Christoph Rinne

Markdown & Code: Open Source & Interaktivität für die primäre Zielgruppe von Windows Nutzerinnen und Nutzern

Abstract The article refers to the possibilities of the Markdown markup language in combination with executable code for the documentation and reuse of research data as well as their results. However, the focus is on the presentation of three application scenarios that utilise this method of working in teaching. The first scenario focuses on the use of specialist software, such as *RStudio*, to quickly produce consistent documents in layout and the result displayed as required. The original text is dynamic, but can also be published as a static handout, e.g. as a PDF. Using the original document in *RStudio*, as stated in the second scenario, offers more possibilities. The third option is completely detached from any specific software use on the part of the students. Instead, the content is displayed via the *Jupyter* software in a web application.

Zusammenfassung Der Beitrag thematisiert die Möglichkeiten der Auszeichnungssprache Markdown in Kombination mit ausführbarem Code zur Dokumentation und Nachnutzung von Forschung und den daraus resultierenden Ergebnissen. Der Schwerpunkt liegt dabei besonders auf drei Anwendungsszenarien in der Lehre. Das erste Szenario befasst sich mit der Verwendung von spezieller Software, z. B. *RStudio*, um im Layout konsistente Dokumente zügig zu erstellen und das Ergebnis je nach Bedarf zu visualisieren. Es entsteht ein dynamisches Dokument, welches als statische Handreichung, z. B. als PDF, herausgegeben werden kann. Die Verwendung des genuinen Dokumentes in *RStudio*, im zweiten Beispiel vorgestellt, bietet zusätzliche Möglichkeiten. Die dritte Option löst sich vollständig von einer Softwarenutzung durch die Lernenden. Vielmehr werden die Inhalte über die Software *Jupyter* in einer Webanwendung dargestellt.

Keywords Markdown, R, Lehre, Reproduzierbarkeit, spezialisierte Tools

Rinne, Christoph (2025): Markdown & Code: Open Source & Interaktivität für die primäre Zielgruppe von Windows Nutzerinnen und Nutzern, in: Bolder-Boos, Marion – Hageneuer, Sebastian – Pantelidis, Georg (Hrsg.), *Digitale Methoden des Lernens und Lehrens in der Archäologie,* Heidelberg: Propylaeum, S. 109–118, [online] https://doi.org/10.11588/propylaeum.1572.c23244.

Einleitung

Das Thema des Workshops „Digitale Methoden des Lehrens und Lernens in der Archäologie. Chancen und Herausforderungen" scheint auf den ersten Blick nicht prädestiniert, eine sehr spezifische Technik vorzustellen, mit der Text und Code zugleich editiert werden können. Dennoch widmet sich sowohl dieser Beitrag, als auch der Beitrag von Maria Shinoto dieser Technik. Die in beiden Beiträgen vorgestellte Software kann zudem als notorische Mahnung für freie und gegen proprietäre Systeme missverstanden werden. Beiden Beiträgen in diesem Themenkomplex liegt dies aber fern. Vielmehr sollte auf eine eigentlich altbewährte Arbeitsweise hingewiesen werden (siehe Beitrag Maria Shinoto), die dank moderner Technik zunehmend leichter und auf vielfältige Weise in den Arbeitsalltag integriert werden kann; eben auch in dem Kontext digitalen Lehrens und Lernens. Folgende Punkte sind hierfür als grundlegend anzusehen:

1. Die weitgehende, technische Reduktion auf Text erleichtert die Archivierung (siehe Beitrag Martina Trognitz).
2. Die Trennung des Schreibens von der technischen Umsetzung des Layouts erlaubt die Fokussierung auf Ersteres.
3. Die Integration von ausführbarem Code im Text schafft zugleich ein dynamisches Dokument mit vielfältigen Optionen.

