datorium

Einreichung von Syntaxen in datorium (Replikationsserver)

Autor: Thomas Ebel, GESIS Datenarchiv, Thomas.Ebel@gesis.org

Zitationsempfehlung: Ebel, Thomas, 2016: Einreichungen von Syntaxen in datorium (Replikationsserver), Stand: 2016-02-29, GESIS Datenarchiv für Sozialwissenschaften, verfügbar unter: http://www.replikationsserver.de/replikationsserver/home/publikationen

Stand: 2016-02-29

Inhalt

Abbildungsverzeichnis

Hinweise

Die vorliegende Handreichung soll Ihnen als Datengeber als Anleitung dienen, wie Syntaxen (z.B. SPSS, Stata, R) für die Erstellung quantitativ-statistischer, sozialwissenschaftlicher Ergebnisse idealerweise in datorium eingestellt werden, sodass sie für Nachnutzende nachvollziehbar, mit wenig Aufwand anpassbar und anwendbar sind. Das Ziel ist es, Ergebnisse wissenschaftlicher Forschung replizierbar zu machen.

Die Reproduzierbarkeit von Forschungsergebnissen bzw. das Einstellen von Syntaxen ist keine Voraussetzung für die Einreichung von Forschungsdaten in datorium. Wenn Sie als Autorin oder Autor eines Artikels der Sozialen Welt oder der Zeitschrift für Soziologie von der Redaktion aufgefordert wurden, Ihre Daten in datorium einzustellen, dann ist es u.U. möglich, dass wir auf das Einreichen von Syntaxen bestehen werden. Dies ist abhängig von den Vereinbarungen, die mit der jeweiligen Redaktionen im Rahmen des Projekts Replikationsserver (http://www.replikationsserver.de) getroffen wurden.

Dies ist explizit keine Einführung in die Thematik Programmierstile (Style Guides). Die Übersichtlichkeit von Code und dessen Ästhetik, bspw. die Art und Tiefe von Einrückungen und Namenskonventionen/-Schemata für Variablen, werden nur am Rande erwähnt und ausschließlich unter dem Aspekt der Schaffung von Transparenz und Reproduzierbarkeit wissenschaftlicher Forschung.[1] Ebenso wird Reproduzierbarkeit ausschließlich in Bezug auf Syntaxen, nicht den gesamten Forschungs- und Arbeitsworkflow, diskutiert (siehe allerdings einige kurze Hinweise in Anhang A).

Alle Syntax-Beispiele sind in R geschrieben. Die dahinterstehende Logik lässt sich aber meistens auch für andere Programmiersprachen und Statistikumgebungen wie SPSS oder Stata umsetzen. Alle Beispiele lassen sich direkt – d.h. ohne zusätzliche Installationen, Einstellungen etc. - in R ausführen.

[1] Es existieren sehr viele „style guides“, auch explizit für R. Wir verweisen exemplarisch auf den Google R Style Guide, siehe https://google.github.io/styleguide/Rguide.xml (aufgerufen 2015-12-14).

1. Definitionen

Einreicher: (hier) Person, die Syntaxen zur Replikation wissenschaftlicher *Ergebnisse* in datorium einstellt. Nicht notwendigerweise der *Urheber* der Daten.

Ergebnis (wissenschaftliches): (hier) Resultate von quantitativ-statistischen Auswertungen von Daten. Wir gehen davon aus, dass *Ergebnisse*, die in datorium eingestellt werden sollen, (in Journalen, als Monographien o.ä.) publiziert oder zumindest auf dieses Ziel hin produziert wurden.

Originaldaten/-satz: Daten, auf denen basierend wissenschaftliche *Ergebnisse* erzeugt wurden.

Wurden nach dem Abschluss der Datenerhebung (Primärdaten) bzw. nachträglich an *Originaldaten* anderer Primärforscher (Sekundärdaten) durch den *Einreicher* Aufbereitungsschritte vorgenommen, bspw. Variablen umcodiert oder neue Variablen generiert, ist es sinnvoll, weiter zwischen *Rohdaten* und *Analysedatensatz* zu unterscheiden.

A) *Rohdaten/-satz (raw data)*: Originaldaten, die keinen durch den *Einreicher* zu verantwortenden Aufbereitungsschritten unterzogen wurden.
B) *Analysedatensatz*: Daten, die durch den *Einreicher* nachhaltig verändert wurden. Analysen werden mit diesem Datensatz durchgeführt.

Idealerweise werden sowohl die *Rohdaten* als auch der *Analysedatensatz* in datorium eingereicht. Ob dies im Einzelfall möglich ist, hängt von verschiedenen Faktoren ab (siehe unten).

Reproduzierbarkeit: Wir unterscheiden für diese Handreiche drei Arten der Reproduzierbarkeit.

A) *Methodische*: Die *Syntax* ermöglicht es Sekundärnutzern, die statistischen Methoden und das methodologische Vorgehen der Primärforscher nachzuvollziehen. Die publizierten *Ergebnisse* selbst lassen sich aufgrund von Unzulänglichkeiten, bspw. fehlender *Originaldaten*, unvollständigen Angaben über deren Aufbereitung, nicht-eindeutigen Anweisungen in der *Syntax* oder zu großem Aufwand, nicht oder nur in Ansätzen replizieren.
B) *Wissenschaftliche*: Zusätzlich zur Nachvollziehbarkeit des methodischen Ansatzes ermöglicht es die *Syntax* Sekundärnutzern, unter Verwendung derselben *Originaldaten* oder einer anderen Datengrundlage, d. h. einer anderen Stichprobe der gleichen (antizipierten) Grundgesamtheit, ein wissenschaftliches *Ergebnis* zu produzieren und dieses mit dem *Ergebnis* der Primärforscher zu vergleichen.
C) *Vollständige*: Die *Syntax* ermöglicht es Sekundärnutzern, unter Verwendung derselben Daten die *Ergebnisse* der Primärforscher exakt zu replizieren. Der Sekundärnutzer wird darüber hinaus in die Lage versetzt, die *Syntax* ggf. für eigene Bedarfe abzuändern und zu erweitern.

Für die Einreichung in datorium im Rahmen des Replikationsservers sollte die *vollständige Reproduzierbarkeit* angestrebt werden. Wir verwenden darüber hinaus die Begriffe Reproduzierbarkeit und Replizierbarkeit als Synonyme.

Syntax: Sequenzen von Befehlen in einer Programmiersprache/Statistikumgebung, die zur Erzeugung von wissenschaftlichen *Ergebnissen* führen. Die Bezeichnungen *Syntax* und Skript werden im Folgenden synonym gebraucht.

Wir unterscheiden drei Arten von *Syntaxen*.

A) *Aufbereitungssyntax*: *Syntax*, die der Nutzbarmachung der *Originaldaten* für wissenschaftliche Zwecke dienlich ist, indem Daten verändert, hinzugespielt oder generiert werden. Transformiert die *Rohdaten* in einen *Analysedatensatz*.
B) *Analysesyntax*: *Syntax*, die die wissenschaftlichen *Ergebnisse* produziert, bspw. das *Ergebnis* einer logistischen Regression, Grafiken und Tabellen.
C) *Mastersyntax*: *Syntax*, die alle anderen *Syntaxen* in der vorgesehenen Reihenfolge aufruft.

Urheber[2]: Person, die die *Originaldaten* erhoben hat. Üblicherweise verfügt nur der *Urheber* über das Recht, die Daten in datorium einzustellen. Sind *Einreicher* und *Urheber* nicht dieselbe Person, benötigt der *Einreicher* die Genehmigung des *Urhebers,* um die Daten in datorium zu veröffentlichen. Andernfalls können zumindest die *Syntaxen* und selbst erstellte Dokumentation eingereicht werden.

