
# Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2025/26

Belinda Fleischmann und Dirk Ostwald

| | Gruppe 1/2 | Gruppe 3 | Format | Thema |
|---|---|---|---|---|
| 1 | Mi, 15.10. | Do, 16.10. | Seminar | (1) Grundbegriffe der Informatik |
| 2 | Mi, 22.10. | Do, 23.10. | Seminar | (2) Arithmetik und Variablen |
| 3 | Mi, 29.10. | Do, 30.10. | Übung | (2) Arithmetik und Variablen |
| 4 | Mi, 05.11. | Do, 06.11. | Seminar | (3) Vektoren und Matrizen |
| 5 | Mi, 12.11. | Do, 13.11. | Übung | (3) Vektoren und Matrizen |
| 6 | Mi, 19.11. | Do, 20.11. | Seminar | (4) Listen und Dataframes |
| 7 | Mi, 26.11. | Do, 27.11. | Übung | (4) Listen und Dataframes |
| **8** | **Mi, 03.12.** | **Do, 04.12.** | **Seminar** | **(5) Datenmanagement** |
| 9 | Mi, 10.12. | Do, 11.12. | Übung | (5) Datenmanagement |
| | Mo, 15.12 | | Abgabe | *Individualleistung (1/2)* |
| 10 | Mi, 17.12. | Do, 18.12. | Seminar | (6) Strukturiertes Progr.: Kontrollfluss, Debugging |
| 11 | Mi, 07.01. | Do, 08.01. | Seminar | (7) Häufigkeitsverteilungen |
| 12 | Mi, 14.01. | Do, 15.01. | Übung | (7) Häufigkeitsverteilungen |
| 13 | Mi, 21.01. | Do, 22.01. | Seminar | (8) Maße der zentralen Tendenz und Datenvariabilität |
| 14 | Mi, 28.01. | Do, 29.01. | Übung | (8) Maße der zentralen Tendenz und Datenvariabilität |
| | Mo, 02.02 | | Abgabe | *Individualleistung (2/2)* |

# (5) Datenmanagement

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

**Daten**

FAIR Prinzipien

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

# Beispiele für Daten

- 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 (2017)

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)

Daten

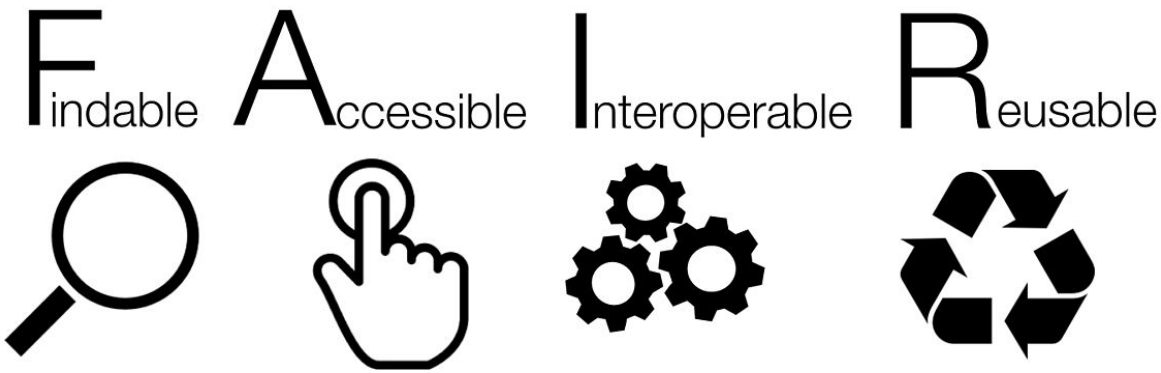
**FAIR Prinzipien**

Datenformate

Verzeichnismanagement

Datenimport und Datenexport

# Das FAIR Datenideal




für Menschen und Maschinen

# Ursprung und Dokumentation

Das FAIR Ideal zum Forschungsdatenmanagement hat seinen Ursprung in Diskussionen im Rahmen der FORCE11 Konferenzserie „Jointly designing a data fairport“ und durch Wilkinson et al. (2016) akademisch kommuniziert.

Die Fair Prinzipien besagt im Kern, dass Forschungsdaten Findable, Accessible, Interoperable, und Reusable für Menschen und Maschinen aufbereitet und vorgehalten werden sollen. Speziell formuliert das FAIR Datenideal folgende Ansprüche an das Management von Forschungs(meta)daten.

# Metadaten

Metadaten repräsentieren Information über andere Daten. Man unterscheidet dabei beispielsweise deskriptive, strukturelle und administrative Metadaten (Riley 2017; Ulrich et al. 2022)

**Deskriptive Metadaten** dienen dem Auffinden und der Identifikation einer Datenquelle. Beispiele sind etwa Titel der einer wissenschaftlichen Publikation oder ihr digital object identifier (DOI, Rosenblatt 1997). DOIs sind alphanumerische Kennungen, die digitale Objekte eindeutig identifizieren und als persistente Links zu Objekten im Internet funktionieren.

