Daten und Information

Data Science Grundlagen für die Life Sciences

Christian Glahn

Inhaltsverzeichnis

Vorwort 3

Copyright 4

I. Ausgangslage und Vorbereitung 5

1. Einleitung 6
1.1. Organisation dieses Buchs . . . 6
1.1.1. Ziele . . . 6
1.1.2. Aufbau . . . 6
1.2. Farbcodes und andere wichtige Hinweise . . . 8
1.3. Begleitmaterial . . . 8

2. Tastatur und Tastaturkürzel 9
2.1. Mac Tastatur . . . 9
2.2. Windows Tastatur . . . 9
2.3. Modifikatoren und Abkürzungen . . . 11
2.4. Funktionstasten . . . 12
2.5. Kontrollsequenzen . . . 12
2.6. Sonderzeichen . . . 13

3. Informationstheorie 14
3.1. Shannon's Informationsproblem . . . 14
3.1.1. Das Shannon Limit . . . 17
3.2. Informationstheorie in den Datenwissenschaften . . . 17
3.3. Zahlensysteme . . . 17
3.3.1. Die wichtigsten Zahlensysteme . . . 18
3.3.2. Binärzahlen . . . 19
3.3.3. Winkelangaben als irrationales Zahlensystem . . . 21
3.3.4. Prinzip eines Zahlensystems . . . 21
3.4. Wissenschaftliche Schreibweise von Zahlen . . . 22
3.4.1. Serialisierung . . . 23
3.5. Zeichenkodierung . . . 23
3.5.1. Ziffernkodierung . . . 26
3.6. Symbole und Kompression . . . 27
3.7. Merksatz der Informationskodierung . . . 28

II. Datenquellen **30**

4. Dokumentation **31**
4.1. Technische Projektdokumentation . . . 31
4.1.1. Entscheidungen und Massnahmen . . . 32
4.1.2. Projektübersicht . . . 32
4.1.3. Externe Abhängigkeiten . . . 32
4.1.4. Vertraulichkeit, Verwendung und Urheberschaft . . . 33
4.2. Labor- und Arbeitsbericht . . . 34
4.2.1. Materialliste . . . 35
4.2.2. Versuchsaufbau . . . 35
4.2.3. Durchführung und Ablauf . . . 36
4.2.4. Ergebnisse . . . 36
4.2.5. Fehlerdiskussion . . . 37
4.2.6. Besondere Ereignisse . . . 37
4.2.7. Beobachtungen und Notizen . . . 37
4.3. Projektbericht . . . 37
4.3.1. Einleitung und Hintergrund . . . 38
4.3.2. Forschungsfrage oder Arbeitsauftrag . . . 38
4.3.3. Methode . . . 39
4.3.4. Ergebnisse . . . 39
4.3.5. Interpretation . . . 39
4.3.6. Ausblick . . . 40
4.3.7. Zitieren und Quellenverzeichnis . . . 40
4.3.8. Management-Zusammenfassung . . . 41

5. Daten sammeln **42**
5.1. Das Datenschema . . . 42
5.2. Arten von Daten . . . 43
5.2.1. Unstrukturierte Daten . . . 43
5.2.2. Strukturierte Daten . . . 44
5.2.3. Semi-strukturierte Daten . . . 45
5.3. Daten erheben . . . 46
5.3.1. Manuelle Datenerhebung . . . 46
5.3.2. Technisch-unterstützte Datenerhebung . . . 47
5.3.3. Automatische Datenerhebung . . . 49

6. Daten organisieren **51**
6.1. Daten strukturieren . . . 51
6.1.1. Daten als Tabellen . . . 51
6.1.2. Begriffe . . . 52
6.1.3. Daten normalisieren . . . 53
6.2. Daten ablegen . . . 55
6.2.1. Dateien und Verzeichnisse . . . 55
6.2.2. Daten-Repositories . . . 56
6.2.3. Datenbanken . . . 56

6.2.4. Datenverlust vermeiden . . . 57
6.3. Datenmanipulation . . . 58

7. Versionierung mit Git und GitHub 61
7.1. `git` installieren . . . 62
7.1.1. `git` unter MacOS installieren . . . 62
7.1.2. `git` unter Windows installieren . . . 62
7.1.3. Grafische Oberflächen für `git` . . . 62
7.1.4. `git`-Hosting-Plattformen . . . 63
7.2. `git`-Konzepte . . . 63
7.2.1. Repositories, Clones und Forks . . . 63
7.2.2. Versionierungsstatus feststellen . . . 64
7.2.3. Versionierung durch Commits . . . 65
7.2.4. Checkout . . . 67
7.2.5. Tags . . . 68
7.2.6. Branching und Merging . . . 69
7.2.7. Fetch, Pull und Push . . . 73
7.3. `git`-Projektmanagement . . . 74
7.3.1. Issues . . . 74
7.3.2. Pull-Requests . . . 75
7.4. Anwendungen und Praxistipps . . . 76
7.4.1. GitHub-Flow . . . 76
7.4.2. Dateien und Verzeichnisse von der Versionierung ausschliessen . . . 77
7.4.3. Repository-Organisation . . . 79

8. Datentypen 81
8.1. Fundamentale Datentypen . . . 81
8.1.1. Undefinierte Werte . . . 81
8.1.2. Zahlen . . . 82
8.1.3. Wahrheitswerte . . . 82
8.1.4. Zeichenketten . . . 83
8.1.5. Fehlerwerte . . . 83
8.2. Klassen von Datentypen . . . 84
8.2.1. Diskrete Daten . . . 84
8.2.2. Kontinuierliche Daten . . . 86
8.3. Datenstrukturen . . . 87
8.3.1. Eindimensionale Datenstrukturen . . . 87
8.3.2. Mehrdimensionale Datentypen . . . 88
8.4. Vektorformen von Matrizen . . . 89

9. Dateiformate 91
9.1. Dateien verwenden . . . 91
9.1.1. Importieren . . . 91
9.1.2. Exportieren . . . 93
9.2. Textdateien . . . 94
9.3. Festkodierung . . . 94

9.4. Zeilenbasierte Textdateien . . . 95
9.5. Mengen und Bäume . . . 95
9.6. Separator-strukturierte Textdateien . . . 96
9.7. Excel-Arbeitsmappen . . . 99
9.8. Markup-Formate . . . 100
9.8.1. XML . . . 101
9.8.2. HTML . . . 103
9.9. JSON . . . 104
9.10. YAML . . . 106
9.11. Dateiformate und Versionierung . . . 108

III. Mathematik der Daten **109**

10. Variablen, Funktionen und Operatoren **110**
10.1. Variablen . . . 110
10.2. Funktionen . . . 111
10.2.1. Spezielle Funktionen . . . 113
10.3. Operatoren . . . 114
10.3.1. Precedence - Prioriät von Operatoren . . . 114
10.3.2. Substitution . . . 115
10.3.3. Besondere Operatoren . . . 116
10.4. Das neutrale Element . . . 119
10.4.1. Überprüfung . . . 120
10.4.2. Redundanz . . . 121
10.5. Funktionsketten . . . 121
10.5.1. Anwendung von Funktionsketten . . . 125
10.6. Funktionen als Werte . . . 126
10.6.1. Callback-Funktionen . . . 126
10.6.2. Closure-Funktionen . . . 127
10.7. Besonderheiten von Funktionsketten . . . 128
10.7.1. Die Identitätsfunktion . . . 128
10.7.2. Verkettung mit Umkehrfunktionen . . . 129

11. Zeichenketten **131**
11.1. Nicht-druckbare Zeichen . . . 131
11.2. Die leere Zeichenkette . . . 132
11.3. Symbolvektoren . . . 132
11.4. Texttrennung . . . 132
11.5. Textverkettung . . . 133
11.6. Normalisierung . . . 133
11.7. Gross- und Kleinschreibung . . . 134

12. Boole'sche Operationen **135**
12.1. Mathematische Operationalisierung der Aussagenlogik . . . 136
12.1.1. Belegungstafeln oder Wahrheitstafeln . . . 137

12.1.2. Boole'sche Arithmetik . . . 138
12.2. Boole'sche Algebra . . . 139
12.2.1. Grundregeln der Boole'schen Algebra . . . 139
12.3. Vergleiche . . . 140
12.4. Entscheidungen . . . 142
12.5. Filtern . . . 144
12.6. Sortieren . . . 144
12.6.1. Sortieren für Fortgeschrittene . . . 145

13. Vektoroperationen **146**
13.1. Sequenzen . . . 146
13.1.1. Besondere Sequenzen . . . 148
13.2. Konkatenation . . . 148
13.3. Tranformationen . . . 149
13.3.1. Transformationen mit einem Skalar . . . 150
13.3.2. Transformationen mit einem Vektor . . . 151
13.4. Aggregationen . . . 152
13.5. Zählen . . . 153
13.5.1. Zählen durch Summieren . . . 154
13.5.2. Zählen durch Filtern . . . 155
13.5.3. Zählen durch Nummerieren . . . 155

14. Matrix-Operationen **156**
14.1. Grundbegriffe . . . 156
14.2. Matrixaddition . . . 158
14.3. Vektoraddition . . . 158
14.4. Skalarmultiplikation (Punktprodukt) . . . 159
14.5. Matrixmultiplikation/ Kreuzprodukt . . . 159
14.5.1. Kreuzprodukt für Vektoren . . . 160
14.5.2. Inverse Matrix . . . 161
14.5.3. Anwendungen des Kreuzprodukts . . . 161
14.6. Das äussere Matrixprodukt . . . 162
14.7. Co-Occurence-Matrizen erzeugen . . . 166

15. Indizieren und Gruppieren **168**
15.1. Indizieren . . . 168
15.1.1. Existierende Indexvektoren. . . . 168
15.1.2. Fehlende Indexvektoren . . . 169
15.1.3. Hashing zur Identifikation . . . 170
15.1.4. Hashing zum Gruppieren . . . 170
15.1.5. Hashing für Querverweise . . . 171
15.2. Datensätze randomisieren . . . 171

16. Daten kombinieren und kodieren **174**
16.1. Kombinieren . . . 174
16.1.1. Kombinationsarten . . . 174

16.2. Kodieren . . . 175
16.2.1. Kodierungstabellen . . . 175

17. Daten umformen **177**
17.1. Transponieren . . . 178
17.2. Hierarchisieren . . . 179

IV. Deskriptive Datenanalyse **180**

18. Daten beschreiben **181**
18.1. Universelle Kennzahlen . . . 181
18.2. Umfang . . . 184
18.3. Skalenniveaus und Lagemasse . . . 185
18.3.1. Lagemasse für nominale Skalenniveaus . . . 186
18.3.2. Lagemasse für metrische Skalenniveaus . . . 188
18.3.3. Lagemasse für ordinale Skalenniveaus . . . 189

19. Daten visualisieren **191**
19.1. Aufbau eines Diagramms . . . 191
19.1.1. Haupt- und Nebendimensionen . . . 192
19.1.2. Darstellungsbereich . . . 193
19.1.3. Achsen . . . 193
19.1.4. Titel . . . 194
19.1.5. Legende . . . 194
19.2. Plots mit einer Variablen . . . 194
19.2.1. Histogramme . . . 194
19.2.2. Boxplot . . . 196
19.2.3. Dichtediagramme . . . 197
19.2.4. Violindiagramm . . . 198
19.3. Plots mit zwei Variablen . . . 198
19.3.1. Funktionsdiagramme . . . 198
19.3.2. Beziehungen . . . 200
19.3.3. Ausgleichsgeraden . . . 203
19.3.4. Zwei unterschiedliche Skalenniveaus kombinieren . . . 204
19.4. Plots mit mehr als zwei Variablen . . . 204
19.4.1. Farbliche Kodierung . . . 206
19.4.2. Grössenkodierung . . . 207
19.4.3. Form-Kodierung . . . 209
19.4.4. Teildiagramme . . . 209

Referenzen **211**

Vorwort

Work in Progress

Copyright

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (CC-BY-NC-SA). Details zur Nutzungsbedingungen und dem Copyright finden sich unter createivecommons.org.

2023, Christian Glahn, Zurich, Switzerland

Die PDF Version liegt hier zum Download.

Teil I.

Ausgangslage und Vorbereitung

1. Einleitung

> **Warnung**
>
> Work in progress

1.1. Organisation dieses Buchs

1.1.1. Ziele

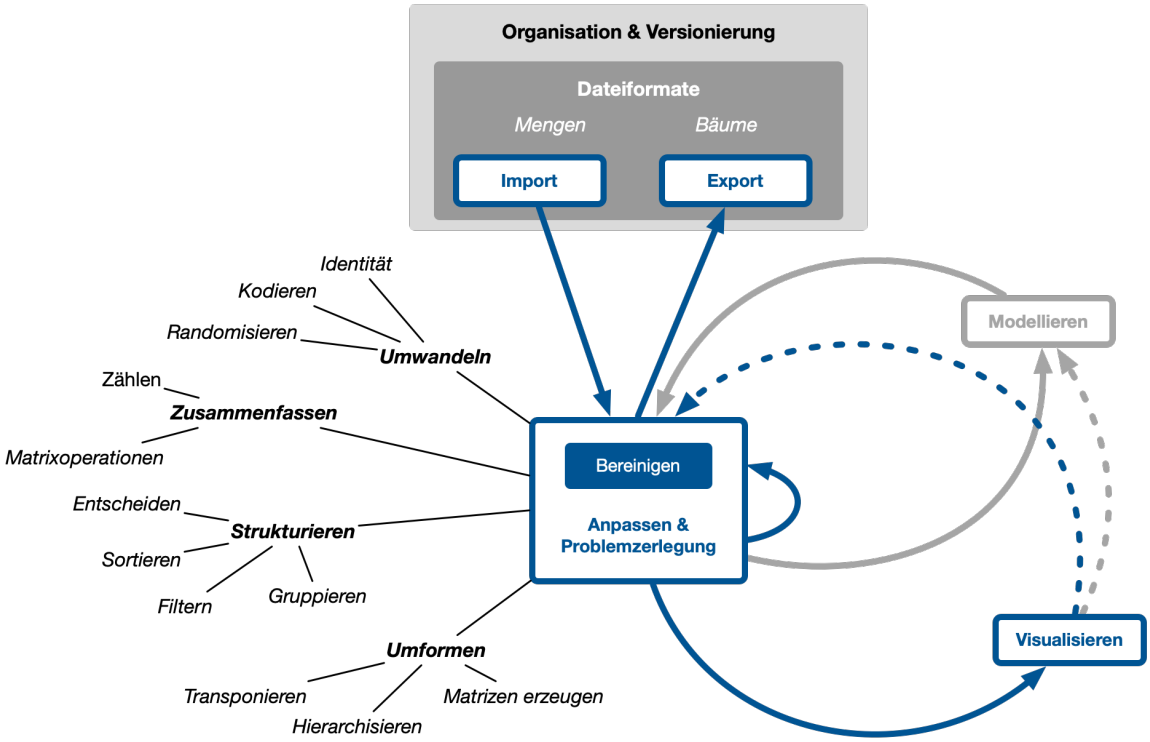


Abbildung 1.1.: Inhaltliche Ziele

1.1.2. Aufbau

Dieses Buch ist in vier Teile gegliedert:

- Teil I: Ausgangslage und Voraussetzungen
- Teil II: Datenquellen
- Teil III: Mathematik der Daten
- Teil IV: Deskriptive Datenanalyse

Das Buch ist so aufgebaut, dass die einzelnen Kapitel aufeinander aufbauen und später als Referenz verwendet werden können. Das Buch konzentriert sich auf die zentralen Konzepte und Praktiken bei der Arbeit mit Daten. Es werden keine spezifischen Werkzeuge oder Programmiersprachen vorausgesetzt oder im Detail behandelt.

Dieses Buch kann auf zwei Arten gelesen und verwendet werden:

- Als Grobeinführung in die digitale Arbeit mit Daten. Diese Einführung richtet sich an Studierende und Praktiker, die sich einen Überblick über die Thematik und Konzepte verschaffen wollen.
- Als vertiefte Einführung in die digitale Arbeit mit Daten. Diese Einführung richtet sich an Studierende und Praktiker, die mit einem Data-Science-Studium beginnen oder sich auf erste Data-Science-Projekte vorbereiten wollen.

Kapitel	Grobeinführung	Vertiefte Einführung
Einleitung	x	x
Tastatur und Tastaturkürzel	x	x
Informationstheorie	bis und mit Abschnitt 3.2	x
Dokumentation		x
Daten sammeln	x	x
Daten organisieren	x	x
Versionierung		x
Datentypen	x	x
Dateiformate	bis und mit Abschnitt 9.6	x
Variablen, Funktionen und Operatoren	bis und mit Abschnitt 10.5	x
Zeichenketten		x
Boole'sche Operationen	bis und mit Abschnitt 12.1.2	x
Vektor-Operationen	x	x
Matrix-Operationen	bis und mit Beispiel 14.13	x
Daten kodieren	x	x
Indizieren und Gruppieren	x	x
Daten umformen	x	x
Daten beschreiben	x	x
Daten visualisieren	bis und mit Abschnitt 19.3.4	x

1.2. Farbcodes und andere wichtige Hinweise

Dieses Buch verwendet Hervorhebungen und Farbcodes, um die verschiedenen Arten von Informationen zu unterscheiden.

Definition 1.1. Eine Definition eines Konzepts oder Begriffs.

Einrückungen kennzeichnen Anmerkungen und Exkurse.

Beispiel 1.1 (Beispiel eines Beispiels)**.** Ein ausführliches Beispiel, das die Anwendung eines Konzepts zeigt.

Hinweis

Beschreiben allgemeine Hinweise zur Verwendung eines Konzepts.

Merke

Kennzeichnet wichtige Konzepte als Merksätze.

Praxis

Kennzeichnet die praktische Anwendung eines Konzepts.

Warnung

Weist auf häufige Missverständnisse und Fehler hin.

Wichtig

Weist auf zentrale Konzepte und Informationen hin.

1.3. Begleitmaterial

Zu diesem Buch gibt es die folgenden Begleitmaterialien mit Details und Anwendungsbeispielen für einzelne Programmiersprachen. Diese Begleitmaterialien sind so aufgebaut, dass sie direkt an die Inhalte dieses Buchs anknüpfen und die Abschnitte für die eine gewünschte Programmiersprache ergänzen und konkretisieren.

- Daten und Information - Einführung in die Datenwissenschaft mit Excel
- Daten und Information - Einführung in die Datenwissenschaft mit R

2. Tastatur und Tastaturkürzel

Es gibt verschiedene Tastaturlayouts bei denen wichtige Symbole unterschiedlichen Tasten zugeordnet sind. In diesem Abschnitt werden alle Tastenkombinationen mit Bezug auf die Schweizer Tastatur beschrieben. Auf diesen Tastaturen finden wir das Plus-Symbol (`+`) über der `1s`. Einige der hier beschriebenen Funktionen sind bei anderen Tastaturlayouts über eine andere Tastenkombination oder auch direkt erreichbar.

Die wichtigsten Standardtasten sind:

- `Tabulator` (`tabstopp` oder `tab`)
- `Escape` (`Esc`)
- `Backspace` (gelegentlich `Rückwärtslöschen`)
- `Löschen` (`delete`)
- `Eingabe` (`return` oder `enter`)
- `Hochstellen` (`shift`)
- `CAPS-Lock` (dauerhaft Hochstellen)
- Steuerung: `Strg` oder `Ctrl` (Windows) / `control` (Mac)
- `Alt` (Windows) / `option` (Mac)
- `AltGr` (Windows)
- Funktion: `Fn` (meistens auf Laptops)
- Kommando: `command` (Mac) / `Windows` (Windows)
- Cursor-Tasten

2.1. Mac Tastatur

> **Hinweis**
>
> Einige **MacBookPro** Modelle verfügen über einen sog. *Touchbar* an der Position, an welcher sich normalerweise die Funktionstasten befinden. Das Drücken der `Fn`-Taste wechselt die Darstellung aus dem jeweiligen App-Kontext zu virtuellen Funktionstasten. Bei Modellen mit Touchbar aus den Reihen vor 2020 fehlt eine `Esc`-Taste. Auch diese Taste können Sie mit der `Fn`-Taste im Touchbar aktivieren.

2.2. Windows Tastatur

Die `Windows`-Taste: Diese Taste befindet sich links neben der `Leerzeichen`-Taste. Im Gegensatz zur Kommandotaste unter Mac hat die `Windows`-Taste keine zentrale Funktion

Abbildung 2.1.: Mac-Tastatur beschriftet

Abbildung 2.2.: Windows-Tastatur beschriftet

im System. Viele Tastaturkürzel, die beim Mac über die `Kommando`-Taste ausgelöst werden, werden unter Windows über die Steuerungstaste bereitgestellt. Die `Windows`-Taste wird unter Windows fast ausschliesslich für Systemfunktionen verwendet.

Die `AltGr`-Taste: Auf Ihren Tastaturen sehen Sie auf manchen Tasten zusätzliche Symbole. Diese Symbole können Sie unter Windows nur über die `AltGr`-Taste erzeugen. Das ist unter Windows die einzige Funktion der `AltGr`-Taste. Der Grund dafür ist, dass europäische Computertastaturen einen grösseren Zeichenumfang als eine US-Tastatur (die sog. ANSI Tastatur) haben. Weil alle Programmiersprachen die Symbole der US-Tastatur verwenden, müssen die europäischen Tastaturen diese Symbole zusätzlich bereitstellen. Unter MacOS sind diese Symbole über beide `Alt`-Tasten erreichbar.

2.3. Modifikatoren und Abkürzungen

Gelegentlich benötigen wir Tastenkombinationen, bei denen wir mehrere Tasten gleichzeitig drücken. Dabei kommen die Tasten `Hochstellen`, `Alt` und `Steuerung` entweder einzeln oder kombiniert als *Modifikatoren* zum Einsatz.

> **Hinweis**
>
> Modifikatoren-Tasten haben keine eigene Bedeutung, sondern **modifizieren** die Bedeutung einer anderen Taste.

Die folgenden Tasten sind immer Modifikatoren:

- `Hochstellen`
- `CAPS-Lock`
- `Alt` / `AltGr`
- `Steuerung`
- `Fn`
- `Kommando` (Mac) / `Win` (Windows)

Um nicht immer die Taste in ganzer Länge anzugeben, werden diese Modifikatoren abgekürzt. Die folgenden Buchstaben stehen für die folgenden Modifikatortasten.

Taste	Symbol
Hochstellen	`S`
Alt (Win) / option (Mac)	`M`
Steuerung	`C`

Alle anderen Tasten werden mit ihrem Namen angegeben.

Um eine Tastaturkombination anzuzeigen, werden die zu drückenden Tasten mit `+` verknüpft. Z.B. `M + C + q` bedeutet, dass Sie die Tasten `Alt`, `Steuerung` und `Q` **gleichzeitig** drücken müssen.

> **Hinweis**
>
> Tasten die nacheinander gedrückt werden sollen, werden durch Leerzeichen getrennt. Z.B. `C + q a` bedeutet, dass Sie zuerst die Tasten `Steuerung` und `Q` **gleichzeitig** tippen und **danach** die Taste `A` drücken müssen.

2.4. Funktionstasten

Als Funktionstasten werden normalerweise die Tasten rechts neben der `Esc`-Taste bezeichnet. Diese Tasten sind oft mit `F1`-`F12` beschriftet. Viele Anwendungen haben diese Tasten mit speziellen Funktionen belegt. Das gilt besonders für Excel.

Auf modernen Laptops und vielen Desktoptastaturen sind diese Tasten standardmässig mit Gerätefunktionen wie z.B. Bildschirmhelligkeit oder Lautstärke belegt. Damit wir die eigentlichen Funktionstasten erreichen können, müssen wir deshalb mit der `Fn`-Taste die Standardfunktion auf die Anwendungsfunktion zurückstellen. Wenn in einer Anleitung steht, dass eine Funktion über eine Funktionstaste erreicht werden kann, dann ist implizit gemeint, dass gleichzeitig die `Fn`-Taste gedrückt werden muss.

2.5. Kontrollsequenzen

Kontrollsequenzen sind Tastaturkürzel, mit denen wir häufig verwendete Funktionen direkt über die Tastatur aktivieren können. Excel hat für fast jede Funktion im Menüband eine Kontrollsequenz festgelegt. Wir können also Excel auch ohne Maus, sondern ausschliesslich mit der Tastatur bedienen.

Die meisten Kontrollsequenzen sind unter Windows und MacOS identisch. Der zentrale Unterschied ist jedoch, dass Windows die Steuerungstaste (`Strg` oder `Ctrl`) verwendet. Unter MacOS muss stattdessen die Kommandotaste (oder `command`, manchmal auch als `Apfel`-Taste bezeichnet) verwendet werden. Die Kommandotaste wird für Macs mit `Cmd` abgekürzt.

Die wichtigsten systemweiten Kontrollsequenzen funktionieren praktisch mit jeder App und nicht nur in Excel:

Funktion	Sequenz (Win)	Sequenz (Mac)
Kopieren	`C + c`	`Cmd + c`
Einfügen	`C + v`	`Cmd + v`
Ausschneiden	`C + x`	`Cmd + x`
Speichern	`C + s`	`Cmd + s`
Rückgängig	`C + z`	`Cmd + z`

Für häufig verwendete Funktionen merken Sie sich die Kontrollsequenz. Dadurch können Sie Ihr Arbeitstempo steigern.

2.6. Sonderzeichen

In der Programmierung werden regelmässig Sonderzeichen verwendet, um bestimmte Funktionen auszulösen. Es hilft die Namen und Tastaturkürzel für diese Symbole zu kennen.

Die am häufigsten auftauchenden Sonderzeichen sind:

Name (englisch)	Symbol	Tasten (Win)	Tasten (Mac)
Apostroph (quote oder tick)	'	'	'
Dach (caret)	^	^	^
Anführungszeichen (double-quote)	"	S + 2	S + 2
Schrägstrich (slash)	/	S + 7	S + 7
Akzent (back-tick)	`	S + ^	S + ^
Kaufmännisches Und (ampersand)	&	S + 6	S + 6
Gatter (hash oder gate)	#	AltGr + 3	M + 3
Eckige Klammer auf	[	AltGr + ü	M + 5
Eckige Klammer zu	]	AltGr + !	M + 6
Geschweifte Klammer auf	{	AltGr + ä	M + 8
Geschweifte Klammer zu	}	AltGr + $	M + 9
Senkrechter Strich (pipe)	|	AltGr + 7	M + 7
Rückstrich (backslash)	\\	AltGr + <	S + M + 7
Tilde	~	AltGr + ^	M + n
Unterstrich (low dash)	_	S + -	S + -

> **Hinweis**
>
> Merken Sie sich die Bezeichnungen und Symbole, denn Sie werden sie regelmässig verwenden.

3. Informationstheorie

Die Arbeit mit Daten schliesst immer die Frage ein, ob diese Daten relevante Information verbergen. Dazu müssen zwei Konzepte erfasst werden:

1. Daten
2. Information

Beide Konzepte sind für den Alltag fast überall von Bedeutung und ein intuitives Verständnis der beiden Konzepte sollte jedem vertraut sein. Beim intuitiven Umgang mit Daten und Information fällt auf, dass die beiden Begriffe häufig synonym gebraucht werden. Hier stellt sich ein erstes Problem:

> **Problem**
>
> Wenn die Begriffe *Daten* und *Information* im Kern identisch sind, warum werden dann zwei Begriffe verwendet?

Um dieses Problem zu verstehen, werden die beiden Begriffe auf ihren wörtlichen Ursprung zurückgeführt:

- *Daten*: Pluralform von Datum, das sich auf das lateinische Verb *dare* zurückgeführt werden kann. *Dare* bedeutet *gegeben* und das zugehörige Substantiv bedeutet *das Gegebene* oder *die Gabe*.
- *Information*: lässt sich ebenfalls auf das Lateinische zurückführen, wobei dieses Wort unverändert in den deutschen Wortschatz eingegangen ist. Die Bedeutung dieses Begriffs steht für *Auskunft* oder *Benachrichtigung*.

Mit diesen beiden Ursprüngen können die beiden Begriffe für Anwendungen vorläufig unterscheiden werden:

- *Daten* bezeichnen konkrete Werte.
- *Information* bezeichnet die *Bedeutung* der Daten.

3.1. Shannon's Informationsproblem

Claude Shannon befasste sich in den 1940er Jahren mit den Herausforderungen der (damals) modernen Kommunikationstechnologien Telegraphie und Telefon. Diese Technologien übertragen Nachrichten über einen Nachrichten*kanal*. Ein solcher Kanal kann ein Kabel oder auch eine Funkfrequenz sein. Dieser Nachrichtenkanal wird als *Medium* bezeichnet.

Definition 3.1. Als *Kommunikation* wird das Übertragen von Informationen über ein Medium bezeichnet.

Diese analogen Technologien haben das Problem, dass sich Signale über längere Distanzen abschwächen. Dieser Effekt ergibt sich aus dem "Medium", dass für eine Kommunikation verwendet wird. Ein Kabel hat z.B. eine Dämpfung, die mit der Länge des Kabels steigt. Je länger ein Kabel wird, desto grösser wird die Dämpfung. Die Dämpfung hat zur Folge, dass ein Signal leiser wird. Dadurch geht ein Teil des ursprünglichen Signals verloren. Dieser Prozess wird als "Equivocation" bezeichnet.

> **Merke**
>
> Durch *Equivokation* gehen Informationen beim Übertragen verloren.

Ein zweites Problem analoger Kommunikationstechnologien sind äussere Störungen des Mediums. Wird z.B. ein Magnet an ein Kabel gehalten, werden die über das Kabel übertragenen Signale verzerrt. Solche Veränderungen des Signals werden als Rauschen (engl. *Noise*) bezeichnet. Rauschen entsteht auch zufällig mit zunehmender Länge eines Mediums.

> **Merke**
>
> Durch Rauschen wird fehlerhafte Information den Daten hinzugefügt.

Shannon hat vor diesem Hintergrund die folgende Fragestellung untersucht:

> **Problem**
>
> Wie kann eine Nachricht ein Ziel erreichen, wenn die Daten durch Rauschen und Equivocation verändert werden?

Shannon (1948) gliedert diese Problemstellung in Teilprobleme, indem der Kommunikationsprozess in Teilschritte gegliedert wird. Dabei ist die "geschickte" Gliederung von Bedeutung. Shannon hat den Kommunikationsprozess in sieben Komponenten unterteilt, indem er die bekannten Störungen der Nachrichtenübertragungen verbunden hat.

- Eine Informationsquelle, die Information erzeugt
- Kodieren der Information in eine Nachricht für ein Medium
- Das Übertragen der Nachricht über einen Kanal
- Das Empfangen und Dekodieren der Nachricht
- Ein Informationsziel, die Information aufnimmt

Zusätzlich muss das *Rauschen* des Kanals berücksichtigt werden. Später wurde das Modell um die *Equivokation* erweitert (Shannon, 1949). Sowohl das Rauschen als auch die Equivokation müssen eigenständig berücksichtigt werden, weil diese die Kommunikation unkontrolliert beeinflussen. Aus diese Überlegungen ergibt sich das Kommunikationschema in Abbildung 3.1.

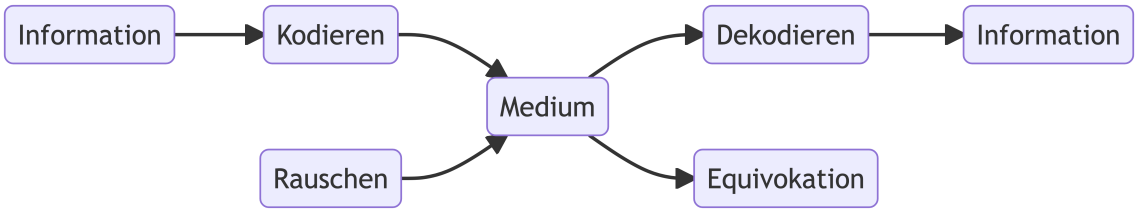


Abbildung 3.1.: Integriertes Kommunikationsmodell (Shannon, 1948; Shannon, 1949)

Shannon's besondere Leistung war, dass er diese Elemente als mathematische Funktionen über Wahrscheinlichkeiten (d.h. Werte zwischen `0` und `1`) formuliert und erkannt hat, dass Kommunikation dem Prinzip der **Entropie** folgt. Dieses Prinzip besagt, dass ein System mit einer niedrigen Entropie (d.h. einer geordneten Struktur) nur zu einer gleichbleibenden oder grösseren Entropie (d.h. zu mehr Unordnung) tendiert. Die Entropie vergrössert sich durch die Fehler bei der Kommunikation.

Daraus ergibt sich für Shannon's Theorie (Shannon, 1949) als direkte Konsequenz, dass Information und Daten über die folgende Funktion verbunden sind:

$$I(D) = P(D) - \epsilon$$

- `D` steht dabei für die empfangenen Daten und
- ϵ ist die Summe der Wahrscheinlichkeiten aller Störungen im Kommunikationsprozess.

$$\epsilon = P(S) + P(E) + P(N) + P(R) = \sum_{a} P(n_a)$$

Mit

- S = Senden/Kodieren
- E = Equivocation
- N = Rauschen (Noise)
- R = Empfangen/Dekodieren (Receiving)

Umgangssprachlich lassen sich diese Terme folgenderweise umschreiben:

Satz 3.1. *Information ergibt sich aus Daten, nachdem alle Fehler und Störungen in den Daten entfernt wurden.*

3.1.1. Das Shannon Limit

Weil $P(D)$ ebenfalls eine Wahrscheinlichkeit ist, ergibt sich, dass Kommunikation nur dann möglich ist, solange die folgende Ungleichung gilt:

$$P(D) \geq \epsilon$$

Entsprechend wird ϵ in der Datenverarbeitung auch als **Shannon Limit** bezeichnet, weil dieser Term die absolute Grenze beschreibt, bis zu der eine fehlerfreie Kommunikation möglich ist.

> **Hinweis**
>
> Diese Ungleichung besagt umgangssprachlich, dass Kommunikation nur möglich ist, solange weniger Fehler und Störungen als Daten übertragen werden.

Shannon hat nachgewiesen, dass jeder Kanal eine Grenze hat, ab der keine Datenübertragung mehr möglich ist.

3.2. Informationstheorie in den Datenwissenschaften

Claude Shannon hat sich mit der technischen Kommunikation beschäftigt. Die Datenwissenschaften befassen sich mit dem Kodieren, dem Organisieren und dem Auswerten von Daten mit dem Ziel der Informationsgewinnung. Diese Schritte sind im Kern ein Kommunikationsprozess: Durch das Messen bestimmter Eigenschaften wird die Information der Realität als Daten erfasst. Anschliessend werden die gemessenen Daten zusammengefasst und strukturiert, damit sie abschliessend ausgewertet werden können. Das Messen entspricht dem Kodiere beim Senden, das Organisieren entspricht dem Daten*medium* und die Auswertung entspricht dem Dekodieren.

3.3. Zahlensysteme

> **Hinweis**
>
> Als **Zahlensystem** wird die Schreibweise für Zahlenwerte bezeichnet.

In der Regel verwenden wir das sog. Dezimalsystem, um Zahlen darzustellen. Das Dezimalsystem hat 10 mögliche Symbole, um Zahlenwerte abzubilden. Diese Symbole sind `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9` und `0`. Damit können wir mit einem Symbol Zahlenwerte zwischen `0` und `9` abbilden.

Gelegentlich lassen sich bestimmte Phänomene nicht gut im Dezimalsystem abbilden. Dadurch lassen sich Werte nur schwer interpretieren. In solchen Fällen hilft der Wechsel in ein anderes Zahlensystem.

> **Merke**
>
> Durch den Wechsel des Zahlensystems ändert sich nur die Darstellung, aber nicht der Wert einer Zahl!

Definition 3.2. Die Zahl, die als Grundlage für ein Zahlensystem dient, wird als **Basis** des Zahlensystems bezeichnet.

Beim in der Schulmathematik üblichen Dezimalsystem ist die Basis `10`.

Definition 3.3. Zahlensysteme kodieren Zahlenwerte zu einer gegebenen Basis.

3.3.1. Die wichtigsten Zahlensysteme

Tabelle 3.1.: Namen und Beispiele von Zahlensystemen

Name	Basis	Symbole
Binärsystem	`2`	`0, 1`
Oktalsystem	`8`	`0, 1, 2, 3, 4, 5, 6, 7`
Dezimalsystem	`10`	`0, 1, 2, 3, 4, 5, 6, 7, 8, 9`
Duodezimalsystem	`12`	`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B`
Hexadezimalsystem	`16`	`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`
Sexagesimalsystem	`60`	-

Das Duodezimalsystem und das Sexagesimalsystem treffen wir im Alltag bei Datums- und Zeitwerten, bei Winkeln sowie in der Musik an. Im Deutschen lässt sich das Duodezimalsystem noch an den Zahlworten elf (`11`) und zwölf (`12`) erkennen.

Das binäre Zahlensystem stellt die Grundlage für digitale Computer dar, weil es nur zwei Werte für die Darstellung von Zahlen benötigt. D.h. alle Werte lassen sich als Vielfache von zweier-Potenzen abbilden. Claude Shannon hat bereits 1938 erkannt, dass diese Darstellung sich direkt die Zustände “ein” und “aus” von Schaltern übersetzen lässt, sodass sich alle Berechnungen mit Hilfe der *Boolschen Algebra* mit einfachen Schaltungen realisieren lassen. Daraus ergibt sich, dass das kleinste Bit der Informationstheorie sich im Binären-Zahlensystem abbilden lässt.

Die Zahlensysteme Oktal und Hexadezimal sind für die Abbildung von Werten in Digitalcomputern von besonderer Bedeutung, weil es sich jeweils um ganzzahlige 2er Potenzen handelt.

Tabelle 3.2.: 2er-Potenzen der wichtigsten Zahlensysteme digitaler Systeme

Name	Basis	2er-Potenz
Binär	`2`	2^1
Oktal	`8`	2^3
Hexadezimal	`16`	$2^4 = 2^{2^2}$

Der Exponent der 2er-Potenz der Basis zeigt an, wie viele Stellen im Binärsystem (Bits) mit dem jeweiligen System abgebildet werden können. Ein Byte bildet per Konvention zwei Stellen im Hexadezimalsystem oder 8 Bit ab.

> **Hinweis**
>
> Hexadezimal-Werte werden recht häufig beim Programmieren verwendet, wie z.B. für das Kodieren von Buchstaben und Satzzeichen. Damit diese Werte leichter von Werten im Dezimalsystem unterschieden werden können, werden Werten im Hexadezimalsystem per Konvention die beiden Symbole `0x` vorangestellt.

Beispiele

Tabelle 3.3.: Darstellung von Zahlenwerten im Dezimal- und Hexadezimalsystem

Dezimal	Hexadezimal
`0`	`0x0`
`1`	`0x1`
`2`	`0x2`
`3`	`0x3`
`4`	`0x4`
`8`	`0x8`
`9`	`0x9`
`10`	`0xA`
`15`	`0xF`
`16`	`0x10`

3.3.2. Binärzahlen

> **Hinweis**
>
> Binärzahlen kodieren Zahlenwerte zur Basis 2.

Daraus folgt, dass für jede Ziffer genau zwei Symbole (Ziffern) zur Verfügung stehen: `0` und `1`.

Wie in anderen Zahlensystemen entspricht eine Stelle im Binärsystem einer Potenz zur gegebenen Basis. Das ist bei Binärwerten 2. Jede Stelle für eine Ziffer kann also einer 2er-Potenz gleichgesetzt werden.

Die Besonderheit des Binärsystems ist, dass alle Werte als Summe von 2er-Potenzen dargestellt werden können. Diese Summe wird als *additive Darstellung* bezeichnet.

Tabelle 3.4.: Binärzahlen und ihre additive Darstellung

Wert	Binärwert	Additive Darstellung	Hexadezimal
0	0000	0	0
1	0001	2^0	1
2	0010	2^1	2
3	0011	$2^1 + 2^0$	3
4	0100	2^2	4
5	0101	$2^2 + 2^0$	5
6	0110	$2^2 + 2^1$	6
7	0111	$2^2 + 2^1 + 2^0$	7
8	1000	2^3	8
9	1001	$2^3 + 2^0$	9
10	1010	$2^3 + 2^1$	A
11	1011	$2^3 + 2^1 + 2^0$	B
12	1100	$2^3 + 2^2$	C
13	1101	$2^3 + 2^2 + 2^0$	D
14	1110	$2^3 + 2^2 + 2^1$	E
15	1111	$2^3 + 2^2 + 2^1 + 2^0$	F

Aus dieser Tabelle kann man ablesen, dass die Ziffer 1 im Binärsystem bedeutet, dass die 2er-Potenz an der entsprechenden Stelle aktiv ist.

Jede Ziffer im Binärsystem kann ausserdem als eigenständiges Symbol einer Nachricht verstanden werden. Weil im Binärsystem nur die beiden Ziffern 0 und 1 möglich sind, müssen beim Dekodieren nur diese Beiden Werte unterschieden werden. Jedes andere Zahlensystem kodiert Zahlen mit mehr als zwei Ziffern.

3.3.2.1. 2er-Potenzen und Speichergrössen

Nach diesem Prinzip werden auch die Kapazitäten von Datenspeichern als 2er-Potenzen beschrieben.

Tabelle 3.5.: Speichergrössen in 2er-Potenzen

Name	Abkürzung	gespeicherte Byte
Byte	B	$2^0 = 1$
Kilobyte	KB	$2^{10} = 1024^1$

Name	Abkürzung	gespeicherte Byte
Megabyte	MB	$2^{20} = 1024^2 = 1048576$
Gigabyte	GB	$2^{30} = 1024^3 = 1073741824$
Terabyte	TB	$2^{40} = 1024^4 = 1099511627776$

> **Warnung**
>
> Die *wissenschaftliche Schreibweise* ist **kein eigenes Zahlensystem**. Sie ist nur eine *Vereinheitlichung* der Schreibweise im Dezimalsystem, um sehr grosse und/oder sehr kleine Zahlen kompakt darstellen zu können.

3.3.3. Winkelangaben als irrationales Zahlensystem

Winkelangaben werden oft als Vielfache von π angegeben. Diese Werte werden auch als *Radiant* anstatt als Grad bezeichnet. Dabei handelt es sich um ein Zahlensystem zur Basis π.

- $\frac{\pi}{6} = 30°$
- $\frac{\pi}{4} = 45°$
- $\frac{\pi}{3} = 60°$
- $\frac{\pi}{2} = 90°$
- $\frac{2\pi}{3} = 120°$
- $\pi = 180°$
- $\frac{3\pi}{2} = 270°$
- $2\pi = 360°$

3.3.4. Prinzip eines Zahlensystems

Die in der Mathematik verwendeten Zahlensysteme sind sog. *additive Zahlensysteme*. Die Schreibweise wird durch Addition und Multiplikation mit der jeweiligen Basis bestimmt.

Das Zählen funktioniert dabei wie folgt:

1. Es wird bei `0` gestartet.
2. Die nächste Ganzzahl wird durch Addition mit `1` erreicht.
3. Es wird das nächste Ziffernsymbol ausgewählt.
4. Gibt es für die jeweilige Basis keine Ziffernsymbole für die Ganzzahl mehr, wird die nächsthöhere Stelle um `1` erhöht.

Beispiele

Tabelle 3.6.: Darstellung von Zahlenwerten in verschiedenen Zahlensystemen

Dezimal	Binär	Oktal	Hexadezimal
0	0	0	0x0
1	1	1	0x1
2	10	2	0x2
3	11	3	0x3
4	100	4	0x4
8	1000	10	0x8
9	1001	11	0x9
10	1010	12	0xA
15	1111	17	0xF
16	10000	100	0x10
255	11111111	377	0xFF
256	100000000	400	0x100

3.4. Wissenschaftliche Schreibweise von Zahlen

> **Hinweis**
>
> Als **wissenschaftliche Notation** wird die Schreibweise von Zahlen mit Hilfe von Potenzen zur Basis 10 bezeichnet.

Bei der wissenschaftlichen Notation wird die erste Ziffer einer Zahl vor ein Komma gesetzt und alle restlichen Ziffern nach dem Komma. Anschliessend werden die restlichen Ziffern gezählt und als 10er-Potenz angegeben.

Bei kleinen Zahlen wird ähnlich vorgegangen: Die erste Ziffer ungleich `0` wird vor ein Komma gesetzt und alle folgenden Stellen danach. Anschliessend werden alle Nullen und die Ziffer vor dem Komma gezählt und als negative 10er-Potenz angegeben.

Neben der ausführlichen wissenschaftlichen Schreibweise wird regelmässig eine abgekürzte Notation verwendet. In dieser Notation wird der Teil $\cdot 10$ durch ein `e` oder `E` ersetzt. Dieses `E` steht für *Exponent*.

Beispiele:

Tabelle 3.7.: Beispiele für die wissenschaftliche Notation verschiedener Zahlenwerte

Normale Notation	Wissenschaftliche Notation	Kurze wissenschaftliche Notation
1	$1 \cdot 10^0$	1e0
10	$1 \cdot 10^1$	1e1
100	$1 \cdot 10^2$	1e2
523140000	$5.2314 \cdot 10^8$	5.2314e8

Normale Notation	Wissenschaftliche Notation	Kurze wissenschaftliche Notation
0.00000000007234	$7.234 \cdot 10^{-}11$	7.234e-11

Die Stärke der wissenschaftlichen Notation ist die Darstellung sehr grosser oder sehr kleiner Zahlen

Mit dieser Schreibweise lassen sich auch schnell Grössenunterschiede zwischen Werten abschätzen: Dazu wird die Differenz der 10er-Potenzen zweier Zahlen gebildet. Dazu wird die kleinere 10er-Potenz von der grösseren subtrahiert. Das Ergebnis ist wieder eine 10er-Potenz.

Beispiel

- Eine Differenz von 1 entspricht einem ungefähr 10-fachen Grössenunterschied.
- Eine Differenz von 5 entspricht einem ungefähr 100000-fachen Grössenunterschied.

3.4.1. Serialisierung

Definition 3.4. Ein Zahlenwert kann bei einer Darstellung zu einer Basis in mehreren Ziffern erfolgen. Diese Zifferndarstellung wird als **Serialisierung** bezeichnet.

Serialisierung bedeutet, dass die Ziffern eines Werts *in einer bestimmten Reihenfolge* dargestellt werden. Jede Ziffer einer solchen Darstellung können wir uns als ein *Symbol* einer Nachricht vorstellen.

Weil ein Zahlenwert in verschiedenen Zahlensystemen dargestellt werden kann, ergibt sich daraus der folgende Merksatz:

> **Merke**
>
> Ein Zahlenwert kann *mehrere* zulässige Serialisierungen haben.

3.5. Zeichenkodierung

Wir schreiben Texte nicht mit Zahlen, sondern mit Buchstaben, Satz- und Steuerzeichen. Im Computer werden Texte als Zahlen abgebildet. Dazu werden die einzelnen Zeichen eines Textes in Zahlenwerte übersetzt. Diese Übersetzung wird als **Zeichenkodierung** bezeichnet und wird per Konvention festgelegt.

Weil sich Buchstaben und andere Zeichen nicht direkt als Zahlen übersetzen lassen, bedarf es eines Tricks. Dazu werden alle zu kodierenden Zeichen in einer Liste aufgeschrieben und anschliessend werden alle Zeichen durchnummeriert. Die Nummer des Zeichens wird als Zahlenwert stellvertretend für das jeweilige Zeichen.

> **Merke**
>
> Mit dem Nummerieren von Symbolen lassen sich beliebige Symbole als Zahlenwerte kodieren.

Bei den meisten Zeichenkodierungen werden die einzelnen Zeichen so aufgereiht, dass die alphabetische und nummerische Reihenfolge in der Regel erhalten bleibt. Zeichenkodierungen sind standardisiert und müssen nicht mehr selbst entwickelt werden. Es gibt allerdings mehrere Standards, die sich in der Kodierung unterscheiden. Deshalb ist es notwendig, die verwendete Zeichenkodierung zu dokumentieren.

Historisch sind vier Kodierungen für die Praxis im deutschsprachigen Raum von Bedeutung.

- ASCII - kodiert das Anglo-amerikanische Alphabet mit Ziffern und Satzzeichen in 7 Bit (Zahlen mit max. 7 Stellen binär), (American National Standards Institute, 1977).
- ANSI - kodiert das Anglo-amerikanische Alphabet mit Ziffern und Satzzeichen in 8 Bit (Zahlen mit max. 8 Stellen binär).
- ISO-8859 - kodiert verschiedene Schriftsysteme in 8 Bit (Zahlen mit max. 8 Stellen binär).
  - ISO-8859-1 (oder ISO Latin 1) - kodiert das westeuropäische Alphabet mit deutschen und französischen Umlauten (ISO/IEC JTC 1/SC 2/WG 3, 1998a).
  - ISO-8859-15 (oder ISO Latin 9) - Kodiert das westeuropäische Alphabet wie ISO-8859-1 aber mit dem Euro Symbol (€) (ISO/IEC JTC 1/SC 2/WG 3, 1998b)
- UTF-8 - kodiert alle gängigen und viele historische Schriftsysteme inkl. Emojis dynamisch mit 8 bis zu 32 Bit (ISO/IEC JTC 1/SC 2, 2020; The Unicode Consortium, 2022, Section 3.9).

Diese Kodierungen sind bis zum Code 01111111 (oder 0x7F) identisch. Die Symbole in diesem Bereich werden deshalb als *ASCII-Codes* bezeichnet.

Tabelle 3.8.: Vollständige ASCII-Kodierungstabelle (American National Standards Institute, 1977, S. 8)

	0	1	2	3	4	5	6	7
0	***NUL***	***DEL***	***SPC***	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	***BS***	CAN	(	8	H	X	h	x
9	***HT***	EM	)	9	I	Y	i	y

	0	1	2	3	4	5	6	7
A	***LF***	SUB	*	:	J	Z	j	z
B	VT	***ESC***	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	***CR***	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	***DEL***

Neben Buchstaben werden auch sog. *nicht-druckbare Zeichen* wie Buchstaben, Satzzeichen und Ziffern kodiert. Dazu gehören u.a. Leerzeichen, Tabulatoren und Zeilenumbrüche. Viele dieser besonderen Buchstaben haben heute keine Bedeutung mehr. In der folgenden Tabelle sind die aktuell verwendeten nicht-druckbaren Zeichen mit * markiert.

Tabelle 3.9.: ASCII-Sonderzeichen (American National Standards Institute, 1977, S. 9)

Zeichen	ASCII-Code	Bedeutung
NUL*	0x00	NULL (Nullzeichen, "Kein Wert", "Ende einer Zeichenkette")
SOH	0x01	Start of Heading
STX	0x02	Start of Text
ETX	0x03	End of Text
ENQ	0x05	Enquiry
EOT/EOF	0x04	End of Transmission/End of File (Dateiende)
ACK	0x06	Acknowledgement
BEL*	0x07	Bell (Klingelzeichen, wird meist ignoriert)
BS	0x08	Backspace (Rückschritt/Rückwärtslöschen)
HT*	0x09	Horizontal Tabulation (Horizontaler Tabulator)
LF*	0x0A	Line Feed (Zeilenvorschub, Zeilenumbruch, Zeilenende)
VT	0x0B	Vertical Tabulation (Vertikaler Tabulator)
FF	0x0C	Form Feed (Seitenvorschub)
CR*	0x0D	Carriage Return (Wagenrücklauf, *nur Windows*)

Zeichen	ASCII-Code	Bedeutung
SO	0x0E	Shift Out
SI	0x0F	Shift In
DLE	0x10	Data Link Escape
DC1	0x11	Device Control 1
DC2	0x12	Device Control 2
DC3	0x13	Device Control 3
DC4	0x14	Device Control 4
NAK	0x15	Negative Acknowledgement
SYN	0x16	Synchronous Idle
ETB	0x17	End of Transmission Block
CAN	0x18	Cancel
EM	0x19	End of Medium
SUB	0x1A	Substitute (Dateiende/Datenende, *nur Windows*)
ESC	0x1B	Escape (Abbruch oder Funktionsänderung)
FS	0x1C	File Separator (Dateiende, veraltet)
GS	0x1D	Group Separator
RS	0x1E	Record Separator
US	0x1F	Unit Separator
SPC*	0x20	Space (Leerzeichen, Leerschlag)
DEL	0x7F	Delete (Vorwärtslöschen)

Die Zeichen für Löschen (BS und DEL) und Funktionsumstellung (ESC) finden sich nicht mehr in Zeichenketten und Dateien. Sie dienen inzwischen nur als Steuerzeichen für die Eingabe mit der Tastatur.

Das Zeichen für das Dateiende (EOF bzw. unter Windows SUB) ist kein kodiertes Zeichen in einer Zeichenkette, sondern wird vom Betriebssystem gesetzt. Dieses Symbol findet sich nicht in einer Datei und sollte nicht eingefügt werden.

Normalerweise werden nicht-druckbare Zeichen nicht in Zeichenketten dargestellt, obwohl sie in der Zeichenkette enthalten sind.

3.5.1. Ziffernkodierung

Arabische Ziffern werden mit den Werten 0x30 (Ziffer 0) bis 0x39 (Ziffer 9) kodiert.

> **Merke**
>
> Ziffern in Zeichenketten sind nicht gleichwertig mit den Ziffern in Zahlen.

Eine Zahl wird als eine Abfolge von Ziffern dargestellt. Wird ein Wert als Zahl dargestellt, dann werden die Ziffern entsprechend der gewählten Basis interpretiert. Werden Ziffern als Zeichenkette kodiert, dann entspricht der *Wert* der Ziffer der entsprechenden Kodierung. D.h.z.B. die Ziffer `"1"` in einer Zeichenkette hat nicht den Wert `1`, sondern den Wert `49` (`0x31`). Folgen mehrere Ziffern aufeinander in einer Zeichenkette, dann werden die kodierten Zahlen aneinandergereiht. Die Ziffern `"123"` entsprechen deshalb nicht dem Wert `123`, sondern dem Wert `3224115` (`0x313233`).

> **Hinweis**
>
> **Excel**, *R* und *Python* konvertieren Ziffern in Zeichenketten *oft* automatisch in die richtigen Zahlenwerte, **solange** keine anderen Zeichen in der jeweiligen Zeichenketten kodiert wurden.
> Nicht alle Programmiersprachen konvertieren Ziffern in Zeichenketten automatisch in Zahlenwerte.

