Python oder *R*? Einstieg zum Programmieren in den Geisteswissenschaften

William Mattingly

Abstract In diesem Beitrag wird beschrieben, wie Programmiersprachen wie *Python* und *R* den Geisteswissenschaften neue Forschungsmöglichkeiten eröffnen, indem sie große Textkorpora analysieren, Muster in Daten visualisieren und repetitive Aufgaben automatisieren können. *Python* und *R* sind wohl die prominentesten Programmiersprachen, um sich in den Digital Humanities zu engagieren. *Python* bietet durch seine einfache Syntax und vielseitigen Bibliotheken Vorteile für die Textanalyse und maschinelles Lernen, während *R* mit seinen statistischen Funktionen und visuellen Darstellungsmöglichkeiten für Datenmanipulationen punktet. Die Wahl zwischen *Python* und *R* hängt somit von den spezifischen Forschungsanforderungen ab, wobei beide Sprachen durch ihre starken Communities und umfangreichen Ressourcen für die Geisteswissenschaften gut geeignet sind. Auch Lernstrategien zum Einstieg in das Programmieren und der Umgang mit möglichen Fallstricken werden thematisiert.*

Keywords Programmiersprachen, *Python*, *R*, Geisteswissenschaften, Digitalisierung, Digitale Lehre

1. Einleitung

Programmieren ist eine grundlegende Fähigkeit in den Naturwissenschaften und der Mathematik. In diesen Disziplinen dienen Programmiersprachen wie *Python* und *R* als Hilfsmittel für die Forschung. Sie ermöglichen es Wissenschaftler*innen, große Datensätze zu verarbeiten, komplexe Simulationen durchzuführen und repetitive Aufgaben zu automatisieren. Eine Biologin könnte z. B. *Python* verwenden, um Milliarden von Gensequenzen vieler Organismen zu analysieren oder ein Statistiker könnte *R* verwenden, um Muster in Daten zu modellieren und zu visualisieren. In diesem Kapitel werden wir untersuchen, wie sich dieses Konzept auf geisteswissenschaftliche Daten übertragen lässt.

* Dieses Kapitel wurde inkl. fremdsprachiger Zitate von der Redaktion aus dem Englischen übersetzt.

William Mattingly: *Python* oder *R*? Einstieg zum Programmieren in den Geisteswissenschaften, in: Christopher A. Nunn und Frederike van Oorschot (Hrsg.): Kompendium Computational Theology 1. Forschungspraktiken in den Digital Humanities, Heidelberg: heiBOOKS, 2024, S. 193–204. DOI: https://doi.org/10.11588/heibooks.1459.c21911

2. Daten und die Geisteswissenschaften

Im Gegensatz zu den Naturwissenschaften werden die Geisteswissenschaften traditionell mit der qualitativen Analyse in Verbindung gebracht und legen Wert auf eine genaue Lektüre des Quellenmaterials. Für den größten Teil des zwanzigsten Jahrhunderts waren die meisten geisteswissenschaftlichen Daten in analoger, d. h. in gedruckter Form verfügbar. Die Methoden, die wir für die Analyse unserer Daten entwickelt haben, waren daher auf die praktischen Grenzen unserer Daten und der menschlichen Anatomie ausgerichtet. Selbst die eifrigsten Wissenschaftler*innen können unmöglich das gesamte Korpus der lateinischen Literatur in einem einzigen Leben lesen. Angenommen, man wäre dennoch dazu in der Lage, müsste man es sich immer noch physisch beschaffen. Bei Handschriften sind diese Ressourcen über verschiedene Kontinente verstreut und befinden sich in Institutionen, die nur begrenzt zugänglich sind. Selbst wenn man diese physischen Beschränkungen überwinden könnte, müsste man das Material in einer sinnvollen Weise zusammenfassen. Wie können wir all diese Informationen aufbewahren und dieses Wissen praktisch an ein Publikum weitergeben? Wir können es einfach nicht.

In der zweiten Hälfte des zwanzigsten Jahrhunderts änderte sich das Medium zur Übermittlung von Daten. Text konnte nun in Form numerischer Daten wiedergegeben werden. Zunächst geschah dies mit Hilfe von Lochkarten, wie sie im *Index Thomisticus* verwendet wurden, dem ersten Projekt, das die gesammelten Werke von Thomas von Aquin (ca. 10 Millionen Wörter) digital wiedergab (Busa 1980; vgl. auch den Beitrag von J. Peters in diesem Band, S. 342–343). Da die digitalen Daten auf physisch greifbaren Lochkarten wiedergegeben wurden, war auch nach wie vor ein physischer Zugriff auf diese erforderlich. Diese konnten jedoch, anders als in gedruckter Form, übertragen und auf verschiedene Maschinen in der ganzen Welt geladen werden.