**Strukturelle Metadaten** sind Metadaten über Datencontainer und repräsentieren den strukturellen Aufbau einer Datenquelle. Beispiele sind die Ordnung der Seiten eines Buches oder die for-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.

# Findability (Auffindbarkeit)

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.

# Accessibility (Zugänglichkeit)

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.

# Interoperability (Interoperabilität)

I1. (Meta)Daten nutzen eine formale, zugängliche, gemeinsam genutzte und breit anwendbare Sprache zur Wissensrepräsentation.

I2. (Meta)Daten nutzen Vokabularien, die den FAIR-Prinzipien folgen.

I3. (Meta)Daten enthalten qualifizierte Referenzen auf andere (Meta)Daten.

# Reusability (Wiederverwendbarkeit)

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.

# FAIR Prinzipien - Fazit

- Der Anspruch an ein FAIRes Forschungsdatenmanagement ist mittlerweile recht verbreitet.
- Die Umsetzung der FAIR-Prinzipien kann mehr oder weniger bürokratisch ausfallen.
- Die FAIR Prinzipien sind ein anzustrebendes Datenmanagementideal. Nicht alle Forschungsdaten sind FAIR verfügbar.
- Der Umgang mit digitalen Forschungsdaten ist oft noch sehr unstrukturiert.
- Universitäten begreifen das digitale Datenmanagement nur sehr langsam als Kernaufgabe.
- Die NFDI Initiative versucht, das deutsche digitale Forschungsdatenmanagement zu verbessern.
- NFDI ist Projekt-basiert und somit nicht in einem nachhaltigen Rahmen.
- Nicht alle Wissenschaftler:innen möchten ihre Daten systematisch aufbereiten und der Öffentlichkeit zur Verfügung stellen.
- Kulturwandel zu mehr wissenschaftlicher Transparenz und offene öffentlich-finanziert Forschung (Open Research, vgl. z.B. Toelch and Ostwald 2018; Miedema 2022; Bertram et al. 2023) bleibt ein anzustrebenes Ideal.

Daten

FAIR Prinzipien

**Datenformate**

Verzeichnismanagement

Datenimport und Datenexport

## 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 propietären (urheberrechtlich geschützten) Dateiformaten.

## Binäre Dateiformate

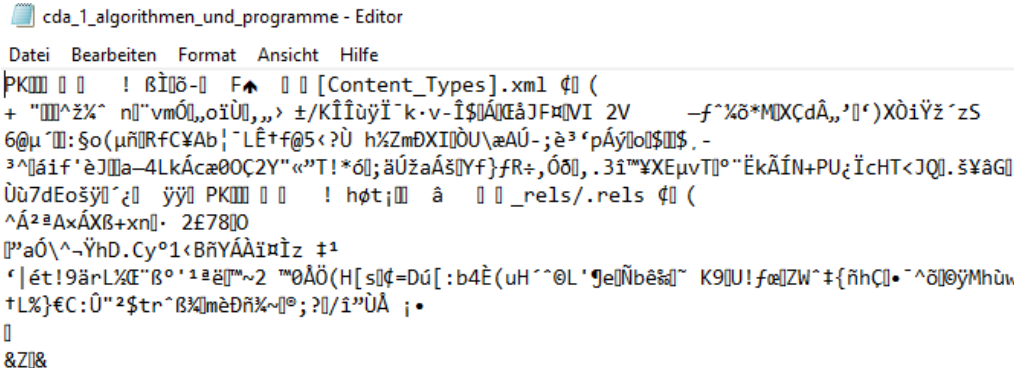
- Einlesen, Inspektion, und Manipulation ist nur mit spezieller Software möglich.
- .pdf, .xlsx, .jpg, .mp4 sind binäre Dateiformate.
- Binäre Dateiformate sind oft propietär.
- Binäre Dateiformate wurden früher aufgrund ihrer kleineren Größe bevorzugt eingesetzt.

