Contents

[Einleitung 2](#_Toc59309950)

[Allgemeine Workflow Regeln 2](#_Toc59309951)

[Dateistruktur 3](#_Toc59309952)

[1. data 3](#_Toc59309953)

[2. plots 3](#_Toc59309954)

[3. scripts 3](#_Toc59309955)

[4. .checkpoint 3](#_Toc59309956)

[Kommentare 4](#_Toc59309957)

[Strukturierende Kommentare 4](#_Toc59309958)

[Erklärende Kommentare 5](#_Toc59309959)

[Das roxygen2-Format 5](#_Toc59309960)

[Tipps für den Umgang mit R und R Studio 5](#_Toc59309961)

[R-Studio Theme 5](#_Toc59309962)

[Über R-packages und checkpoint 6](#_Toc59309963)

[Daten laden und speichern 6](#_Toc59309964)

[Codelänge 6](#_Toc59309965)

[Zufallsdaten 6](#_Toc59309966)

[Variablen, Funktionen und die Benennung 7](#_Toc59309967)

[Praktische R-Befehle 7](#_Toc59309968)

[Weiterführende Literatur: 8](#_Toc59309969)

[Questions to SH 9](#_Toc59309970)

## Einleitung

Ein R-Skript zu schreiben ist eine Sache. Ein ganzes Projekt zu organisieren eine andere. Es werden Fragen auftauchen wie: Wo speichert man die Daten? Wie benennt man die Dateien? Wie lädt man die Dateien in den Skripten unabhängig von dem Computer, an dem man sich gerade befindet? Wie erzeugt, sammelt und berichtet man die Ergebnisse? Es scheint eine gute Idee zu sein, ein paar fundierte Antworten zu haben!

Antworten und Anleitungen ohne konkrete Beispiele sind nur die halbe Miete. Darum liegt diese Anleitung in einem Beispiel Projekt, dass genau die hier angeführten Regeln verwendet und jederzeit als Rohling für eigene Projekte verwendet werden kann. Dabei gibt es ein how2R\_example und ein how2R\_template

Obwohl der Guide in deutscher Sprache geschrieben ist, werden Ordner, Scripte und R-Code in englischer Sprache geschrieben sein. Das kommt aus einer praktischen Erwägung. Warum Skripte deutsch benennen und beschriften, wenn sie für eine Publikation dann ins englische übersetzt werden müssen?

## Allgemeine Workflow Regeln

Es gibt eine Zillionen verschiedener Antworten im Netz, wie der effizienteste Workflow mir R aussieht. Aber alle diese Arbeitsabläufe haben die gleichen allgemeinen Antworten gemeinsam.

- **Übersichtlichkeit:** Die Objekte des Projekts sind logisch und übersichtlich organisiert, so dass ein Beobachter einen einfachen Weg hat, zu verstehen: Wo finde ich die Dinge im Projekt? Was macht die Datei, die ich mir gerade anschaue? Informative Dateinamen, eine README.txt und eine logische Dateistruktur sind ein guter Ansatz.

- **Wartbarkeit:** Das Projekt kann leicht verändert und angepasst werden. Standardisierte Skriptnamen, aussagekräftige Kommentierungen und das Auslagern von Stilentscheidungen in Variablen sind hier der Schlüssel.

**- Modularität:** Wiederkehrende Aufgaben werden in separate Komponenten (z.B. Skripte) ausgelagert, so dass immer klar ist, wo Änderungen vorgenommen werden müssen und Komponenten wie Funktionen für andere Projekte wiederverwendbar sind!

- **Portabilität:** Es ist einfach das Projekt auf ein anderes System oder eine andere Person zu übertragen. Das kann durch relative Dateinamen und ein Versionskontrollsystem wie GitHub erreicht werden.

- **Reproduzierbarkeit:** Es ist einfach den Weg zu den Ergebnissen zu reproduzieren und nachzuvollziehen.

Gerade in Arbeitsgruppen und Forschungsgruppen ist es sinnvoll Projekte und Code ähnlich oder gleich zu strukturieren. So ist es leicht sich in den Skripten anderer zurecht zu finden. Im Folgenden wird ein Weg empfohlen, um R-Projekte zu strukturieren und zu codieren, mit dem Ziel die Lesbarkeit von Projektordnern und Skripten zu verbessern und vereinheitlichen.