Typische Szenarien für Markdown & Code

Der in der obigen Auflistung überwiegend eher technisch konnotierte Kontext hat zugleich essentielle Vorteile für die Nachvollziehbarkeit unseres wissenschaftlichen Arbeitens (*reproducible research*). Dies trifft zugleich auf zwei Ebenen zu, der informatorischen (*computational*) Reproduzierbarkeit und den ergänzenden Daten zur Analyse (vgl. u. a. Marwick 2017). Einfache Quantifizierungen, Berechnungen oder Grafiken können als Code im Textdokument hinterlegt werden und sind neben dem Resultat zudem ergänzend abrufbar. Komplexere Prozeduren, wie z.B. eine Abfolge von Befehlen an den Computer oder auch ganze Programme, können als Supplement nicht nur in der Form einer komplexen, kaum nachvollziehbaren Codesammlung mit minimalen Kommentaren hinterlegt werden, sondern als erläuternder Text, der eine Verbindung zwischen der Publikation der Ergebnisse und der gesamten Datenverarbeitung schafft (vgl. Kruschke 2020). Zugleich lassen sich hier auch die Daten selbst gut dokumentiert oder dynamisch hinterlegen, sowohl als kompakter Download, als Weblink, als Link auf beiliegende Daten oder einfache Listen im

Code (z. B. Rinne 2024). Die Kombination aus Erläuterung, Code und Ergebnis ist für Handreichungen in einer Übung und zum ergänzenden Selbststudium geradezu prädestiniert.

Markdown ist eine prinzipiell von dem Betriebssystem und der Software unabhängige Auszeichnungssprache. Inzwischen liegen einzelne Varianten vor, z. B. Markdown in R (Rmd) in der Software *RStudio*, welche die anfängliche Unabhängigkeit zusehends einschränkt. Hinzu kommen spezifische Anweisungen, die am Anfang jedes Dokuments gelistet werden (YAML Header) und damit eine weitere Differenzierung mit sich bringen, z. B. zwischen *RStudio* und anderen Programmen mit einer Markdown Integration. Eine schlichte Variante von Markdown, z. B. für den Anfang dieses Beitrages, würde wie folgt aussehen.

```
---
title: „Markdown & Code: Open Source & Interaktivität"
subtitle: „Für die primäre Zielgruppe von Windows Nutzerinnen und Nut-
zern"
author:
- Christoph Rinne[^1]
date: „18.12.2024"
papersize: a4
urlcolor: blue
lang: de-DE
---
[^1]: [ORCID: 0000-0002-9829-6182](https://orcid.org/0000-0002-9829-
6182)

# Einleitung
Das Thema des Workshops "Digitale Methoden des Lehrens und Lernens
in der Archäologie. Chancen und Herausforderungen" scheint \[...\]
digitalen Lehrens und Lernens. Folgende Punkte sind hierfür als grund-
legend anzusehen:

1.Die weitgehende, technische Reduktion auf Text erleichtert die
Archivierung (siehe Beitrag Martina Trognitz).
2.Die Trennung des Schreibens von der technischen Umsetzung des
Layouts erlaubt die Fokussierung auf Ersteres.
3.Die Integration von ausführbarem Code im Text schafft zugleich ein
dynamisches Dokument mit vielfältigen Optionen.

# Typische Szenarien für *Markdown* & Code
```

Dieses Beispiel zeigt den definitorischen Kopf (YAML) mit der sogenannten Titelei und ergänzenden Angaben zum Layout. Eckige Klammern sind stets Verweise und können unterschiedlich konnotiert werden. In der Abfolge treten ein interner Verweis (Fußnote) und ein klassischer Hyperlink auf. Aus diesem Grund sind die eckigen Klammern im folgenden Text als „`\[`“ gesetzt, um diese besondere Bedeutung aufzuheben. Weitere Optionen in Markdown machen die Verarbeitung von z. B. Tabellen und Abbildungen möglich. Eine mögliche Umsetzung zu einem PDF kann u. a. in dem Markdown-Editor *ghostwriter* erfolgen, der auf *GitHub* entwickelt wird.[1] Die Umsetzung von Code in einer Programmiersprache setzt allerdings die Verwendung einer anderen Software voraus, z. B. das bereits erwähnte Programm *RStudio*.