Mit voranschreitender Technologie konnten Daten nicht mehr nur auf Lochkarten, sondern auf Disketten gespeichert werden. Da diese Disketten immer kleiner wurden und große Datenmengen speichern konnten, ließen sie sich leichter transportieren und billiger reproduzieren. In den 1990er Jahren entwickelte sich der *Index Thomisticus* mit dieser Technologie weiter und stellte seine Daten auf CD-ROMs zur Verfügung (vgl. The Economist 2020). Dies bedeutete, dass Forschende identische Daten erwerben und unabhängig von ihrem Standort auswerten konnten. Diese Daten konnten vervielfältigt, verbreitet und von einem größeren Publikum rechnerisch ausgewertet werden.

In den letzten zwei Jahrzehnten haben wir nicht nur größere Mengen digitaler Daten produziert, auch die Zugänglichkeit hat sich dank der Einführung des Internets und der Cloud verbessert. Die Cloud ermöglicht es, dass die Daten auf einem Server an einem Ort liegen und von jemandem auf der anderen Seite des Globus abgerufen werden können. So können wir z. B. die gesamte *Patrologia Latina* von einem Strand in Florida aus über das frei zugängliche Projekt *Corpus Corporum* der Universität

Zürich[1] studieren. Diese Technologie ist heute so allgegenwärtig, dass es manchmal schwierig ist, wahrzunehmen, wie unglaublich diese Leistung wirklich ist.

Heutzutage verschwinden die Grenzen der Quantität und des Zugangs allmählich. Da sich die geisteswissenschaftliche Forschung in den Bereich der *Big Data* ausweitet, wird die Fähigkeit, diese Informationen zu verarbeiten und zu interpretieren, entscheidend. Was können wir mit diesen Daten tun? Wie können wir systematisch auf sie zugreifen? Wie nutzen wir sie, um Fragen zu formulieren und dieses Wissen in etwas Nützliches zu übersetzen? Wie wir im Laufe dieses Kapitels sehen werden, bietet uns das Programmieren potenzielle Lösungen für diese Fragen. Programmieren ist auch eine aufkommende Fähigkeit, die Fachbereiche in ihre Lehrpläne aufnehmen, wie z. B. im Rahmen des neuen Promotionsstudiengangs in *Digital History* an der Clemson University.

Mit Hilfe von Programmiersprachen wie *Python* und *R* können Geisteswissenschaftler*innen große Textkorpora analysieren, komplexe Beziehungen in Daten visualisieren und Muster aufdecken, die manuell unmöglich zu finden wären. Ob es sich um die Untersuchung der gesamten gesammelten Werke von Augustinus (gest. 430) oder um die Kartierung des sozialen Netzwerks von Alkuin (gest. 804) handelt, das Programmieren eröffnet der geisteswissenschaftlichen Forschung eine ganz neue Welt der Möglichkeiten. In den Geisteswissenschaften werden Programmiersprachen, ähnlich wie in der Mathematik und den Naturwissenschaften, nicht unbedingt zur Erstellung von Software verwendet; Skriptsprachen wie *Python* und *R* fungieren als Werkzeuge, mit denen wir große Datenmengen analysieren, interpretieren und die Ergebnisse visualisieren können. Im Gegensatz zum Einsatz von Software können Forscher*innen mit Skriptsprachen vom schnellen Nutzen des Programmierens ohne die für die Wartung von Software erforderliche langfristige Nachhaltigkeit profitieren.

3. Warum programmieren lernen?

Eine der wichtigsten Fragen, die wir uns stellen können, lautet: *Warum sollte man programmieren lernen?* Hinter dieser Frage verbergen sich zwei weitere Fragen: *Welchen Nutzen hat das Programmieren und welchen unmittelbaren Vorteil hat es für mich?* Um diese Fragen zu beantworten, wollen wir uns ein Problem vorstellen. Stellen wir uns vor, wir müssten alle namentlich genannten Personen in den Schriften von Augustinus identifizieren. Wir könnten natürlich Monate oder Jahre damit verbringen, jede Person durchzugehen und manuell zu markieren. Das wäre zeitaufwendig und repetitiv. Wenn wir jedoch programmieren könnten, wäre dieses

1 S. https://mlat.uzh.ch, zuletzt aufgerufen am 22.06.2024.

Problem viel einfacher zu lösen und die Lösung könnte in wenigen Stunden (oder Tagen, je nach Komplexität) entwickelt werden. Wir könnten entweder eine Reihe von Regeln für alle Personen aufstellen, die wir in den Briefen zu finden erwarten oder (falls wir dies nicht wüssten) eine maschinelle Lernlösung entwickeln, die die Merkmale der Wörter, die den Personen entsprechen, erlernt und sie automatisch für uns klassifiziert. Diese Aufgabe wird als *Named-Entity-Recognition* bezeichnet (vgl. den Beitrag von E. Gius in diesem Band).

