map; man beachte die drei Parameter in der URL: Zoom-Stufe, Spalte, Zeile.

http://a.tile.openstreetmap.org/15/9798/14664.png

Ein **Web Feature Service** (WFS) erlaubt die Abfrage und Bearbeitung von Features.

Ein **Web Coverage Service** (WCS) erlaubt die Abfrage von Feldern, z.B. von Höhenwerten aus einem Höhenmodell, oder von Pixelwerten aus einem Satellitenbild. Die Ergebnisse werden typischerweise als Rasterdaten geliefert, aber auch andere Repräsentationen sind möglich (z.B. Triangulation). Beispiel für einen Request:

http://ows.eox.at/cite/mapserver?service=wcs&version=2.0.1&request=getcoverage&coverageid=MER_FRS_1PNUPA20090701_124435_000005122080_00224_38354_6861_RGB

Wegen der vielfältigen Möglichkeiten sind vorgängig unbedingt die Metadaten der Coverage mit der DescribeCoverage-Operation zu prüfen:

http://ows.eox.at/cite/mapserver?service=wcs&version=2.0.1&request=DescribeCoverage&coverageid=MER_FRS_1PNUPA20090701_124435_000005122080_00224_38354_6861_RGB

Weitere OGC Standards

Neben den schon erwähnten Standards, entwickelt und pflegt das OGC noch viele weitere Standards, auf die hier nicht eingegangen wird. Einige wenige seien jedoch noch erwähnt, um einen Hinweis zu haben, wo bei Bedarf offizielle Information gefunden werden kann: GML, GeoPackage, KML, CSW. Die offizielle Web-Präsenz findet sich unter www.ogc.org und unter opengeospatial.github.io/e-learning findet sich eine Einführung zu vielen der OGC Standards.

Esri REST API

Esri dokumentiert und publiziert die REST- Schnittstelle zu ihren Server-Produkten unter dem Namen «Esri REST API». Dies ist kein formeller Standard, aber durch die Marktmacht von Esri zumindest eine interessante Schnittstelle (und war vor etwa 10 Jahren mal als OGC-Standard angedacht). Informationen: developers.arcgis.com/rest/services-reference

INTERLIS

Ein Standard für die Speicherung und den Transport von räumlichen Daten in der Schweiz, seit 2008 gesetzlich vorgeschrieben, basiert auf Modellen, relativ komplex und im Umgang anspruchsvoll, Software-Unterstützung limitiert, siehe www.interlis.ch für Details.

XML, JSON, HTTP, etc.

XML und JSON sind wichtige Basis-Standards für die Serialisierung von Information, weil von anderen Standards verwendet werden (z.B. GML und GeoJSON). Die ganze Web-Technologie und alle REST-Schnittstellen bauen auf HTTP auf. Im Cloud-Bereich (und etwa bei Condas *environment.yaml*) wird oft mit YAML gearbeitet. Kenntnisse dieser Standards sind nützlich, können hier aber nicht weiter behandelt werden. Eine Internetsuche wird schnell zu geeigneten Informationen führen.

ASCII, Unicode, UTF-8

Bei Text Dateien (*plain text*) stellt sich die Frage, wie die Zeichen des Textes den Bytes der Datei zugeordnet werden. Diese Abbildung geschieht in zwei Schritten: zunächst eine 1:1

Abbildung von den Zeichen (characters) zu ganzen Zahlen, dann von diesen Zahlen auf die Bytes der Datei. Die erste Abbildung wird oft als **Charset** (character set) bezeichnet, die zweite als **Encoding**. Gleiches gilt auch für Textfelder in Datenbanken und Text in ansonsten binären formatierten Dateien.

ASCII, der American Standard Code for Information Interchange, ordnet die lateinischen Gross- und Kleinbuchstaben, die zehn Ziffern, und einige Satz- und Sonderzeichen den Zahlen 0 bis 127 zu; diese Zahlen lassen sich in je einem Byte speichern, womit ASCII sowohl *Charset* als auch *Encoding* ist. ASCII funktioniert für die englische Sprache, doch schon für die deutsche Sprache fehlen wichtige Buchstaben, ganz zu schweigen von exotischeren Sprachen. Diesem Problem nimmt sich der **Unicode** Standard (unicode.org) an: er ordnet Tausenden von Schriftzeichen eine Zahl zu, genannt *code point*. Weil ein Byte nur 256 verschiedene Zahlen repräsentieren kann, muss bei Unicode ein Encoding verwendet werden, das eine Zahl auf mehrere Bytes abbilden kann. Im Web-Bereich wird heutzutage dazu fast ausschliesslich **UTF-8** verwendet. Die Details gehen hier zu weit, es sei Ihnen aber folgende Empfehlung mitgegeben: wenn Sie eine Wahl haben, wählen Sie UTF-8.

Dates and Times

Anmerkung am Rande und ganz unabhängig von räumlicher Datenanalyse: für die Notation von Datum und Zeit gibt es den Standard **ISO 8601**, welcher im Wesentlichen verlangt, ein Datum in der Form *yyyy-mm-dd* zu schreiben, eine Zeit in der Form *HH*:*MM*:*SS*, ein Datum mit (Lokal-)Zeit in der Form *yyyy-mm-dd*T*HH*:*MM*:*SS*, und ein Datum mit Zeit in UTC in der Form *yyyy-mm-dd*T*HH*:*MM*:*SS*Z. Ein Ersatz des T durch einen Leerschlag sei verziehen. Details sind im Internet zu finden.[23] Diese Konventionen sind sehr zu empfehlen, weil sie Klarheit schaffen und die lexikographische Sortierung automatisch die zeitliche Ordnung wiedergibt.

[23] Details auf Wikipedia (en.wikipedia.org/wiki/ISO_8601), der Standardtext von der ISO ist nur käuflich zu erwerben.

8 Isolation

Der Datenanalyst arbeitet an verschiedenen Projekten, oft gleichzeitig, meist auf einem Gerät, dem eigenen Notebook, allenfalls auf einer grösseren Infrastruktur. Jedes Projekt hat spezifische und oft inkompatible Anforderungen an die Infrastruktur. Durch eine sinnvolle Isolation der Projekte können Probleme vermieden und zugleich die Nachvollziehbarkeit der Projekte gewährleistet werden. Wir betrachten dazu Isolation in drei Bereichen: Python-Umgebungen, Container, und Datenbanken.

Python-Umgebungen

Die globale (System-) Installation von Python ist bequem, weil schon vorhanden, aber problematisch: weil sie systemweit gilt, sind Paketkonflikte vorprogrammiert und Systemupdates machen eine schön eingerichtete Paketlandschaft zunichte. Eine Lösung besteht darin, mit virtuellen Umgebungen zu arbeiten.

Der klassische Weg verwendet den virtualenv-Befehl, der nach einer globalen Installation mit `pip install virtualenv` zur Verfügung steht; unter Windows muss noch der PATH angepasst werden! Die Verwendung ist dann wie folgt (Beispiel für Windows; Linux ähnlich):

```
cd MyProject
virtualenv --python python3 myenv
.\myenv\Scripts\activate
```

Das Skript `activate` modifiziert im Wesentlichen den PATH so, dass zuerst die Programme (und damit python.exe) in der virtuellen Umgebung gefunden werden.[24] Pakete werden nun relativ zu dieser virtuellen Umgebung installiert, ohne die globale (systemweite) Installation zu berühren. Um die Reproduzierbarkeit zu fördern, sollte unbedingt eine Liste der installierten Pakete erstellt werden, was wie folgt geht:

```
pip freeze > requirements.txt
```

Unter Verwendung von pip und *requirements.txt* können diese Pakete an einem anderen Ort (wiederum in einer virtuellen Umgebung!) einfach installiert werden mittels

```
pip install -r requirements.txt
```

Der Dateiname *requirements.txt* ist eine dringend empfohlene Konvention. Das Skript `deactivate` ändert den Pfad auf den ursprünglichen Wert zurück und verlässt damit die virtuelle Umgebung.

