



Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2024/25

Belinda Fleischmann

Datum	Einheit	Thema	Form
15.10.24	R Grundlagen	(1) Einführung	Seminar
22.10.24	R Grundlagen	(2) R und Visual Studio Code	Seminar
29.10.24	R Grundlagen	(2) R und Visual Studio Code	Übung
05.11.24	R Grundlagen	(3) Vektoren, (4) Matrizen	Seminar
12.11.24	R Grundlagen	(5) Listen und Dataframes <i>Leistungsnachweis 1</i>	Seminar
19.11.24	R Grundlagen	(6) Datenmanagement	Seminar
26.11.24	R Grundlagen	(2)-(6) R Grundlagen	Übung
03.12.24	Deskriptive Statistik	(7) Häufigkeitsverteilungen	Seminar
10.12.24	Deskriptive Statistik	(8) Verteilungsfunktionen und Quantile <i>Leistungsnachweis 2</i>	Seminar
17.12.24	Deskriptive Statistik	(9) Maße der zentralen Tendenz und Datenvariabilität Weihnachtspause	Seminar
07.01.25	R Grundlagen	(10) Strukturiertes Programmieren: Kontrollfluss, Debugging	Seminar
14.01.25	Deskriptive Statistik	(11) Anwendungsbeispiel <i>Leistungsnachweis 3</i>	Übung
21.01.25	Deskriptive Statistik	(11) Anwendungsbeispiel	Seminar
28.01.25	Deskriptive Statistik	(11) Anwendungsbeispiel, Q&A	Seminar

(6) Datenmanagement

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

- Zahlenarrays
- Characterarrays
- Software
- Digitale Werkzeuge
- Workflows
- Analysispipelines
- u.v.a.m.



Forschungsdaten

“Grundsätzlich handelt es sich bei **Forschungsdaten** um elektronisch repräsentierte analoge oder digitale Daten, die im Zuge wissenschaftlicher Vorhaben entstehen oder genutzt werden, z.B. durch Beobachtungen, Experimente, Simulationsrechnungen, Erhebungen, Befragungen, Quellenforschungen, Aufzeichnungen von Audio- und Videosequenzen, Digitalisierung von Objekten, und Auswertungen.”

Rat für Informationsinfrastrukturen

Empfehlungen zur Nutzung und Verwertung von Daten im wissenschaftlichen Raum (09/2021)

Herausforderung Datenqualität (11/2019)

Digitale Kompetenzen – dringend gesucht! (07/2019)

Aktuelle Empfehlungen zu Datenschutz und Forschungsdaten (03/2017)

Metadaten repräsentieren Information über Daten

Deskriptive Metadaten dienen dem Auffinden und der Identifikation einer Datenquelle. Beispiele für deskriptive Metadaten sind Titel, Abstrakt, Autor:in, oder Keywords einer wissenschaftlichen Publikationen.

Strukturelle Metadaten sind Metadaten über Datencontainer und repräsentieren den strukturellen Aufbau einer Datenquelle. Beispiele sind die Ordnung der Seiten eines Buches, oder die Schleifenenkodierung dreidimensionaler Datenobjekte.

Administrative Metadaten sind Daten, die das Management einer Datenquelle erleichtern. Beispiele sind die Provenienz, das Dateiformat, die Zugangsrechte, oder weitere technische Informationen zu einer Datenquelle.

Daten

FAIR Prinzipien

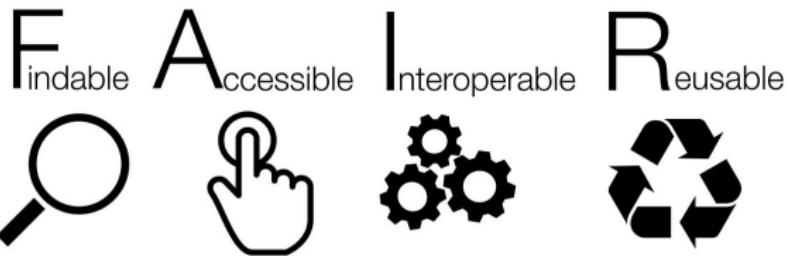
Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Das FAIR Datenideal



für Menschen und Maschinen

„Jointly designing a data fairport“ workshop in Leiden 2014

FORCE11

Wilkinson et al.(2016) The FAIR Guiding Principles for scientific data management and stewardship Scientific Data 160018

[go-fair.org/FAIR Principles](http://go-fair.org/FAIR%20Principles)

- F1. (Meta)Daten haben einen persistenten global einzigartigen Identifikator.
- F2. Daten werden mit Metadaten angereichert.
- F3. Metadaten sind zweifelsfrei einem Datensatz zuzuordnen.
- F4. (Meta)Daten sind in einer durchsuchbaren Ressource indexiert.