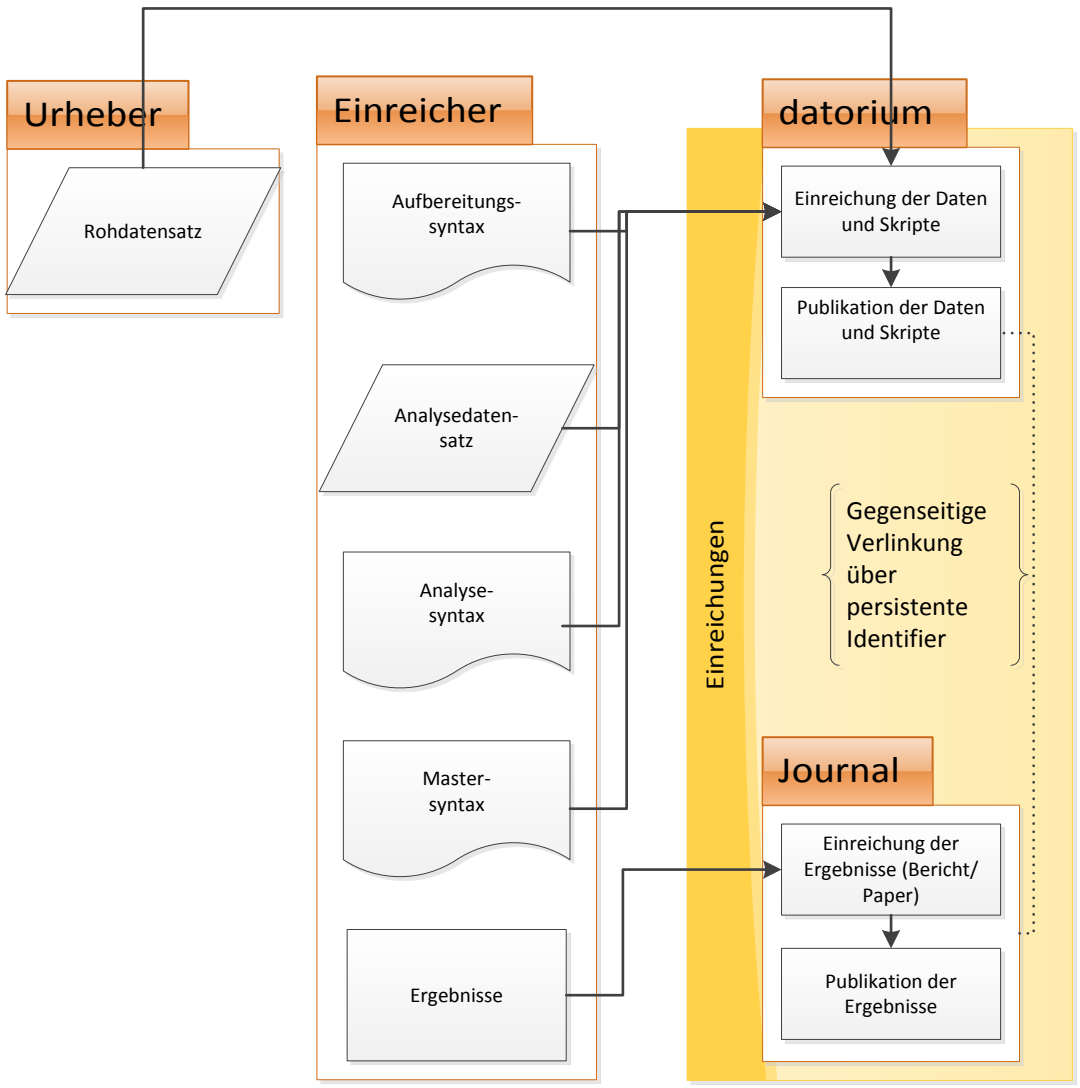


Abbildung 1: Gesamtübersicht über die Organisation

[2] Der Begriff Urheber wird verwendet, obwohl Forschungsdaten in Deutschland nicht generell dem Urheberrecht unterliegen (siehe http://www.forschungsdaten.org/index.php/Urheberrecht, aufgerufen 2016-02-26). Dies ist ein weiterer wichtiger Grund, für Forschungsdaten, die nachnutzbar gemacht werden sollen, Lizenzen zu verwenden.

2. Checkliste

Im Folgenden finden Sie eine Übersicht über Empfehlungen bzgl. der Einreichung von Syntaxen in datorium. Es handelt sich hierbei um Best-Practice-Empfehlungen, von denen abgewichen werden kann. Detaillierte Hinweise zu jedem Punkt erfolgen im anschließenden Abschnitt.

Dimension	Aspekte
Organisation der Syntax(en)	○ Die Originaldaten selbst sind nicht verändert worden. ○ Alle Änderungen der Daten sind in einem neuen Datensatz (Analysedatensatz) gespeichert worden. ○ Alle Aufbereitungsschritte und Auswertungen sind (ausschließlich) via Syntax durchgeführt worden. ○ (ggf.) Aufbereitungsschritte und Analysen sind getrennt in 2 (oder mehr) Syntaxen organisiert. ○ (ggf.) Sehr lange und/oder komplexe Syntaxen sind in kleinere - modular aufgebaute - Einheiten unterteilt. ○ (ggf.) Eine Mastersyntax ruft alle anderen Syntaxen in der vorgesehenen Reihenfolge auf (falls mehr als 2 Syntaxen existieren). ○ Jeder für die Reproduktion der Ergebnisse der Publikation notwendige Output wird via Syntax erzeugt.
Kopfangaben/Header	○ Basale Metadaten stehen am Anfang jeder Syntax (alternativ: im Kopf der Mastersyntax): Name der Autorin bzw. des Autors, Kurzbeschreibung, Version und –Versionsdatum, Zugriff, Dateninput, Benutzereingaben. ○ Idealerweise stehen zusätzliche Metadaten in der Mastersyntax oder einer beiliegenden ReadMe (siehe Abschnitt 2.2).
Dateninput	○ Alle Originaldaten sind benannt. Dazu zählen auch alle erforderlichen Erweiterungen der Statistikumgebung bzw. der Programmiersprache, Pakete/Bibliotheken, Skripte Dritter, etc. ○ Alle Originaldaten werden, sofern möglich, eingereicht. Alternativ ist der Zugriff auf die Originaldaten erläutert, ggf. inkl. Quelle, Ansprechpartner, persistentem Identifier und Zugriffsbeschränkungen. ○ (ggf.) Code von Dritten (Skripte, Pakete, etc.) ist – nach Möglichkeit - in den eigenen Code übernommen worden.
Benutzereingaben	○ Alle Benutzereingaben, die für die Lauffähigkeit der Syntax(en) notwendig sind, sind gekennzeichnet. ○ Es gibt einen zentralen Ort, an dem alle Benutzereingaben getätigt werden.
Inhalt/Code	○ Der Code ist dynamisch geschrieben. Folgecode arbeitet mit Variablen, nicht mit konkreten Werten. ○ (ggf.) Es wird ein Seed gesetzt (falls mit (Pseudo-)Zufallszahlen gearbeitet wird).
Dokumentation des Codes	○ Der Code ist dokumentiert. ○ Es ist dokumentiert, was der Code macht und warum, nicht wie. ○ Leitspruch zur Dokumentation: Soviel wie nötig, so wenig wie möglich.
Einreichung	○ Es werden alle Originaldaten, Rohdaten sowie Analysedatensätze, soweit möglich, eingereicht. ○ Alternativ wird der Zugriff auf die Originaldaten erläutert (insbesondere falls der Einreicher nicht Urheber ist). ○ Es werden alle Syntaxen eingereicht. ○ (ggf.) Es wird eine Mastersyntax eingereicht (falls mehr als 2 Syntaxen existieren). ○ Es werden keine Ergebnisse (statistischen Resultate, Grafiken, Tabellen, etc.) eingereicht, sondern der Code, der sie generiert. ○ Die Dateien liegen in einem möglichst einfachen, offenen und weitverbreiteten Format vor.

Abbildung 2: Checkliste

2.1 Organisation der Syntax(en)

Nach dem Abschluss der Datenerhebung (Primärdaten) bzw. nach Erhalt der Daten durch die Urheber/Archive (Sekundärdaten), werden die Originaldaten (Rohdaten) schreibgeschützt abgelegt. Die Rohdaten werden ab diesem Zeitpunkt unter keinen Umständen mehr überschrieben.