Dieser Prozess, bei dem wir eine Lösung schaffen, die wiederholt auf ähnliche Daten angewendet werden kann, wird als Automatisierung bezeichnet. Automatisierung ist einer der wichtigsten Gründe, um programmieren zu lernen. Als Menschen sind wir sehr schlecht darin, dieselbe Aufgabe wiederholt und konsequent auszuführen. Computer hingegen sind perfekt in diesen beiden Aufgaben. Programmieren zu lernen bedeutet zum Teil, zu lernen, Aufgaben zu automatisieren und das ist einer der Hauptvorteile. Wissenschaftler*innen können hierdurch mehr Zeit für die Forschung aufwenden und weniger Zeit für repetitive Aufgaben.

Neben der Automatisierung bietet uns das Programmieren auch die Möglichkeit, Lösungen zu entwickeln, die in großem Maßstab funktionieren; das bedeutet, dass wir in der Lage sind, praktisch dieselbe Lösung für Millionen von Daten durchzuführen. Wenn wir alle Personen in einem Text mit 10 000 Wörtern in fünf Sekunden identifizieren können (eine angemessene Zeitspanne), dann könnten wir diesen Prozess für Millionen von Texten leicht in nur wenigen Stunden wiederholen, je nach Rechenleistung.

Mit Programmierkenntnissen können wir auch Lösungen für das maschinelle Lernen mit wenigen Zeilen Code entwickeln und anwenden. Stellen wir uns vor, wir müssten dasselbe Problem mit den Werken des Hieronymus lösen. Hier könnten wir die maschinelle Lernlösung, die wir für die Werke von Augustinus entwickelt haben, auf die Werke von Hieronymus anwenden und hätten vergleichbare Ergebnisse.

Durch Programmieren können Geisteswissenschaftler*innen weit mehr erreichen als nur neue Lösungen zu entwickeln. Es verändert die Art und Weise, wie wir Probleme angehen, grundlegend. Die Kenntnis des Programmierens bringt das Wissen um Lösungen oder potenzielle Lösungen für unbekannte Probleme (oder nicht gestellte Fragen) mit sich. Diejenigen, die über Programmierkenntnisse verfügen und wissen, was damit möglich ist, können neue Fragen stellen und neue Forschungsrichtungen einschlagen. Es ermöglicht, Studien in einem Umfang und einer Tiefe durchzuführen, die zuvor unvorstellbar waren. Mit Hilfe des Programmierens können Forschende Muster und Trends in großen Datenbeständen erkennen, was zu neuen Einsichten und Erkenntnissen führt.

Die Wahl der richtigen Programmiersprache hängt weitgehend von der jeweiligen Problemstellung ab. Wenn jemand eine Website mit benutzerdefinierter Funktionalität erstellen muss, ist das Erlernen von HTML, *JavaScript* und *React* (eine Methode zur einfachen Erstellung von *JavaScript*-Komponenten) vielleicht am sinnvollsten. Wenn Forscher*innen in irgendeiner Weise mit Daten arbeiten und diese

manipulieren müssen, gibt es zwei dominante Sprachen, die in Betracht gezogen werden sollten: *Python* und *R*.

4. Die Vorteile von *Python* und *R*

Python und *R* haben sich zu den wichtigsten Programmiersprachen in vielen Bereichen entwickelt, insbesondere in den Bereichen Datenwissenschaft, maschinelles Lernen, Verarbeitung natürlicher Sprache (orig. „Natural Language Processing", NLP), Statistik, Naturwissenschaften, Mathematik und Geisteswissenschaften. *Python* wurde in den 1980er Jahren von Guido van Rossum mit der Idee entwickelt, dass Codes leicht lesbar und einfach zu korrigieren sein sollten, mit einer Syntax oder einem Schreibstil, der prägnant und einfach ist (van Rossum & Drake 1995). Im Gegensatz zu *Python* steht *R*, das von Ross Ihaka und Robert Gentleman in den 1990er Jahren entwickelt wurde (R Core Team 2021). Anders als *Python* wurde *R* ausschließlich für die statistische Analyse entwickelt. Seine Syntax ist etwas unkonventionell, aber es bietet den User*innen die Möglichkeit, statistische Methoden einfach auf quantitative Daten anzuwenden und die Ergebnisse zu visualisieren.