## Textuelle Dateiformate

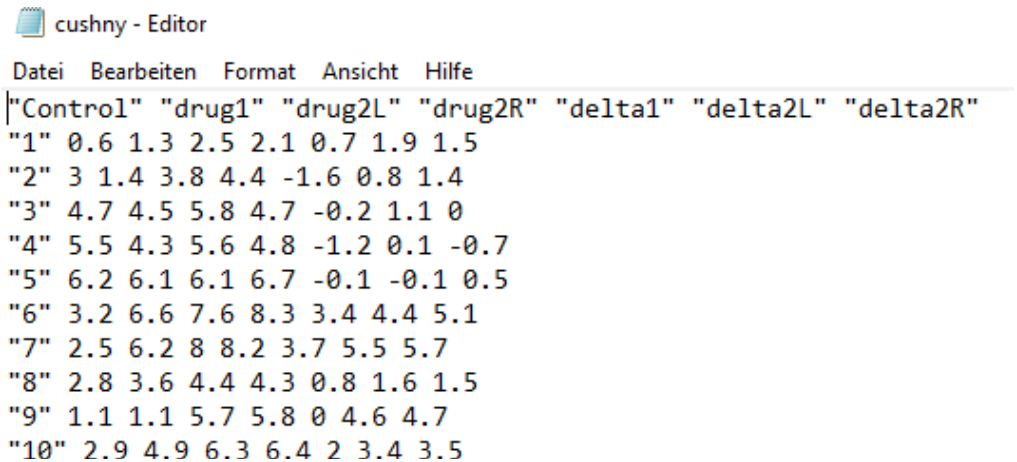
- Einlesen, Inspektion, und Manipulation ist mit einfachen allgemeinen Editoren möglich.
- .txt, .csv., .tsv, .json sind textuelle Dateiformate.
- Textuelle Dateiformate sind generell offene Dateiformate.

# Beispiele

## Binäres Dateiformat




## Textuelles Dateiformat



- CSV = Comma-separated values (.csv), TSV = Tab-separated values (.tsv)
- Zentrale Dateiformate zur Speicherung einfacher, tabellarisch strukturierter Daten
- Repräsentation zeilenweise miteinander verknüpfter Datensätze
  - Trennung von Datenfeldern (Spalten) durch Komma (.csv) oder Tab (.tsv)
  - Trennung von Datensätzen (Zeilen) durch Zeilenumbruch
- Erste Zeile (Kopfdatensatz/Header) enthält typischerweise die Definition der Spaltennamen
- Typische Struktur:
  - Zeilen repräsentieren experimentelle Einheiten (z.B. Versuchspersonen)
  - Spalten repräsentieren Variablen mit jeweils gemessenen Werten

Beispiel

.csv Dateiinhalt

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

Tabellenrepräsentation

| Einheit | Variable 1 | Variable 2 |
|---|---|---|
| 1 | 10.1 | 67.5 |
| 2 | 12.9 | 51.2 |
| 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 |
| 1 | Variable 2 | 67.5 |
| 2 | Variable 1 | 12.9 |
| 2 | Variable 2 | 51.2 |
| 3 | Variable 1 | 20.4 |
| 3 | 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 Listen 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

# 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 weiterer Funktionalitäten

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

```
[1] "Rote Blume"  "Gelbe Blume"
```

```
paste(c("Rote", "Gelbe"), "Blume",  sep = "-")        # Separatorspezifkation
```

```
[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 Adressen 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                      Pfad der auf einem Verzeichnisnamen endet
H:\Lehre\Daten\cushny.csv           Pfad der auf einem Dateinamen endet

Unix-like OS:
/home/user/Lehre/Daten              Pfad der auf einem Verzeichnisnamen endet
/home/user/Lehre/Daten/cushny.csv   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 als Working Directory für jede neue R-Terminal-Session verwendet.
- Dies entspricht dem im Explorer sichtbaren übergeordneten Verzeichnis.

`getwd()` gibt das Working Directory an.

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

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26/5_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* gegeben die Adresse in der Gesamtverzeichnisstruktur der Festplatte an.
- Absolute Dateipfade sind weniger anfällig für Dateiverwechselungen.
- **Generell wird die Verwendung adaptiv generierter absoluter Pfade 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 von Dateipfaden

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

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

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

```
daten_pfad             <- gsub("\\\\", "/", windows_pfad)
print(daten_pfad)
```

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

`sys.frame(1)$ofile` ermittelt den absoluten Pfad des aktuellen Skripts (funktioniert nicht in der Konsole).

```
skriptpfad             <- sys.frame(1)$ofile        # Pfad des Skripts
skriptordner           <- dirname(skriptpfad)       # Ordner, in dem sich das Skript befindet
print(skriptordner)
```

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26/5_Datenmanagement"
```

Daten

FAIR Prinzipien

Datenformate

Verzeichnismanagement

**Datenimport und Datenexport**

## `read.csv()`

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

```
work_dir      <- getwd()                                  # Pfad zum Working directory
kursordner    <- dirname(work_dir)                        # Kursordner
datenordner   <- file.path(kursordner, "Daten")           # Datenverzeichnispfad
dateiname     <- "cushny.csv"                             # (base) filename
dateipfad     <- file.path(datenordner, dateiname)        # filepath
D             <- read.csv(dateipfad)                      # 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

Im Folgenden werden die einzelnen Schritte der oben verwendeten Dateipfadkonstruktion ausgegeben