Sind keine Aufbereitungsschritte durchzuführen, erfolgt keine Unterscheidung zwischen Rohdaten- und Analysedatensatz bzw. Aufbereitungs- und Analysesyntax. Ansonsten gilt: Alle Aufbereitungsschritte werden via Syntax produziert. Die geänderten Datenbestände werden in einer neuen Datei, dem Analysedatensatz, abgespeichert (bzw. in mehreren Analysedatensätzen). Als Wissenschaftlerin bzw. Wissenschaftler bürgen Sie dafür, dass die Ergebnisse Ihrer Auswertungen sich bis zu den Originaldaten zurückverfolgen lassen. Jede Änderung der Originaldateien muss dokumentiert sein und sich replizieren lassen.[3]

Da nicht-dokumentierte Aufbereitungsschritte für Nachnutzerinnen und Nachnutzer, und nach einiger Zeit auch für Sie selbst – nicht nachvollziehbar sind, sollten Datenänderungen sowie Auswertungen ausschließlich via Syntax vorgenommen werden. Auf die Nutzung graphischer Menüs ist zu verzichten. Excel - auch wenn es weitverbreitet ist – ist kein geeignetes Programm für die wissenschaftliche Nutzung von Daten.[4]

> ***Beispiel**: Sie haben Kennziffern zu einigen europäischen Ländern aus öffentlichen Quellen in einer Excel-Tabelle zusammengetragen. Anschließend nehmen Sie Aufbereitungsschritte vor: Sie löschen einige nicht plausibel erscheinende Zellenwerte, generieren eine neue Variable/Spalte aus vorhandenen Informationen, errechnen eine lineare Regression, deren Ergebnisse Sie in der Tabelle abspeichern, usw.*
>
> *Wie könnte eine Nachnutzerin bzw. ein Nachnutzer oder Sie selbst - Wochen, Monate oder Jahre nach der letzten Beschäftigung mit den Daten - diese Aufbereitungsschritte nachvollziehen? Möglich wäre dies nur durch das Aufzeichnen jeder Änderung in einer Historie. Dann stünden Sie aber vor zwei gravierenden Problemen: 1) Es ist sehr aufwendig und fehleranfällig, alle Änderungen konsistent und nachvollziehbar gesondert händisch zu dokumentieren und 2) ändern sich nachträglich einzelne Aufbereitungsschritte, müssten Sie wieder neu bei den Rohdaten anfangen, die Ihnen aber nicht mehr vorliegen.*
>
> *Eine Syntax ermöglicht es, alle Aufbereitungsschritte zu dokumentieren, die Daten konsistent zu halten und nachträglich Änderungen vorzunehmen. Die Rohdaten gehen dabei nicht verloren.*

[3] Das Minimum ist, dass eine Historie/Log-Funktion eingeschaltet wird, die alle eingegebenen Befehle aufzeichnet. Wir empfehlen allerdings stattdessen direkt die Syntax(en) einzureichen.

[4] Ein bezeichnender und sehr bekannter Fall von nicht gewissenhafter wissenschaftlicher Arbeit und erschwerter Replikation von Ergebnissen durch Dritte, u.a. aufgrund der Benutzung von Excel, betrifft das Paper „Growth in a Time of Debt" von Carmen Reinhart und Kenneth Rogoff, siehe https://en.wikipedia.org/wiki/Growth_in_a_Time_of_Debt (aufgerufen 2015-12-14).

Abbildung 3: Beispiel: Vorteile Syntax gegenüber Excel/graphischen Menüs

Idealerweise trennen Sie Ihre Syntax in eine Analyse- und eine Auswertungssyntax auf. Dies erleichtert die Nachvollziehbarkeit. Sehr lange oder komplexe Syntaxen können in kleinere Einheiten aufgeteilt werden. Achten Sie dabei auf einen modularen Aufbau: Jede Syntax sollte eine klar umgrenzte Funktionalität aufweisen und sich – soweit möglich – ohne zusätzliche Benutzerinteraktion/-eingaben ausführen lassen. Bei zwei oder mehr Syntaxen empfehlen wir das Erstellen einer Mastersyntax. Dies ist ein Skript, das alle anderen Syntaxen in der vorgesehenen Reihenfolge aufruft und eine zentrale Stelle für Metadaten und Benutzereingaben bietet.