3.6. Symbole und Kompression

Das zentrale Element von Shannon's Informationstheorie sind *Nachrichten*, die aus Symbolen bestehen. Entsprechend trägt jedes Symbol zur Information einer Nachricht (`N`) bei. Shannon versteht unter dem Begriff *Symbol* sowohl Zahlen, Buchstaben, Worte als auch Wortfolgen. Dabei lassen sich Wortfolgen in mehrere Worte und Worte in Buchstaben aufteilen.

> **Hinweis**
>
> Ein Symbol, das nicht in einfachere Symbole unterteilt werden kann, wird als (Informations-) *Bit* bezeichnet.

Sich wiederholende Bits oder Bitfolgen können abgekürzt werden, indem die Bitfolge nur einmal zusammen mit Anzahl der Wiederholungen angegeben wird.

> **Hinweis**
>
> Das Abkürzen einer Bitfolge wird als **Kompression** bezeichnet.

Veranschaulichen wir uns das mit Hilfe der Nachricht `"aaaaaaaaaa"`. Diese Bitfolge kann als $\texttt{[a]}^{10}$ abgekürzt werden.

Der Exponent zeigt uns, wie stark eine Bitfolge komprimiert wurde.

Mit diesem Wissen können wir die Nachricht `"abababababab"` komprimieren. Die Kompression ist in diesem Fall `[ab]`^{5}.

Dieses Spiel können wir weiter treiben: Die Nachricht `"aber aber "` lässt sich als [aber]^2 und die Nachricht `"aber hallo"` lässt sich als `[aber hallo]` ^{1} komprimieren.

Der *Kompressionsgrad* (`K`) ergibt sich aus der Länge der ursprünglichen Nachricht `l(N)` und der Kompression `k`:

$$K = \frac{k}{l(N)}$$

Im Beispiel haben alle Nachrichten die Länge `10`.

Daraus ergeben sich die Kompressionsgrade für unterschiedliche, gleich lange Zeichenketten in Tabelle 3.10.

Tabelle 3.10.: Kompressionsgrade verschiedener Zeichenketten

Nachricht	Kompressionsgrad
`"aaaaaaaaaa"`	1
`"abababababab"`	.5
`"aber aber "`	.2
`"aber hallo"`	.1

Die Kompressionsgrade stehen im umgekehrten Verhältnis zum Informationsgehalt (I_g) einer Nachricht. Es gilt also:

$$I_g = \frac{1}{K} = \frac{l(N)}{k}$$

> **Merke**
>
> Je stärker eine Nachricht komprimiert werden kann, desto weniger Information enthält sie.
> Eine Nachricht mit dem Kompressionsgrad `1` wird als *informationslos* bezeichnet. Sie enthält also *keine* Information.

3.7. Merksatz der Informationskodierung

Ausgehend von der Informationstheorie bestehen Nachrichten aus Symbolen. Symbole können Sätze, Worte, Wortkombinationen, Buchstaben oder Buchstabenfolgen sein. Ein Symbol repräsentiert einen Teil einer Nachricht.

Definition 3.5. Ein Symbol einer Nachricht wird als **Bit** (dt. *Teil*) bezeichnet.

In den vorherigen Abschnitten haben wir wichtige Erkenntnisse abgeleitet:

1. Symbole können aus einfacheren Symbolen zusammengesetzt sein.
2. Zahlensysteme **kodieren** Zahlenwerte zu einer Basis.
3. Die Basis eines Zahlensystems legt fest, wie viele Ziffern für Zahlenwerte zur Verfügung stehen.
4. Die kleinste Basis für ein Zahlensystem ist 2.
5. Beliebige Symbole als Zahlenwerte abgebildetet werden können.
6. Ziffern sind Symbole.

Daraus ergibt sich der folgende Merksatz.

> **Merke**
>
> Das die einfachste Bit-Kodierung für eine Nachricht ist die Unterscheidung zwischen `0` und `1`.

Teil II.

Datenquellen

4. Dokumentation

Die Arbeit mit Daten erfordert eine genaue Dokumentation der Arbeitsschritte und der Arbeitsorganisation. Diese Dokumentation umfasst in der Regel die *technische Projektdokumentation*, *Labor- und Arbeitsberichte* sowie *Projektberichte*.

Die **technische Projektdokumentation** dokumentiert die Organisation und Struktur eines Projekts.

Labor- und Arbeitsberichte dokumentieren den Ablauf und alle Ergebnisse eines Projekts.

Ein **Projektbericht** dokumentiert die Ausgangslage, die Methode, die Ergebnisse und die Schlussfolgerungen eines Projekts. Projektberichte greifen oft auf die technische Projektdokumentation und auf Laborberichte zurück.

Dieses Kapitel beschreibt die Anforderungen und die Funktion der Projektdokumentation.

4.1. Technische Projektdokumentation

Die technische Projektdokumentation dokumentiert die Organisation und Struktur eines Projekts und enthält alle relevanten Informationen, um die Dateien in einem `git`-Repository richtig zu interpretieren und die Arbeitsschritte zu reproduzieren.

Eine vollständige **Projektdokumentation** enthält die folgenden Teile.

- Projekthistorie
- Entscheidungen und Massnahmen
- Projektmotivation bzw. Ausgangslage
- Repository-Organisation
- Technische Information zur Arbeit mit dem Projekt
- Externe Bibliotheken bzw. Abhängigkeiten.
- Vertraulichkeit und Verwendungsrechte

Die Versionierungsgeschichte wird von `git` automatisch erstellt. Die **Commit-Meldungen** dokumentieren die Massnahmen, welche in der zugehörenden Version umgesetzt wurden.

4.1.1. Entscheidungen und Massnahmen

Weil Die Commit-Meldungen nicht viel Platz zu dokumentation bieten, werden *Entscheidungen* in der Regels als **Issues** und *Massnamen* als **Pull-Requests** einer `git`-Hosting-Plattform dokumentiert. Dabei gilt, dass *offene* Issues oder Pull-Requests noch nicht und *geschlossene* Issues und Pull-Requests vollständig im Projekt umgesetzt wurden. Commit-Meldungen müssen dann nur noch auf die zugehörigen *Issues* referenzieren.

4.1.2. Projektübersicht

Per Konvention werden in der Datei `README.md` bzw. `README` die Ausgangslage und Motivation, technische Informationen sowie die Repository-Organisation dokumentiert. Diese Dokumentation ist notwendig, um mit dem Projekt arbeiten zu können. Zu den technischen Informationen gehören alle Informationen, die benötigt werden, um mit den Daten oder dem Code zu arbeiten.

Strukturierte Daten müssen in der Projektübersicht dokumentiert werden. Die folgenden Informationen über die Daten sollten angegeben werden.

- die Namen von Variablen
- die Datentypen der Variablen
- die Bedeutung der Variablen

Für Code ist die Programmiersprache, eine Anleitung, um den Code laufen zu lassen, sowie ein Beispielaufruf anzugeben. Dieser Teil der Dokumentation hilft beim Aufsetzen des Projekts in einer neuen Arbeitsumgebung.

4.1.3. Externe Abhängigkeiten

In Datenprojekten werden oft externe Bibliotheken, Module oder Pakete verwendet, die separat installiert werden müssen. Diese Komponenten bezeichnet man als **externe Abhängigkeiten**, die ebenfalls dokumentieren werden müssen, damit alle externen Abhängigkeiten in einer neuen Arbeitsumgebung korrekt installiert werden können. Dieser Teil der Projektdokumentation benötigt den Namen der verwendeten Bibliothek sowie die erforderliche Versionsnummer. Für meisten modernen Programmiersprachen existieren sog. *Packetmanager*, welche die externen Abhängigkeiten in einer Datei dokumentieren und installieren können.

Die folgende Tabelle zeigt Paketmanager und die Datei für externe Abhängigkeiten in ausgewählten Programmiersprachen. Diese Liste ist nicht vollständig. Die Datein für externe Abhängigkeiten erfüllen die Dokumentation der externen Abhängigkeiten und können gleichzeitig für die Installation der externen Abhängigkeiten verwendet werden.

Programmiersprache	Paketmanager	Datei für externe Abhängigkeiten
Python	pip	`requirements.txt`
R	renv	`renv.lock`, `renv/settings.json`
R	pak	`pkg.lock`
Julia	Pkg	`Project.toml`
JavaScript	npm	`package.json`
Java	Maven	`pom.xml`
PHP	composer	`composer.json`

4.1.4. Vertraulichkeit, Verwendung und Urheberschaft

Per Konvention wird die **Vertraulichkeit** und **Verwendungsrechte** in der Datei `LICENCE.md` oder `LICENSE` dokumentiert. Dabei handelt es sich um ein rechtliches Dokument, dass oft von einer Organisation oder Auftraggeber vorgegeben wird oder, wie im Fall von Open Source Lizenzen, durch einen Verband erstellt wurden. Die Lizenzdatei wird in aller Regel in einem Projekt nicht oder nur sehr selten geändert.

> **Achtung**
>
> Fehlt eine Lizenzdatei ist das Projekt nicht lizenzlos, sondern unterliegt dem gesetzlichen Urheberrecht am Wohnsitz der Projektbeteiligten. Damit ein Projekt lizenzlos veröffentlicht werden kann, muss dieses explizit in der Lizenzdatei festgehalten werden. Ein Beispiel für eine solchen Vermerk ist die sog. MIT Lizenz.

> **Achtung**
>
> Die Verwendungsrechte *externer Abhängigkeiten* können die Verwendung des eigenen Projekts beeinflussen. Möglicherweise sind einzelne Module mit den Projektanforderungen nicht vereinbar und dürfen in diesen Fällen auch nicht im Projekt verwendet werden. Es ist deshlab wichtig, die Lizenzbedingungen **aller** externen Abhängigkeiten auf ihre rechtliche Kompatibilität mit dem eigenen Projekt zu prüfen.

`git`-Hosting-Plattformen bieten verschiedene Lizenzen beim Erstellen eines neuen Projekts an.

> **Achtung**
>
> Projekte, die mit Bezug auf die Daten und/oder die Auswertungsalgorithmen als *vertraulich* gelten, dürfen nicht in *öffentlichen* Projektrepositories verwaltet werden.

In diesem Fall muss ein *privates* Repository verwendet werden. Unter Umständen ist eine Verwendung einer öffentlichen Plattform nicht möglich. In diesem Fall muss eine eigene `git`-Hosting-Plattform genutzt werden.

Neben der Verwendung und Vertraulichkeit ist auch die **Urheberschaft** zu dokumentieren. Die Urheberschaft ist die Liste der Personen, die an einem Projekt mitgearbeitet haben. Traditionell wurde die Urheberschaft wird in der Regel in der Datei `AUTHORS.md` oder `AUTHORS` dokumentiert.

Tipp

`git`-Hosting-Plattformen erstellen inzwischen die Autorenliste automatisch aus den Commits. Zusätzlich wird auch der Umfang der Beiträge dokumentiert.

4.2. Labor- und Arbeitsbericht

Ein Laborbericht dokumentiert den Ablauf und alle Ergebnisse eines Experiments oder einer Untersuchung. Ein Laborbericht ist eine *technische Dokumentation* einer Untersuchung oder eines Experiments. Diese Dokumentation ist der Beleg für die sachgemässe Durchführung einer Untersuchung. Bei Untersuchungen, die mehr als eine Arbeitssitzung oder -Schicht benötigen, sollten Laborberichte pro Sitzung bzw. Schicht erstellt werden.

Warnung

Laborberichte sind Teil von Complience-Anforderungen und werden oft von Auftraggebern oder Behörden verlangt. In diesen Anforderungen werden die zwingend zu dokumentierenden Teile festgelegt. Unterliegen bestimmte Untersuchungen einer Complience-Anforderung, dann müssen diese Anforderungen eingehalten werden, selbst wenn diese nicht explizit gefordert wurde. Stellt sich später heraus, dass ein vorliegender Laborbericht unvollständig ist oder falsche Angaben zur Durchführung enthält, dann kann dies unter Umständen als Urkundenfälschung gewertet werden.

Laborberichte sollten am Besten während oder unmittelbar nach einer Untersuchung erstellt werden. Werden regelmässig Laborberichte erstellt, bietet sich die Verwendung einer Versionierung mit `git` an.

Bei der Verwendung von Laborberichten wird angenommen, dass undokumentierte Materialien nicht verwendet wurden und nicht dokumentierte Arbeitsschritte oder Ereignisse nicht stattgefunden haben. Ein fehlender oder unvollständiger Laborbericht entspricht einer nicht ordnungsgemäss durchgeführten Untersuchung.

Ein Laborbericht besteht aus den folgenden Teilen.

- Titel
- Beteiligte Untersucher

- Datum, Uhrzeit und Ort
- Fragestellung bzw. Auftrag
- Materialliste
- Detaillierter Versuchsaufbau
- Versuchsdurchführung bzw. Ablauf
- Ergebnisse
- Fehlerdiskussion
- Beobachtungen
- Besondere Ereignisse

4.2.1. Materialliste

Die Materialliste umfasst alle Materialien, die für die Durchführung des Experiments benötigt werden. Grundsätzlich müssen alle Materialien und Geräte, die für die Durchführung einer Untersuchtung verwendet werden, in der Materialliste aufgeführt werden müssen. Zu den Materialien gehören zum Beispiel:

- Chemikalien und Reagenzien
- Proben
- Behälter, Gefässe und Verbindungen
- Messgeräte
- Kabel und Stecker
- Werkzeuge
- Verbrauchsmaterialien
- Schutzkleidung
- Entsorgungsmaterialien

Falls eine Studie oder ein Experiment nicht in einer kontrollierten Laborumgebung durchgeführt wird, müssen auch die **Umweltbedingungen** am Untersuchungsort dokumentiert werden.

4.2.2. Versuchsaufbau

Der Versuchsaufbau beschreibt die Anordnung der Materialien und Geräte bei der Durchführung eines Experiments. Dieser Abschnitt beschreibt im Detail den Aufbau eines Versuchs oder einer Untersuchung, so dass diese später genau gleich wiederholt werden kann. Für den Versuchsaufbau werden zusätzlich Skizzen und Schaltpläne angegeben. Falls der Versuchsaufbau spezielle Software für die Durchführung benötigt, sind alle Komponenten und Einstellungen ebenfalls im Versuchsaufbau zu dokumentieren.

> **Hinweis**
>
> Die Software für die Auswertung ist *kein* Teil des Versuchsaufbaus.

Bei wiederholten Durchführungen mit dem gleichen Versuchsaufbau kann der Versuchsaufbau in einem separaten Dokument oder in einem separaten Abschnitt dokumentiert werden. Dieses Dokument ist dann für alle Versuchsdurchführungen mit dem gleichen Versuchsaufbau gültig und muss in *allen zugehörigen Laborberichten* referenziert werden.

4.2.3. Durchführung und Ablauf

Der Abschnitt Durchführung beschreibt die genaue Vorgehensweise einer Untersuchung oder eines Experiments. Dabei werden alle ***durchgeführten*** Arbeitsschritte in *chronologischer* Reihenfolge festgehalten. Die Beschreibung der Durchführung muss so genau sein, dass die Untersuchung oder das Experiment später genau gleich wiederholt werden kann.

Wenn besondere Vorsichtsmassnahmen bei der Durchführung notwendig sind, müssen diese ebenfalls im Abschnitt Durchführung dokumentiert werden.

Die **Entsorgung** von Nebenprodukten und Abfällen ist Teil der Versuchsdurchführung. Die Entsorgung von Nebenprodukten und Abfällen wird oft in einem separaten Abschnitt dokumentiert.

Bei manchen Untersuchung ist ein besonderer Rückbau des Versuchsaufbaus notwendig. Der Rückbau des Versuchsaufbaus wird ebenfalls in der Versuchsdurchführung dokumentiert.

4.2.4. Ergebnisse

Alle gemessenen Ergebnisse eines Experiments oder einer Untersuchung müssen im Abschnitt *Ergebnisse* dokumentiert werden. Das gilt auch für Messwerte, die später nicht für die Auswertung verwendet werden. Die Ergebnisse werden in Tabellen *und* Diagrammen dargestellt. Die Tabellen und Diagramme müssen mit einer Nummer versehen werden und eine aussagekräftige Beschriftung haben. Die Tabellen mit den Messwerten sind immer verpflichtend. Diagramme sind *optional*.

> **Achtung**
>
> Die Ergebnisse müssen *vollständig* dokumentiert werden. Das bedeutet, dass alle Messwerte, die für die Auswertung nicht verwendet werden, trotzdem dokumentiert werden müssen.

Weil die Messwerte später ausgewertet werden sollen, können die Tabellen in einer separaten Datei gespeichert werden. Diese Datei wird dann im Abschnitt Ergebnisse als *Anhang* referenziert. Sollte eine Untersuchung mehrere Tabellen erzeugt haben, dann sollten diese als eigenständige Anhänge geführt werden.

4.2.5. Fehlerdiskussion

Die Fehlerdiskussion eines Projektberichts umfasst die bekannten Fehlerquellen und Fehlerabschätzungen einer Studie. Dieser Teil dient zur Bestimmung der Messgüte der Einschätzung der präsentierten Ergebnisse.

In diesem Teil müssen auch methodische Fehler berichtet werden, falls diese die ordnungsgemässe Messung oder Auswertung beeinflussen.

Messausfälle oder nicht durchgeführte Messungen sind nicht Teil der Fehlerdiskussion, sondern fallen in die Rubrik *besondere Ereignisse* und müssen dort berichtet werden.

4.2.6. Besondere Ereignisse

In einem Laborbericht werden grundsätzlich alle besonderen Ereignisse dokumentiert. Besondere Ereignisse sind Ereignisse, die nicht zum normalen Ablauf gehören oder für das Experiment nicht zu erwarten waren. Beispiele für besondere Ereignisse sind:

- Ungewöhnliche Messwerte und Messausfälle
- Ungewöhnliche Geräusche, Gerüche oder Verfärbungen
- Verzögerungen und Unterbrechungen
- Ausfall von Geräten
- Warnmeldungen
- Defekte oder fehlerhafte Geräte
- Unfälle und Verletzungen
- Unkontrolliertes oder ungeplantes Austreten von Flüssigkeiten oder Gasen

Falls keine besonderen Ereignisse aufgetreten sind, können diese Teile weggelassen werden.

4.2.7. Beobachtungen und Notizen

Laborberichte sind eine wichtige Quelle für Innovation und Erkenntnisse. Deshalb sollten Beobachtungen, Erfahrungen und andere Notizen in Laborberichten festgehalten werden. Dadurch wird sichergestellt, dass dieser Teil der Dokumentation nicht von der Untersuchtung getrennt und so dekontextualisiert wird.

4.3. Projektbericht

Ein Projektbericht dokumentiert die Ausgangslage, die Methode, die Ergebnisse und die Schlussfolgerungen eines Projekts.

Ein Projektbericht besteht ***immer*** aus den folgenden Teilen.

- Titel
- Autoren

- Zusammenfassung (Abstract)
- Einleitung und Hintergrund
- Forschungsfrage/Arbeitsauftrag
- Methode
- Ergebnisse
- Interpretation (Diskussion)
- Ausblick
- Quellenverzeichnis

Bei längeren Projektberichten muss zusätzlich ein Inhaltsverzeichnis, ein Abbildungsverzeichnis, ein Tabellenverzeichnis und ein Abkürzungsverzeichnis sowie eine Managementzusammenfassung erstellt.

4.3.1. Einleitung und Hintergrund

Die Einleitung legt den Kontext des Projekts fest. Dazu gehört die übergeordnete Problemstellung und die bekannten Fakten zum Thema. Die Problemstellung sollte so formuliert sein, dass auch interessierte Personen, die nicht mit dem Thema vertraut sind, die Ausgangslage verstehen können. Die Aufstellung der bekannten Fakten ist eine Zusammenfassung der Literatur zu dieser Problemstellung. Es kommt nicht selten vor, dass Querbezüge zu anderen Themen oder Problemstellungen hergestellt werden müssen. Auch diese Literatur muss in der Einleitung zusammengefasst werden.

Die Literatur sollte so zusammengefasst werden, dass für die projektrelevante Aspekte, die in der Literatur identifiziert wurden, ein Absatz im Abschnitt gewidmet wird. Dabei müssen die entsprechenden Quellen referenziert werden. Falls mehrere Quellen die gleiche oder sehr ähnlich Aussagen treffen, dann können diese im gleichen Absatz zusammengefasst und gemeinsam referenziert werden.

4.3.2. Forschungsfrage oder Arbeitsauftrag

Der Abschnitt Forschungsfrage oder Arbeitsauftrag legt die konkrete Motivation des Projekts fest. Diese sollte durch den Kontext der bekannten Fakten und der Problemstellung in der Einleitung sachlich begründet sein.

Als Frage formuliert liefert der Abschnitt Ausblick die Antwort auf diese Frage. Dadurch entsteht eine rhetorische Klammer im Projektbericht. Gleichzeitigt hiflt die Formulierung der Fragestellung bei der kontinuierlichen Überprüfung, ob die durchgeführten Schritte im Projekt auch zur Beantwortung der Fragestellung beitragen. Deshalb sollten auch Arbeitsaufträge, die nicht als Frage präsentiert werden, intern als Frages umformuliert werden.

4.3.3. Methode

Die Methode beschreibt das Vorgehen bei der Erhebung und Auswertung von Daten. Die Beschreibung muss eine Wiederholung oder das systematische Nachvollziehen der Untersuchung ermöglichen. In diesem Abschnitt sind alle Arbeitsschritte zu dokumentieren, die in den Abschnitten Ergebnisse und Interpretation zu den präsentierten Ergebnissen führen.

Die Methode umfasst nur die systematische Vorgehensweise, aber keine technische Dokumentation der Auswertung. Hierzu sollte auf die technische Projektdokumentation verwiesen werden.

4.3.4. Ergebnisse

Der Abschnitt Ergebnisse fasst alle erhobenen Messungen zusammen. Dabei werden die Messwerte in Tabellen und Diagrammen dargestellt. Alle in diesem Abschnitt präsentierten Daten sollten sich direkt aus Laborberichten ableiten lassen.

Im Abschnitt Ergebnisse werden selten die gemessenen Rohdaten als Tabelle präsentiert. Sollten die Rohdaten für die Interpretation bedeutsam sein, dann sollten diese Daten in einem Anhang getrennt präsentiert werden.

Dieser Abschnitt muss alle im Abschnitt *Interpretation* bzw. *Diskussion* verwendeten Daten enthalten. Für *jede Aussage* im Abschnitt Interpretation muss ein Bezug zu den Ergebnissen möglich sein.

Dieser Abschnitt enthält für empirischen Studien normalerweise *keine Referenzen* auf externe Quellen.

4.3.5. Interpretation

Die Interpretation umfasst die *Bewertung* der Ergebnisse. Dabei werden die Ergebnisse mit den Erwartungen aus der Einleitung verglichen. In diesem Abschnitt werden die Ergebnisse in den in der Einleitung hergestellten Kontext gestellt und die Schlussfolgerungen aus der Auswertung gezogen. In diesem Abschnitt werden die Argumente für die Beantwortung der Forschungsfrage bzw. des Arbeitsauftrags vorbereitet.

Die Interpretation enthält *keine* neuen Daten, sondern nur eine *Bewertung* der Ergebnisse. Die Bewertung der Ergebnisse muss *immer* mit den Erwartungen aus der Einleitung und Fragestellung verglichen werden.

4.3.6. Ausblick

Der Ausblick beantwortet die Forschungsfrage bzw. den Arbeitsauftrag. Dabei werden die Ergebnisse der Untersuchung zusammengefasst und die Schlussfolgerungen daraus gezogen. Häufig werden die Schlussfolgerungen als *Empfehlungen für die Praxis* formuliert.

Falls in einem Projekt unerwartete Ergebnisse auftraten oder Phänomene beobachtet wurde, die nicht durch as Projekt erklärt werden können, dann sollten Fragen für die weitere Forschung formuliert werden. Diese Fragen sollten als Forschungsfragen analog zum Abschnitt Forschungsfrage formuliert werden.

> **Tipp**
>
> Falls über ein Teilprojekt berichtet wird, dann gelten die nächsten *geplanten* Projektfragen *nicht* als Fragen für die weitere Forschung.

4.3.7. Zitieren und Quellenverzeichnis

Das Zitieren von Quellen ist ein wichtiger Teil der wissenschaftlichen Arbeit. Das Zitieren von Quellen dient dazu, die Herkunft von Ideen, Aussagen und Ergebnissen zu dokumentieren. Das Referenzieren externer Quellen ist verpflichtend. Im Bericht wird auf die Quellen in einer Kurzform verweisen. Die vollständigen Angaben aller Quellen werden im Quellenverzeichnis aufgeführt.

Wie Quellen richtig referenziert werden, ist in sog. Zitierstandards festgelegt. Wichtige Zitierstandars sind der APA Styleguide (American Psychological Association, 2020), der Chicago Styleguide (The University Of Chicago Press Editorial Staff, 2017), der IEEE Styleguide (IEEE, 2018) oder der ACM Styleguide (Rodkin, 2023). In einem Projektbericht wird nur ein Zitierstandard verwendet. Es ist nicht vorgesehen, Zitierstandards zu mischen.

Eine wörtliche Übernahme einer anderen Quelle wird als Zitat bezeichnet. Zitate sollten sehr sparsam in Berichten verwendet werden. Zitate sind oft nur dann notwendig, wenn die wörtliche Übernahme einer Aussage oder eines Ergebnisses für die Argumentation erforderlich ist.

Werden Ideen, Aussagen oder Ergebnisse aus einer externen Quelle verwendet, dann ist das eine Paraphrase und erfordert wie ein Zitat eine Referenz der Quelle. Bei Paraphrasen ist die Angabe der Seitenzahl nur dann notwendig, wenn die referenzierte Idee sich nicht direkt aus der Quelle ergibt. Das ist zum Beispiel der Fall, wenn die Quelle eine Monographie mit mehreren Studien ist und nur eine bestimmte Studie referenziert wird.

> **Achtung**
>
> Werden externe Quellen verwendet und *nicht referenziert*, dann handelt es sich um ein Plagiat. Plagiate sind ein schwerer Verstoss gegen die wissenschaftliche Integrität und

das Urheberrecht. Bei Zertifizierungen sind Plagiate ausserdem ein Betrug. Das gilt auch für Paraphrasen ohne Referenz.

4.3.8. Management-Zusammenfassung

Eine Management zusammenfassung ist eine Kurzfassung eines Projektberichts, die sich an die Entscheidungsträger richtet. Sie ist deutlich kürzer als die Zusammenfassung und enthält nur die wichtigsten Informationen des Projektberichts. In der Regel ist die Managementzusammenfassung nicht länger als eine Seite und ist Stichpunktartig formuliert.

Eine gute Managementzusammenfassung kann direkt in einer Präsentation verwendet werden. Als Richtlinie für eine Managementzusammenfassung sind sieben Punkte. Diese Punkte sollten die folgenden Fragen beantworten.

- Was ist das Problem?
- Wie wurde gemessen?
- Was sind die wichtigsten Ergebnisse?
- Welche Schlussfolgerungen können gezogen werden?
- Welche Massnahmen werden empfohlen? (optional)

5. Daten sammeln

Bevor Daten analysiert werden können, müssen sie zuerst gesammelt werden. Je nach dem Ziel und Quelle können Daten *unstrukturiert*, *semi-strukturiert* und *strukturiert* gesammelt werden. Das Ergebnis ist immer ein Datensatz, welcher Werte für die Datenanalyse bereitstellt. Je nach Vorgehensweise kommen unterschiedliche Werkzeuge zum automatisierten und manuellen Datensammeln in Frage.

Das Ziel des Datensammelns ist das Erstellen von *Datensätzen*.

Definition 5.1. Ein **Datensatz** umfasst zusammengehörende Werte.

Beim Sammeln von Daten werden *Beobachtungen* erfasst und dokumentiert. Dabei entspricht ein Datensatz immer einer Beobachtung. Über den zugehörigen Datensatz sollte eine Beobachtung zu einem späteren Zeitpunkt nachvollziehbar sein. Das ist allerdings nur mithilfe der erfassten Daten möglich.

Durch nicht erfasste Daten geht Information verloren. Dieser Informationsverlust ist grundsätzlich *unvermeidbar*, weil nicht jeder Aspekt einer Beobachtung in Daten kodiert oder erfasst wird. Manche Daten sind für die spätere Analyse von besonderer Bedeutung, weil ohne sie eine Auswertung unmöglich ist. Deshalb müssen die relevanten Daten ***vor*** der Datenerhebung bestimmt werden.

5.1. Das Datenschema

Welche Daten in einem Datensatz erfasst wurden bzw. erfasst werden sollen, werden in einem **Datenschema** festgehalten. Das Datenschema hat zwei Funktionen.

1. Es legt fest, welche Daten für einen Datensatz erhoben werden müssen. In dieser Funktion ist das Datenschema eine grobe Anleitung welche Daten gesammelt werden sollen.
2. Es dokumentiert, welche Daten in einem Datenssatz vorliegen. Ein Datenschema hilft so Analysten, die Daten richtig zu auszuwerten und zu interpretieren.

Ein Datenschema legt Felder für die Ablage der beobachteten Daten fest.

Jedes Datenfeld erhält immer einen eindeutigen **Namen** für ein Datenfeld. Über den Namen können die im Datenfeld erfassten Daten zugegriffen werden. Die Namen von Datenfeldern dienen der einfacheren Handhabung bei der digitalen Auswertung. Deshalb sollten diese Namen möglichst einfach und gleichzeitig die Daten etwas beschreiben. Damit

beim Auswerten weniger Fehler passieren, sollten Namen nur die Zeichen `a-z`, `0-9` und `_` enthalten und mit einem Buchstaben beginnen.

Zusätzlich kann ein Datenfeld einen **Wertebereich** haben. Ein Wertebereich legt die zulässigen Werte für ein Datenfeld fest. Für die automatisierte Auswertung kann es hilfreich sein, auch das **Skalenniveau** (s. Kapitel 8.2) des Wertebereichs zu dokumentieren.

Falls die Daten eine **Masseinheit** haben, muss diese auch dokumentiert werden.

Es ist üblich, für jedes Datenfeld eine **Kurzbeschreibung** anzugeben. Diese Beschreibung gibt zusätzliche Hinweise über die erfassten Werte. Die Beschreibung ist rein *informativ* und richtet sich an Menschen, die Daten für das Schema erheben oder diese Daten auswerten.

Grundsätzlich können alle Datenfelder frei bestimmt werden. Einige Datenfelder sind für fast jede Studie gegeben. Zu diesen Datenfelder gehören beispielsweise der *Zeitpunkt der Beobachtung*, der *Ort der Beobachtung*, die *beobachtende Person* und die *Dauer der Beobachtung*. Bei der technischen Datenerfassung können weitere Felder durch die verwendeten Werkzeuge automatisch bereitgestellt werden. Deshalb sollten die technischen Möglichkeiten vor einer Datenerhebung geprüft werden, um das mehrfache abfragen der gleichen Daten zu vermeiden.

5.2. Arten von Daten

Beim Daten sammeln werden drei Arten von Daten unterschieden:

- Unstrukturierte Daten
- Strukturierte Daten
- Semi-strukturierte Daten

5.2.1. Unstrukturierte Daten

Definition 5.2. Unstrukturierte Daten haben keinen festgelegten Wertebereich.

Typische unstrukturierte Daten sind Notizen, Transkripte gespochener Sprache oder Videos. Unstrukturierte Daten können Werte von Interesse enthalten, auf die jedoch nicht direkt zugegriffen werden kann.

Unstrukturierte Daten haben den Vorteil, dass sie *Ergebnisoffen* ein. Dadurch sind unstrukturierte Daten offen für neue und unvorgesehene Information. Entsprechend werden Techniken, die unstrukturierte Daten erheben auch als ***offene*** Erhebungstechniken bezeichnet.

Diese Offenheit hat ihren Preis, denn unstrukturierte Daten sind anfällig gegenüber Rauschen und Equivokation. Ausserdem können einzelne Werte nicht direkt aus unstrukturierte Daten abgelesen werden, sondern müssen (oft umständlich) aus den Daten extrahiert werden.

> **Hinweis**
>
> Unstrukturierte Daten erfordern immer zusätzliche Arbeitsschritte, bevor mit den in ihnen vorhandenen Daten gearbeitet werden kann. Diese Arbeitsschritte zusammenfassend werden **Kodieren** (engl. **coding**) genannt. Das Kodieren unstrukturierter Daten führt einheitliche Markierungen, die sog. *Annotationen*, ein, die eine Struktur über die Daten legen.

Ohne vorbereitende Kodierung lassen sich unstrukturierte Daten kaum systematisch analysieren. Sehr häufig handelt es sich bei unstrukturierten Daten um Texte. Im Kapitel 11 werden einfache Techniken zum Verarbeiten von unstrukturierten Daten beschrieben.

5.2.2. Strukturierte Daten

Definition 5.3. Strukturierte Daten haben einen festen Wertebereich.

Weil strukturierte Daten einen festen Wertebereich haben, ist sichergestellt, dass nur Werte innerhalb dieses *abgeschlossenen* Bereichs gültig sind. Deshalb werden Techniken, mit denen strukturierte Daten erhoben gelten auch als ***geschlossene*** Erhebungstechniken.

Strukturierte Daten basieren immer auf einer festen Wertebereich, die Information kodiert. In dieser Struktur werden die Daten nach ihrer Bedeutung und ihren zulässigen Wertebereichen organisiert. Die feste Struktur macht strukturierte Daten innerhalb der Struktur weniger anfällig gegenüber Rauschen als unstrukturierte Daten.

Weil strukturierte Daten eine Struktur zur Informationskodierung vorgeben, lassen sich nur Informationen über diese Struktur kodieren. Das führt zu Equivokation, weil nur die Teile der Information kodiert werden können, die in den Wertebereichen zulässig sind.

Tabellen sind typische strukturierte Daten, die in Spalten und Zeilen organisiert sind.

Strukturierte Daten sind nicht auf Tabellen beschränkt.

Listing 5.1 Strukturierte Daten im YAML-Format

```yaml
hamster:
- name: Fred
  gewicht: 175
  zeit: 2023-04-11T10:20:30Z
- name: Frida
  gewicht: 160
  zeit: 2023-04-12T09:10:11Z
```

5.2.3. Semi-strukturierte Daten

Neben diesen zwei Arten existieren noch sog. semi-strukturierte Daten. Semistrukturierte Daten folgen einer logischen Struktur, die allerdings nicht durchgehend strickt eingehalten wird.

Definition 5.4. Semi-strukturierte Daten sind *unstrukturierte Daten*, die Markierungen für Werte beinhalten.

Semi-strukturierte Daten werden oft für das Einbetten strukturierter Daten in unstrukturierte Datenfelder verwendet.

Ein typisches Beispiel für semistrukturierte Daten sind Social-Media-Meldungen mit Hashtags und `@`-Nennungen. Mit dem Hash-Symbol (`#`) werden Themen und Inhalte hervorgehoben und mit dem `@`-Symbol werden Nutzende markiert. Durch die beiden Symbole wird eigendlich unstrukturierten Daten eine zusätzliche Struktur gegeben, auf welche zugegriffen werden kann. Eine Social-Media-Meldung kann keine, ein oder mehrere Hashtags und keine, eine oder mehrere Nennungen enthalten.

Listing 5.2 Tweet mit Hashtag (ZHAW, 2023)

```
Das neue Laborgebäude auf dem ZHAW-Campus Reidbach
in #Wädenswil wurde gestern eingeweiht. Genutzt wird der
Neubau vom Institut für Lebensmittel- und Getränkeinnovation,
wo unter einem Dach die gesamte Wertschöpfungskette von
Lebensmitteln erforscht wird.
https://zhaw.ch/de/ueber-uns/aktuell/news/
detailansicht-news/event-news/
lebensmittelforschung-unter-einem-dach-vereint/
```

Ein anderes Beispiel finden sich regelmässig in Bankmiteilungen im Feld “Verwendungszweck”. In diesem Feld werden oft zusätzliche Daten für die Kommunikation zwischen den Beteiligten einer Transaktion hinterlegt. Listing 5.3 zeigt semistrukturierte Daten im Verwendungszweck von Banktransaktionen, diese können für eine Analyse verwendet werden.

Listing 5.3 Semistrukturierte Daten

```
1   PV2332 PG386.5 SV375.26
2   PG374.23
3   Ref: 106030763221202, Knr: 783924
```

In diesem Beispiel sind einmal drei Werte, einmal ein Wert und einmal zwei Werte in einem Eintrag vorhanden. Dabei kommt die Buchstabenfolge `PG` einmal als zweiter Wert

und einmal alleinstehend vor. Eine Behandlung als Tabelle dieser Werte ist wahrscheinlich nicht zielführend.

Die Buchstabenfolgen `PV`, `PG` und `SV` sowie `Ref:` und `Knr:` erfüllen den gleichen Zweck, wie Symbole `#` und `@` in Social-Media-Meldungen. Diese Markierungen sind *Codes*, die wichtige Daten hervorheben und relativ leicht identifizieren lassen. Die Codes nehmen das Kodieren unstrukturierter Daten vorweg.

Solche Codes sind *transaktionsspezifisch*, so dass nicht alle Codes vorkommen müssen und bestimmte Codes nur bei bestimmten Transaktionen vorkommen oder die gleichen Codes in unterschiedlichen Transaktionen andere Bedeutungen haben. Oft folgen diese Codes einem eigenen Schema.

5.3. Daten erheben

Die eigentliche Datensammlung kann manuell, technisch unterstützt oder automatisch erfolgen. Bei der *manuellen Datenerhebung* werden alle Daten von einer Person erfasst und festgehalten. Bei der *technisch-unterstützten Datenerhebung* helfen spezielle Erhebungswerkzeuge, die Datenerhebung zu vereinfachen, indem sie Werte überprüft werden oder automatisch erfasst werden. Bei der *automatischen Datenerhebung* werden Daten durch *Sensoren* erfasst und gespeichert.

5.3.1. Manuelle Datenerhebung

In der Laborarbeit ist die Erfassung von Messwerten in Tabellen oder Listen üblich. Dabei werden die Werte von Hand direkt in eine Tabelle oder Liste eingetragen. So erfasste Tabellen oder Listen sind *strukturierte Daten*. Beim Eintragen der Daten muss darauf geachtet werden, dass die Werte an der richtigen Stelle eingetragen werden, weil solche Fehler oft unerkannt bleiben und eine nachträgliche Korrektur oft nicht möglich ist.

Die einfachste Form der manuellen Datenerfassung sind *Notizen*. Notizen sind persönliche Aufzeichnungen über die eigene Arbeit bzw. über Beobachtungen bei der Arbeit. Diese Daten sind entweder *unstrukturiert* oder *semi-strukturiert*. Sie erlauben eine grosse Flexibilität falls die Daten nicht vorstrukturiert werden können oder die Struktur weitgehend unbekannt ist. Notizen lassen sich schwer *objektivisieren*, weil sie immer einen subjektiven Anteil der Person haben, welche die Notiz verfasst hat.

> **Praxis**
>
> Es ist eine gute Idee, Messwerte in Tabellen zu erfassen und zusätzliche Notizen zu machen. Aus beiden ergeben sich Laborberichte, die wiederum zu Labortagebüchern wachsen. In der wissenschaftlichen Praxis sind die strukturierten Daten zwar das Hauptergebnis einer Untersuchung, aus den Notizen ergeben sich jedoch oft neue Einsichten, die sich nicht aus den strukturierten Daten herauslesen lassen.

Neben Notizen sind *Transskripte* ein wichtiges Werkzeug der manuellen Datenerfassung. Transskripte sind eine Verschriftlichung einer anderen Aufzeichnung. Oft sind das Tonaufzeichnungen von Gesprächen oder Videoaufzeichnugen eines Experiments. Diese Aufzeichnungen liefern immer *unstrukturierte Daten*. Die ursprüngliche Aufzeichnung ist dabei *meistens* eine Primärquelle für die Daten. Das Transskript ist eine *abgeleitete Sekundärquelle* der Primärquelle. Weil beim Transskribieren die Primärquelle *neu kodiert* wird, muss beim Transskribieren besonders sorgfältig gearbeitet werden, um die usprünglichen Daten nicht zu sehr zu verfälschen.

Inzwischen gibt es brauchbare Werkzeuge zur Transskription von Mediendateien. Diese automatisch erstellten Transkripte müssen aber immer nachkontrolliert werden. Nur so wird das Risiko reduziert, dass sich die Bedeutung der Daten durch Transskriptionsfehler ändert.

5.3.2. Technisch-unterstützte Datenerhebung

Bei der technisch-unterstützten Datenerhebung soll das manuelle Datensammeln durch technische Werkzeuge vereinfacht werden, indem bestimmte Daten automatisch erfasst werden. Ein wichtiges Instrument für die technisch-unterstützte Datenerhebung sind *Formulare*. Formulare sind strukturierte Dokumente zur systematischen und wiederholten manuellen Eingabe von Daten durch Menschen.

Im Gegensatz zu Papierfromularen können digitale Formulare die Eingaben überprüfen und bieten unterschiedliche Eingabeformate. Digitale Formulare können auch automatisch Daten erfassen, die nicht direkt eingegeben werden, wie beispielsweise den Zeitpunkt der Eingabe. Die populärste Art der digitalen Formulare sind inzwischen online Formulare, weil sie die grösste Flexibilität bei einer vergleichsweise niedrigen Nutzungsschwelle bieten.

Es lassen sich zwei Arten von Diensten für online Formulare unterschieden:

1. Formulardienste, wie z.B. MS-Forms (Microsoft, 2023) oder Google Forms (Google, 2023).
2. Umfrage- bzw. Survey-Dienste, wie z.B. Survey Monkey oder Lime Survey.

Der wesentliche Unterschied zwischen diesen Diensten sind zusätzliche Funktionen zur Datenerhebung. Survey-Dienste bieten zusätzliche Funktionen, wie die Anonymisierung der Daten, zusätzliche Daten über die Ausführung, Fragebogendokumentation oder die Unterstützung von wiederholten Befragungen.

Alle Formulare erlauben die Eingabe von Daten mit *offenen* und *geschlossenen* Wertebereich sowie das Auslesen der Daten in tabellarischer Form. Für jedes ausgefüllte Formular wird dazu eine Zeile erstellt, wobei die Eingabefelder jeweils als eine Spalte abgebildet werden.

Definition 5.5. Eine Eingabemöglichkeit eines Formulars wird als **Fragebogen-Item** oder schlicht als **Item** bezeichnet.

Items mit offenen Wertebereich sind Langantworten und Dateien. Alle anderen Eingaben haben einen geschlossenen Wertebereich. Die sog. Kurzantworten erscheinen als offene Eingabefelder. Ihr Wertebereich kann aber durch Validierungsregeln eingeschränkt werden.

Bei den Items mit geschlossenen Wertebereich kann zwischen einfachen und mehrfachen Eingaben unterschieden werden. Bei einfachen Eingaben kann nur eine Antwortmöglichkeit ausgewählt werden (sog. Single-Choice). Der Wertebereich dieser Items wird über die Antwortmöglichkeiten festgelegt.

Skalen sind **Single-Choice-Items** für einen *ordinalen Wertebereich*. Oft werden die Antwortmöglichkeiten mit einem Rang versehen.

Ein Spezialfall der Skalen ist die **Likert-Skala**, bei der eine lineare Skala mit einer Bewertung zwischen zwei Extremwerten versehen wird. Die Extrem-Pole sollten möglichst weit auseinander gewählt werden. Z.B. *"trifft absolut nicht zu"* und *"trifft voll und ganz zu"* und ***nicht*** *"trifft nicht zu"* und *"trifft zu"*. Obwohl Likert-Skalen *subjketive Ansichten* messen, wird der Wertebereich meist als *intervallskalierter Wertebereich* behandlet.

Bei mehrfachen Eingaben können mehrere Antwortmöglichkeiten ausgewählt werden. Mehrfache Eingaben werden auch als **Multiple-Choice-Items** bezeichnet. Im Gegensatz zu Single-Choice-Items haben Multiple-Choice-Items *immer* einen *binären Wertebereich*. Das bedeutet, dass jede Antwortmöglichkeit nur zwei Werte annehmen kann: *ausgewählt* oder *nicht ausgewählt*. Intern werden Multiple-Choice-Items als separate Items gespeichert und auch so tabellarisch dargestellt. Ein Multiple-Choice-Item hat deshalb immer eine *Ja/Nein*-Frage als Grundlage, wobei die fehlende Auswahl durch einen nicht vorhandenen Eintrag repräsentiert wird.

Die sog. **Grid-Items** oder **Fragebatterien** sind Items, bei denen mehrere Fragen mit der gleichen Antwortmöglichkeiten gestellt werden. Entsprechend haben diese Items den *gleichen Wertebereich*. Zu den Grid-Items sind oft als Skalen oder als Multiple-Choice-Items organisiert. Diese Items werden oft als Matrix dargestellt, wobei die Fragen als Zeilen und die Antwortmöglichkeiten als Spalten dargestellt werden. Intern werden die einzelnen Fragen von Grid-Items als separate Items gespeichert und werden auch tabellarisch so abgebildet. Dabei gelten die gleichen Regeln wie bei einfachen Single-Choice- und Multiple-Choice-Items.

Eine Fragebatterie aus Likert-Skalen wird **Semantisches Differential** genannt. Oft handelt es sich bei den Items um *Aussagen*, die über die gleiche Skala bewertet werden.

Ein besonderes Grid-Item ist das **Ranking** oder **Sortieren**. Bei dieser Art von Items werden mehrere Antwortmöglichkeiten in *eine* Reihenfolge gebracht. Die Antwortmöglichkeiten werden als eigene Items gespeichert, wobei der Wertebereich für jede Antwortmöglichkeit durch die möglichen Ränge festgelegt ist. Diese Items haben einen *ordialen Wertebereich*.

Eine Variante des Sortieren sind **Zuordnungs-Items**. Bei diesen Items muss jedem Wert aus einer Liste von Werten jeweils ein Wert aus einer zweiten Liste zugeordnet werden. Die Werte aus der ersten Liste werden als Items gespeichert und die Werte aus der zweiten Liste legen den Wertebereich der Items. Diese Items haben einen *nominalen Wertebereich*.

Gelegentlich werden Fotos oder Bilder angeboten und die Teilnehmenden werden gebeten, Bereiche auf diesen Bildern auszuwählen. Dabei handelt es sich um eine Variante eines **Multiple-Choice-Grids**, bei dem die Antwortmöglichkeiten als Bilder dargestellt werden.

Durch die Wahl der Fragetypen und dem Festlegen der Wertebereiche von Formular-Items werden wird die Information *kodiert*.

5.3.3. Automatische Datenerhebung

Bei der automatischen Datenerhebung werden Daten automatisch erfasst und gespeichert.

Bei der automatischen Datenerhebung werden grundsätzlich drei Kategorien unterschieden:

- **Snapshots** erfassen eine Momentaufnahme eines Systems. Bei Snapshots handelt es sich immer um komplexe *strukturierte Daten*. Ein Snapshot kann beispielsweise ein Foto, ein MRI-Scan oder eine Gen-Sequenz sein. Ein Snapshot liefert einen *Datensatz*, der in der Regel sehr viele Werte umfassst. Snapshots werden oft von speziellen Geräten erfasst und gespeichert.
- **Metriken** geben Auskunft über *Zustände* zu einem bestimmten Zeitpunkt. Oft werden Metriken in regelmässigen Abständen erfasst. Bei Metriken handelt es sich immer um einfache *strukturierte Daten*. Eine Metrik kann beispielsweise die Anzahl der Besucher einer Webseite zu einem bestimmten Zeitpunkt oder die Temperatur in einem Raum sein. Eine Metrik hat üblicherweise drei Eigenschaften:
  - *Name*
  - *Wert*
  - *Zeitpunkt*
- **Logs** geben Auskunft über *Ereignisse* in einem System. Ein Log-Eintrag wird erfasst, wenn ein Ereignis eintritt. Logs sind meist *strukturierte* oder *semi-strukturierte Daten*. Ein Log-Eintrag kann beispielsweise die Anmeldung eines Benutzers an einem System oder das Öffnen einer Webseite sein. Ein Log-Eintrag ist oft über zwei Hauptmerkmale definiert:
  - *Zeitpunkt*
  - *Ereignismeldung*

  Log-Einträge werden oft chronologisch in der Reihenfolge der Ereignisse gespeichert.

Nur weil Sensoren und andere datengenerierende Instrumente verwendet werden, findet eine automatische Datenerhebung unter Umständen nicht statt. Ein entsprechendes Instrument muss die Funktion bereitstellen, um Metriken und/oder Logs zu generieren und bereitzustellen.

Bei der automatischen Datenerhebung für Metriken und Logs werden meistens mehrere Systeme gemeinsam eingesetzt. Ein System erfasst die Daten und ein anderes System

speichert die Daten in einer Datenbank. Die Daten werden in der Regel in einem *Datenstrom* bzw. *Zeitreihe* erfasst und gespeichert.

6. Daten organisieren

Daten sind das wichtigste Gut der Datenwissenschaften. Deshalb sollte sorgsam und systematisch mit ihnen umgegangen werden. Zu diesem Zweck müssen Daten organisiert werden.

Das Ziel der Datenorganisation ist es, Daten so zu strukturieren, damit sie einfach und effizient verarbeitet werden können. Zur Datenorganisation gehört die Strukturierung, Ablage, Versionierung und Bereitstellung von Daten.

6.1. Daten strukturieren

Im Kapitel 5 wurde die Bedeutung des Datenschemas für das Sammeln von Daten behandelt. Ein Schema gibt vor, welche Werte einen Datensatz bilden. Dieser Abschnitt konzentriert sich auf die Umsetzung des Datenschemas.

6.1.1. Daten als Tabellen

Daten werden oft als Tabellen präsentiert. Tabellen sind eine einfache und intuitive Form der *Datenorganisation*. Vorläufig lassen sich Tabellen wie in Definition 6.1 definieren. Diese Definition wird in Kapitel 8.3 erweitert, um verschiedene Datenstrukturen zu unterscheiden.

Definition 6.1. Tabellen formen eine Datenstruktur, die Zeilen und Spalten hat und in der alle Zeilen die gleiche Breite und alle Spalten die gleiche Länge haben.

Tabellen organisieren Daten in Zeilen und Spalten, wobei jede Zeile und jede Spalte eine Überschrift haben kann.

Definition 6.2. Die Überschriften einer Tabelle werden als *Namen*, bzw. *Spaltennamen* und *Zeilennamen* bezeichnet.

Eine bestimmte Spalte in einer bestimmten Zeile heisst *Tabellenzelle* oder schlicht *Zelle*. Eine Zelle enthält genau einen Wert.

Definition 6.1 legt fest, dass eine *Tabelle* keine Zellen haben kann, die sich über mehrere Zeilen oder Spalten erstrecken. Viele Publikationen zeigen jedoch Datenstrukturen mit solchen Zellen. Solche Datenstrukturen sind keine Tabellen im Sinne von Definition 6.1.

Definition 6.3. Eine tabellenartige Struktur mit Tabellenzellen, die sich über mehr als eine Zeile oder eine Spalte erstrecken heisst **tabellarische Darstellung**.

Tabellarische Darstellungen dienen der *Präsentation* von Daten und Ergebnissen. Sie sind nicht für die *Datenorganisation* geeignet.

> **Tipp**
>
> In Berichten und Publikationen wird *nicht* zwischen *Tabellen* und *tabellarischen Darstellungen* unterschieden. Beide Strukturen werden unter *Tabellen* zusammengefasst. Beide Strukturen einheitlich als *Tabelle* beschriftet und nummeriert. **Diese Vereinfachung ist nur für die *Präsentation* von Daten erlaubt.**

6.1.2. Begriffe

Tabellen dienen zur systematischen Erfassung von Daten. Meist repräsentieren Tabellen Messungen, die später ausgewertet und analysiert werden sollen. Die Datenorganisation in Tabellen strukturiert Daten entlang zwei Dimensionen. Für die Datenerfassung werden die Daten in *Zeilen* und *Spalten* organisiert, wobei die Spalten meistens als *Merkmale* und die Zeilen als *Messereignisse* bezeichnet werden. Gemeinsam bilden die Merkmale und Messereignisse eine *Stichprobe* (engl. *sample*).

Definition 6.4. Ein **Merkmal** ist eine Eigenschaft, die in einer Messung durch einen *Messwert* erfasst wird. In der Statistik werden Merkmale in der *Statistik* als **Variablen** bezeichnet.

Ein Merkmal ist immer als Vektor organisiert. Alle Werte eines Merkmals haben also den gleichen Datentyp und den gleichen Wertebereich.

Definition 6.5. Ein **Messereignis** fasst ein oder mehrere *gemeinsam gemessenen* Merkmale zusammen. Ein Messereignis wird auch als **Datensatz** bezeichnet. Bei der Datenvisualisierung werden Messereignisse als **Datenpunkte** bezeichnet.

Ein Datensatz ist immer eine Liste von Werten, deren Datentypen und Wertebereichen voneinander verschieden sein können.

> **Hinweis**
>
> Der Begriff *Datensatz* wird im Deutschen mehrdeutig verwendet. Es ist nicht immer klar, ob ein Datensatz ein Zeile in einer Tabelle oder eine ganze Tabelle bezeichnet. Im Englishen wird der Begriff *Datensatz* entweder mit *data record* (etwa Dateneintrag) für eine Zeile in einer Tabelle benutzt. Die Begriffe *data set* (Datenmenge) oder *data frame* (Datenraster) bezeichnen eine ganze Tabelle.

Die *Merkmale* beschreiben gemeinsam ein *Messereignis*. Weil es sich bei den Merkmalen eines Messereignisses um *zusammengehörende* Werte handelt, wird auch der Begriff **Entität** (gegebene Einheit; eindeutig identifizierbare, zusammenhängende Größe) verwendet.

Messungen bilden einen Ausschnitt einer **Grundgesamtheit** ab. Eine Grundgesamtheit umfasst alle *prinziell* messbaren Entitäten. Eine Stichprobe ist eine *Teilmenge* der Grundgesamtheit, die nur die *tatsächlich gemessenen* Entitäten enthält.

Definition 6.6. Alle prinzipiell messbaren Entitäten bilden eine **Grundgesamtheit**.

Definition 6.7. Eine **Stichprobe** ist die Gesamtheit der gemessenen Entitäten ab.

Die **Statistik** befasst sich mit den Methoden, um von Stichproben auf die ursprüngliche **Grundgesamtheit** zu schliessen. Aus Sicht der Datenwissenschaft und Datenverarbeitung ist die Grundgesamtheit *unerheblich*, weil die nicht gemessenen Entitäten nicht in den Daten nicht abgebildet sind und deshalb unbekannt bleiben müssen!