Fallbeispiel aus der Lehre: SQL für Datenbanken

Übungen in der Archäoinformatik stellen auf einer sehr grundlegenden Ebene mehrfache Anforderungen an die Studierenden. 1. Die Studierenden befinden sich an einem Computer im PC-Labor, auf dem die notwendige Software zwar installiert und unmittelbar nutzbar ist, der ihnen damit aber *a priori* fremd ist. Oder es muss eine neue Software auf dem eigenen Gerät installiert werden, sodass diese einwandfrei nutzbar ist. 2. Die Studierenden müssen der Erläuterung und der in den Raum projizierten Aktion folgen und diese in einer zumeist unbekannten Software nachvollziehen. 3. Die Studierenden müssen diesen Vorgang über die Dauer der Veranstaltung memorieren oder nebenher aufzeichnen. Hierbei entlastet eine Handreichung zum vermittelten Inhalt bei der eigenen Mitschrift und ergänzt die visuelle und praktische Erfahrung aus der Veranstaltung um grundlegende sowie zentrale Informationen. Je mehr Prozesse nicht als Bild der Nutzeroberfläche, sondern als Befehlsfolge (als *short cut* oder Code) integriert sind, desto schlanker wird die Handreichung und ist zugleich konsistenter bei versionsbedingten Änderungen im Design der Softwareoberfläche. Durch die Dokumentation der Grundlagen in der Handreichung wird zugleich Zeit für ergänzende Informationen in der Veranstaltung geschaffen.

Im Folgenden werden drei Nutzungsszenarien dargestellt, von denen die ersten beiden vielfach vom Autor in der Lehre angewendet werden: Statisches Handout, RMarkdown Dokument und *JupyterLab*. Letzteres ist zwar vielfach bewährt, von dieser wird aber nicht aus der eigenen Erfahrung im Lehrbetrieb, sondern nur aus der Vorbereitung berichtet. Alle drei Nutzungsszenarien stellen eine Einführung in die

1 https://github.com/KDE/ghostwriter [zugegriffen am 06.02.2025].

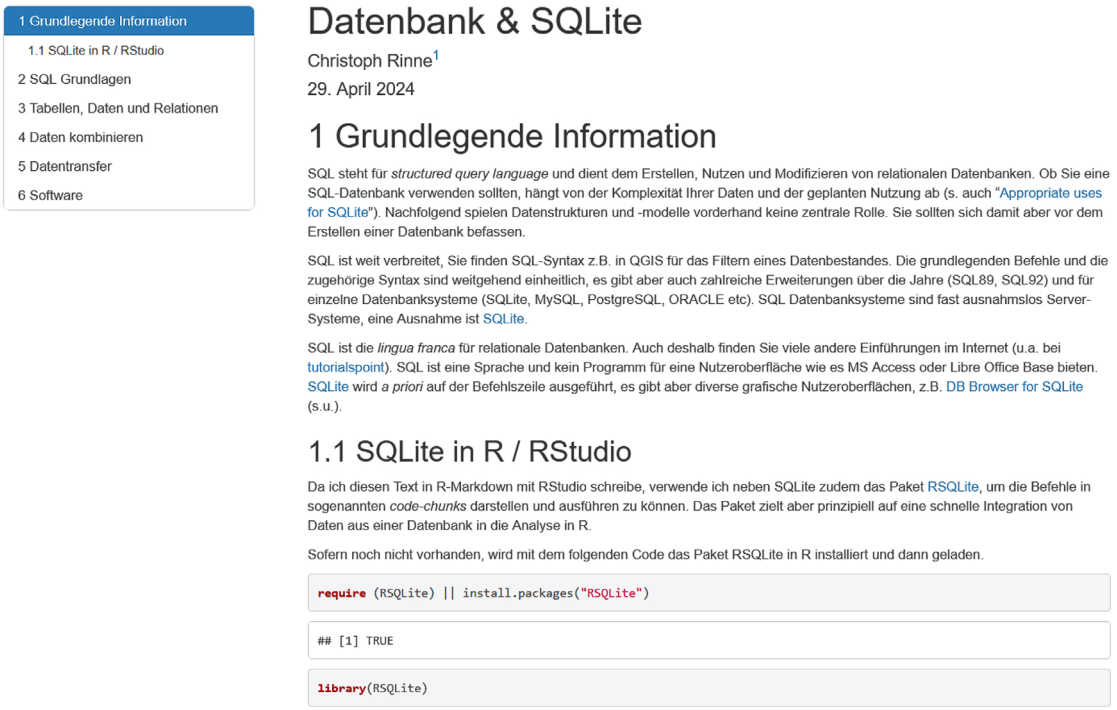

Abb. 1: Das HTML-Dokument zur Einführung in SQLite.

Programmiersprache SQLite im Kontext einer Übung zu Datenbanken dar. Mit der gewählten Software für die Umsetzung ist dies auf dem angestrebten Niveau weitgehend uneingeschränkt möglich. Lediglich die Integration eines Moduls für Raumanalysen, *SpatiaLite*, führt zu einem deutlich erhöhten technischen Aufwand bei der Abstimmung der jeweiligen Softwareversionen.