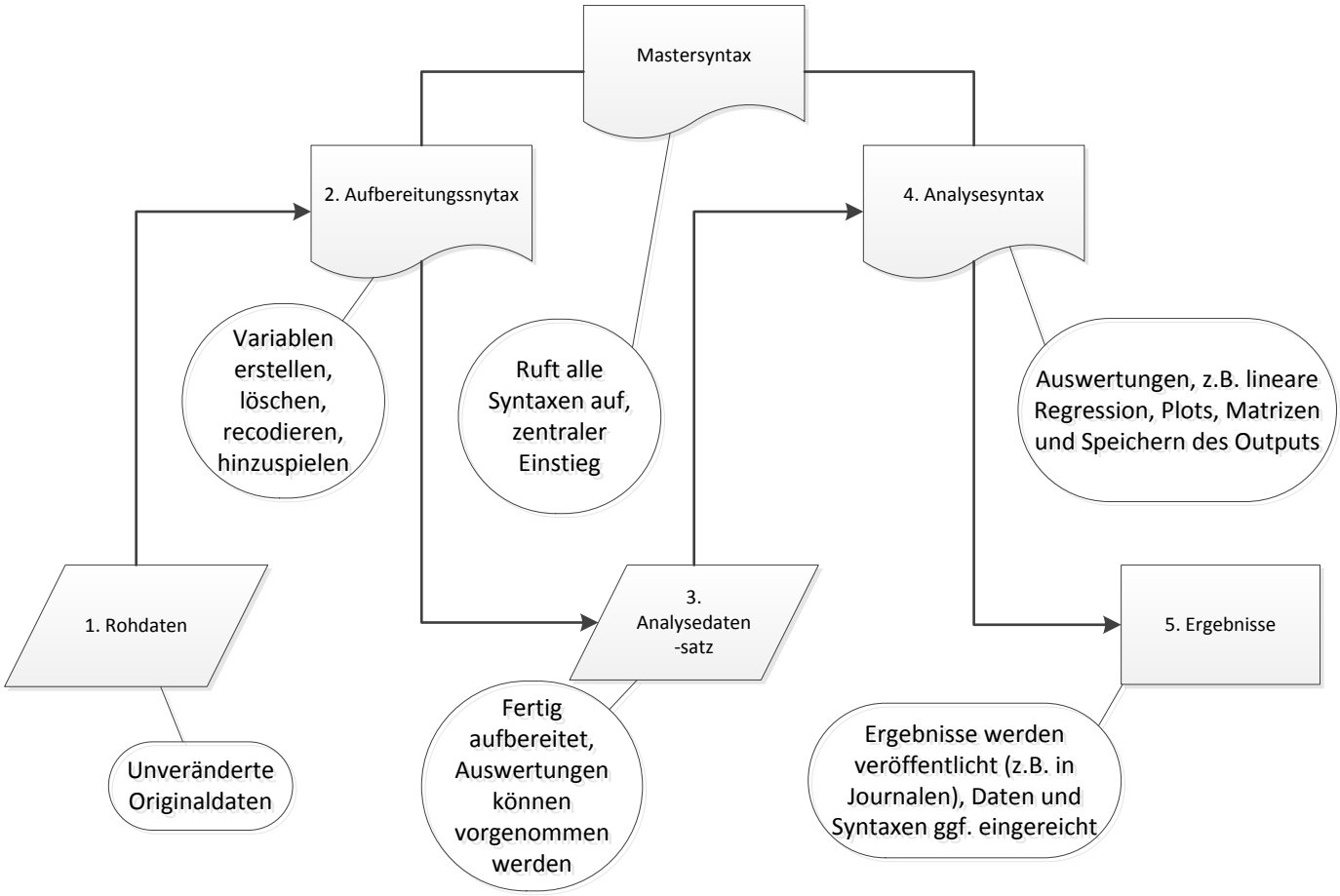


Abbildung 4: Organisation der Syntaxen

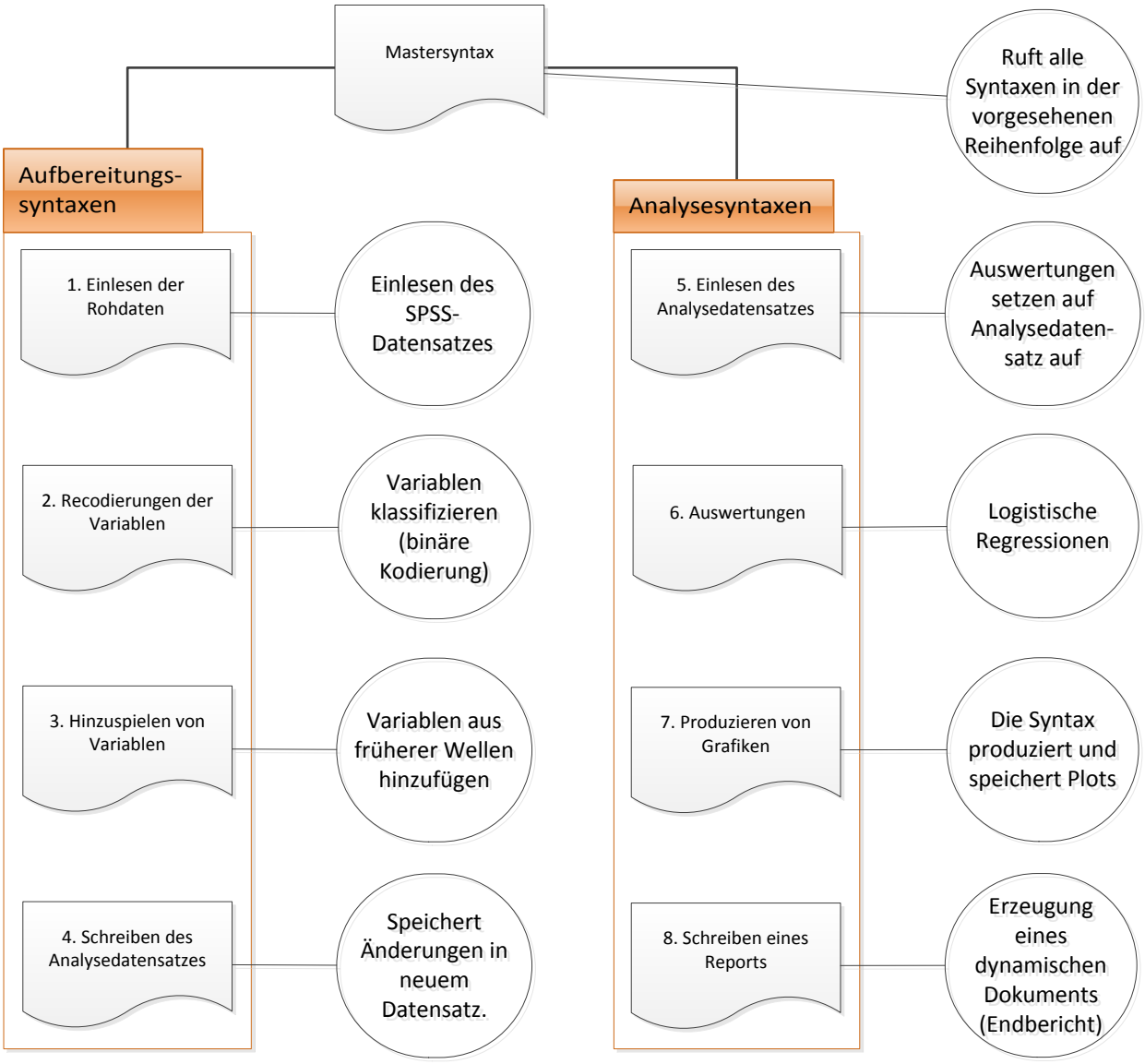


Abbildung 5: Unterteilung in kleine (modulare) Syntaxen

Jeder für die Reproduktion der Ergebnisse der Publikation notwendige Output wie bspw. Grafiken wird ebenfalls via Syntax erzeugt. Ergebnisse werden nicht gesondert in datorium eingestellt. Auf diese Weise ist sichergestellt, dass der Output - wie veröffentlicht - von der Syntax erzeugt werden kann und beide intern konsistent sind.

2.2 Kopfangaben/Header

Schreiben Sie die wichtigsten Metadaten bzgl. Ihrer Syntaxen in den Anfang der Dateien (als Kommentar). Dadurch erleichtern Sie Nachnutzerinnen und Nachnutzern das Verständnis. Alternativ, insbesondere falls viele Syntaxen vorliegen, schreiben Sie die Kopfangaben nur in die Mastersyntax.

Im Folgenden machen wir Ihnen einige Vorschläge zu Metadaten. Bedenken Sie, dass die Syntax einem Nachnutzer bzw. einer Nachnutzerin auch ausgedruckt bzw. außerhalb von datorium vorliegen

kann. Daher sollten wenigstens basale Metadaten wie Autorin bzw. Autor und Zugriffsmöglichkeit in jeder Syntax selbst stehen (und nicht nur in den Metadaten von datorium).

Pflichtfeld	Metadatum	Beschreibung	Beispiel
X	Autorin bzw. Autor	Autorin bzw. Autor der Syntax, ggf. mit Kontaktdaten.	Max Mustermann (max_Mustermann(at)uni-bielefeld(dot)de)
X	Titel	Titel der Syntax.	Aufbereitung.R
X	Kurzbeschreibung	Kurze Beschreibung der wichtigsten Aufgaben der Syntax.	Diese Syntax lädt den Datensatz Eurobarometer 73.1 (ZA5000) und generiert Variablen für Auswertungen (siehe auch Analyse.R).
X	Versionsbeschreibung	Versionsangaben und -beschreibung.	v1-0-0. Finale Version der Syntax, mit der die Ergebnisse aus Mustermann, Max 2015: „Replication attempt of Mustermann 2014 results fails“ erzeugt wurden.
X	Versionsdatum	Datum. Idealerweise im Format JJJJ-MM-TT.	2015-11-27
X	Zugriff	Ablageort der Syntax. Wenn möglich mit persistentem Identifier.	http://dx.doi.org/10.7802/999
	Statistiksoftware	Die Statistiksoftware bzw. Programmiersprache.	R
	Version der Statistiksoftware	Versionsangaben.	R version 3.2.0 (2015-04-16)[5]
	Abhängigkeiten	Erweiterungen (z.B. Pakete, ado-Files), Programme, Syntaxen etc., die verfügbar sein müssen, um die Syntax ausführen zu können.	R-Pakete: ggplot2, stringr, caret[6]
	Version der Abhängigkeiten	Versionsangaben zu den Abhängigkeiten.	ggplot2: 1.0.1, stringr: 1.0.0, caret: 6.0.62[7]
	Betriebssystem und Hardware	Falls davon auszugehen ist, dass Betriebssystem oder Hardware einen Unterschied für die Replikation bzw. Lauffähigkeit der Syntax machen, benennen Sie die Systeme, mit denen Sie die Syntax erfolgreich getestet haben.	Platform: x86_64-w64-mingw32/x64 (64-bit) Running under: Windows 7 x64 (build 7601) Service Pack 1[4]
	Dateninput	Beschreibung der Datengrundlage. Nach Möglichkeit inkl. Zitation, persistentem Identifikator und Zugriffsmöglichkeit, insbesondere für Sekundärdaten.	Europäische Kommission (2012): Eurobarometer 73.1 (Jan-Feb 2010). TNS OPINION & SOCIAL, Brussels [Producer]. GESIS Datenarchiv, Köln. ZA5000 Datenfile Version 4.0.0, doi:10.4232/1.11428

[5] In R aufrufbar über sessionInfo(). Alte Versionen von R sind verfügbar unter http://cran.r-project.org/src/base (aufgerufen 2016-01-07).
[6] In R ist es möglich, Funktionsaufrufe so zu schreiben, dass eindeutig angegeben ist, zu welchem Paket die jeweilige Funktion gehört: paket::funktion, z.B. ggplot2::qplot().
[7] In R für Pakete aufrufbar über packageVersion(). Alte Versionen von R-Paketen sind verfügbar unter http://cran.r-project.org/src/contrib/Archive (aufgerufen 2016-01-07).

			Zugriff: http://dx.doi.org/10.4232/1.11428 Daten und Begleitmaterial sind für jedermann freigegeben.
	Benutzereingaben	Beschreibung aller Code-Stellen, die durch den Benutzer angepasst werden müssen, damit die Syntax auf dem Gerät der Nachnutzerin bzw. des Nachnutzers ausführbar ist.	Um die Syntax bei Ihnen zum Laufen zu bringen, ändern Sie die Pfadangabe in Zeile 3 zu dem Verzeichnis, in dem der Datensatz ZA5000 liegt.
	Lizenz	Falls Sie Angaben dazu machen möchten, wie und unter welchen Bedingungen die Syntax nachgenutzt werden darf. datorium empfiehlt Ihnen die Verwendung einer freien Lizenz, z.B. Creative Commons BY oder GNU General Public Licence.[8] Muss der in datorium gewählten Lizenz entsprechen.	CC BY 4.0 (http://creativecommons.org/licenses/by/4.0/)
	Publikation	Falls die Daten zu einer Publikation gehören: Zitieren Sie die Publikation, idealerweise inkl. persistentem Identifier und Zugriffsmöglichkeit.	Mustermann, Max 2015: „Replication attempt of Mustermann 2014 results fails“, in: Zeitschrift für die Förderung von Replikationen in den Sozialwissenschaften, Heft 4: 63-109. DOI: http://dx.doi.org/10.7802/109 Zugriff: Die Publikation ist frei zugänglich für jedermann.
	Wichtige Hinweise	Wichtige Hinweise, die die Ausführung bzw. Ausführbarkeit der Syntax betreffen. Dieses Feld sollte nicht zur allgemeinen Dokumentation des Codes missbraucht werden.	Die leave-one-out cross-validation in Zeile 100 berechnet über 53000 Modelle. Dies dauert auf einem handelsüblichen PC (Stand Nov. 2015) mehrere Stunden. Alternativ Zeile 99 umschreiben zu „k <- 5“ (ohne Anführungsstriche) (für 5-fold cross-validation). Dies repliziert die Ergebnisse in Mustermann, Max 2015: 13 allerdings nur näherungsweise.