Mit dem Paketmanager conda (Teil von Anaconda und Miniconda[25]) steht eine bequemere Lösung zur Verfügung, weil conda sowohl Pakete als auch virtuelle Umgebungen beherrscht. Conda kann mit einer Datei *environment.yaml* arbeiten, welche eine virtuelle Umgebung beschreibt: Name, zu installierende Pakete, zu verwendende Conda Channels. Gängige Befehle:

```
conda env list                # list Conda environments
conda env create -f myenv.yaml # create env and inst packages
conda env update -f myenv.yaml # update an environment (outside env!)
```

[24] Experiment: `where python` (Windows) bzw. `which python` (Linux) zeigt, welches Python ausgeführt wird; analog für pip.

[25] Miniconda ist ein minimales Subset der Anaconda-Distribution, welches im Wesentlichen nur den conda Paket- und Umgebungsmanager beinhaltet.

```
conda activate myenv           # activate environment
conda list                     # list packages in current env
conda deactivate               # deactivate enviornment
conda env remove --n myenv     # delete environment
conda install -n myenv package # install package into myenv
```

Wenn Pakete mittels `conda install` installiert werden, ist *environment.yaml* davon nicht berührt, sollte aber unbedingt nachgeführt werden, so dass sie eine aktuelle Dokumentation der Umgebung darstellt.

Container

Container bieten Isolation auf einer tieferen Ebene, indem sie das Betriebssystem virtualisieren. Dadurch unterscheiden sie sich von der älteren Technologie der virtuellen Maschinen, welche die Hardware virtualisieren. Abbildung 9 stellt die beiden Konzepte einander gegenüber. Wesentlich ist, dass sich alle Container denselben Betriebssystem-Kern teilen, jedoch alle anderen Ressourcen (v.a. das File System) streng isolieren. Ein Container ist im Vergleich zu einer virtuellen Maschine «light weight», braucht also weniger physische Ressourcen (Rechenleistung, Speicher), weniger Software-Ressourcen (insbesondere kein vollständiges Betriebssystem), ist schneller gestartet, und vor allem ist er einfacher zu erstellen. Anders als eine virtuelle Python-Umgebung kann ein Container relativ einfach von einem System auf ein anderes verschoben werden und hilft damit nicht nur bei der Isolation, sondern auch bei der Reproduzierbarkeit.

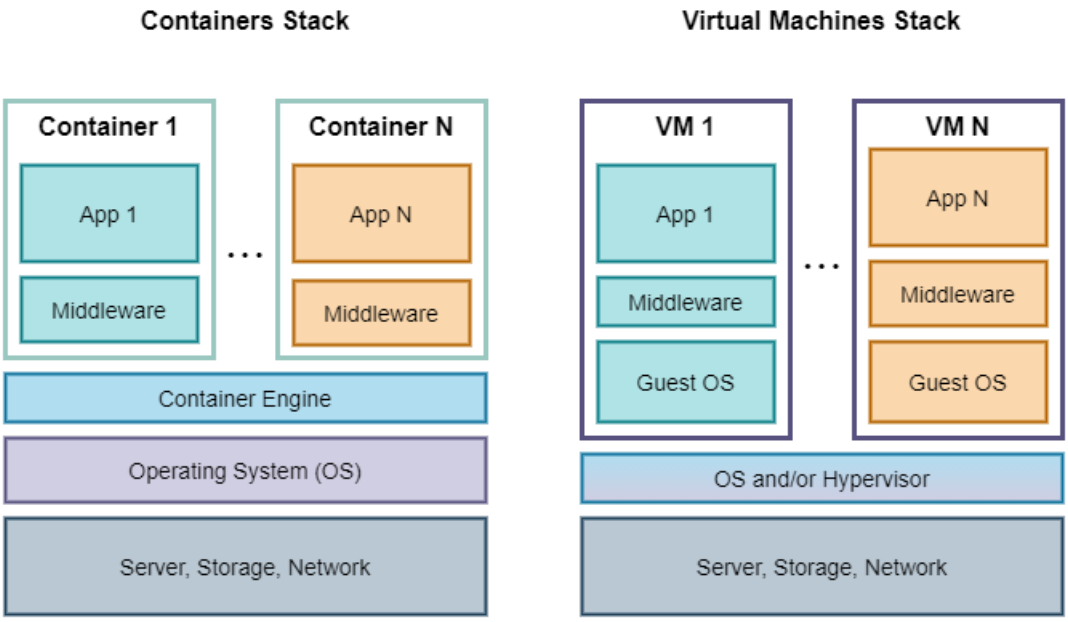


Abbildung 9. Containers vs. Virtual Machines

Eine vollständige Isolation ist jedoch nie erwünscht, weil sinnlos: ein Programm ohne Ein- und Ausgabe ist für den aussenstehenden Menschen wertlos. Container-Technologien erlauben kontrollierte Lücken in der Isolation. Davon gibt es im Wesentlichen zwei Arten: Interaktion mit einem Speichersystem und Interaktion mit dem Netzwerk. Beispiel: Es ist möglich und sinnvoll Python und alle für eine Analyse benötigten Pakete in einen Container zu packen; die verwendeten Daten könnten auch mitgepackt werden, es ist aber flexibler, diese von aussen in den Container einzublenden; das für die Analyse verwendete Jupyter Notebook ist technisch

eine Web-Anwendung, es muss also ein Zugriff über das Netzwerk von aussen in den Container ermöglicht werden.

Die momentan dominante Container-Technologie ist Docker (www.docker.com). Ein Docker Container basiert immer auf einem **Container Image**, welches wiederum aus allen Dateien besteht, die im Container verfügbar sind. Ein Container Image wird erstellt, indem Docker eine Beschreibung des Images, bestehend aus einzelnen Schritten, abarbeitet. Diese Beschreibung wird als *Dockerfile* bezeichnet und ist eine einfache Textdatei. Der erste Schritt in einem *Dockerfile* referenziert ein Basis-Image, welches oft ein kleines freies Betriebssystem enthält,[26] jeder weitere Schritt fügt dem Basis-Image einen Layer hinzu (ergänzt es um weitere oder geänderte Dateien), z.B. einen Python Interpreter und Pakete. Weiter kann ein *Dockerfile* Informationen über zu öffnende Ports (Netzwerkzugänge) und einzublendende Verzeichnisse aus dem Dateisystem enthalten. Wenn ein Container ausgeführt wird, kommt über sein Image ein weiterer Layer; dieser oberste Layer (und nur dieser) ist schreibbar; alle anderen sind nur lesbar (**Abbildung 10**). Wenn der Container gestoppt wird, bleibt dieser oberste Layer bestehen, so dass der Container bei einem neuen Start in der gleichen Dateiumgebung weitermachen kann. Wird der Container gelöscht, wird dieser oberste Layer gelöscht. Die tieferen Layer des Images bleiben bestehen.