Handout zu SQLite

Datensammlungen und deren Auswertung sind ein integraler Bestandteil der archäologischen Arbeit. Die Einführung in Datenbanken, u. a. mit der Strukturierung von Daten in einem relationalen System, die Nutzung dieser Daten und deren Weitergabe zur effizienten Nachnutzung durch Dritte sind ein regelmäßiger Bestandteil der Lehre an der Christian-Albrechts-Universität zu Kiel.

Statisches Handout mit RMarkdown

Das Attribut „statisch" bezieht sich hier auf den dargestellten Code, der vom Leser oder der Leserin an dieser Stelle nicht ausgeführt werden kann, sondern in die entsprechende Anwendung übertragen werden muss. Das inzwischen für die Handreichungen bevorzugt verwendete HTML-Format ist *per se* ein dynamisches Dokument mit Inhaltsverzeichnis, Links, einblendbarem Code oder auch dynamischen Tabellen **(Abb. 1)**. Zugleich ist das genuine Dokument in RMarkdown dynamisch und setzt die eingetragenen Befehle nicht nur im Layout inklusive der farblich hervorgehobenen

Codesyntax um, sondern führt die Befehle für den Druck bzw. den Export auch aus. Hierdurch entsteht eine spürbare Effizienz bei der Erstellung der Handreichung. Leider werden, wie eingangs erwähnt, nicht alle Programmiersprachen unmittelbar unterstützt und die Module für *SpatiaLite* können zur Wahrung der Systemintegrität deshalb nicht eingebunden werden. Das Layout und die Hervorhebung der Syntax werden aber dennoch umgesetzt. Damit liegt der Vorteil des schnellen Entwurfs vollständig beim Dozenten, der Nachteil, den Code in eine Anwendung übertragen zu müssen, liegt bei den Studierenden und der daraus resultierende Zeitverlust bei allen an der Lehrveranstaltung Beteiligten.

In Ergänzung zum bereits gezeigten Markdown werden in dem statischen Dokument in RMarkdown in *RStudio* weitere Optionen genutzt, bei denen u. a. R-Code eingesetzt wird. Das Druckdatum kann im YAML-Kopf als aktuelles Systemdatum abgefragt werden: `date: „`r Sys.Date()`"`. Zudem wird zusätzlich das Layout für HTML präzisiert, wobei in der Praxis neue Zeilen (`\n`) und Einrückungen (`\t`) die Angaben strukturieren:

```
html_document: \n\t toc: true \n\t toc_depth: 2 \n\t toc_float: true
\n\t number_sections: true.
```

Im Text selbst sind es dann sog. *code chunks*, die einer Programmiersprache zugewiesen werden und ergänzende Angaben mit einem Titel, zur Darstellung und zur Umsetzung erhalten. In der folgenden Abbildung wird für die Überprüfung, ggf. die Installation und das abschließende Laden des Paketes *RSQLite* folgender R-Code angewendet:

```
```{r Install RSQLite if not present, echo=TRUE, message=FALSE}
require (RSQLite) || install.packages(„RSQLite")
library(RSQLite)
```
```

RMarkdown Dokument in *RStudio*

Bei dieser Variante handelt es sich um das genuine Markdown-Dokument des vorangehenden Beispiels, welches in *RStudio* geöffnet werden muss und nicht um das Derivat, z. B. das gezeigte HTML-Dokument. Zwei Vorteile müssen hervorgehoben werden: 1. Die Studierenden brauchen den Code lediglich ausführen und gewinnen somit Zeit, diesen zu lesen und zu verstehen. 2. Der Code kann bewusst unvollständig oder falsch vorliegen, vergleichbar mit einer Prüfungsaufgabe. Zudem kann der begleitende Text dazu auffordern, selbständig Variationen durchzuspielen oder die Syntax mit bereits erworbenem Wissen zu kombinieren. Der Erfolg oder der Misserfolg sind selbständig sowie unmittelbar überprüfbar und das erstellte Dokument kann abschließend wie ein ausgefüllter Fragebogen als PDF gespeichert werden. Der wesentliche Nachteil ist die notwendige Installation der Software, hier *RStudio* und R, als auch die minimale Einarbeitung in *RStudio*.