A1. (Meta)Daten sind mit standardisierten Protokollen abrufbar.

A1.1. Das genutzte Protokoll ist offen, kostenlos und nutzbar.

A1.2. Das Protokoll ermöglicht Authentifizierung und Rechtevergabe.

A2. Metadaten bleiben zugänglich, auch wenn Daten nicht mehr vorliegen.

- I1. (Meta)Daten nutzen eine formale, zugängliche, gemeinsam genutzte und breit anwendbare Sprache zur Wissensrepräsentation.
- I2. (Meta)Daten nutzen Vokabularen, die den FAIR-Prinzipien folgen.
- I3. (Meta)Daten enthalten qualifizierte Referenzen auf andere (Meta)Daten.

R1. (Meta)Daten haben eine Vielzahl genauer und relevanter Attribute.

R1.1. (Meta)Daten enthalten eine eindeutige Nutzungslizenz.

R1.2. (Meta)Daten enthalten detaillierte Provenienz-Informationen.

R1.3. (Meta)Daten genügen den Standards der jeweiligen Fachcommunity.

- Die FAIR Prinzipien sind ein anzustrebendes Datenmanagementideal.
- Der Umgang mit digitalen Forschungsdaten ist oft noch sehr unstrukturiert.
- Die Universitäten begreifen das digitale Datenmanagement nur sehr langsam.
- Die Digitalisierung bleibt eine gesellschaftliche Hauptaufgabe.
- Die [NFDI Initiative](#) versucht, deutsches Wissenschaftsdatenmanagement zu verbessern.
- Beteiligung von OVGU // CBBS an [NFDI Neurowissenschaft](#).
- NFDI ist dezentral, community, und Drittmittelprojekt-basiert ⇒ Nicht nachhaltig.
- Nicht alle Wissenschaftler:innen wollen ihre Daten organisieren und teilen.
- [Open Science](#) bleibt eine wichtige Initiative verantwortungsvoller Wissenschaftler:innen.

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Datenformate

Dateiformate

- Ein Dateiformat definiert Syntax und Semantik von Daten innerhalb einer Datei.
- Dateiformate sind bijektive Abbildungen von Information auf binären Speicher.
- Allgemein unterscheidet man
 - Daten- gegenüber Softwareformaten,
 - textuelle gegenüber binären Dateiformaten, und
 - offene gegenüber proprietären (urheberrechtlich geschützten) Dateiformaten.

Binäre Dateiformate

- Einlesen, Inspektion, und Manipulation ist nur mit spezieller Software möglich.
- .pdf, .xlsx, .jpg, .mp4 sind binäre Dateiformate.
- Binäre Dateiformate sind oft proprietär.
- Binäre Dateiformate wurden früher aufgrund ihrer kleineren Größe bevorzugt eingesetzt.

Textuelle Dateiformate

- Einlesen, Inspektion, und Manipulation ist mit einfachen allgemeinen Editoren möglich.
- .txt, .csv., .tsv, .json sind textuelle Dateiformate.
- Textuelle Dateiformate sind generell offene Dateiformate.