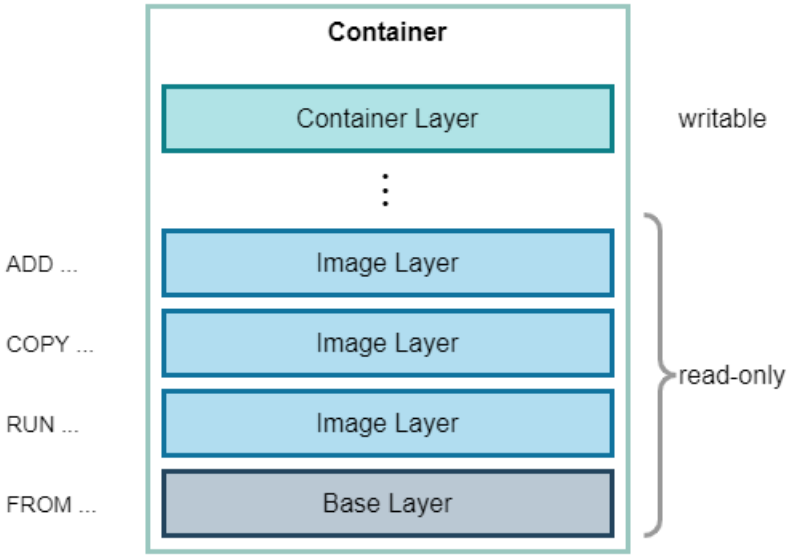


Abbildung 10. Container Image Layers; links mögliche Instruktionen aus dem Dockerfile

Eine nützliche Komponente im Umgang mit Container Images ist eine **Registry** solcher Images. Eine solche kann privat im eigenen Unternehmen existieren, oder aber auf einer öffentlichen Plattform. Die grösste solche Registry aktuell ist Docker Hub (hub.docker.com), wo tausende Container Images zum Download zur Verfügung stehen. Diese Images stammen von grossen Institutionen wie auch von Privaten Anwendern. Oft bauen die Entwickler von Software gleich auch ein Container Images, welches die Einsatzfertige Software enthält. Ein Beispiel in unserem Kontext ist etwa PostgreSQL.[27] Weil von einer Software oft mehrere Container Images vorliegen, ist die Auswahl nicht einfach. Docker selbst kuratiert Container Images für

[26] Nur den sogenannten User Space, nicht den Kern, denn dieser wird bei der Container Technologie ja virtualisiert, also mit allen anderen Containern geteilt.
[27] Beispiel einer Suche auf Docker Hub: hub.docker.com/search?q=postgresql

die gängigsten Open Source Werkzeuge, was als Gütesiegel gelten kann und Orientierung bringt.

Für praktische Übungen mit PostGIS auf PostgreSQL (siehe Kapitel über Datenbanken) könnte z.B. das offizielle PostGIS Docker Image[28] verwendet werden und es bliebe einem die Installation von PostgreSQL und PostGIS erspart (natürlich müsste zuerst Docker installiert werden, ganz ohne Investition geht es nicht).

Weil die Kombination aus Docker und Jupyter beliebt ist und oft verwendet wird, gibt es dafür unter dem Namen Jupyter-Docker-Stacks schon fertige Container Images. Als Einstieg dient die offizielle Dokumentation auf jupyter-docker-stacks.readthedocs.io. Von Container Images können weitere Images abgeleitet werden. Ein gutes Beispiel dafür ist **GDS_env**, eine Umgebung für Geographic Data Science von Dani Arribas-Bel unter der URL darribas.org/gds_env. Diese Umgebung enthält viele für die räumliche Analyse notwendige Python-Pakete fertig installiert und wird in universitären Kursen zur geographischen Datenanalyse eingesetzt. Bei technischem Interesse empfiehlt sich auch ein Blick in das GitHub Repository github.com/darribas/gds_env

Die Verwendung dieser vorgefertigten Container Images sei der praktischen Beübung sehr empfohlen! Es ist dies die Grundlage für eigene Anpassungen an diesen Images für eigene Analysen. Docker wird am besten über die mitgelieferte Kommandozeilen-Programm `docker` bedient. Die wichtigsten Befehle für die Anwendung sind:

```
docker pull IMAGE              # pull image from container registry
docker run IMAGE               # run default command in new container
docker exec CONTAINER COMMAND  # run command in running container
docker stop CONTAINER          # stop running container
docker container rm CONTAINER  # delete container
docker image rm IMAGE          # delete image (reclaim disk space)
```

Dabei sind die gross geschriebenen Worte Platzhalter für die effektiven Namen von Images, Container und Befehlen. Alle Docker-Befehle verfügen über eine Vielzahl von Optionen, wofür auf die entsprechende Dokumentation verwiesen sei; beachten Sie unbedingt bei `docker run` die Optionen `-p` (publish ports) und `-v` (mount volume). Für einige Konfusion sorgen die Image-Namen, welche aus diversen Teilen bestehen; alle ausser dem repository sind optional:

Muster: `registry/user/repository:tag`

- registry – Server-Name der Container Registry (Standard: docker.io)
- user – Eigner des Images, Benutzer oder Organisation (Standard: library)
- repository – der eigentliche Name des Images
- tag – Version und Variante des Images (Standard: latest)

Beispiel: `docker.io/postgis/postgis:latest`
dies könnte abgekürzt werden zu `postgis/postgis`

Schliesslich sei noch erwähnt, dass Docker nicht die einzige Realisierung der Container-Technologie ist, wohl aber die aktuell am weitesten verbreitete.[29]

[28] PostGIS Container Image auf Docker Hub: hub.docker.com/r/postgis/postgis
[29] Alternativen zu Docker: www.educba.com/docker-alternatives (letzter Zugriff am 12.9.2022)

Datenbanken

Auch bei den Daten ist Isolation ein Thema, einerseits bezüglich Zugriffsberechtigung (hier nicht thematisiert), andererseits damit in Mehrbenutzersystemen die Anwender unabhängig voneinander arbeiten können. Bei lesendem Zugriff sollten hier keine Probleme entstehen, sobald jedoch Daten erzeugt werden, können sich mehrere Benutzer in die Quere kommen. Daher kennen Relationale Datenbanken und SQL das Konzept des **Schemas** (*schema*): dieses stellt eine Art Namensraum dar und hilft die von unterschiedlichen Anwendern oder Applikationen erstellten Daten auseinanderzuhalten. Ein Schema qualifiziert den Namen einer Tabelle: *SchemaName*.*TableName*. Ein unqualifizierter Tabellenname wird entlang eines Suchpfades gesucht. Ein Anwender hat i.a. ein Default Schema. Die folgenden SQL-Befehle demonstrieren die Verwendung von Schemata in PostgreSQL:

```
CREATE SCHEMA A;
CREATE TABLE T(…);     -- wird im default Schema “public” erzeugt
CREATE TABLE A.T(…);
SELECT * FORM T;       -- verwendet Tabelle public.T
SET search_path = A, public;
SELECT * FROM T;       -- verwendet Tabelle A.T
```

> **Achtung:** «Schema» wird bei Datenbanken in zwei unterschiedlichen Bedeutungen verwendet: zum einen wie eben erwähnt als Mechanismus um Namenskollisionen zu vermeiden, zum anderen zur Bezeichnung der Struktur einer Datenbank (die Tabellen mit ihren Kolonnen und deren Datentypen).