6.1.3. Daten normalisieren

Ein Datensatz kann Werte aus einer oder mehreren Messungen beinhalten. Wenn mehrere *unabhängige* Messungen in einem Datensatz zusammengefasst werden, dann beschreiben die entsprechenden Vektoren die *gleichen Merkmale* aus unterschiedliche Messereignissen.

Definition 6.8. Unabhängige Messungen sind Messungen für die gleichen Merkmale zu unterschiedlichen Zeitpunkten.

Eine Stichprobe mit unabhängigen Messungen für die gleiche Entität ist *nicht normalisiert*.

Abbildung 6.1 zeigt einen Ausschnitt einer Stichprobe mit Werten aus mehreren unabhängigen Messungen bzw. Messereignissen. Jeder Vektor in dieser Stichprobe entspricht dabei unabhängigen Messungen.

Ein **Messereignis** bezeichnet das gleichzeitige Erheben zusammengehörender Daten. Wenn beispielsweise das gleiche Objekt zu unterschiedlichen Zeitpunkten gemessen wird, dann liegen *unabhängige Messereignisse* vor.

Wenn Daten in einer Tabelle jeweils ein Messereignis repräsentieren, dann liegen sie in **Normalform** vor. In der Normalform ist die Tabelle gleichbedeutend mit der zugehörigen Stichprobe. Jede Zeile ist der Datensatz einer beobachteten Entität. Die Spalten repräsentieren die Messungen.

> **i** Hinweis
>
> Beachten Sie, dass *unabhängige Messereignisse* nicht mit *unabhängigen Variablen* in der Statistik verwechselt werden darf.

Spezies \ Stichprobe	1145	1357	1838	2819	3480	3778	3965	4152	4728	5725	5811	6867
Abax parallelepipedus	0	0	0	277	0	0	0	0	68	0	81	65
Analgidae sp.	0	0	0	0	0	0	0	0	0	0	0	0
Carabus auronitens	0	0	0	0	0	22	0	0	286	0	0	61
Carabus nemoralis	16	0	0	0	0	18	0	0	0	0	0	0
Clitellata sp.	0	0	0	0	0	0	0	0	0	0	0	0
Melanostoma sp.	41	19	45	0	0	0	0	9	0	0	213	67
Myotis myotis	372	301	193	395	0	524	1590	1573	1137	1108	574	2489
Neostasina sp.	0	0	0	0	0	0	0	0	0	0	0	0
Nitophyllum punctatum	0	0	0	0	0	0	0	0	0	0	0	0
Zophophilus curticornis	0	0	0	0	2	0	0	0	0	0	0	0

Abbildung 6.1.: Beispiel einer Stichprobe mit mehreren Messereignissen pro Datensatz

Definition 6.9. Die **Normalform** einer Stichprobe ist eine Tabelle, in der jedes Merkmal genau einmal vorkommt und alle Werte in einem Datensatz zum gleichen Messereignis gehören.

Abbildung 6.2 zeigt einen Ausschnitt der Normalform der Stichprobe aus Abbildung 6.1. Die Normalform einer Stichprobe erscheint auf dem ersten Blick unübersichtlicher als Varianten mit mehreren Messereignissen pro Datensatz. Die Normalform hat jedoch den Vorteil, dass die Stichprobe *einfacher* verarbeitet werden kann.

Spezies	Probe	Sequenzen
Zophophilus curticornis	19015	0
Zophophilus curticornis	43889	0
Zophophilus curticornis	69954	0
Zophophilus curticornis	31384	0
Zophophilus curticornis	43382	0
Zophophilus curticornis	59103	0
Zophophilus curticornis	35533	0
Zophophilus curticornis	1145	0
Zophophilus curticornis	19100	0
Zophophilus curticornis	4152	0
Zophophilus curticornis	5725	0
Zophophilus curticornis	3965	0
Zophophilus curticornis	2819	0

Abbildung 6.2.: Beispiel einer Stichprobe in der Normalform

Praxis

Weil die Normalform oft schwer lesbar ist, werden Daten für die *Präsentation* in *nicht-normalisierter* Form bereitgestellt. Die Grundlage für diese Darstellung der Daten sollte möglichst die Normalform sein.
Beim Datensammeln sollte möglichst die Normalform eingehalten werden.

Beispiel 6.1 (Raumtemperatur und Helligkeit)**.** Die am Montag um 13 Uhr gemessene Temperatur und Helligkeit gehören zum gleichen Messereignis. Wird die Messung am Dienstag um 8 Uhr wiederholt, dann gehört die Temperatur und Helligkeit ebenfalls zum gleichen Messereignis.

Die Normalform dieser Daten wäre entsprechend:

Tag	Uhrzeit	Temperatur	Helligkeit
Montag	13	21.5	0.9
Dienstag	8	22.1	0.5

Die Temperatur am Montag und 13 Uhr und am Dienstag um 8 Uhr sind *unabhängige* Messereignisse für die Temperatur. Werden die Werte nicht-normalisiert gegenübergestellt, können Menschen die Werte oft leichter vergleichen. Eine solche Tabelle sähe dann wie folgt aus:

Temperatur Montag, 13h	Temperatur Dienstag, 8h	Helligkeit Montag, 13h	Helligkeit Dienstag, 8h
21.5	22.1	0.9	0.5

Diese zweite Tabelle macht deutlich, dass viele nicht normalisierte Tabellen einen Teil der Daten in Vektornamen kodieren. Diese Daten sind für die Datenverarbeitung nur indirekt zugänglich. Deshalb sollten Daten möglichst in der Normalform vorliegen.

6.2. Daten ablegen

6.2.1. Dateien und Verzeichnisse

Datendateien sollten möglichst isoliert und vor versehentlichen überschreiben geschützt werden.

Daten werden am Besten immer in eigenen Dateien abgelegt. Diese Dateien enthalten nur Werte und keine Umformungen oder Berechnungen. Wenn ein Projekt mehrere Datenerhebungen umfasst, dann sollten die Daten für jede Erhebung in einer eigenen Datei abgelegt werden. Hierzu sollte eine eindeutige Bezeichnung verwendet werden, die auf die

Erhebung hinweist. Ein geeignetes Format für Dateinamen ist ein Datum-Kennungs-Format, die die Erhebung eindeutig identifiziert. Dabei wird das Datum in der Form `YYYY-MM-DD` angegeben, gefolgt von einer kurzen Bezeichnung der Erhebung.

Beispiel 6.2 (Dateiname im Datum-Kennungsformat)**.**

```
2020-10-01-erhebung-1.csv
```

Durch die inverse Datumsschreibweise dieses Formats lassen sich die Dateien leicht nach Datum sortieren und schneller wiederfinden.

Damit die Daten von anderen Teilen eines Projekts getrennt werden können, sollten die Daten in einem eigenen Verzeichnis abgelegt werden. Dabei sollte das Verzeichnis einen Namen haben, der anzeigt, dass nur Datendateien in diesem Verzeichnis abgelegt werden. Eine solche Bezeichnung könnte beispielsweise `data` oder `daten` sein.

6.2.2. Daten-Repositories

Eine deutlichere Trennung der Daten von den Ergebnissen ist mit Hilfe von eigenen *Daten-Repositories* möglich. Dabei werden die Daten separat von den Ergebnissen gespeichert, versioniert und synchronisiert.

Ein Daten-Repository sollte immer nur die Daten eines Projekts enthalten. Das Repository sollte einen Namen haben, der anzeigt, dass es sich um ein Daten-Repository handelt.

Bei der Verwendung eines getrennten Daten-Repositories entfällt die Notwendigkeit, die Daten in einem eigenen Verzeichnis abzulegen. Diese Funktion übernimmt das Repository.

6.2.3. Datenbanken

Datenbanken sind eine weitere Möglichkeit, Daten zu abzulegen. Datenbanken werden über spezielle Software verwaltet. Sie eigenen sich besonders für die Verwaltung von grossen und kontinuierlich wachsenden Datenmengen, die von mehreren Personen bearbeitet werden und die über eine längere Zeit verfügbar sein müssen.

Datenbanken haben den Vorteil, dass gezielt Teile der Daten für spezielle Analysen geladen werden können. Dadurch können auch sehr grosse Datenmengen effizient verarbeitet werden.

6.2.4. Datenverlust vermeiden

Allein die Organisation von Daten in Tabellen, die in den richtigen Dateien in einem eigenen Verzeichnis abgelegt sind, ist keine Garantie, dass die Daten *sicher* sind. Daten können durch *versehentliches Überschreiben* oder *Löschen* verloren gehen.

Daten können beispielsweise durch *Hardware-Fehler* oder einen anderen Verlust der Hardware verloren gehen. Deshalb sollten Daten gesichert werden.

Die einfachste Form der Sicherung sind getrennte Speicherorte. Dazu werden die Daten an mindestens zwei Orten gespeichert. Diese Art der Sicherung heisst **Datenreplikation**. Wenn die Daten an einem Ort verloren gehen, können sie über den anderen Ort wiederhergestellt werden. Dazu müssen zwei Bedingungen erfüllt sein:

1. Die Daten müssen in beiden Daten *vollständig* vorhanden sein.
2. Die Speicherorte müssen *unabhängig* voneinander sein.

Zum Beispiel ist eine externe Festplatte für Sicherungen eines Laptops kein geeigneter Speicherort, wenn beide Geräte im gleichen Rucksack transportiert werden. Wenn der Rucksack verloren geht oder gestohlen wird, wären die Daten auf der externen Festplatte ebenfalls verloren.

Die **Versionierung** der Daten ein wichtiges Instrument zur Vermeidung von Datenverlusten. Moderne Versionierungssysteme arbeiten dabei zweistufig (s. Kapitel 7):

1. Die Daten werden lokal *versioniert*, so dass die Daten aus den Versionierungspunkten wiederhergestellt werden können.
2. Die Versionen werden in der verteilten Versionsverwaltung *synchronisiert*. Dadurch werden die Daten automatisch *repliziert*.

Durch diese Vorgehensweise sind die Daten an mindestens zwei Orten gespeichert. Wenn die Daten an einem Ort verloren gehen, können sie über den anderen Ort wiederhergestellt werden.

Die dritte Technik zur Vermeidung von Datenverlusten ist die **Archivierung**. Für die Datenarchivierung werden die Daten gebündelt und in einem *Archiv* abgelegt. Die Archivierung erfolgt meist nach Abschluss eines Projekts oder einer Studie. Alternativ sollten Daten bereits archiviert werden, sobald die Datenerhebung abgeschlossen wurde.

> **Praxis**
>
> Versionierungssysteme wie git können aus Versionen von Daten *automatisch* Archivdateien erzeugen. git Hosting-Plattformen bieten hierfür eigene Ablagen. Die Archivdateien können anschliessend an einem sicheren Ort abgelegt werden.

⚠ Datenbanken sind kein Ersatz für Sicherungen

Obwohl Datenbanken eine sehr gute Möglichkeit sind, Daten zu verwalten, sind sie kein Ersatz für Sicherungen. Datenbanken können zwar Daten *speichern*, aber sie können keine Daten *sichern*.
Eine Datenbank kann erst dann einen Datenverlust vorbeugen, wenn die Datenbank selbst gesichert wird. Das bedeutet, dass die Datenbank zumindest repliziert werden muss.

6.3. Datenmanipulation

Nach dem Importieren und vor dem Exportieren muss die *Datenintegrität* sichergestellt werden. Das bedeutet, dass zwischen allen Werten und Ergebnissen eine systematische Verbindung besteht. Werte dürfen deshalb nicht *willkürlich* verändert, gelöscht oder hinzugefügt werden.

Ein grosses Problem bei der Arbeit mit Daten ist die *nachträgliche Datenmanipulation*. Dabei werden die erhobenen Daten verändert, wobei sich die Veränderung nicht einwandfrei reproduzieren lässt.

Definition 6.10. Eine **Datenmanipulation** heisst jede Veränderung von Daten bei denen Werte unsystematisch hinzugefügt, beliebige Werte verändert oder gelöscht werden.

Weil nach einer Datenmanipulation die ursprünglichen Daten nicht mehr eindeutig reproduziert werden können, lassen sich die Ergebnisse nicht mehr bestätigen oder wiederlegen. Das kommt einem vollständigen Datenverlust gleich, denn alle vorliegenden Daten können manipuliert worden sein.

❗ Keine Datenreproduktion möglich

Wird eine Datenmanipulation entdeckt oder ist eine Manipulation sehr wahrscheinlich, dann ist es **nicht mehr möglich**, die ursprünglichen Daten zu reproduzieren. In diesem Fall gelten **alle Daten** eines Datensatzes als **komprimitiert**. Solche Daten dürfen auf keinen Fall für Analysen weiterverwendet werden.

Eine Datenmanipulation muss von der systematischen Datenverarbeitung abgegrenzt werden. Bei der systematischen Datenverarbeitung werden Daten mit Hilfe von Werkzeugen und definierten Methoden verarbeitet. Dabei gehen keine Daten verloren, werden neue Werte erzeugt oder bestehende Werte verändert. Bei der systematischen Datenverarbeitung lassen sich alle Ergebnisse aus den ursprünglichen Daten herleiten.

Falls Werte erzeugt werden, dürfen diese nicht willkürlich mit den ursprünglichen Daten vermischt werden. Stattdessen sind solche neuen Werte von den Daten zu trennen.

Praxis

Generierte Werte dürfen nur über eine Kodierung (Kapitel 16) mit den ursprünglichen Daten verknüpft werden. Die Kodierung muss so gestaltet sein, dass die ursprünglichen Daten jederzeit reproduziert werden können.

Merke

Um eine systematische Datenverarbeitung belegen zu können, müssen **Daten**, **Ergebnisse** und **alle Operationen**, die von den Daten zu den Ergebnissen, vorgehalten werden.

Keine Datenmanipulation

Solange alle Werte im Rahmen des Schemas erhalten bleiben, liegt keine Datenmanipulation vor. Dazu gehören insbesondere:

- Nachvollziehbare logische, methodische und systematische Fehler.
- Korrektur von Vektornamen/Variablen und deren Übersetzung in eine andere Sprache.
- Begründetes Runden, wenn dadurch keine Datenverzerrung entsteht. Als Richtlinie gilt die Messgenauigkeit der verwendeten Instrumente.
- Umbenennen von Dateien und Verzeichnissen.
- Umwandlung von Masseinheiten, z.B. Konvertierung von Grad Fahrenheit in Grad Celsius.
- Anonymisierung von Datensätzen.

Es gibt zwei Arten der Datenmanipulation.

Die **absichtliche Manipulation** von Daten erfordert *aktives Eingreifen* einer Person mit Zugang zu den Daten. ist ein schweres professionelles Delikt. Diese Art von Manipulation kann das Löschen von unpassenden Daten sein, das Verändern von Werten betreffen bzw. Werte für die Studienergebnisse *passend machen* oder auch neue Daten *erfinden*. Solche Datenmanipulationen wiegen wissenschaftlich oft schwerer als Plagiarismus und können bei professionellen Analysen als Urkundenfälschung gewertet werden.

Die **versehentliche Datenmanipulation** kommt sehr viel häufiger vor als eine absichtliche Datenmanipulation. Die versehentliche Datenmanipulation kann viele Ursachen haben und reicht von der fehlerhaften Bedienung von Tools und Werkzeugen über Software-Fehler bis zum Überschreiben von Daten mit Ergebnissen.

Eine sehr häufig vorkommende Form der versehentlichen Datenmanipulation ist die automatische Datentyperkennung von Excel. Dabei werden Daten beim Import oder bei der Eingabe in einer Excel-Arbeitsmappe in einen *unerwünschten* Datentyp geändert, wobei sich auch der Wert der Daten ändert. Wenn durchgehend mit Excel gearbeitet wird, müssen diese Fehler vorgebeugt werden. Diese Praktiken erfordern von allen Beteiligten eine grosse Arbeitsdisziplin, was sich im Alltag nicht immer fordern lässt. Deshalb ist es oft

einfacher, die Daten in einem einfacheren Dateiformat (z.B. CSV oder JSON) zu erfassen und anschliessend diese Daten in Excel als externe Daten zu importieren.

Praxis

Um versehentliche Datenmanipulation zu vermeiden, sollte ausser bei Messungen **nie** direkt mit den Daten gearbeitet werden. Stattdessen sollten die Daten **immer** in einer separaten Datei gespeichert und versioniert werden. Aus dieser Datei werden die Daten anschliessend *importiert* und *bearbeitet*. Die Ergebnisse können in *andere* Dateien *exportiert* werden.
Durch die Versionierung kann jederzeit auf die ursprünglichen Daten zurückgegriffen und wiederhergestellt werden, selbst wenn sie versehentlich oder absichtlich überschrieben oder gelöscht wurden.

7. Versionierung mit Git und GitHub

Definition 7.1. Versionierung ist ein Prozess, bei dem Änderungen an einem Dokument oder einer Datei nachverfolgt werden.

Die meisten Cloud-Dienste bieten inzwischen das Speichern von Dateiversionen an. Diese Funktionen sind in der Regel so gelöst, dass die Datei in regelmässigen Abständen gespeichert wird, so dass eine *zeitliche* Abfolge von geänderten Versionen entsteht. Für manche Dateiformate ist es möglich, einzelne Änderungen des Inhalts nachzuvollziehen.

> **i** Hinweis
>
> Für die Arbeit mit Daten ist das Versionsmanagement wichtig, damit das Datenmanagement durchgängig und nachvollziehbar ist.

Das regelmässige Speichern von Dateien ist der Startpunkt für die eigentliche Versionierung, dem *Versionsmanagement* oder der *Versionskontrolle* (engl. Version Control).

Bei der Versionierung werden die Änderungen an einer Datei festgehalten und mit einer Versions*nummer* gespeichert. Der entscheidende Unterschied zum regelmässigen Speichern auf Cloud-Diensten ist, dass einzelne Versionen für den späteren Gebrauch oder zur späteren Kontrolle markiert und wiederhergestellt werden können. Die Versionskontrolle stellt sicher, dass für jede Datei die jeweils aktuelle Version eindeutig definiert ist.

Die Versionierung betrifft alle erzeugenden Projektdateien. Dazu gehören:

- Datendateien
- Code-Dateien
- Code-Konfiguration
- Projektkonfiguration
- Dokumentation

Es gibt verschiedene Systeme für die Versionierung. Im Bereich der Software-Entwickung und den Datenwissenschaften hat sich `git` als Industriestandard etabliert.

`git` ist ein verteiltes Versionierungssystem. *Verteilt* bedeutet dabei, dass die Versionierung mit allen teilnehmenden Systemen vollständig geteilt wird und es keine zentrale Instanz für die Versionierung gibt. Es handelt sich daher um ein redundantes *Peer-to-Peer*-System. `git` unterteilt die Versionierung in Projekte, den sog. ***Repositories***, dass auf verschiedenen Computern repliziert bzw. ge-**cloned** wird. Dadurch erhält jeder Computer mit einem Repository-Clone eine vollständige Kopie der Versionierung eines Projekts.

Diese Kurzeinführung stellt die zentralen `git`-Konzepte für die tägliche Arbeit vor:

- `git` installieren
- Repositories, Clone und Forks
- Dateiversionen einem Repository hinzufügen (commit)
- Pull und Push
- Branches, Merges und Pull-Requests

7.1. `git` installieren

`git` besteht im wesentlichen aus einem Kommandozeilen-Werkzeug, das alle Funktionen von `git` bereitstellt. Dieses Werkzeug wird oft auch als *CLI-Tool* bezeichnet. Es für die meisten GUI-Apps die Voraussetzung für die Arbeit. Ausserdem stellen nicht alle GUI-Apps alle `git`-Funktionen bereit und gelegentlich kann es in Spezialfällen notwendig sein, `git` direkt zu verwenden. Deshalb sollte `git` immer auf einem Computer installiert sein.

7.1.1. `git` unter MacOS installieren

Es wird dringend empfohlen die offiziell von Apple vertriebene Version von `git` zu verwenden.

Unter MacOS ist `git` Teil der internen Entwicklungsumgebung. Dazu sollte XCode installiert sein. XCode muss mindestens einmal gestartet worden sein, damit die Lizenzbedingungen erstmalig akzeptiert werden. Anschliessend müssen die XCode Command Line Tools dem System hinzugefügt werden. Wählen Sie die Version dieser Tools immer so aus, dass sie zu der von Ihnen installierten Version von XCode passt. Für diesen Download benötigen Sie einen Entwicker-Account auf Apple's Entwicklerplattform. Dieser Account ist kostenlos. Nach der Installation der XCode Command Line Tools ist auch `git` auf Ihrem Rechner installiert.

7.1.2. `git` unter Windows installieren

Unter Windows stellt das Projekt git for Windows eine für Windows Systeme angepasste Version von `git` bereit. Diese Version bietet neben `git` zusätzliche Werkzeuge, mit denen sich Anleitungen leichter nachvollziehen lassen.

7.1.3. Grafische Oberflächen für `git`

GUI-Apps die Arbeit mit `git`. Für die Arbeit mit diesem Buch werden die folgenden Werkzeuge empfohlen.

- Visual Studio Code unterstützt `git` in der Basisinstallation.
- GitLens ist eine Erweiterung für Visual Studio Code und bindet zusätzliche Funktionen für das Versionsmangement direkt in die IDE ein.

- GitHub Desktop stellt eine einfache Benutzeroberfläche für die Basisaufgaben mit GitHub am Computer Desktop bereit.
- GitKraken bietet unfangreiche Funktionen zum Versionsmanagement mit `git`. Mit GitKraken lassen sich fast alle `git`-Aufgaben über die GUI lösen. Ausserdem ist GitKraken nicht auf GitHub als Hosting-Plattform beschränkt.

7.1.4. `git`-Hosting-Plattformen

`git` ist zuerst auf den eigenen Computer beschränkt. Für das Zusammenarbeiten mit anderen und die Sicherung der eigenen Versionierung empfielt sich die Verwendung einer sog. `git`-Hosting-Plattform. Eine `git`-Hosting-Plattform erlaubt es die Versionierung eines Projekts online zu sichern und mit anderen zu teilen, die bietet Projektmanagementfunktionen sowie Automatisierungsmöglichkeiten, die über die Kernfunktionen von `git` hinausgehen.

Die wichtigsten `git`-Hosting-Plattformen mit gleichwertigem Funktionsumfang sind:

- GitHub
- GitLab
- Gitea

Jede dieser Plattformen benötigt ein eigenes Benutzerkonto.

> **Hinweis**
>
> Dieses Buch verwendet ***GitHub*** für Code-Beispiele, Diskussionen und andere Funktionen. ***GitLab*** und ***Gittea*** sind Alternativen zu GitHub mit gleichwertigen Funktionsumfang und ähneln sich bei der Bedienung und im Funktionsumfang stark. Aus Sicht der Projektorganisation sind die Plattformen fast identisch. Es gibt vor Allem emotionale und wirtschaftliche Gründe, sich für die eine oder die andere Plattform zu entscheiden.

7.2. `git`-Konzepte

7.2.1. Repositories, Clones und Forks

Die Grundlage der Versionierung mit `git` ist das Repository. Ein Repository enthält die Dateiversionen eines Projekts. Aus den Versionen kann das Projekt auf den Zustand zu jedem Versionierungszeitpunkt wiederhergestellt werden.

Ein Repository enthält alle versionierten Dateien und deren Versionen. Auf dem eigenen Computer sieht ein `git`-Repository ersteinmal wie ein normales Verzeichnis aus. Die Versionen sind in einer versteckten Versionsdatenbank gespeichert, über die mit dem `git`-Kommando oder einem entsprechenden GUI-Tool interagiert wird.

Um aus einem normalen Verzeichnis ein `git`-Repository zu erstellen, muss das Verzeichnis für `git` *initianlisiert* werden. Im Terminal kann ein beliebiges Verzeichnis zu einem Repository initialisiert werden.

```
git init
```

`git`-Repositories können aus zwei Arten synchronisiert werden: Durch Clone und durch Forks. Jedes Kopie eines `git`-Repositories enthält immer alle Versionen des Projekts. Wird ein Repository auf einen anderen Computer synchronisiert, dann wird das entfernte Repository als **Remote** bezeichnet. Handelt es sich bei diesem Repository um ein Quell-Repository, dann wird dieses auch als **Upstream**-Repository bezeichnet. Die lokale Variante des Repositories wird auch als **Downstream**-Repository bezeichnet.

Zwei gleichwertige Kopien eines Repositories werden als **Clone** bezeichnet. Gleichwertig bedeutet, dass die Versionen zwischen diesen Repositories direkt synchronisierbar sind.

Neben gleichwertigen Repositories gibt es noch hierarchische Beziehungen zwischen Repositories. Dabei kann nur vom Upstream Remote direkt in die lokale Kopie sysnchronisiert werden. Änderungen im Downstream-Repository können nicht direkt in das Upstream Remote synchronisiert werden. Das Downstream Repository wird als **Fork** bezeichnet. Änderungen eines Forks können über sog. Pull-Requests an das Upstream-Repository gesendet werden.

7.2.2. Versionierungsstatus feststellen

Mit dem Kommando `git status` kann der aktuelle Änderungsstatus eines Repositories gestgestellt werden. In einem frisch initialisierten Repository liefert dieses Kommando die folgende Meldung.

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Wird einem leeren Repository eine neue Datei hinzugefügt, zeigt `git status` an, dass die neue Datei noch nicht versioniert wird. Als Beispiel wird die Datei `README.md` mit dem folgenden Inhalt erstellt.

```
# Mein erstes Projekt
```

Nach dem diese Datei gespeichert wurde, liefert `git status` die folgende Meldung.

```
On branch main
```

```
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

nothing added to commit but untracked files present (use "git add" to track)
```

7.2.3. Versionierung durch Commits

`git` muss mitgeteilt werden, dass eine neue, noch nicht verfolgte Datei in die Versionierung aufgenommen wird. Die Meldung von `git status` gibt den Hinweis, dass das Kommando dazu verwendet werden muss.

Im obigen Beispiel kann die neue mit `git add README.md` in die Versionierung aufgenommen werden.

Im Gegensatz zu Cloud-Speichern führt `git` keine automatische Synchronisation bei Änderungen einer Datei durch. Stattdessen müssen zu versionierende Dateien und Änderungen ihnen explizit versioniert werden. Auf den ersten Blick sieht das kompliziert aus, bietet im Entwicklungsprozess mehr Flexibilität und Kontrolle.

Die Versionierung erfolgt immer in zwei Schritten:

1. Auswahl der zuversionierenden Dateien mit `git add`.
2. Erstellen einer neuen Verion mit den Änderungen in den ausgewählten Dateien mit `git commit`.

Auf der Kommandozeile werden die beiden Schritte durch zwei Aufrufe von `git`:

```
# Alle geänderten und neuen Dateien für die Versionierung auswählen
git add .
# Eine neune Version für die ausgewählten Dateien erzeugen.
git commit
```

Falls nur verfolgte Dateien in einer Version aufgenommen werden sollen, dann können die beiden Schritte mit dem folgenden Kommando zusammengefasst werden.

```
git commit -a
```

Sollen neue, noch nicht versionierte Dateien in die Versionierung aufgenommen werden, dann müssen diese Dateien zwingend mit `git add` ausgewählt werden.

> GUI-Apps fassen die beiden Schritte automatisch zusammen.

Im obigen Beispiel wurde die Datei `README.md` bereits zur Versionierung markiert. Diese Datei soll nun versioniert werden. Das wird mit `git commit` erreicht. Wird `git commit` direkt aufgerufen, startet `git` den sog. Message-Editor, um die Meldung für diese Version einzugeben. `git` versucht einen Vorschlag für die Meldung vorzuschlagen. Diese Meldung kann übernommen werden, indem die Rautesymbole am Zeilenanfang gelöscht werden. Abschliessend muss die Meldung gespeichert und geschlossen werden, damit `git` die Meldung übernimmt.

`git` erfordert, dass alle Versionen eine Meldung haben, die kurz die vorgenommenen Änderungen beschreibt. Diese Versionsmeldung erlaubt es, die Entwicklung eines Projekts schrittweise nachzuvollziehen.

> **Hinweis:** Es ist zwar prinzipiell möglich auch leere Versionsmeldungen zu speichern, dadurch lässt sich die Versionsgeschichte nur schwer reproduzieren. Ausserdem wird die Auswahl einer bestimmten Version erschwert, weil aus der Versionsnummer nicht direkt hervorgeht, welche Änderungen vorgenommen wurden.

Das Kommando `git status` zeigt nun an, dass keine Änderungen gefunden wurden.

Nachdem eine Version erstellt wurde, kann die Versionsgeschichte mit `git log` angezeigt werden. Das Ergebnis sieht nach dem initialen Commit des obigen Beispiels sieht etwas folgendermassen aus:

```
commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (HEAD -> main)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 14:37:49 2023 +0200

    Initial commit
        new file:   README.md
```

Die kryptische Symbolfolge in der ersten Zeile hinter dem Wort `commit` ist die Versionsnummer. Die Versionsnummer ist eindeutig im Projekt und über die meisten Projekte hinweg. In Klammern stehen die beiden Worte `HEAD` und `main`. `HEAD` zeigt die Markierung der aktuellen Version. `main` zeigt das Ende des aktuellen Versionszweig mit dem Namen `main` an. Dieser Information folgt der Name der Person, welche die Version erzeugt hat und dem Datum und der Uhrzeit, was den Versionszeitpunkt markiert. Abschliessend folgt die Versionsmeldung.

Dieser Schritt kann nun mit einer Änderung an der Datei `README.md` wiederholt werden. Hierzu wird zusätzlicher Text `"Eine Übung zur Git Versionierung"` ans Ende der Datei hizugefügt. Der Inhalt der Datei sollte nun wie folgt aussehen:

```
# Mein erstes Projekt

Eine Übung zur Versionierung mit git.
```

Diese Änderungen werden nun mit `git commit -a` versioniert.

`git log` zeigt nun die folgende Information.

```
commit 1504d416bb1ef926fde4d5e949f3ed330d97b065 (HEAD -> main)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:27:19 2023 +0200

    Zusatzinformation zum Projekt
      modified:   README.md

commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Die beiden Markierungen `HEAD` und `main` wurden zur aktuellen Version verschoben.

7.2.4. Checkout

Eine Versionierung macht nur dann Sinn, wenn auf eine ältere Version zugegriffen werden kann. Das erlaubt das Kommando `git checkout`. Dieses Kommando benötigt eine Versionsnummer oder eine Markierung, um die entsprechede Version zuzugreifen.

So kann auf die erste Version im Projekt zugegriffen werden:

```
git checkout b281e4a03a4e71bcc41fa2f68e822b98c1555e1d
```

Dieses Kommando ersetzt alle Dateien im Arbeitsverzeichnis in den Zustand der angegebenen Version.

Der Inhalt der Datei `README.md` ist nun wieder wie folgt.

```
# Mein erstes Projekt
```

Der Befehlt `git log` zeigt nun die Versionsgeschichte bis zum aktuell vorliegenden Inhalt.

```
commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (HEAD)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Hier gilt es zu beachten, dass die Markiergung `main` nicht mehr angezeigt wird. Diese Markierung ist nicht verloren, liegt aber aus Sicht der aktuellen Version in der Zukunft und wird deshalb nicht dargestellt. Wenn diese Versionsnummer nicht notiert wurde, kann auf die "zukunftigen" Versionen nicht leicht zugegriffen werden. Damit wieder auf die letzte Version zugegriffen werden kann, kann die Markierung `main` verwendet werden.

```
git checkout main
```

Mit diesem Kommando werden alle Dateien auf den neusten Stand gebracht. `git log` zeigt nun wieder die vollständige Versionsgeschichte.

7.2.5. Tags

Die Navigation über die Versionsnummern ist wegen der kryptischen Versionsnummern nicht ganz einfach. Ausserdem haben nicht alle Versionen in einem `git`-Repository die gleiche Bedeutung für die Nutzenden. Um zwischen wichtigen Versionen leichter navigieren zu können dienen Tags.

Tags sind spezielle Markierungen von Versionen in einem `git`-Repository. Das besondere Tag `HEAD` markiert immer die aktuell vorliegende Version. Mit dem Kommando `git tag` können neue Markierungen angelegt werden, um die Navigation zwischen Versionen zu erleichtern oder wichtige Zeitpunkte zu markieren.

Ein Tag darf keine Leerzeichen enthalten und sollte kompakt sein. Das folgende Beispiel markiert die erste Version im Repository.

```
git tag erste_version b281e4a03a4e71bcc41fa2f68e822b98c1555e1d
```

Tags werden in der Versionsgeschichte angezeit. `git log` zeigt dann das folgende Ergebnis.

```
commit 1504d416bb1ef926fde4d5e949f3ed330d97b065 (HEAD -> main)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:27:19 2023 +0200

    Zusatzinformation zum Projekt
      modified:   README.md

commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (tag: erste_version)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Nun kann auf die erste Version direkt zugegriffen werden, ohne die kryptische Versionsnummer kennen zu müssen.

```
git checkout erste_version
```

Dieses Kommando springt nun auf die erste Version in diesem Repository. Das kann mit `git log` überprüft werden.

Das Kommando `git tag -l` listet alle verfügbaren Tags eines Repositories.

7.2.6. Branching und Merging

Die Markierung `main` ist kein Tag, sondern markiert einen Versionszweig. Diese Markierung wandert mit jedem Commit immer zur neusten Version. Es ist möglich mehrere Versionszweige im gleichen Repository zu führen. Ein Versionszweig wird als **Branch** bezeichnet.

Die Versionszweige eines Repositories lassen sich mit dem Kommando `git branch` anzeigen. Im Beispiel-Repository liefert dieses Kommando das folgende Ergebnis.

```
* main
```

Jedes `git`-Repository hat mindestens einen Versionszweig. Neuere Versionen von `git` verwenden `main` als Namen für diesen Haupt-Branch.

Versionszweige werden in der Praxis dafür verwendet, um komplexe Änderungen im Projekt vorzunehmen oder um zusammenzuarbeiten.

Um einen neuen Branch zu erstellen wird das Kommando `git branch -c` verwendet. Es fügt an der aktuell aktiven Version einen neuen Branch ein.

```
git checkout erste_version
git branch -c extras
```

Dieses Kommando erstellt am Tag `erste_version` einen neuen Branch mit dem Namen `extras`. `git branch` zeigt nun das folgende Ergebnis.

```
* (HEAD detached at erste_version)
  main
  extras
```

Um den `extras`-Branch zu aktivieren, muss dieser nur `git checkout extras` aufgerufen werden.

`git log` zeigt nun als Versionsgeschichte.

```
commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (HEAD -> extras, tag: erste_version)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Der Pfeil von `HEAD` nach `extras` zeigt an, dass neue Versionen zum `extras`-Branch und nicht mehr zum `main`-Branch hinzugefügt werden. So lassen sich Verzweigungen umsetzen. Das veranschaulicht das folgende Beispiel: Es wird eine neue Datei mit dem Namen `AUTHORS.md` und dem folgenden Inhalt.

```
Christian Glahn (ZHAW)
```

erstellt. Es handelt sich um eine neue Datei und deshalb muss sie mit `git add` zum Repository hinzugefügt und eine Version mit `git commit` erstellt werden.

```
git add AUTHORS.md
git commit -m "Autorenliste"
```

`git log` zeigt nun als Versionsgeschichte:

```
commit 5d10e17be5a2fc0d987db10644b35c73224f51f2 (HEAD -> extras)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 16:58:01 2023 +0200

    Autorenliste

commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (tag: erste_version)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Diese neue Datei existiert vorerst nur im `extras`-Branch. Diesem Branch können nun weitere Änderungen hinzugefügt werden.

Sind die Änderungen abgeschlossen, dann können diese Änderungen in den Hauptzweig übernommen werden. Dieses Zusammenführen von zwei Branches wird als **Merge** bezeichnet. Die folgenden Kommandos führen den Merge der Änderungen von `extras` in den `main`-Branch aus.

```
git checkout main
git merge -m "Extras übernehmen" extras
```

Dieses Zusammenführen bindet die Versionen aus dem `extras`-Branch in den `main`-Branch ein. Es gehen also keine Versionen verloren. `git log` zeigt das.

```
commit 4f33342df57ec5f60b2ec1daeb581b49836f05bf (HEAD -> main)
Merge: 1504d41 5d10e17
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 17:01:27 2023 +0200

    Extras übernehmen

commit 5d10e17be5a2fc0d987db10644b35c73224f51f2 (extras)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 16:58:01 2023 +0200

    Autorenliste

commit 1504d416bb1ef926fde4d5e949f3ed330d97b065
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:27:19 2023 +0200

    Zusatzinformation zum Projekt
      modified:   README.md

commit b281e4a03a4e71bcc41fa2f68e822b98c1555e1d (tag: erste_version)
Author: Christian Glahn <christian.glahn@zhaw.ch>
Date:   Fri Aug 18 15:26:36 2023 +0200

    Initial commit
        new file:   README.md
```

Nun existiert die Datei `AUTHORS.md` auch im `main`-Branch.

7.2.6.1. Merge-Konflikte lösen

Gelegentlich entstehen beim Mergen Konflikte. Ein Merge-Konflikt sind widersprüchliche Inhalte an der gleichen Code-Position. Solche Konflikte können auch entstehen, wenn mehrere Personen gleichgzeitig auf dem gleichen Branch arbeiten.

`git` löst solche Widersprüche nicht selbst, sondern erfordert ein aktives Eingreifen der Entwickler:innen. Um diesen Schritt zu erleichtern, fügt git Markierungen für die Konflikte ein. Diese Markierungen sind in den aktiven Daten zu finden, aber sind keine eigenständigen Versionen im Repository.

Ein solcher Konflikt kann dadurch provoziert werden, indem im `extras`-Branch der Datei `README.md` am Ende zusätzlicher Text hinzugefügt wird, so dass die Datei den folgenden Inhalt hat.

```
# Mein erstes Projekt

Alle Informationen zu den Autoren finden sich in der Datei [AUTHORS.md]
```

Wird nun versucht den `extras` und den `main`-Branch zusammenzuführen, meldet `git merge` den folgenden Konflikt:

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Der Inhalt der Datei `README.md` zeigt nun den Merge-Konflikt.

```
# Mein erstes Projekt

<<<<<<< HEAD
Eine Übung zur Versionierung mit git.
=======
Alle Informationen zu den Autoren finden sich in der Datei [AUTHORS.md].
>>>>>>> extras
```

Durch die Gleichheitszeichen sind der aktuelle Branch (oben) und der einzubindende Branch (unten) getrennt. Jeweils am Anfang und Ende des Konfliktbereichs stehen die Namen des betreffenden Branches.

Es bleibt den Entwickler:innen überlassen, sich für eine, beide oder keine der Änderungen zu entscheiden. In diesem Beispiel sollen beide Inhalte erhalten bleiben. Dazu werden die `git`-Markierungen aus der Datei entfernt und durch Zeilenumbrüche ersetzt. Der Dateiinhalt ist nun:

```
# Mein erstes Projekt

Eine Übung zur Versionierung mit git.

Alle Informationen zu den Autoren finden sich in der Datei [AUTHORS.md].
```

Die Datei ist nun konfliktfrei und kann in die Versionierung aufgenommen werden.

7.2.6.2. Rebase

Normalerweise erfolgt das Auflösen von Merge-Konflikten durch die Verantwortlichen des Hauptzweigs. Das ist aus Sicht des Projektmanagements ungünstig. Besser wäre es, wenn nur

konfliktfreie Änderungen übernommen werden müssten. `git` bietet hierzu das Kommando `git rebase` an. Das Kommando zieht alle Änderungen des Zielzweigs in den aktuellen Arbeitszweig.

```
git checkout extras
git rebase main
```

Dadurch wird der gleiche Merge-Konflikt wie beim regulären Merge ausgelöst. Dieser Konflikt kann nun durch die Person gelöst werden, die für den Arbeits-Branch verantwortlich ist.

Anschliessend können beide Branches konfliktfrei zusammengeführt werden.

> **Praxis**
>
> Es hat sich als gute Praxis etabliert, Merge-Konflikte immer mit `git rebase` und nicht mit `git merge` zu lösen.

7.2.7. Fetch, Pull und Push

`git` synchronisiert Repositories nicht automatisch. Damit Änderungen in einem Clone oder Fork im eigenen Repository übernommen werden, müssen die Repositories *synchronisiert* werden.

Das Kommando `git fetch` synchronisiert die Versionen der Remotes mit dem lokalem Repository. Falls ein Repository mehrere Remotes registriert hat, können alle Änderungen in allen Remotes mit `git fetch --all` auf einmal abgeglichen werden.

Nach `git fetch` enthält das lokale Repository alle Versionen der Remotes. Diese Versionen werden allerdings nicht automatich in die lokalen Versionszweige übernommen. Die so synchornisierten Änderungen existieren als eigene Branches im Repository. Im Gegensatz zum regulären Merge, werden diese "Remote-Branches" mit dem Kommando `git pull` in die lokalen Versionen übernommen. `git pull` löst alle Unterschiede für den aktuellen Branch im lokalen und remote Repository auf.

Weil intern ein Merge durchgeführt wird, kann auch `git pull` zu Merge-Konflikten führen. Diese Konflikte müssen gelöst werden, bevor weitergearbeitet werden kann.

> **Hinweis**
>
> `git pull` führt automatisch ein `git fetch` aus, falls keine Änderungen für ein Remote-Repository lokal gefunden wurden.

> **Praxis**
>
> Sind für ein Repository mehrere Remotes konfiguriert, dann sollten die einzelnen Remotes einzeln synchronisiert werden.

Das Kommando `git push` wird zum Synchronisieren des lokalen Repositories mit den Remotes genutzt. Beim `git push` wird im Remote ein Merge durchgeführt. Führt dieser zu Konflikten, meldet `git push` einen Fehler, jedoch *keinen* Merge-Konflikt. Bei Fehlern von `git push` muss zuerst `git fetch` und anschliessend ein `git rebase` ausgeführt werden und eventuelle Merge-Konflikte lokal gelöst werden.

`git push` kann nur für *Clones* verwendet werden. *Forks* können normalerweise nicht direkt in das Upstream-Remote synchronisieren. Sollen Änderungen für ein Upstream Repository vorgeschlagen werden, dann muss ein sogenannter *Pull-Request* ausgelöst werden. In diesem Fall werden die Änderungen durch ein `git pull` im Upstream-Repository übernommen.

7.3. `git`-Projektmanagement

Die im folgenden beschrieben Funktionen sind kein Teil von `git`, sondern finden sich als Erweiterungen in `git`-Hosting-Plattformen. Diese Funktionen wurden für den Einstieg in die Verwendung von `git`-Hosting-Plattformen ausgewählt und zeigen nur einen Bruchteil der Managementfunktionen moderner `git`-Hosting-Plattformen.

7.3.1. Issues

`git`-Hosting-Plattformen führen neben dem eigentlichen Repository eine Liste mit *Issues*. **Issues** bezeichnen traditionell ein *Problem*, *Fragestellung* oder *Angelegenheit* im oder mit dem Projekt. Das können z.B. noch nicht realisierte Projektziele, fehlende Dokumentation oder auch Sicherheitsproblemen durch Programmierfehler sein. Damit *Issues* nicht verloren und vergessen werden, sollten sie dokumentiert werden. Dazu dient der Issues-Bereich in `git`-Hosting-Plattformen.

Die Issues eines Projekts sind nummeriert, so dass über diese Nummer auf das jeweilige Issue referenziert werden kann. Ein Issue besteht immer aus einem Titel, welcher die notwendige Änderung kurz beschreibt. Zusätzlich verfügen Issues über eine Beschreibung, in der detailliert beschrieben werden kann, was das Problem ist und welche Anforderungen erfüllt sein müssen, damit ein Issue erfüllt ist.

Die *Issues* von `git`-Hosting-Plattformen haben zwei Zustände: *Offen* (engl. open) und *geschlossen* (engl. closed). Offene Issues sind noch nicht bearbeitet. Geschlossene Issues sind bearbeitet und erledigt. Issues können deshalb als To-Do-Liste für ein Projekt verwendet werden: Jedes Issue lässt sich mit Änderungen im Repository verbinden. Weil jede Änderung mit einer Version gleichzusetzen ist, kann ein Issue geschlossen werden, wenn im Repository ein Commit existiert, das die notwendige Änderung enthält.

`git`-Hosting-Plattformen unterstützen sog. *Action-Keywords* in Commit-Meldungen. Wird ein solches Keyword zusammen mit der Issue-Nummer verwendet, dann schliesst die Hosting-Plattform das genannte Issue automatisch. Gleichzeitig wird dokumentiert, mit welchem Commit das Issue geschlossen wurde.

Die Möglichkeit ein Issue zu kommentieren ist weitere sehr praktische Funktion. So können Notizen, Gedanken und Recherchen einem Issue zugeordnet werden. Diese Dokumentation wird automatisch mit einem Datum und dem Account-Namen versehen, so dass sich die Arbeit an einem Issue leicht reproduzieren lässt. Der Vorteil dieser Funktion ist, dass so Konzepte und Ideen einem Issue zugeordnet werden können, selbst wenn sich diese nicht im eigentlichen Repository wiederfinden. So kann die Entscheidungsfindung in einem Projekt nachvollziehbar dokumentiert werden.

7.3.2. Pull-Requests

Wenn Code in zwei Code-Zweigen zusammengeführt werden soll, werden sog. ***Pull-Requests*** bzw. ***Merge-Requests*** erstellt. Ein Pull-Request ist eine Anfrage zum Zusammenführen eines Branches oder eines Forks mit einem anderen Branch.

> **Hinweis**
>
> Ursprünglich wurden Pull-Requests in Form von E-Mails an die Projektkoordination geschickt und enthieleten einen Verweis auf den einzubindenen Fork. In `git`-Hosting-Plattformen sind Pull-Requests spezielle Issues, die einen Branch oder einen Fork mit dem Repository verknüpfen.

> **Praxis**
>
> Sobald ein Commit in einem Branch vorliegt, kann ein Pull-Request erstellt werden. Es ist eine gute Vorgehensweise, möglichst früh einen Pull-Request zu öffnen.

Wie Issues können Pull-Requests kommentiert werden, so dass notwendige Anpassungen angeregt werden können. Es kann auch die Qualität der Versionen in einem Branch überprüft werden. Entspricht ein Pull-Request nicht den Qualitätsanforderungen eines Projekts, können vor einem Merge die notwendigen Änderungen vorgenommen werden. Alle `git`-Hosting-Plattformen bieten die Möglichkeit für jeden Pull-Request Projektstandards automatisch zu überprüfen und verhindern das Zusammenführen bei Problemen.

Pull-Requests habe wie Issues den Zustand offen und geschlossen. Sobald ein Merge mit dem Ziel-Branch erfolgt durchfegührt wurde, schliesst die `git`-Hosting-Plattform den Pull-Request automatisch. Pull-Requests können auch Issues zugeordent werden, so dass ein oder mehrere zugeordnete Issues automatisch geschlossen werden, sobald ein Pull-Request gemerged wurde.

7.4. Anwendungen und Praxistipps

7.4.1. GitHub-Flow

Es hat sich bewährt, **jede** Aktvität am Repository in einem eigenen **Branch** durchzuführen. Dabei wird jeder Branch immer mit einem **Issue** verknüpft. Ist die Arbeit abgeschlossen, dann wird ein Pull-request zur Qualitätskontrolle erstellt. Sind die Änderungen in Ordnung, dann kann dieser Pull-Request mit einem **Merge** abgeschlossen werden. Der `main`-Branch enthält bei dieser Vorgehensweise immer eine korrekte und funktionierende Version des Projekts. Diese Vorgehensweise wird als ***GitHub-Flow*** (GitHub Inc., o. J.) bezeichnet.

Abbildung 7.1 zeigt ein einfaches Beispiel für den Ablauf des GitHub-Flows.

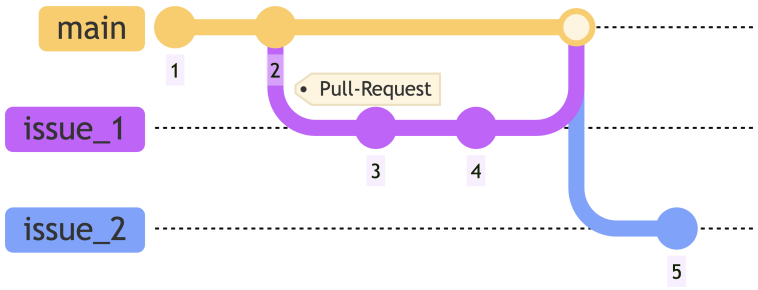


Abbildung 7.1.: Vereinfachter GitHub-Flow

Der GitHub-Flow arbeitet immer nach dem gleichem Grundschema:

1. Es wird ein *Issue* mit den Anforderungen für die Aufgabe erstellt.
2. Es wird ein *Issue-Branch* erstellt.
3. Es werden die Änderungen im *Issue-Branch* festgehalten.
4. Es wird ein *Pull-Request* für den *Issue-Branch* erstellt.
5. Sind alle Anforderungen erfüllt und existieren keine Merge-Konflikte, wird ein Merge des Issue-Branch mit dem `main`-Branch durchgeführt.
6. Der *Issue-Branch* wird aus dem Repository gelöscht.

Im GitHub-Flow gelten die folgenden Regeln für die Projektarbeit:

- Branches werden *ausschliesslich* über Pull-Requests mit dem `main`-Branch zusammengeführt.
- *Alle* Pull-Requests beziehen sich auf mindestens ein Issue.
- Es erfolgen *keine* Commits in den `main`-Branch.
- Es werden ausschliesslich konfliktfreie Merges durchgeführt. Alle Merge-Konflikte müssen im zugehörigen Issue-Branch gelöst werden.

Ein *Issue-Branch* ist ein normaler `git`-Branch mit dem Namen oder der Nummer des Issues. Am einfachsten sind Branchnamen die mit `issue_` beginnen und mit der jeweiligen Issue-Nummer enden. Z.B. für das erste Issue eines Projekts würde der zugehörige Issue-Branch `issue_1` heissen. Der Vorteil dieser Benennung ist, dass ein Branch einem Issue zugeordnet werden kann, selbst wenn noch kein Pull-Request erstellt wurde.

Es wird empfohlen, einen Pull-Request zu öffnen, sobald eine Version in den Branch committed wurde. Dadurch können automatische Qualitätssicherungsschritte die Arbeit unterstützen. Ein Pull-Request muss spätestens nach vollendeter Arbeit an einem Issue erstellt werden.

Pull-Requests sollten mit *Action-Keywords* konfiguriert werden, so dass mit dem schliessen des Pull-Requests durch einen Merge das zugehörige Issue automatisch geschlossen wird.

Abschliessend wird der Issue-Branch gelöscht. Technisch wird nur die Verzweigung aus dem Repository entfernt, weil die Versionierung beim Merge erhalten bleibt. Dadurch wird sichergestellt, dass an dieser Aufgabe nicht versehentlich weiter gearbeitet wird.

7.4.2. Dateien und Verzeichnisse von der Versionierung ausschliessen

`git` meldet "nicht verfolgte"-Dateien, um Datenverlusten vorzubeugen. Nicht alle Dateien in einem Projekt müssen oder sollten versioniert werden. Das gilt meist für die folgenden Kategorien.

- Dateien des Betriebssystems
- Konfigurationsdateien
- Hilfsmodule und Komponenten von Dritten
- Temporäre Dateien und Zwischenergebnisse

Damit diese Dateien nicht versehentlich in der Versionierung erfasst werden, sollten sie aus dem Prozess ausgenommen werden. Dateien und Verzeichnisse können von der Versionierung ausgenommen werden, indem sie in der Datei `.gitignore` erwähnt werden. `.gitignore` kann gezielt einzelne Namen oder Namesmuster ausklammern. `.gitignore` ist Teil der versionierten Daten, so dass sauber nachvollziehbar ist, welche Dateien aus der Versionierung ausgenommenen werden sollten.

Eine Datei wird nur ignoriert, wenn sich noch nicht in der Versionierung erfasst wurde.

> Achtung
>
> Eine Datei wird von der Versionierung ausgeschlossen, wenn eine der folgenden Regeln zutrifft.

- Der Datei- oder Verzeichnisname ist explizit in `.gitignore` gelistet. Für Namen in Unterverzeichnissen muss dazu der Gesamtepfad angegeben sein.
- Eine Datei oder ein Verzeichnis liegt in einem Verzeichnis, dass ignoriert wird.
- Der Datei- oder Verzeichnisname wird durch ein Namesmuster erfasst.

Die folgenden Namesmuster sind gebräuchlich:

- `*/dateiname` - eine Datei oder ein Verzeichnis mit einem bestimmten Namen liegt in einem Verzeichnis im Projektverzeichnis.
- `**/dateiname` - eine Datei oder ein Verzeichnis liegt an beliebiger Stelle im Projekt
- `*.tmp` - alle Dateien oder Verzeichnisse im Projektverzeichnis, die auf `.tmp` enden.
- `bild*` - alle Dateien oder Verzeichnisse im Projektverzeichnis, die mit `bild` beginnen.

Gelegentlich kommt es vor, dass festgestellt wird, dass eine oder mehrere Dateien ignoriert werden soll, obwohl sie bereits in die Versionierung aufgenommen wurde. In diesem Fall erscheit es, als ob `.gitignore` nicht funktioniert. Das liegt daran, dass `git` eine Datei nur ignoriert, wenn diese nicht in der Versionierung erfasst wurde. Um eine bereits erfasste Datei zu irgnorieren, muss diese Datei aus dem Versionsdatenbank mit `git rm` entfernt werden.

Beispiel

Das folgende Beispiel zeigt typische `.gitignore`-Datei für ein R-Projekt.