Abbildung 6: Vorschläge für Metadaten zu Syntaxen

2.3 Dateninput

Der Zugriff auf die Originaldaten (Rohdaten und Analysedaten) ist, neben der Syntax, die wichtigste Grundvoraussetzung für die Reproduzierbarkeit Ihrer Ergebnisse.

Stellen Sie Daten, die Sie selbst erhoben haben (Primärdaten), in datorium ein, sofern keine wichtigen Gründe, bspw. datenschutzrechtliche Bedenken, dagegensprechen. Das datorium Team kann Sie zu datenschutzrechtlichen und anderen Fragen beraten. Falls Sie mit Sekundärdaten gearbeitet haben und das Recht, die Originaldaten einzustellen, bei einer anderen Person bzw. Stelle

[8] Freie Lizenz definieren wir mit Bezug auf https://de.wikipedia.org/wiki/Freie_Lizenz (aufgerufen 2016-01-15) als Lizenz, „die die Nutzung, Weiterverbreitung und Änderung urheberrechtlich geschützter Werke erlaubt“.

liegt (Urheber der Daten, Arbeitgeber, etc.), verweisen Sie auf die Datenquelle. Führen Sie aus, wie Nachnutzerinnen und Nachnutzer Zugriff auf die Daten erhalten könnten. Idealerweise verwenden Sie hierzu einen persistenten Identifikator und benennen Nachnutzungsbedingungen und Lizenzen, soweit vorhanden und bekannt.

Beachten Sie alle Daten und Skripte, auf denen die Lauffähigkeit Ihrer Syntax(en) und die Reproduzierbarkeit Ihrer Ergebnisse beruhen. Gerne übersehen werden im Code eingebettete Aufrufe von weiteren (selbstverfassten) Skripten.[9] Bei von Dritten verfassten Paketen und Erweiterungen[10] ist zu bedenken, dass deren Weiterentwicklung nicht der eigenen Kontrolle unterliegt. Sollten sich die genutzten Erweiterungen verändern, könnte dies dazu führen, dass die Replikation Ihrer Daten fehlschlägt. Es sind auch nicht immer frühere Versionen solcher Pakete auffindbar. Hier ist zu überlegen, ob es rechtlich möglich und organisatorisch sinnvoll ist, den Code dieser Erweiterungen in die eigenen Syntaxen zu integrieren (Christensen/Soderberg 2015: 48).[11]

2.4 Benutzereingaben

Fassen Sie am Anfang Ihres Skriptes bzw. bei mehreren Skripten an einer für Nachnutzerinnen und Nachnutzer ersichtlichen und nachvollziehbaren Stelle – idealerweise der Mastersyntax - jeglichen Code zusammen, der vom Benutzer eingegeben oder angepasst werden muss, damit die Syntax lauffähig wird. Bspw. sollten Sie hier alle Pfadangaben sammeln. Selbstverständlich sollte Ihr Code so geschrieben sein, dass so wenige Angaben wie möglich durch die Nachnutzerin bzw. den Nachnutzer zu machen sind.

2.5 Inhalt/Code

Schreiben Sie nach Möglichkeit dynamischen Code. Damit ist gemeint, dass Sie statt konkrete Werten in der Syntax zu verwenden, diese in Variablen speichern und mit letzteren weiterarbeiten. Dies betrifft insbesondere auch das Ergebnis von Auswertungen.[12] Durch diese Maßnahme tragen Sie zur internen Konsistenz Ihrer Syntaxen sowie zur leichteren Überarbeitung und Anpassung bei.

Haben Sie einen random numbers generator genutzt, d.h. pseudorandomisierte Zufallszahlen erzeugt bzw. mit Funktionen gearbeitet, die auf solchen basieren, bspw. bei k-means clustering, setzen Sie einen seed, ansonsten sind Ihre Ergebnisse nicht *vollständig replizierbar*.[13]

2.6 Dokumentation des Codes

Dokumentieren Sie Ihren Code über das Setzen von Kommentaren. Beschreiben Sie dabei, was der Code tut und wieso, nicht wie. Dokumentieren Sie so ausführlich wie nötig und gleichzeitig so

[9] In R bspw. via source().
[10] In R bspw. Pakete, die nicht zum Standardumfang von R gehören und über den Aufruf von install.packages() (o.ä.) installiert werden müssen.
[11] Auf CRAN (https://cran.r-project.org) gelistete Pakete sind unter einer „freien“ Lizenz veröffentlicht, siehe https://cran.r-project.org/web/packages/policies.html und https://svn.r-project.org/R/trunk/share/licenses/license.db (aufgerufen 2015-12-14)
[12] Beispiel in R:

```
model1 <- lm(mpg ~ wt, data = mtcars)
# Schlechte Lösung
print("ß0 beträgt 37.285")
# Besser
print(paste0("ß0 beträgt ", round(coef(model1)[[1]], digits = 3)))
```

[13] Ein Seed ist ein Startwert für die Generierung von Pseudozufallszahlen. Durch das Setzen eines seeds vor Aufruf eines Zufallszahlengenerators werden dessen Ergebnisse replizierbar. Siehe https://de.wikipedia.org/wiki/Seed_key (aufgerufen 2015-12-14). In R über den Aufruf von set.seed().

sparsam wie möglich. Das Setzen von sehr vielen und/oder sehr langen Kommentaren kann sich kontraproduktiv auf das schnelle Verständnis der Syntax auswirken. Verzichten Sie insbesondere auf das Dokumentieren von selbstverständlichem bzw. augenblicklich erschließbarem Code.[14]

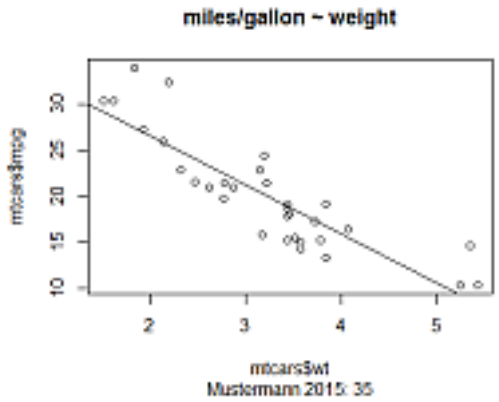
2.7 Einreichung

Reichen Sie nach Möglichkeit alle zur Replikation der Ergebnisse erforderlichen Originaldaten und Skripte ein. Verfügen Sie nicht über die entsprechenden Rechte oder verhindern andere Gründe wie bspw. datenschutzrechtliche Bedenken die Einreichung von Daten, verweisen Sie auf die Originaldaten. Die Syntaxen sollten immer vollständig eingereicht werden, sofern die Lizenzbestimmungen dies zulassen.

Es werden keine Outputs an sich, bspw. Grafiken und Tabellen der Ergebnisse statistischer Analysen, eingereicht, sondern stattdessen der Code, der diese erzeugt.

Beispiel: Stellen Sie die Visualisierung Ihrer linearen Regression nicht als Grafik in datorium ein, siehe a), sondern den generierenden Code, siehe b).

a)



b)

```
plot(mtcars$wt,
mtcars$mpg, main =
"miles/gallon ~ weight",
sub = "Mustermann 2015:
35")

abline(lm(mtcars$mpg ~
mtcars$wt))
```

Abbildung 7: Beispiel: Keinen Output einreichen

Wählen Sie ein möglichst freies[15], einfaches und weitverbreitetes Format für alle Ihre einzureichenden Dateien. Idealerweise werden Dateien als reine Textdateien eingereicht.[16] Bedenken Sie, dass die Kompatibilität, Lesbarkeit und damit die langfristige Nutzbarkeit von proprietären Formaten v.a. von der Herstellerfirma der dazugehörigen Software abhängt. Es ist nicht sichergestellt, dass ein proprietäres Format in bspw. 20 Jahren noch geöffnet/eingelesen werden kann oder der Inhalt identisch zum heutigen Stand angezeigt wird. Daher sollte überlegt werden, ob

[14] Beispiel in R:
```
# Verzichten Sie auf diesen Kommentar: Lädt das Paket foreign
library(foreign)
# Auch dieser Kommentar ist überflüssig: Weist der Variable x den Wert 12 zu.
x <- 12
```
[15] „Frei“ im Sinne von freie Lizenz (siehe oben).
[16] z. B. Daten im Comma-separated-values (csv) Format statt als Excel-Datei.

die Nutzung eines freien Formats bzw. eine Konvertierung in ein solches nicht die bessere Lösung darstellt.

Möchte man den Komfort von proprietären Softwarelösungen, bspw. Variablen- und Wertelabels in SPSS, nicht missen, kann man Daten in den meisten Fällen sowohl als Textdatei als auch im proprietären Format in datorium einstellen.