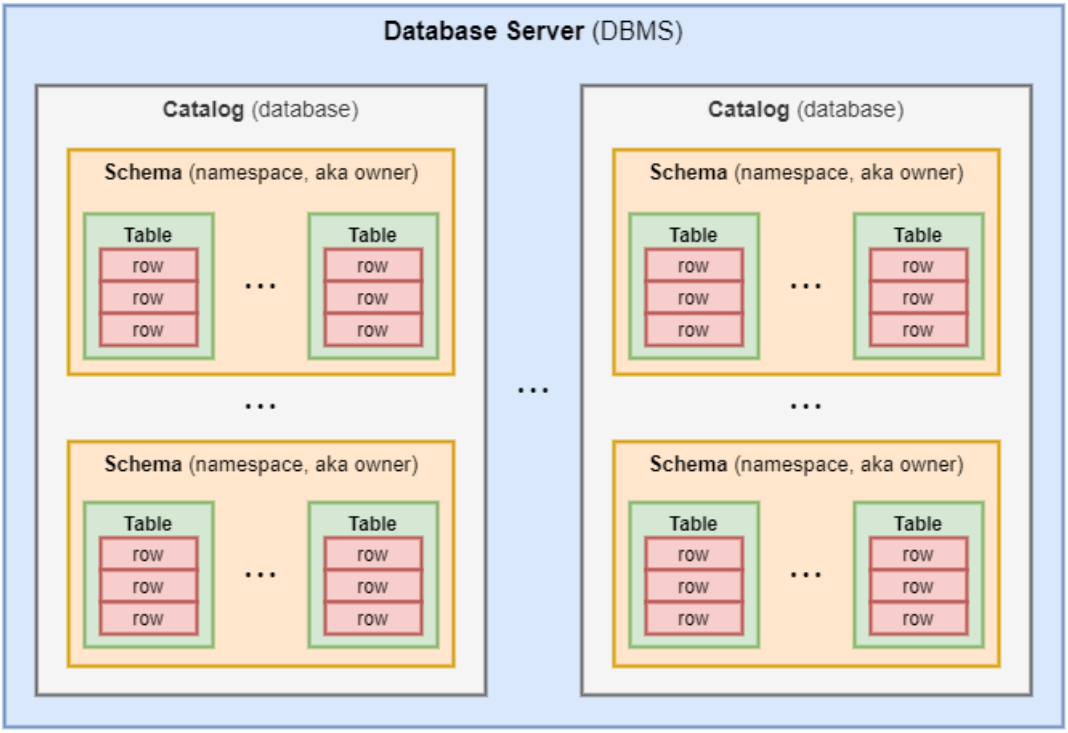
Ein weiteres Konzept zur Isolation von SQL ist der **Catalog**, welcher eine benannte Sammlung von Schemas darstellt; in der Praxis ist Catalog zumeist Synonym zu Datenbank. Vorsicht ist geboten bei der Syntax zur Benennung von Tabellen: oft ist es `Catalog.Schema.Table` (so etwa bei SQL-Server und PostgreSQL), bei Oracle jedoch ist es `Schema.Table@Catalog`, und in beiden Fällen ist Catalog = Datenbank und wird meist weggelassen, weil durch den Kontext gegeben. Das Schema wird gelegentlich auch als Owner bezeichnet.[30]

Zusammengefasst implementieren Datenbanken nach dem SQL-Standard dreistufige Tabellennamen, um eine sinnvolle Isolation zu gewährleisten (um Namenskollisionen zu vermeiden und die Rechtevergabe zu organisieren): Catalog, Schema, Table. Zur Illustration:

```
SELECT * FROM information_schema.tables;  -- in jeder SQL-Datenbank
SELECT * FROM geometry_columns;  -- räumlichen Datenbank nach SFSQL
```

In den Ergebnissen beider Abfragen ist ersichtlich, wie Tabellen durch dreistufige Namen angesprochen werden. In den Abfragen selbst werden die Tabellen nur durch Schema und Name adressiert, der Catalog ist implizit in der Datenbankverbindung. Das verwendete `information_schema` ist Teil vom SQL-Standard und enthält Informationen zu den in der Datenbank vorhandenen Schemata, Tabellen, und sonstigen Konstruktionen. **Abbildung 11** illustriert die Isolationen in einer relationalen Datenbank.

[30] In ArcObjects findet sich die Funktion `ISQLSyntax.ParseTableName(`*name*`, out` *dbName*`, out` *ownerName*`, out` *tableName*`)`, welche aus einem qualifizierten Tabellennamen *name* seine Bestandteile *dbName* (catalog), *ownerName* (schema) und *tableName* extrahiert, dies unter Berücksichtigung der von der aktuellen Datenbank verwendeten Syntax.



Quelle: nach der Abbildung in https://stackoverflow.com/questions/7022755, verändert

Abbildung 11. Isolation in einer relationalen Datenbank (Catalog, Schema, Table)

9 Automation

Automatisierung ist ein Kernthema der ganzen Informatik. Auch im Bereich der räumlichen Datenanalyse und ihrer Infrastruktur sollten wir die Automatisierung anstreben. Das steht dem oft explorativen Charakter der Datenanalyse gegenüber, bringt aber den wesentlichen Vorteil der Wiederholbarkeit und Reproduzierbarkeit und somit einer Grundanforderung wissenschaftlichen Arbeitens: meine Ergebnisse sind nur so gut, wie sie unabhängig überprüfbar sind, und das geht nur, wenn die Analyse reproduzierbar ist, und das wiederum geht nur, wenn der verwendete Software-Stack wiederholt zur Verfügung steht. Das Ziel der Automatisierung in unserem Kontext muss also sein, dass die infrastrukturelle Umgebung einer Analyse mit vertretbarem Aufwand zu einem späteren Zeitpunkt wieder bereitgestellt werden kann, auch durch andere Personen als der ursprüngliche Autor der Analyse, und idealerweise auch an anderen Standorten mit anderen Umgebungen.

Im einfachsten Fall kann Automatisierung durch Dokumentation erfolgen: eine Anleitung für jemand anderes, einschliesslich mich zu einem späteren Zeitpunkt! Besser jedoch ist es, die Automatisierung als ausführbare Dokumentation zu betrachten.

Computational Notebooks sind ein erster Schritt: die Analyse selbst ist notiert, dokumentiert, und mit ihren Ergebnissen gespeichert. Für die Reproduzierbarkeit muss aber auch die Infrastruktur der Analyse, hier also etwa alle verwendeten Python-Pakete in der korrekten Version, wieder erstellt werden können.

Im Bereich der Infrastruktur sind viele Werkzeuge als Kommandozeilen-Tool realisiert (z.B. pip und conda) oder verfügen über ein *Command Line Interface* (**CLI**, z.B. `docker`). Historisch war die Kommandozeile die einzige oder zumindest die einfachste Wahl, um eine Benutzerschnittstelle zu realisieren, neuerdings und insbesondere im Bereich der Cloud erlebt diese Art der Benutzerinteraktion eine regelrechte Renaissance: der Grund liegt darin, dass eine CLI (1) einfach zu dokumentieren ist, (2) sehr einfach aus Skripten verwendet werden kann, und (3) dass einfach dagegen zu programmieren ist, etwa für Automatisierungstools.

Scripting

Gängige Skriptsprachen für die Automatisierung sind die *Shell*[31] (in Linux/Unix-Systemen) und *Batch Script* oder *PowerShell* (in Windows). Auf komplexe Konstrukte in diesen Sprachen sollte verzichtet und eine möglichst lineare Abarbeitung angestrebt werden.

Im Abschnitt über die Isolation hatte die Docker CLI einen Auftritt, es ging u.a. darum, einen Container zu starten. Das war ein einzelner Befehl, aber mit mehreren Argumenten und Optionen. Diesen einen Befehl in ein einzeiliges Skript zu setzen und diesem einen sprechenden Namen zu geben, stellt für die Reproduzierbarkeit (und Ihr Gedächtnis) schon einen grossen Wert dar. Konkretes Beispiel:

[31] Im Kontext von Automatisierung, Cloud und Linux wird oft von «Bash» gesprochen. Die Bash ist eine Shell (Bourne Again Shell), es gibt andere. Für gute Portabilität sollte auf «Bashismen» (Bash-spezifische Konstrukte) verzichtet und nur verwendet werden, was auch im POSIX-Standard zur Shell steht.

Datei: *Run_Ephemeral_GDS_Lab.bat*

Inhalt:
```
docker run -it --rm -p 8888:8888 \
  -v C:\Projektordner:/home/jovyan/work" darribas/gds_py:8.0
pause
```

Ablage: im Projektordner

Das Zeichen `\` soll anzeigen, dass die zwei Zeilen auf einer stehen müssen. Der Projektordner ist hier hart codiert (`C:\Projektordner`); das ist nicht falsch, u.U. wäre es aber angezeigt, sich mit `-v "%cd%:/home/jovyan/work"` auf das aktuelle Verzeichnis zu beziehen (in Linux: `$PWD` anstatt `%cd%`). Das `pause` auf der zweiten Zeile bewirkt, dass das Konsolenfenster offenbleibt, nachdem der `docker`-Befehl beendet ist, was nützlich ist, um allfällige Fehlermeldungen lesen zu können.