Beispiele

Binäres Dateiformat

```
cda_1_algorithmen_und_programme - Editor

Datei Bearbeiten Format Ansicht Hilfe
PKW<|||> ! B[<|||> F<|||> [Content_Types].xml ()+
<|||> <|||> "application/vnd.ms-office+xml">;#<|||> /Kilobyte<|||> k-v<|||> IS&#226;&#226;JFqVI 2V<|||> -f"X&#226;MXC&#226;A,">)X&#226;iY&#226;zS
6@<|||> :So(<|||> m/nRCFCYAb!> L&#226;ft@5+?U<|||> H/xZmDXI&#226;OU\&#226;AÜ; >^<|||> p&#226;y&#226;lo&#226;S$. ->
<|||> ^&#226;J@<|||> jll@-4LkAc&#226;a80CZY!> ?U&#226;@> ül&#226;ua&#226;zfYfR+&#226;0B!>.3i*<|||> X&#226;evpV!> Ek&#226;In+PU<|||> l&#226;CHT<|||> JQ!.>v&#226;G
<|||> d7D&#226;eO&#226;sS!> <|||> y&#226; PKW<|||> !> h&#226;t@<|||> ä<|||> _rels/.rels ()(
<|||> ^&#226;k<|||> A&#226;x&#226;B+xn!> 2E78JO
<|||> @<|||> ^&#226;H.D.Cy<|||> ^&#226;BRY&#226;Ai&#226;lz +>
<|||> |<|||> ^&#226;19&#226;nL&#226;P<|||> ^&#226;0:1@<|||> ^&#226;~2<|||> ^&#226;O&#226;H([s]<|||> ^&#226;Dü:<|||> b4&#226;E<|||> uH<|||> ^&#226;0L<|||> ^&#226;je&#226;ß&#226;K<|||> K9&#226;U!> f&#226;Z&#226;W<|||> ^&#226;h&#226;C!> ^&#226;8&#226;ß&#226;Mh&#226;u<|||> +&#226;L%> EC:<|||> Ü<|||> $tr<|||> B&#226;M&#226;de&#226;R&#226;X<|||> ?> ?> ^&#226;U<|||> +>
```

Textuelle Dateiformat

```
File Edit View Help  
"Control" "drug1" "drug2L" "drug2R" "delta1" "delta2L" "delta2R"  
"1" 0.6 1.3 2.5 2.1 0.7 1.9 1.5  
"2" 3 1.4 3.8 4.4 -1.6 0.8 1.4  
"3" 4.7 4.5 5.8 4.7 -0.2 1.1 0  
"4" 5.5 4.3 5.6 4.8 -1.2 0.1 -0.7  
"5" 6.2 6.1 6.1 6.7 -0.1 -0.1 0.5  
"6" 3.2 6.6 7.6 8.3 3.4 4.4 5.1  
"7" 2.5 6.2 8 8.2 3.7 5.5 5.7  
"8" 2.8 3.6 4.4 4.3 0.8 1.6 1.5  
"9" 1.1 1.1 5.7 5.8 0 4.6 4.7  
"10" 2.9 4.9 6.3 6.4 2 3.4 3.5
```

- CSV = Comma- (oder auch character)-separated values, Dateiendung .csv
- Zentrales Format zur Speicherung einfacher strukturierter Daten
- Repräsentation zeilenweise miteinander verknüpfter Datensätze
 - Trennung von Datenfeldern (Spalten) durch Komma oder Tab (TSV, .tsv)
 - Trennung von Datensätzen (Zeilen) durch Zeilenumbruch
- Erster Datensatz typischerweise Kopfdatensatz (Header) mit Spaltennamendefinition

Beispiel

- Einheit (experimental unit) repräsentiert z.B. eine Versuchsperson

.csv Dateinhalt		Tabellenrepräsentation		
Einheit	Variable 1, Variable 2	Einheit	Variable 1	Variable 2
1	10.1, 67.5	1	10.1	67.5
2	12.9, 51.2	2	12.9	51.2
3	20.4, 70.8	3	20.4	70.8

Wide vs. Long Format

Wide Format: Alle Variablen einer Einheit in einer Zeile

Einheit	Variable 1	Variable 2
1	10.1	67.5
2	12.9	51.2
3	20.4	70.8

Long Format: Variablen einer Einheit über Zeilen verteilt

Einheit	Variable	Messwert
1	Variable 1	10.1
	Variable 2	67.5
2	Variable 1	12.9
	Variable 2	51.2
3	Variable 1	20.4
	Variable 2	70.8

Das Wide Format ist generell übersichtlicher als das Long Format

Übersicht

- JSON = JavaScript Object Notation
- Textuelles Datenformat zum Speichern strukturierter Daten in Key-Value Form.
- Ähnlichkeit mit R Listen mit benannten Listenelementen.
- Sinnvolles Format für das Speichern von Metadaten.