Literaturverzeichnis

Christensen, Garret and Courtney Soderberg, 2015: Manual of Best Practices in Transparent Social Science Research. August 11, 2015. Verfügbar unter: https://github.com/garretchristensen/BestPracticesManual/blob/master/Manual.pdf.

Gandrud, Christopher, 2015: Reproducible Research with R and RStudio. Second Edition. Chapman & Hall/CRC (The R Series).

Gentzkow, Matthew and Jesse M. Shapiro, 2014: Code and Data for the Social Sciences: A Practitioner's Guide. March 11, 2014. Verfügbar unter: http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf.

Jensen, Uwe, 2012: Leitlinien zum Management von Forschungsdaten. GESIS Technical Reports 2012 | 07. Verfügbar unter: http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/gesis_methodenberichte/2012/TechnicalReport_2012-07.pdf.

Jensen, Uwe, Schweers, Stefan und Zeljko Carevic, 2014: Die Metadateneditoren der VFU soeb 3. GESIS Technical Reports, 2014 | 14. Verfügbar unter: http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/gesis_methodenberichte/2014/TechnicalReport_2014-14.pdf.

Xie, Yihui, 2015: Dynamic Documents with R and knitr. Second Edition. The R Series. Chapman & Hall/CRC (The R Series).

Anhang A. Die Arbeit mit Syntaxen

In diesem Anhang werden kurze Hinweise gegeben, wie Sie die Arbeit mit Syntaxen so gestalten können, dass die Reproduktion Ihrer Ergebnisse erleichtert wird. Keiner der folgenden Abschnitte umfasst dabei alle wichtigen Gesichtspunkte des jeweiligen Themas. Stattdessen verweisen wir auf weiterführende Literatur.

Gandrud (2015) hat eine exzellente und umfangreiche Einführung in das Sicherstellen von Replizierbarkeit in der Arbeit mit R vorgelegt. Gentzkow/Shapiro (2014) bieten ausführliche Erläuterungen zu den in diesem Anhang nur angerissenen Punkten. Christensen/Soderberg (2015) befassen sich auch mit theoretischen Aspekten der Replizierbarkeit wissenschaftlicher Ergebnisse, der Entstehung von Bias (Publication Bias, Selective Reporting, Betrug in der Wissenschaft, etc.) und entsprechenden Kontrollmechanismen (bspw. Trial Registration).[17]

A.1 Code leserlich gestalten

Teilen Sie Ihren Code in kleine Einheiten ein, die jeweils ein konkretes, eng umfasstes Problem behandeln. Das erleichtert die Nachvollziehbarkeit.

Jeder Code, der in derselben oder ähnlicher Form mehrfach ausgeführt wird, sollte der Übersichtlichkeit halber und um die Fehlerquellen gering zu halten, als allgemeine Funktion geschrieben werden. Eine einzige Funktion zu warten, ist einfacher und weniger fehleranfällig, als redundanter Code.

Beispiel: Es sollen für mehrere Variablen des Datensatzes mtcars einfache lineare Modelle mit der abhängigen Variable mpg (miles per gallon) gerechnet und als Grafik visualisiert werden. Dies resultiert in der mehrfachen Verwendung derselben 3 Zeilen Code, die für jede unabhängige Variable stets (allerdings nur geringfügig) angepasst werden müssen.

```
fit <- lm(mtcars$mpg~mtcars$wt)

plot(mtcars$wt, mtcars$mpg)

abline(fit,col="red")
```

Alternativ kann eine allgemeine Funktion geschrieben werden:

```
regplot <- function(x,y) {

 fit <- lm(y~x)

 plot(x,y)
```

[17] Trial registration, das Registrieren möglichst aller Studien eines Fachgebiets zur Vermeidung von selektivem Publizieren „erwünschter“ Studien (vgl. Christensen/Soderberg 2015: 18ff), ist besonders in der Medizin weitverbreitet. Exemplarische Registrierungsagenturen finden Sie unter http://www.who.int/ictrp/trial_reg/en/ (Medizin) oder – für die Politikwissenschaften – unter http://egap.org/design-registrations (aufgerufen 2015-12-18).

abline(fit,col="red") }[18]

Mithilfe dieser Funktion können nun beliebige einfache lineare Modelle aufgerufen werden (z.B. via regplot(mtcars$mpg, mtcars$carb)), gewartet werden müssen aber nur 4 Zeilen Code.

Abbildung 8: Beispiel: Funktionen schreiben

Wenn der Benutzer die Möglichkeit hat, Parameter einzustellen, oder dies sogar tun muss, um die Syntax korrekt auszuführen, beschreiben Sie eindeutig, wo dies möglich bzw. nötig ist und welche Parameter für die Publikation gewählt wurden.

Schreiben Sie eine Mastersyntax oder ein Skript (z.B. Makefile, Shell-Skripte, Pythonskript), das alle für die Replikation der Ergebnisse notwendigen Dateien in der korrekten Reihenfolge aufruft.

A.2 Variablen- und Funktionsnamen

Wählen Sie einfache, konsistente, leicht zu merkende Benennungen für Variablen und Funktionen. Der Name verweist auf die Aufgabe der Funktion bzw. die Art oder den Wert der Variable.

Empfehlungen für die Benennung von Variablen finden Sie in Jensen 2012: 27ff.

A.3 Dokumentation

Dokumentieren Sie Ihre Syntax. Dokumentieren Sie so ausführlich wie nötig und so sparsam wie möglich. Verzichten Sie darauf, Selbstverständliches zu dokumentieren.

Lassen Sie Ihre Arbeit von der Statistikumgebung aufzeichnen. R erstellt automatisch eine History mit den zuletzt benutzen Befehlen. Sie können sich diese durch die Pfeil-Tasten (aufwärts und abwärts) anschauen und ggf. erneut ausführen.[19]

A.4 Testen

Testen Sie Ihre Syntax ausgiebig, bevor Sie diese einreichen. Die Tests sollten in einer leeren R Umgebung ausgeführt werden, damit keine existierenden Objekte die Ausführung beeinflussen.[20] Es wird empfohlen, die Syntax ebenfalls in einem anderen Verzeichnis zu testen. Idealerweise bitten Sie außerdem eine nicht mit Ihrem Projekt vertraute Person, die Syntax(en) auf das Ziel der vollständigen Replikation der Ergebnisse hin auszuführen.

A.5 Arbeit mit Verzeichnissen

Arbeiten Sie mit relativen statt mit absoluten Pfaden, um Nachnutzerinnen und Nachnutzern die Mühe zu ersparen, diese anzupassen, und um eine Fehlerquelle auszuschließen.[21]

---

[18] Das Beispiel ist entlehnt aus James, G. et. al, 2014: An Introduction To Statistical Learning with Applications in R. Springer Science+Business Media. New York 2013 (Corrected at 6th printing 2015) (verfügbar unter http://www-bcf.usc.edu/~gareth/ISL, aufgerufen 2015-12-17).

[19] In R können sie die History durch den Aufruf von savehistory() speichern und durch loadhistory() laden. In Stata und SPSS können Sie eine log-Datei erstellen lassen, die die letzten Befehle und Outputs enthält (weitere Informationen in Stata über den Befehl help log, in SPSS in der Hilfe unter dem Schlagwort OMS).

[20] In R löscht man alle geladenen Objekt über den Aufruf von rm(list = ls()).

[21] Beispiel in R:

```
# Schlecht!
load(file = "C:\\my_project\\original\\raw_data.RData")
```

Wenn Ihr Projekt viele Dateien enthält, macht es Sinn, eine Ordnerstruktur zu erstellen, die diese nach Funktion ordnet.

Beispiel[22]:

Unterordner	*Funktion*
/input	*Originaldaten (schreibgeschützt, zum Einlesen)*
/code	*Syntaxen/Skripte*
/output	*Tabellen, Grafiken, sonstiger Output (Dateien werden via Syntax erstellt und sind in diesem Sinne selbst nicht archivierungs-würdig)*
/temp	*Temporäre Dateien (z.B. Output sehr aufwendiger Berechnungen, der zwischengespeichert wird)*

Abbildung 9: Beispiel: Verzeichnisstruktur nach Funktion

Wenn Sie mit vielen Dateien und/oder Ordnern arbeiten, legen Sie in das höchste Verzeichnis eine ReadMe-Datei (Textdatei), die Nachnutzerinnen und Nachnutzer über die Verzeichnisstruktur bzw. die Funktion der einzelnen Dateien und die korrekte Reihenfolge, in der diese aufgerufen werden müssen, aufklärt.