```
# Ignore MacOS Filesystem Metadata
**/.DS_Store

# R-specific Files
.Rhistory
.Rapp.history
.RData
.RDataTmp
.Ruserdata
.Renviron

*-Ex.R

/*.tar.gz
/*.Rcheck/
.Rproj.user/

vignettes/*.html
vignettes/*.pdf

rsconnect/
.httr-oauth

*_cache/
/cache/
*.utf8.md
*.knit.md
po/*~
_bookdown_files/
```

```
_book
.quarto/
/.quarto/
```

7.4.3. Repository-Organisation

Die Arbeit mit Git und den anderen Hosting-Plattformen erscheint oft mühsam und aufwändig. Das gilt speziell für grösseres Umorganisieren eines Repositories. Gerade bei komplexen Projekten sollte deshalb möglichst früh die Organisation des Versionsmanagements festgelegt werden.

In der Praxis finden sich drei Organisationsstrategien.

1. Monorepo
2. Projektrepo
3. Microrepo

Die Wahl der Repository-Organisation hängt stark von den Projektanforderungen ab.

7.4.3.1. Monorepo

Bei der Monorepo-Strategie werden alle Dateien in einem einzigen Repository organisiert. Diese Strategie ist besonders bei Projektbeginn oder bei einfachen Projekten mit geringer funktionaler Trennung geeignet. Der Vorteil dieser Strategie ist, dass alle Daten in einem Repository zusammengefasst vorliegen und als Ganzes verwaltet werden können.

Für komplexe Projekte wird diese Organisationsstrategie normalerweise mit dem Git-Flow als Branching-Strategie kombiniert. Dabei müssen alle Projektteile aufeinander abgestimmt werden und benötigen eine konsitente Qualitätskontrolle, weil alle Änderungen voneinander abhängig sind. Komplexe Änderungen beeinflussen wegen der linearen Abhängigkeit der Versionen deshalb alle anderen Bereich und sind deshalb wesentlich schwerer zu korrigieren.

7.4.3.2. Projektrepo

Die Projektrepo-Strategie ist eine Variante der Monorepo-Strategie für komplexe Projekte, mit klar getrennten aber voneinander abhängigen Projektteilen. Dazu werden einzelene Projektteile in eigene Repositories ausgegliedert und als *`git`-Submodule* in das Haupt-Repository eingebunden. Solche Submodule können unabhängig vom Haupt-Projekt entwickelt werden und auch in anderen Projekten wiederverwendet werden. Die Integration eines Submoduls erfolgt für einen bestimmten Commit oder ein bestimmtes Tag. Dadurch wird eine differenziertere Qualitätskontrolle möglich.

7.4.3.3. Microrepo

Bei der Microrepo-Strategie werden alle Projektteile in eigenen Repositories unabhängig voneinander geführt, so dass nur der jeweilige Projektteil und die zugehörige Konfiguration in einem Repository vorliegen.

Bei der Microrepo-Strategie werden Daten, Code und Konfiguration grundsätzlich immer voneinander getrennt und werden in eigenen Repositories verwaltet. Damit ist diese Strategie die Antithese zum Monorepo. Die Integration der Komponenten erfolgt über das lokale Dateisystem oder über Packetmanager und `git`-Tags.

Diese Strategie eignet sich für Datenprojekte, Projekte mit wenigen direkten Abhähngigkeiten oder Projekte mit verschiedenen Anwendungsgebieten (engl. Deployments). Durch klar getrennte Repositories lassen sich die einzelnen Komponenten in unterschiedlichen Kontexten problemlos unabhängig voneinander publizieren und anpassen. Deshalb ist diese Strategie besonders für Projekte mit sensitiven Daten, wie z.B. Patientendaten, oder Konfigurationen, wie beispielsweise Passwörtern, geeignet, bei denen die Datenverarbeitung keine sensitiven Teile hat.

8. Datentypen

8.1. Fundamentale Datentypen

Datentypen helfen uns Werte zu organisieren. Wir haben bereits drei wichtige Datentypen kennengelernt. Damit kennen wir bereits den Grossteil der wichtigsten Datentypen für einzelne Werte. Die vollständige Auflistung besteht aus fünf Datentypen:

1. Zahlen (engl. *numerics*)
2. Wahrheitswerte (engl. *booleans*)
3. Zeichenketten (engl. *strings*)
4. Fehlerwerte (engl. *error values*)
5. Undefinierte Werte (engl. *undefined values*).

Diese Datentypen beschreiben jeweils einzelne Werte.

Definition 8.1. Fundamentale Datentypen heissen Datentypen, die Eigenschaften für einzelne Werte festlegen.

Definition 8.2. Ein Wert von einem fundamentalen Datentyp heisst **Skalar**.

Die fundamentalen Datentypen legen die allgemeinen Wertebereiche fest, auf welchen die speziellen Wertebereiche der Datenschemata aufbauen.

8.1.1. Undefinierte Werte

Gelegentlich fehlen Werte in den Daten oder wurden bei der Datenverarbeitung (noch) nicht zugewiesen. In diesen Fällen werden die "fehlenden" Werte nicht ignoriert, sondern als **undefinierte Werte** festgehalten.

Definition 8.3. Als **undefinierte Werte** die keine Werte repräsentieren.

In den meisten Programmiersprachen kennen für undefinierte Werte eine konstanten Wert. Dieser Wert ist in der Regel vom beliebigen Datentyp.

8.1.2. Zahlen

Definition 8.4. Als **Zahlenwerte** werden nummerische Werte bezeichnet.

Zahlenwerte können verschiedene Darstellungen haben, ohne dass sich der *Wert* der Zahl verändert (s. Kapitel 16).

Zahlenwerte können in zwei Klassen unterteilt werden:

- *Ganze Zahlen*
- *Reelle Zahlen*

Daneben unterstützen die meisten Programmiersprachen für die Datenanalyse auch *komplexe Zahlen*. Diese werden jedoch selten direkt eingegeben, sondern sind in der Regel Teil von Berechnungen.

> **Warnung**
>
> Ein Teil der reellen Zahlen sind die *Gleitkommazahlen*. Gleitkommazahlen arbeiten wie die wissenschaftliche Schreibweise von Zahlen (s. Abschnitt 3.4) mit einer Basis und einer Potenz, durch welche die Grössenordnung einer Zahl definiert ist.
> Alle reellen Zahlen werden in Computern grundsätzlich als Gleitkommazahlen abgebildet. Dadurch wird der mögliche Zahlenraum prinzipiell erweitert. Zahlen lassen sich so nicht mehr mit beliebiger Genauigkeit behandeln. Deshalb kann mit Gleitkommazahlen nicht immer exakt gerechnet werden. Das ist immer dann der Fall, wenn sehr kleine und sehr grosse Zahlen miteinander verrechnet werden. In solchen Fällen kann es zu unerwarteten Rundungsfehlern kommen.
> Falls sehr genaue Berechnungen notwendig sind, dann müssen spezielle Bibliotheken verwendet werden, die mit der entsprechenden Genauigkeit umgehen können.

8.1.3. Wahrheitswerte

Definition 8.5. Als **Wahrheitswerte** werden die Werte `Wahr` (`True`) und `Falsch` (`False`) bezeichnet.

Weil es nur zwei Wahrheitswerte gibt, werden sind Wahrheitswerte ein **binärer Datentyp**.

In den meisten Programmiersprachen lassen sich Zahlenwerte als Wahrheitswerte verwenden. Dabei wird die Zahl `0` als `Falsch` und alle anderen Zahlen als `Wahr` interpretiert.

Wahrheitswerte werden meistens zur Steuerung der Programmlogik verwendet (s. Kapitel 12).

8.1.4. Zeichenketten

Definition 8.6. Als **Zeichenketten** werden Werte bezeichnet, die aus einer Zeichenfolge bestehen. Dazu gehören Buchstaben, Ziffern, sowie Leer-, Satz- und Sonderzeichen.

Zeichenketten setzen sich aus Symbolen zusammen, weshalb Zeichenketten eine *Länge* haben. Die Länge einer Zeichenkette entspricht der Anzahl ihrer Symbole. Entsprechend hat jedes Symbol eine eindeutige *Position* in der Zeichenkette.

Zeichenketten werden für alle Werte verwendet, die nicht als Zahlenwerte oder Wahrheitswerte dargestellt werden können. Gelegentlich müssen Zahlen als Zeichenketten dargestellt werden, um sie in einem bestimmten Format auszugeben oder für mathematische Operationen zu sperren.

In den meisen Programmiersprachen müssen Zeichenketten besonders markiert werden, damit sie von anderen Datentypen und Bezeichnern unterschieden werden können (s. Kapitel 10). Die Markierung erfolgt meistens durch Anführungszeichen.

> **Hinweis**
>
> Programmiersprachen wie bspw. C, C++, C# oder Java verwenden für Zeichenketten den fundamentalen Datentyp des *Symbols* (oder *Character*). In diesen Programmiersprachen werden Zeichenketten als Liste von Symbolen behandelt.
> Diese Unterscheidung ist für die Programmiersprachen der Datenwissenschaften nur insofern von Bedeutung, als dass eine Zeichenkette eine Länge hat und jedes Symbol in einer Zeichenkette über dessen Position abgefragt werden kann.

8.1.5. Fehlerwerte

Definition 8.7. Als **Fehlerwerte** werden Werte bezeichnet, die Fehler repräsentieren.

Fehlerwerte unterscheiden sich von den anderen Datentypen. Mit ihnen eine Programmierumgebung oder eine Programmiersprache an, dass ein Fehler aufgetreten ist oder eine unzulässige Operation ausgeführt wurde. Für die meisten Programmiersprachen sind die möglichen Fehler von Operationen und Funktionen dokumentiert.

Fehlerwerte können nicht mit anderen Datentypen verknüpft werden. Wird ein Fehlerwert mit einem anderen Datentyp verknüpft, dann wird der Fehlerwert als Ergebnis zurückgegeben.

Fehlerwerte geben an, welche Art von Fehler aufgetreten ist. Dadurch können regelmässig auftretende Fehler automatisch behandelt werden. Hierzu stellen alle Programmiersprachen spezielle Funktionen zur *Fehlerbehandlung* bereit.

> **Praxis**
>
> Ein besonderer Fehlerwert in der Datenanalyse ist der **fehlende Wert**. Er wird verwendet, wenn ein Wert nicht vorhanden ist oder nicht zugewiesen wurde. Dieser Wert ist notwendig, weil fehlende Werte (normalerweise) nicht durch den Wert `0` ersetzt werden dürfen. Er ist dem Datentyp *undefinierter Wert* vorzuziehen, weil dieser Fehlerwert einen fehlenden Wert *explizit* kennzeichnet und diesen Wert für normale Operationen sperrt, was bei undefinierten Werten nicht immer der Fall ist.

8.2. Klassen von Datentypen

Es werden zwei Klassen von Datentypen unterschieden:

1. Diskrete Daten
2. Kontinuierliche Daten

> **Hinweis**
>
> In der Statistik heissen die folgenden Wertebereichklassen **Skalenniveaus**.

8.2.1. Diskrete Daten

Definition 8.8. Wenn ein Wertebereich nur eindeutige Werte enthält, dann heisst der Wertebereich *diskret*.

Ganze Zahlen, Wahrheitswerte, Zeichenketten und Fehlerwerte haben immer diskrete Wertebereiche. Undefinierte Werte umfassen nur einen einzigen Wert und können je nach Kontext diskret oder kontinuierlich sein.

Es werden drei Arten von diskreten Daten unterschieden:

- Nominale Daten
- Ordinale Daten
- Intervallskalierte Daten

Zusätzlich erfahren *binäre Wertebereiche* oft eine besondere Behandlung in der Literatur.

8.2.1.1. Nominalskalierte Daten

Definition 8.9. Lassen sich alle Werte in einem Wertebereich eindeutig voneinander unterscheiden, dann liegt ein **nominalskalierter Wertebereich** vor.

Nominalskalierte Daten können nur durch Gleichheit oder Ungleichheit voneinander unterschieden werden. Nominalskalierte Daten können nur über Häufigkeiten zusammengefasst werden.

8.2.1.2. Ordinalskalierte Daten

Definition 8.10. Lassen sich alle Werte eines nominalskalierten Wertebereichs in eine eindeutige Reihenfolge sortieren, dann liegt ein **ordinalskalierter Wertebereich** vor.

Ordinale Daten sind eine Untermenge der nominalen Daten, bei der sich die Werte im Wertebereich *sortieren* lassen. Dadurch kann jedem Wert ein *eindeutiger Rang* zugewiesen werden. Der Rang markiert die *Position* eines Werts im *sortierten Wertebereich*.

> **Hinweis**
>
> Eine Reihenfolge bedeutet *nicht*, dass auch die *Abstände* zwischen allen Werten gleich sind.

8.2.1.3. Binäre Daten

Definition 8.11. Hat ein Wertebereich genau zwei Werte, dann liegt ein **binärer Wertebereich** vor.

Binäre Wertebereiche sind ein Sonderfall unter den diskreten Daten, denn sie können entweder nominal- oder ordinalskaliert sein.

Ein binärer Wertebereich ist ordinalskaliert, wenn die zulässigen Werte sortierbar sind. Ein häufiger Fall eines ordinalen binären Wertebereichs sind *vorher-nachher*-Markierungen.

> **Merke**
>
> Alle binären Daten können als Wahrheitswerte dargestellt werden.

Mehrere binäre Wertebereiche können durch die Verknüpfung von Zweier-Potenzen zu einem **nominalskalierten** Wertebereich zusammengefasst werden.

8.2.1.4. Intervallskalierte Daten

Definition 8.12. Haben alle Werte in einem Wertebereich die gleichen Abstände, ist der Wertebereich **intervallskaliert**.

Intervallskalierte Daten sind über die *Differenz* definiert: Werden zwei beliebige benachbarte Wertepaare ausgewählt, dann muss die Differenz zwischen Paaren, zwischen den beiden kleineren Werten und zwischen den beiden grösseren Werten jeweils zueinander gleich sein. Benachbarte Werte unterscheiden sich im Rang um 1.

Beispiel 8.1. Aus dem ordinalskalierten Wertebereich der ganzen Zahlen $\mathbb{I}$ werden die Werte `1`, `2`, `5` und `6` gewählt. Die Wertepaare `1` und `2` sowie `5` und `6` sind benachbart.

Wenn der Wertebereich intervallskaliert ist, dann müssen beide Gleichungen in Formel 8.1 gelten.

$$2-1=6-5 \Leftrightarrow 1=1 und 6-2=5-1 \Leftrightarrow 4=4 \tag{8.1}$$

Beide Gleichungen müssen für alle Wertepaare im Wertebereich gültig sein, was bei den ganzen Zahlen der Fall ist. Deshalb ist $\mathbb{I}$ intervallskaliert.

> **Wichtig**
>
> Die Abstände zwischen den einzelnen Werten lassen sich nicht für alle ordinalen Wertebereiche feststellen. *Solche Wertebereiche sind* ***nie*** *intervallskaliert.*

8.2.2. Kontinuierliche Daten

Lässt ein Wertebereich beliebige Abstufungen zwischen Werten zu, dann heisst dieser Wertebereich *kontinuierlich.*

Definition 8.13. Kontinuierliche Daten liegen vor, wenn alle Werte im Wertebereich *sortierbar* sind, die *gleichen Abstände* haben und die Verhältnisse von Werten ebenfalls im Wertebereich liegen.

> **Hinweis**
>
> *Kontinierliche Daten* werden auch als *metrische Daten*, *varianzskalierte Daten* oder *kardinale Skalenniveaus* bezeichnet.

Definition 8.13 verweist auf die Verhältnisse zwischen zwei Werten. Das Verhältnis ist das Gleiche wie ein Bruch bzw. die Division. Bei diskreten Daten können die Verhältnisse zwischen beliebigen Wertepaaren nicht so gebildet werden, dass auch alle Verhältnisse im gleichen diskreten Wertebereich liegen.

Beispiel 8.2. Gegeben sind die Werte `1`, `2`, `3` und `4` aus dem *intervallskalierten* Wertebereich der ganzen Zahlen $\mathbb{I}$.

Die Verhältnisse zwischen diesen Werten sind $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, $\frac{2}{1}$, $\frac{2}{3}$, $\frac{2}{4}$, $\frac{3}{1}$, $\frac{3}{2}$, $\frac{3}{4}$, $\frac{4}{1}$, $\frac{4}{2}$ und $\frac{4}{3}$. Nur die Verhältnisse $\frac{2}{1}$, $\frac{3}{1}$, $\frac{4}{1}$ und $\frac{4}{2}$ sind wieder ganze Zahlen und würden zum Wertebereich gehören.

Wäre der Wertebereich mit $\mathbb{R}$ festgelegt worden, dann liegen alle Verhältnisse ebenfalls im Wertebereich von $\mathbb{R}$. Daraus folgt, dass $\mathbb{R}$ ein *kontinuierlicher Wertebereich* ist.

Alle kontinuierlichen Daten lassen sich auf reelle Zahlen abbilden.

> **Praxis**
>
> Häufig stellt sich bei der Analyse der Daten heraus, dass die Werte eines eigentlich kontinuierlichen Wertebereichs nur (wenige) diskrete Werte annehmen. In diesem Fall **muss** der Wertebereich als *diskret* behandelt werden.

8.3. Datenstrukturen

Aus fundamentalen Datentypen können *komplexe Datentypen* zusammengesetzt werden, um mehrere Werte zusammenzufassen. Komplexe Datentypen können ausserdem wiederum komplexe Datentypen enthalten. So lassen sich komplexe Strukturen bilden.

Definition 8.14. Eine **Datenstruktur** ist ein Datentyp, in dem mehrere Werte zusammengefasst werden können.

Weil eine Datenstruktur ebenfalls ein Datentyp ist, lässt sie sich wie ein einzelner Wert behandeln. Dadurch haben Datenstrukturen eine Reihe von Vorteilen:

- Datenstrukturen können als *Einheit* behadelt werden.
- Datenstrukturen können *verschachtelt* werden, um noch komplexere Strukturen zu bilden.
- Datenstrukturen können als Sammlung von Einzelwerten betrachtet werden, welche getrennt verarbeitet werden können.

Durch diese Eigentschaften sind Datenstrukturen besonders gut für die Datenorganisation geeignet.

8.3.1. Eindimensionale Datenstrukturen

Datenfelder (engl. *arrays*) sind eine besondere Datenstruktur. Datenfelder fassen Werte so zusammen, dass jeder Wert eine *eindeutige Position* hat. Die Position eines Wertes wird durch einen *Index* angegeben. Die Anzahl der Werte in einer Datenstruktur ist deren **Länge**. Die Länge und die Idizes einer Datenstruktur sind natürliche Zahlen.

Über die zusammengefassten Datentypen lassen sich zwei Arten von Datenfeldern unterscheiden.

Definition 8.15. Vektoren sind Datenfelder mit Werten vom *gleichen Datentyp*.

Weil alle Werte in einem Vektor vom gleichen Datentyp sind, sind Vektoren *homogene Datenstrukturen*.

Definition 8.16. Listen sind Datenfelder mit Werten mit *beliebigen Datentypen*.

Weil die Datentypen der Listenwerte nicht festgelegt sind, sind Listen *heterogene Datenstrukturen*.

8.3.2. Mehrdimensionale Datentypen

Die Logik der komplexen Datentypen erlaubt die Verknüpfung von Listen und Vektoren zu noch komplexeren Strukturen.

Definition 8.17. Eine **geschachtelte Liste** ist eine Liste, die aus beliebigen anderen Datenfeldern (d.h. Listen oder Vektoren) besteht, wobei die eingebetteten Datenfelder *unterschiedliche Längen* haben können.

Ein **Spezialfall** der geschachtelten Liste sind **Listen, die Vektoren mit *gleicher Länge* schachteln**. Diese Spezialfälle haben besondere Namen:

Definition 8.18. Eine **Datenrahmen** (engl. *data frame*) verknüpft mehrere gleichlange Vektoren mit *unterschiedlichen Datentypen*.

Ein Datenrahmen kann als eine Liste von Vektoren mit gleicher Länge verstanden werden. Diese Datenstruktur muss eine Liste sein, weil die einzelnen Vektoren unterschiedliche Datentypen haben können. Entsprechend gelten Datenrahmen ebenfalls als *hetreogene Datenstrukturen*.

Die gleiche Logik lässt sich auch auf Vektoren anwenden. In diesem Fall ist der **Spezialfall** ein **Vektor, der Vektoren gleicher Länge und des gleichen Datentyps schachtelt**. Diese Datenstruktur hat ebenfalls einen eigenen Namen:

Definition 8.19. Eine **Matrix** verknüpft mehrere gleichlange Vektoren vom Datentyp *Zahlen*. Die Länge der Vektoren einer Matrix muss mindestens 1 sein.

Weil alle Vektoren in einer Matrix vom gleichen Datentyp sind, sind Matrizen *homogene Datenstrukturen*.

Eine Matrix hat eine Höhe (oft als m Zeilen gekennzeichnet) und eine Breite (oft als n Spalten gekennzeichnet). Die Anzahl der Zeilen und Spalten müssen natürliche Zahlen sein. Daraus folgt, dass nur Matrizen existieren, für die m und n > 0 gilt.

Eine Matrix mit m Zeilen und n Spalten wird als $m \times n$-Matrix bezeichnet. Die Anzahl der Zeilen und Spalten einer Matrix sind ihre *Dimensionen*.

> **Hinweis**
>
> Die *Dimensionen* einer Matrix müssen beide > 0 sein, aber sie müssen genau nicht gleich sein. Weil m und n einer Matrix grösser als `0` sein müssen, folgt daraus: **Vektoren** sind *spezielle Matrizen* mit m-Zeilen und *einer Spalte* (n = 1).

Definition 8.20. Eine Matrix mit gleichvielen Spalten und Zeilen wird als **quadratische Matrix** bezeichnet.

Für eine quadratische Matrix gilt immer $m = n$.

Der Wert an einer Position in einer Matrix ist durch den Zeilen- und Spaltenindex eindeutig identifiziert.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

8.4. Vektorformen von Matrizen

Neben der üblichen Schreibweise von Matrizen, können Matrizen auch als *Vektoren* dargestellt werden. Dabei handelt es sich um eine besondere Schreibweise der Matrix, die *Vektorform einer Matrix* genannt wird. Für jede Matrix gibt es immer *zwei* Vektorformen. Die eine Vektorform fügt die Werte einer Matrix *spaltenweise* aneinander, die andere Vektorform fügt die Werte *zeilenweise* aneinander.

Die Vektorform einer Matrix kann nur gebildet werden, weil wegen Definition 8.19 die Datentypen aller Werte in einer Matrix gleich sind. Deshalb wird die Bedingung aus Definition 8.15 für die Vektorform einer Matrix erfüllt.

Beispiel 8.3. Vektorformen einer Matrix

Für die folgende Matrix A mit $m = 3$ Zeilen und $n = 2$ Spalten soll in ihre Vektorform überführt werden.

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 6 & 2 \\ 4 & 6 \\ 9 & 7 \end{bmatrix}$$

Für alle Matrizen gibt es zwei Vektorformen.

Die erste Vektorform A_s fügt die Werte *spaltenweise* aneinander.

$$A_s = \begin{bmatrix} 1 \\ 0 \\ 6 \\ 4 \\ 9 \\ 0 \\ 2 \\ 2 \\ 6 \\ 7 \end{bmatrix}$$

Die zweite Vektorform A_z fügt die Werte *zeilenweise* aneinander.

$$A_z = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 6 \\ 2 \\ 4 \\ 6 \\ 9 \\ 7 \end{bmatrix}$$

9. Dateiformate

Zum Speichern und zum Austausch von Daten spielen Dateiformate enie wichtige Rolle. Dateiformate legen fest, wie Daten serialisiert werden. Die Serialisierung von Daten ist die Grundlage für die Speicherung und den Austausch von Daten. Eine spezielle Serialisierung wird als *Datenformat* bezeichnet.

In Computern werden Daten immer als *Datenströme* abgebildet.

Definition 9.1. Ein **Datenstrom** ist eine Folge von binärkodierten Bits.

Ein Datenstrom ist also immer eine Abfolge von `0` und `1` Werten. Diese Abfolgen sind eine *Datenserialisierung* (s. Abschnitt 3.4.1). Für die Interpretation werden diese Abfolgen in Gruppen zusammengefasst. Üblich sind Gruppen von 8-Bit (dem sog. *Byte*), 16-Bit, 32-Bit oder 64-Bit.

Datenströme können eine *feste Länge* haben. In diesem Fall wird die Länge eines Datenstroms wird *Byte* angegegeben.

Definition 9.2. Eine **Datei** ist ein Datenstrom mit fester Länge, der von einem Speichermedium geladen wird.

Weil eine Datei auf einem Speichermedium gespeichert ist, kann der Datenstrom aus einer Datei wiederholt abgerufen werden. Deshalb gehören Dateien zu den sog. ***persistenten Datenströmen***. Persistente Datenströme liefern also *dauerhafte Daten*.

Nicht alle Datenströme sind persistent. Deshalb ist es wichtig, die Art eines Datenstroms zu kennen, denn bei nicht-persisten oder **flüchtigen Datenströmen** kann auf die Daten nur einmal zugegriffen werden. *Flüchtige Datenströme* sind kommen vorwiegend bei der *automatischen Datenerfassung* vor. In diesem Fällen entspricht jeder Zugriff auf den Datenstrom einer Messung, wobei es nicht möglich ist, eine Messung zu wiederholen.

9.1. Dateien verwenden

9.1.1. Importieren

Definition 9.3. Das Lesen eines Datenstroms in eine Datenstruktur wird **Importieren** genannt.

Das Importieren von Daten folgt immer in mehreren Schritten:

1. *Lesen* eines Datenstroms.
2. *Zuordnung* der Werte zu den richtigen Datentypen.
3. *Auswahl* der zu verarbeitenden Daten.

9.1.1.1. Daten einlesen

Beim ersten Schritt werden die Symbole eines Datenstroms in eine geeignete Datenstruktur überführt. Dabei werden die gelesenen Symbole überprüft, ob es sich um Werte oder um eine Trennmarkierung handelt. Die Werte werden entsprechend ihrer Position im Datenstrom oder über Markierungen einer Datenstruktur zugeordnet.

Definition 9.4. Die Zuordnung von Werten in einem Datenstrom zu einer Datenstruktur wird **Parsen** genannt. Eine Funktion, die einen Datenstrom in eine Datenstruktur überführt, heisst **Parser**.

Jedes Dateiformat erfordert einen eigenen *Parser*. Wird ein ungeeigneter *Parser* verwendet, dann werden die Daten nicht korrekt importiert und können nicht weiterverarbeitet werden, weil die Datenstruktur nicht die Organisation der Daten wiedergibt. Deshalb muss beim Importieren immer ein passender Parser für das vorliegende Dateiformat verwendet werden.

> **Hinweis**
>
> Für die gängigsten Datenformate existieren eigene Parser. Die Entwicklung eines Parsers für ein bestimmtes Format ist nur selten notwendig.

Die Auswahl eines geeigneten Parsers liegt in der Verantwortung der Person, die die Daten importiert. In manchen Fällen kann das Dateiformat automatisch erkannt werden. In diesen Fällen wird ein geeigneter Parser automatisch ausgewählt. Diese automatische Auswahl ist aber nicht immer korrekt. Deshalb muss das Ergebnis des Parsers immer überprüft werden.

9.1.1.2. Werte zuordnen

Der zweite Schritt beim Importieren ist die Zuordnung der Werte zu den richtigen Datentypen und der richtigen Struktur. Normalerweise versucht ein Parser bereits die richtigen Datentypen zu erkennen. Dabei kann es zu Fehlern kommen, wenn sich die Werte nicht eindeutig einem Datentyp zuordnen lassen oder ein spezieller Datentyp verwendet werden soll.

Diesem Schritt umfasst auch das Erkennen und Behandeln von Markierungen, welche in internen Strukturen als Variablennamen oder benannten Datenfeldern abgebildet werden. Bei Separator-strukturierten Dateien umfasst dieser Teilschritt die Zuordnung von

Überschriften. Bei Markup-Daten werden Tags und Markierungen für die interne Struktur verwendet.

Dieser Schritt umfasst oft auch die Behandlung von ungültigen oder fehlenden Werten.

9.1.1.3. Daten auswählen

Gelegentlich werden mehr Daten in einem Datenstrom bereitgestellt, als für die weitere Verarbeitung benötigt werden. In solchen Fällen werden nur die Daten ausgewählt, die für die weitere Verarbeitung benötigt werden. Bei diesem Schritt werden nicht benötigte Daten aus der internen Datenstruktur entfernt.

> **!** Wichtig
>
> Das Auswählen von Daten bezieht sich ausschliesslich auf die interne Datenstruktur. Die ursprünglichen Daten im Datenstrom dürfen nicht verändert werden.

9.1.2. Exportieren

Definition 9.5. Das Schreiben von Daten aus einer Datenstruktur in einen Datenstrom heisst **Exportieren**.

Beim Exportieren werden die Daten aus einer Datenstruktur in einen Datenstrom *serialisiert*. Eine Funktion die eine Datenstruktur in einen Datenstrom überführt, heisst **Serialisierer**.

Ähnlich wie beim Importieren muss auch beim Exportieren existiert für jedes Datenformat ein geeigneter Serializer. Die Wahl eines Serialisierers hängt von der späteren Verwendung der Serialisierung ab.

> **i** Hinweis
>
> Es ist üblich, die gleichen Daten in verschiedenen Formaten zu serialisieren.

Einige Dateiformat sind sehr komplex. Dadurch sind unvollständige Serialisierungen möglich. Eine unvollständige Serialisierung kann zu Datenverlusten führen. Deshalb sollte für gängige Dateiformate immer ein existierender Serializer verwendet werden.

> Praxis
>
> Beim Exportieren von Daten die vorher importiert wurde dürfen die ursprüngliche Daten nicht überschrieben werden, weil sonst Daten verloren gehen können. Stattdessen sollten Daten beim Exportieren in eine separate Datei geschrieben werden.

9.2. Textdateien

Definition 9.6. Eine Textformat legt fest, dass alle Werte in einer Datei als Zeichenkettensymbole interpretiert werden müssen.

Die Bits einer Textdatei sind nach einem festgelegten Standard kodiert (s. Kapitel 3.5). Dadurch kann der Inhalt einer Textdatei immer zuverlässig gelesen und interpretiert werden. Inzwischen verwenden alle modernen Betriebssysteme einheitlich den Kodierungsstandard UTF-8 (ISO/IEC JTC 1/SC 2, 2020; The Unicode Consortium, 2022).

Textdateien haben den grossen Vorteil, dass die Dekodierung der Bits automatisch erfolgt. Eine Programm oder eine Funktion kann also direkt auf die kodierten Symbole zugreifen.

i Textkodierungen

Ältere Datenerfassungs- oder Steuerungssysteme sind noch heute im Einsatz. Einige dieser Systeme verwenden für Textdaten einen älteren Kodierungsstandard. In solchen Fällen muss beim Zugriff auf die Daten die abweichende Kodierung explizit angegeben werden.
Eine abweichende Textkodierung wird oft an merkwürdigen Zeichenfolgen in den Daten erkannt. Nicht immer ist die verwendete Kodierung dokumentiert. Oft lässt sich die Textkodierung aus dem kulturellen Kontext der Daten ableiten. Die folgende Tabelle gibt für Europa und die USA eine Orientierung, welche Textkodierungen in welchen Zeiträumen verwendet wurden.

Land/Region	Zeitraum	Wahrscheinliche Kodierung
Weltweit	seit ca. 2012	UTF-8 (ISO/IEC JTC 1/SC 2, 2020)
West-Europa	2000-2012	iso-8859-15 (ISO/IEC JTC 1/SC 2/WG 3, 1998b)
USA	2000 - 2012	iso-8859-1 (ISO/IEC JTC 1/SC 2/WG 3, 1998a)
West-Europa	1990 - 2000	iso-8859-1 (ISO/IEC JTC 1/SC 2/WG 3, 1998a)
USA	vor 2000	ANSI/ASCII (American National Standards Institute, 1977)

9.3. Festkodierung

Bei der Festkodierung werden die Werte in einer festen Reihenfolge und festen Symbol- oder Bitlängen in den Datenstrom geschrieben. Bei der Festkodierung ist die Reihenfolge der Werte und die Länge der Werte festgelegt.

Festkodierungen erlauben sehr effizientes speichern und lesen von Daten. Damit festkodierte Daten verarbeitet können müssen die Datenfelder in einem gemeinsamen Schema festgelegt sein, dass beim Kodieren und Dekodieren verwendet wird.

Beispiel 9.1 (Kodierung der IBAN)**.** Die IBAN ist ein festkodiertes Datenformat mit vier Werten. Die Kodierung ist international einheitlich festgelegt (ISO/TC 68/SC 8, 2020).

- Länderkennung (2 Buchstaben)
- Prüfzahl (2 Ziffern)
- Bankkennung (5 Symbole)
- Kontonummer (6-25 Symbole)

Die Prüfzahl erlaubt die Überprüfung der anderen Werte. Dadurch lassen sich Zahlendreher oder fehlerhafte Eingaben erkennen.

Die Länge des letzten Felds wird durch nationale Gremien vereinheitlicht. In Deutschland ist die Kontonummer 10-stellig, in der Schweiz 9-stellig und in Österreich 8-stellig. Die restlichen Stellen bis zur Maximallänge 25 werden mit Leerzeichen aufgefüllt.

Oft werden IBANs mit Leerzeichen gruppiert, um die Lesbarkeit zu erhöhen. Die Leerzeichen werden bei der Verarbeitung gelöscht.

9.4. Zeilenbasierte Textdateien

Normalerweise sind Textdateien unstrukturiert und die verwendeten Symbole dienen nur der Anzeige und haben keine eigene Bedeutung. Das ist nicht immer praktisch, um Daten zu strukturieren. Hier greifen sog. zeilenbasierte Textformate. Diese Textformate sind eine einfache Form der Datenorganisation, wobei alle Symbole in der gleichen Zeile zusammengehören und von den Symbolen in den anderen Zeilen abzugrenzen sind.

Zeilenbasierte Textdateien sind eine spezielle Form der *Listenkodierung*, wobei das Symbol für den Zeilenumbruch als Listentrennzeichen behandelt wird.

Eine typische Anwendung der zeilenbasierten Textkodierung sind sog. *Log-Dateien*, mit denen Systemereignisse protokolliert werden.

9.5. Mengen und Bäume

Die Werte lassen sich auf verschiedene Arten in Datenströmen kodieren. Die einfachste Form ist die **Mengenkodierung**. Bei der Mengenkodierung werden die Werte in einer festen Reihenfolge in den Datenstrom geschrieben. Dabei werden die Werte als eine zusammenhängende *Menge* behandelt.

Die verbreitesten Formen der Mengenkodierung sind die *Listenkodierung* und die *Tabellenkodierung*. Bei der Listenkodierung werden die einzelnen Werte nacheinander in den Datenstrom geschrieben. Dabei ist die Reihenfolge der Werte nich von Bedeutung. Bei der Tabellenkodierung werden die Werte in einer tabellarisch organisiert. Dabei werden die Werte in Zeilen und Spalten organisiert, wobei jede Zeile und jede Spalte als eine Teilmenge

behandelt wird. Die Reihenfolge der Zeilen und Spalten ist dabei nicht von Bedeutung, solange die Zeilen- und Spaltenteilmengen erhalten bleiben.

> **Hinweis**
>
> Die Mengenkodierung ist gleichzeitig die Grundlage für *Datenbanken* und *tabellarischen Dateien*. In Datenbanken werden die Werte tabellenartig in sog. *Relationen* organisiert.

Die Mengenkodierung ist einfach und effizient. Sie hat aber den Nachteil, dass die Werte nicht leicht hierarchisch organisiert werden können, weil jede Hierarchieebene eine eigene Menge erfordert und damit von den anderen Ebenen abgegrenzt wäre. Dieses Problem löst die **Baumkodierung**. Die Baumkodierung kodiert Werte in einer Hierarchie. Dadurch lassen sich komplex-geschachtelte Datenstrukturen in einer Datei abbilden.

In einer Baumkodierung werden die Werte auf einer Hierarchiestufe zusammegefasst. Eine Hierarchiestufe wird als *Knoten* bezeichnet. In einer solchen Kodierung hat jeder Knoten eine Vorgängerstufe. Diese Vorgängerstufe wird *Elternknoten* genannt. Die Knoten auf der gleichen Hierachiestufe haben immer den gleichen Elternknoten. Diese Knoten heissen *Geschwisterknoten*. Die Knoten ohne Nachfolger nennt man *Blattknoten*. Die eigentlichen Werte finden sich nur in Blattknoten.

Mit der Baumkodierung lassen sich beliebig komplexe Datenstrukturen abbilden. Für die Kodierung von Datenstrukturen mit *höchstens zwei* Hierarchiestufen und *vielen Werten* ist die Baumkodierung weniger effizient als die Mengenkodierung.

9.6. Separator-strukturierte Textdateien

Eine besondere Klasse der zeilenbasierten Textdateien sind die sog. "*Separator-strukturierten*" Dateien. Diese Dateiformate sind eine einfache Form der *Tabellenkodierung*, bei der die Werte in der gleichen Zeile durch ein zusätzlichens *Trennzeichen* voneinander abgegrenzt werden.

Separator-strukturierte Dateien sind zeilenbasierte Textdateien, bei denen die Werte in einer Zeile durch ein *Trennzeichen* voneinander abgegrenzt werden. Dadurch lassen sich die Werte in einer Zeile leichter als bei einer Festkodierung kodieren und dekodieren. Für die Trennung der Werte muss nur das verwendete Trennzeichen bekannt sein. Die Position und die Länge der Werte wird durch das Trennzeichen festgelegt und muss nicht vorab definiert werden.

Die verbreitetsten Separator-strukturierten Dateiformate sind:

- *Tab-separated values* (TSV)
- *Comma-separated values* (CSV)
- *Pipe-separated values* (PSV)

Der Unterschied zwischen diesen Formaten ist das verwendete Trennzeichen. TSV-Dateien werden durch ein Tabulatorzeichen getrennt, CSV-Dateien durch ein Komma und bei PSV-Dateien durch ein Pipe-Symbol.

Das einfachste Trennzeichen wäre das *Leerzeichen*. Das Leerzeichen ist als Trennzeichen nur dann geeignet, wenn es als Symbol in den Werten *nicht* vorkommen kann. Das ist bei vielen Daten nicht der Fall. Deshalb wird häufig das Tabulator-Symbol (`\t`) als Alternative zum Leerzeichen verwendet. Das Tabulator-Symbol ist ein nicht-druckbares Zeichen, das dem Leerzeichen ähnelt, aber seltener in Werten vorkommt. Dieses einfache Format wird als *Tab-separated values* (TSV) bezeichnet (Lindner, 1993).

Beispiel 9.2 (TSV-Daten)**.**

```
Name    Vorname Geburtsdatum    Geburtsort  Grösse
Müller  Hans 01.01.1990  Berlin   1.76
Untermayr Peter   01.01.1980  Wien 1.82
Osterwalder Urs    01.01.1970   Bern  1.78
```

Die Basis für die meisten anderen Separator-strukturierten Dateiformate ist das standardisierte CSV-Format (Shafranovich, 2005). Die Trennzeichen der CSV Spezifikation sind das Komma (`,`) und der Zeilenumbruch. Zusätzlich können die Werte in Anführungszeichen gesetzt werden, um die Werte zu schützen, falls diese ein Komma enthalten.

Beispiel 9.3 (CSV-Daten)**.**

```
Name,Vorname,Geburtsdatum,Geburtsort,Grösse
Müller,Hans,01.01.1990,Berlin,1.76
Untermayr,Peter,01.01.1980,Wien,1.82
Osterwalder,Urs,01.01.1970,Bern,1.78
```

Im deutschsprachigen Raum wird die CSV-Spezifikation oft abgewandelt. Dabei wird das Komma durch ein Semikolon (`;`) ersetzt. Dadurch kann das Komma als Dezimaltrennzeichen verwendet werden, ohne es zusätzlich durch Anführungszeichen geschützt zu werden. Diese Abwandlung wird als *Semikolon-separierte Werte* bezeichnet und ebenfalls als `CSV` abgekürzt.

Beispiel 9.4 (Semikolon-Daten)**.**

```
Name;Vorname;Geburtsdatum;Geburtsort;Grösse
Müller;Hans;01.01.1990;Berlin;1,76
Untermayr;Peter;01.01.1980;Wien;1,82
Osterwalder;Urs;01.01.1970;Bern;1,78
```

> **!** Achtung
>
> Weil beim `CSV`-Format zwei unterschiedliche Trennzeichen verwendet werden, treten beim Importieren oft Fehler auf, weil der Parser für das falsche Trennzeichen verwendet wurde. Bei der Arbeit mit `CSV`-Dateien sollte deshalb eine Datendatei auf das verwendete Trennzeichen geprüft werden. Diese Überprüfung kann entfallen, wenn das Trennzeichen dokumentiert wurde.

> **!** Achtung
>
> Beim Importieren von CSV-Dateien in der Schweiz muss zusätzlich das Dezimaltrennzeichen geprüft werden. Werden die Daten aus einer Excel-Version mit *deutschen Regionseinstellungen* exportiert, dann wird das Semikolon als Feldtrennzeichen und das Komma als Dezimaltrennzeichen verwendet. Werden die Daten aus einer Excel-Version mit *englischen Regionseinstellungen* exportiert, dann wird das Komma als Feldtrennzeichen und der Punkt als Dezimaltrennzeichen verwendet.
> In der Schweiz wird als Dezimaltrennzeichen wie im Englischen meistens ein Punkt verwendet. Werden Daten aus einer Excel-Version mit *schweizer Regionseinstellungen* werden die Trennzeichen gemischt: Als Feldtrennzeichen wird das Semikolon und als Dezimaltrennzeichen der Punkt verwendet. Das kann beim Import der Daten zu Fehlern führen, wenn das Dezimaltrennzeichen von einem Parser nicht angepasst wurde oder werden kann.

Das PSV-Format verwendet das Pipe-Symbol (`|`) als Trennzeichen. Während im CSV-Format Überschriften und Werte nicht klar unterschieden werden können, ist beim PSV-Format möglich. Dazu wird nach dem Überschriften eine Zeile mit Minuszeichen (`-`) und Trennzeichen eingefügt, um die Überschriften von den Werten abzugrenzen. Das PSV-Format wird oft als eingebettetes Format in Markdown-Dateien verwendet.

Beispiel 9.5 (PSV-Daten mit Überschriften)**.**

```
Name|Vorname|Geburtsdatum|Geburtsort|Grösse
----|-------|------------|----------|------
Müller|Hans|01.01.1990|Berlin|1.76
Untermayr|Peter|01.01.1980|Wien|1.82
Osterwalder|Urs|01.01.1970|Bern|1.78
```

Eine Erweiterung des PSV-Formats ist das sog. *Markdown-Tabellenformat*. Dieses Format erlaubt Zellenformatierungen für die Spalten. Dazu wird im Überschriftentrenner (`-`) ein Doppelpunkt (`:`) verwendet. Die Position des Doppelpunkts legt die Ausrichtung der Spalte fest. Fehlt ein Doppelpunkt oder steht ein Doppelpunkt am Anfang der Spalte formatiert die Spalte linksbündig, ein Doppelpunkt am Ende der Spalte formatiert die Spalte rechtsbündig und ein Doppelpunkt am Anfang und Ende der Spalte formatiert die Spalte zentriert.

Beispiel 9.6 (Markdown-Tabellenformat)**.**

```
Name|Vorname|Geburtsdatum|Geburtsort|Grösse
:---|:---:|---:|---|---:
Müller|Hans|01.01.1990|Berlin|1.76
Untermayr|Peter|01.01.1980|Wien|1.82
Osterwalder|Urs|01.01.1970|Bern|1.78
```

Das Ergebnis wird wie folgt dargestellt.

Name	Vorname	Geburtsdatum	Geburtsort	Grösse
Müller	Hans	01.01.1990	Berlin	1.76
Untermayr	Peter	01.01.1980	Wien	1.82
Osterwalder	Urs	01.01.1970	Bern	1.78

9.7. Excel-Arbeitsmappen

Excel-Arbeitsmappen (`xlsx`) fassen mehrere Tabellen zusammen. Excel-Arbeitsmappen werden als komprimiertes Dateiarchiv gespeichert. Deshalb wird das Format von EXCEL-Arbeitsmappen als *Binärformat* bezeichnet.

Anders als bei Textdateien in denen nur die Werte und deren Struktur kodiert wird, werden bei EXCEL-Arbeitsmappen auch *komplexe Formatierungen*, *Grafiken* und die *Berechnungen* kodiert (Carpenter, 2023). Die einzelnen Arbeitsblätter werden zwar als Mengenkodierung behandelt, das eigentliche Dateiformat folgt einer Baumstruktur.

Excel-Arbeitsmappen fassen mehrere benannte Arbeitsblätter zusammen. Ein Arbeitsblatt ist eine Struktur aus Zeilen und Spalten, die *tabellarisch* organisiert ist. Dadurch ist es möglich, dass mehrere *Tabellen* auf einem Arbeitsblatt vorhanden sind. Falls diese Tabellen nicht explizit als Datentyp `Tabelle` markiert wurden, muss diese Kodierung separat dokumentiert werden.

> Praxis
>
> Um Excel-Arbeitsmappen mit anderen Programmiersprachen zu verarbeiten muss die Struktur der Arbeitsmappe berücksichtigt werden. Daten sind immer nur über das Arbeitsblatt zugreifbar. Deshalb muss immer zuerst ein Arbeitsblatt ausgewählt werden, bevor auf die Daten zugegriffen werden kann.
> Arbeitsblätter haben immer zwei Modi: Den *Wertemodus* und den *Formelmodus*. Der gewünschte Moduls muss programmatisch festgelegt werden.
> Im Wertemodus werden die Werte der Zellen angezeigt. Im Formelmodus werden die Formeln der Zellen angezeigt. Wenn eine Zelle nur einen Wert aber keine Formel enthält, wird auch im Formelmodus der Wert bereitgestellt.

9.8. Markup-Formate

Separator-strukturierte Textdateien sind eine einfache Form der *Tabellenkodierung*. Diese Kodierung ist nur für einfache Datenstrukturen geeignet. Für komplexere Datenstrukturen reichen diese Formate nicht aus.

Ein Ansatz zur Speicherung von komplexen Datenstrukturen als Textdateien sind die sog. *Markup-Sprachen*. Markup-Sprachen sind Textdateien, mit denen Datenstrukturen durch Markierungen beschrieben werden. Sie sind als Datenbeschreibung eine Form der *Baumkodierung*. Dadurch lassen sich beliebig komplexe Datenstrukturen abbilden.

Die zwei am häufigsten benutzten Markupsprachen sind *XML* (Bray et al., 2008) und *HTML* (Hickson et al., o. J.). Beide Markupsprachen basieren auf dem gemeinsamen *SGML*-Standard (ISO/IEC JTC 1/SC 34, 1999). XML schränkt Datenorganisation ein, so dass Daten immer eindeutig abgebildet werden können. HTML stellt die Syntax für die Strukturierung von Webseiten bereit. Bei HTML stehen deshalb die Abbildung von Datenstrukturen nicht im Vordergrund.

Beide Formate verwenden sog. *Tags* als Markierungen im Text. Ein Tag wird durch eine öffnende spitze Klammer eingeleitet und eine schliessende spitze Klammer beendet. Das Tag hat einen Namen, welcher unmittelbar der öffnenden Klammer folgt. Tags können abgeschlossen werden, indem ein Schrägstrich (`/`) dem Tag-Namen vorangestellt wird.

Beispiel 9.7 (Markup-Tag)**.**

```
<tag-name></tag-name>
```

Tags können ausserdem *Attribute* enthalten. Attribute sind Schlüssel-Wert-Paare, die dem Tag zugeordnet sind. Die Werte von Attributen werden in der Regel in Anführungszeichen gesetzt.

Beispiel 9.8 (Markup-Tag mit einem Attribut)**.**

```
<tag-name name="Wert"></tag-name>
```

Tags können *geschachtelt* werden, wobei die Schachtelung durch die Reihenfolge der Tags festgelegt wird.

Beispiel 9.9 (Geschachtelte Tags)**.**

```
<tag-name><unter-tag>Wert</unter-tag></tag-name>
```

Beide Markup-Sprachen ignorieren Leerzeichen und Zeilenumbrüche. Dadurch können die Daten in einer Markup-Sprache beliebig formatiert werden, ohne die Bedeutung der Daten zu verändern. Entsprechend sind Beispiel 9.9 und Beispiel 9.10 gleichwertig.

Beispiel 9.10 (Geschachtelte Tags mit Zeilenumbrüchen und Leerzeichen)**.**

```
<tag-name>
   <unter-tag>
      Wert
   </unter-tag>
</tag-name>
```

Die Flexibilität von Markup-Sprachen hat die Folge, dass sie meistens sehr komplex sind und sich nicht direkt in die Datenstrukturen von Programmiersprachen übersetzen lassen. Deshalb wird für Markup-Sprachen ein sog. *Document-Object-Model* (DOM) definiert, damit die Daten in die für Programmiersprachen üblichen Datenstrukturen übersetzt werden können.

9.8.1. XML

XML wurde als flexibles Datenbeschreibungsformat konzipiert, dass an die Anforderungen unterschiedlicher Anwendungen angepasst werden kann. XML gibt die Formatierung von Datenserialisierungen vor. Die eigentlichen Datenstrukturen müssen über ein separates Schema spezifiziert werden. Deshalb ist XML eine *Metasprache* und keine Markup-Sprache im eigentlichen Sinn.

Ein XML-Tag steht dabei für eine Datenstruktur. Bei geschachtelten Strukturen erfordert XML, das die hierarchischen Abhängigkeiten explizit im Format markiert werden. Deshalb müssen geschachtelte Tags immer in der umgekehrten Reihenfolge geschlossen werden, in der sie geöffnet wurden.

Die Namen für XML-Tags dürfen frei gewählt werden, wobei Leerzeichen im Namen von Tags und Attributen verboten sind.

XML kann Werte als *Attribut* oder als *Inhalt* eines Tags kodieren. Die Wahl der Kodierung ist abhängig vom vorgegebenen Datenschema. Für neue Datenschema hat sich eingebürgert, Werte als Inhalte in hierarchischer Form zu kodieren.

Beispiel 9.11 (XML-Format einer tabellarischen Struktur in hierarchischer Form)**.**

```
<daten>
  <person>
   <name>Müller</name>
   <vorname>Hans</vorname>
   <geburtsort>Berlin</geburtsort>
   <geburtsdatum>01.01.1990</geburtsdatum>
   <grösse>1.76</grösse>
  </person>
  <person>
   <name>Untermayr</name>
```

```
    <vorname>Peter</vorname>
    <geburtsort>Wien</geburtsort>
    <geburtsdatum>01.01.1980</geburtsdatum>
    <grösse>1.82</grösse>
  </person>
  <person>
    <name>Osterwalder</name>
    <vorname>Urs</vorname>
    <geburtsort>Bern</geburtsort>
    <geburtsdatum>01.01.1970</geburtsdatum>
    <grösse>1.78</grösse>
  </person>
</daten>
```

Attribute sollten nur für die Kodierung von ergänzenden Daten verwendet. Zu solchen Daten ghört beispielsweise der Datentyp eines Wertes, weil XML alle Werte nur als Zeichenketten abbildet.

Beispiel 9.12 (XML-Format einer tabellarischen Struktur mit ergänzenden Attributen)**.**

```
<daten>
  <person>
    <name typ="text">Müller</name>
    <vorname typ="text">Hans</vorname>
    <geburtsort typ="text">Berlin</geburtsort>
    <geburtsdatum typ="datum">01.01.1990</geburtsdatum>
    <grösse typ="zahl">1.76</grösse>
  </person>
  <person>
    <name typ="text">Untermayr</name>
    <vorname typ="text">Peter</vorname>
    <geburtsort typ="text">Wien</geburtsort>
    <geburtsdatum typ="datum">01.01.1980</geburtsdatum>
    <grösse typ="zahl">1.82</grösse>
  </person>
  <person>
    <name typ="text">Osterwalder</name>
    <vorname typ="text">Urs</vorname>
    <geburtsort typ="text">Bern</geburtsort>
    <geburtsdatum typ="datum">01.01.1970</geburtsdatum>
    <grösse typ="zahl">1.78</grösse>
  </person>
</daten>
```

Die Kodierung von Werten als Attribute findet sich nur noch in älteren XML-Dokumenten.

Beispiel 9.13 (XML-Format einer tabellarischen Struktur mit Werten als Attribute)**.**

```
<daten>
  <person name="Müller" vorname="Hans" geburtsort="Berlin"
          geburtsdatum="01.01.1990" grösse="1.76"></person>
  <person name="Untermayr" vorname="Peter" geburtsort="Wien"
          geburtsdatum="01.01.1980" grösse="1.82"></person>
  <person name="Osterwalder" vorname="Urs" geburtsort="Bern"
          geburtsdatum="01.01.1970" grösse="1.78"></person>
</daten>
```

Aus den Beispielen wird deutlich, dass die Kodierung im XML-Format zwar sehr flexibel ist, aber auch sehr komplex werden kann und wenig speichereffizient ist. Das kann ein Problem werden, wenn viele Daten zwischen Rechnern übertragen werden müssen. Deshalb wird XML heute nur noch für Anwendungen verwendet, bei denen die Flexibilität der Kodierung wichtiger ist als die Speichereffizienz.

> **Hinweis**
>
> Eine Anwendung, die alle Daten im XML-Format speichert ist *Excel*. Diese Daten werden aber hinter den Kulissen in einem Binärformat gespeichert. Deshalb ist es nicht möglich, Excel-Arbeitsmappen direkt als XML-Dateien zu öffnen.

9.8.2. HTML

HTML wurde als Format für die Strukturierung von Webseiten entwickelt. Deshalb ist HTML eine echte *Markup-Sprache*, bei der die Namen der zulässigen Tags vorgegeben ist. Bei HTML steht dabei die Strukturierung von Webseiten im Vordergrund.

HTML ist für die Datenkodierung nur von Bedeutung, wenn Daten als Webseiten dargestellt werden sollen. Tabellarische Daten können auf Webseiten leicht identifiziert werden, weil es dafür spezielle HTML-Tags gibt.

Beispiel 9.14 (HTML-Format einer tabellarischen Struktur)**.**

```
<table>
  <tr>
    <th>Name</th>
    <th>Vorname</th>
    <th>Geburtsdatum</th>
    <th>Geburtsort</th>
    <th>Grösse</th>
  </tr>
  <tr>
    <td>Müller</td>
```