Hier kann das Gegenargument die für das wissenschaftliche Arbeiten unabdingbare Aneignung von R und ergänzend *RStudio* sein.

In den *code chunks* von *RStudio* können 65 Programmiersprachen[2] umgesetzt werden (s. auch Xie 2016). Allerdings sind neben R selbst und SQL vermutlich nur Python und Julia in der Archäologie noch von Interesse. Die Verwendung der Kommandozeile (*bash*) ist leider nur unter Linux und MacOS unkompliziert möglich. Damit entfällt die interaktive Möglichkeit, grundlegende und oft noch sehr nützliche IT-Kompetenz zu vermitteln, so z.B. Tastaturbefehle für eine Dateiliste als Regest der Archivierung, eine Stapelverarbeitung oder effiziente Systemadministration, z.B. `dir * > dateiliste.txt` oder `type text*.txt >> alle_texte.txt` oder auch `winget list --upgrade-available`. Damit ist das Nutzungsszenario gegenüber dem statischen Text, bei dem ggf. die Umsetzung des Layouts reicht (s.o.), deutlich eingeschränkt.

Jupyter Notebook oder *JupyterLab*

Die dritte, hier vorgestellte Variante ist die Verwendung von *Jupyter Notebook* (einfacher) oder *JupyterLab* (etwas komplexer). Hierbei werden Dokumente und Daten über einen Webzugang bereitgestellt, d. h. die Studierenden benötigen lediglich einen Browser und müssen keine ergänzende Software installieren. Die Teilnahme ist nicht einmal an eine physische Präsenz im Kursraum gebunden. Auch hier ergeben sich Erfolg oder Misserfolg unmittelbar durch die Ausführung der einzelnen Codezellen. Damit ist hier ebenfalls das Experimentieren mit eigenen Variationen zur Lösung der gestellten Aufgabe möglich und fördert so die kreative Erweiterung der eigenen Fähigkeiten. Allerdings liegt der nicht unerhebliche technische Aufwand vollumfänglich bei den Anbietenden, also den Lehrenden. Zudem lernen Studierende die üblicherweise verwendete Software zur realen Umsetzung der Sprache nicht kennen. Dies kann nur bedingt als Manko angesehen werden, denn die Umgebung kann je nach Nutzung von z. B. *MySQL*, *pgAdmin*, *Data Browser for SQLite*, *SpatiLiate GUI*, *QGIS*-Abfrage bzw. Filter oder der *QGIS DB-Verwaltung* stark variieren. Die Reduktion auf die eigentliche Sprache schafft hier eine willkommene Abstraktion und die Voraussetzung für die selbständige Übertragung in das jeweils verwendete Umfeld.

JupyterLab stellt eine Arbeitsumgebung im Browser zur Verfügung, die speziell für die wissenschaftliche Arbeit insbesondere mit Code entwickelt worden ist. Die Basiskonfiguration nutzt bzw. bietet die Programmiersprache Python. Darauf aufgesetzt werden aber weitere vorkonfigurierte Arbeitsumgebungen bereitgestellt, so z. B. R. Kern ist stets die Kombination von Text, ausführbarem Code und der zugehörigen

2 Dies schließt asis, asy, awk, bash, block, block2, bslib, c, cat, cc, coffee, comment, css, ditaa, dot, embed, eviews, exec, fortran, fortran95, gawk, go, groovy, haskell, highlight, js, julia, lein, mysql, node, octave, perl, php, psql, python, R, Rcpp, Rscript, ruby, sas, sass, scala, scss, sed, sh, sql, stan, stata, targets, tikz, verbatim und zsh ein.