## Dateistruktur

Wenn eine Person das Projekt öffnet, wird sie normalerweise eine bestimmte Frage im Kopf haben. Das Ziel sollte sein, diese Frage so schnell wie möglich zu beantworten. Dazu ist es notwendig, dass man leicht herausfinden kann, in welchem Ordner und in welcher Datei sich die Antwort befindet. Dazu bieten sich folgende Ordner und Unterordner (mit bspw. 1.1 gekennzeichnet) an.

### 1. data

Es ist immer eine gute Idee, alle datafiles an einem Ort zu speichern. Dieser Ordner ist dieser Ort. Die Daten werden hier in zwei Kategorien aufgeteilt.

#### 1.1 raw

Unterscheide zwischen den Rohdateien, die du niemals bearbeiten solltest...

#### 1.2 processed

...und den Daten, die Sie in Ihrem Analyseprozess erstellt oder modifiziert haben.

### 2. plots

Wenn du in deiner Analyse plots erzeugst, speichere diese! In scripts\\helpers ist eine solche Funktion. Auf diese Weise musst du nicht das gesamte Skript neu starten, um einen Plot nachzuschlagen.

### 3. scripts

Habe nicht alle deine Analysen in einem Skript! Versuche, Skripte für bestimmte Zwecke zu schreiben, wie z.B.: Kennenlernen der Daten, RSA-Analyse des IQ gegen andere Variablen und so weiter.

#### 3.1 helpers

Lagere Code, den du mehr als zwei Mal verwendst, immer in Funktionen aus. Speichere jede davon in einem eigenen R-Skript. Das hält deinen Code übersichtlich und spart Arbeit, wenn du diesen Codeblock für alle Anwendungsfälle ändern willst.

### 4. .checkpoint

Hier werden datierte Versionen von R-Paketen abgespeichert. Dadurch funktioniert der Code auch dann noch, wenn ein Paket geupdatet wird. Die alte Version ist hier gespeichert.

Direkt im Projektordner liegen folgende Dateien:

*NamedesProjekts.RProjekt*verwende immer eine Rproject-Datei. Das ist die Basis für relative Dateinamen denn es setzt dein Arbeitsverzeichnis dorthin, wo die .RProject-Datei liegt! Zudem wird dein letzter Arbeitsstand gespeichert. Wenn du also ein Skript aus deinem Projekt starten willst, öffne erst die .Rproject Datei und dann das gewünschte Skript. Jetzt laufen alle Befehle, die auf andere Dateien zugreifen reibungslos, egal wo dein Projektordner gerade ist.

*README.txt*Schreibe eine README niemals als .docx. Nur als .txt! Auf diese Weise kann sie jeder ohne Formatierungsprobleme öffnen und lesen. Eine README sollte:

- kurz erklären, worum es in dem Projekt geht.

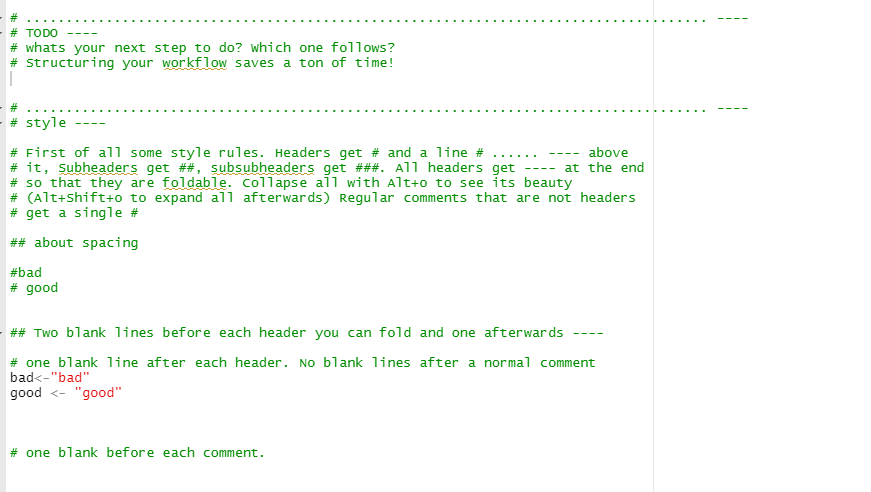
- ein Inhaltsverzeichnis / eine Übersicht über die Dateien enthalten. Das macht das Auffinden der gesuchten Dinge noch einfacher