```
    <td>Hans</td>
    <td>01.01.1990</td>
    <td>Berlin</td>
    <td>1.76</td>
  </tr>
  <tr>
    <td>Untermayr</td>
    <td>Peter</td>
    <td>01.01.1980</td>
    <td>Wien</td>
    <td>1.82</td>
  </tr>
  <tr>
    <td>Osterwalder</td>
    <td>Urs</td>
    <td>01.01.1970</td>
    <td>Bern</td>
    <td>1.78</td>
  </tr>
</table>
```

Praxis

Der Datenaustausch über HTML-Tabellen ist inzwischen vernachlässigbar. HTML-Tabellen werden fast ausschliesslich für die Darstellung von Daten innerhalb von Anwendungen verwendet. Es gibt praktische keine Datenquellen mehr, deren Daten ausschliesslich als HTML-Tabellen vorliegen. Häufig werden Mess-Daten auch nicht mehr auf Web-Seiten bereitgestellt, sondern nur über Dateien in einem einem effizienteren Format verlinkt.

9.9. JSON

Mit der wachsenden Popularität des WWW wurde eine effizientere Form zur Kodierung komplexer Datenstrukturen erforderlich, um die Komplexität von Markupsprachen zu umgehen. Deshalb wurde die *JavaScript Object Notation* (JSON) entwickelt. JSON ist eine einfache Form der *Baumkodierung*. JSON ist eine *Untermenge* der JavaScript-Syntax und umfasst nur die Elemente zur Beschreibung von Datenstrukturen. Ein JSON-Dokument beschreibt deshalb immer eine gültige JavaScript-Datenstruktur. Deshalb kann JSON von JavaScript-Programmen in die üblichen Datenstrukturen überführt werden. Das vereinfacht die Programmierung und die Interaktivität von Anwendungen, die in Webbrowsern ausgeführt werden.

JSON ist ein standardisiertes Datenformat (Bray, 2017) und wird inzwischen von allen

wichtigen Programmiersprachen unterstützt. Auch JSON ist ein *Markup-Format*, bei dem die Markierungen der JavaScript-Syntax folgen.

JSON kennt nur sechs Datentypen, wodurch die Kodierung sehr einfach umzusetzen ist. Die Datentypen sind:

- Zahlen
- Zeichenketten
- Boole'sche Werte
- Listen
- Objekte
- `null` (undefiniert)

Zeichenketten müssen immer durch doppelte Anführungszeichen (`"`) eingeleitet und abgeschlossen werden.

Der Listendatentyp wird durch Block-Klammern (`[` und `]`) markiert.

Der Objekt-Datentyp gibt dem Format seinen Namen. Objekte werden durch geschweifte Klammern (`{` und `}`) markiert und sind eine *Menge* von *Schlüssel-Wert-Paaren*. Die Schlüssel sind Zeichenketten, wobei jeder Schlüssel nur einmal pro Objekt vorkommen darf. Die Werte können von beliebigen JSON-Datentypen sein. Schlüssel und Werte sind immer durch einen Doppelpunkt getrennt.

Tabellarische Strukturen lassen sich in JSON auf zwei Arten abbilden:

- Als Liste von Objekten
- Als Objekt mit Listen

Die erste Variante ist die gebräuchlichste, weil beim die Bedeutung der zusammenhängenden Werte durch die Schlüssel meistens einfacher verarbeitet werden kann.

Beispiel 9.15 (Tabellenstruktur im JSON-Format als Liste von Objekten)**.**

```
[
    {
        "name": "Müller",
        "vorname": "Hans",
        "geburtsdatum": "01.01.1990",
        "geburtsort": "Berlin",
        "grösse": 1.76
    },
    {
        "name": "Untermayr",
        "vorname": "Peter",
        "geburtsdatum": "01.01.1980",
        "geburtsort": "Wien",
        "grösse": 1.82
```

```
    },
    {
        "name": "Osterwalder",
        "vorname": "Urs",
        "geburtsdatum": "01.01.1970",
        "geburtsort": "Bern",
        "grösse": 1.78
    }
]
```

Die Objekt-mit-Listen Strukturierung eignet sich, wenn die Vektorstruktur der Daten im Vordergrund steht. In diesem Fall werden die Werte in einer Liste zusammengefasst und die Schlüssel werden als Index verwendet.

Beispiel 9.16 (Tabellenstruktur im JSON-Format als Objekt mit Listen)**.**

```
{
    "name": ["Müller", "Untermayr", "Osterwalder"],
    "vorname": ["Hans", "Peter", "Urs"],
    "geburtsdatum": ["01.01.1990", "01.01.1980", "01.01.1970"],
    "geburtsort": ["Berlin", "Wien", "Bern"],
    "grösse": [1.76, 1.82, 1.78]
}
```

> **Warnung**
>
> Die Objekt-mit-Listen-Strukturierung ist weniger verbreitet, weil diese Strukturierung den Erhalt der Zusammenhänge zwischen den Werten nicht sicherstellt. Es wäre also denkbar, die Werte einer Liste in einer anderen Reihenfolge neu anzuordnen, ohne dass über die Positionen zusammenhängenden Werte ebenfalls angepasst werden.
> Dieses Problem besteht bei der Liste-von-Objekten-Strukturierung nicht.

9.10. YAML

In den letzten Jahren findet eine weitere Formatierung immer häufigere Verwendung: YAML (Ben-Kiki et al., 2021). YAML ist eine einfache "menschenfreundliche" Formatierung, die auf der Einrückung von Textblöcken basiert. YAML ist ein Obermenge von JSON und deshalb ist jedes JSON-Format gleichzeitig ein gültiges YAML-Format. Das Ziel des YAML-Formats ist das vereinfachte Erstellen und Bearbeiten von strukturierten Datendokumenten durch Menschen.

YAML hat die gleichen Datentypen wie JSON und folgt den gleichen Prinzipien zur Datenkodierung. Dadurch hat YAML die gleichen Vorteile wie JSON. YAML erlaubt

zusätzlich die Verwendung von Kommentaren, die in JSON nicht vorgesehen sind. Solche Kommentare dienen nur der Dokumentation für die menschliche Interaktion und werden beim Laden eines YAML-Dokuments ignoriert.

Der Aufälligste Unterschied zwischen YAML und JSON ist die Verwendung von Einrückungen zur Strukturierung von Daten. Dadurch ist YAML für Menschen etwas leichter lesbar und verständlich. Auch müssen Zeichenketten meistens nicht explizit markiert werden. YAML legt Regeln zur automatischen Erkennung des Datentyps fest. So müssen nur in Ausnahmefällen Zeichenketten explizit markiert werden, was bspw. zur Unterscheidung von Wahrheitswerten und Zeichenketten müssen Zeichenketten explizit markiert werden, wenn eine Zeichenkette nur aus einem Schlüsselwort für einen Boole'schen Wert besteht. Zeichenketten müssen auch mit doppelten Anführungszeichen (") markiert werden, wenn die Zeichenkette einen Doppelpunkt enhält.

Beispiel 9.17 (Zeichenkettenmarkierung in YAML)**.**

```yaml
# Zeichenkette mit Doppelpunkt
- "Müller: Hans"
# Objekt mit Schlüssel "Müller" und Wert "Hans"
- Müller: Hans
# Zeichenkette mit Wahrheitswert
- "true"
- "no"
# Wahrheitswerte werden intern in Wahr und Falsch umgewandelt
- true
- no
```

Weil YAML die Objektstruktur visuell unterstützt, erfolgt eine Datenkodierung von tabellarischen Strukturen in der Regel über Listen von Objekten.

Beispiel 9.18 (Tabellenstruktur im YAML-Format als Liste von Objekten)**.**

```yaml
- name: Müller
  vorname: Hans
  geburtsdatum: 01.01.1990
  geburtsort: Berlin
  grösse: 1.76
- name: Untermayr
  vorname: Peter
  geburtsdatum: 01.01.1980
  geburtsort: Wien
  grösse: 1.82
- name: Osterwalder
  vorname: Urs
  geburtsdatum: 01.01.1970
  geburtsort: Bern
```