Ein Skript kann auch mehrere Befehle enthalten, etwa um conda zu installieren, eine virtuelle Umgebung zu erstellen, diese zu aktivieren, darin Pakete zu installieren. Bei der Verwendung von conda sollte unbedingt ein *environment.yaml* in Erwägung gezogen werden, im Falle von pip ein *requirements.txt* verwendet werden. Beide vereinfachen die Automatisierung und dokumentieren die Intention, welche Pakete vorhanden sein müssen.

Falls Sie Ihr eigenes Container Image erstellen, verfügen Sie schon mal über ein Dockerfile, welches im Wesentlichen eine Sequenz von Befehlen darstellt, die den Inhalt des Images erzeugen. Der dann notwendige `docker build` Befehl und allenfalls ein `docker tag` und `docker push` sollten auf jeden Fall auch in einem Skript festgehalten sein.

Make

Ein uraltes aber äusserst wertvolles Werkzeug aus der Unix-Welt heisst make: Es liest eine Textdatei, genannt *Makefile*, und führt die darin aufgelisteten Programme (und Skripte) aus, unter Berücksichtigung schon gemachter Arbeit. Details führen hier zu weit, ich empfehle aber die folgenden beiden Artikel Ihrer Aufmerksamkeit (sie führen zugleich in einen von Python und Notebooks gänzlich unterschiedlichen Software-Stack, jenen von JavaScript und Node).

- M Bostock, *Why Use Make*, blog article, 2013, https://bost.ocks.org/mike/make/
- M Bostock, *Command Line Cartography*, article series on medium.com, 2016, https://medium.com/@mbostock/command-line-cartography-part-1-897aa8f8ca2c

Make arbeitet mit «targets», Zielen, die gemacht werden sollen. Hier empfiehlt sich ein Ziel «clean» (konventioneller Name), welches aufräumt, d.h., temporäre Dateien und Zwischenergebnisse löscht. Dies zwingt zur Überlegung, was denn erhaltenswert ist, nämlich sämtliches Quellmaterial, und was nicht, nämlich alles was automatisiert aus Quellmaterial hergeleitet wird. Wenn Docker verwendet wird, könnten bei «clean» heruntergeladene oder erstellte Container Images gelöscht werden, um Platz zu sparen.

Git

Jedes Projekt beginnt mit einem Projektordner – jedenfalls ist das eine Empfehlung. Darin werden die Artefakte des Projekts abgelegt, z.B. ein Jupyter Notebook, Daten, Quellverweise, Skripte zur Automatisierung. Sie werden im Verlauf des Projekts in Iterationen (vgl. CRISP-DM im Abschnitt Daten, Dateien, Dienste) damit arbeiten, Veränderungen vornehmen, experimentieren. Vielleicht wollen Sie auch mal wieder zurück zu einem früheren Stand und von da eine andere Abzweigung nehmen, etwas anderes ausprobieren. Hierbei helfen sogenannte *Source Code Management* Systeme (SCM, auch *Version Control* oder *Source Control* genannt). Das gegenwärtig bekannteste SCM heisst Git (git-scm.com), ist frei

verfügbar, läuft auf allen gängigen Systemen (Linux, Windows, macOS), hat eine CLI (`git`) und über Drittprojekte diverse graphische Oberflächen. Mit Git erhalten Sie eine Zeitmaschine und Sie werden mutiger Sachen ausprobieren, weil Sie wissen, dass Sie jederzeit zurückkönnen.

Zur Funktionsweise: mit `git init` verwandeln Sie Ihr Projektverzeichnis in ein Git Repository. Mit `git add` und `git commit` werden alle Änderungen zum Repository hinzugefügt und ein neuer Commit erstellt. Mit `git checkout` können Sie zu einem früheren Commit zurückgehen. Sie sollten sich auch ein Remote Repository (ein Klon Ihres Repositories auf einem anderen Rechner) einrichten und Ihre lokalen Commits mit `git push` auf das Remote Repository übertragen. Der bekannteste Dienst für Remote Git Repositories heisst GitHub (github.com) und ist bis zu einem gewissen Umfang gratis verwendbar. Sie können ein Remote Repository auch öffentlich machen, also für die ganze Welt einsehbar. Das ist dann ganz im Trend der Open Science.[32] Für die Details sei auf die Dokumentation und die zahlreichen Tutorien zu Git im Internet verwiesen.

Infrastructure as Code

Die Konzepte und Werkzeuge bis hierher entstammen weitgehend der Software-Entwicklung und haben sich in anderen Domänen bewährt, in denen mit (Text-)Dateien gearbeitet und automatisiert wird. Wir sind damit ein grosses Stück des Weges hin zum Konzept der Infrastruktur als Code gegangen. Dieses aktuelle Konzept bedeutet, dass die Infrastruktur basierend auf maschinenlesbaren Beschreibungen automatisch bereitgestellt werden und kommt vor allem bei der Bereitstellung von Cloud-Infrastrukturen zum Einsatz. Es lässt sich aber problemlos und vorteilhaft auf kleinere Infrastrukturen und einzelne Analyse-Umgebungen übertragen. Infrastrukturen in der Cloud (IaaS) können zwar in einer graphischen Oberfläche «zusammengeklickt» werden; besser aber ist es, sie programmatisch zu erstellen. Dazu verfügen die gängigen Cloud-Anbieter über ein CLI (bei Microsoft Azure z.B. heisst dieses `az`). Über diese CLI können beliebige Ressourcen (z.B. Server, Netzwerke, Speicher) erstellt und verwaltet werden können. Die damit ausgeführten Befehle werden wiederum in einem Skript festgehalten, damit sie repliziert werden können. Wo früher Server individuell konfiguriert und gepflegt wurden, werden sie heute einfach automatisch erzeugt und können daher auch hemmungslos wieder gelöscht werden. Man sagt dazu etwas verächtlich, Computer seien “cattle, not pets” (Vieh, nicht Haustiere).

Für das Starten von Docker Containern gibt es das Werkzeug `docker compose`, welches basierend auf einer Beschreibungsdatei (*compose.yaml*) eine Anzahl Container konfiguriert und startet, und damit das Konzept von Infrastruktur als Code umsetzt. Für grössere Umgebungen wird das zu komplex und es kommen Werkzeuge für die «Orchestrierung» von Containern zum Einsatz, das bekannteste davon ist Kubernetes mit der CLI `kubectl`.

Für den heutigen Themenblock geht das zu weit, es ist aber wichtig, den grösseren Kontext zu kennen, ebenso aber auch den Wert eines einfachen Skriptes, um eine Umgebung zu erstellen oder zu starten. Automatisierung ist Disziplinierung und Aufwand, jedoch mit dem grossen Nutzen der Reproduzierbarkeit.

[32] Siehe «Open Science» auf Wikipedia: en.wikipedia.org/wiki/Open_science

10 Cloud

Cloud Computing ist ein neues Modell des Computings, definiert als «ubiquitous, convenient, on-demand, shared, configurable computing resources» (NIST 2011). Einfacher gesagt ist die Cloud «der Computer eines anderen» (Peery 2019). Hinter den Kulissen handelt es sich bei der Cloud um grosse Rechenzentren, die von spezialisiertem und qualifiziertem Personal betrieben werden. Es geht um virtualization, networking, scheduling, load balancing, communication, security, monitoring, aber auch um energy, cooling, compliance, lobbying, etc. In diesem Bereich gibt es diverse Anbieter; die «grossen drei» sind Amazon AWS, Microsoft Azure und Google Cloud.