Elemente von JSON Dateien

- *Objekte* enthalten durch Kommata geteilte Liste von *Eigenschaften* in { }
- *Eigenschaften* bestehen aus Key-Value Paaren
- *Key* ist immer ein String mit Hochkommata “ ”
- *Value* ist ein Objekt, ein Array, ein String, ein Boolean, oder eine Zahl

JSON - Beispiel

```
{  
    "Vorname" : "Maxi",  
    "Nachname" : "Musterfrau",  
    "Matrikelnummer" : 12345,  
    "Fachsemester" : 2,  
    "Studiengang" : "BSc Psychologie",  
    "Module" :  
    {  
        "Deskriptive Statistik" : {"Abgeschlossen": true, "Note" : 1.0 },  
        "Inferenzstatistik" : {"Abgeschlossen" : false, "Note": null }  
    }  
}
```

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Arbeiten mit Strings

Die Grundeinheit für Text in R sind atomic vectors vom Typ **character**.

Die Elemente von character vectors sind **strings**, nicht einzelne "characters".

Der Begriff "String" in R ist also nur informeller Natur.

Strings werden mit Anführungszeichen oder Hochkommata erzeugt

```
c("Dies ist ein character vector") # Anführungszeichen sind der String Standard
```

```
[1] "Dies ist ein character vector"
```

```
c('Dies ist ein "string"')           # Hochkommata nützlich für Anführungszeichen im String
```

```
[1] "Dies ist ein \"string\""
```

paste() konvertiert Vektoren in character und fügt sie elementweise zusammen.

```
paste(1, 2)                      # Konvertierung u. Konkatenation einelementiger double vectors
```

```
[1] "1 2"
```

```
paste("Dies ist", "ein String")    # Konkatenation einelementiger character vectors
```

```
[1] "Dies ist ein String"
```

Arbeiten mit Strings

`paste()` hat eine Reihe von weiteren Funktionalitäten

```
paste(c("Rote", "Gelbe"), "Blume")          # Vector recycling, elementweise Veknüpfungen

[1] "Rote Blume"  "Gelbe Blume"
paste(c("Rote", "Gelbe"), "Blume",  sep = "-")      # Separatorspezifikation

[1] "Rote-Blume"  "Gelbe-Blume"
paste(c("Rote", "Gelbe"), "Blume",  collapse = ", ") # Zusammenfügen mit spezifiziertem Separator

[1] "Rote Blume, Gelbe Blume"

'toString()' ist eine paste() Variation für numerische Vektoren
toString(1:10)          # Konversion eines double Vektors in formatierten String

[1] "1, 2, 3, 4, 5, 6, 7, 8, 9, 10"
toString(1:10, width = 10)    # mit Möglichkeit der Beschränkung auf width Zeichen

[1] "1, 2, ...."
```

Datei- und Verzeichnispfade

- Daten sind üblicherweise in Dateien im permanenten Speicher (SSD, HD) abgelegt
- Um Daten einzulesen, benötigt man ihre Position bzw. Adresse innerhalb der Verzeichnisstruktur des Rechners.
- Die Addressen von Dateien in der Verzeichnisstruktur heißen *Dateipfade*.
- Ein Pfad besteht aus einer durch Schrägstriche getrennten Liste von Verzeichnisnamen.
- Die Art der Schrägstriche hängt vom Betriebssystem ab.
 - Windowspfade haben Back-Slashes (\)
 - Pfade in Unix-ähnlichen Betriebssystemen (z. B. macOS und Linux) nutzen Forward-Slashes (/)

Beispiele

Windows:

H:\Lehre\Daten
H:\Lehre\Daten\cushny.csv

Pfad der auf einem Verzeichnisnamen endet
Pfad der auf einem Dateinamen endet

Unix-like OS:

/home/user/Lehre/Daten
/home/user/Lehre/Daten/cushny.csv

Pfad der auf einem Verzeichnisnamen endet
Pfad der auf einem Dateinamen endet

Working directory

- Der *working directory* kann als der “aktuelle Standort” im lokalen Verzeichnissystem verstanden werden.
- In VSCode wird der beim Start ausgewählte Ordner automatisch zum Working Directory für jede neue R-Terminal-Session.
- Es entspricht dem übergeordneten Verzeichnis, das im Explorer sichtbar ist.