## Kommentare

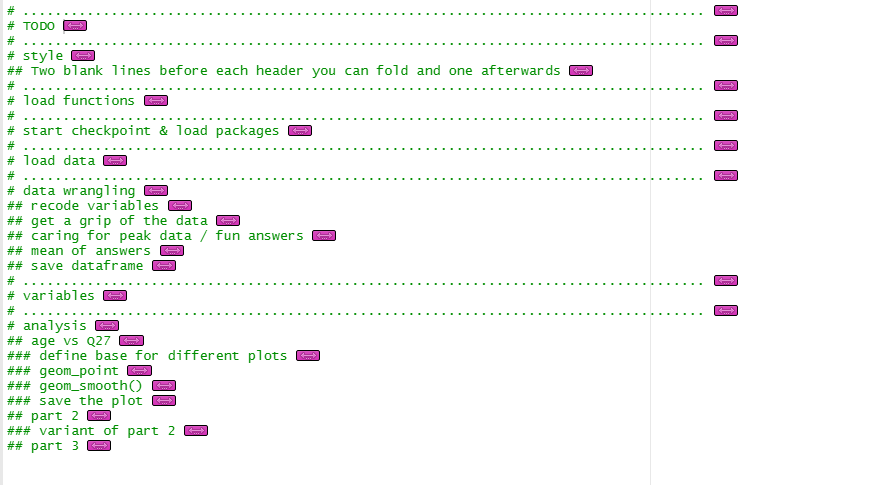
Es gibt **drei Typen von Kommentaren**. Kommentare, die erklären was eine spezifische Codezeile tut, Kommentare die den Code strukturieren und Kommentare im roxygen2 Format, die die Funktionsweise von Funktionen erklären.

### Strukturierende Kommentare

Dieser Guide ist auch in scripts/how2r\_example\_script.R zu finden.



Ein gut strukturiertes Skript sieht dann, wenn alles eingeklappt ist, so aus:



### Erklärende Kommentare

**Kommentieren heißt dokumentieren**. Kommentiere immer dann, wenn etwas nicht selbsterklärend ist. Kommentiere immer in der Zeile, in der sich auch der kommentiere Code befindet.

Schreibe erklärende Kommentare nicht in mehreren Zeilen, egal wie lang die eine Zeile wird. Personen, die eine Erklärung für die Codezeile wollen, drücken einfach Strg+Shift+/, um ihn umzubrechen und zu lesen. Alle anderen können den Code lesen, ohne von Kommentarwüsten erschlagen zu werden.

Beschreiben Sie beim Kommentieren, wie und warum etwas passiert. Was passiert, sollte durch das Lesen der Header und der Dateinamen klar sein.

### Das roxygen2-Format

Dieses Format ist ein Kommentarstil, der für die Erklärung und Dokumentation von Funktionen entwickelt wurde. Ein Beispiel findest du unter script/helpers. Verwende #', um den roxygen2-Kommentarstil aufzurufen. Die folgenden Befehle sind verfügbar

- @title: kurzer Satz, was der Code macht

- @description: erweiterte Beschreibung des Codes, falls erforderlich

- @details: Alternative zu @Beschreibung

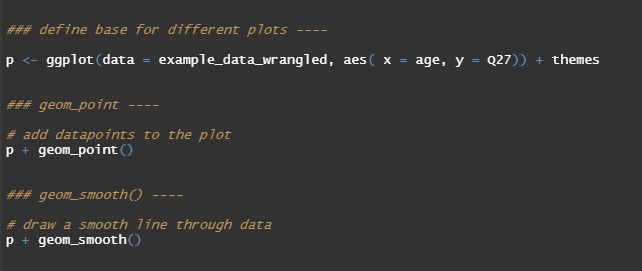
- @param: welcher Datentyp ist der Parameter? Wofür ist er gedacht?