In RStudio gibt es die Möglichkeit, mit Projekten zu arbeiten. Ein Projekt in RStudio hat mehrere Vorteile gegenüber der „normalen" Arbeit mit Skripten: u.a. a) einfacher Zugriff auf das Projekt, b) das Arbeitsverzeichnis wird automatisch bei jedem Start korrekt gesetzt, c) zuletzt bearbeitete Syntaxen sowie der Workspace und die History werden geladen und d) die Versionierung mit git wird vereinfacht (Gandrud 2015: 70).

A.6 Verringerung von Benutzerinput

Die Syntaxen sind so zu gestalten, dass die Nachnutzerin bzw. der Nachnutzer möglichst wenige Eingaben und/oder Änderungen vornehmen muss, da Benutzereingaben schwer zu kontrollieren sind. Insbesondere wenn Sie mehr als 2 Syntaxen einstellen, achten Sie darauf, dass keine

# Besser!
load(file = ".\\original\\raw_data.RData")

[22] Beispiel in Anlehnung an Gentzkow/Shapiro 2014: 16.

Benutzereingaben/-änderungen „zwischen“ den einzelnen Aufrufen von Syntaxen notwendig sind. Stattdessen erstellen Sie idealerweise eine Mastersyntax.

Alle notwendigen Benutzereingaben sollten sich, soweit möglich, an einer zentralen Stelle befinden.

A.7 Arbeiten mit einer IDE

Das Arbeiten mit einer IDE (Integrated Development Environment, bspw. RStudio) erleichtert das Schreiben und Organisieren von Syntaxen enorm, indem i.d.R. u.a. a) der Code durch farbliche Hervorhebungen spezifischer Parts übersichtlicher und leichter lesbar dargestellt wird und b) Code auto-vervollständigt werden kann. Außerdem bieten IDEs üblicherweise viele weitere sinnvolle Ergänzungen/Features an.

A.8 Versionierung

Insbesondere für Kollaborationen, aber auch für einzelne Forschende, empfiehlt es sich, Dateien zu versionieren, damit a) ggf. auf alte Versionen zurückgegriffen werden kann (bspw. wenn sich Fehler eingeschlichen haben), b) nachvollziehbar bleibt, welche Version einer Datei die aktuellste ist und c) Datei-Konflikte, die bei gleichzeitiger Bearbeitung in Kollaborationen entstehen, aufgeklärt werden können. Die einfachste Variante, aber zugleich eine wenig effiziente, ist es, ein Änderungsdatum und/oder ggf. eine Versionskennung in den Dateinamen oder –kopf aufzunehmen, bspw. „Analyse_2015-12-14.R“.

Optimal ist die Verwendung professioneller Versionierungssoftware wie bspw. git, die zwar einige Einarbeitungszeit in Anspruch nehmen, aber sehr mächtige Features bieten.[23] Im Gegensatz zu den sehr häufig mittels git verwalteten Software-Syntaxen ist es für die Replikation sozialwissenschaftlichen Forschung nicht notwendig oder auch nur angezeigt, mehr als die letzte Version der Syntax einzureichen.[24]

A.9 Dynamische Dokumente

Schreiben Sie dynamische Dokumente, die sowohl Ihren Code als auch Ihren Report/Endbericht enthalten. Auf diese Weise erleichtern Sie die Einhaltung interner Konsistenz zwischen Syntax(en) und finalem Bericht. Eine kurze Einführung in das Thema dynamische Dokumente finden Sie in Anhang B. Umfassende Informationen erhalten Sie in Xie 2015 und Gandrud 2015.

[23] git ist kostenlos und frei (GNU GPL-lizensiert). Es ist verfügbar unter https://git-scm.com/. Github (https://github.com) ist ein Service, der auf git aufsetzt und die Kollaboration mehrere Forscher miteinander erleichtert, indem die versionsüberwachten Dateien eines Projekts online gehostet werden.

[24] Welche Version ist die „letzte“ Version bzw. welche ist die Version, die in datorium eingestellt werden soll? Gemeint ist die Version der Syntax, die für die Publikation genutzt wurde bzw. die den dort veröffentlichten Output (möglichst) vollständig repliziert. Daran ändert sich im Wesentlichen auch nichts, falls nachträglich Fehler in der Syntax festgestellt werden: Zentral ist die Replikation der publizierten Ergebnisse. Allenfalls ist denkbar, zusätzlich eine fehlerbereinigte Version der Syntax oder eine Errata in datorium anzubieten.

Anhang B. Dynamische Dokumente/Literate Programming

Empirisch arbeitende Forscherinnen und Forscher halten Code und darauf basierende Dokumentation (Forschungsbericht, Präsentation, etc.) oftmals in separaten Dateien vor. Sie schreiben Code, führen ihn aus und kopieren dann einzelne Ergebnisse und Outputs in einen gesonderten Bericht. Dieses Vorgehen bringt eine Reihe von gravierenden Nachteilen mit sich (Xie 2015: xiii):

1. Es ist fehleranfällig, manuell und selektiv Ergebnisse zwischen Dokumenten zu kopieren. Die interne Konsistenz zwischen den Dokumenten einzuhalten, gestaltet sich schwierig.
2. Es ist sehr zeitaufwendig und mühselig.
3. Das Vorgehen ist nicht dokumentiert und damit äußerst schwer zu replizieren, falls sich nachträgliche Änderungen ergeben.
4. Kleinste Änderungen in den Daten können dazu führen, dass erneut langwierig Ergebnisse und Outputs von einem ins andere Dokument kopiert werden müssen.

Alternativ können Code und Dokumentation miteinander in einem Dokument verwoben werden. Alle oben genannten Probleme entfallen dann. Ein solches Dokument wird dynamic document bzw. dynamisches Dokument genannt. Das Konzept dahinter – literate programming – geht auf Donald Knuth zurück (Xie 2015: 1). Ohne auf die konzeptionellen Grundlagen einzugehen, wird im Folgenden gezeigt, wie Sie in RStudio ein dynamisches Dokument - in R-Markdown mit knitr - erzeugen.[25]

Das allgemeine Vorgehen erfolgt in diesen Schritten:

1. Sie erstellen eine R-Markdown-Datei (Dateiendung .Rmd). R-Markdown ist eine markup-Sprache, mit Sie bestimmen können, wie bestimmter Text dargestellt werden soll.[26]
    a. In diese Datei schreiben Sie sowohl den Fließtext für Ihre Dokumentation als auch den Code für die statistischen Auswertungen. Letzteren kennzeichnen Sie auf festgelegte Weise als Code (siehe weiter unten), damit knitr erkennt, dass diese Stellen des Dokuments in R auszuführen sind.
    b. Sie formatieren Ihren Text nach eigenem Wunsch bzw. nach den Möglichkeiten von R-Markdown, bspw. setzen Sie einen Titel, schreiben einigen Textstellen kursiv und wählen das gewünschte Output-Format.
    c. Alle Änderungen an Ihrem Code oder Ihrer Dokumentation nehmen Sie in dieser Datei vor.
2. Sie kompilieren die R-Markdown-Datei (Dateiendung .md).[27] knitr lässt den Code von R ausführen und formatiert den Output in der markup-Sprache.

[25] Es gibt Alternativen zu knitr – z.B. Sweave – aber wir beschränken uns hier auf knitr aufgrund seiner vielfältigen Features. Die Nutzung von RStudio ist keine Voraussetzung, erleichtert aber das Erstellen von dynamischen Dokumenten, indem alle notwendigen Pakete mitinstalliert und nutzerfreundlich in die GUI integriert werden. Eine Anleitung zur Erstellung dynamischer Dokumente mit Stata findet sich unter http://www.haghish.com/statistics/stata-blog/reproducible-research/markdoc.php (aufgerufen 2016-02-22).

[26] Für einen Überblick über den R-Markdown-„Dialekt“ siehe http://rmarkdown.rstudio.com und https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf („Cheat Sheet“, aufgerufen 2016-01-07).

[27] Diese Datei ist aus Sicht des (RStudio-)Nutzers in der Regel nur ein Zwischenschritt und nicht weiter von Interesse.

3. Entsprechend den Benutzereinstellungen erstellt knitr eine HML-, PDF- oder Worddatei oder eine HTML5-Slides-Präsentation (ggf. inkl. interaktiver Shiny-Elemente).[28] Siehe Beispiele in Anhang C. Diese Datei verwenden Sie für die Präsentation Ihrer Ergebnisse.