```
    grösse: 1.78
```

9.11. Dateiformate und Versionierung

Bei der Versionierung von Daten spielt das Dateiformat eine zentrale Rolle.

Daten in separator-strukturierte Textdateien oder im YAML-Format lassen sich besonders einfach versionieren, weil sie als zeilenorientierte Formate direkt von den Versionierungssystemen unterstützt werden.

Bei XML- und JSON-Daten ist die Situation etwas komplexer: Diese Formate sind nicht zeilenorientiert und die gleiche Struktur kann auf unterschiedliche Arten im gleichen Format serialisiert werden. Dadurch wird die Versionierung dieser Dateiformate erschwert.

Die Versionierung von Binärformaten ist noch komplexer, weil Versionierungssystem diese Formate nicht auf der Inhaltsebene unterstützen. Deshalb ist es üblich, Daten in diesem Formaten vor der Versionierung in ein zeilenorientiertes Format überführt werden. Das betrifft sehr häufig Excel-Arbeitsmappen, weil sich die Binärdaten bei jedem *Zugriff* ändern, ohne dass die Daten selbst verändert wurden.

> **Praxis**
>
> Es ist üblich Daten anstatt in Excel-Arbeitsmappen in CSV-Dateien zu speichern, um die Daten versionieren zu können. Die CSV-Dateien werden dann in Excel als externe Daten importiert.

Teil III.

Mathematik der Daten

10. Variablen, Funktionen und Operatoren

10.1. Variablen

Bei der Datenverarbeitung müssen Daten auch gespeichert werden, um sie wiederzufinden. Dazu werden die Daten mit einem Bezeichner versehen, über den auf die Daten zugegriffen werden kann. Diese Bezeichner sind in der Folge mit dem zugewiesenen Wert gleichwertig.

Definition 10.1. Ein Bezeichner ist ein Platzhalter für Werte. Ein Bezeichner hat immer einen *eindeutigen Namen* in einem Geltungsbereich.

Der Begriff *Wert* ist hier *möglichst allgemein* zu denken. Ein Wert kann ein fundamentaler Datentyp oder eine Datenstruktur sein. Dateien oder `git`-Versionen können ebenfalls als Werte behandelt werden.

Bezeichner sind in einem Geltungsbereich immer *eindeutig*. Das bedeutet, dass Bezeichner mit dem gleichen Namen auf den gleichen Wert verweisen. Ein Bezeichner kann nicht gleichzeitig auf zwei Werte verweisen. Das Wort *Geltungsbereich* bezieht sich auf dem Zusammenhang, in dem ein Bezeichner verwendet wird. Die Definition besagt daher, dass ein Bezeichner genau einmal im gleichen Zusammenhang verwendet werden darf. Deshalb ist es wichtig, den Geltungsbereich von Bezeichnern genau zu definieren.

> **i** Hinweis
>
> Die Definition eines Geltungsbereichs ist in den meisten Programmiersprachen und -Umgebungen genau festgelegt.

Mit Bezeichnern kann wie mit Werten gearbeitet werden, weil ein Bezeichner für einen Wert steht. Ein Bezeichner kann einen einzelnen Wert oder eine beliebige komplexe Datenstruktur aufnehmen. Ein Bezeichner kann auf einen einzelne Zahl oder Zeichenkette genauso verweisen, wie auf einen Vektor, eine Liste oder eine Matrix. Bezeichner sind nicht auf die fundamentalen Datentypen beschränkt. Z.B. ist ein *Dateiname* ein Bezeichner für die Daten in einer Datei oder ein `git`-Tag ist ein Bezeichner für eine bestimmte Version eines Projektes. Durch den Bezeichner wird vom eigentlichen Wert *abstrahiert*.

Damit ein Wert über einen Bezeichner *abstrahiert* werden kann, muss er *benannt* werden. Das Benennen eines Wertes wird *Zuweisen* genannt. Der Wert wird also einem Bezeichner *zugewiesen*. Eine Zuweisung wird in der mathematischen Notation über einen Pfeil ($\rightarrow$) dargestellt. Dabei zeigt der Pfeil vom Bezeichner auf den Wert. Diese Schreibweise bedeutet,

dass ein Bezeichner auf den angegebenen Wert *verweist*. Die Formel 10.1 zeigt die Zuweisung des Wertes 12 an den Bezeichner $Wert$.

$$Wert \rightarrow 12 \tag{10.1}$$

Es werden zwei Arten von Bezeichnern unterschieden:

- unveränderliche Bezeichner
- veränderliche Bezeichner

Definition 10.2. Eine **Konstante** ist ein *unveränderlicher* Bezeichner.

Konstanten verweisen in ihrem Geltungsbereich immer auf den gleichen Wert. Einer Konstante kann nur einmal ein Wert zugewiesen werden.

Definition 10.3. Eine **Variable** ist ein veränderlicher Bezeichner.

Variablen kann im gleichen Geltungsbereich mehrmals ein Wert zugewiesen werden. Welcher Wert gerade für eine Variable gültig ist, hängt vom Zeitpunkt einer Verarbeitung ab.

10.2. Funktionen

Funktionen bilden die Grundlage für die Arbeit mit Daten. Funktionen legen fest, wie zu einem bestimmten Ergebnis gelangt wird.

> **Merke**
>
> Eine Funktion erzeugt Ergebnisse.

Alle Ergebnisse einer Funktion sind die *Funktionseffekte*. Nicht alle Effekte einer Funktion sind erwünscht. Solche unerwünschten Effekte werden *Nebeneffekte* genannt.

Ist das Ergebnis einer Funktion ein Wert, dann heisst dieser Wert, der **Rückgabewert** der Funktion. Eine Funktion kann mehr als einen Effekt haben. Entsprechend muss das Ergebnis einer Funktion in einer **Rückgabeliste** zusammengefasst werden.

Wie eine Funktion die Effekte erzeugt, wird im **Funktionskörper** festgelegt.

Manche Funktionen benötigen Eingaben, um daraus die Ergebnisse zu erzeugen.

Definition 10.4. Die Eingaben einer Funktion heissen **Parameter** oder **Funktionsargumente**. Eine Funktion **akzeptiert** diese Parameter als Eingabe.

Parameter sind spezielle *Variablen* mit dem Geltungsbereich des Funktionskörpers.

Eine Funktion kann natürlich auch mehrere Parameter haben. Die Parameter einer Funktion werden als **Parameterliste** bezeichnet. Eine solche Parameterliste darf auch eine leere Liste sein.

Die Datentypen der Parameter legen den *Definitionsbereich* einer Funktion fest. Der Datentyp der Rückgabe werde legt den *Zielbereich* fest. Eine Funktion stellt also eine *Beziehung* zwischen dem Definitionsbereich und dem Zielbereich her.

Eine Funktion, die keine Parameter akzeptiert, wird ist eine **parameterlose Funktion**. Eine parameterlose Funktion hat deshalb eine leere Parameterliste als Eingabe.

So lässt sich die Funktionsdefinition erweitern und präzisieren:

Definition 10.5. Eine Funktion erzeugt Effekte im Zielbereich mithilfe einer Parameterliste mit Werten im Definitionsbereich.

Formal lässt sich diese Defition wird der durch die Rückgabeliste definierte Zielbereich dem Bezeichner E zugewiesen und den Defitionsbereich der Parameterliste mit P gekennzeichnet, dann ist entsprechend dieser Definition **jede** Funktion entsprechend Formel 10.2 definiert.

$$f : P \to E \tag{10.2}$$

Diese Formel wird wie folgt gelesen: *Die Funktion f bildet den Definitionsbereich P auf den Zielbereich E ab.* Die Formel kann auch analog zur Definition von Bezeichnern gelesen werden: *Für die Funktion f verweisen alle Werte im Definitionsbereich P auf Werte im Zielbereich E.* Diese Beziehung zwischen Definitionsbereich P und Zielbereich E legt die Funktion fest. f ist der *Bezeichner* der Funktion. Durch den Bezeichner und den Definitionsbereich der Parameterliste ist eine Funktion eindeutig beschrieben.

Diese Funktionsdefinition benötigt **keinen** *Funktionskörper*. Tatsächlich können die gleichen Ergebnisse auf unterschiedlichen Wegen aus den gleichen Parametern erzeugt werden. Das kann bspw. durch die Verwendung unterschiedlicher Programmiersprachen sein.

Definition 10.6. Wenn zwei Funktionen für die gleichen Parameter das gleiche Ergebnis erzeugen, dann sind diese Funktionen **funktional gleich**.

Um eine Funktion zu verwenden, muss diese *aufgerufen* werden. Ein *Funktionsaufruf* wird meistens durch eine rundes Klammerpaar dargestellt. Wird eine Funktion aufgerufen, wird der Funktionskörper mit den Werten in den Parametern ausgeführt. Der Funktionsaufruf $f(x)$ wählt für einen Wert x aus dem Definitionsbereich P den entsprechenden Wert y aus dem Zielbereich E aus.

Manche Programmiersprachen ermöglichen die Verwendung des gleichen Bezeichners für unterschiedliche Definitions- und Zielbereiche.

Definition 10.7. Hat ein Funktionsbezeichner mehrere Definitionsbereiche, dann heisst diese Funktion **polimorph** (gr. *poli*: viel, gr. *morphos*: Gestalt).

Bei polimorphen Funktionen muss der Funktionsbezeichner und die Parameterliste gemeinsam eindeutig sein.

In den Datenwissenschaften kommen im Wesentlichen drei Arten von Funktionen zum Einsatz:

1. Transformationen, welche einen Wert in einen anderen Wert *überführen*;
2. Aggregatoren, die mehrere Werte *zusammenfassen*; und
3. Generatoren, mit denen Werte *erzeugt* werden.

10.2.1. Spezielle Funktionen

10.2.1.1. Konstante Funktionen

Definition 10.8. Eine **konstante Funktion** erzeugt für jede Eingabe immer den gleichen Ergebniswert w.

Konstante Funktionen sind formal wie in Formel 10.3 definiert:

$$f_{kW}(x) \to W \tag{10.3}$$

Das bedeutet, dass unabhängig von der Eingabe immer der Wert von W als Ergebnis folgt.

10.2.1.2. Die Identitätsfunktion

Die nächste Funktion ähnelt in ihrer Definition konstanten Funktionen.

Definition 10.9. Die **Identitätsfunktion** gibt für eine Eingabe diese Eingabe als Ergebnis zurück.

Die formale Definition der Identitätsfunktion zeigt Formel 10.4.

$$f_{id}(x) \to x \tag{10.4}$$

Im Unterschied zu konstanten Funktionen ist das Ergebnis kein konstanter Wert, sondern der Wert der Parameter. Auf den ersten Blick erscheint diese Funktion sinnlos, denn sie verändert die Eingabeparameter nicht. Diese Funktion erfüllt in der Mathematik und in den Datenwissenschaften die gleiche Bedeutung wie die Zahl `1` für die Multiplikation oder die Zahl `0` für die Addition.

Die Identitätsfunktion ist umkehrbar. Die Umkehrfunktion der Identitätsfunktion ist die Identitätsfunktion selbst.

10.2.1.3. Projektionen

Definition 10.10. Eine **Projektion** bezeichnet eine Funktion, die den gleichen Eingaben **immer** die *gleichen Ergebnisse* zuweist.

Definition 10.11. Eine *Projektion* heisst **umkehrbar**, wenn eine zweite Funktion existiert, welche die Rückgabeliste der ersten Funktion als Parameterliste hat und aus dieser die Parameterliste der ersten Funktion als Rückgabeliste erzeugt. Diese zweite Funktion ist die **Umkehrfunktion** der ersten Funktion.

Die Umkehrfunktion einer Funktion f wird als f^{-1} geschrieben.

Aus diesen beiden Definitionen folgt im Umkehrschluss, dass nicht jede Funktion eine Projektion und nicht jede Projektion umkehrbar ist.

10.3. Operatoren

Aus den Grundrechenarten sind Operatoren bekannt. Diese Operatoren sind z.B. `+`, `-`, `*` und `/`. Diese Operatoren sind in der Mathematik und den Datenwissenschaften *Funktionen*.

Definition 10.12. Eine **Operation** bezeichnet die Anwendung eines Operators. Eine Operation besteht aus einem Operator und den Operanden.

Ein **Operator** ist eine alternative Schreibweise für häufig verwendete *Funktionen*.

Die *meisten* Operatoren sind unär oder binär. Unäre Operatoren haben nur einen Parameter. Ein unärer Operator ist das Vorzeichen `-` oder das `%`-Zeichen. Binäre Operatoren haben zwei Parameter. Alle Grundrechenarten sind binäre Operatoren.

Bei Operatoren wird zwischen *Präfix*, *Infix* und *Postfix* unterschieden. Präfix-Operatoren stehen vor den Parametern, Infix-Operatoren zwischen den Parametern und Postfix-Operatoren stehen nach den Parametern. Die Grundrechenarten sind Infix-Operatoren.

10.3.1. Precedence - Prioriät von Operatoren

In komplexen Operationen ist nicht immer eindeutig, welche Parameter zu welchem Operator gehören. Damit richtigt gerechnet wird, besteht zwischen den Operatoren eine Hierachie, in welcher jeder Operator eine bestimmte Priorität für die Ausführung hat. Das bedeutet, dass ein Operator mit höherer Priorität vor einem Opeartor mit niedrigerer Priorität ausgeführt wird. Bei Operatoren mit gleicher Priorität wird **immer** der am weitesten links stehende Operator zuerst ausgeführt.

Um einen Operator in der Anwendung vorzureihen, werden Klammern verwendet. Klammern legen die Reihenfolge der Operationen fest.

Eine wichtige Hierarchie für die Grundrechenarten ist die sog. KEPS-Regel. Diese Regel legt die Reihenfolge der Operationen fest. Die KEPS-Regel ist eine Abkürzung für die Operationen:

1. **K**lammern
2. **E**xponenten
3. **P**unktrechnung
4. **S**trichrechnung.

Diese Abkürzung dient als Eselsbrücke für die Prioritäten der Grundrechenarten. Sind zwei Operationen auf der gleichen Hierarchiestufe, dann werden die Operationen von links nach rechts ausgeführt.

Exponenten werden in der Mathematik in der Regel hochgestellt dargestellt. Die meisten Programmiersprachen unterstützen keine Formatierung von Formeln als Kennzeichnung von Operatoren. Deshalb wird der Potenz-Operator mit einem expliziten Operator dargestellt. Meist ist dies der ^-Operator.

> **Achtung**
>
> Die Programmiersprache Python weicht von dieser Konvention ab und verwendet den **-Operator für Exponenten. Der ^-Operator ist in Python der sog. *bitweise XOR-Operator*. (Python Software Foundation, 2013)

10.3.2. Substitution

> **Hinweis**
>
> Wird ein Teil einer komplexen Operation in eine einfachere Funktion ausgelagert, dann spricht man von **Substitution**.

In der Mathematik wird beim Substituieren die Teiloperation in der Regel nur durch Bezeichner der ersetzenden Funktion angegeben. In der Praxis müssen solche Ersetzungsfunktionen explizit parametrisiert und ausgeführt werden.

Die folgenden beiden Beispile illustrieren die Substituion mit einfachen mathematischen Funktionen.

Die Operation in Formel 10.5 führt zwei Operationen nacheinander aus.

$$f(a,b) \rightarrow (a-b)^2 \tag{10.5}$$

Wird die Teiloperation $a-b$ substituiert, dann ergibt sich Formel 10.6.

$$f(a,b) \rightarrow s^2; s = a-b \tag{10.6}$$

In diesem Beispiel wurde nur die Subtraktion substituiert. Weil der Bezeichner s die Subtraktion versteckt, können die Klammern der ursprünglichen Operation entfallen.

Durch Substituieren lassen sich häufig die grundsätzlichen Ideen von komplexen Operationen leichter erkennen und vereinfachen. Das Beispiel in Formel 10.7 zeigt diesen Effekt.

$$g(a,b) \rightarrow \frac{a+b-2ab}{b-a} \Leftrightarrow -\frac{(a-b)^2}{a-b} \tag{10.7}$$

Für die rechte Operation wird in Formel 10.8 wieder die Substitution $s = a-b$ verwendet.

Bei einer Substitution sollen möglichst viele Teilterme mit dem gleichen Bezeichner substituiert werden. Deshalb wurde der *Nenner* $b-a$ so umgeformt, dass auch im Nenner der Term $a-b$ steht. Dazu wurde ausgenutzt, dass gilt $b-a \Leftrightarrow -(a-b)$. Das Vorzeichen kann in diesem Fall vor dem Bruch gezogen werden, so dass die Klammern um $a-b$ entfallen können.

$$g(a,b) \rightarrow -\frac{s^2}{s} \Leftrightarrow -s; s = a-b \tag{10.8}$$

Der Bezeichner der Substitution unterscheidet sich nicht von anderen Bezeichnern. Es kann also angenommen werden, dass s ein beliebiger Wert ist. So lässt sich leicht erkennen, dass s gekürzt werden kann. Rein technisch wurde für den Substitutionsterm eine Funktion s mit zwei Parametern erstellt. Diese Funktion ist in diesem Beispiel identisch mit der Funktion `subtraktion()`, mit der ein Wert von einem anderen abgezogen wird.

Um die ursprüngliche Operation zu erhalten, muss die Substitution rückgängig gemacht werden. Dazu wird der substituierte Bezeichner durch die ursprüngliche Teiloperation ersetzt. Daraus ergibt sich Formel 10.9.

$$g(a,b) \rightarrow -(a-b) \Leftrightarrow b-a \tag{10.9}$$

Diese Operation $b-a$ ist wesentlich einfacher als die ursprüngliche Funktion. Solches Vereinfachen von Funktionen ist für die Praxis wichtig, weil dadurch zum einen die Komplexität von Operationen verringert werden kann und zum anderen die Ausführungsgeschwindigkeit durch den Wegfall von Operationen erhöht werden kann.

10.3.3. Besondere Operatoren

10.3.3.1. Divisions-Operatoren

Unter den Grundrechenarten ist die Division ein Unikum. Als Einzige der Grundrechenarten kann das Ergebnis einen anderen Zahlentyp haben als die Operanden. Beispielsweise ergeben sich aus ganzen Zahlen rationale Zahlen und aus reellen Zahlen können sich ganze oder rationale Zahlen ergeben. Daraus ergibt sich, dass die Division über einen kontinuierlichen Wertebereich arbeitet.

Die Division kann in zwei Unteroperationen gegliedert werden:

- Der **Ganzzahldivision**
- Dem **Modulo**

Für beide Operationen gilt, dass das Ergebnis von gleichem Zahlentyp ist, wie die Operanden.

Die *Ganzzahldivision* hat den ganzzahligen Anteil einer Division als Ergebnis. Damit ist der Wert vor dem Komma gemeint. Diese Operation ähnelt dem Runden, allerdings wird nur der Nachkommateil entfernt.

Die Ganzzahldivision eignet sich zur Gruppierung von Werten in gleich grosse Blöcke. Ein *Block* ist dabei ein Intervall im Wertebereich der Variablen. Die Grösse eines Blocks wird durch den Divisor festgelegt.

Weil die Ganzzahldivision für mehrere Werte in einem Intervall das gleiche Ergebnis hat, wird die Ganzzahl mit der Intervallschreibweise gekennzeichnet.

Beispiel 10.1 (Ganzzahldivision)**.**

$$\lfloor \frac{6}{4} \rfloor = 1$$

Die *Modulo*-Operation gibt den Rest nach der Ganzzahldivision zurück. Das Ergebnis von Modulo ist die *Differenz des Dividenden* bis zum Ergebnis der Ganzzahldivision mulipliziert mit dem Divisor.

Beispiel 10.2 (Modulo-Operation)**.**

$$mod(6,4) = 6 - 4 \cdot \lfloor \frac{6}{4} \rfloor = 2$$

Modulo erzeugt nur Werte im Intervall $0 \leq m < |d|$, wobei m das Ergebnis von Modulo und d der Divisor ist. Auf diese Weise lassen sich beliebige Werte auf ein Intervall mit fester Länge d projizieren.

> **Praxis**
>
> Die Ganzzahldivision und Modulo werden häufig als einfache und effiziente Gruppiereungsoperatoren verwendet.

10.3.3.2. Der Zuweisungsoperator

Der Zuweisungsoperator ist ein besonderer Operator. Er ist ein binärer Operator, der einen Wert einem Bezeichner zuweist. Die meisten Programmiersprache verwenden dafür den `=`-Zeichen. Dabei steht der Bezeichner links und der Wert rechts vom Operator.

Manche Programmiersprachen unterstützen gerichtete Zuweisungsoperatoren, bei denen der Bezeichner rechts vom Operator steht. Diese gerichteten Zuweisungsoperatoren sind in der Regel eine Abkürzung für eine Operation mit dem Bezeichner und dem Wert.

Der Zuweisungsoperator ist zu allen anderen Operatoren nachrangig. Das bedeutet, dass der Zuweisungsoperator immer als letzter Operator einer Operation ausgeführt wird.

> **Tipp**
>
> Der Zuweisungsoperator ist beim Programmieren wichtig, deshalb sollte man sich beim Erlernen einer Programmiersprache unbedingt mit allen Formen dieses Operators vertraut machen.

10.3.3.3. Der Anwendungsoperator

Werden Klammern unmittelbar nach einem Funktionsbezeicher angegeben, dann ändern diese Klammern ihre Bedeutung, indem sie die *Anwendung* der bezeichneten Funktion erfordern. Mithilfe des Anwendungsoperators kann zwischen der Anwendung einer Funktion und ihrem Bezeichner unterschieden werden.

> **Warnung**
>
> Der Anwendungsoperator kann ausschliesslich mit Bezeichnern verwendet werden, die auf eine Funktion verweisen.

Die Funktion $addieren(a, b)$ hat den Bezeichner $addieren$ und die Parameter a und b. Der Bezeichner $addieren$ ohne den Anwendungsoperator (`()`), verweist auf die Funktion selbst. So kann die gleiche Funktion einem anderen Bezeichner zugewiesen werden, bspw. dem Bezeichner $zusammenzhlen$. Es gilt dann Formel 10.10.

$$\begin{aligned} addieren(a, b) &\rightarrow a + b \\ zusammenzhlen &\rightarrow addieren \end{aligned} \tag{10.10}$$

Nun können die beiden Bezeichner mit dem Anwendungsoperator verwendet werden, um die *Ergebnisse* der Addition zu erhalten.

$$\begin{aligned} addieren(1, 2) &= 3 \\ zusammenzhlen(2, 3) &= 5 \\ zusammenzhlen(2, 1) &= 3 \end{aligned}$$

10.3.3.4. Der Index-Operator

In der Mathematik werden Idicees durch tieferstellen gekennzeichnet. Ein Index wird dann verwendet, wenn ein Bezeichner auf eine Datenstruktur verweist. Der Index bezeichnet den Wert an der entsprechende Position in der Datenstruktur. Am Beispiel des Summe-Operators wird dies in Formel 10.11 verdeutlicht.

$$\sum_{i=1}^{n} x_i \cdot y_i \tag{10.11}$$

Das Symbol Σ ist der Summe-Operator. Dieser Operator addiert alle Werte im angegebenen Bereich. Hier ist das der Bereich $i = 1$ bis n, wobei n für die Anzahl der Werte in einer Datenstruktur steht. In diesem Fall werden zwei Listen `x` und `y` verwendet. Das tiefergestellte `i` bei x_i und y_i ist der rechte Operand des Indexoperators. Das `i` ist weiterhin ein Bezeichner. Die Summe-Funktion zählt diesen Wert hoch, bis der Wert n erreicht ist. Dieser Index gibt an, welche Werte aus den Datenstrukturen x und y addiert werden sollen. Weil der Bezeichner `n` nicht explizit angegeben ist, wir dieser als Konstante für die Anzahl der Werte in den Datenstrukturen angenommen. Weil der Wert von `n` bei zwei Datenstrukturen mehrdeutig sein *könnte*, bedeutet diese Schreibweise, dass beide Datenstrukturen gleich lang sind.

10.4. Das neutrale Element

Manche Funktionen haben die besondere Eigenschaft, dass es einen Wert im Definitionsbereich gibt, der für alle Werte im Definitionsbereich das gleiche Ergebnis *unabhängig* davon erzeugt, als welcher Operand dieser Wert verwendet wird. Dieser Wert wird als das **neutrales Element** des Operators bezeichnet. Für die Addition ist das die Zahl `0` und für die Multiplikation ist das die Zahl `1`.

Definition 10.13. Das **neutrale Element** eines Operators ist ein Wert im Definitionsbereich, der für **alle** Werte im Definitionsbereich des jeweils anderen Operanden diesen Wert als Ergebnis erzeugt.

Ist der Definitionsbereich durch den Datentyp Zahl festgelegt, dann muss das neutrale Element genau eine Zahl sein.

Gemäss dieser Definition muss für das neutrale Element die Gleichung Formel 10.12 gelten.

$$e_n \circ x = x \circ e_n = x \tag{10.12}$$

e_n ist hier kein Index, sondern der Bezeichner für das neutrale Element. $\circ$ ist ein Operator, für den ein neutrales Element geprüft werden soll. Dieses Symbol ist ein Bezeichner für einen geeigneten Operator. x ist der jeweils andere Operand. e_n ist für den jeweiligen Operator *konstant*. D.h. Es muss genau einen Wert geben, für den mit **allen** anderen Werten des Definitionsbereichs, inklusive sich selbst, diese Gleichung erfüllt ist.

10.4.1. Überprüfung

Das neutrale Element einer Operation muss auf beiden Positionen des zugehörigen Operators immer den jeweilig anderen Operanden als Ergebnis erzeugen, so dass Formel 10.12 gilt.

Für die Grundrechenarten lässt sich das neutrale Element durch Umformen der Definition des neutralen Elements bestimmen. Formel 10.13 zeigt das für die Addition.

$$\begin{aligned} & e_n + a = a \\ \Leftrightarrow \quad & e_n = a - a \\ \Leftrightarrow \quad & e_n = 0 \end{aligned} \tag{10.13}$$

Dieses Ergebnis wird durch Einsetzen in den restlichen Teil der Definition in Formel 10.14 überprüft.

$$\begin{aligned} & e_n + a = a + e_n \\ \Leftrightarrow \quad & 0 + a = a + 0 \\ \Leftrightarrow \quad & a = a + 0 - 0 \\ \Leftrightarrow \quad & a = a \end{aligned} \tag{10.14}$$

Die gleichen Überlegungen lassen sich auf andere Operationen, wie z.B. der Division anwenden. Das wird in Formel 10.15 gezeigt.

$$\begin{aligned} & a : e_n = a \\ \Leftrightarrow \quad & \frac{a}{e_n} = a \\ \Leftrightarrow \quad & e_n = \frac{a}{a} \\ \Leftrightarrow \quad & e_n = 1 \end{aligned} \tag{10.15}$$

Zum Überprüfen wird dieses Ergebnis entsprechend eingesetzt, wie Formel 10.16.

$$\begin{aligned} & a : e_n = e_n : a \\ & a : 1 = 1 : a \\ \Leftrightarrow \quad & a = 1 : a \\ \Leftrightarrow \quad & a \cdot a = 1 \\ \Leftrightarrow \quad & a^2 = 1 \end{aligned} \tag{10.16}$$

Dieses Ergebnis gilt nicht für alle möglichen Werte von a, sondern nur für den einen Fall $a = 1$. Damit kann die Lösung $e_n = 1$ nicht das neutrale Element der Division sein. Möglicherweise liefert die andere Teilgleichung ein besseres Ergebnis.

$$
\begin{aligned}
& e_n : a = a \\
\Leftrightarrow \quad & \frac{e_n}{a} = a \\
\Leftrightarrow \quad & e_n = a \cdot a \\
\Leftrightarrow \quad & e_n = a^2
\end{aligned}
\tag{10.17}
$$

Auch dieser Weg führt zu keinem Ergebnis für das neutrale Element, weil e_n *ein* Element des Definitionsbereich sein muss. In diesem Fall muss dieser Wert eine Zahl sein. Die Lösung $e_n = a^2$ liefert für unterschiedliche Werte für a verschiedene Werte für e_n kann deshalb auch keine Lösung für das neutrale Element sein.

Aus diesen Betrachtungen folgt, dass die Division kein neutrales Element hat, weil über beide Definitionen des neutralen Elements kein eindeutiges Ergebnis für die Division gefunden wurde.

> **Merke**
>
> Nicht jede Operation hat ein neutrales Element.

10.4.2. Redundanz

Für das *neutrale Element* eines Operators ist die Operation mit der Identitätsfunktion funktional gleich.

Definition 10.14. Ergeben mehrere Teiloperationen **immer** das neutrale Element für die nächste Operation, dann sind diese Operationen **redundant**.

> **Merke**
>
> *Redundante* Operationen können aus einer Operation entfernt werden, ohne dass sich das Ergebnis ändert.

> **Praxis**
>
> Redundante Operationen sollten in der Praxis möglichst *immer* vermieden werden, weil sie keinen Einfluss auf das Ergebnis haben, aber trotzdem Kapazitäten beanspruchen. Das verlangsamt den Arbeitsprozess unnötig. Leider sind redundante Operationen nicht immer als solche erkennbar.

10.5. Funktionsketten

Wenn das Ergebnis einer Funktion als Parameter einer beliebigen anderen Funktion verwendet wird, dann sind die beiden Funktionen **verkettet**. Mit verketteten Funktionen

lassen sich komplexere Funktionen bilden. Umgekehrt lassen sich komplexe Funktionen in eine Verkettung einfacherer Teilfunktionen gliedern. Diese Technik heisst **Problemzerlegung**

::: {#def-fkt-kette} Eine **Funktionskette** ist eine Abfolge von Funktionen, wobei eine nachfolgende Funktion das Ergebnis einer vorangehenden Funktion als Parameter akzeptiert.

Die Funktionsverkettung dient dem Festlegen der Reihenfolge von Operationen. Das Prinzip der Funktionsverkettung ist einfach: Es wird das Ergebnis einer Funktion als Parameter einer anderen Funktion verwendet.

> **i** Hinweis
>
> In der Literatur wird die *Funktionsverkettung* auch als **Composting** bezeichnet.

Das folgende Beispiel nutz aus, dass die Operatoren der Grundrechenarten eigentlich Funktionen sind. Die Operation aus Formel 10.5 verwendet also die beiden Funktionen `subtraktion()` und `potenz()`. Formel 10.18 zeigt die Operation aus Formel 10.5 als verkettete Funktionen.

$$f(a,b) \rightarrow potenz(subtraktion(a,b),2) \tag{10.18}$$

Bei dieser Funktionsdefinition fällt auf, dass die Funktionen die durch die Verkettung gebildete Funktion f zwei Parameter hat. Die beiden verketteten Funktionen `subtraktion()` und `potenz()` verwenden jedoch die drei Parameter `a`, `b` und `2`. In diesem Fall ist der Wert `2` eine *Konstante*, die nur im Geltungsbereich der Funktion f verwendet wird.

Im Gegensatz zu Operatoren ist die Ausführungsreihenfolge verketteter Funktionen eindeutig: Verkettete Funktionen werden immer von *innen* nach *aussen* ausgeführt. In Formel 10.18 wird die Funktion `subtraktion()` vor der Funktion `potenz()` ausgeführt.

Werden Funktionsaufrufe als Parameter von anderen Funktionen geschrieben, dann sind diese Funktionen *geschachtelt* geschrieben. In *geschachtelten Funktionsketten* lässt sich die Reihenfolge der Funktionsaufrufe nicht immer leicht erkennen. Das ist immer dann der Fall, wenn viele Funktionen verkettet wurden. In solchen Fällen hilft die Darstellung der Funktionskette als *Baum*. Abbildung 10.1 zeigt den Funktionsbaum für Formel 10.18.

> **i** Hinweis
>
> Ein Funktionsbaum ist eine Spezialform eines sog. *Abstract Syntax Tree*. (Knuth, 1968)

In Abbildung 10.1 sind Funktionsnamen in den abgerundeten Feldern und Parameter in Rechtecken dargestellt. Werte und Variablen sind in Kreisen dargestellt. Ein Funktionsbaum wird von unten nach oben gelesen: Je weiter unten im Baum eine Funktion steht, desto früher wird sie ausgeführt. Die Funktionskette wird deutlich, indem man den Verbindungen zwischen den Funktionen folgt.

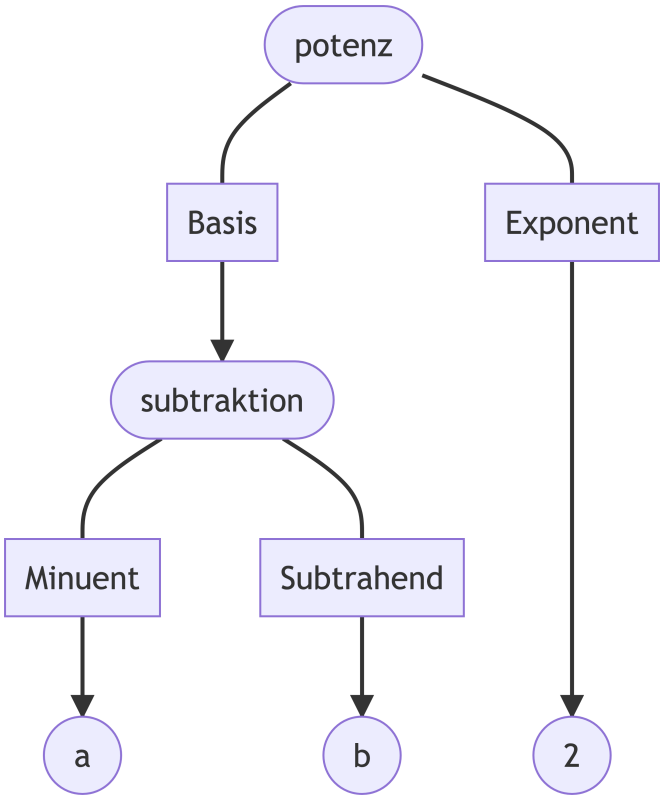


Abbildung 10.1.: Funktionsbaum für Formel 10.18

Um die Ausführungsreihenfolge bereits in der Notation einer Funktionskette zu verdeutlichen, wird der **allgemeine Verkettungsoperator** ($\circ$) verwendet. Dieser Operator ist ein spezieller Operator für Funktionen, der zwei Funktionen miteinander verkettet. Die mit diesem Operator verkettete Version von Formel 10.18 wird in Formel 10.19 gezeigt.

$$f(a, b) \rightarrow (potenz \circ subtraktion)(a, b, 2) \tag{10.19}$$

In dieser Schreibweise wird die *Ausführungsreihenfolge* von **rechts** nach **links** gelesen. Die *Parameter* werden von **links** nach **rechts** den Funktionen übergeben.

Der allgemeine Verkettungsoperator findet sich regelmässig in Literatur über die Analyse von Algorithmen, um verkettete Funktionen anzuzeigen. In diesem Fall fehlen in der Regel die Parameter, wie in Formel 10.20.

$$f \rightarrow potenz \circ subtraktion \tag{10.20}$$

Ausgehend vom allgemeinen Vekettungsoperator hat sich in den Datenwissenschaften ein **spezieller Verkettungsoperator** ($\triangleright$) für die Praxis als bedeutsam herausgestellt. Bei dieser Verkettung folgt die *Ausführungsreihenfolge* dem *Fluss der Daten* durch eine Funktionskette von **links** nach **rechts**. Dabei wird angenommen, dass das Ergebnis einer vorangehenden Funktion der *erste* Parameter der nachfolgenden Funktion ist. Bei den nachfolgenden Funktionen wird in dieser Notation der erste Parameter bei nachfolgenden Funktionen weggelassen. Ansonsten bleiben die Funktionsaufrufe unverändert. Formel 10.21 zeigt diese Schreibweise mit dem speziellen Verkettungsoperator für die Funktionskette aus Formel 10.18.

$$f(a, b) \rightarrow subtraktion(a, b) \triangleright potenz(2) \tag{10.21}$$

Die Funktionsverkettung mit dem *speziellen Verkettungsoperator* zeigt in vielen Fällen die Logik einer komplexen Funktion besser als die geschachtelte Schreibweise. Die Parameterliste P einer verketteten Funktion wird so getrennt, dass der erste Parameter P_1 nicht mehr Teil der gekürzten Parameterliste P' ist, so dass Formel 10.22 gilt.

Für diese Eigenschaft der geteilten Parameterliste für die spezielle Funktionsverkettung muss der Spezialfall berücksichtigt werden, dass die Parameterliste P leer ist. In diesem Fall kann diese Liste nicht geteilt werden, weil es keinen ersten Parameter P_1 gibt. Formel 10.22 gilt also nur, wenn die Parameterliste P *nicht* leer ist.

$$P = P_1 \cap P'; P \neq \emptyset \tag{10.22}$$

Aus dieser Bedingung hat die Konsequenz, dass die spezielle Funktionsverkettung nur verwendet werden kann, wenn die nachfolgende Funktion mindestens einen Parameter hat.

> **Merke**
>
> Die spezielle Funktionsverkettung ist für parameterlose Funktionen **undefiniert**.

Weil die verbleibende Parameterliste Teil der Operation ist, darf sie vom Operator nicht vernachlässigt werden. Der Operator muss also den ersten Parameter, die nachfolgende Funktion **und** die restlichen Parameter der nachfolgenden Funktion berücksichtigen. Daraus ergibt sich, dass der spezielle Funktionsverkettungsoperator *kein* binärer Operator ist. Die Funktion der speziellen Funktionsverkettung ist in Formel 10.23 definiert.

$$P_1 \rhd f(P') \Leftrightarrow \rhd(P_1, f, P') \rightarrow f(P_1, P') \tag{10.23}$$

In der Definition der speziellen Funktionsverkettung fehlt die vorangehende Funktion. Dieses Fehlen ergibt sich aus der Logik von Operatoren, weil höherwertige Teiloperationen vor niederwertigen Teiloperationen oder weiter links stehende Teiloperationen zuerst ausgeführt werden müssen. Der erste Parameter der nachfolgenden Funktion ist deshalb immer das Ergebnis einer vorangehenden Funktion. Diese Überlegung lässt sich weiter verallgemeinern:

> **Merke**
>
> Jeder Wert oder Bezeichner im Wertebereich von P_1 kann als linker Operand der speziellen Funktionsverkettung verwendet werden.

10.5.1. Anwendung von Funktionsketten

Funktionsketten sind ein leistungsfähiges Werkzeug für die Problemzerlegung bei der Datenarbeit. Funktionen lassen sich jedoch nicht beliebig verketten.

> **Merke**
>
> Funktionen lassen sich nur verketten, wenn die Rückgabewerte der vorangehenden Funktion den gleichen Wertebereich haben, wie der entsprechende Parameter der nachfolgenden Funktion.

Weil unter dieser Voraussetzung jede Funktion als Parameter einer anderen Funktion verkettet werden kann, lassen sich mit verketteten Funktionen beliebig komplexe Operationen bilden. Jede diese Operationen ist automatisch eine Funktion. Deshalb werden Funktionsketten in der Praxis häufig als zentraler Teil von Funktionen eingesetzt.

Mit diesem Wissen lassen sich die Eigenschaften von Funktionen in Funktionsketten untersuchen.

10.6. Funktionen als Werte

Bei der Substitution im Abschnitt 10.3.2 wurde eine Teiloperation einem Bezeichner zugewiesen, der anschliessend wie eine normale Variable behandlet wurde. Gleichzeitigt war dieser Bezeicher eine Funktion für die substituierte Teiloperation. Diese Überlegungen behandeln Funktionen genau gleich wie andere Datentypen.

Der Datentyp *Funktion* gibt einen gültigen Wertebereich vor. Die Symbole dieses Wertebereichs sind jedoch nicht Zahlen oder Zeichenfolgen, sondern Funktionen. Der Bezeichner einer Funktion kann so durch Zuweisung geändert werden, ***ohne*** die Funktion anzuwenden.

Eine Funktion ohne Bezeichner wird eine **anonyme Funktion** genannt.

> **i** Exkurs
>
> Im Internet und (seltener) in der Literatur werden anonyme Funktionen auch als **Lambda-Funktionen** bezeichnet. Dieser Name ist dem sog. λ-Kalkül (Church, 1940) aus der Berechenbarkeitstheorie entlehnt, weil das λ-Kalkül keine Bezeichner und Operatoren benötigt. Deshalb können Operationen im λ-Kalkül ausschliesslich durch anonyme Funktionen realisiert werden.
> *Eine Lambda-Funktion unterscheidet sich nicht von anderen Funktionen!*

Dieser besondere Datentyp kann wie jeder andere Datentyp für Parameter und Ergebnisse von Funktionen verwendet werden.

10.6.1. Callback-Funktionen

Definition 10.15. Eine **Callback-Funktion** ist eine Funktion, die als Parameter an eine andere Funktion übergeben wurde.

Bei der Übergabe einer Callback-Funktion wird die Funktion als Parameter übergeben. Das entspricht der Zuweisung einer Funktion an einen neuen Bezeichner. Dadurch muss die aufgerufene Funktion den Bezeichner der Callback-Funktion zum Zeitpunkt ihrer Definition nicht kennen, sondern kann über den Parameternamen auf diese Funktion zugreifen.

Bei der Verwendung von Callback-Funktionen muss klar zwischen dem Funktionsbezeicher und der Anwendung der Funktion unterschieden werden. Damit eine Funktion und nicht ihr Ergebnis als Parameter an eine andere Funktion übergeben werden kann, muss der Anwendungsoperator und die Parameterliste der Funktion weggelassen werden.

In Formel 10.24 wird der Funktion g das *Ergebnis* der Funktionsanwendung $f(3)$ übergeben. Im Gegensatz dazu wird in Formel 10.25 die Funktion f selbst an die Funktion g übergeben. Die Funktion g kann nun die Funktion f aufrufen.

$$g(f(3)) \tag{10.24}$$

$$g(f) \tag{10.25}$$

Callback-Funktionen ermöglichen das **Verallgemeinern** von Algorithmen, indem der spezielle Teil einer Funktion durch eine Callback-Funktion festgelegt wird.

10.6.2. Closure-Funktionen

Definition 10.16. Closure-Funktionen sind Funktionen, die als Rückgabewert von einer anderen Funktion erzeugt werden.

Eine Closure-Funktion wird im Geltungsbereich der erzeugenden Funktion definiert und übernimmt alle Bezeichner dieses Geltungsbereichs, so dass für eine komplexe Operation alle notwendigen Werte vorhanden sind.

Das Beispiel in Formel 10.26 erzeugt die Funktion $pDef$ Closure-Funktionen zum Potenzieren mit einem festen Exponenten.

$$pDef(exponent) \rightarrow (closure(basis) \rightarrow potenz(basis, exponent)) \tag{10.26}$$

Die Funktion $pDef$ kann nun dazu verwendet werden, neue Funktionen zu erzeugen. Z.B. die Funktionen $quadrat$ und $kubik$. Die Funktion $quadrat$ hat den festen Exponenten 2 und die Funktion $kubik$ den festen Exponenten 3. Diese Funktionsdefinitionen zeigt Formel 10.27.

$$\begin{aligned} quadrat &\rightarrow pDef(2) \\ kubik &\rightarrow pDef(3) \end{aligned} \tag{10.27}$$

Die beiden Bezeichner enthalten nun eigene Versionen der *closure*-Funktion.

Jede Anwendung von $pDef$ erzeugt eine *neue* Funktion. Dabei ist der Bezeichner *closure* auf den Geltungsbereich von $pDef$ beschränkt. Deshalb kann dieser nach dem Anwenden der Funktion $pDef$ nicht direkt verwendet werden.

Die erzeugten Funktionen lassen sich wie andere Funktionen über ihre Bezeichner anweden. Im aktuellen Beispiel ergeben sich so die Gleichungen in Formel 10.28.

$$\begin{aligned} quadrat(2) &= 4 \\ quadrat(3) &= 9 \\ kubik(2) &= 8 \\ kubik(3) &= 27 \end{aligned} \tag{10.28}$$

Closure-Funktionen **vereinfachen** komplexe Funktionen, indem deren Parameter über die erzeugende Funktion festgelegt werden. Diese Technik wird in der Literatur gelegentlich als *currying* bezeichnet.

10.7. Besonderheiten von Funktionsketten

10.7.1. Die Identitätsfunktion

Als erstes soll die Verkettung einer beliebigen *Projektion* F mit der Identitätsfunktion f_{id} untersucht werden. Es ist wichtig, dass F eine Projektion ist, weil nur diese einer Eingabe immer den gleichen Ergebniswert zuweisen. Die Identitätsfunktion ist in Formel 10.4 definiert. Diese Funktion kann für jeden Wertebereich angepasst werden, so dass für jeden Wertebereich eine geeignete Identitätsfunktion für die Verkettung existiert. All diese Versionen der Identitätsfunktion werden unter f_{id} zusammenfasst und als **die Identitätsfunktion** bezeichnet.

Zuerst wird die Funktionsverkettung mit dem allgemeinen Verkettungsoperator betrachtet.

Wird die Identitätsfunktion mit einer beliebigen Funktion F nachfolgend verkettet, dann wird das Ergebnis dieser Funktion an die Identitätsfunktion als Parameter übergeben. Weil die Identitätsfunktion die übergebenen Parameter unverändert als Ergebnis zurückgibt, ist das Ergebnis der nachfolgenden Verknüpfung gleich dem Ergebnis der Funktion F. Es gilt also Formel 10.29.

$$f_{id} \circ F = F \tag{10.29}$$

Im vorangehenden Fall wird das Ergebnis der Identitätsfunktion an die Funktion F als Parameter übergeben. Weil F beliebig viele Parameter haben kann, werden ihre Parameter als *eine* Parameterliste behandelt. Diese Parameterliste wird der Identätsfunktion übergeben, weil der Wertebereich der Parameter und der Rückgabewerte der Identitätsfunktion identisch sind, kann die Identitätsfunktion immer mit einer Funktion verkettet werden, die diese Parameterliste akzeptiert. Weil die Parameter unverändert der Funktion F übergeben werden und F eine Projektion ist, ist das Ergebnis dieser Funktionskette identisch mit dem Ergebnis von F. Es gilt also Formel 10.30.

$$F \circ f_{id} = F \tag{10.30}$$

Aus Formel 10.29 und Formel 10.30 ergibt sich Formel 10.31.

$$f_{id} \circ F = F \circ f_{id} = F \tag{10.31}$$

Diese Beziehung erfüllt die Kriterien für das neutrale Element aus Formel 10.12 für den allgemeinenen Verkettungsoperator.

Für die spezielle Funktionsverkettung vereinfachen sich diese überlegungen, weil nur der erste Parameter P_1 und nicht die ganze Parameterliste P betrachtet werden muss. Wie oben beschrieben, darf die restliche Parameterliste der Funktion F nicht vernachlässigt werden.

Für den Fall, dass die Identitätsfunktion die vorangehende Funktion ist, gilt Formel 10.32. Die Identitätsfunktion betrifft also nur den ersten Parameter von F. Wegen der Definition der Identitätsfunktion gilt die Gleichung $f_{id}(P_1) = P_1$.

$$f_{id}(P_1) \rhd F(P') = P_1 \rhd F(P') = F(P) \tag{10.32}$$

Entsprechend dieser Logik kann auch Ergebnis der vorangehenden Funktion als ein einzelner Wert behandelt werden. Das ist für den zweiten Fall wichtig, wenn die Identitätsfunktion die nachfolgende Funktion ist. In diesem Fall wird die Identitätsfunktion auf das Ergebnis angewandt, so dass die Gleichung $f_{id}(E) = E$ gilt. Weil die Identitätsfunktion keine weiteren Parameter akzeptiert, entfällt die Behandlung von P'. Daraus ergibt sich die Gleichung 10.33.

$$F(P) \rhd f_{id}() = E = F(P) \tag{10.33}$$

Werden die beiden Fälle zusammengefasst, dann ergibt sich die Gleichung 10.34.

$$F(P) \rhd f_{id}() = f_{id}(P_1) \rhd F(P') = F(P) \tag{10.34}$$

Die Identitätsfunktion ist also auch das neutrale Element der speziellen Funktionsverkettung.

> **Merke**
>
> Die Identitätsfunktion ist das neutrale Element der allgemeinen und speziellen Funktionsverkettungsoperators.

10.7.2. Verkettung mit Umkehrfunktionen

In Abschnitt 10.2.1 wurden Projektionen definiert und festgehalten, dass einige Projektionen eine Umkehrfunktion haben.

Existiert für eine beliebige Projektion F eine Umkehrfunktion F^{-1}, dann kann die Verkettung zwischen der Funktion und der Umkehrfunktion untersucht werden. Die Umkehrfunktion einer Funktion akzeptiert die Rückgabewerte der Funktion als Parameter und hat als Rückgabewert die Parameter der ursprünglichen Funktion. Die Parameter von F^{-1} und die Rückgabewerte von F haben gemäss dieser Definition den gleichen Wertebereich. Das Gleiche gilt für die Rückgabewerte von F^{-1} und die Parameter von F. Damit ist für beide Richtungen die Voraussetzung für die Verkettung erfüllt. Es sind also sowohl die Verkettung $F^{-1} \circ F$ als auch $F \circ F^{-1}$ für alle umkehrbaren Funktionen zulässig.

Werden nun die beiden Funktion F und F^{-1} als $F \circ F^{-1}$ verkettet, dann werden die Parameter von F^{-1} auf die Rückgabewerte projiziert. Dieser werden anschliessend als Parameter der F als Paramter übergeben und wieder projiziert. Weil das Ergebnis einer umkehrbaren

Funktion identisch mit den Parametern ihrer Umkehrfunktion ist, muss das Ergebnis dieser Funktionskette den eingegebenen Parametern entsprechen.

Wird die Verkettung zu $F^{-1} \circ F$ umgekehrt, dann muss das Ergebnis dieser Funktionskette aus den gleichen Gründen wie oben, mit den Parametern von F gleich sein.

Funktionen mit der Eigenschaft, dass Ergebnis und Parameter gleich sind, sind mit der Identitätsfunktion funktional gleich. Es gilt also Formel 10.35.

$$F^{-1} \circ F = F \circ F^{-1} = f_{id} \tag{10.35}$$

> **Merke**
>
> Wird eine *umkehrbare Projektion* mit ihrer Umkehrfunktion verkettet, dann ist diese verkettete Funktion mit der Identitätsfunktion funktional gleich.

Nun ist f_{id} auch das neutrale Element der Funktionsverkettung. Deshalb gilt zusätzlich, dass jede Verkettung einer umkehrbaren Projektion mit ihrer Umkehrfunktion *redundant* ist. Diese Überlegung lässt sich weiter verallgemeinern:

> **Merke**
>
> Jede (Teil-)Funktionskette, die mit der Identitätsfunktion funktional gleich ist, ist ***redundant*** und kann weggelassen werden.

11. Zeichenketten

Neben Zahlen gehören Zeichenketten zu den wichtigsten Datentypen für die Datenanalyse. Bei Zeichenketten werden als erstes Worte oder Sätze assoziiert. Im Kapitel 3.5 wurde die Kodierung von Symbolen als Zahlen vorgestellt und Kapitel 8 hat Zeichenketten als fundamentalen Datentyp eingeführt. Diese Ideen werden in diesem Kapitel erweitert und durch die wichtigsten Operationen für Zeichenketten ergänzt.

Definition 11.1. Eine Zeichenkette ist eine beliebige Kette von Textsymbolen. Textsymbole können Buchstaben, Ziffern, Satzzeichen sowie nicht-druckbare Zeichen sein.

Zeichenkette sind immer **diskrete Daten**.

Aus dieser Definition folgt, dass Zeichenketten eine Länge haben müssen.

Definition 11.2. Die **Länge** einer Zeichenkette entspricht der Anzahl aller enthaltenen Symbole.

Zeichenketten sind also *Werte* und haben gleichzeitig die Eigenschaften eines Vektors.

11.1. Nicht-druckbare Zeichen

Definition 11.3. Nicht-druckbare Zeichen sind Symbole, die bei der Darstellung einer Zeichenkette nicht angezeigt werden können. Die nicht-druckbaren Zeichen zählen zur Länge einer Zeichenkette und verändern den Inhalt einer Zeichenkette.

Beispiel: Die Zeichenkette `Hallo` unterscheidet sich von der Zeichenkette `Hal<0x08>lo`.

Zu den nicht-druckbaren Zeichen gehören auch Leerzeichen, Tabulatoren und Zeilenumbrüche. Diese speziellen nicht-druckbaren Zeichen sind nur dann zu erkennen, wenn sie von druckbaren Zeichen umgeben sind.

Beispiel 11.1. Deutlich wird das an den folgenden Zeichenketten:

- `Hallo`
- `Hal<0x07>lo`, wobei das Symbol `0x07` für einen Piepton steht
- `Hal<0x08>lo`, wobei das Symbol `0x08` für einmal Rückwärtslöschen steht.

Die erste Zeichenkette hat die Länge 5, die zweite Zeichenkette hat die Länge 6 und die dritte Zeichenkette hat ebenfalls die Länge 6.

11.2. Die leere Zeichenkette

Definition 11.4. Eine Zeichenkette der Länge 0 enthält keine Symbole. Diese Zeichenkette heisst die **leere Zeichenkette**.

In einigen Umgebungen lässt sich die leere Zeichenkette nicht leicht von Zeichenketten unterscheiden, die aus nicht-druckbaren Zeichen bestehen. Für viele Funktionen ist die leere Zeichenkette problematisch oder als Wert nicht zulässig. In solchen Fällen muss die leere Zeichenkette über die Länge einer Zeichenketten-Variablen explizit kontrolliert werden.

11.3. Symbolvektoren

In einer Zeichenkette sind die Symbole angeordnet, so dass jedes Symbol über dessen Position angesprochen werden kann.

Wird eine Zeichenkette in die einzelnen Symbole gegliedert, dann ergibt sich in den meisten modernen Programmiersprachen ein *Zeichenkettenvektor* mit Zeichenketten der Länge `1`.

11.4. Texttrennung

Damit Zeichenketten getrennt werden können wird eine spezielle Funktion benötigt. Diese Funktion wird als *Texttrennung* (engl. *split*) bezeichnet. Die Funktion hat zwei Parameter: Die Zeichenkette und ein Trennsymbol. Das Trennsymbol ist eine Zeichenkette, die zwei Teile einer Zeichenkette trennt. Beispiele für solche Trennsymbole wurden im Kapitel 9 vorgestellt.

Die Texttrennung teilt eine Zeichenkette so auf, dass für alle Vorkommnisse des Trennsymbols in der Zeichenkette der Teil vor dem Trennsymbol und der Teil nach dem Trennsymbol als zwei separate Zeichenketten vorliegen. Das Trennsymbol wird entfernt. Das Ergebnis der Texttrennung ist ein Zeichenkettenvektor mit den getrennten Zeichenketten.

> **Merke**
>
> Die Länge des Vektors nach der Texttrennung entspricht der Anzahl der Vorkommnisse des Trennsymbols in der Zeichenkette plus 1.

Definition 11.5. Die durch die Texttrennung entstehenden Zeichenketten heissen *Token*.

Token können Sätze, Worte oder auch einzelne Buchstaben sein. Ein Token muss nicht zwingend sprachlich sinnvoll sein.

Definition 11.6. Ein Vektor der Tokens enthält heisst *Token-Vektor*.

Die Texttrennung hat also einen *Token-Vektor* als Ergebnis.

Um die einzelnen Symbole einer Zeichenkette zu erhalten, muss etwas um die Ecke gedacht werden, um die Zeichenkette zu erkennen, die zwischen jedem Symbol steht: Der Abstand zwischen zwei Symbolen in einer Zeichenkette ist immer genau 1. Weil Zeichenketten *diskrete Daten* sind, muss die Zeichenkette zwischen zwei Symbolen eine Länge von `0` haben. Die Zeichenkette mit der Länge `0` ist nach Definition 11.4 die leere Zeichenkette.

Entsprechend enthält der Token-Vektor nach der Texttrennung mit der leeren Zeichenkette den einzelnen Symbolen der ursprünglichen Zeichenkette.

11.5. Textverkettung

Die Texttrennung ist umkehrbar. Die Umkehrfunktion der Texttrennung ist die Textverkettung. Die Textverkettung fügt mehrere Zeichenketten zu einer neuen Zeichenkette zusammen.

Die Textverkettung ist eine spezielle From der Konkatenation von Vektoren (s. Kapite 13.2).

Die Textverkettung hat zwei Parameter:

- Einen Token-Vektor
- Ein Trennsymbol

Die Textverkettung fügt alle Zeichenketten des Token-Vektors zusammen, indem die erst das vorangehende Token, dann das Trennsymbol und zuletzt die nachfolgende Zeichenkette zusammengefügt werden. Das Ergebnis ist eine neue Zeichenkette.

Wird der Token-Vektor einer Texttrennung mit dem gleichen Trennsymbol wieder verkettet, dann ist das Ergebnis die ursprüngliche Zeichenkette.

Die leere Zeichenkette ist das neutrale Element der Textverkettung.

11.6. Normalisierung

Definition 11.7. Das Entfernen und Vereinheitlichen von Leerzeichen und nicht-druckbaren Zeichen aus einer Zeichenkette heisst **Normalisieren von Zeichenketten**.

Eine *normalisierte Zeichenkette* enthält keine führenden oder nachfolgenden Leerzeichen, keine nicht-druckbaren Zeichen und keine aufeinanderfolgenden Leerzeichen.

Die Operation des Normalisieren ist das Suchen und Ersetzen. Beim Suchen und Ersetzen werden Zeichenketten durch andere Zeichenketten ersetzt. Die Basis-Funktion des Suchens und Ersetzens hat drei Parameter:

- Die Zeichenkette, in der gesucht werden soll

- Die gesuchte Zeichenkette (das *Suchmuster*)
- Die Ersetzungszeichenkette

Die Funktion sucht in der Zeichenkette nach dem Suchmuster und ersetzt das Suchmuster durch die Ersetzungszeichenkette. Das Ergebnis ist eine neue Zeichenkette, in der das Suchmuster durch die Ersetzungszeichenkette ersetzt wurde.

Wird als Ersetzung die leere Zeichenkette verwendet, dann wird das Suchmuster aus der Zeichenkette entfernt.

Wenn das Ergebnis einer Suchen-und-Ersetzen-Funktion die ursprüngliche Zeichenkette ist, wurde keine Ersetzung durchgeführt.

Beim *Normalisieren* von Zeichenketten werden Suchen-und-Ersetzen-Funktionen solange angewendet, bis keine Ersetzung mehr vorgenommen wird. Die Zeichenkette ist dann normaliesiert.

Weil das Normalisieren von Zeichenketten eine sehr häufige Operation ist, gibt es dafür spezielle Funktionen. Die wichtigsten dieser Funktionen sind:

- Die Bereinigung von Leerzeichen am Anfang und Ende einer Zeichenkette (engl. *trim*)
- Die Bereinigung von aufeinanderfolgenden Leerzeichen.

> Praxis
>
> Oft müssen Zeichenketten auch von anderen Symbolen bereinigt werden. Dabei handelt es sich um Erweiterungen des Normalisierens.

11.7. Gross- und Kleinschreibung

Zeichenketten können in der Gross- oder Kleinschreibung variieren. Die Gross- und Kleinschreibung ist ein *semantisches Merkmal* von Zeichenketten, kann aber auch auf Tippfehler zurückgehen. Um Varianten der Gross- und Kleinschreibung zu ignorieren, werden alle Symbole einer Zeichenkette in Gross- oder in Kleinbuchstaben umgewandelt, falls es sich um Buchstaben handelt. Alle anderen Symbole bleiben unverändert.

Die Umwandlung von Gross- in Kleinbuchstaben ist leicht umzusetzen, weil Kleinbuchstaben in der Zeichenkodierung immer um den gleichen Wert (`0x20`)grösser sind als Grossbuchstaben. Die Umwandlung wird in der Regel mit einer entsprechenden Funktion durchgeführt und muss nicht selbst implementiert werden.

Satzzeichen, Zahlen und viele Schriften verfügen über keine grossen und kleinen Buchstaben. Die entsprechen Symbole werden nicht umgewandelt.

12. Boole'sche Operationen

Boole'sche Operationen sind für das Programmieren von zentraler Bedeutung, weil mit ihnen regelbasierte Entscheidungen umgesetzt werden.

> **Hinweis**
>
> Als *Logischer Ausdruck* werden alle Funktionsketten bezeichnet, die einen Wahrheitswert als Ergebnis haben.

> **Merke**
>
> Die einfachsten logischen Ausdsrücke sind die Wahrheitswerte selbst.

Die klassische Aussagenlogik ist seit der Antike Teil der Rethorik und der Philosophie. Sie wurde von Aristoteles in seiner Schrift "Peri hermeneias" (Über die Interpretation) beschrieben. Das Ziel der Aussagenlogik ist es, die Struktur von Argumenten zu analysieren und zu bewerten.

Ein wesentliche Aufgabe der Aussagenlogik ist die Analyse von Argumenten, um sinnvolle und nicht-sinnvolle Aussagen zu unterscheiden. Dazu werden die Argumente in Voraussetzungen (*Prämissen*) und Schlussfolgerungen (*Konklusionen*) unterteilt. Die Prämissen sind die Voraussetzungen, die für die Konklusion erfüllt sein müssen.

Die klassische Aussagenlogik ist eine *zweiwertige* Logik. Das bedeutet, dass die Aussagen entweder *wahr* oder *falsch* sein können. Es gibt keine Zwischenwerte.

Die klassische Aussagenlogik beschäftigte sich mit der Verknüpfung von Argumenten zu *Aussagen*. Dabei werden drei Arten von Aussagen unterschieden:

- Die Tautologie bezeichnet Aussagen, die immer wahr sind.
- Die Kontradiktion bezeichnet Aussagen, die immer falsch sind.
- Die Kontingenz ist eine Aussage, die weder eine Tautologie noch eine Kontradiktion ist.

Aussagen sind also *Konklusionen*, die sich aus der Verknüpfung von Argumenten (*Prämissen*) ergeben. Die klassische Aussagenlogik untersucht, wie sich die Wahrheitswerte der Prämissen auf die Wahrheitswerte der Konklusion auswirken. Es handelt sich dabei also um die Analyse der Verknüpfung von Argumenten zu Aussagen in Form von Wenn-Dann-Beziehungen.

Klassisch, d.h. seit der Antike, werden die folgenden Beziehungen zwischen zwei Aussagen unterschieden (Rautenberg, 2008).

- Die Negation (NICHT) kehrt den Wahrheitswert einer Aussage um. Eine wahre Aussage wird falsch und eine falsche Aussage wird wahr.
- Die Konjunktion (UND) ist nur dann wahr, wenn beide Aussagen wahr sind.
- Die Nihilition ist nur dann wahr, wenn beide Aussagen falsch sind.
- Die Inversion ist wahr, wenn die erste Aussage wahr ist oder die zweite Aussage falsch ist.
- Die Äquivalenz ist wahr, wenn beide Aussagen den gleichen Wahrheitswert haben.
- Die Antivalenz (entweder-oder) ist wahr, wenn die beiden Aussagen unterschiedliche Wahrheitswerte haben.
- Die Disjunktion (ODER) ist wahr, wenn mindestens eine der beiden Aussagen wahr ist.
- Die Unverträglichkeit ist wahr, wenn mindestens eine der beiden Aussagen falsch ist.
- Die Implikation ist wahr, wenn die erste Aussage falsch ist oder die zweite Aussage wahr ist.

Bereits den antiken Philosophen war bekannt, dass die Negation, die Konjunktion und die Disjunktion die Grundlage für die Aussagenlogik bilden. Die anderen Beziehungen lassen sich aus diesen drei Beziehungen ableiten.

12.1. Mathematische Operationalisierung der Aussagenlogik

George Boole hat mit seiner Arbeit "The Mathematical Analysis of Logic" (Boole, 1847) die Grundlagen für die moderne Informatik gelegt. Er hat die Grundlagen für die Boole'sche Algebra gelegt, die die Grundlage sowohl für die moderne Kommunikationstechnologie als auch für die Informatik ist. Seine Überlegungen standen im Kontext der industriellen Revolution und der damit verbundenen Entwicklung von Maschinen, die mit Hilfe von mathematischen Gleichungen gesteuert werden. Er erkannte, dass die Sprache und damit die Philosophie der Logik genau wie die Mathematik auf Grundlage von Symbolen basierte. Er fragte sich, ob Sprache und Logik einer mathematischen Analyse unterzogen werden können. Speziell interessierte ihn für seine Analyse der Zweig des *Syllogismus* in der Logik, der sich mit dem deduktiven Schlussfolgern aus Argumenten beschäftigt.

Boole wurde durch die Arbeiten von Leibniz beeinflusst, die sich mit der Formalisierung von Argumenten beschäftigten. Leibniz hatte die Idee, dass Argumente in Form von mathematischen Gleichungen dargestellt werden können. Boole hat diese Idee aufgegriffen und weiterentwickelt.

Boole versuchte, die klassische Aussagenlogik mathematisch zu formalisieren, indem er jeder Aussage einen Wahrheitswert in Form von `0` für falsch und `1` für wahr zuordnete. Auf dieser Grundlage untersuchte er, welche mathematischen Operationen zu den bekannten Ergebnissen der klassischen Aussagenlogik führen.

Ausgehend von den Beziehungsarten der klassischen Aussagenlogik stellte er die *Belegungen* für die möglichen Kombinationen von Wahrheitswerten in der jeweiligen Beziehung auf. D.h. er stellte die möglichen Kombinationen von Wahrheitswerten der Prämissen den Wahrheitswerten der Konklusion gegenüber. Für jede dieser Belegungen suchte Boole

anschliessend nach einer arithmetischen Operation, die alle Belegungen einer Beziheung erzeugt.

12.1.1. Belegungstafeln oder Wahrheitstafeln

Eine Wahrheitstafel oder Wahrheitstabelle ist eine Tabelle, die alle möglichen Kombinationen von Wahrheitswerten für einen logischen Ausdruck enthält. Weil die klassische Aussagenlogik nur zwei Wahrheitswerte kennt, müssen nur alle Kombinationen dieser beiden Werte für einen logischen Ausdruck gefunden werden.

Die Wahrheitstafel für die Negation `NICHT` sieht wie folgt aus:

a	NICHT a
0	1
1	0

Die in den Wahrheitstafeln aufgeführten Ergebniswerte werden auch als *Belegungen* bezeichnet. Für die logischen Basisoperationen werden diese Belegungen als *Grundbelegung* bezeichnet.

Die Grundbelegungen sind die Basis für die Boole'sche Algebra und Arithmetik.

Die Wahrheitstafel für die Konjunktion `UND` sieht wie folgt aus:

a	b	a UND b
0	0	0
0	1	0
1	0	0
1	1	1

Für Wahrheitstafeln ist auch die Matrix-Schreibweise üblich. Dabei werden die Wahrheitswerte als Spaltenvektoren geschrieben. Die Wahrheitstafel für die Konjunktion `UND` sieht wie folgt aus:

$\downarrow$ b | a $\rightarrow$	0	1
0	0	0
1	0	1

Verkürzt werden die Spalten und Zeilenüberschriften weggelassen und können die Belegung als Matrix schreiben:

$$\begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$$

Die Wahrheitstafel für die Disjunktion `ODER` sieht wie folgt aus:

a	b	a ODER b
0	0	0
0	1	1
1	0	1
1	1	1

Oder in der Matrix-Schreibweise:

$$\begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix}$$

Die Wahrheitstafel für die Exklusiv-Oder `XODER` sieht wie folgt aus:

a	b	a XODER b
0	0	0
0	1	1
1	0	1
1	1	0

Die entsprechende Belegung sieht als Matrix wie folgt aus:

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$$

12.1.2. Boole'sche Arithmetik

Obwohl die Arbeit von Geoge Boole wegweisend für die Logik als mathematische Disziplin war, konzentrierte sich seine Arbeit auf der Übersetzung von logischen Aussagen in *arithmetische* Ausdrücke. Die Boole'sche Arithmetik ist eine Erweiterung der Arithmetik. Dabei werden logische Ausdrücke mithilfe der Grundrechenarten in berechenbare Ausdrücke übersetzt.

Die Boole'schen Arithmetik im engeren Sinn basiert auf der Übersetzung von Wahrheitswerten in Zahlen. Dabei wird `WAHR` als `1` und `FALSCH` als `0`.

Logische Operation	Arithmetische Operation
`NICHT`	$1 - a$
`UND`	$a \cdot b$
`ODER`	$a + b - ab$
`Entweder-Oder` (`XODER`)	$a + b - 2ab$ oder $(a - b)^2$

Weil Wahrheitswerte immer `0` oder `1` sein müssen, stellen diese Operationen sicher, dass die Ergebnisse logischer Ausdrücke ebenfalls immer `0` oder `1` sind. Das ist vor allem für die beiden Oder-Operationen wichtig, weil die arithmetische Addition einen Wert ausserhalb der erlaubten Werte liefert, wenn beide Operanden `Wahr` bzw. `1` sind.

Boole konnte zeigen, dass die arithmetischen Operationen die gleichen Ergebnisse liefern wie die logischen Operationen der klassischen Aussagenlogik. Ausgehend von den Wahrheitstafeln zeigte Boole auch, dass die Begrenzung aus Wahrheitswerte `0` und `1` bei Additionen eine Ausgleichsoperation erfordert, damit das Ergebnis in den gleichen Wertebereich fällt.

12.2. Boole'sche Algebra

Die Bool'sche Arithmetik ist für regelmässige Aufgaben etwas unhantlich, weil die beiden Operationen `ODER` und `Entweder-Oder` sich nicht mit einer arithmetischen Operation ausdrücken lassen. Weil die logischen Operationen einen besonderen Fall der Mengenlehre darstellen, wurden die Symbole für die Konjunktion und Disjunktion der Symbolik der Mengenlehre entlehnt.

Logische Operation	logischer Operator	Mengenoperator	Mengenoperation
Negation	$\neg$	$\notin$	Negation
Konjunktion (UND)	$\wedge$	$\cap$	Schnittmenge
Disjunktion (ODER)	$\vee$	$\cup$	Vereinigung
Antivalenz (XODER)	$\oplus$	$\triangle$	Symmetrische Differenz

Die Antivalenz kann durch die anderen drei Operationen ausgedrückt werden, weshalb sie seltener in logischen Ausdrücken verwendet wird.

12.2.1. Grundregeln der Boole'schen Algebra

Grundsätzlich gelten für die Boole'sche Algebra die gleichen Regeln wie für die Arithmetik. D.h. zuerst wird die Negation berechnet, dann `UND` abschliessend `ODER`. Diese Reihenfolge ist damit begründet, dass die `UND`-Operation der Multiplikation und die `ODER`-Operation der Addition entsprecht. Um die Reihenfolge zu ändern, werden wie üblich Klammern verwendet.

Logische Ausdrücke werden schnell komplex und unübersichtlich. Die Boole'sche Algebra definiert Regeln, die die Umformung von logischen Ausdrücken vereinfachen. Die wichtigsten Regeln sind in der folgenden Tabelle aufgelistet.

Name	Gleichung
Idempotenzgesetz	$a \wedge a = a$

Name	Gleichung
	$a \vee a = a$
Tautologie	$a \vee \neg a = 1$
Kontradiktion	$a \wedge \neg a = 0$
Kommutativgesetz	$a \wedge b = b \wedge a$
	$a \vee b = b \vee a$
Assoziativgesetz	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$
	$(a \vee b) \vee c = a \vee (b \vee c)$
Distributivgesetz	$a \wedge (b \vee c) = a \wedge b \vee a \wedge c$
	$a \vee b \wedge c = (a \vee b) \wedge (a \vee c)$
Absorptionsgesetz	$a \wedge (a \vee b) = a$
	$a \vee a \wedge b = a$
De Morgan'sche Regeln	$\neg(a \wedge b) = \neg a \vee \neg b$
	$\neg(a \vee b) = \neg a \wedge \neg b$

Viele Programmiersprachen werten logische Ausdrücke von links nach rechts aus und brechen die Auswertung ab, sobald das Ergebnis feststeht. Das ist bei der Boole'schen Algebra eigentlich nicht möglich, weil die Reihenfolge der Auswertung nicht festgelegt ist. Um die Auswertung logischer Ausdrücke in Programmiersprachen zu unterstützen, sollten die Teilaussagen in ihrer Wichtigkeit und Komplexität absteigend angeordnet werden.

Praxis

Logische Ausdrücke haben in der Programmierpraxis eine grosse Bedeutung. Aus diesem Grund verfügen alle Programmiersprachen die Möglichkeit, *beliebige* Zahlen als Wahrheitswerte zu behandeln. Dabei gilt die Konvention, dass die Zahl `0` als `FALSCH` und alle anderen Zahlen als `WAHR` interpretiert werden.

12.3. Vergleiche

Die Vergleichsoperatoren sind in der Programmierung sehr wichtig, weil sie die Grundlage für die Kontrollstrukturen bilden.

Definition 12.1. Vergleichsoperatoren sind binäre Operatoren, die zwei Operanden miteinander vergleichen. Das Ergebnis ist immer ein Wahrheitswert.

Der zentrale Vergleich ist die Gleichheit (`=`). Die Gleichheit ist gegeben, wenn beide Operanden des Vergleichs gleich sind. In diesem Fall gibt dieser Vergleich `WAHR` zurück.

Für Zahlenwerte und andere sortierbare Werte sind die Vergleiche kleiner als (`<`) und grösser als (`>`) definiert. Dabei wird der Vergleich `WAHR` zurückgegeben, wenn der linke Operand kleiner bzw. grösser als der rechte Operand ist.

Zusätzlich sind die kombinierten Vergleiche wichtig:

Operation	Symbol	Alternative Schreibweise
ungleich	$a \neq b$	$\neg(a = b)$
kleiner oder gleich	$a \leq b$	$(a < b) \vee (a = b)$
grösser oder gleich	$a \geq b$	$(a > b) \vee (a = b)$

Bei Vergleichsoperatoren muss darauf geachtet werden, dass die Operanden vom gleichen Typ sind. Eine Zahl kann nicht mit einer Zeichenkette verglichen werden.

Für logische Ausdrücke sind direkte Vergleiche zwischen zwei Werten nicht immer geeignet. Immer wenn der gleiche Wert mit mehreren anderen verglichen werden muss werden logische Ausdrücke mit direkten Vergleichen schnell komplex. Für solche Vergleiche ist der *Existenz*-Vergleich wichtig. Dabei wird der Wert `WAHR` zurückgegeben, wenn der linke Operand ein Element des rechten Operanden ist.

Die *Existenz* wird mithilfe des $\in$-Operators überprüft. Der $\in$-Operator ist ein spezieller Vergleichsoperator, der `Wahr` zurückgibt, wenn der linke Operand im rechten Operand vorkommt. Dabei steht der linke Operand für den *Suchwert* und der rechte Operand für den *Suchbereich*. Der Suchbereich ist dabei immer eine Menge bzw. ein Vektor. Der Vergleich der beiden Operanden wird formal als $a \in B$ geschrieben, wobei B eine Menge bzw. ein Vektor von Werten ist. Dieser Vergleich entspricht einer `ODER`-Operation, mit der die Elemente des Vektors B einzeln mit dem Wert a auf Gleichheit geprüft werden, wie Formel 12.1 zeigt.

$$\begin{aligned} & a \in \{1, 2, 3, 4, 5\} \\ \Leftrightarrow & (a = 1) \vee (a = 2) \vee (a = 3) \vee (a = 4) \vee (a = 5) \end{aligned} \tag{12.1}$$

In vielen Fällen sind die zu prüfenden Elemente in B nicht vorab bekannt oder die Anzahl der Elemente variiert. In diesem Fall ist eine explizite `ODER`-Operation nicht möglich. Auch in weniger komplexen Fällen, empfielt es sich, die `ODER`-Operation zu vermeiden und die Existenzprüfung vorzuziehen, weil sie die Lesbarkeit eines logischen Ausdrucks erhöht.

Formel 12.2 zeigt die Anwendung des $\in$-Operators.

$$7 \in \{1; 2; 3; 4; 5; 6; 7; 8; 9; 10\} \tag{12.2}$$

Dieser Ausdruck ist in diesem Beispiel `Wahr`.

Der $\in$-Operator kann für Vektoren als linker Operator verallgemeinert werden. In diesem Fall werden die linken Operanden ebenfalls als Vektor dargestellt. Nun wird für jeden Wert des linken Operators der Vergleich mit der rechten Seite durchgeführt.

$$\{7; 11\} \in \{1; 2; 3; 4; 5; 6; 7; 8; 9; 10\} \tag{12.3}$$

Der Vergleich in Formel 12.3 entspricht also den beiden separaten Vergleichen in Formel 12.4.

$$\begin{aligned} 7 &\in \{1;2;3;4;5;6;7;8;9;10\} \\ 11 &\in \{1;2;3;4;5;6;7;8;9;10\} \end{aligned} \tag{12.4}$$

Im Beispiel von Formel 12.3 ist das Ergebnis des Vergleichs: `{WAHR; FALSCH}`.

12.4. Entscheidungen

Definition 12.2. Eine **Entscheidung** beschreibt eine Funktion, die mit Hilfe eines *logischen Ausdrucks* aus eines von zwei alternativen Ergebnissen *auswählt*.

Bei Entscheidungen werden in der Regel die beiden Fälle des logischen Ausdrucks unterschieden. Dabei wird der Fall, der `WAHR` ergibt als *positiver Fall* und der Fall, der `FALSCH` ergibt als *negativer Fall* bezeichnet.

Entscheidungen können nacheinander ausgeführt werden. Dabei führen die beiden Fälle der ersten Entscheidung in jeweils eine weitere Entscheidung. Solche verschachtelten Entscheidungen werden als **Entscheidungsbäume** bezeichnet.

Definition 12.3. Eine Verkettung von Entscheidungen wird als **Entscheidungsbaum** bezeichnet.

Die logischen Ausdrücke eines Entscheidungsbaums sind grundsätzlich *unabhängig* voneinander. Die einzige Beziehung zwischen den logischen Ausdrücken ist die *Reihenfolge*, in der sie geprüft werden. Abbildung 12.1 zeigt das Schema eines einfachen Entscheidungsbaum mit zwei aufeinanderfolgenden Entscheidungen.

Ein oft vorkommender Speziallfall von Entscheidungsbäumen sind verschachtelte Entscheidungen, die so arrangiert sind, dass jeder logische Ausdruck genau ein Ergebnis auswählt.

Definition 12.4. Ein *Entscheidungsbaum*, der für einen logischen Ausdruck mindestens ein Ergebnis und höchstens einen nachfolgende Entscheidung, heisst **linearer Entscheidungsbaum**.

Lineare Entscheidungsbäume können das Ergebnis sowohl für den `Wahr` oder den `Falsch`-Fall festlegen. Per Konvention werden die logischen Ausdrücke linearer Entscheidungsbäume so formuliert, dass die Ergebnisse immer für den Fall `Wahr` und die nachfolgende Entscheidung immer für den Fall `Falsch` folgen. Abbildung 12.2 zeigt das Schema eines linearen Entscheidungsbaums mit zwei Entscheidungen.

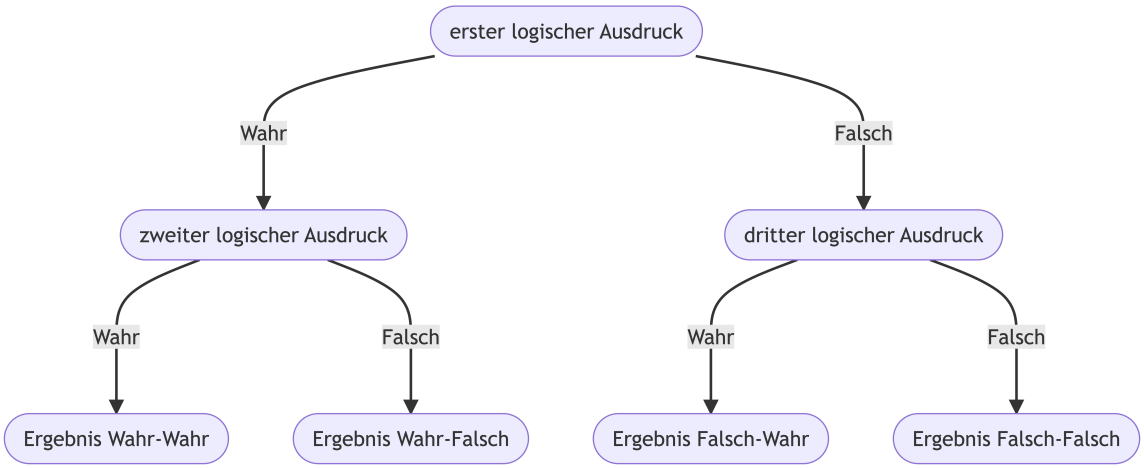


Abbildung 12.1.: Schema eines Entscheidungsbaums mit zwei Entscheidungen

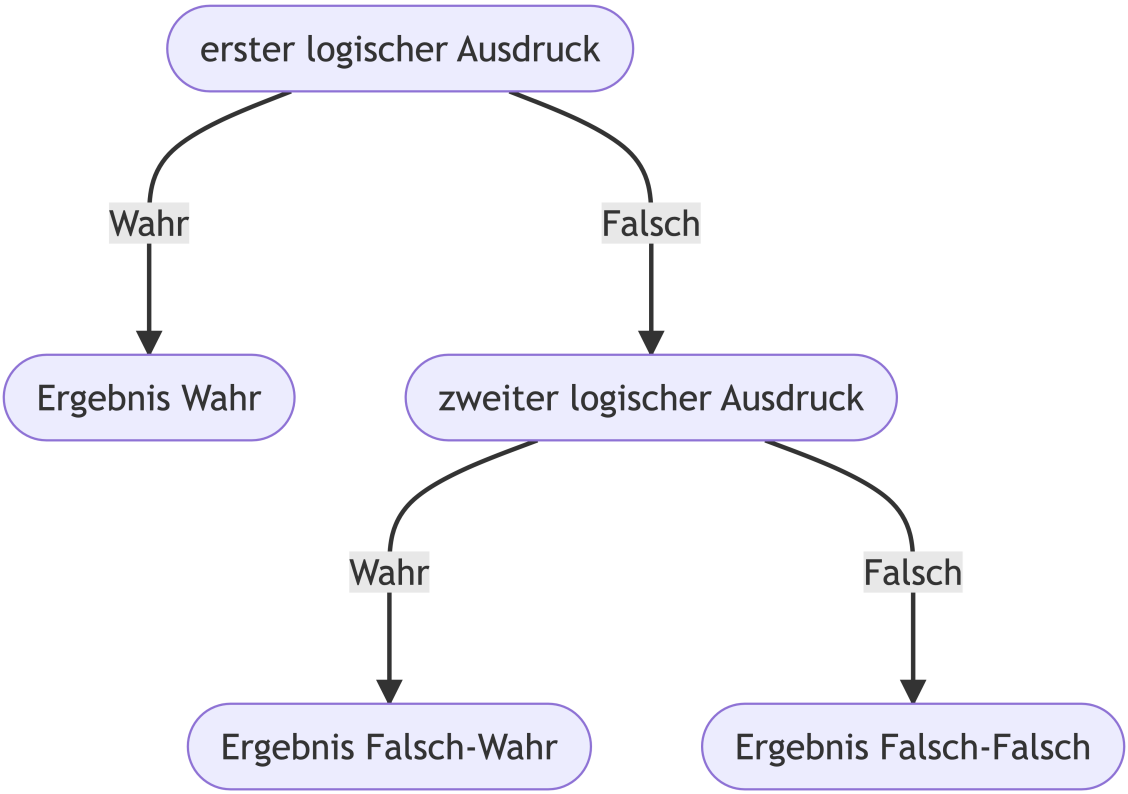


Abbildung 12.2.: Schema eines linearen Entscheidungsbaum mit zwei Entscheidungen

12.5. Filtern

Definition 12.5. Als **Filter** werden Funktionen bezeichnet, die Werte eines Vektors mithilfe eines logischen Ausdrucks auswählen.

Ein Vektor ist gefiltert, wenn der logische Ausdruck für alle Werte `Wahr` ergibt. Als Konsequenz werden alle Werte aus einem Vektor entfernt, für die der logische Ausdruck `Falsch` ergibt.

> **Merke**
>
> Durch das Filtern wird die Länge von Vektoren verändert. Das Ergebnis ist immer höchstens so lang wie der ursprüngliche ungefilterte Vektor.

Der logische Ausdruck muss sich nicht auf die Werte des Vektors beziehen. Damit Werte mit einem solchen logischen Ausdruck ausgewählt werden können, bedarf es einen Referenzvektor mit gleicher Länge. Ein Wert wird mit dieser Technik ausgewählt, wenn der logische Ausdruck für den Wert an der gleichen Position im Referenzvektor `Wahr` ergibt.

Weil die Vektoren von Stichprobenobjekten immer die gleiche Länge haben, lassen sich Filter zum Auswählen von Datensätzen verwenden.

12.6. Sortieren

Definition 12.6. Als **Sortieren** werden Funktionen bezeichnet, die Reihenfolge von Werten mittels eines logischen Ausdrucks bestimmen.

> **Merke**
>
> Durch Sortieren wird die Länge von Vektoren *nicht* verändert.

Die Basis für das Sortieren sind Vektoren. Ein Vektor ist sortiert, wenn der logische Ausdruck für alle Werte paarweise `Wahr` ergibt. Die einfachsten logischen Ausdrücke zum Sortieren sind die Vergleiche *grösser oder gleich* und *kleiner oder gleich*.

Die Reihenfolge der Werte wird beim Sortieren immer dann vertauscht, wenn der logische Ausdruck `Falsch` ergibt. Das `Falsch` bedeutet in diesem Fall, dass die Werte noch nicht in der richtigen Reihenfolge vorliegen.

> **Merke**
>
> Der Vergleich auf *Gleichheit* ist zum Sortieren ungeeignet, weil es keine Reihenfolge gibt, so dass die Gleichheit für alle Wertepaar `Wahr` ergibt.

Grundsätzlich werden 2 Sortierreihenfolgen unterschieden. Diese sind für Zahlen, Zeichenketten und Wahrheitswerte definiert:

1. Aufsteigende Sortierung (engl. ascending)
2. Absteigende Sortierung (engl. descending)

Die Sortierrichtung basiert auf zwei paarweisen Vergleichen zwischen den Elementen. Um die Sortierung zu ändern, muss nur der logische Vergleichsoperator umgekehrt werden.

- Die aufsteigende Sortierung beginnt mit dem kleinsten Wert des Sortierkriteriums und endet mit dem grössten Wert der Sortierung. Dabei gilt für alle Werte des sortierten Vektors die Ungleichung 12.5.

$$v_{Vorgnger} \leq v_{Nachfolger} \tag{12.5}$$

- Die absteigende Sortierung arbeitet genau entgegengesetzt vom grössten Wert des Sortierkriteriums zum kleinsten Wert. Entsprechend gilt für diese Reihenfolge die Ungleichung 12.6.

$$v_{Vorgnger} \geq v_{Nachfolger} \tag{12.6}$$

12.6.1. Sortieren für Fortgeschrittene

Wie beim Filtern können sich die logischen Ausdrücke beim Sortieren auf andere Vektoren beziehen. Dabei wird ebenfalls ein Referenzvektor benötigt. Die Sortierung des Vektors erfolgt entsprechend der Positionen im Referenzvektor. Deshalb müssen Referenzvektoren immer die gleiche Länge wie die Sortiervektoren haben.

Beim Sortieren können komplexe logische Ausdrücke für spezielle Sortiuerungen eingesetzt werden. Dabei muss beachtet werden, dass diese Ausdrücke eine eindeutige Reihenfolge zulassen.

13. Vektoroperationen

Im Kapitel 8 wurden Vektoren unter der allgemeinen Definition 8.15 eingeführt.

Definition 13.1. Die Werte eines Vektors heissen **Elemente**.

Ein Vektor ist eine Datenstruktur mit mehreren Werten, wobei die Anzahl der Werte eine ganze Zahl ≥ 0 ist. Dieser Wert entspricht der Länge des Vektors.

Definition 13.2. Jedes Element hat eine eindeutige Position im Vektor. Diese Position heisst **Index**.

Vektoren sind bezüglich des Datentyps *homogen*. D.h. alle Elemente eines Vektors haben den gleichen Datentyp.

Definition 13.3. Ein Vektor der Länge `0` heisst **leerer Vektor**.

Der leere Vektor wird endweder als $\emptyset$ oder als $\{\}$ geschrieben. Beide Schreibweisen sind gleichwertig.

Der leere Vektor ist eine besondere Datenstruktur, die keine Elemente enthält. Der leere Vektor kann als *Startwert* für die Konstruktion von Vektoren verwendet werden oder als *Ergebnis* von Operationen auf Vektoren vorkommen.

13.1. Sequenzen

Eine besondere Gruppe von Vektoren sind Sequenzen. Sequenzen sind Vektoren deren Werte der Ordnung des Wertebereichs folgen.

$$v_i < v_{i+1}, \text{wenn aufsteigende Reihenfolge } v_i > v_{i+1}, \text{wenn absteigende Reihenfolge} \quad (13.1)$$

Eine Sequenz erfordert also immer einen *ordinalen Wertebereich*.

> **Sequenzen in den Life Sciences**
>
> In den Life Sciences werden *zusammenhängende Abfolgen von Werten* als Sequenzen bezeichnet. In diesen Sequenzen ist die *Reihenfolge* der Werte in diesen Sequenzen von Interesse. Solche Sequenzen haben nicht zwingend einen ordinalen Wertebereich und werden in der Regel als eigenständige Werte und nicht als Vektoren behandelt.

Nachfolgend werden nur Sequenzen vom Datentyp *Zahl* behandelt.

Definition 13.4. Eine **lineare Sequenz** ist ein Vektor, bei dem die Werte aufeinanderfolgender Indizes immer den gleichen Abstand haben.

Deshalb gilt für lineare Sequenzen Formel 13.2.

$$v_{i+1} - v_i = v_{j+1} - v_j \tag{13.2}$$

> **Konvention**
>
> Für die praktische Anwendung sind lineare Sequenzen von zentraler Bedeutung. Deshalb wird im Folgenden der Begriff *Sequenz* synonym für *lineare Sequenz* verwendet. Alle anderen Sequenzen werden als Reihenfolgen bezeichnet oder explizit hervorgehoben.

Definition 13.5. Der *Abstand einer Sequenz* wird als **Schrittweite** bezeichnet. Wird keine Schrittweite für eine Sequenz angegeben, wird die Schrittweite `1` angenommen.

Definition 13.6. Der **Anfangswert** einer Sequenz wird auch **Startwert** oder **Initialwert** genannt. Wird kein Anfangswert für eine Sequenz angegeben, dann wird der Wert `1` angenommen.

Beispiel 13.1 (Anfangswert)**.** Eine Sequenz mit der Länge 5 entspricht dem Vektor `{1,2,3,4,5}`.

Beispiel 13.2 (Länge und Schrittweite)**.** Eine Sequenz mit der Schrittweite 3 und der Länge 4 entspricht dem Vektor `{1,4,7,10}`.

Beispiel 13.3 (Startwert und Länge)**.** Eine Sequenz mit dem Startwert 3 und der Länge 6 entspricht dem Vektor `{3,4,5,6,7,8}`.

Beispiel 13.4 (Startwert, Schrittweite und Länge)**.** Eine Sequenz mit dem Startwert 3, der Schrittweite 3 und der Länge 10 entspricht dem Vektor `{3,6,9,12,15,18,21,24,27,30}`.

> **Merke**
>
> Eine Sequenz mit gleichem Anfangswert und Schrittweite entspricht der jeweiligen Multiplikationsreihe. Der Startwert und die Indizes der Sequenz sind hierbei den Multiplikatoren.

13.1.1. Besondere Sequenzen

Definition 13.7. Der **Einheitsvektor** ist ein Vektor mit der geometrischen Länge 1.

Der *Einheitsvektor* ist *keine* Sequenz, weil die Werte der Indizes nicht mit der gleichen Schrittweite ansteigen.

Definition 13.8. Ein Vektor mit beliebiger Länge heisst **Nullvektor**, wenn an allen Indizes der Wert `0` steht.

Der *Nullvektor* ist damit eine besondere Sequenz, bei der der Startwert und die Schrittweite `0` ist.

Ein zweiter Vektor mit der Schrittweite `0` ist der *Einsvektor*.

Definition 13.9. Der **Einsvektor** ist ein Vektor mit beliebiger Länge mit dem Wert `1` bei jedem Index.

Der *Einsvektor* ist damit eine besondere Sequenz, mit dem Startwert `1` und der Schrittweite `0`.

In der mathematischen Literatur wird der *Einsvektor* oft nicht benannt. Wegen der besonderen Bedeutung dieses Vektors für verschiedene Operationen, wird diese Sequenz hier benannt.

> **Wichtig**
>
> Der **Einsvektor** darf nicht mit dem **Einheitsvektor** verwechselt werden. Im Gegensatz zum Einsvektor hat der *Einheitsvektor* die geometrische Länge 1. Der Einsvektor der Länge `3` hat etwa die geometrische Länge von $\sqrt{3} \approx 1.7321$

13.2. Konkatenation

Datenvektoren können zu neuen Vektoren zusammengefügt werden. Dieser Vorgang heisst **Konkatenation**. Dabei werden zwei Vektoren vom **gleichen Datentyp** *hintereinander* aneinandergehängt.

Bei der Konkatenation muss die Bedingung des **gleichen Datentyps** zwingend erfüllt sein. Eine Konkatenation von Vektoren mit unterschiedlichen Datentypen ist unzulässig, weil die resultierende Datenstruktur kein Vektor mehr wäre.

Beispiel 13.5 (Konkatenation)**.** Die Konkatenation der Vektoren $v = \{1; 2; 3\}$ und $w = \{4; 5; 6\}$ ergibt einen neunen Vektor $v \circ w$. Es gilt dabei die folgende Vorgehensweise:

$$v \circ w = \{1; 2; 3\} \circ \{4; 5; 6\} = \{1; 2; 3; 4; 5; 6\}$$

Der leere Vektor ist das neutrale Element der Konkatenation. D.h. die Konkatenation eines Vektors mit dem leeren Vektor ergibt den ursprünglichen Vektor. Es gilt als Formel 13.3.

$$v \circ \emptyset = \emptyset \circ v = v \tag{13.3}$$

Ein Skalar kann als Vektor mit der Länge `1` aufgefasst werden. Die Konkatenation eines Vektors mit einem Skalart ergibt einen Vektor, der um `1` länger als der ursprüngliche Vektor ist. Es gilt also Formel 13.4.

$$v \circ a \Leftrightarrow v \circ \{a\}$$

$$\begin{aligned} v \circ a &= \{v_1; v_2; \ldots; v_n\} \circ a \\ &= \{v_1; v_2; \ldots; v_n\} \circ \{a\} \\ &= \{v_1; v_2; \ldots; v_n; a\} \end{aligned} \tag{13.4}$$

Weil die Reihenfolge der Konkatenation bedeutungsvoll ist, kann ein einzelner Wert auch *vor* einem Vektor angefügt werden.

$$\begin{aligned} a \circ v &= a \circ \{v_1; v_2; \ldots; v_n\} \\ &= \{a\} \circ \{v_1; v_2; \ldots; v_n\} \\ &= \{a; v_1; v_2; \ldots; v_n\} \end{aligned}$$

13.3. Tranformationen

Definition 13.10. Eine **Transformation** bezeichnet eine Umformung der Element eines Vektors, wobei die Länge des Vektors unverändert bleibt.

Eine Transformation erfordert also immer eine Funktion, die auf die Elemente des Vektors angewandt wird. Diese Funktion wird für jedes Element des Vektors separat ausgeführt. Dadurch ist sichergestellt, dass es für jedes Element des Vektors genau ein Ergebnis gibt.

> **Merke**
>
> Beim Transformieren sind alle Operationen auf die Elemente eines Vektors **unabhängig** voneinander und die Ergebnisse beeinflussen sich nicht gegenseitig.

Beim Transformieren kann der Datentyp eines Vektors verändert werden.

13.3.1. Transformationen mit einem Skalar

Eine Transformation mit einem Skalar wird auch als **Skalierung** bezeichnet.

Für die Skalierung eines Vektors ist eine Transformationsfunktion mit zwei Parametern erforderlich. Beim Skalieren wird die Transformationsfunktion mit dem Skalar mit jedem Element des Vektors separat durchgeführt.

Es gilt also die Logik der Formel 13.5. Der Operator $\circ$ ist hier der Platzhalter für die Transformationsfunktion.

$$\begin{aligned} s \circ v &= s \circ \{v_1; v_2; \dots; v_n\} \\ &= \{s \circ v_1; s \circ v_2; \dots; s \circ v_n\} \end{aligned} \tag{13.5}$$

Beispiel 13.6 (Skalierung mit Zahlen)**.** Die Skalierung eines Vektors $v = \{1; 2; 3\}$ mit dem Skalar $s = 2$ ergibt den Vektor $v' = \{3; 4; 5\}$. Es gilt dabei die folgende Vorgehensweise:

$$\begin{aligned} s + v &= 2 + \{1; 2; 3\} \\ &= \{2 + 1; 2 + 2; 2 + 3\} \\ &= \{3; 4; 5\} \end{aligned}$$

Diese Vorgehensweise wird als Skalaraddition bezeichnet. Die Skalaraddition ist eine Transformation mit einem Skalar und der Addition als Transformationsfunktion.

Analog zur Skalaraddition gibt es die Skalarmultiplikation. Bei der Skalarmultiplikation wird die Multiplikation als Transformationsfunktion verwendet. Die Skalarmultiplikation wird wie folgt durchgeführt.

$$\begin{aligned} s \cdot v &= 2 \cdot \{1; 2; 3\} \\ &= \{2 \cdot 1; 2 \cdot 2; 2 \cdot 3\} \\ &= \{2; 4; 6\} \end{aligned}$$

13.3.2. Transformationen mit einem Vektor

Die Transformation mit einem Skalar kann auf Vektoren erweitert werden. Dabei wird die Transformationsfunktion mit den Elementen eines zweiten Vektors anstelle des Skalars durchgeführt. Eine solche Transformation erfordert, dass die Werte der beiden Vektoren *paarweise* durch die Transformationsfunktion verknüpft werden. Damit solche paarweisen Operationen möglich sind, müssen die beiden Vektoren die *gleiche Länge* haben.

> **Merke**
>
> Die Skalar-Transformation kann nur mit einem Vektor der Länge 1 *oder* mit einem Vektor mit der gleichen Länge wie der zu transformierende Vektor durchgeführt werden.

Dann gilt die Logik der Formel 13.6. Der Operator $\circ$ ist hier der Platzhalter für die Transformationsfunktion.

$$\begin{aligned} v \circ w &= \{v_1; v_2; \dots; v_n\} \circ \{w_1; w_2; \dots; w_n\} \\ &= \{v_1 \circ w_1; v_2 \circ w_2; \dots; v_n \circ w_n\} \end{aligned} \tag{13.6}$$

Beispiel 13.7 (Vektoraddition)**.** Für die beiden gleichlangen Vektoren $v = \{1; 2; 3\}$ und $w = \{4; 5; 6\}$ wird die Vektoraddition paarweise durchgeführt.

$$\begin{aligned} v + w &= \{v_1; v_2; v_3\} + \{w_1; w_2; w_3\} \\ &= \{1; 2; 3\} + \{4; 5; 6\} \\ &= \{1 + 4; 2 + 5; 3 + 6\} \\ &= \{5; 7; 9\} \end{aligned}$$

Deutlicher wird diese Mechanik, wenn die Vektoren als Spaltenvektoren geschrieben werden.

$$v + w = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

$$= \begin{pmatrix} 1+4 \\ 2+5 \\ 3+6 \end{pmatrix}$$

$$= \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}$$

13.4. Aggregationen

Definition 13.11. Eine **Aggregation** ist eine *Funktion*, die Elemente eines Vektors zusammenfasst. Durch aggregieren kann sich die Länge des Vektors verändern.

Eine Aggregationsfunktion heisst *Aggregator*. Aggregatoren haben in der Regel nur einen Parameter, der ein Vektor ist. Ein Aggregator wendet eine zweite Funktion auf die Elemente des Vektors an. Diese zweite Funktion heisst *Reduktionsfunktion*. Die Reduktionsfunktion legt fest, *wie* die Elemente des Vektors zusammengefasst werden.

> **i** Merke
>
> Beim Aggregieren hängen die Operationen von der jeweiligen vorangehenden Operation ab. Die einzelnen Operationen sind voneinander **abhängig** und die Ergebnisse beeinflussen sich gegenseitig.

Beispiel 13.8 (Aggregation und Reduktion beim Summieren)**.** Der $\sum$-Operator ist ein Aggregator. Die dem Operator nachfolgenden Terme legen die zu aggregierenden Vektorelement fest. Die Reduktionsfunktion ist die Addition (+).

Für den Vektor $v = \{1; 2; 3; 4; 5\}$ kann die die Summe der Quadrate wie folgt geschrieben werden.

$$\sum_{i=1}^{5} v_i^2$$

Diese Schreibweise legt fest, dass die Vektorelemente quadriert werden müssen. Dabei handelt es sich um eine Transformation mit einem Skalar (`2`) und der Potenz als Transformationsfunktion. Diese vorgelagerte Transformation erzeugt einen impliziten Vektor mit den quadrierten Elementen $\{1, 4, 9, 16, 25\}$.

Die Reduktionsfunktion der Summe ist die Addition (`+`). Beim Reduzieren werden die Elemente des impliziten Vektors nacheinander addiert. Dabei wird das Element mit dem bisherigen Ergebnis *reduziert*. Beim ersten Schritt liegt noch kein Ergenis vor, weshalb das Element direkt übernommen wird. Dadurch ergeben sich in diesem Beispiel die folgenden Reduktionsschritte.

$$\begin{aligned}
\sum v^2 = \sum_{i=1}^{5} v_i^2 &= 1^2 + 2^2 + 3^2 + 4^2 + 5^2 \\
&= 1 + 4 + 9 + 16 + 25 \\
&= 5 + 9 + 16 + 25 \\
&= 14 + 16 + 25 \\
&= 30 + 25 \\
&= 55
\end{aligned}$$

Die Summe der Quadrate des Vektors v ist also 55.

Im Kapitel 12.5 wurden Filter als Funktionen eingeführt, die Werte aus Vektoren mithilfe eines logischen Ausdrucks auswählen. Das Filtern ist eine spezielle Aggregation, bei dem die ursprünglichen Werte unverändert bleiben, aber die Länge des Vektors verändert wird. Anders als die Summe, liefert das Filtern einen Vektor und keinen einzelnen Wert als Ergebnis. Beim Filtern wird die Reduktionsfunktion als *Selektionsfunktion* bezeichnet. Die Selektionsfunktion fügt einen Wert dem Ergebnisvektor durch *Konkatenation* hinzu, wenn der logische Ausdruck `Wahr` ergibt. Der Startwert für die Reduktion ist beim Filtern der *leere Vektor*.

13.5. Zählen

Definition 13.12. Als **Umfang** bezeichnen wir die Anzahl der Elemente eines Vektors oder einer Liste.

Definition 13.13. Zählen bezeichnet das Bestimmen der Anzahl von Elementen eines Vektors oder einer Liste.

Die Anzahl der Elemente eines Vektors ist immer gleich der Länge des Vektors. Deshalb kann der Umfang eines Vektors direkt aus der Länge des Vektors abgelesen werden.

> **i** Mathematik vs. Datenwisssenschaft
>
> In der Mathematik (Mengenlehre) existiert das Konzept der **Abzählbarkeit**. Dieses Konzept wird auf *beliebige* und insbesondere unendliche Mengen angewandt, um deren abstrakte Umfänge zu vergleichen.
> Beim Rechnen und Problemlösen mit Computern liegen **immer** mit **speziellen, endlichen** Strukturen vor. Damit muss beim Zählen immer die Frage nach dem *konkreten Umfang* der vorliegenden Elemente beantwortet werden.

Nicht immer muss der Umfang des gesamten Vektors bestimmt werden. Stattdessen sollen einzelne Elemente eines Vektors gezählt werden, die bestimmte Bedingungen erfüllen. Eine solche Bedingung wird immer als **logischer Ausdruck** formuliert.

Definition 13.14. Eine **zählbare Einheit** bezeichnet ein Element, für das eine Zählbedingung `Wahr` ergibt. Eine **nicht zählbare Einheit** bezeichnet ein Element, für das eine Zählbedingung `Falsch` ergibt.

- Eine **zählbare Einheit** wird durch den Wert `1` (oder `Wahr`) repräsentiert. Ein zählbares Element wird gezählt.
- Eine **nicht-zählbare Einheit** wird durch den Wert `0` (oder `Falsch`) repräsentiert. Diese Elemente werden nicht gezählt.

13.5.1. Zählen durch Summieren

Beim Zählen durch Summieren geht eine Transformation in die Werte `0` und `1` mithilfe eines logischen Ausdurcks einer Summen-Aggregation unmittelbar voraus.

Gelegentlich werden nicht alle Elemente einer Datenstruktur gezählt, sondern nur diejenigen, die bestimmte Bedingungen erfüllen.

Definition 13.15. Es wird vom **Abzählen** gesprochen, wenn zum Zählen eine **Summe über die zählbaren Einheiten** gebildet wird.

Dabei wird ausgenutzt, dass die Summe über einen Vektor von `0` und `1` die Anzahl der `1`-Werte ergibt.

> **i** Merke
>
> Immer wenn eine Summe über einen Vektor von `0` und `1` gebildet wird, wird eine Zählen-Operation vorbereitet.

Die Transformation in die Werte `0` und `1` durch einen logischen Ausdruck wird als **Indikatorfunktion** bezeichnet. Die Indikatorfunktion ist eine Transformation mit

einem logischen Ausdruck. Eine solche Funktion kann für nachgelagerte arithmetische Transformations-Operationen als Ersatz für eine vorgelagerte Entscheidung verwendet werden. In solchen Fällen wird das Ergebnis der Indikatorfunktion über eine Skalar-Multiplikation mit der nachgelagerten Operation verknüpft. Dabei stellt die Indikatorfunktion sicher, dass der Vektor die gleiche Länge wie die nachgelagerte Operation hat.

13.5.2. Zählen durch Filtern

Beim Zählen durch Filtern wird die Summe über die zählbaren Einheiten durch eine Filter-Aggregation zusammegefasst. Nach dem Filtern bleibt ein Vektor mit den zählbaren Einheiten übrig, der keine nicht-zählbaren Elemente enthält. Die Länge dieses Vektors entspricht der Anzahl der zählbaren Einheiten.

> **Merke**
>
> Werden die zählbaren Einheiten durch einen Filter ausgewählt, dann entspricht die Summe der zählbaren Einheiten der Länge des gefilterten Vektors.

Weil das Filtern die Länge des Vektors verändert, kann das Filtern **nicht** als *Indikatorfunktion* verwendet werden.

13.5.3. Zählen durch Nummerieren

Beim Nummerieren wird eine Sequenz erzeugt, die für jede zählbare Einheit einen Wert enthält. Die Länge dieser Sequenz entspricht der Anzahl der zählbaren Einheiten. Gleichzeigit entspricht der Maximal-Wert der Sequenz ebenfalls der Anzahl der zählbaren Einheiten.

> **Merke**
>
> Wenn die zählbaren Einheiten durchnummeriert werden, dann entspricht der Umfang dieser Einheiten der grössten Nummerierung (dem Maximum).

Nummerierungen werden häufig verwendet, um einzelnen Datensätzen eindeutige Identifikatoren zuzuweisen. Diese Nummerierungen können zum Zählen leicht benutzt werden.

14. Matrix-Operationen

Matrizen sind für die Datenwissenschaften eine besondere Datenstruktur. In diesem Kapitel wird Definition 8.19 erweitert, um die wichtigsten Matrixoperationen zu behandeln.

Für Matrizen gelten etwas andere Regeln als für einzelne Werte. Bei genauer Betrachtung lässt sich feststellen, dass es sich dabei um Verallgemeinerungen der bekannten Rechenregeln handelt.

Dazu stellen wir uns vor, dass ein einzelner Wert auch als eine 1x1-Matrix dargestellt werden kann. 1x1-Matrizen sind gleichzeitig die kleinsten quadratischen Matrizen.

> **Hinweis**
>
> Für alle Rechenregeln muss beachtet werden, dass die Grösse aller verwendeten Matrizen für die jeweilige Operation entscheidend ist.

14.1. Grundbegriffe

Definition 14.1. Die **Hauptdiagonale** einer Matrix sind alle Positionen mit gleichen Zeilen- und Spaltenindizes.

Definition 14.2. Eine **symmetrische Matrix** ist eine quadratische Matrix, die bezüglich der Hauptdiagonalen symmetrisch ist.

Daraus folgt, dass die Werte einer symmetrischen Matrix an einer Position (i, j) gleich den Werten an der Position (j, i) sind.

Beispiel 14.1 (Symmetrische Matrix)**.**

$$\begin{bmatrix} 1 & 0 & 6 & 4 & 9 \\ 0 & 2 & 2 & 6 & 14 \\ 6 & 2 & 3 & 7 & 8 \\ 4 & 6 & 7 & 4 & 9 \\ 9 & 14 & 8 & 9 & 5 \end{bmatrix}$$

Definition 14.3. Eine **Diagonalmatrix** hat nur Werte auf der Hauptdiagonalen. Alle anderen Werte sind `0`.

Definition 14.4. Die **Einheitsmatrix** ist eine Diagonalmatrix mit `1` entlang der Hauptdiagonalen.

Beispiel 14.2 (Einheitsmatrix mit 3 Zeilen und 6 Spalten)**.**

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Definition 14.5. Die **Identitätsmatrix** ist eine quadratische Einheitsmatrix.

Die Identitätsmatrix wird in Formeln als I gekennzeichnet.

Beispiel 14.3 (Identitätsmatrix)**.**

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Definition 14.6. Eine **Dreiecksmatrix** ist eine Matrix, die nur Werte unterhalb oder oberhalb der Hauptdiagonalen hat.

Die Hauptdiagonale einer Dreiecks-Matrix wird nicht immer zum Wertebereich hinzugezählt. Je nach Anforderung, muss dieser Wertebereich entsprechend ein- oder ausgeschlossen werden.

Beispiel 14.4 (Untere Dreiecksmatrix)**.**

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 11 & 2 & 0 & 0 & 0 \\ 6 & 2 & 3 & 0 & 0 \\ 4 & 6 & 7 & 4 & 0 \\ 9 & 14 & 8 & 9 & 5 \end{bmatrix}$$

Beispiel 14.5 (Obere Dreiecksmatrix)**.**

$$\begin{bmatrix} 1 & 11 & 6 & 4 & 9 \\ 0 & 2 & 2 & 6 & 14 \\ 0 & 0 & 3 & 7 & 8 \\ 0 & 0 & 0 & 4 & 9 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Definition 14.7. Eine **dünnbesetzte Matrix** (oder *Sparse Matrix*) ist eine Matrix, die überwiegend `0`-Werte enthält.

Beispiel 14.6 (Sparse Matrix)**.**

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 2 & 6 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 4 & 0 & 3 & 0 \end{bmatrix}$$

Definition 14.8. Eine Matrix, welche die Werte einer anderen Matrix A mit vertauschten Zeilen- und Spaltenindizes enthält, wird **transponierte Matrix** von A genannt.

Für die *transponierte Matrix von* A wird A^T geschrieben.

Beispiel 14.7 (Transponierte Matrix)**.**

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 6 & 2 \\ 4 & 6 \\ 9 & 7 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 0 & 6 & 4 & 9 \\ 0 & 2 & 2 & 6 & 7 \end{bmatrix}$$

Wir beachten, dass per Konvention der Zeilenindex immer als Erstes und der Spaltenindex immer als Zweites angegeben wird. Anstelle der mathematischen Schreibweise trennen wir die beiden Indizes. Auf diese Weise können wir auf jeden Wert in einer Matrix zugreifen.

14.2. Matrixaddition

Die Matrizenaddition addiert die Elemente zweier Matrizen *paarweise*. Damit die Addition funktioniert, muss es für jeden Wert in Matrix A einen Partnerwert in Matrix B mit gleichem Zeilenindex i und Spaltenindex j geben. Daraus folgt direkt, dass die Matrixaddition nur für Matrizen mit gleichen Dimensionen m und n definiert ist.

14.3. Vektoraddition

Die Vektoraddition funktioniert etwas anders als die Matrixaddition. In diesem Fall liegt uns eine m x n-Matrix und ein m-Vektor vor.

Die Vektoraddition ist nur dann definiert, wenn die Matrix und der Vektor die gleiche Anzahl an Zeilen haben.

Bei der Vektoraddition wird der Vektor zu jeder Spalte in der Matrix addiert. Dabei werden die Werte *paarweise* zusammengezählt.

::: {#exm-matrix-vektoraddition} ## Vektoraddition eines 2-Vektor und eine 2 x 3-Matrix.

$$v + M = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{bmatrix}$$

$$= \begin{bmatrix} v_1 + m_{11} & v_1 + m_{12} & v_1 + m_{13} \\ v_2 + m_{21} & v_2 + m_{22} & v_2 + m_{23} \end{bmatrix}$$

14.4. Skalarmultiplikation (Punktprodukt)

Die Skalarmultiplikation oder das Punktprodukt multipliziert einen Wert a mit einer Matrix (oder Vektor) M. Dabei wird \(a \) als **Skalar** bezeichnet, weil dieser alle Werte um den gegebenen Wert *skaliert*. Bei der Skalarmultiplikation wird jeder Wert in der Matrix mit dem Skalar multipliziert.

$$a \cdot M = a \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{bmatrix}$$

$$= \begin{bmatrix} a \cdot m_{11} & a \cdot m_{12} & a \cdot m_{13} \\ a \cdot m_{21} & a \cdot m_{22} & a \cdot m_{23} \end{bmatrix}$$

Dieses Konzept lässt sich auf Vektoren übertragen. Dabei ist der Skalar a ein Vektor mit der gleichen Anzahl an Zeilen für den Vektor und die Matrix. Danach funktioniert die Skalarmultiplikation analog zur Vektoraddition.

$$a \cdot M = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{bmatrix}$$

$$= \begin{bmatrix} a_1 \cdot m_{11} & a_1 \cdot m_{12} & a_1 \cdot m_{13} \\ a_2 \cdot m_{21} & a_2 \cdot m_{22} & a_2 \cdot m_{23} \end{bmatrix}$$

14.5. Matrixmultiplikation/ Kreuzprodukt

Das Kreuzprodukt ist eine andere Variante zwei Matrizen zu multiplizieren. Dabei werden zwei Matrizen A und B über Kreuz multipliziert. Dazu muss gegeben sein, dass die Matrix A gleich viel Spalten hat, wie Matrix B Zeilen. Es muss also gelten, dass wir eine m x n-Matrix

mit einer n x p-Matrix multiplizieren, wobei n für beide Matrizen gleich sein muss. Sind diese Voraussetzungen nicht erfüllt, kann das Kreuzprodukt nicht gebildet werden.

Das Kreuzprodukt ist wie folgt definiert:

$$A \times B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} a_{1i} \cdot b_{i1} & \sum_{i=1}^{n} a_{1i} \cdot b_{i2} & \cdots & \sum_{i=1}^{n} a_{1i} \cdot b_{ip} \\ \sum_{i=1}^{n} a_{2i} \cdot b_{i1} & \sum_{i=1}^{n} a_{2i} \cdot b_{i2} & \cdots & \sum_{i=1}^{n} a_{2i} \cdot b_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} a_{mi} \cdot b_{i1} & \sum_{i=1}^{n} a_{mi} \cdot b_{i2} & \cdots & \sum_{i=1}^{n} a_{mi} \cdot b_{ip} \end{bmatrix}$$

Das Ergebnis eines Kreuzprodukts ist immer eine Matrix mit m-Zeilen und p-Spalten.

Aus der Definition des Kreuzprodukts zeigt sich, dass die Operanden beim Kreuzprodukt nur vertauscht werden können, wenn beide Matrizen quadratisch sind. Dabei gilt für beliebige Matrizen ausserdem Formel 14.1.

$$A \times B \neq B \times A \tag{14.1}$$

Das Kreuzprodukt hat ein *neutrales Element*: Die **Identitätsmatrix** I. Die Identitätsmatrix ist eine quadratische Matrix, die an den Positionen der abfallenden Diagonalen den Wert 1 und sonst den Wert 0 hat.

Beispiel 14.8 (3-dimensionale Einheitsmatrix)**.**

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Für die Identitätsmatrix gilt Formel 14.2.

$$A \times I = I \times A = A \tag{14.2}$$

14.5.1. Kreuzprodukt für Vektoren

Aus der Anforderung für das Kreuzprodukt folgt direkt das Kreuzprodukt für Vektoren, die wir als m x 1- sowie als 1 x p-Matrizen verstehen können. Es gilt also: n = 1. Deshalb mussten wir für das Einmaleins-Beispiel den zweiten Vektor *transponieren*. Dadurch vereinfacht sich die komplizierte Formel des Kreuzprodukts auf die paarweise-überkreuzte Multiplikation.

14.5.2. Inverse Matrix

Definition 14.9. Die **inverse Matrix** A^{-1} ist die Matrix, die mit der Matrix A multipliziert die Identitätsmatrix I ergibt. Es gilt also Formel 14.3.

$$A \times A^{-1} = A^{-1} \times A = I \tag{14.3}$$

Die inverse Matrix kann nicht für beliebige Matrizen gebildet werden, sondern ist nur für bestimme quadratische Matrizen definiert. Die Inverse Matrix wird für verschiedene Anwendungen benötigt und wird deshalb von den meisten Softwarepaketen bereitgestellt. Es ist deshalb selten notwendig, eine inverse Matrix selbst zu berechnen.

14.5.3. Anwendungen des Kreuzprodukts

14.5.3.1. Zeilen- und Spaltensummen berechnen

Für Zeilen- bzw. Spaltensummen werden drei Eigenschaften ausgenutzt:

1. Vektoren lassen sich als $1 \times n$- bzw. $n \times 1$-Matrizen verstehen.
2. Das Ergebnis des Kreuzpodukts ist immer eine Matrix mit $m \times p$-Dimensionen.
3. Das neutrale Element der Multiplikation ist `1`.

Weil für jede Ergebnisposition des Kreuzprodukts Formel 14.4 gilt.

$$\sum_{i=1}^{n} a_{ji} \cdot b_{ik} \tag{14.4}$$

Wird einer der beiden Parameter a_{ji} oder b_{ik} eins, dann reduziert sich dieser Term auf eine einfache Summe entsprechend Formel 14.5.

$$\sum_{i=1}^{n} a_{ji} \cdot b_{ik} = \sum_{i=1}^{n} a_{ji} \cdot 1 = \sum_{i=1}^{n} a_{ji} \tag{14.5}$$

Für die Spalten- bzw. Zeilensumme soll jeweils ein Vektor bestimmt werden, der die Summe der Werte in der jeweiligen Zeile bzw. Spalte enthält. Dazu wird die Matrix mit einem geeigneten Einsvektor multipliziert.

Für die Spaltensumme, muss dieser Einsvektor die Länge der Anzahl der Zeilen haben. Für die Zeilensumme muss der Einsvektor die Länge der Anzahl der Spalten haben.

Beispiel 14.9 (Spaltensumme für eine 3×4-Matrix)**.**

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 6 & 2 & 1 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 11 & 7 & 6 \end{bmatrix}$$

Beispiel 14.10 (Zeilensumme für eine 3×4-Matrix)**.**

$$\begin{bmatrix} 3 & 6 & 2 & 1 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 10 \end{bmatrix}$$

14.5.3.2. Vorgänger- und Nachfolgersummen

Wird ein Vektor mit einer Dreiecks-Matrix multipliziert, dann werden die Vorgänger- bzw. Nachfolgerwerte addiert.

Beispiel 14.11 (Nachfolgersumme für einen Vektor der Länge 4)**.**

$$\begin{bmatrix} 3 & 6 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 9 & 3 & 1 \end{bmatrix}$$

Diese Logik lässt sich auch auf Matrizen erweitern. Dabei werden die Vorgänger- bzw. Nachfolgerwerte für jede Zeile berechnet.

Beispiel 14.12 (Nachfolgersumme für eine 2×4-Matrix)**.**

$$\begin{bmatrix} 3 & 6 & 2 & 1 \\ 4 & 3 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 9 & 3 & 1 \\ 10 & 6 & 3 & 1 \end{bmatrix}$$

14.6. Das äussere Matrixprodukt

Bei den Vektoren und Matrizen haben wir bereits das Kreuzprodukt kennengelernt, mit dem wir aus zwei Vektoren eine Matrix konstruieren können. Das Kreuzprodukt ist allerdings nur für die Multiplikation als paarweiser Operator definiert. Neben dem Kreuzprodukt gibt es noch eine zweite und flexiblere Möglichkeit, um aus zwei Vektoren eine Matrix zu generieren: Das sog. äussere Produkt.

Beim **äusseren Produkt** oder *dyadischen Produkt* werden die Werte zweier *Vektoren paarweise* miteinander verknüpft, wodurch eine Matrix erzeugt wird, bei der die Anzahl der Zeilen entsprechend der Länge des linken und die Anzahl der Spalten mit der Länge des rechten Vektors übereinstimmt.

Wenn also zwei Vektoren x mit der Länge m und y mit der Länge n gegeben sind, dann können wir das äussere Produkt wie folgt schreiben:

$$x \otimes y = M^{m \times n}$$

Das Ergebnis des äusseren Produkts ist also immer eine Matrix. Wir beachten, dass die Definition nur eine Verknüpfung fordert, aber nicht festlegt, welche Verknüpfung verwendet werden soll. Wir können also *beliebige Operationen* zur Verknüpfung verwenden.

Wird als Verknüpfungsoperator für das äussere Produkt die Multiplikation gewählt, dann entspricht das Ergebnis des äusseren Produkts für zwei Vektoren dem Kreuzprodukt zwischen Vektoren.

Anders als beim Kreuzprodukt, können beim äusseren Matrixprodukt beliebige Operatoren verwendet werden. In der Praxis werden sehr oft Vergleichsoperatoren für die Verknüpfung verwendet.

Beispiel 14.13 (Eine Dreiecks-Matrix mit dem äusseren Produkt erzeugen.)**.** Eine Dreiecks-Matrix eine wichtige Matrix, weil durch die `0` ober- bzw. unterhalb der Hauptdigonalen bei der Matrixmultiplikation viele Rechenoperationen eingespart werden können. Diese Einsparung kommt in der Praxis immer dann zum Tragen, wenn nur Vorgänger- oder Nachfolgerwerte berückstichtigt werden müssen. Dafür ist es oft sinnvoll, eine Dreiecks-Matrix zu erzeugen, die nur die Werte `1` und `0` enthält.

Eine Dreiecksmatrix lässt sich mit dem äusseren Produkt aus zwei Sequenzen erzeugen. Dazu wird in zwei Schritten vorgegangen.

1. Eine Sequenz wird mit der gewünschten Länge erzeugt.
2. Die Sequenz wird mit sich selbst multipliziert. Als Operator wird ein Vergleichsoperator verwendet, der die Werte der Sequenz paarweise vergleicht.

Eine Dreiecks-Matrix entsteht, wenn ein grösser- oder kleiner-Vergleich durchgeführt wird. Soll die Hauptdiagonale einbezogen werden, dann jeweils ein grösser-oder-gleich- bzw. kleiner-oder-gleich-Vergleich durchgeführt werden.

Um eine 3×3-Matrix zu erzeugen, wird eine Sequenz mit der Länge 3 benötigt.

$$\{1; 2; 3\}$$

Diese Sequenz wird mit sich selbst multipliziert. Als Operator wird ein grösser-Vergleich verwendet.

$$\{1; 2; 3\} \otimes_{>} \{1; 2; 3\} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Weil das äussere Produkt beliebige Operatoren erlaubt, lassen sich mit der gleichen Technik wie für die Dreiecks-Matrix auch andere Strukturen systematisch erzeugen. Falls dabei komplexe logische Ausdrücke verwendet werden, ist es sinnvoll, diese in einer Funktion zu kapseln.

Beispiel 14.14 (Eine komplexe Struktur mit dem äusseren Produkt erzeugen.)**.** Es soll eine 6×6-Matrix erzeugt werden, die entlang Hauptdiagonalen und der direkt benachbarten Nebendiagonalen den Wert `1` enthält. Alle anderen Werte sollen `0` sein.

Dazu wird Sequenz der Länge `6` als Basis für das äussere Produkt verwendet.

$$\{1; 2; 3; 4; 5; 6\}$$

Weil diese Struktur nicht durch einen direkten Vergleich erzeugt werden kann, wird eine Funktion f für den logischen Ausdruch erstellt. Daraus ergibt sich das folgende äussere Produkt.

$$\{1; 2; 3; 4; 5; 6\} \otimes_f \{1; 2; 3; 4; 5; 6\}$$

Um die gewünschte Struktur zu erzeugen, bedarf es eines komplexen logischen Ausdrucks mit zwei Vergleichen. Zum einen muss für die Struktur der eine Index kleiner oder gleich dem um eins erhöhten anderen Index sein. Zum anderen darf die Differenz zwischen den beiden Indizes nicht grösser als eins sein. Daraus ergibt sich die folgende Definition für die Funktion f.

$$f(i, j) \to (i \leq j + 1) \wedge (j - i \leq 1)$$

Das Ergebnis des äusseren Produkts mit dieser Funktion als Operator ist die gewünschte Matrix.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Das äussere Produkt erlaubt beliebige Operatoren für die Multiplikation und ist deshalb sehr flexibel. Weil Operatoren nur spezielle Funktionen sind, lassen sich komplexe Operatoren z.B. mittels logischer Ausdrücke erzeugen.

Beispiel 14.15 (Werte in einem Vektor markieren)**.** Häufig ist es notwendig, bestimmte Werte in einem Vektor zu markieren, damit sie von nachfolgenden Matrix-Operationen berücksichtigt werden können. Ein typisches Beispiel für solche Markierungen ist das Markieren der einzelnen Werte eines diskreten Wertebereichs.

Dazu wird zuerst ein Hilfsvektor erstellt, der alle (vorkommenden) Werte des Wertebereichs enthält. Anschliessend wird das äussere Produkt zwischen dem originalen Vektor und dem Hilfsvektor mit einem Vergleichsoperator durchgeführt.

Hier soll der Wertebereich die Werte 1 bis 5 enthalten. Daraus ergibt sich der Hilfsvektor $v_h = \{1; 2; 3; 4; 5\}$. Der Vektor v soll markiert werden. Dieser Vektor enthält die Werte $\{1; 3; 2; 3; 4; 5; 3; 2; 4; 4; 5; 1; 4\}$.

Als Vergleichsoperator wird die Gleichheit verwendet, um die Werte zu markieren. Das Ergebnis ist die folgende Matrix.

$$v \otimes_{=} v_h = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Nun lässt sich bspw. die Spaltensumme entsprechend von Beispiel 14.9 mithilfe eines Einsvektors v_1 der Länge `13` berechnen, um die Häufigkeit der einzelnen Werte zu bestimmen.

$$v_1 \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 & 4 & 2 \end{bmatrix}$$

14.7. Co-Occurence-Matrizen erzeugen

Co-Occurance Matrizen helfen beim Feststellen der Häufigkeiten des gemeinsamen Auftretens von Werten in zwei Vektoren. Das Ergebnis zeigt an den einzelnen Positionen jeweils die Anzahl steht mit der die korrespondierenden Werte der jeweiligen Spalte bzw. Zeilen gleichzeitig vorkommen. Bei der Erstellung einer Co-Occurrence-Matrix müssen zuerst die zählbaren Elemente identifiziert werden. Das wird durch das äussere Produkt erreicht. Das erfolgt in drei Schritten.

1. Es werden zwei Vektoren erstellt, in dem die Werte aus den Vektoren genau einmal vorkommen.
2. Es werden zwei Matrizen über das äussere Produkt des jeweiligen original Vektors und dem zugehörigen eindeutigen Vektor aus Schritt 1 erstellt. Der Operator für das äussere Produkt ist in diesem Fall der Gleichheitsoperator.
3. Mit den beiden Matrizen aus Schritt 2 wird das Kreuzprodukt gebildet. Dazu muss eine der beiden Matrizen transponiert werden. Dieses Kreuzprodukt ist immer zulässig, weil beide Matrizen die gleiche Anzahl an Zeilen haben, wenn sie aus dem gleichen Datenrahmen gebildet wurden.

Die Schritte 1 und 2 sind eine direkte Anwendung von Beispiel 14.15.

Das Ergebnis ist eine Matrix mit der Anzahl der gemeinsamen Vorkommen der Werte aus den beiden eindeutigen Vektoren aus Schritt ein. Die Werte lassen sich über das Kreuzprodukt zuordnen:

- Die Spalten entsprechen den eindeutigen Werten der linken (nicht transponierten) Matrix.
- Die Zeilen entsprechen den eindeutigen Werten der rechten (transponierten) Matrix.

Alternativ zur Position der Werte kann auch ein *Sekundärindex* (s.Kapitel 15) für das äussere Produkt verwendet werden.

Eine Co-occurance-Matrix ist immer symmetrisch. Durch eine Skalarmultiplikation mit einer Dreiecks-Matrix aus Beispiel 14.13 kann die Matrix in eine untere oder obere Dreiecksmatrix umgewandelt werden.

15. Indizieren und Gruppieren

⚠ Work in Progress

15.1. Indizieren

Definition 15.1. Ein **Index** bezeichnet Werte, mit denen sich ein oder mehrere Datensätze von anderen Datensätzen unterscheiden lassen.

Definition 15.2. Mit **Indizieren** wird die Arbeitsweise von Algorithmen bezeichnet, mit der Datensätze *identifiziert* werden.

Es werden drei Arten von Indizes unterschieden:

1. Der **Primärindex**, mit dem ein einzelner Datensatz eindeutig *identifiziert* werden kann.
2. **Sekundärindizes** erlauben *Querverweise* auf eine zweite Stichprobe (eine sog. *Indextabelle* oder engl. *Lookup-Table*).
3. **Sekundärindizes** erlauben die Identifikation von Datensätzen mit *gemeinsamen Eigenschaften*. Diese Indizes werden auch als *Gruppenindex* bezeichnet.

Weil ein Index Werte über einen Datensatz enthält, gehört ein Index zum jeweiligen Datensatz und wird über einen *Indexvektor* in einer Stichprobe abgebildet.

Definition 15.3. Ein **Indexvektor** bezeichnet einen Vektor, mit dessen Werten Datensätze identifiziert werden können.

Definition 15.4. Ein **Hash** bezeichnet einen Wert in einem Indexvektor.

15.1.1. Existierende Indexvektoren.

Häufig liegen Indexvektoren bereits in einer Stichprobe vor.

15.1.1.1. Beispiel bestehende Primär- und Sekundärindizes

Tabelle 15.1.: Ausschnit der `mtcars`-Stichprobe mit Primär- und Sekundärindizes

modell	mpg	cyl	disp	vs	am
Mazda RX4	21.0	6	160.0	0	1
Mazda RX4 Wag	21.0	6	160.0	0	1
Datsun 710	22.8	4	108.0	1	1
Hornet 4 Drive	21.4	6	258.0	1	0
Hornet Sportabout	18.7	8	360.0	0	0
Valiant	18.1	6	225.0	1	0
Duster 360	14.3	8	360.0	0	0
Merc 240D	24.4	4	146.7	1	0
Cadillac Fleetwood	10.4	8	472.0	0	0
Fiat 128	32.4	4	78.7	1	1
Honda Civic	30.4	4	75.7	1	1

Der Vektor `modell` ist der **Primärindex**, weil dieser Vektor nur Werte enthält, die einen Datensatz eindeutig identifizieren.

Die Vektoren `cyl` (Zylinder), `vs` (Motortyp) und `am` (Automatikschaltung) sind *Sekundärindizes* und *Gruppenindizes*, die Modelle nach verschiedenen Kriterien zusammenfassen.

15.1.2. Fehlende Indexvektoren

Gelegentlich liegen in einer Stichprobe keine Primär- oder Sekundärindizes vor oder die vorhandenen Indizes erlauben keine Zusammenfassungen für eine konkrete Fragestellung. In solchen Fällen muss ein entsprechender Index erzeugt werden.

Definition 15.5. Eine Funktion, die *Hashes* für Indexvektoren erzeugt, heisst **Hashing-Funktion**.

Hashing-Funktionen werden in der Industrie als Unterstützung zur Suche von Datensätzen in Datenbanken eingesetzt. Durch die geschickte Berechnung von Hashes beschleunigen diese Funktionen die Suche einzelner Werte um ein Vielfaches, indem sie den Bereich für die Suche einschränken. Deshalb haben viele Hashing-Funktionen ein anderes **Anwendungsziel** als die hier beschriebenen Hashing-Funktionen.

15.1.3. Hashing zur Identifikation

Die einfachste Technik zur eindeutigen Indizierung ist das ***Durchnummerieren*** der Datensätze einer Stichprobe. Bei dieser Technik wird jedem Datensatz eine Nummer zugewiesen. In R verwenden wir dazu die Funktion `row_number()`. Diese Funktion ist einer *Sequenz* vorzuziehen, weil diese Funktion auch bei leeren Stichproben fehlerfrei arbeitet.

In Excel muss zum Durchnummerieren die `SEQUENZ()`-Funktion verwendet werden. Das erreichen wir mit der folgenden Operation: `=SEQUENZ(ZEILEN(StichprobenBereich))`, wobei `StichprobenBereich` eine Excel-Adresse sein muss. Weil mit Excel keine leeren Stichproben erzeugt werden dürfen, gibt es mit Excel nicht das gleiche Problem wie mit R. Wegen dieser Eigenschaft muss ein entsprechender Bereich mindestens einen Stichprobenumfang von eins haben. Diese Eigenschaft gilt auch für Tabellen, die nur aus Überschriften bestehen.

> **Fingerübung:** Nummerieren Sie die Stichprobe `mtcars` und speichern Sie die Nummern im Vektor `nr`.

15.1.4. Hashing zum Gruppieren

Beim Hashing zum Gruppieren müssen wir Werte erzeugen, die eine Zuordnung zu einer Gruppe oder einen Wert in einer anderen Stichprobe ermöglichen. Die Hashing-Funktion orientiert sich dabei an den konkreten Analyseanforderungen.

Vier gängige Techniken können dabei unterschieden werden:

- Kodieren (alle Datentypen)
- Reihenfolgen bilden durch Ganzzahldivision (nur Zahlen)
- Reihenfolgen bilden durch Modulo-Operation (nur Zahlen)
- Reihenfolgen durch Anfangsbuchstaben (nur Zeichenketten)

15.1.4.1. Beispiel Hashing zum Gruppieren.

Das folgende Beispiel bildet einen Index, um die Motorisierung der Fahrzeugtypen in der Stichprobe `mtcars` zu bestimmen. Dabei sollen die Modelle in schwach-, mittel-, stark- und sehr starkmotorisierte Typen unterschieden werden. Die Motorisierung richtet sich dabei zum einen nach der Leistung (`hp`). Zum anderen richtet sich die Motorisierung nach dem Fahrzeuggewicht (`wt`), weil für ein schweres Fahrzeug mehr Leistung zum Bewegen benötigt wird als für ein leichtes. Um beide Werte zu berücksichtigen, wird das Verhältnis der beiden Werte bestimmt. Ein Verhältnis ist eine *Division*. In diesem Fall wird das Gewicht als Nenner verwendet und die Leistung als Zähler. So ergeben sich immer Werte grösser als 1, weil die Leistung immer viel grösser als das Gewicht ist.

In diesem Beispiel besteht die Hashing-Funktion aus zwei Teilen:

1. Das Verhältnis zwischen Leistung und Gewicht wird bestimmt und im Vektor `verhaeltnis` abgelegt.

2. Die Leistungsklassen werden durch *Kodieren* den oben festgelegten Klassen zugewiesen und im Vektor `klasse` gespeichert.