- @return: beschreibt die Ausgabe des Codes

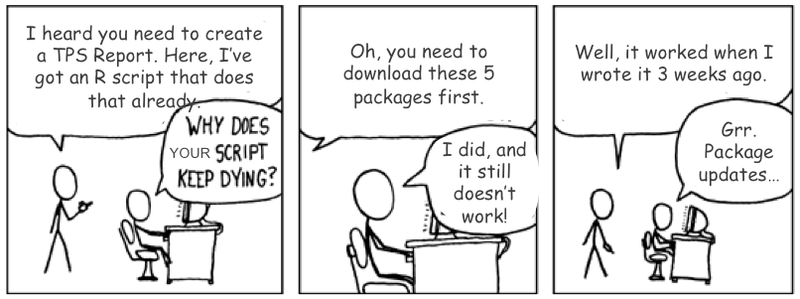
- @Beispiele: Beispielcode, der die Funktionsweise des Codes zeigt

## Tipps für den Umgang mit R und R Studio

### R-Studio Theme

Unter Tools -> Global Options -> Appearances -> Editor-Theme kannst du das Aussehen deines R Studio ändern. Tu deinen Augen einen Gefallen und ändere es zu einem mit schwarzem Hintergrund. Ein solches Theme heißt IdleFingers und sieht dann so aus:

### Über R-packages und checkpoint



R ist eine lebendige Sprache. Darum werden packages die du benutzt von den Entwickler\*innen hin und wieder geupdatet. Das ist erstmal toll, denn es führt häufig zu Funktionen die einfacher zu verstehen sind und besser funktionieren. Dabei können aber auch Namen von Variablen einer Funktion oder die Funktionsweise verändert werden, so dass dein Script von einem auf den anderen Tag nicht mehr funktioniert. Dafür gibt es das checkpoint package. Mit diesem Paket entscheidest du dich zunächst für ein Datum. Am besten der Beginn deines Projektes. Nun werden im Folgenden die Versionen der verwendeten Pakete zu diesem Datum gedownloadet, abgespeichert und genutzt

Daten laden und speichern  
Verwende eine .Rproject Datei und relative Dateipfade, wie in diesem Beispiel:  
 

Dadurch kannst du deinen Projektordner an beliebige andere Orte kopieren und die Daten werden trotzdem geladen. Speichere alle (Zwischen-)Ergebnisse. Seien es plots, bearbeitete Daten oder Ergebnistabellen. Verwende dazu die Funktionen save() oder saveplot().

### Codelänge

Jeder zusammenhängende Codeblock sollte einen einzelnen Bildschirm (ca. 50 Codezeilen) nicht überschreiten. So kann er gelesen und verstanden werden, ohne dass gescrollt werden muss. Ein Überschreiten der Grenze ist ein gutes Zeichen, dass ein Teil des Codes in eine Funktion ausgelagert werden sollte.

Jede Codezeile sollte 80 Zeichen pro Zeile nicht überschreiten. Das passt auf eine gedruckte Seite und kann daher von einem Menschen leicht verarbeitet werden.

### Zufallsdaten

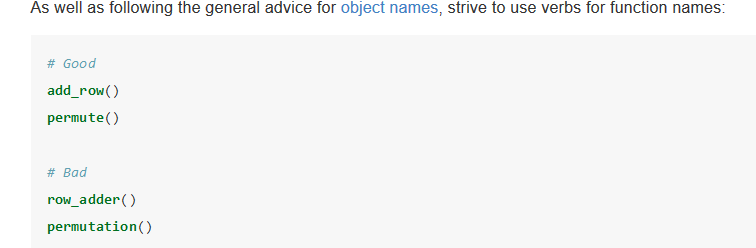
Bei Zufallsfunktionen kann es schnell passieren, dass andere Personen andere Ergebnisse erhalten. Das ist gerade bei geplotteten Graphen sehr schade. Das lässt sich verhindern, indem wir immer die gleichen Zufallszahlen produzieren. Das geht mit dem set.seed(eine Zahl deiner Wahl einfügen) Befehl. Hiermit stellst du sicher, dass bei jeder Ausführung des Skriptes dieselben Zufallszahlen erstellt und genutzt werden.