Beide Sprachen haben vor allem aufgrund ihrer hervorragenden und aktiven Communities an Popularität gewonnen. Diese Communities schreiben Pakete oder Bibliotheken für jede Programmiersprache. Ein Paket ist eine Sammlung von Klassen und Funktionen (stellen Sie sich diese als große Codeblöcke vor), die von anderen Mitgliedern der Community genutzt werden können. Dies bedeutet, dass neue Anwender*innen der Programmiersprache komplexe Aufgaben mit sehr wenig Code erledigen können, was sie zu idealen Programmiersprachen für den Einstieg macht. So können z. B. Studierende, die neu in *Python* sind, ein *Machine Learning* Modell herunterladen und *Named Entities* im gesamten Korpus von Thomas Moore identifizieren (zu NER s. den Beitrag von E. Gius in diesem Band). Dies ist der NLP-Bibliothek *spaCy*[2] und den Beiträgen von Patrick J. Burns zu diesem Projekt zu verdanken, der *LatinCy* entwickelt hat (Burns 2023). Wenn man an Transformer-Modellen für *Topic Modeling*, einem neueren Ansatz dieser jahrzehntealten Methodik, interessiert ist, lässt sich dies mit nur zwei Zeilen Code unter Verwendung von *BERTopic* tun (vgl. den Beitrag von M. Althage in diesem Band, v. a. S. 280, Anm. 18). Obwohl weitere methodische Schritte notwendig sind, um den Forschungsansatz zu verfeinern, wie z. B. Themenidentifizierung, Validierung, Anpassung der Hyperparameter, bleibt der dafür erforderliche Code minimal. Der Grund dafür ist, dass *BERTopic* viele fortgeschrittene Methoden nacheinander für die User*innen anwendet. Es ermöglicht sogar die schnelle Visualisierung eines Themenmodells. Beiträge wie diese machen *Python* und *R* sowohl für Personen mit geringer als auch mit fortgeschrittener Programmiererfahrung attrak-

2 S. https://spacy.io, zuletzt aufgerufen am 22.06.2024.

tiv. Diese Attraktivität wiederum fördert eine gesunde Community, die immer weiter wächst. Je größer die Community wird, desto mehr Mitglieder fügen ihre eigenen Pakete hinzu. Im Laufe der Zeit wiederholt sich dieser Kreislauf immer wieder.

5. *Python* und *R* für die geisteswissenschaftliche Forschung im Vergleich

Die Bedeutung von *Python* und *R* für die geisteswissenschaftliche Forschung ist vielfältig. Die Simplizität und Lesbarkeit von *Python* machen es zu einem hervorragenden Ausgangspunkt für Geisteswissenschaftler*innen, die gerade erst mit Programmieren angefangen haben. Die umfangreichen Bibliotheken wie *Pandas*[3] für die Datenmanipulation, *spaCy* und das *Natural Language Toolkit* für NLP sowie *Matplotlib*[4] oder *Seaborn*[5] zur Visualisierung bieten wertvolle Werkzeuge für verschiedene geisteswissenschaftliche Forschungsaufgaben. Auf der anderen Seite eignet sich *R* aufgrund seiner mächtigen Datenverarbeitungsmöglichkeiten, seiner bereits integrierten breiten Palette statistischer Anwendungen sowie seiner leistungsstarken Visualisierungsbibliotheken besonders für Geisteswissenschaftler*innen, die viel mit statistischen Daten arbeiten oder komplexe Visualisierungen erstellen müssen. Wenn es um die Wahl zwischen *Python* und *R* für geisteswissenschaftliche Forschung geht, hängt die Entscheidung oft von persönlichen Vorlieben, spezifischen Projektanforderungen und der Art der Daten ab, mit denen gearbeitet werden soll. Sowohl *Python* als auch *R* haben ihre Stärken und sind einsatzfähige Werkzeuge für den Umgang mit geisteswissenschaftlichen Daten.

Python hat einige entscheidende Vorteile. Erstens entspricht seine Syntax anderen Programmiersprachen und ist im Allgemeinen einfacher für Programmieranfänger*innen zu handhaben. Zweitens ermöglicht *Python* seinen Anwender*innen, mithilfe von Bibliotheken wie *Django*[6] und *Flask* (s. Grinberg 2018) schnell Websites zu erstellen. Mit *Streamlit*[7] können *Python*-Anfänger*innen mit nur wenigen Zeilen Code eine benutzerdefinierte datenbasierte Anwendung erstellen und in die Cloud stellen. Drittens ist *Python* in der Community für maschinelles Lernen oft die erste Wahl. Das bedeutet, dass die meisten neuen Entwicklungen in diesem Bereich zuerst in *Python* verfügbar sind. Viertens werden die meisten Fortschritte bzgl. NLP oft in *Python* entwickelt, was es ideal für Aufgaben wie Textklassifizierung, *Topic Modeling* und *Named Entity Recognition* macht.

3 S. https://pandas.pydata.org, zuletzt aufgerufen am 22.06.2024.
4 S. https://matplotlib.org, zuletzt aufgerufen am 22.06.2024.
5 S. https://seaborn.pydata.org, zuletzt aufgerufen am 22.06.2024.
6 S. https://djangoproject.com, zuletzt aufgerufen am 22.06.2024.
7 S. https://streamlit.io, zuletzt aufgerufen am 22.06.2024.