```
work_dir      <- getwd()                                    # Pfad zum Working directory
print(work_dir)
```

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26/5_Datenmanagement"
```

```
kursordner    <- dirname(work_dir)                          # Kursordner
print(kursordner)
```

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26"
```

```
datenordner   <- file.path(kursordner, "Daten")             # Datenverzeichnispfad
print(datenordner)
```

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26/Daten"
```

```
dateiname     <- "cushny.csv"                               # (base) filename
dateipfad     <- file.path(datenordner, dateiname)          # filepath
print(dateipfad)
```

```
[1] "/home/belindame_f/OVGU/2026_1_WiSe_PDS/progr-und-deskr-stat-26/Daten/cushny.csv"
```

# Datenimport

`read.csv()` ist ein Wrapper für `read.table()` mit Voreinstellungen `sep = ","` und `header = TRUE`.

```r
dateipfad <- file.path(datenordner, "cushny.csv")           # Pfad zur .csv Datei

# Beide Varianten liefern dasselbe Ergebnis
D         <- read.csv(dateipfad)                            # Einlesen der .csv Datei
D         <- read.table(dateipfad, sep = ",", header = TRUE) # Einlesen der .csv 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

Ein paar weitere Beispiele für Spezifikationen sind

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

```
D <- read.csv(dateipfad, header = TRUE, sep = ",", nrow = 2)  # Nur 2 Zeilen einlesen
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.csv(dateipfad, header = TRUE, sep = ",", skip = 7)  # Ab Zeile 7 einlesen
print(D)
```

```
  X2.5 X6.2  X8 X8.2 X3.7 X5.5 X5.7
1  2.8  3.6 4.4  4.3  0.8  1.6  1.5
2  1.1  1.1 5.7  5.8  0.0  4.6  4.7
3  2.9  4.9 6.3  6.4  2.0  3.4  3.5
```

# Weitere Möglichkeiten des Datenimports

## Textdateien (.csv, .tsv, .txt, .json)

- `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 (.xlsx, .sav, .mat)

- `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.

## Für Datenbanken:

- SQL Datenbanken können mithilfe der Pakete `DBI` und `RSQLite` abgefragt werden.

# 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 (Damian)                       Project Talent data set from Marion Spengler and Rodica Damian
Spengler.stat (Damian)                  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 (ability)                  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
all.income (income)                     US family income from US census 2008
bfi                                     25 Personality items representing 5 factors
```

Alle Datensätze aller installierten Pakete 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
```

# Datenexport

## `write.table()`

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

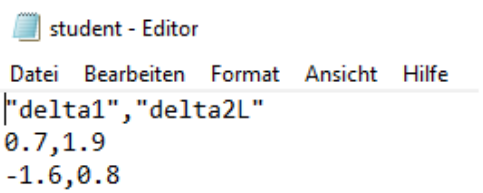
## Spezifikationen bei der Anwendung

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

```
input_dateipfad  <- file.path(datenordner, "cushny.csv")    # Pfad zur Datei, die eingelesen werden soll
output_dateipfad <- file.path(datenordner, "student.csv")   # Pfad der Datei, die geschrieben werden soll

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

## Ergebnisdatei student.csv



# Referenzen


Bertram, Michael G., Josefin Sundin, Dominique G. Roche, Alfredo Sánchez-Tójar, Eli S. J. Thoré, and Tomas Brodin. 2023. "Open Science." *Current Biology* 33 (15): R792–97. https://doi.org/10.1016/j.cub.2023.05.036.

Miedema, Frank. 2022. *Open Science: The Very Idea*. Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-94-024-2115-6.

Rat für Informationsinfrastrukturen. 2017. "Datenschutz Und Foschungsdaten - Aktuelle Empfehlungen."

Riley, Jenn. 2017. *Understanding Metadata: What Is Metadata, and What Is It for*. NISO Primer Series. Baltimore, MD: National Information Standards Organization.

Rosenblatt, Bill. 1997. "The Digital Object Identifier: Solving the Dilemma of Copyright Protection Online." *Journal of Electronic Publishing* 3 (2). https://doi.org/10.3998/3336451.0003.204.

Toelch, Ulf, and Dirk Ostwald. 2018. "Digital Open Science—Teaching Digital Tools for Reproducible and Transparent Research." *PLOS Biology* 16 (7): e2006022. https://doi.org/10.1371/journal.pbio.2006022.

Ulrich, Hannes, Ann-Kristin Kock-Schoppenhauer, Noemi Deppenwiese, Robert Gött, Jori Kern, Martin Lablans, Raphael W Majeed, et al. 2022. "Understanding the Nature of Metadata: Systematic Review." *Journal of Medical Internet Research* 24 (1): e25440. https://doi.org/10.2196/25440.

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, et al. 2016. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data* 3 (1): 160018. https://doi.org/10.1038/sdata.2016.18.