### Variablen, Funktionen und die Benennung

#### Benennen von Variablen

Wähle einen intuitiven Namen für Funktionen und Daten.

* Eine Funktion sollte immer nach dem benannt werden, was sie tut. Verwende Verben für Funktionsnamen!



* Eine Variable sollte nach dem benannt werden, was sie enthält.

#### Variablen und Funktionen

Nehmen wir an, du hast ein Skript, das 20 Diagramme aufzeichnet. Nun möchtest du den Hintergrund aller Plots auf weiß ändern. Das wäre jetzt echt umständlich, in allen 20 Zeilen Code den entsprechenden Befehl anzupassen. Hier würde sich eine Variable mit dem Namen backgroundcolor\_plots eignen. Diese steht stellvertretend in allen Codezeilen, in denen die Farbe des plot Hintergrund geändert wird und wird ganz zu Beginn des Skriptes definiert. Jetzt reicht es, ganz zu Beginn den Wert der Variable zu verändern.

Das geht auch mit anderen Stilentscheidungen. Wenn möglich, mach daraus eine Variable oder eine Funktion! Einige Beispiele dafür gibt es im Beispiel Skript.

## Praktische R-Befehle

|  |  |
| --- | --- |
| Strg + Shift + C | Alle ausgewählten Zeilen auf einmal kommentieren/unkommentieren |
| Alt + O | Alle Ordner einklappen |
| Alt + Shift + O | Alle Ordner ausklappen |
| Strg + Shift + / | Lange Zeile eines Kommentars in mehrere Teilen |
|  |  |

## Workflow mit Git

Git ist eine kostenlose Versionierungs software. Versionierung heißt, dass du dein Projekt immer wieder zwischenspeichern kannst. So kannst du immer wieder zu alten Versionen zurückkehren. Und außerdem kannst du gemeinsam mit anderen Menschen an einem Projekt arbeiten und nachvollziehen, was sie für Änderungen vorgenommen haben. Wenn du Git und GitHub noch nicht kennst, schau dir den Guide an! Für alle die es bereits kennen gibt es hier einige Workflow Regeln.

1. Checke in GitHub ob es Änderungen im Projekt gab und pulle diese
2. Arbeite an deinem Projekt
3. Wenn eine sinnvolles Arbeitspaket fertig ist, schreibe einen commit. Nicht früher, nicht später! Im commit sollte stehen:
   1. In die erste Zeile: Was macht der commit?
   2. In der Beschreibung: Was verändert der commit und warum?
4. Löse eventuelle merge Konflikte
5. Pushe den commit

Und als **Reminder:**

GitHub ist keine sichere Plattform um Passwörter oder sensible Daten aufzubewahren! Speichere diese immer nur lokal ab und füge die entsprechenden Order zu .gitignore hinzu!

## Weiterführende Literatur:

Kleine Tipps und Reminder Sammlung: <https://www.r-statistics.com/2010/09/managing-a-statistical-analysis-project-guidelines-and-best-practices/>

5 Workflow Regeln für R: <https://blog.revolutionanalytics.com/2010/10/a-workflow-for-r.html>

Ausführliche Beschreibung und viele Best Practice Tipps für ein R-Projekt: <https://chrisvoncsefalvay.com/2018/08/09/structuring-r-projects/>

Ausführlicher offizieller tidyverse style Guide für Code: [*https://style.tidyverse.org/*](https://style.tidyverse.org/)

Ausführlicher ergänzender Style Guide für Code mit weiteren wertvollen Punkten:   
[*https://www.r-bloggers.com/2019/01/%f0%9f%96%8a-r-coding-style-guide/*](https://www.r-bloggers.com/2019/01/%f0%9f%96%8a-r-coding-style-guide/)

Video mit theoretischen Überlegungen, was ist wichtig am Schreiben? Einordnung für R-Coding <https://rviews.rstudio.com/2016/12/02/writing-good-r-code-and-writing-well/>

Einführung zum checkpoint package: <https://www.r-bloggers.com/2014/10/introducing-the-reproducible-r-toolkit-and-the-checkpoint-package/>

### Questions to SH

SF@SH Als Praxis übernehmen? Ist hier bei dir schonmal was schief gelaufen?