R hingegen wurde für die Statistik entwickelt. Es hat mehrere Vorteile gegenüber *Python.* Erstens sind die Syntax und die Funktionalität von *R* auf die statistische Modellierung zugeschnitten und ermöglichen komplexe Analysen mit prägnantem Code. Zweitens bietet *R* eine umfangreiche Sammlung von Paketen wie *ggplot2* (Wickham 2016) und *Shiny*[8], die eine hochwertige Datenvisualisierung und interaktive Webanwendungen ermöglichen. Während *Python* mit guten Visualisierungsbibliotheken wie *Plotly*[9] und *Seaborn* (Waskom et al. 2017) aufwarten kann, sind die Visualisierungen in *R* einfacher zu erstellen, sehen in der Regel schöner aus und lassen sich leichter anpassen. Das bedeutet, dass man nicht nur Daten analysieren, sondern auch mit Leichtigkeit visuell ansprechende Darstellungen erstellen kann. Drittens: Die Integration von *R* in verschiedene Datenquellen und die Möglichkeiten der Datenmanipulation durch Pakete wie *dplyr* machen *R* zu einem leistungsstarken Werkzeug für die Datenverarbeitung. Viertens: Obwohl die Möglichkeiten von *R* zum maschinellen Lernen nicht so umfangreich sind wie die von *Python,* bieten Pakete wie *caret*[10] und *randomForest*[11] dennoch robuste Werkzeuge zu diesem Zweck an.

Beide Sprachen haben aktive und hilfsbereite Communities, sodass in beiden Fällen reichlich Ressourcen und Hilfe gefunden werden kann.

Die Entscheidung zwischen *Python* und *R* hängt von den spezifischen Bedürfnissen und Zielen der jeweiligen Forschung ab. Es gibt drei Leitaspekte, die zu berücksichtigen sind. Zunächst sollten die eigenen Forschungsbedürfnisse berücksichtigt werden. Wenn viel Textanalyse oder NLP zu erwarten ist, könnte *Python* aufgrund von Bibliotheken wie *spaCy* die bessere Wahl sein. Wenn die Arbeit umfangreiche statistische Analysen erfordert oder detaillierte Visualisierungen erstellt werden müssen, ist *R* vielleicht die bessere Wahl. Zweitens sollte der eigene Lernstil und bisherige Erfahrungen im Programmieren berücksichtigt werden. Für Geisteswissenschaftler*innen, die noch gar keine Berührungspunkte mit Programmiertechniken aufweisen, ist *Python* vielleicht einfacher zu erlernen. Wichtig ist, sich ein paar Codeschnipsel in *Python* und *R* anzusehen, um ein Gefühl dafür zu entwickeln, wie unterschiedlich diese beiden Sprachen sind. Drittens sollten die Communities beachtet werden, die hinter beiden Programmiersprachen stehen. Diese können bei Herausforderungen wertvolle Hilfe leisten. Sowohl *Python* als auch *R* verfügen über starke Communities, aber je nach Fachrichtung hat die eine Sprache möglicherweise mehr relevante Ressourcen und Diskussionsforen als die andere.

Letztendlich gibt es keine endgültige richtige oder falsche Wahl zwischen *Python* und *R* für die geisteswissenschaftliche Forschung. Es geht darum, das Werkzeug zu

8 S. https://shiny.posit.co, zuletzt aufgerufen am 22.06.2024.

9 S. https://plotly.com/python, zuletzt aufgerufen am 22.06.2024.

10 S. https://cran.r-project.org/web/packages/caret/vignettes/caret.html, zuletzt aufgerufen am 22.06.2024.

11 S. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf, zuletzt aufgerufen am 22.06.2024.

wählen, das den eigenen Bedürfnissen am besten entspricht und die eigene Forschung sinnvoll ergänzt. Geisteswissenschaftler*innen, die im Rahmen ihrer Forschung programmieren möchten, werden wahrscheinlich lernen, in beiden Sprachen zu schreiben, aber i. d. R. eine der beiden Sprachen bevorzugen. Das liegt daran, dass die meisten Dinge, die in einer dieser Sprachen erledigt werden, auch in der anderen vollbracht werden können, selbst wenn es vielleicht nicht so einfach ist. Wenn z. B. ein Projekt komplett in *Python* geschrieben wurde, aber zum Schluss noch eine ansprechende Visualisierung erstellt werden muss, macht es vielleicht keinen Sinn, *R* für diesen einzigen Schritt in den Arbeitsablauf einzuführen. Stattdessen könnte weiterer Code geschrieben und die *Python*-Bibliothek *Seaborn* verwendet werden. Ferner, wenn Ergebnisse im Bereich der Textanalyse und Statistik zu präsentiert sind, ist es vielleicht wenig ratsam, *Python* und *SpaCy* zur Lemmatisierung oder zur Reduzierung aller Wortformen auf ihren Stamm zu verwenden.