```
mtcars %>%
    as_tibble(rownames = "modell") %>%
    mutate(
        verhaeltnis = hp/wt,
        klasse = case_when(
            verhaeltnis > 60 ~ "sehr stark",
            verhaeltnis > 50 ~ "stark",
            verhaeltnis > 40 ~ "mittel",
            TRUE ~ "schwach")
    )
```

Fingerübung: Bestimmen Sie die Effizienz der Modelle in `mtcars` indem Sie den Verbrauch (`mpg`) und den Hubraum (`disp`) berücksichtigen. Legen Sie Effizienzklassen fest.

15.1.5. Hashing für Querverweise

Beim Hashing für Querverweise gibt es zwei Stichproben. Die erste Stichprobe ist die Hauptstichprobe mit den eigentlichen Werten. Die zweite Stichprobe ist die Referenzstichprobe, die zusätzliche Informationen enthält. Ein Indexvektor für Querverweise in der ersten Stichprobe bezieht sich immer auf einen Primärindex aus der zweiten Stichprobe.

Die Hashing-Funktion muss deshalb einen Verweis zur zweiten Stichprobe herstellen. Diese Verbindung kann mit der gleichen Strategie erzeugt werden, wie beim Gruppieren. Dabei muss jedoch darauf geachtet werden, dass alle Zuordnungen des Primärvektors korrekt abgebildet sind.

15.2. Datensätze randomisieren

Wenn wir mit Teilstichproben arbeiten und diese mit anderen teilen, müssen wir vermeiden, dass zwei Stichproben leicht zusammengesetzt werden können und so Rückschlüsse über die Probanden möglich werden.

> **!** Wichtig
>
> Sobald personenbezogene Daten statistisch ausgewertet und zur Publikation vorbereitet werden, **müssen** die Daten randomisiert werden!

Dieses kleine Rezept beschreibt eine Technik zur Anonymisierung von Daten durch Mischen. Entscheidend bei dieser Technik ist, dass wir die Werte für unsere Analyse zusammenhalten

möchten, sodass unsere Ergebnisse nachvollziehbar bleiben. Gleichzeitig soll es unmöglich werden, diese Daten mit anderen Teilen unserer Studien in Verbindung zu bringen.

Die Technik der Anonymisierung durch Mischen besteht aus vier Schritten:

1. Auswahl der Vektoren, die wir in einer Publikation teilen möchten,
2. Erzeugung eines eindeutigen Vektors,
3. zufälliges Mischen,
4. Entfernen der eindeutigen Vektoren und exportieren der Daten.

15.2.0.1. Schritt 1: Auswahl der Vektoren

Wie üblich wählen wir die Vektoren mit der `select()`-Funktion aus.

15.2.0.2. Schritt 2: Erzeugung eines eindeutigen Vektors

Alle unsere Werte müssen zusammengehalten werden, weil unsere Analyse sonst nicht mehr nachvollziehbar ist. Wir nummerieren dazu unsere Datensätze durch.

15.2.0.3. Schritt 3: Mischen

Dieser Schritt greift auf die Funktion `sample()` zurück. Wir erzeugen aus den ursprünglichen Nummerierungen eine neue Nummerierung durch `daten %>% mutate( id_neu = sample(id) )`. Nach dieser neuen Nummerierung sind unsere Datensätze aber immer noch in der gleichen Reihenfolge und noch nicht gemischt. Wir müssen also die Reihenfolge so anpassen, dass die neue Nummerierung gilt. Das erreichen wir mit dem Funktionsaufruf `daten %>% arrange( id_neu )`.

15.2.0.4. Schritt 4: Entfernen des eindeutigen Vektors und exportieren der Daten

Abschliessend müssen wir **unbedingt** die beiden Hilfsvektoren, die wir zum Mischen verwendet haben, aus unserer Stichprobe wieder entfernen. Das erreichen wir mit einer Vektorauswahl: `daten %>% select(-c(id, id_neu))`.

15.2.0.5. Vollständige Lösung

Wir greifen hier auf eine Stichprobe zurück, die Geschlechtsinformationen, Alter und digitale Nutzungsgewohnheiten umfasst. Wir erstellen zwei getrennte Teilstichproben, von denen eine nur die Nutzungsgewohnheiten und das Geschlecht und eine nur die Nutzungsgewohnheiten und das Alter beinhaltet.

Sie müssen die Datei vor dem Einlesen noch in `beispielstichprobe.csv` umbenennen.

```
daten = read_delim("beispielstichprobe.csv")

# mischen Funktion aus einer Funktionskette erstellen, damit
#    wir nicht so viel tippen müssen.
mischen = . %>%
    mutate(
        id = row_number(),
        id_neu = sample(id)
    ) %>%
    arrange(id_neu) %>%
    select(-c(id, id_neu))

daten %>%
    select(geschlecht, starts_with("technik")) %>%
    mischen() %>%
    write_csv("teilstichprobe_geschlecht_technik.csv")

daten %>%
    select(alter, starts_with("sozial")) %>%
    mischen() %>%
    write_csv("teilstichprobe_alter_sozial.csv")
```

Die zwei Teilstichproben lassen sich nicht mehr zusammenführen. Damit erkennen wir auch die Grenzen dieser Technik: Wenn zwei gemischte Stichproben ausreichend viele gemeinsame oder sehr detaillierte Vektoren haben, die in beiden Teilstichproben vorkommen, dann können diese Stichproben trotz mischen wieder zusammengeführt werden.

Mit den gemischten Daten ist es nun nicht mehr möglich, die Werte mit einem anderen Teil der Stichprobe zu kombinieren und so tiefere Rückschlüsse über die Teilnehmenden (womöglich unwissend) zuzulassen. Nur noch durch den Zugriff auf die ursprünglichen Daten können diese Zusammenhänge hergestellt werden. Daher sind die ursprünglichen Daten oft besonders schützenswert und sollten ohne Randomisierung nicht weitergegeben werden.

16. Daten kombinieren und kodieren

⚠ Work in Progress

16.1. Kombinieren

Definition 16.1. Kombinieren von Daten bedeutet die Verknüpfung von Daten aus verschiedenen Quellen zu einer Datenstruktur.

Beim Kombinieren von Daten werden die Daten aus verschiedenen Quellen zu einer gemeinsamen Datenstruktur zusammengeführt. Es gibt verschiedene Arten von Kombinationen, die sich in der Art der Verknüpfung unterscheiden.

16.1.1. Kombinationsarten

- Zeilenweise Konkatenation

Bei der Konkatenation wird davon ausgegangen, dass beide Quellen genau die gleichen Merkmale haben. Die Daten werden dann einfach aneinandergehängt. Die Reihenfolge der Datensätze bleibt dabei erhalten.

- Vereinigung (union/ outer join)

Bei der Vereinigung werden alle Werte aus beiden Quellen über ein gemeinsames Merkmal kombiniert. Das Ergebnis enthält anschliessend Datensätze mit allen Merkmalen aus beiden Quellen. Gibt es für Datensätze in einer Quelle keine Entsprechung in der anderen, werden die fehlenden Werte mit einem speziellen Wert (z.B. *undefinierte Werte*) aufgefüllt.

- spaltenkonkatenation ist eine spezielle Form der Vereinigung

Für eine Spaltenkonkatenation müssen beide Stichproben den gleichen Umfang haben. Meistens fehlen jedoch gemeinsame Merkmale für die Vereinigung. In diesem Fall wird die Vereinigung über einen *gedachten Wert* durchgeführt. Dazu werden alle Datensätze in beiden Quellen *durchnummeriert*. Diese Nummer wird dann als gemeinsames Merkmal verwendet. Anschliessend wird die Nummerierung aus dem Ergebnis entfernt.

- partielle Vereinigung (partial union)

Die partielle Vereinigung kombiniert nur Werte, die in beiden Quelle ein gemeinsames Merkmal teilen. Das Ergebnis enthält anschliessend nur noch Datensätze mit allen Merkmalen aus beider Quellen für die es eine Entsprechung für die gemeinsamen Merkmale gibt.

- Schnittmenge (intersection/inner join)

Die Schnittmenge kombiniert nur Werte, die in beiden Quellen vorkommen. Die Schnittmenge ist also eine Teilmenge der Vereinigung.

- Differenz (difference)

Bei der Differenz werden alle Datensätze mit einer Entsprechung von gemeinsamen Merkmalen in beiden Quellen aus dem Ergebnis entfernt. Das Ergebnis umfasst also nur Datensätze, die in der ersten Quelle vorkommen, aber nicht in der zweiten Quelle.

Die *Differenz* entspricht einen Filter mit einem oder mehreren $\notin$-Vergleichen.

16.2. Kodieren

Definition 16.2. Kodieren von Daten bedeutet die Umwandlung von Daten in ein anderes Format oder einen anderen Wertebereich.

Beim Kodieren wird eine **Kodierungsfunktion** verwendet, die jeden Wert des urspünglichen Wertebereichs einem Wert des gewünschten Wertebereichs zuordnet. Dabei ist es nicht notwendig, dass alle ursprünglichen Werte eindeutig zugewiesen werden. Das heisst, dass mehrere Werte des ursprünglichen Wertebereichs dem gleichen Wert des neuen Wertebereichs zugeordnet werden können.

Sehr häufig werden Kodierungsfunktionen als **Entscheidungsbäume** (s. Definition 12.3) umgesetzt. Dabei werden logische Ausdrücke für die Zuweisung der Ergebniswerte verwendet. Die logischen Ausdrücke werden dabei der Reihe nach geprüft. Die erste zutreffende Entscheidung, bestimmt den Ergebniswert.

16.2.1. Kodierungstabellen

Eine besondere Technik des Kodierens ist die Verwendung von Kodierungstabellen.

Definition 16.3. Eine **Kodierungstabelle** ist eine Tabelle, die jedem Wert eines Wertebereichs einen Wert eines anderen Wertebereichs zuordnet.

Die *Kodierungsfunktion* ist in diesem Fall die Vereinigung oder eine partielle Vereinigung der Stichprobe mit der Kodierungstabelle.

Praxis

Kodierungstabellen sollten immer für die Kodierung von nominal- oder ordinalskalierten Wertebereichen verwendet werden, weil sie die Kodierungsfunktion explizit machen und gleichzeitig die Kodierung *dokumentieren*.

17. Daten umformen

> ⚠ Work in Progress

Viele Operationen der vorangegangenen Kapitel setzen eine vektorisierte Datenstruktur voraus. Echte Daten liegen aber meistens nicht in einem Vektor vor, sondern sind über mehrere Vektoren verteilt. Dadurch erscheinen viele Operationen komplexer als sie eigentlich sind. Durch das Umformen von Daten lassen sich viele Operationen stark vereinfachen und in der Ausführung beschleunigen. Die Grundlage vieler komplexer Algorithmen bilden Primär- und Sekundärindizes (Kapitel 15). Das Ziel des Umformens ist es, Daten in eine Form zu bringen, die die effiziente Verwendung von Indizes und Gruppierungen ermöglicht.

> ℹ Merke
>
> Das Umformen von Daten dient der Vereinfachung und Beschleunigung von Operationen.

Oft werden nur einzelne Vektoren einer Datenstruktur und nicht die gesamte Datenstruktur umgeformt.

> ❗ Wichtig
>
> Alle Umformungen müssen umkehrbar sein, so dass für eine Datenstruktur D die Gleichung 17.1 gilt.

$$f_u(D) \triangleright f_u^{-1}() = f_{id}(D) = D \tag{17.1}$$

Lässt sich durch wiederholtes Umformen die ursprüngliche Datenstruktur nicht erzeugen, dann liegt eine **partielle Umformung** vor.

> ℹ Merke
>
> Jede *partielle Umformung* führt zu Informationsverlusst durch *Equivokation*.

17.1. Transponieren

Definition 17.1. Transponieren ist eine Operation, die eine Datenstruktur um 90-Grad dreht.

Es gibt zwei Arten des Transponierens.

1. Das Transponieren von Matrizen, bei dem die Zeilen und Spalten vertauscht werden. Dabei bleibt die grundsätzliche Struktur der Daten erhalten.
2. Das Transponieren von Daten, bei mehrere Vektoren zu einem Vektor zusammengefasst werden oder ein Vektor in mehrere Vektoren aufgegliedert wird. Weil sich die Struktur der Daten verändert, wird diese Operation auch als **Reshaping** oder **Pivoting** bezeichnet.

Kapitel 14 zeigt das Transponieren von Matrizen. Diese Operation wird nur verwendet, wenn eine Matrix mit einer zweiten Matrix multipliziert werden soll und die Orientierung der Matrix nicht für das Kreuzprodukt geeignet ist.

Das Transponieren von Daten entspricht dem Überführen einer Matrix in eine ihrer **Vektorformen** (s. Abschnitt 8.4). Das Transponieren ist jedoch nicht auf Matrizen beschränkt, sondern ist für Vektoren mit beliebeigen Datentyp definiert. Hat eine Datenstruktur aus geschachtelten Vektoren für alle Werte den gleichen Datentyp und haben alle Vektoren die gleiche Länge, dann wird die Datenstruktur **Matrixform** genannt und kann transponiert werden.

> **Merke**
>
> Vektoren können nur transponiert werden, wenn alle Vektoren vom gleichen Datentyp sind.

Beim Transponieren wird zwischen der **Vektorform** bzw. *Langform* und der **Matrixform** bzw. *Breitform* unterschieden. Ein Primärindex der *Matrixform* wird zum Sekundärindex der *Vektorform* und umgekehrt. Weil aus der *Vektorform* nicht eindeutig hervorgeht wie die *Matrixform* aufgebaut ist, benötigt das Transponieren einen *zusätzlichen* Sekundärindex, der die Struktur der *Matrixform* beschreibt.

Die *Vektorform* muss nicht zwingend die gleiche Anzahl Werte haben, wie die *Matrixform*. Die *Vektorform* kann auch *weniger Werte* als die *Matrixform* haben.

Definition 17.2. Enthält eine Vektorform weniger Werte als ihre Matrixform, dann wird sie als **kurze Vektorform** bezeichnet.

Beim Transponieren einer *Vektorform* mit weniger Werten als die *Matrixform*, werden die fehlenden Werte aufgefüllt. Dadurch werden neue Werte erzeugt, die nicht in den ursprünglichen Daten enthalten waren.

> **Merke**
>
> Das Transponieren einer kurzen Vektorform in ihre Matrixform erzeugt neue Werte und führt zu Informationsverlust durch *Rauschen*.

Dieses Rauschen ist jedoch kontrollierbar, wenn diese Eigenschaft bei der Umformung berücksichtigt wird.

17.2. Hierarchisieren

Nicht immer sind alle umzuformenden Vektoren vom gleichen Datentyp. In diesem Fall können die Werte nicht transponiert werden. Stattdessen müssen die Vektoren in eine hierarchische Datenstruktur überführt werden.

Definition 17.3. Das Hierarchisieren ist eine Operation, bei der Daten in eine hierarchische Datenstruktur überführt werden.

Die wichtigste Operation beim Hierarchisieren ist das **Einbetten** von Daten. Das Einbetten von Daten ist eine wichtige Operation, um Daten zu strukturieren und zu organisieren. Es bildet die Grundlage für **komplexe Datenstrukturen** mit baumartigen Datenstrukturen (s. Kapitel 9.5). Diese Umformung ist auch als **nesting** bekannt.

> **Merke**
>
> Beim Einbetten können die umgeformten Vektoren unterschiedliche Datentypen haben.

Beim Einbetten werden die umgeformten Vektoren entlang eines Sekundärindex gruppiert. Anschliessend werden die gruppierten Vektoren in Teilstichproben getrennt und aus der ursprünglichen Datenstruktur entfernt. Die Teilstichproben enthalten nur die umzuformenden Vektoren. Weil die resultierende Datenstrukturen für alle Gruppen gleich sind, können die Teilstichproben zu einem Vektor zusammengefasst werden.

Weil alle Einträge des Sekundärindex in den neuen Vektor mit den Teilstichproben verlagert wurde, kann der Sekundärindex so reduziert werden, dass jeder *Hash* genau einmal vorkommt. Dadurch hat der Vektor des Sekundärindex die gleiche Länge wie der Vektor mit den Teilstichproben. Die beiden Vektoren können zu einem Datenrahmen zusammengefasst werden, so dass jeder Indexwert mit der zugehörigen Teilstichprobe einen Datensatz bildet.

Die resultierende Datenstruktur ist eine hierarchische Datenstruktur, die aus einem Vektor von Datenrahmen besteht. In der neuen, eingebetteten Struktur kommen die Indexwerte nur noch einmal vor. Daraus folgt, dass der Sekundärindex in einen Primärindex überführt wurde.

Teil IV.

Deskriptive Datenanalyse

18. Daten beschreiben

18.1. Universelle Kennzahlen

> ⚠ Work in Progress

Wir haben das Zählen mit R und Excel kennengelernt. Diese Operation wenden wir an, um die zwei universellen Kennzahlen von Stichproben zu bestimmen.

Definition 18.1. Mit **universellen Kennzahlen** sind abgeleitete Werte gemeint, die wir für *alle* Stichproben bestimmen können.

Die beiden universellen Kennzahlen für Stichproben sind:

- Der *Stichprobenumfang*, d.h. die Anzahl der Datensätze
- Die *Anzahl der Variablen*

Wir erkennen sofort, dass es sich bei der Bestimmung der beiden Kennwerte um ein **(Ab-) Zählproblem** handelt.

> ℹ Merke
>
> In **jeder** wissenschaftlichen Arbeit, in der Sie Daten präsentieren, **müssen** Sie **immer** die beiden Kennwerte angeben.

Für die Anzahl der Variablen könnten wir im Prinzip auf die Dokumentation einer Stichprobe zurückgreifen. Wir bestimmen trotzdem *beide* Kennzahlen, um sicherzustellen, dass wir wirklich alle Variablen in einer Stichprobe berichten und nicht versehentlich Variablen unterschlagen, die in der Dokumentation nur implizit festgehalten werden.

> 💡 Konvention
>
> Stichprobenumfänge werden in wissenschaftlichen Arbeiten mit dem Buchstaben `n` gekennzeichnet.

Konvention

In **Fragebogenstudien** wird die Anzahl der Vektoren als Anzahl der **Items** bezeichnet. Damit sind die unabhängig festgehaltenen Antwortmöglichkeiten gemeint.

Konvention

In **technischen Studien** wird die Anzahl der Vektoren als Anzahl der *Parameter*, *Vektoren* **oder** *Variablen* bezeichnet. Damit sind die unabhängigen, gemessenen Merkmale gemeint.

Bei beiden Kennzahlen müssen wir eine Anzahl bestimmen. Beim **Stichprobenumfang** bestimmen wir die Anzahl der Datensätze. Für die **Anzahl der Variablen** müssen wir die Vektoren zählen.

Merke

Stichprobenobjekte sind immer **rechteckig**. Alle Variablen sind immer in allen Datensätzen vorhanden (horizontaler Umfang) und alle Datensätze haben für jede Variable einen Wert (vertikaler Umfang). Wir erhalten die Kennwerte, indem wir horizontal und vertikal zählen.

Hinweis

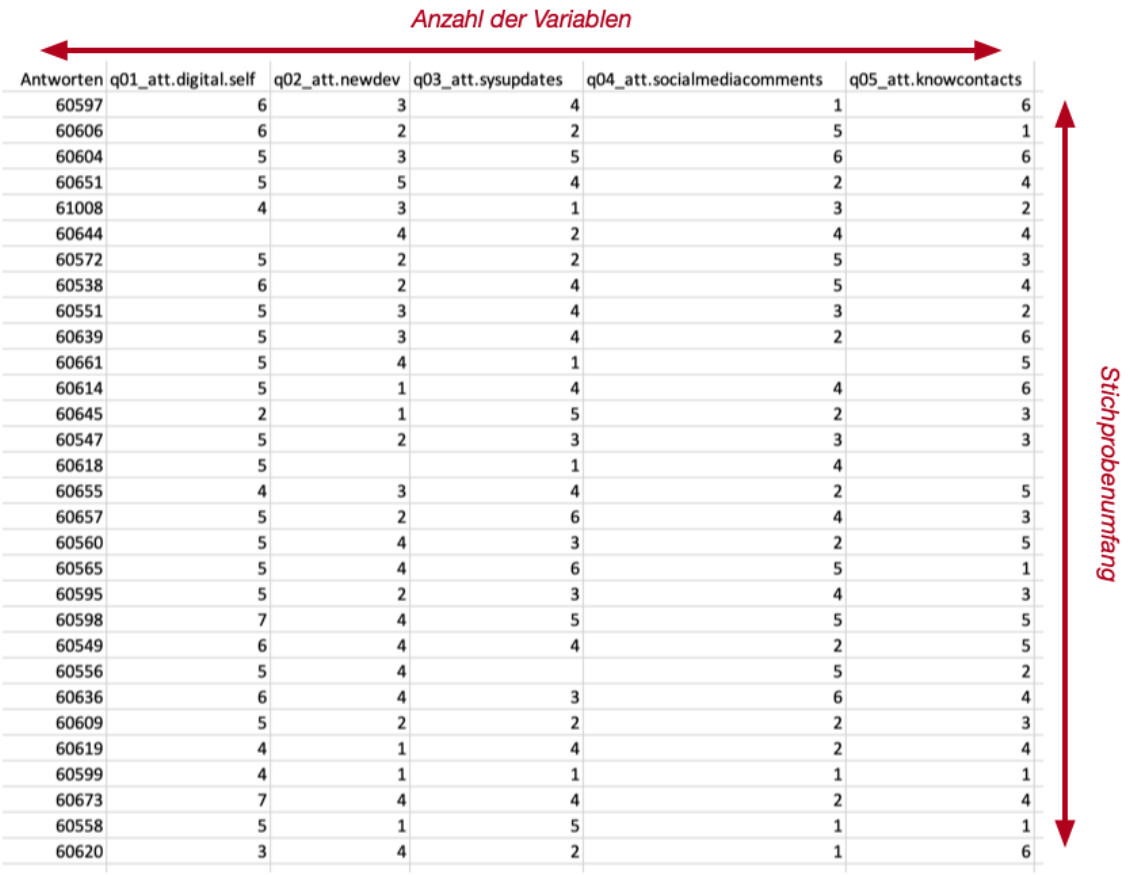
Die Aufgabe der Deskriptivstatistik ist es, Daten prägnant zusammenzufassen.
Sauer (2019), S. 103

Die deskriptive Statistik wird oft mit wenigen Sätzen in der Literatur abgehandelt und auf die folgenden Punkte beschränkt.

1. Umfang
2. Skalenniveau
3. Lagemasse

Das Zitat von Bortz & Schuster deutet bereits an, dass jeder *schliessenden Statistik* eine Stichprobenbeschrreibung mit Hilfe beschreibender Kennwerte vorangehen **muss**. Dabei müssen wir für die Beschreibung angemessene Lagemasse für das jeweilige Skalenniveau einer Variable wählen. Hier müssen wir etwas aufpassen und zwischen drei Arten von Vektoren in unseren Stichprobendaten unterscheiden:

1. Konstanten
2. Indikatoren
3. Variablen



Antworten	q01_att.digital.self	q02_att.newdev	q03_att.sysupdates	q04_att.socialmediacomments	q05_att.knowcontacts
60597	6	3	4	1	6
60606	6	2	2	5	1
60604	5	3	5	6	6
60651	5	5	4	2	4
61008	4	3	1	3	2
60644		4	2	4	4
60572	5	2	2	5	3
60538	6	2	4	5	4
60551	5	3	4	3	2
60639	5	3	4	2	6
60661	5	4	1		5
60614	5	1	4	4	6
60645	2	1	5	2	3
60547	5	2	3	3	3
60618	5		1	4	
60655	4	3	4	2	5
60657	5	2	6	4	3
60560	5	4	3	2	5
60565	5	4	6	5	1
60595	5	2	3	4	3
60598	7	4	5	5	5
60549	6	4	4	2	5
60556	5	4		5	2
60636	6	4	3	6	4
60609	5	2	2	2	3
60619	4	1	4	2	4
60599	4	1	1	1	1
60673	7	4	4	2	4
60558	5	1	5	1	1
60620	3	4	2	1	6

Abbildung 18.1.: Universelle Stichprobenkennwerte

> **Hinweis**
>
> Nur die **Variablenvektoren** können unsere Stichprobe beschreiben, weil Sie *gemessene Merkmalsausprägungen* unser Entitäten enthalten.

Im folgenden arbeiten wir mit einer Beispielstichprobe. Diese Stichprobe ist ein Ausschnitt aus einer Studie zur Relevanz von digitalen Kompetenzen bei Dozierenden während des ersten COVID-Lockdowns im FS2020.

18.2. Umfang

Mit dem Umfang einer Stichprobe meinen wir die *Anzahl der Entitäten*, die wir gemessen haben. In R ergibt sich dieser Wert direkt aus dem Stichprobenobjekt: Der **Stichprobenumfang** entspricht der **Anzahl der Datensätze** in unserem Stichprobenobjekt. Diese Anzahl bestimmen wir mit Hilfe der `count()`-Funktion oder innerrhalb einer Transforrmation mit `mutate()` mit Hilfe der `n()`-Funktion.

Der *Stichprobenumfang* ist einer der drei allgemeinen Kennwerte, die jede Stichprobe beschreiben. Wir bestimmen zuerst die numerischen Kennwerte unserer Beispielstichprobe.