Die Umsetzung in RStudio geschieht folgendermaßen[29]:

1. Öffnen Sie eine neue R-Markdown-Datei (File -> New File -> R Markdown …)
2. Im aufklappenden Menü wählen Sie Titel, Autor und Standard-Outputformat (lässt sich nachträglich ändern).
3. Das R-Markdown-Dokument öffnet sich bereits mit einem minimalen Beispiel.
    a. Ganz oben im Dokument finden Sie Metadaten des Dokuments (Titel, etc.).
    b. Fließtext steht ohne besondere Kennzeichnung in der Datei. Formatierungen nehmen Sie über R-Markdown Syntax vor.[30]
    c. Code wird entweder in eigene Blöcke (Chunks) aufgenommen, die vom Fließtext folgendermaßen abgegrenzt sind:

```
```{r}
# Alles zwischen ```{r} und ``` ist R Code
```
```

Abbildung 10: Code-Chunk (knitr/R-Markdown)

oder – in einer Zeile zusammen mit Dokumentation („Inline“) – durch „`r `“ (ohne doppelte Anführungszeichen):

```
Zuerst Dokumentation, dann  Code: `r 3+5`, abschließend Dokumentation
```

Abbildung 11: Inline-Code (knitr/R-Markdown)

    d. Abschließend klicken Sie auf den Button „Knit HTML“ (bzw. Knit PDF/Word).

Beispiel:

Option A: Sie haben eine lineare Regression gerechnet und einzelne Ergebnisse händisch in Ihren Endbericht (eine Worddatei) kopiert. Später stellen Sie fest, dass Sie vergessen haben, einige nicht-plausible Angaben in einer der unabhängigen Variablen auszuschließen. Sie ändern Ihren Code und führen ihn aus. Alle oder die meisten Ergebnisse der linearen Regression haben sich verändert. Sie kopieren erneut händisch Ergebnisse in die Worddatei. Nach

[28] Shiny ist ein Paket für R, das die Erstellung interaktiver Webapplikationen ermöglicht. Für mehr Informationen über Shiny siehe https://www.rstudio.com/products/shiny (aufgerufen 2016-01-07).

[29] Die Umsetzung in R ohne RStudio ist ebenfalls sehr einfach. Folgender Code erstellt bspw. aus einer R-Markdown-Datei eine HTML-Datei.

1. install.packages("knitr", dependencies=TRUE)
2. library(knitr)
3. knit2html("pfad/name_Rmarkdown_datei.Rmd")

Hinweis: knit2html nutzt das markdown-Paket. Inzwischen wird empfohlen, das rmarkdown-Paket zu nutzen, da dieses R-Markdown v2 unterstützt (Xie 2015: 168 und https://cran.r-project.org/web/packages/knitr/knitr.pdf, 25). In R: install.packages("rmarkdown"); library(rmarkdown); render("pfad/name_Rmarkdown_datei.Rmd", "html_document")
Dies erfordert die Installation von pandoc (http://pandoc.org, wird automatisch mit RStudio installiert).

[30] Eine Anleitung finden Sie unter http://rmarkdown.rstudio.com/authoring_basics.html (aufgerufen 2015-12-17).

einer Präsentation Ihrer Forschung erhalten Sie durch Ihre Betreuerin bzw. Betreuer die Empfehlung, weitere oder andere unabhängige Variablen aufzunehmen. Sie führen diese Auswertungen durch und kopieren erneut händisch einzelne Ergebnisse in Ihren Bericht.

Option b: Sie erstellen ein dynamisches Dokument, in das Sie Ihre Dokumentation und den Code aufnehmen. Outputs des Codes, bspw. die Ergebnisse der linearen Regression, werden an geeigneten Stellen ausgegeben. Ihnen wird empfohlen, Auswahl und Inhalt der Variablen Ihrer Analyse anzupassen. Sie verändern den Code und kompilieren das Dokument. Es sind keine weiteren Änderungen notwendig. Die Konsistenz von Code und Dokumentation ist gewährleistet.

Abbildung 12: Beispiel: Vorteile von dynamischen Dokumenten

Nicht jeder Code und Output müssen im kompilierten Dokument ausgegeben werden. knitr bietet eine Vielzahl von Optionen an, mit denen Sie bestimmen können, was letztlich im Endbericht zu sehen sein soll (vgl. Xie 2015, Gandrud 2015: 45, 154-156).

Anhang C. Beispiel für ein dynamisches Dokument

````
---
title: "my_markdown"
author: "Thomas Ebel"
date: "17 Dezember 2015"
output: html_document
---

Dies ist ein R Markdown Dokument. Markdown ist eine einfache Formatierungs-(=Markup) Sprache, um HTML,
PDF oder Word Dokumente zu erstellen.
Dieses Beispiel basiert auf dem RStudio-Beispiel, das automatisch mit einer neuen
R Markdown-Datei erstellt  wird. Für mehr Informationen siehe <http://rmarkdown.rstudio.com>

**Fließtext**
Wie Sie sehen: Fließtext, Dokumentation, steht in R Markdown ohne besondere Kennzeichnung. Sie können
Formatierungen vornehmen. Um bspw. Zeichen fettzudrucken, umgrenzen Sie den Text auf beiden Seiten
zweimal mit dem "*"-Zeichen.

**R Code Chunks**
R Code wird entweder über sog. Code Chunks eingebunden.
```{r}
summary(cars)
```
Oder - wenn in einer Zeile zusammen mit Dokumentation - über `` `r ` ``. Die maximale Geschwindigkeit
ist `r max(cars$speed)`.

**Optionen** ermöglichen es, einzustellen, ob der Code und dessen Output zusätzlich zum Fließtext
ausgegeben werden sollen.
```{r, echo=FALSE}
plot(cars)
```

`echo = FALSE` bedeutet, nur der Output, nicht der Code, erscheint im kompilierten Dokument.

`include = FALSE` bedeutet, weder Code noch Output werden angezeigt (der Code wird ausgeführt).

Es gibt viele weitere Optionen, siehe Link oben oder Gandrud 2014: 146ff.
````

Abbildung 13: Beispiel eines R-Markdown Dokuments (dynamisches Dokument)

my_markdown

Thomas Ebel

17 Dezember 2015

Dies ist ein R Markdown Dokument. Markdown ist eine einfache Formatierungs-(=Markup) Sprache, um HTML, PDF oder Word Dokumente zu erstellen. Dieses Beispiel basiert auf dem RStudio-Beispiel, das automatisch mit einer neuen R Markdown-Datei erstellt wird. Für mehr Informationen siehe http://rmarkdown.rstudio.com

Fließtext Wie Sie sehen: Fließtext, Dokumentation, steht in R Markdown ohne besondere Kennzeichnung. Sie können Formatierungen vornehmen. Um bspw. Zeichen fettzudrucken, umgrenzen Sie den Text auf beiden Seiten zweimal mit dem **-Zeichen.

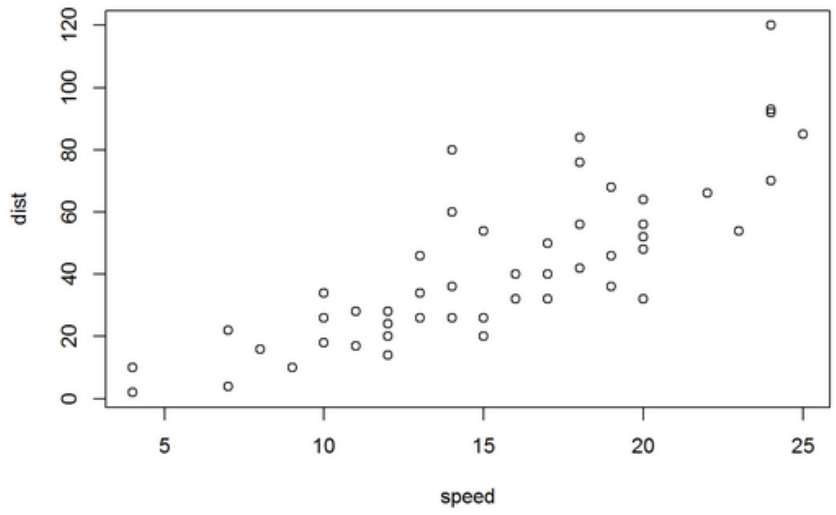
R Code Chunks R Code wird entweder über sog. Code Chunks eingebunden.

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Oder - wenn in einer Zeile zusammen mit Dokumentation - über `` `r ` ``. Die maximale Geschwindigkeit ist 25.

Optionen ermöglichen es, einzustellen, ob der Code und dessen Output zusätzlich zum Fließtext ausgegeben werden sollen.



`echo = FALSE` bedeutet, nur der Output, nicht der Code, erscheint im kompilierten Dokument.

`include = FALSE` bedeutet, weder Code noch Output werden angezeigt (der Code wird ausgeführt).

Es gibt viele weitere Optionen, siehe Link oben oder Gandrud 2014: 146ff.

Abbildung 14: Kompilierter Output/HTML-Datei