6. Erste Schritte in der Programmierung

Einer der schwierigsten Aspekte beim Erlernen des Programmierens ist die Installation der Programmiersprache auf dem Computer. Jedes Betriebssystem wie Mac, Windows oder eine Linux-Distribution (wie Ubuntu), erfordert eine andere Installation der Sprache. Jedes Betriebssystem hat seine eigenen Schritte. Unter Windows muss z. B. sichergestellt werden, dass *Python* im PATH des Systems liegt (oft manuell). Jedes Betriebssystem bringt auch kleine, aber entscheidende Unterschiede mit sich. Auf einigen Macs ist z. B. *Python* 2 auf dem System vorinstalliert. Das bedeutet, nachdem die neueste Version von *Python* (zum Zeitpunkt der Abfassung dieses Beitrags ist es *Python* 3.12) installiert wurde, befinden sich zwei Versionen der gleichen Programmiersprache auf dem Computer. Folglich muss der Befehl „Python3“ in der Befehlszeile verwendet werden, um eine *Python*-Datei auf einigen Macs auszuführen. Unter Windows und Linux hingegen genügt „Python“ als Befehl. Viele dieser Probleme lassen sich jedoch umgehen, wenn man virtuelle Umgebungen oder *Conda*[12] verwendet.

Die Installation der Programmiersprache ist nur eine Hürde. User*innen möchten in der Regel auch eine benutzerdefinierte Methode zur Interaktion mit der Programmiersprache installieren. Für *Python* bedeutet dies in der Regel die Installation einer integrierten Entwicklungsumgebung (orig. „integrated development environment“, IDE) wie *JupyterLab* oder *VS Code.* Für *R* bedeutet dies die Installation von *R Studio.* Mit diesen Tools können Sie Codes in einem einzigen Bereich schreiben und ausführen. Sie erleichtern Ihnen das Lernen und definieren auch die Art und Weise, wie Sie sich normalerweise mit einer Programmiersprache beschäftigen. Die Instal-

12 S. https://docs.conda.io/en/latest, zuletzt aufgerufen am 22.06.2024.

lation und Einrichtung einer IDE hängt von den jeweiligen Systemanforderungen ab. Auch dies ist ein Schritt, der zu Verwirrung und Problemen führen kann.

Meiner Erfahrung nach können die Frustrationen, die Studierende während dieses Prozesses erleben, davon abhalten, überhaupt programmieren zu lernen. Dies stellt einige der wichtigsten Augenblicke in ihrer potentiellen Programmierlaufbahn dar. Zu diesem Zeitpunkt sind sie neugierig auf das Programmieren und wollen unbedingt etwas lernen. Solche Frustrationen können das Licht ihrer Neugierde schnell verdunkeln. Um dies zu vermeiden, empfehle ich allen Studienanfänger*innen, die Installation der Programmiersprache und der IDE ganz zu überspringen. Stattdessen gibt es zahlreiche Unternehmen, die cloudbasierte Lösungen für diese Probleme anbieten. Sie ermöglichen es, aus der Ferne auf einen Server zuzugreifen und *Python* über den Browser auszuführen.

Constellate von ITHAKA[13] ist eine solche Lösung, die allerdings einen institutionellen Zugang erfordert. Es bietet Studierenden eine virtuelle Umgebung mit genügend Ressourcen, um sogar maschinelles Lernen zu betreiben, wenn sie dies wünschen. Es ist mit Bibliotheken vorinstalliert, die häufig für geisteswissenschaftliche Projekte benötigt werden. Auf jeder Instanz ist außerdem *JupyterLab* vorinstalliert, eine IDE, die die Datenverwaltung und das Lernen (über *Jupyter*-Notebooks) erleichtert. Das macht es ideal für den Einsatz im Unterricht. Ich habe drei Jahre lang mit *Constellate* unterrichtet und kann es nur empfehlen.

Nicht alle Studierenden haben jedoch institutionellen Zugang zu *Constellate.* In diesem Fall gibt es vergleichbare Dienste. Der populärste ist *Google Colab*[14], das mehrere Stufen hat, darunter eine kostenlose Version. Es kann mit *Google Drive* verknüpft werden. Das heißt, Daten können auf *Google Drive* hochgeladen werden. Es kann mit ihnen interagiert werden, sie können bearbeitet und gespeichert werden. Die kostenlose Version neigt dazu, gelegentlich abzustürzen, eignet sich aber dennoch gut für den Einstieg.

7. Ressourcen zum Lernen

Programmieren lernen als Geisteswissenschaftler*in ist oft ein individuelles Unterfangen. Selbst wenn eine formale Ausbildung in *Python* in einem College-Klassenzimmer erworben wurde, muss man sich für den Rest seiner Programmierkarriere auf seine Fähigkeit zum Selbststudium verlassen. Der Grund dafür ist, dass Daten und Probleme in der realen Welt unübersichtlich sind. Klare und einfache Lösungen sind selten zu finden. Man muss darauf vorbereitet sein, neue Aspekte einer Programmiersprache zu erlernen, um neuartige Probleme zu lösen, sobald sie auftauchen.