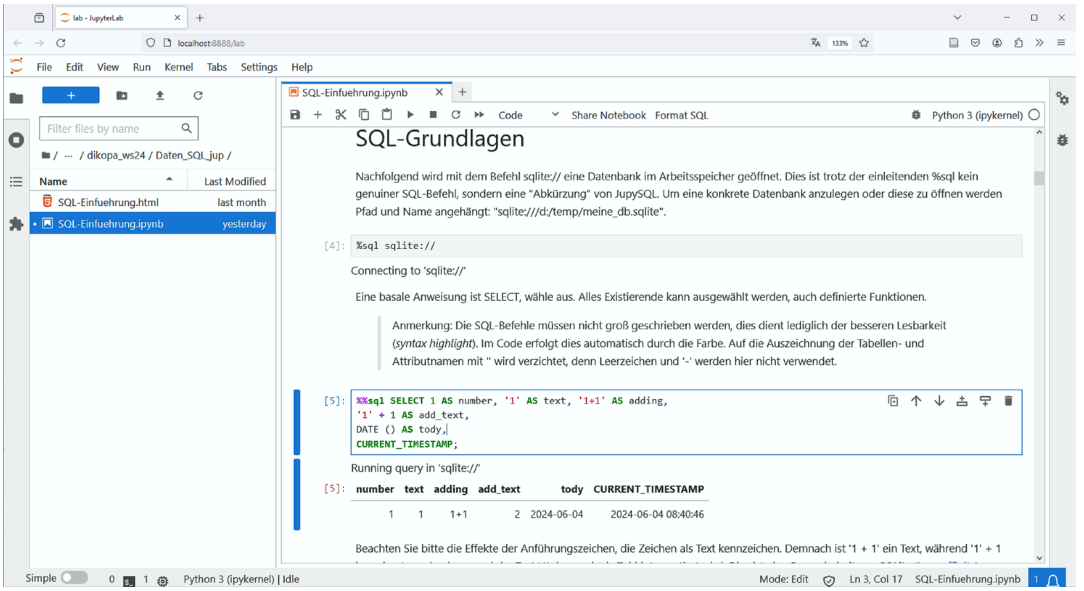

Abb. 2: Ansicht eines Übungsbogens in *JupyterLab*.

Rückgabe, z. B. einer Graphik. Das Ziel ist dabei vollkommen offen: wissenschaftlicher Text, (Labor-)Protokoll, Visualisierung der Auswirkung von veränderbaren Parametern in einer Funktion mittels dynamischer Elemente wie Schiebereglern oder auch Übungsblättern. Einen Überblick mit konkreten Testmöglichkeiten bietet die *Jupyter*-Website.[3]

Das Rechenzentrum der Universität Kiel bietet im Rahmen seines Cloudlab-Projektes *JupyterLab* für die Lehre an.[4] Die jeweilige technische Ausgestaltung erfolgt in Abstimmung mit den Anforderungen der Lehrenden, die lediglich den jeweiligen Kursinhalt liefern müssen. In Vorbereitung für eigene Inhalte zu SQLite und SpatiaLite wurde eine lokale Lösung mit der Installation des Service auf dem Dozierenden-PC und der exklusiven Nutzung innerhalb des PC-Labors des Instituts für Ur- und Frühgeschichte in Kiel getestet. Ein erstes Python-Umfeld (*environment*) mit installiertem *JupyterLab* und dem integrierten Modul für Raumabfragen (SpatiaLite) funktionierte anfänglich vielversprechend, scheiterte jedoch nach der Aktualisierung des verwendeten Paketes ipython-sql. Letzteres wird für die Verwendung der sog. „*magic*"-Syntax benötigt. Ein „`%sql`" bzw. „`%%sql`" Befehl am Zeilenanfang reicht aus, um dem genuin laufenden Python mitzuteilen, dass es sich nachfolgend um eine SQL-Anweisung handelt. Das folgende Beispiel verweist deshalb ausschließlich auf SQLite und orientiert sich aktuell an einer Installation des Docker Containers quay.io/jupyter/base-notebook[5].

3 https://jupyter.org [zugegriffen am 06.02.2025].

4 https://www.rz.uni-kiel.de/de/angebote/hiperf/jupyterlab_in_der_forschung [zugegriffen am 06.02.2025].

5 https://docs.docker.com/guides/jupyter [zugegriffen am 06.02.2025].

Zukünftig sollen Container mit speziellen Erweiterungen als spezifische Container bzw. gespeicherte Images für diverse Nutzungsszenarien vorgehalten werden.