`getwd()` gibt das working directory an.

```
getwd() # Get current working directory
```

```
[1] "/home/belindame_f/OGVU/2024_WiSe_PDS/progr-und-deskr-stat-25/6_Datenmanagement"
```

Ändern des Working directorys

`setwd()` ändert das working directory

- R verwendet Forward-Slashes (/).
- Bei der „wörtlichen“ Angabe eines Windows-Pfades müssen doppelte Backslashes (\\\).

Änderung des Working Directory mit Angabe eines absoluten Pfades

```
setwd("C:\\Lehre\\Daten") # Wechsel zu "wörtlich" angegebenem Pfad (Windows-Bsp.)
```

```
setwd("/home/belindame_f/Lehre/Daten") # Wechsel zu "wörtlich" angegebenem Pfad (Unix Bsp.)  
getwd()
```

```
[1] "/home/belindame_f/Lehre/Daten"
```

Relativ vs. Absolute Pfade

- *Relative Dateipfade* bezieht sich auf einen Speicherort in Relation zum aktuellen Verzeichnis.
- Bei relativen Dateipfaden bezeichnen . und .. aktuelles und übergeordnetes Verzeichnis.
- *Absolute Dateipfade* geben die Adresse in der Gesamtverzeichnisstruktur der Festplatte an.
- Absolute Dateipfade sind weniger anfällig für Dateiverwechslungen.
- **Die Verwendung adaptiv generierter absoluter Pfade wird stark empfohlen.**

Beispiel

Wenn das aktuelle Verzeichnis "/home/user/Lehre" lautet, dann beziehen sich folgende Pfade auf *den selben* Ordner:

```
/home/user/Lehre/Daten # Abs. Pfad: Startet in Root-Verzeichnis ("")  
. /Daten             # Rel. Pfad: Startet in aktuellen Arbeitsverzeichnis (/Lehre)  
.. /Lehre/Daten      # Rel. Pfad: Startet in übergeordneten Verzeichnis (/user)  
... / /user/Lehre/Daten # Rel. Pfad: Startet zwei Ebenen über akt. Arbeitsverzeichnis (/home)
```

Verzeichnismanagement

Änderung des Working Directory mit Angabe eines relatives Pfades

```
setwd("/home/belindame_f/Lehre/Daten") # Wechsel zu "wörtlich" angegebenem Pfad  
getwd()
```

```
[1] "/home/belindame_f/Lehre/Daten"  
setwd("../")                                # Wechsel zum übergeordneten Verzeichnis "/Lehre"  
getwd()
```

```
[1] "/home/belindame_f/Lehre"  
setwd("../")                                # Wechsel zum übergeordneten Verzeichnis "/belindame_f"  
getwd()
```

```
[1] "/home/belindame_f"  
setwd("./Lehre")                             # Wechsel in das Unterverzeichnis "/Lehre"  
getwd()
```

```
[1] "/home/belindame_f/Lehre"  
setwd("../")                                # Wechsel zum übergeordneten Verzeichnis "/belindame_f"  
getwd()
```

```
[1] "/home/belindame_f"  
setwd("Lehre")                               # Wechsel in das Unterverzeichnis "/Lehre"  
getwd()
```

```
[1] "/home/belindame_f/Lehre"
```

Hilfreiche Funktionen für die Definition des Dateipfades

`file.path()` konstruiert Verzeichnis- und Dateipfade Betriebssystem-passend

```
ein_pfad <- file.path("home", "user", "Lehre", "Daten")           # Konstruiert eine character-Variable  
print(ein_pfad)
```

```
[1] "home/user/Lehre/Daten"
```

`basename()` gibt die unterste Ebene eines Datei- oder Verzeichnispfades an.

```
basename(ein_pfad)
```

```
[1] "Daten"
```

`dirname()` gibt den Pfad des übergeordneten Verzeichnisses zurück, das die angegebene Datei oder das angegebene Verzeichnis enthält.

```
dirname(ein_pfad)
```

```
[1] "home/user/Lehre"
```

Hilfreiche Funktionen für die Definition des Dateipfades

`gsub()` ersetzt bestimmte Zeichen in einer character-variable.

```
windows_pfad      <- "C:\\\\Users\\\\Username\\\\Lehre\\\\Daten"  
print(windows_pfad)
```

```
[1] "C:\\\\Users\\\\Username\\\\Lehre\\\\Daten"  
r_readable_pfad <- gsub("\\\\\\\", "/", windows_pfad)  
print(r_readable_pfad)
```

```
[1] "C:/Users/Username/Lehre/Daten"
```