Mit Hilfe von Cloud-Diensten ist es möglich, eine Infrastruktur ganz ohne eigene IT-Abteilung aufzubauen; in der Tat wurde die Umgehung der eigenen IT in der Frühzeit der Cloud oft als Grund für die Inanspruchnahme von Cloud-Diensten genannt. Plakativ gesprochen braucht es für die Nutzung der Cloud genau drei Dinge: ein Notebook, einen Internet-Anschluss und eine Kreditkarte.

Um etwas Ordnung in die Vielzahl von Cloud-Diensten und deren Kombination mit anderen Cloud-Diensten und eigenen (on-premise) Diensten zu bringen, unterscheidet man drei Service Models und drei Deployment Models:

Service Models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS). Diese unterscheiden sich darin, auf welchem Abstraktions-Level sich die in Anspruch genommenen Cloud-Dienstleistungen befinden. Siehe **Abbildung 12**. Beispiele: Virtueller Server in der Cloud (IaaS), Esri's ArcGIS Online (PaaS), mapshaper.org zur Bearbeitung und Vereinfachung von Vektordaten (SaaS).

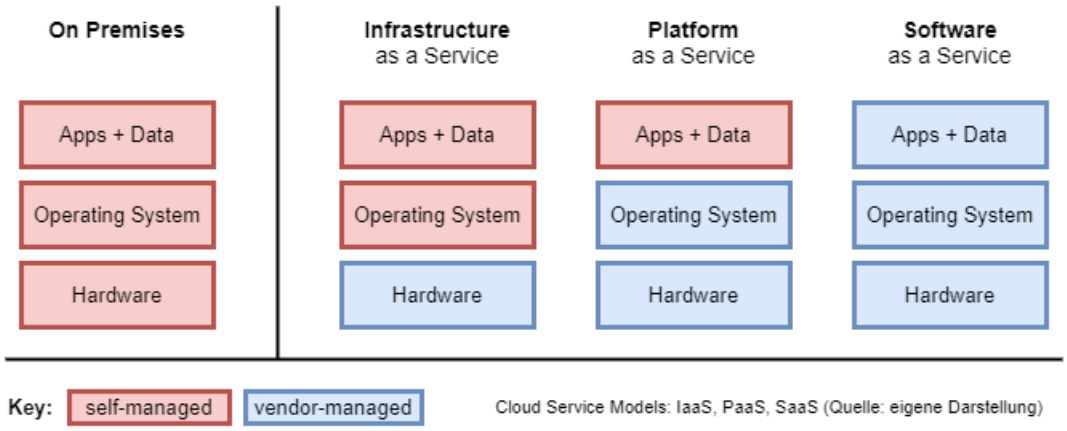


Abbildung 12. Cloud Service Models

Deployment Models: Public, Private, Hybrid. Dies hat damit zu tun, wer die Cloud-Technologie betreibt, ein öffentlicher Anbieter (*public*), die Firmeneigene IT (*private*), oder Teils/Teils (*hybrid*).

Im Abschnitt zur Isolation haben wir Docker Container aus der Cloud bezogen; dabei haben wir eine Stück PaaS (nämlich eine Container Registry) in einer Public Cloud (nämlich Docker Hub) konsumiert.

Zwei für uns interessante Beispiele seien hier vorgestellt: das Binder Project von Jupyter und Esri's ArcGIS Online Cloud Offering.

Binder ist ein Projekt innerhalb des Jupyter Projekts. Idee und Ziel ist, ab einem Repository (z.B. in GitHub) ein Docker Image zu erstellen, und dieses in einer Cloud-Infrastruktur laufen zu lassen. Das ist als Konzept interessant, weil es alle wesentlichen Konzepte und Prinzipien der Abschnitte zur Isolation und zur Automation verwirklicht: ab einem versionierten Repository voll automatisiert die notwendige Infrastruktur in der Cloud bereitstellen und damit die in Jupyter Notebooks vorgenommenen Analysen für beliebige Anwender nachvollziehbar machen. Das Projekt erarbeitet nicht nur die Konzepte und die notwendige Software, sondern betreibt mit mybinder.org auch einen «proof of concept», mit dem jedermann sein Repository laufen lassen kann. Interessant sind auch die vom Projekt veröffentlichten Angaben zu den Kosten für den Betrieb (Project Jupyter 2018); leider sind sie seit einiger Zeit nicht aktualisiert.

«Plattform» ist ein Begriff eher aus der kommerziellen Welt. Die führende Plattform im Bereich GIS ist wohl jene, welche von Esri unter dem Namen ArcGIS entwickelt und vermarktet wird. Integrieren viele Aspekte der räumlichen Analyse und stellen diese bequem zur Verfügung. Existiert in der Public Cloud als *ArcGIS Online* (www.arcgis.com) und on-premise als *ArcGIS Enterprise*[33] (bestehend aus *ArcGIS Server* und *Portal for ArcGIS*). Eine Plattform eine grosse und komplexe, aber aufeinander abgestimmte und unterhaltene Infrastruktur. Wenn diese in die IT-Landschaft einer (grossen) Organisation eingebunden ist, spricht man auch von «Enterprise GIS». Nutzung (Cloud) oder Betrieb (on-premises) dieser Plattform ist mit Kosten verbunden, dafür werden einem viele der in diesem Skript angesprochenen Aufgaben abgenommen. Ob sich das lohnt, hängt nicht zuletzt mit persönlichen Präferenzen und dem institutionellen Umfeld zusammen.

Die *Cloud Native Compute Foundation* (CNCF, unter dem Dach der Linux Foundation) publiziert mit der Cloud Native Landscape (landscape.cncf.io) eine Übersicht über die überwältigende Anzahl an Open Source Projekten und proprietären Produkten im Bereich der Cloud. Gemessen daran haben wir haben die Cloud nur knapp berührt und doch einige Vorzüge daraus gezogen. Die Vorzüge bestanden aber oft darin, dass andere für uns gearbeitet haben, etwa funktionierende Python-Umgebungen zusammengestellt. Seien wir uns dessen bewusst.

Ebenfalls von der CNCF stammt die Cloud Native Trail Map, eine nette Darstellung des Weges zu einer reinen Cloud-Existenz. Es ist dies ein Weg, den man gehen kann, aber nicht muss. Er kann auch nur teilweise begangen werden. Die Entscheidungen sind im Einzelfall zu treffen. Der aktuelle Trend aber geht klar in Richtung Cloud. Wirtschaftlich gesprochen: der Trend geht bei der Infrastruktur von Make zu Buy. Bis in 5 oder 10 Jahren mag sich das schon wieder ganz anders darstellen – wir werden es sehen.

[33] Esri ArcGIS Enterprise: www.esri.com/en-us/arcgis/products/arcgis-enterprise

11 Schluss

Nach einigen Iterationen (CRISP-DM), vielleicht auch nie, führt eine Analyse zu einer Anwendung, die im Geschäftsalltag verwendet wird und damit dem produktiven Betrieb standhalten muss. Mit dem Übergang von den explorativen Datenanalysen in die Produktion ändern sich die Anforderungen an die Infrastruktur und es kommen neue Anforderungen hinzu: Belastbarkeit, Sicherheit, Umgang mit Credentials, Ergonomie, Ästhetik, Compliance, Life Cycle. Dadurch gewinnen die *Prinzipien von Infrastructure as Code* an Bedeutung. Es sind dies (Morris 2015):

- Systeme sind einfach reproduzierbar: jeder Teil einer Infrastruktur (in unserem weiten Sinne) sollte ohne grosse Aufwände neu erstellt werden können.
- Systeme sind entbehrlich (*disposable*): eine Anwendung sollte so gebaut sein, dass der Ausfall anderer Systeme gut weggesteckt werden kann. Mindestens sollte die Situation richtig erkannt werden und in eine aussagekräftige Fehlermeldung münden.
- Systeme sind konsistent: wenn viele Systeme (Hard- wie Software) im Einsatz sind, es sollten alle einer Klasse gleich und damit austauschbar und ohne Überraschungen sein.
- Arbeitsschritte sind wiederholbar: Anpassungen werden nicht am System selbst, sondern an dessen Beschreibung vorgenommen (z.B. an einem Skript oder an einem *Dockerfile*) und das System neu erzeugt.