13 S. https://constellate.org, zuletzt aufgerufen am 22.06.2024.
14 S. https://colab.google, zuletzt aufgerufen am 22.06.2024.

Hat man z. B. gelernt, in *Python* zu programmieren, um *Named Entity Recognition* zu betreiben, braucht man jetzt eine Möglichkeit, die Named Entities in einem Netzwerkdiagramm zu visualisieren. Wie kann man das tun? In diesem Fall wäre es an der Zeit, *NetworkX*[15] zu lernen, um die Daten zu sammeln und zusätzlich entweder *Matplotlib* oder *PyVis*[16], um sie zu visualisieren.

Da *R* und *Python* über große Communities und eine Vielzahl von Bibliotheken zur Lösung gängiger Probleme verfügen, gibt es zahlreiche Ressourcen, um sich weiterzubilden. Eine großartige Ressource ist das Open-Source-Lehrbuch *Introduction to Cultural Analytics and Python* von Walsh (2021). Dieses Lehrbuch vermittelt nicht nur die Grundlagen von *Python,* sondern bietet auch eine Einführung in einige wichtige Methoden wie Textanalyse und Netzwerkanalyse. Das Standardlehrbuch für geisteswissenschaftliche Daten und *R* bleibt *Humanities Data in R* von Arnold & Tilton (2015). Es verfügt über Open-Source-Online-Ressourcen, die jedoch neben dem erworbenen Lehrbuch verwendet werden sollten. Eine weitere *R*-spezifische Ressource ist das Open-Source-Lehrbuch *Computational Historical Thinking: With Applications in R* von Mullen (2018). Im Gegensatz zu neueren Open-Source-Lehrbüchern (wie dem von Walsh) ist dieses nicht mit *JupyterBook* konzipiert. Nichtsdestotrotz bietet es nützliches Zusatzmaterial wie z. B. Arbeitsblätter.

Nachdem nun ein Grundverständnis von *Python* oder *R* angeeignet wurde, wird häufig eine bestimmte Bibliothek zu erlernen sein. Oft ist die Dokumentation für diese Bibliotheken spärlich. Sie wird i. d. R. von Expert*innen für fortgeschrittene Programmierer*innen geschrieben. In diesen Fällen kann es notwendig sein, ein verständliches Tutorial zu haben. Wenn Sie sich an akademische Tutorials halten wollen, ist der *Programming Historian*[17] wahrscheinlich die beste verfügbare Ressource. Derzeit gibt es 101 Lektionen sowohl für *Python* als auch für *R.* Diese drehen sich oft um ein bestimmtes Problem oder eine bestimmte Methode. Die hier veröffentlichten Lektionen sind sowohl quelloffen als auch über *GitHub* peer-reviewed. Sie sind in der Regel eher auf bestimmte Probleme oder bestimmte Bibliotheken ausgerichtet.

Einige der besten verfügbaren Ressourcen werden jedoch von Nicht-Akademiker*innen verfasst. Diese Ressourcen werden auf *YouTube* und *Medium* veröffentlicht. Der einfachste Weg, Ressourcen für ein spezifisches Problem zu finden, ist die Suche auf diesen Plattformen nach einem Thema, das von entsprechendem Interesse ist.

15 S. https://networkx.org, zuletzt aufgerufen am 22.06.2024.
16 S. https://pyvis.readthedocs.io/en/latest, zuletzt aufgerufen am 22.06.2024.
17 S. https://programminghistorian.org, zuletzt aufgerufen am 22.06.2024.

8. Häufige Fallstricke beim Programmieren

Im Laufe der Programmierkarriere werden sich viele Fallstricke auftun. Eine der häufigsten ist ein Fehler im Code. In den meisten Programmiersprachen erhält man in diesem Fall eine Fehlermeldung, die angibt, warum ein Teil des Codes nicht funktioniert hat, und die oft auf die betreffende Zeile hinweist. Wenn jedoch mit externen Bibliotheken gearbeitet und komplexerer Code geschrieben wird, kann die Fehlersuche erschwert werden. Glücklicherweise gibt es (oder gab es) eine gute Community bei *Stack Overflow,* die es den User*innen ermöglicht, Fehler zu posten und um Hilfe zu bitten. Normalerweise antwortet jemand aus der Community innerhalb von ein paar Stunden. Seit der Einführung von *ChatGPT* hat der Verkehr auf *Stack Overflow* jedoch abgenommen. Das bedeutet, dass mehr Fragen über *ChatGPT* oder einen anderen ähnlichen Dienst gestellt werden. Bei grundlegenden Programmierproblemen bietet *ChatGPT* ziemlich gute und spezifische Ratschläge zur Behebung eines Fehlers. Dafür gibt es einen guten Grund: Es wurde auf der Grundlage vieler *Stack Overflow*-Daten trainiert. Wenn *ChatGPT* oder *Stack Overflow* eingesetzt wird, ist es immer wichtig, nicht einfach die Lösung zu kopieren und einzufügen, sondern zu verstehen, *warum* der Fehler überhaupt aufgetaucht ist, so dass das Problem korrigiert *und* aus den Fehlern gelernt werden kann.