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Datenimport mit `read.table()`

`read.table()`

- ist die zentrale Funktion zum Einlesen von CSV Dateien.
- liest eine Datei ein und speichert ihre Inhalte in einem Dataframe.
- bietet eine Vielzahl weiterer Spezifikationsmöglichkeiten.

```
work_dir_path <- getwd()                                # Pfad zum Working directory
data_dir_path <- file.path(work_dir_path, "Daten")      # Datenverzeichnispfad
file_name     <- "cushny.csv"                            # (base) filename
file_path     <- file.path(data_dir_path, file_name)    # filepath
D            <- read.table(file_path)                   # Einlesen der Datei
print(D)
```

	Control	drug1	drug2L	drug2R	delta1	delta2L	delta2R
1	0.6	1.3	2.5	2.1	0.7	1.9	1.5
2	3.0	1.4	3.8	4.4	-1.6	0.8	1.4
3	4.7	4.5	5.8	4.7	-0.2	1.1	0.0
4	5.5	4.3	5.6	4.8	-1.2	0.1	-0.7
5	6.2	6.1	6.1	6.7	-0.1	-0.1	0.5
6	3.2	6.6	7.6	8.3	3.4	4.4	5.1
7	2.5	6.2	8.0	8.2	3.7	5.5	5.7
8	2.8	3.6	4.4	4.3	0.8	1.6	1.5
9	1.1	1.1	5.7	5.8	0.0	4.6	4.7
10	2.9	4.9	6.3	6.4	2.0	3.4	3.5

Datenimport mit `read.table()`

Einige weitere Spezifikationen bei Anwendung von `read.table()` sind

- `sep` für die Auswahl des Separators
- `dec` für die Auswahl des Dezimalpunktes
- `nrow` für die Anzahl der einzulesenden Zeilen
- `skip` für die Anzahl der am Anfang der Datei zu überspringenden Zeilen (inkl. Header)

```
D <- read.table(file_path, nrow = 2) # Auswahl von nur 2 Zeilen  
print(D)
```

```
Control drug1 drug2L drug2R delta1 delta2L delta2R  
1     0.6   1.3   2.5   2.1   0.7   1.9   1.5  
2     3.0   1.4   3.8   4.4  -1.6   0.8   1.4
```

```
D <- read.table(file_path, skip = 7) # Auswahl ab Zeile 7  
print(D)
```

```
V1 V2 V3 V4 V5 V6 V7 V8  
1 7 2.5 6.2 8.0 8.2 3.7 5.5 5.7  
2 8 2.8 3.6 4.4 4.3 0.8 1.6 1.5  
3 9 1.1 1.1 5.7 5.8 0.0 4.6 4.7  
4 10 2.9 4.9 6.3 6.4 2.0 3.4 3.5
```

Import interner R Datensätze

R und R packages beinhalten eine Vielzahl von Beispieldatensätzen.

Die Core R Datensätze werden aus der R Konsole mit `data()` angezeigt.

Die Datensätze in Paket P werden mit `data(package = "P")` angezeigt.

```
install.packages("psychTools") # Installation des Pakets psychTools  
data(package = "psychTools") # Anzeige der psychTools Datensaetze
```

```
Data sets in package 'psychTools':  
  
Damian          Project Talent data set from Marion Spengler and Rodica Damian  
Pollack         Pollack et al (2012) correlation matrix for mediation example  
Schutz          The Schutz correlation matrix example from Shapiro and ten Berge  
Spengler        Project Talent data set from Marion Spengler and Rodica Damian  
Spengler.stat   Project Talent data set from Marion Spengler and Rodica Damian  
USAF            17 anthropometric measures from the USAF showing a general factor  
ability          16 ability items scored as correct or incorrect.  
ability.keys    16 ability items scored as correct or incorrect.  
affect           Two data sets of affect and arousal scores as a function of personality and movie  
conditions      conditions  
all.income      US family income from US census 2008  
bfi              25 Personality items representing 5 factors
```

Alle Datensätze werden mit `data(package = .packages(TRUE))` angezeigt.