Mit den in den Abschnitten zur Isolation und zur Automation vorgestellten Praktiken und Techniken (v.a. Scripting und Versionierung[34]) folgt man diesen Prinzipien ein gutes Stück von selbst. Wichtig auch: Änderungen in kleinen Schritten und permanentes Testen. Dank virtuellen Umgebungen und Containern können Sie sich für jede Analyse eine weitgehend isolierte Infrastruktur leisten – und sollten das auch tun.

Stellen räumliche Daten und die räumliche Datenanalyse spezielle Anforderungen? An die statistischen Verfahren schon (Anselin 1989; insbesondere gilt wegen Toblers «First Law» die Annahme der Unabhängigkeit nicht), an die Infrastruktur kaum. Natürlich muss der Software-Stack in den oberen Schichten mit Bibliotheken und Werkzeugen für räumliche Daten ausgestattet werden (z.B. geopandas anstatt nur pandas in Python), oder es muss eine Plattform wie ArcGIS gewählt werden, welche räumliche Funktionen bietet und die einschlägigen Standards unterstützt. Für die Dimensionierung der tieferen Schichten spielt es eine Rolle, ob viel oder wenig Daten verarbeitet werden, nicht aber ob es sich um räumliche oder andere Daten handelt.

[34] Credentials (z.B. Passwörter) sollten *nie* in ein Repository eingecheckt werden!

12 Glossar

Die Gebiete Infrastruktur, Technik, Cloud sind von einer Vielzahl an Begriffen und Abkürzungen umgeben. Diese können alle problemlos im Internet gefunden und erklärt werden. Trotzdem empfehle ich die Führung eines persönlichen Glossars. Der Rumpf hier mag als Beispiel und Einstieg dienen.

API = Application Programming Interface, eine Schnittstelle zu einem System, die für die Verwendung durch Programme gedacht ist (im Unterschied etwa zur Benutzerschnittstelle, die für den Menschen gedacht ist; Vergleiche auch CLI).

CLI = Command Line Interface, eine alte Idee, die aktuell wieder an Popularität zulegt: Docker, Kubernetes, Azure und die meisten anderen modernen Dienste können über ein Kommandozeilen-Tool genutzt und verwaltet werden. Die CLI für Docker heisst `docker`, jene für Kubernetes heisst `kubectl`, jene für Azure heisst `az`. Solche CLI sind attraktiv, weil sie aus Automatisierungs-Skripts einfach zu verwenden sind.

Container sind isolierte Laufzeitumgebungen für Prozesse. Sie enthalten ein ganzes Dateisystem, inklusive Betriebssystem-Tools, nicht aber den Kern des Betriebssystems: dieser wird virtualisiert (dies im Unterschied zu einer virtuellen Maschine, welche die Hardware virtualisiert).

Container Registry ist ein Dienst, der Container Images speichert und über ein API bereithält. Die erste und grösste solche Registry ist Dockers Hub auf hub.docker.com. GitHub Packages bieten eine freie (für öffentlich zugängliche Images) Alternative, Microsoft hat die ACR, und es gibt viele weitere.

Docker ist eine freie Software für die Erstellung, Verwaltung und den Betrieb von Containern (Docker Engine und CLI), sowie das Unternehmen dahinter, welches diese Software entwickelt und dafür auch kommerzielle PaaS-Produkte anbietet.

Environment, kann in der Informatik ganz unterschiedliche Aspekte der Umgebung eines Prozesses bezeichnen, etwa als “Environment Variables” gewisse Variablenzuweisungen, die einem Programm mitgegeben werden, die Paketlandschaft, in der ein Python-Programm läuft, oder auch das Betriebssystem; ferner werden organisatorisch mehrfach ausgeführte Stacks als Environment bezeichnet: z.B. Entwicklungs-, Test- und Produktionsumgebung.

GIS = Geographic Information System, ein Softwaresystem zur Erfassung, Speicherung, Analyse, Manipulation und Präsentation von Daten mit einem Raumbezug.

OGC = Open Geospatial Consortium, ein Standardisierungsgremium im Bereich räumlicher Daten und Prozesse; wichtige Standards: SFS, WMS, WMTS, WFS, WCS (und weitere).

SDI = Spatial Data Infrastructure, bezeichnet sowohl die Technologie als auch die Organisation, um geographische Daten zu katalogisieren und zu verbreiten, indem sie suchbar, auffindbar, und verwendbar gemacht wird.

SDK = Software Development Kit, eine Zusammenstellung von Werkzeugen für die Software-Entwicklung für eine bestimmte Plattform oder Umgebung.

SRID = Spatial Reference ID, eine Nummer, die ein Raumbezugssystem (“Koordinatensystem”) bezeichnet; im Esri-Kontext auch als WKID (Well-Known ID) und Factory Code bekannt.

SRS = Spatial Reference System, ein räumliches Bezugssystem, definiert das Verhältnis von Koordinaten zu einem Ort auf der Erde. Man unterscheidet Geographische Koordinatensysteme (Länge und Breite), und Projizierte (oder geodätische) Koordinatensysteme (x und y, oder auch Easting und Northing).

Stack, im Sinne von Software-Stack oder Solution-Stack, bezeichnet die Menge von Komponenten oder Subsystemen, die zusammen eine «Plattform» bilden, auf der eine Anwendung läuft oder entwickelt werden kann. Es geht immer um die konkrete Auswahl von Komponenten (z.B. Linux-Apache-MySQL-Python), nicht um die Kategorien (Betriebssystem-Webserver-Datenbank-Skriptsprache).

System (gr. σύστημα = ein aus Teilen zusammengesetztes Ganzes) bezeichnet eine erkennbare Einheit, die aus miteinander in Beziehung stehenden Komponenten besteht. Ein weit verwendeter Begriff in der Informatik und anderen Wissenschaften.
Scherzhaft: “Ein System ist alles, was keines hat” (Quelle unbekannt).

13 Literatur

Wo verfügbar und bekannt ist ein Link zum Dokument angegeben. Eine sehr nützliche Quelle ist der *Geographic Information Science & Technology Body of Knowledge*, eine kuratierte und öffentliche Sammlung von Artikeln zum genannten Thema von Wissenschaftlern im jeweiligen Bereich.