In der Anwendung laden die Nutzerinnen und Nutzer in der *JupyterLab*-Umgebung in ihrem Browser ein vorbereitetes Dokument, z. B. SQL-Einfuehrung.ipynb **(Abb. 2)**. Dieses speichert textbasiert den Inhalt und die jeweiligen Zustände der integrierten Textbestandteile und Codezellen. Die Ansicht im Browser ist weitgehend intuitiv bedienbar, jede Zelle kann mit einem „Play"-Icon in der Menüleiste ausgeführt werden und der Fokus springt zugleich zum nächsten Absatz. In einer Codezelle können dann Beispiele, bewusst unvollständige oder fehlerhafte Anweisungen oder auch gar kein Code für die selbständige Formulierung einer Anweisung enthalten sein. Das vollständig ausgefüllte Formular kann abschließend in diverse Formate, z. B. ein PDF, exportiert werden. Da in einem einleitenden Abschnitt spezifische Pakete erst in dem im Hintergrund laufenden Pythonskript geladen werden müssen, empfiehlt sich eine betreute Einführung in die Handhabung von *JupyterLab*. Ein weiteres Spezifikum ist die bereits erwähnte „*magic*"-Syntax mit dem Befehl „`%sql`" bzw. „`%%sql`" am Zeilenanfang, wodurch die Pythonumgebung die nachfolgenden Befehle als SQL-Syntax erkennt. Zudem misslich und etwas umständlich ist die Ausführung der jeweiligen Zelle durch das „Play"-Icon in der Menüleiste, was zu einem permanenten Wechsel zwischen Tastatur und Maus führt.

Fazit

Die im Vorangehenden dargestellte Arbeitsweise verwendet einerseits spezialisierte Tools, andererseits werden zugleich mit Markdown, der Kombination von plattformunabhängigem Text und ausführbarem Code Grundlagen des wissenschaftlichen Arbeitens behandelt. Es wurden drei Versionen einer Übung zu der Datenbanksprache SQLite dargestellt, die jeweils Vor- und Nachteile bieten. Das statische, aus einem RMarkdown-Dokument generierte Skript kann als HTML-Dokument auf jedem Endgerät gelesen werden. Die benötigte Software muss auf dem eigenen Gerät selbständig installiert werden, um die Befehle aus dem Skript dann zu übertragen. Dies bedingt einen höheren Arbeits- und Zeitaufwand, vertieft damit aber voraussichtlich auch das Lernergebnis. Im gezeigten Beispiel können SQL-Anweisungen auch im Kontext des RMarkdown-Dokumentes in *RStudio* ausgeführt werden. Dies beschleunigt das Durchgehen in der Übung, findet dabei aber nicht in der genuinen Software, z. B. der *QGIS DB-Verwaltung* statt. Der positive Nebeneffekt ist die sicher wünschenswerte Einführung in *RStudio* und die Schnittstelle zwischen einem guten Datenrepositorium (SQLite) und der sehr guten Nachvollziehbarkeit eines R-Skripts bei wissenschaftlichen Analysen. Die dritte Variante, das *Jupyter*-Dokument, ist eindeutig den

spezialisierten Tools zuzuweisen. Hier werden die Lerninhalte über einen Server in einem Netzwerk für den Browser bereitgestellt. Die Anforderungen an die Lernenden sind also auf das Öffnen des Browsers reduziert und sie können sich vollständig dem Inhalt widmen. Die Entkopplung von der zugehörigen Software ist dabei zugleich das größte Problem, denn die Einführung in eben diese ist somit obsolet. Zudem ist dies mit einem deutlich erhöhten technischen Aufwand auf der Seite der Lehrenden verbunden. Die Kompetenz der Lehrenden, die technischen Voraussetzungen der jeweiligen Institution sowie die Lerninhalte sind die entscheidenden Faktoren für eine Wahl zwischen den hier vorgestellten oder weiteren Möglichkeiten zur Vermittlung digitaler Inhalte in der Archäologie.

ORCID®

Christoph Rinne https://orcid.org/0000-0002-9829-6182

Referenzen

Kruschke, John K. (2020): *Bayesian Analysis Reporting Guidelines*, [online] https://osf.io/w7cph [zugegriffen am 23.07.2024].

Marwick, Ben (2017): Computational Reproducibility in Archaeological Research: Basic Principles and a Case Study of Their Implementation, in: *Journal of Archaeological Method and Theory*, Bd. 24, Nr. 2, 424–450, [online] https://www.jstor.org/stable/26748313 [zugegriffen am 23.07.2024].

Rinne, Christoph (2024): *Neolithic life tables de 2024.02*, [online] https://doi.org/10.5281/zenodo.10778019.

Xie, Yihui (2016): *Dynamic documents with R and knitr. The R series*, Boca Raton: CRC Press.