Nach Installation und Laden eines Pakets werden Datensätze mit `data()` geladen.

```
library(psychTools) # Laden des Paktes psychTools  
data(cushny)        # Laden des cushny Datensatzes aus psychTools
```

Weitere Möglichkeiten des Datenimports

CSV und Text Dateien

- `read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()` als `read.table()` Varianten.
- `readlines` für low-level Textdateiimport.
- `fromJSON()` aus dem Paket `rjson` für `.json` Dateien.

Binäre Dateien

- `read.xlsx()` und `read.xlsx2()` aus dem Paket `xlsx` für Excel `.xlsx` Dateien.
- `read.spss()` aus dem Paket `foreign` für SPSS `.sav` Dateien.
- `readMat` aus dem Paket `R.matlab` für Matlab `.mat` Dateien.

Webdaten und Datenbanken

- Twitterdaten können mithilfe der Pakete `rtweet` oder `twitteR` eingelesen werden.
- SQL Datenbanken können mithilfe der Pakete `DBI` und `RSQLite` abgefragt werden.

Datenexport mit write.table()

write.table()

- ...ist die zentrale Funktion zum Speichern von Daten in CSV Dateien.
- ...erzeugt eine Datei und schreibt Daten eines Dataframes hinein.

Spezifikationen bei der Anwendung

- Der Dateipfad wird mit dem Argument file angegeben, der Werteseparator mit sep
- Das Argument row.names = FALSE unterdrückt das Schreiben von Zeilennahmen

```
input_file_path <- file.path(data_dir_path, "cushny.csv") # Pfad zur Datei, die eingelesen werden soll
output_file_path <- file.path(data_dir_path, "student.csv") # Pfad der Datei, die geschrieben werden soll
```

```
D <- read.table(input_file_path)                      # Dateneinlesen
D <- D[,5:6]                                         # Reduktion des Dataframes
write.table(                                           # .csv Schreibfunktion
  D,                                                 # Zu speichernder Dataframe
  file = output_file_path,                          # Dateiname
  sep = ",",                                         # Werteseparator fuer .csv
  row.names = F)                                     # keine Zeilennamen
```

Ergebnisdatei student.csv



student - Editor

Datei Bearbeiten Format Ansicht Hilfe

```
|"delta1","delta2L"
0.7,1.9
-1.6,0.8
```

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

Programmierübungen und Selbstkontrollfragen

Programmierübungen

1. Dokumentiere alle in dieser Einheit verwendeten R-Befehle in einem R-Skript.
2. Lass dir dein aktuelles Working Directory ausgeben. Ändere es anschließend zu einem anderen Ordner, überprüfe mit einer erneuten Ausgabe, ob die Änderung erfolgreich war. Setze das Working Directory danach wieder zurück und überprüfe erneut, ob die Rücksetzung funktioniert hat.
3. Schreibe ein einfaches Skript, in dem Daten in ein Dataframe eingelesen werden. Füge neue Daten hinzu und speichere das resultierende Dataframe in einer neuen Datei. Verwende dabei Pfadvariablen (d.h., einen ein-elementigen character vector für den Dateipfad).
4. Wiederhole die vorherige Aufgabe, jedoch mit einer geänderten Pfadvariable.
 - Ändere dafür zunächst dein Working Directory in ein anderes Verzeichnis, z.B. in das übergeordnete Verzeichnis (parent directory).
 - Passe anschließend die Pfadvariable im Code an, die du zum Einlesen der Daten verwendest, damit sie zum neuen Working Directory passt.
 - Beachte, dass sich in deiner Ordnerstruktur auf der Festplatte nichts ändert, sondern lediglich das „Working Directory“ und die Pfadvariable angepasst werden.

Selbstkontrollfragen

1. Erläutere den Begriff "Forschungsdaten".
2. Erläutere den Begriff "Metadaten".
3. Erläutere das FAIR Datenideal.
4. Diskutiere Unterschiede und Gemeinsamkeiten von binären und textuellen Dateien.
5. Nenne und erläutere zwei textuelle Dateiformate.
6. Erläutere den Unterschied zwischen dem Wide und Long Format von Tabellen.
7. Erläutere den Unterschied zwischen absoluten und relativen Dateipfaden.
8. Erläutere den Begriff des "Working Directories" in R.
9. Nenne eine R Funktion zum Einlesen von .csv Dateien.
10. Nenne eine R Funktion zum Schreiben von .csv Dateien.