Daten

- L Anselin, "What is special about spatial data? Alternative perspectives on spatial data analysis" (89-4). National Center for Geographic Information and Analysis, U of California Santa Barbara, 1989.
- E Chow, Y Yuan, "GIS APIs." *The Geographic Information Science & Technology Body of Knowledge*, John P. Wilson (ed.), 2019. DOI: 10.22224/gistbok/2019.2.15.
- A U Frank, *The Surveying Activities at the Austrian Federal Office for Metrology and Surveying: An Economic Analysis (Volkswirtschaftliche Studie zu den Leistungen des Bundesamtes für Eich- und Vermessungswesen im Vermessungswesen)*. Austrian Federal Ministry of Economics and Labour, Wien 2003. Project No. 96000/11-IV/13/01, Final Report.
- M F Goodchild "Citizens as sensors: the world of volunteered geography." *GeoJournal* 69, pp. 211-221, 2007. citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.525.2435
- Y Hu, "Spatial Data Infrastructures." *The Geographic Information Science & Technology Body of Knowledge*, UCGIS, 2017, dx.doi.org/10.22224/gistbok/2017.2.1
- L Rose, "Geospatial portal reference architecture: a community guide to implementing standards-based geospatial portals." OpenGIS Discussion Paper, OGC, 04-039, 2004. Darin das publish-find-bind Muster für SDI.
- Bundesgesetz über Geoinformation (GeoIG, SR 510.62) http://www.fedlex.admin.ch/eli/cc/2008/388/de
- Topology in GIS: en.wikipedia.org/wiki/Geospatial_topology

Datenbanken

- E F Codd, "A Relational Model for Large Shared Data Banks," *CACM*, 13:6, June 1970 – das Original.
- R Elmasri und S B Navathe, *Fundamentals of Database Systems*, Pearson, 1989ff – ein Lehrbuch-Klassiker.
- T Nyerges, "Spatial Database Management Systems." *The Geographic Information Science Technology Body of Knowledge*, UCGIS, 2021, doi.org/10.22224/gistbok/2021.1.11
- PostgreSQL documentation is at www.postgresql.org and includes a good tutorial.
- PostGIS documentation is at postgis.net and includes an excellent tutorial.

Data Science

- T H Davenport, DJ Patil, "Data Scientist: The Sexiest Job of the 21st Century," *Harvard Business Review*, October 2012, hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century
- T H Davenport, DJ Patil, "Is Data Scientist Still the Sexiest Job of the 21st Century?", *Harvard Business Review*, July 2022, hbr.org/2022/07/is-data-scientist-still-the-sexiest-job-of-the-21st-century – eine Betrachtung des vorigen Artikels 10 Jahre später.
- J Grus, *Einführung in Data Science*, O'Reilly, 2016 – Prinzipien mit Python.
- J D Kelleher und B Tierney, *Data Science*, MIT Press, 2018 – eine umfassende Einführung.

– S J Rey, D Arribas-Bel, L J Wolf, *Geographic Data Science with Python*, online at geographicdata.science/book – eine universitäre Lehrveranstaltung zum Thema.

Python & Notebooks

– G Boeing, D Arribas-Bel, "GIS and Computational Notebooks." *The Geographic Information Science & Technology Body of Knowledge*, 2021, https://doi.org/10.22224/gistbok/2021.1.2
– Project Jupyter, *Jupyter Project Documentation*, online at docs.jupyter.org
– Python Software Foundation, *Python documentation*, online at docs.python.org
– L Ramalho, *Fluent Python*, O'Reilly, 2015, 2nd ed. 2022 – Python wirklich können.
– K Reitz and T Schlusser, *The Hitchhiker's Guide to Python*, O'Reilly, 2016 – best practices.
– S J Rey, "Python for GIS." *The Geographic Information Science & Technology Body of Knowledge*, 2017, https://gistbok.ucgis.org/bok-topics/python-gis
– H Tenkanen, V Heikinheimo, D Whipp, *Introduction to Python for Geographic Data Analysis*, CRC Press, upcoming, online at pythongis.org
– Python Ecosystem for GIS and Earth Observation, online at ecosystem.pythongis.org
– Vergleich von Conda and Pip: www.anaconda.com/blog/understanding-conda-and-pip
– Beispiel für das Einrichten einer Python-Umgebung mit conda und *environment.yml*: tdhopper.com/blog/my-python-environment-workflow-with-conda

Standards

– D Crockford, *Introducing JSON*, online at www.json.org
– M J Egenhofer, R D Franzosa, "Point Set Topological Spatial Relations," *Geographical Information Systems*, vol 5, no 2, pp 161-174, April 1991, dx.doi.org/10.1080/02693799108927841
– European Petroleum Surveying Group (EPSG), *The EPSG Geodetic Parameter Set*, online at epsg.org
– J Fee, "Esri REST API could be an OGC Standard", in his blog *Spatially Adjusted*, June 2011, spatiallyadjusted.com/esri-rest-api-could-be-an-ogc-standard
– Open Geospatial Consortium, *OGC E-learning*, 2017ff, online at opengeospatial.github.io/e-learning/index.html – Über OGC und seine Standards.
– Open Geospatial Consortium (OGC), *Simple Feature Access – Part 1: Common Architecture*, Version 1.2.1, 2011. www.ogc.org/standards/sfa
– Open Geospatial Consortium (OGC), *Simple Feature Access – Part 2: SQL Option*, Version 1.2.1, 2011. www.ogc.org/standards/sfs

Isolation, Automation, Cloud

– M Bostock, *Why Use Make*, blog article, 2013, bost.ocks.org/mike/make
– M Bostock, *Command Line Cartogaphy*, article series on medium.com, 2016, medium.com/@mbostock/command-line-cartography-part-1-897aa8f8ca2c
– Cloud Native Computing Foundation, *The CNCF Landscape Guide*, landscape.cncf.io/guide
– Docker Inc., *Docker Reference Documentation*, online at docs.docker.com/reference
– Esri Inc., *Architecting the ArcGIS System: Best Practices*, 2022 (continuously updated)
– P Mell, T Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology, 2011. doi.org/10.6028/NIST.SP.800-145
– Microsoft, *PowerShell Documentation*, online at docs.microsoft.com/en-us/powershell
– J Petazzoni, *Container Training*, online at container.training – free slides and videos.
– Project Jupyter et al., "Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale." *Proceedings of the 17th Python in Science Conference*, 2018,

doi.org/10.25080/Majora-4af1f417-011 – eine für die Datenanalyse interessante Cloud-Architektur und Anwendung.
- Project Jupyter, *The Binder Project*, online at jupyter.org/binder
- C Ramey et al., *Bash Reference Manual*, Free Software Foundation, 1988ff., www.gnu.org/software/bash – Handbuch zur Bash Shell
- R M Stallman et al., *GNU Make*, Free Software Foundation, 1988ff., www.gnu.org/software/make – Handbuch zu GNU make
- Wikipedia, *Bash (Unix shell)*, en.wikipedia.org/wiki/Bash_(Unix_shell)
- Wikipedia, *Stapelverarbeitungsdatei*, de.wikipedia.org/wiki/Stapelverarbeitungsdatei – als Einstieg zu Batch Files (*.bat*, *.cmd*)

Weiterführend

- P Chapman et al., *CRISP-DM 1.0: Step-by-step data mining guide*, 2000, api.semanticscholar.org/CorpusID:59777418
- K Morris, *Infrastructure as Code*, O'Reilly, 1st ed. 2015, 2nd ed. 2020.
- S Peery, "Enterprise GIS," *The Geographic Information Science & Technology Body of Knoweldge*, 2019, doi.org/10.22224/gistbok/2019.2.7
- J P Wilson (ed.), *The Geographic Information Science & Technology Body of Knowledge*, University Consortium for Geographic Information Science, 2016ff, gistbok.ucgis.org

14 Danksagung

Einige Inspiration für das vorliegende Skript entstammt einem Foliensatz von Dr. Reik Leiterer (ExoLabs GmbH) zu Daten und Datenanalyse. Weiter bin ich dankbar für die vielen guten Unterlagen, die von engagierten Leuten im Internet zur Verfügung gestellt werden, und aus denen ich reichlich geschöpft habe (die Quellen sind jeweils angegeben). Ein Dank geht auch an die Unternehmen Dira GeoSystems AG (vertreten durch Emanuel Mahler) und Esri Schweiz AG (vertreten durch Stefan Graf), die ebenfalls Materialien und Ressourcen bereitgestellt haben. Schliesslich geht ein grosser Dank an meine Familie, die mein Engagement grosszügig mitgetragen hat.

dirageosystems.ch

www.esri.ch