Ein weiterer häufiger Fallstrick ergibt sich aus der Verwendung von Algorithmen, die die Programmierenden nicht vollständig verstehen. Sie müssen zwar nicht wissen, wie neuronale Netze funktionieren, um sie zu verwenden und nützliche Ergebnisse zu erzielen, aber es wäre unklug, ein Argument auf die statistischen Ergebnisse eines Modells zu stützen, dessen Algorithmus der Forschende nicht versteht. Wenn man die Methoden noch nicht vollständig durchdrungen hat, sollte die Programmierung daher als Hilfsmittel für die Forschung verwendet werden, niemals als Werkzeug zur Validierung von Argumenten.

9. Fazit

Programmieren soll die traditionelle humanistische Forschung nicht ersetzen, sondern eröffnet den Geisteswissenschaftler*innen neue Wege der Forschung. Es ermöglicht uns, Fragen zu stellen, die wir sonst nicht beantworten könnten. Es ermöglicht uns, Aufgaben in wenigen Stunden zu automatisieren, die sonst Jahre dauern würden. Und es ermöglicht uns, aus großen Datenmengen neue Erkenntnisse zu gewinnen. In dem Maße, wie sich die Geisteswissenschaften weiterentwickeln und Technologie allgegenwärtig wird, werden die Geisteswissenschaftler*innen von morgen wahrscheinlich mehr technische Fähigkeiten erwerben, so wie sie es mit dem Aufkommen des Word-Prozessors taten. Das Word-Verfahren von morgen ist das Programmieren.

Literaturverzeichnis

Arnold, T., & Tilton, L. (2015). *Humanities Data in R. Exploring Networks, Geospatial Data, Images, and Text.* Cham: Springer. https://doi.org/10.1007/978-3-319-20702-5 [zuletzt aufgerufen am 22.06.2024].

Burns, P. J. (2023). LatinCy. Synthetic Trained Pipelines for Latin NLP. *arXiv.* https://doi.org/10.48550/arXiv.2305.04365 [zuletzt aufgerufen am 22.06.2024].

Busa, R. (1980). The Annals of Humanities Computing. The Index Thomisticus, *Computers and the Humanities,* 14(2), 83–90. URL: https://www.jstor.org/stable/30207304 [zuletzt aufgerufen am 22.06.2024].

The Economist (2020). How data analysis can enrich the liberal arts. URL: https://www.economist.com/christmas-specials/2020/12/19/how-data-analysis-can-enrich-the-liberal-arts [zuletzt aufgerufen am 22.06.2024].

Grinberg, M. (2018). *Flask web development. Developing web applications with python.* Sebastopol: O'Reilly Media.

Mullen, L. A. (2018). Computational Historical Thinking. With Applications in R. In *Computational Historical Thinking [Blog/Preprint].* URL: https://dh-r.lincolnmullen.com [zuletzt aufgerufen am 22.06.2024].

R Core Team (2021). R. A language and environment for statistical computing [Software]. Wien: R Foundation for Statistical Computing. URL: https://www.R-project.org [zuletzt aufgerufen am 22.06.2024].

Van Rossum, G., & Drake Jr, F. L. (1995). Python reference manual. Version 1.2. Amsterdam: Centrum voor Wiskunde en Informatica. URL: https://ir.cwi.nl/pub/5008/05008D.pdf [zuletzt aufgerufen am 22.06.2024].

Walsh, M. (2021). Introduction to Cultural Analytics & Python. Version 1. Zenodo. https://doi.org/10.5281/zenodo.4411250 [zuletzt aufgerufen am 22.06.2024].

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, M., Ram, Y., Yarkoni, T., Williams, M. L., Evans, C., Fitzgerald, C., Fonnesbeck, B. Ch., Lee, A., & Qalieh, A. (2017). mwaskom/seaborn. Version 0.8.1 [Python-Package]. Zenodo. https://doi.org/10.5281/zenodo.883859 [zuletzt aufgerufen am 22.06.2024].

Wickham, H. (2016). ggplot2. Elegant Graphics for Data Analysis. Cham: Springer [= *Use R!*]. https://doi.org/10.1007/978-3-319-24277-4 [zuletzt aufgerufen am 22.06.2024].