```
# Stichprobenumfang
stichprobe %>%
    count() %>%
    pull() -> stichprobenumfang

stichprobenumfang

stichprobe %>%
    names() %>%
    length()
```

Dabei erkennen wir, dass der Stichprobenumfang 27 beträgt. Wir erkennen zusätzlich, dass wir 53 Vektoren in unserem Stichprobenobjekt vorliegen haben.

Neben dem *Stichprobenumfang* unterscheiden wir zusätzlich den *Variablenumfang*. Der **Variablenumfang** bezeichnet die **Anzahl der gemessenen Merkmalsausprägungen**. Damit ist die Gesamtzahl der gemessenen Werte für eine Variable gemeint. Für diesen Wert müssen wir für eine Variable *nicht vorhandene* Werte aus der Variable entfernen. Diese Werte sind in R mit dem Wert `NA` gekennzeichnet. Wir entfernen die `NA`-Werte mit Hilfe der `drop_na()`-Funktion.

Es ist normal, dass sich der Stichprobenumfang und die Variablenumfänge unterscheiden. Diese Unterschiede entstehen dadurch, dass nicht alle Messungen erfolgreich verlaufen. Zum Beispiel passiert es häufig, dass Teilnehmende bei einem Fragebogen nicht alle Fragen beantworten oder beantworten können. In solchen Fällen ist der Stichprobenumfang grösser

als die jeweiligen Variablenumfänge. So kommt es regelmässig vor, dass verschiedene Variablenumfänge sich ebenfalls unterscheiden.

Die deskriptive Statistik muss daher **immer** den Stichprobenumfang **und** die Variablenumfänge anführen.

18.3. Skalenniveaus und Lagemasse

Für Variablen unterscheiden wir nur die folgenden Skalenniveaus:

1. Nominal
2. Ordinal
3. Metrisch (Intervall- oder Varianzskaliert)

Die Skalenniveaus unserer Variablen müssen wir den Vektoren zuordnen können. Dazu müssen wir sowohl die Variablen und Vektoren unserer Stichprobe kennen. Diese Information erschliesst sich nicht aus den Daten, sondern ist in der Regel separat dokumentiert.

Unsere Beispielstichprobe hat 53 Vektoren. Diese Vektoren sind vier Variablengruppen zugeordnet. Eine Variablengruppe sind unterschiedliche Merkmalsausprägungen mit dem gleichen Skalennivau, die wir in Beziehung setzen. Das Beispiel umfasst die folgenden Variablengruppen und Skalenniveaus:

1. `q1` ist ein Indikator
2. `q3`, `q4` und `q16` sind nominalskalierte Variablen

- `q3` hat die Skalierung von `F`, `M`, `X` und `O`.
- `q4` hat die Skalierung von `A`, `G`, `L`, `N`, `P`, `S`, `T` und `W`.
- `q16` hat die Skalierung von `0` und `1`.

3. `q10` sind ordinalskalierte Variablen mit der Skalierung von `0-10` in ganzzahligen Schritten. Die Messungen stammen von sog. Likert-Skalen.

> Übung
>
> Bestimmen Sie alle Vektorennamen in der Beispielstichprobe.

Für die Variable `q10` es 6 Untermerkmale und für die Variable `q16` gibt es 19 Untermerkmale. Für alle Merkmale in `q10` und `q16` wurden jeweils zwei Aspekte (`0` und `1`) erhoben. Diese Aspekte können für das jeweilige Untermerkmal direkt verglichen werden. Wir haben also im Vektor `q16_13_1` den zweiten Aspekt des 13. Untermerkmals der Variable q16 erfasst.

18.3.1. Lagemasse für nominale Skalenniveaus

Für nominale Variablen dürfen wir als Lagemass für die Verteilung nur den *Modus* bestimmen. Damit ist die am häufigsten auftretende Merkmalsausprägung gemeint.

Das zweite "Lagemass" sind die tatsächlich auftretenden Skalenwerte.

Beide Kennwerte erschliessen sich über die *Häufigkeiten* der vorkommenden Werte. Dazu bestimmen wir die Häufigkeit des Auftretens einer Merkmalsausprägung in unserer Stichprobe. Das bestimmen der Häufigkeiten ist eine Abzählaufgabe, bei der wir vorher die gleichen Merkmalsausprägungen gruppiert haben.

Das folgende Beispiel zwei Varianten für das Feststellen der Häufigkeiten von nominalskalierten Variablen. Die erste Variante ist die direkte Umsetzung des gerade beschriebenen Algorithmus.

```
stichprobe %>%
    group_by(q3) %>%
    summarise(n = n())
```

q3

n

<chr>

<int>

F

10

M

17

Die zweite Variante verwendet die `count()`-Funktion, um das gleiche Ziel zu erreichen. Die Variante mit `count()` ist etwas leichter zu lesen, weil wir *explizit* klarstellen, dass wir die Merkmalsausprägungen *abzählen*.

```
stichprobe %>%
    count(q4)
```

q4

n

<chr>

<int>

N

17

P

10

Wir können an diesem Beispiel erkennen, dass in unserer Stichprobe nicht alle Ausprägungen der jeweiligen Skalierung in unseren Variablen vorkommen. Uns fällt auch auf, dass die Häufigkeiten für die gemessenen Merkmalsausprägungen der Variablen gleich sind.

Neben den absoluten Häufigkeiten interessieren uns ebenfalls die *relativen Häufigkeiten*. Diese **müssen** bei der Beschreibung von nominalskalierten Variablen mit angegeben werden.

Relative Häufigkeiten sind nichts anderes als die prozentualen Anteile der absouluten Häufigkeit in der Gesamtstichprobe. Wir können die relativen Häufigkeiten leicht aus den absoluten Häufigkeiten bestimmen.

```
stichprobe %>%
    count(q4) %>%
    mutate(
        prozent = n / sum(n)
    )
```

Damit erhalten wir die prozentualen Anteile für alle beobachteten Merkmalsausprägungen.

> ⚠ Achtung
>
> Beachten Sie hier, dass `n` den Vektor `n` bezeichnet, der durch den Aufruf der `count()`-Funktion erzeugt wird. Sie dürfen diesen Vektor nicht mit der Funktion `n()` verwechseln! Die `n()`-Funktion liefert die Anzahl der Datensätze einer Stichprobe. Nachdem wir `count()` aufgerufen haben, ist das Ergebnis von `n()` in diesem Beispiel gleich 2.

Wir können nun leicht den **Modus** der Variable mit Hilfe der `max`-Aggregation bestimmen. Anstelle der `summerise()`-Funktion verwenden wir jedoch besser die `filter()`-Funktion, um auch das zugehörige Skalenniveau zu erhalten.

```
# Bestimmen des Modus von q3
stichprobe %>%
    count(q3) %>%
    filter(n == max(n))
```

q3

n

<chr>

<int>

M

17

> **Tipp**
>
> Der **Modus** ist in der deskriptiven Statistik nur in Ausnahmefällen von Bedeutung. Eine solche Ausnahme liegt vor, wenn aus mehreren Merkmalsausprägungen der Modus den stark überwiegenden Anteil in der Stichprobe ausmacht. Dieser Anteil ist ungefähr ab einer relativen Häufigkeit von `.75` oder grösser gegeben. In solchen Fällen sollten Sie trotzdem die Häufigkeiten der anderen Merkmalsausprägungen berichten.

18.3.2. Lagemasse für metrische Skalenniveaus

Die grundlegenden Kennwerte für metrische Skalenniveaus sind die *Bandbreite*, die sich aus dem kleinsten (*Minimum*) und grössten gemessenen Wert (*Maximum*) ergibt.

Für metrischskalierte Variablen können wir immer den Median und Quantile bestimmen, wobei wir uns in der Regel auf die *Quartile* beschränken. Bei der Beschreibung der Quartile geben wir immer den *Interquartilsabstand* mit an. Dieser ist das Gegenstück zur Bandbreite und zeigt uns zwischen welchen Ausprägungen die Hälfte der gemessenen Variablenwerte liegen.

Das folgende Beispiel zeigt die Berechnung der relevanten Kennwerte für die Variable `q10_1_0`. Diese Berechnung entspricht der `favstats` der in Sauer (2019) beschriebenen `mosaic`-Bibliothek

```
stichprobe %>%
    select(q10_1_0) %>%
    drop_na() %>%
    summarise(
        n = n(),
        min = min(q10_1_0),
        max = max(q10_1_0),
        bw = max - min,

        iqr = IQR(q10_1_0), # berücksichtigt ausreisser!

        q1 = quantile(q10_1_0, .25),
        # q2 = quantile(q10_1_0, .5), # der Median ist die zweite Quartilsgrenze
        md = median(q10_1_0),
        q3 = quantile(q10_1_0, .75),
        # q4 = quantile(q10_1_0, 1) , # Das Maximum ist die vierte Quartilsgrenze.

        mad = mad(q10_1_0, constant = 1)
    )
```

> **Merke**
>
> Diese Ergebnisse werden nicht interpretiert!

> **Merke**
>
> Es handelt sich bei diesen Ergebnisse um sog. **á priori Fakten**. Es sind Fakten, die sich direkt aus den Messungen und der Definition der Lagemasse ergeben und keiner weiteren Begründung bedürfen.

18.3.3. Lagemasse für ordinale Skalenniveaus

Im Unterschied zu den metrischen Skalenniveaus ist der Mittelwert und die Standardabweichung für *ordianle Skalenniveaus* **nicht definiert**. Ausserdem werden Quartilsgrenzen und die davon abhängigen Variablen *etwas* anders berechnet als bei metrischskalierten Variablen. Damit wir keine Folgefehler für die spätere Analyse erhalten müssen wir für ordinalskalierte Variablen die entsprechenden Kennwerte richtig berechnen.

> **Achtung**
>
> In vielen Fällen ergeben die verschiedenen Berechnungsarten die *gleichen Ergebnisse*. Das bedeutet nicht, dass die Ergebnisse in *allen* Fällen gleich und damit austauschbar sind!

Die Berechnungsart wird mit dem Parameter `type` festgelegt. Dieser Parameter können wir die Ganzzahlen von `1` bis `8` übergeben. Der Standardwert für diesen Parameter ist `7`. Die Werte `1`, `2` und `3` sind für die Berechnung bei ordinalen Skalenniveaus gedacht. Die anderen Werte beschreiben verschiedene Varianten für metrische Skalenniveaus.

> **Hinweis**
>
> Verwenden Sie nur die Werte `1` für ordinale Skalenniveaus und `7` für metrische Skalenniveaus. Weil `7` der Vorgabewert ist, wird dieser bei metrischen Skalenniveaus einfach weggelassen.

Alle anderen Werte für den `type`-Parameter sind für besondere Fälle, wenn Ergebnisse aus anderen Quellen kontrolliert werden müssen.

Das folgende Codesegment zeigt Berechnung der statistischen Kennwerte für ordinale Skalenniveaus.

```
stichprobe %>%
    select(q10_1_0) %>%
```

```
    drop_na() %>%
    summarise(
        n = n(),
        min = min(q10_1_0),
        max = max(q10_1_0),
        bw = max - min,

        iqr = IQR(q10_1_0, type = 1), # berücksichtigt ausreisser!

        q1 = quantile(q10_1_0, .25, type = 1),
        # q2 = quantile(q10_1_0, .5, type = 1), # der Median ist die zweite Quartilsgre
        md = median(q10_1_0, type = 1),
        q3 = quantile(q10_1_0, .75, type = 1),
        # q4 = quantile(q10_1_0, 1, type = 1) , # Das Maximum ist die vierte Quartilsgr

       mad = mad(q10_1_0, constant = 1)
    )
```

19. Daten visualisieren

Die Datenvisualisierung ist ein wichtiges Werkzeug für die Datenanalyse. Die Datenvisualisierung hilft Systematiken zu erkennen, Beziehungen zu verstehen und Strukturen zu erfassen, die in strukturierten Daten nicht direkt nachvollzogen werden können. Sie ist ein wesentliches Instrument für die Kommunikation von Daten, weil sich graphische Darstellungen leichter erfassen lassen als strukturierte Daten.

> **Merke**
>
> Die Datenvisualisierung ist ein Hilfsmittel zur Analyse und zur Kommunikation. Sie ist kein Selbstzweck und stellt *keine* Analyse dar. Die Datenvisualisierung kann helfen, Daten vor einer weiterführenden Analyse zu verstehen.

Die Datenvisualisierung kann Informationstheoretisch betrachtet werden. Dabei werden die Werte visuell *kodiert*. Dieser Vorgang folgt den gleichen informationstheoretischen Prinzipien wie die Datenverarbeitung. Entsprechend muss die Datenvisualisierung als eine Form der Informationskodierung verstanden werden. Aus dieser Einsicht folgt konsequent, dass Information bei der Visualisierung zwangsläufig durch Rauschen und Equivokation verloren geht. Gleichzeitig müssen Visualisierungen nicht nur erstellt sondern auch gelesen und systematisch interpretiert werden. Bei wissenschaftlichen Visualisierungen werden spezielle visuelle Codes verwendet, um Werte und ihre Beziehungen zueinander sichtbar zu machen.

> **Künstlerische Freiheit**
>
> Bei der Datenvisualisierung sollten ästhetische Elemente ohne Bezug zu den Daten **vermieden** werden, wenn sie das Dekodieren behindern.

19.1. Aufbau eines Diagramms

Eine Visualisierung ist in ein *Diagramm* eingebettet. Ein Diagramm enthält meist mehr Elemente als die eigentliche Visualisierung. Abbildung 19.1 zeigt die wichtigsten Elemente eines Diagramms.

Jedes Diagramm umfasst den **Darstellungsbereich** mit den **Datenpunkten** (s. Definition 6.5), **Achsen** für die *Hauptdimensionen* mit **Achsmarkierungen** für die Position ausgewählter Werte im Wertebereich der Dimension und **Achsbeschriftungen**, die die *Hauptdimensionen* kennzeichnen.

Komplexe Diagramme erfordern zusätzlich eine **Legende** für **jede** dargestellte Nebendimension.

Optional können Visualisierungen mit einer erklärenden **Überschrift** versehen werden. Ebenfalls *optional* sind **Hilfslinien**, die die *Orientierung* im Wertebereich der *Hauptdimensionen* erleichtern.

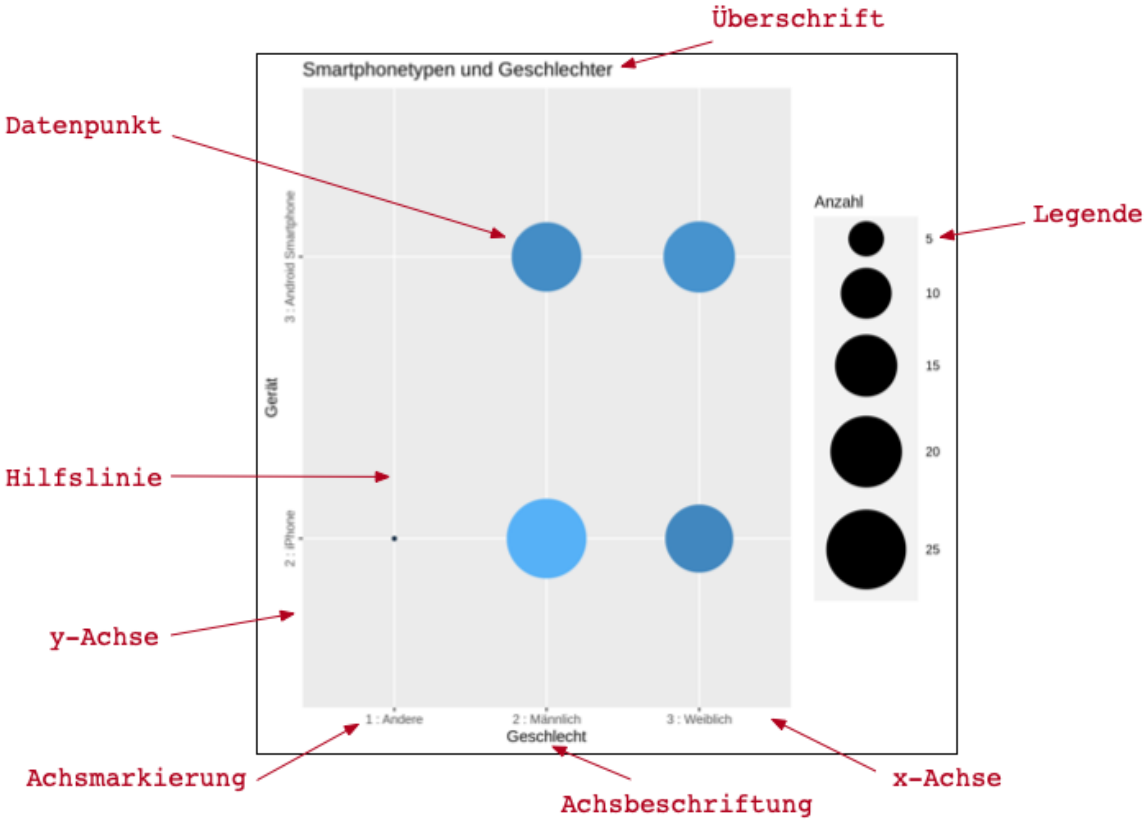


Abbildung 19.1.: Komponenten eines Diagramms am Beispiel eines Ballondiagramms

19.1.1. Haupt- und Nebendimensionen

Für eine Visualisierung müssen immer zuerst die darzustellenden Merkmale ausgewählt werden.

Definition 19.1. Eine **Dimension** eines Diagramms entspricht einem dargestellten *Merkmal* der verwendeten Stichprobe.

> ⚠ Warnung
>
> Die Merkmale aus unterschiedlichen Stichproben dürfen beim Visualisieren nicht gemischt werden!

Beim Visualisieren werden *Haupt-* und *Nebendimensionen* unterschieden.

Definition 19.2. Eine **Hauptdimension** eines Diagramms bildet den Wertebereich eines Merkmals als eine *horizontale oder vertikale Länge* ab.

Die Hauptdimensionen legen die **Position** der *Datenpunkte* in der Visualisierung fest.

Jede Visualisierung hat *mindestens eine* Hauptdimension.

Definition 19.3. Nebendimensionen eines Diagramms bilden den Wertebereich eines Merkmals als *visuelle Eigenschaften* von Datenpunkten ab.

Visuelle Eigenschaften können *Grösse*, *Form* oder *Farbe* von Datenpunkte sein.

Nebendimensionen werden nicht in allen Diagrammen verwendet.

19.1.2. Darstellungsbereich

Der Darstellungsbereich enthält die eigentliche Visualisierung der *Datenpunkte*. Dieser Bereich muss für eine korrekte Darstellung *alle gemessenen Werte*, aber nicht zwingend den gesamten Wertebereich umfassen. Deshalb ist es nicht erlaubt, das Achsintervall des Darstellungsbereich so zu verändern, dass einzelne Werte nicht mehr dargestellt werden.

Der Darstellungsbereich wird durch die *Hauptdimensionen* der Visualisierung aufgespannt.

19.1.3. Achsen

Die **Achsen** markieren die *Hauptdimensionen* eines Diagramms. Es sollten nicht mehr als zwei Hauptdimensionen in einem Diagramm verwendet werden, weil sich aus der Darstellung nicht mehr erkennen lässt, welche Achse zu welchen visuellen Elementen grhört.

Die horizontale Achse wird per Konvention **x-Achse** und die vertikale Achse **y-Achse** genannt.

Achsen sollten mit den wichtigsten Werten an **Achsmarkierungen** beschriftet werden und müssen eine aussagekräftige **Achsbeschriftung** haben. Die meisten Analyseumgebungen übernehmen standardmässig den Titel des Datenvektors.

Für **nominalskalierte Wertebereiche** müssen **alle** dargestellten Werte über eine Achsmarkierung beschriftet werden. Bei sehr vielen Werten kann es vorkommen, dass diese Markierungen unleserlich werden. In diesem Fall sollten die Werte alphabetisch oder nummerisch sortiert werden und nur markannte Werte (z.B. jeden fünften Wert) hervorgehoben werden. In diesem Fall muss auf den vollständigen Wertebereich verwiesen werden, damit die Datenpunkte eindeutig identifiziert werden können.

Bei **ordinalskalierten Wertebereichen** müssen die **Extreme des Wertebereichs** beschriftet werden. Gegebenenfalls sollte auch der *Mittelpunkt* des Wertebereichs auf der Achse beschriftet werden.

Bei **kontinuierlichen Wertebereichen** sollten Werte in **regelmässigen Abständen** markiert und beschriftet werden. Diese genauen Abstände hängen vom Umfang des dargestellten Intervalls ab.

19.1.4. Titel

Der Diagrammtitel ist ein aussagekräftiger Titel über den Inhalt eines Diagramms.

> Praxis
>
> Der Diagrammtitel wird oft nicht in ein Diagramm aufgenommen, sondern erst später als Abbildungstitel hinzugefügt.

19.1.5. Legende

Eine *Legende* beschreibt die *Wertebereiche* der *Nebendimensionen* einer Visualisierung. Ohne eine Legende lassen sich zusätzliche Kodierungen nicht nachvollziehen und dekodieren. Ein Diagramm muss eine Legende haben, sobald Nebendimensionen abgebildet sind. Fehlt eine Legende sind zusätzliche grafische Elemente als künstlerische Ergänzung ohne inhaltlichen Wert zu interpretieren.

19.2. Plots mit einer Variablen

19.2.1. Histogramme

Definition 19.4. Ein Histogramm zeigt die Häufigkeiten einer Variablen mithilfe von Balken an. Umgangssprachlich werden Histogramme auch als *Balkendiagramme* bezeichnet.

Die Höhe der Balken entspricht der Häufigkeit der Werte. Ein Histogramm macht die Verteilung einer Variable sichtbar.

- Bei *kontinuierlich-skalierten Variablen* werden die Werte zu **Intervallen** zusammengefasst. Die Häufigkeit bezieht sich dann auf Anzahl der Werte im entsprechenden Intervall. Die Breite des Balkens entspricht dann der Breite des Intervalls.

- Bei *diskret-skalierten Variablen* wird jeder Wert durch einen eigenen Balken repräsentiert. Die Häufigkeit der Werte bezieht sich dann auf die einzelnen Werte. Die Breite der Balken ist dann beliebig.

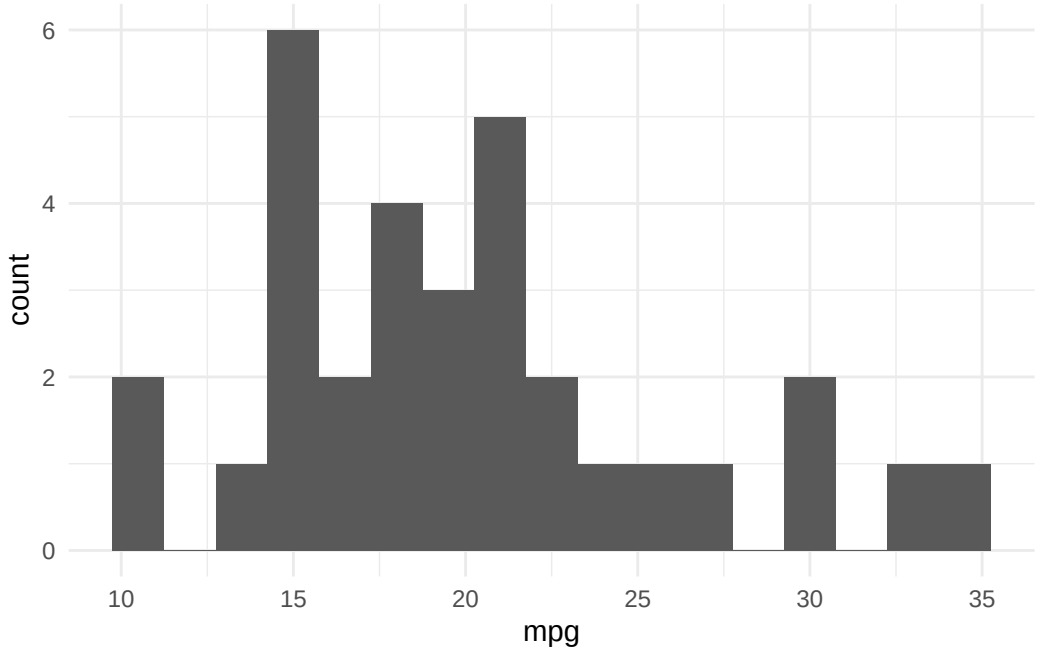


Abbildung 19.2.: Beispiel für ein Histogramm für einen kontinuierlichen Wertebereich

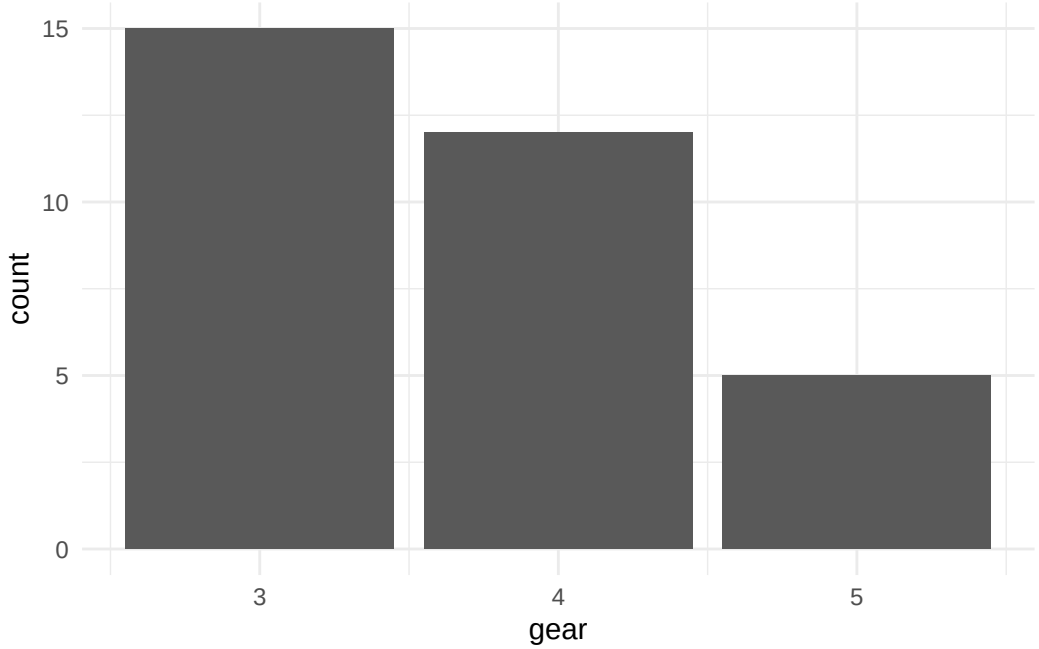


Abbildung 19.3.: Beispiel für ein Histogramm für einen diskreten Wertebereich

Die Reihenfolge der Balken ist nicht beliebig. Die Balken werden für alle ordinalskalierten Wertebereiche in der Reihenfolge der Werte bzw. der Intervalle angeordnet. Nominalskalierte Wertebereiche haben keine natürliche Reihenfolgen. Deshalb werden die Balken eines Histogramms für nominalskalierte Wertebereiche in der Reihenfolge der Häufigkeiten angeordnet.

Eine Sonderform eines Histogramms ist das sog. *Kreis-* bzw. *Tortendiagramm*. Ein Tortendiagramm ist ein Histogramm, bei dem die Balken durch Kreissegmente ersetzt werden. Die Grösse der Kreissegmente entspricht dem Anteil der Häufigkeit an der Gesamthäufigkeit.

> **i** Merke
>
> Ein Histogramm ist einem Kreisdiagramm immer vorzuziehen. Kreisdiagramme sollten nur verwendet werden, wenn wenige Kreissegmente vorhanden sind. Dabei sollten entweder ein Kreissegment besonders hervorstechen oder alle Segmente ungefährt *gleichverteilt* sein.

19.2.2. Boxplot

Definition 19.5. Ein **Boxplot** stellt die Grösse der Verteilungsintervalle von Daten als Rechteck und Linien dar. Aussergewöhneliche Werte (Ausreisser) werden als Punkte dargestellt.

Box-Whisker-Diagramm und **Kastengrafik** sind gebräuchliche Synonyme für Boxplots.

> **i** Merke
>
> Boxplots dürfen nur für ordinal- oder kontinuierlichskalierte Wertebereiche verwendet werden.

Ein Boxplot hat nur eine Achse für den Wertebereich der dargestellten Werte. Die zweite Achse in einem einfachen Boxplot ist nur notwendig, um das Rechteck zeichnen zu können und hat sonst keine Bedeutung.

Ein Boxplot gliedert die vorliegenden Daten in vier gleichgrosse Bereiche. Diese Bereiche heisse Quartile. Jedes Quartil umfasst jeweils ein Viertel der Werte. Die beiden mittleren Quartile werden als zwei Rechtecke dargestellt. Die Länge der beiden Rechtecke heisst **Interquartilsabstand**. Die Linie die am Übergang zwischen den beiden Rechtecken entsteht markiert den *Median* der Verteilung. Ausreisser sind Werte, die um mehr als das 1.5-fache des Interquartilsabstands von der nächsten Quartilsgrenze entfernt sind.

Weil ein Boxplot die Werte in gleichgrosse Intervalle gliedert, geben sie eine grobe Orientierung über die Verteilung der Werte.

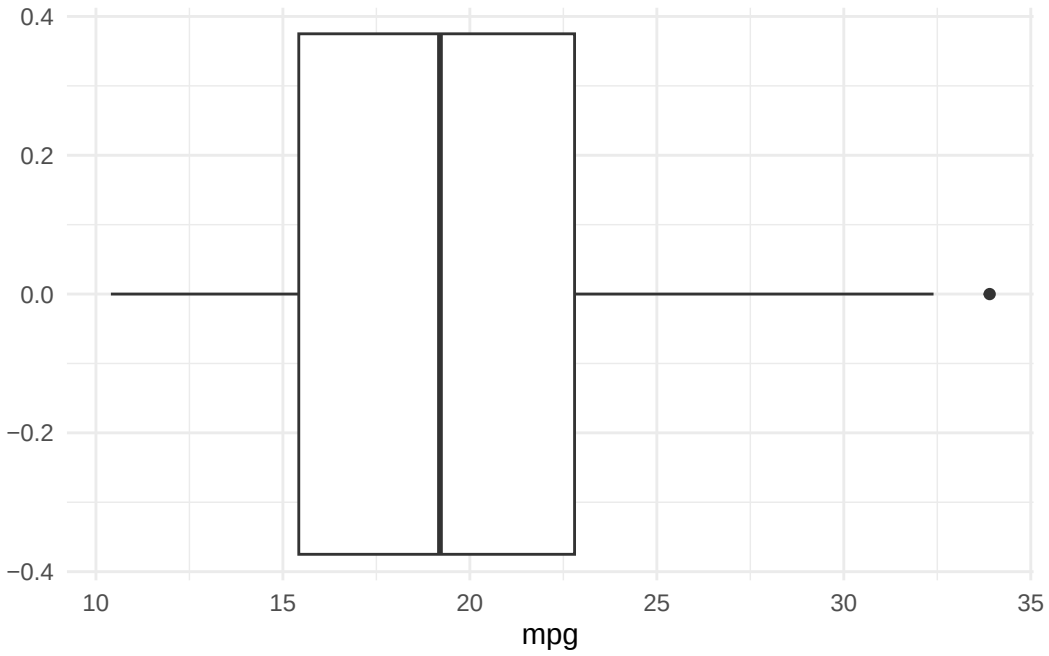


Abbildung 19.4.: Beispiel für einen Boxplot

19.2.3. Dichtediagramme

Boxplots und Histogramme fassen die Werte immer zu Intervallen zusammen. Dadurch lässt sich die genaue Verteilung der Daten nicht genau nachvollziehen. Bei **kontinuierlichen Wertebereichen** zeichen die Intervalle nicht immer ein akkurates Bild der Verteilung. Eine detailliertere Annäherung als ein Histogramm liefert die *Dichtefunktion* einer Verteilung.

Die *Dichtefunktion* liefert die Wahrscheinlichkeiten, mit der Werte in einer Verteilung auftreten, als eine stetige Funktion von Wahrscheinlichkeitswerten. Weil nicht alle Werte in den Daten vorkommen, ist die Dichtefunktion eine ***Annäherung*** an die wahrscheinlichste Verteilung der dargestellten Werte.

Definition 19.6. Dichtediagramme sind eine Visualisierung der *Dichtefunktion* einer Verteilung und zeigen die Wahrscheinlichkeiten mit denen Werte eines *kontinuierlichen* Wertebereichs in einer Verteilung vorkommen.

> **Merke**
>
> Dichtediagramme sind nur für kontinuierliche Wertebereiche erlaubt.

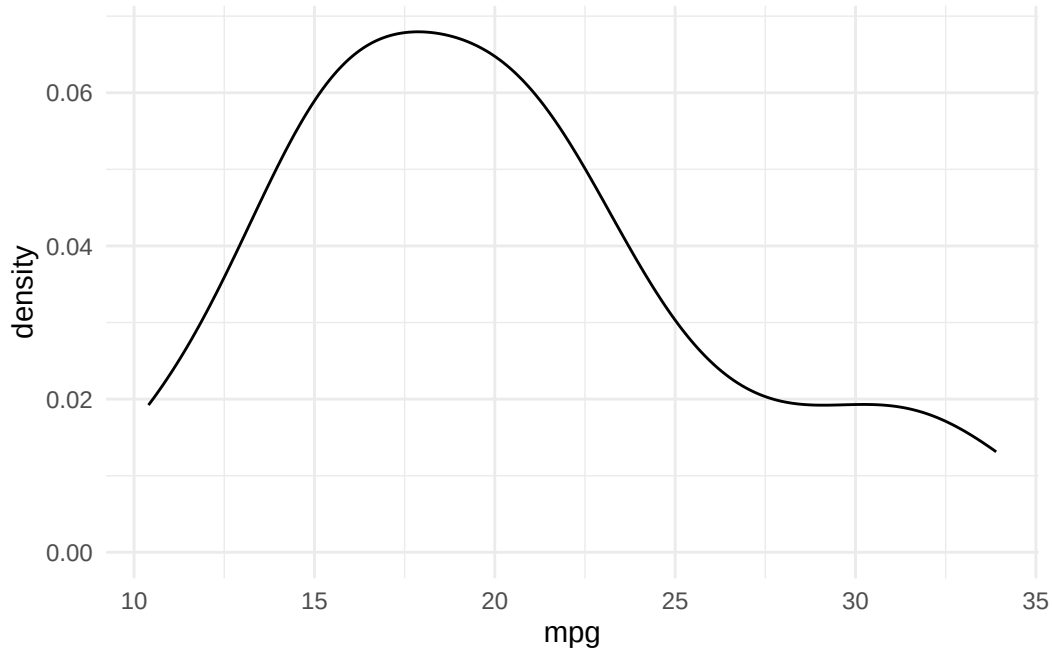


Abbildung 19.5.: Beispiel eines Dichtediagramms

19.2.4. Violindiagramm

Ein **Violindiagramm** verbindet die Idee des Box-Plots mit der Dichtefunktion. Für *kontinuierliche Wertebereiche* zeigen Violindiagramm mehr Details als ein Boxplot.

> **Merke**
>
> Violindiagramme verwenden die Annäherung der Dichtefunktion für die Visualisierung. Entsprechend werden auch nicht gemessene Werte abgebildet.

Die zweite Achse eines Violindiagramms hat keine Bedeutung, sondern dient nur der Darstellung der Dichtefunktion. Hierbei ist zu beachten, dass die Kurve der Dichtefunktion *gespiegelt* ist. Deshalb sollten Violindiagramme nie als Grundlage für eine Schnellanalyse dienen, weil die Flächen leicht falsch interpretiert werden können.

19.3. Plots mit zwei Variablen

19.3.1. Funktionsdiagramme

Definition 19.7. Ein **Funktionsdiagramm** stellt die Parameter und die Ergebnisse einer oder mehrerer (mathematischer) Funktionen gegenüber.

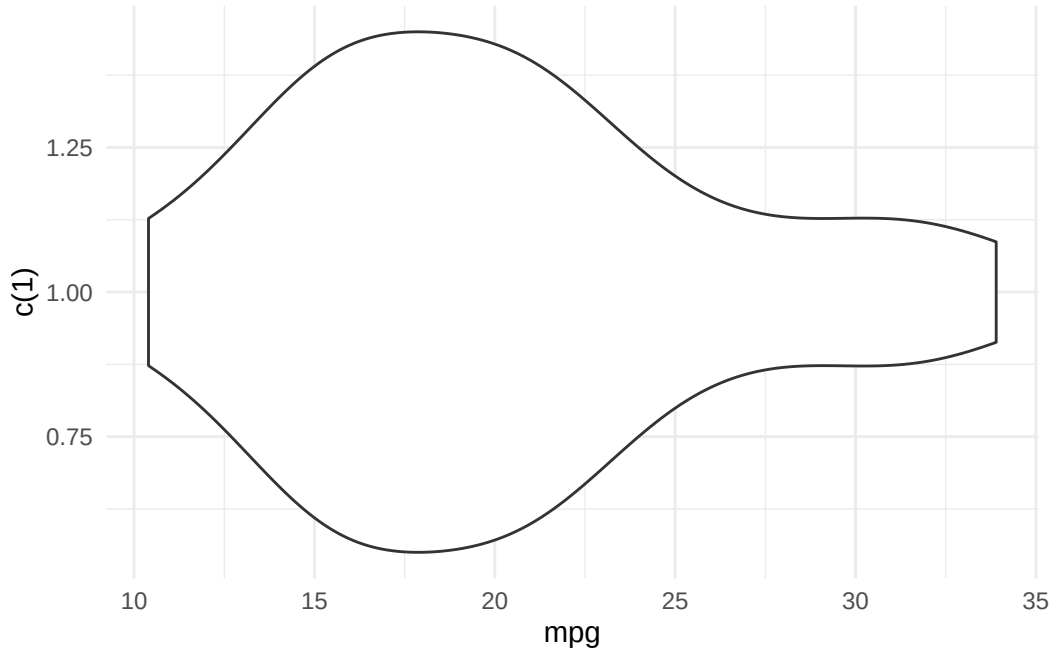


Abbildung 19.6.: Beispiel eines violinplot

In Funktionsdiagrammen werden die Parameter einer Funktion ihren Ergebnissen gegenübergestellt. Zwischen diesen Werten gibt es eine Beziehung, die durch die verwendete Funktion festgelegt wird.

Definition 19.8. Die Parametern und den Ergebnissen eine Funktion sind ***funktional abhängig***. Für den Spezialfall einer linearen Funktion sind die Werte **linear ahbängig**.

Funktionsdiagramme visualisieren also die funktionale Abhängigkeit zwischen Parametern und Ergenissen von Funktionen.

Für ein Funktionsdiagramm wird neben der Funktion zusätzlich ein Parameterintervall benötigt. Das Parameterintervall bildet die Grundlage für den Darstellungsbereich. Anschliessend wird eine Wertetabelle für das Parameterintervall erstellt. Diese Tabelle enthält zwei Vektoren:

- Eine *Sequenz* (s Kapitel 13.1) zwischen der Unter- und der Obergrenze des Parameterintervalls.
- Die Funktionsergebnisse für alle Werte gewählten Parameterintervall.

Praxis

Viele Visualisierungsumgebungen erstellen die Wertetabelle automatich vor der Visualisierung einer Funktion.

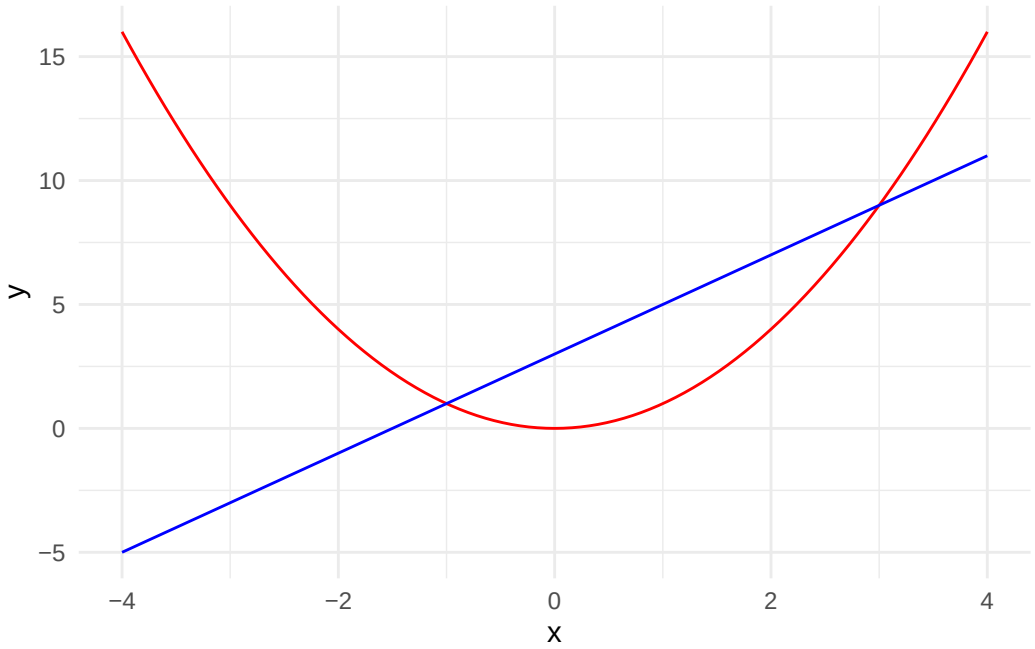


Abbildung 19.7.: Beispiel eines Funktionsplots mit einer linearen und einer quadratischen Funktionen

Bei der Datenvisualisierung werden Funktionsdiagramme zur Vorhersage oder zum Vergleich der Werte eines Vektors durch die Werte eines anderen Vektors verwendet.

Definition 19.9. Ein ***Modell*** ist eine Funktion, die eine Beziehung zwischen Variablen als funktionale Anhängigkeit beschreibt.

19.3.2. Beziehungen

Werden die Werte von zwei Variablen gegenübergestellt, dann wird die Beziehung zwischen den Variablen sichtbar.

> **!** Achtung
>
> Gelegentlich werden Beziehungen zwischen Variablen mit Balken dargestellt. Diese Darstellung ist unzulässt. Balken sollten ausschliesslich zur Darstellung von Häufigkeiten verwendet werden.

Beziehungen zwischen Variablen können auf verschiedene Arten dargestellt werden.

19.3.2.1. Streudiagramme

Normalerweise werden Beziehungen zwischen zwei Variablen mit einem sog. *Streudiagramm* dargestellt.

Ein Streudiagramm stellt die Werte aus zwei Variablen als Punkte dar. Umgangssprachlich werden Streudiagramme als *Punktdiagramme* bezeichnet.

Die Werte der ersten Variablen werden z.B. auf der x-Achse abgetragen, die Werte der zweiten Variablen auf der y-Achse. Die Punkte werden dann an der entsprechenden Stelle im Koordinatensystem dargestellt. Die Wahl der Achse ist beliebig.

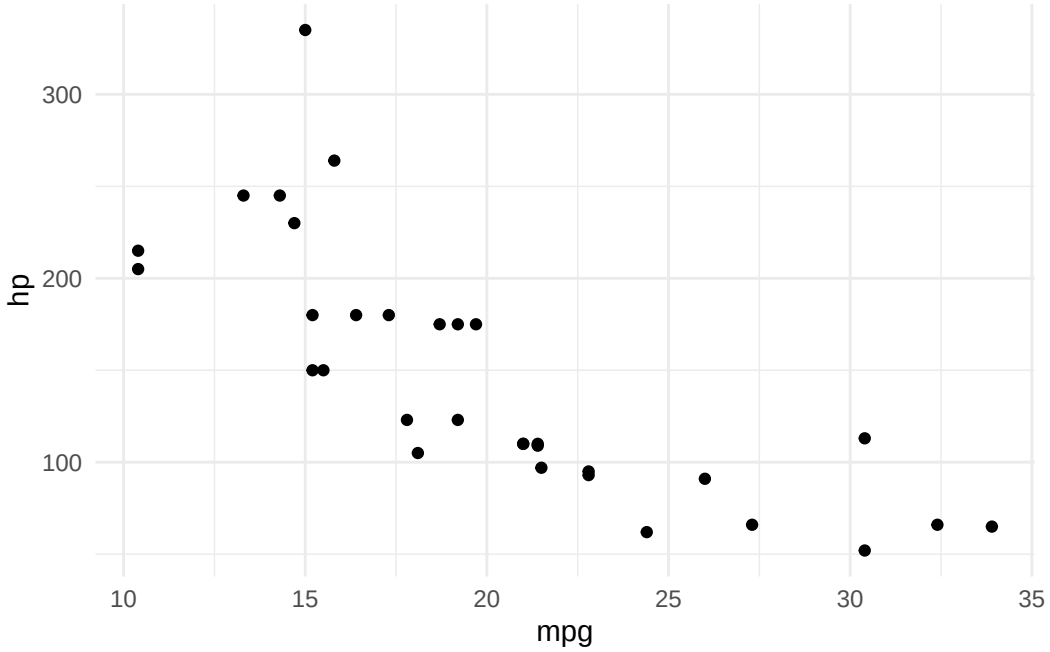


Abbildung 19.8.: Beispiel eines Streudiagramms

19.3.2.2. Jitter-Diagramme

Wenn beide Variablen diskrete Wertebereiche haben, haben die Punkte im Streudiagramm die Tendenz, dass sich die Punkte zu überlagern, weil es nur eine begrenzte Anzahl möglichen Wertepaaren gibt. Um dieses Problem zu umgehen, wird eine besondere Variante des Streudiagramms eingesetzt: Das sog. *Jitter-Diagramm* (etwa *Zitterdiagramm*). Ein Jitter-Diagramm ist eine Variante von Streudiagrammen.

Bei einem Jitter-Diagramm wird um die diskreten Werte ein Bereich festgelegt. **Der eigentliche Wert liegt im Mittelpunkt dieses Bereichs**. Für jeden Messpunkt, wird ein Punkte zufällig innerhalb dieses Bereichs markiert. Durch diese zufällige Positionierung entsteht der Eindruck einer zittrigen Hand.

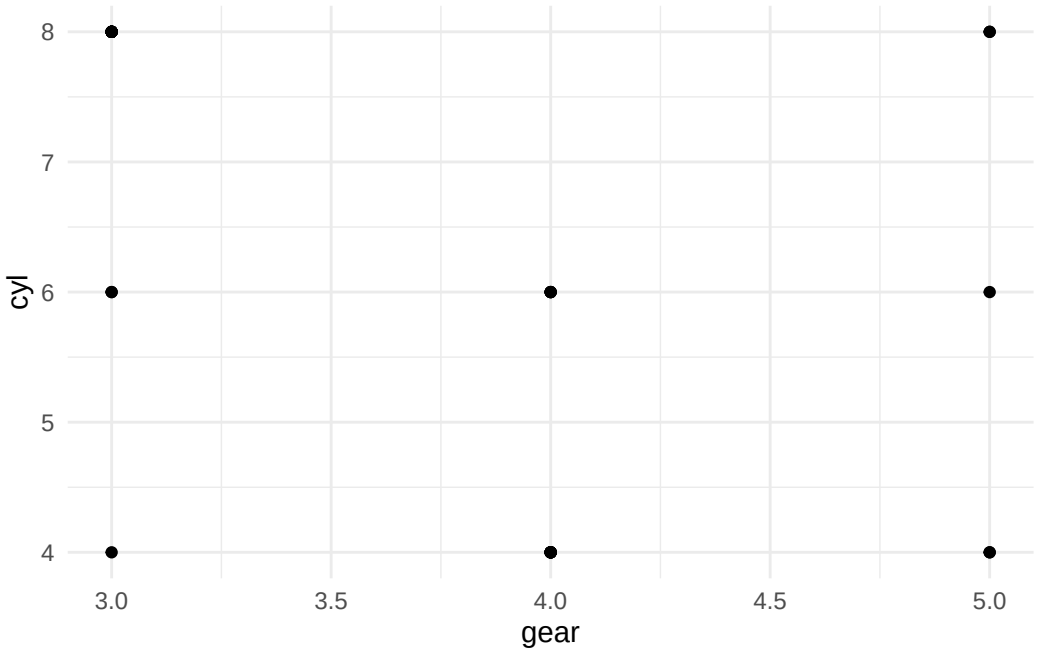


Abbildung 19.9.: Beispiel eines Streudiagramms mit überlagernden Punkten aus zwei diskreten Variablen

Durch das zufällige Positionieren werden die Punkte aufgelockert und die Beziehung der Variablen wird sichtbar. Die Grösse des Bereichs wird so gewählt, dass sich mehrere Bereiche *nicht überlappen*.

> Praxis
>
> Bei der Grösse des Jitter-Bereichs sollte etwas Abstand zu den umgebenden Bereichen gelassen werden. Dadurcht entsteht ein Leerraum, welcher keine Punkte enthält, so lassen sich die einzelnen Bereiche optisch leichter voneinander abgrenzen.

Beim Lesen eines Jitter-Diagramm muss beachtet werden, dass der Bereich für die Punkte dem zugehörigen diskreten Wert entspricht. Durch die Anordnung der Punkte in einem Jitter-Diagramm kann der Eindruck einer Beziehung zwischen zwei Variablen visuell entstehen, ohne dass diese Beziehung tatsächlich existiert.

19.3.2.3. Liniendiagramme

Definition 19.10. Ein ***Liniendiagramm*** eine Variante eines Streudiagramms, bei dem aufeinanderfolgende Punkte durch eine Linie verbunden werden.

Liniendiagramme werden verwendet, wenn die Werte einer Variablen in einer bestimmten Reihenfolge zueinander stehen. Die Werte werden dann auf der entsprechenden Achse in

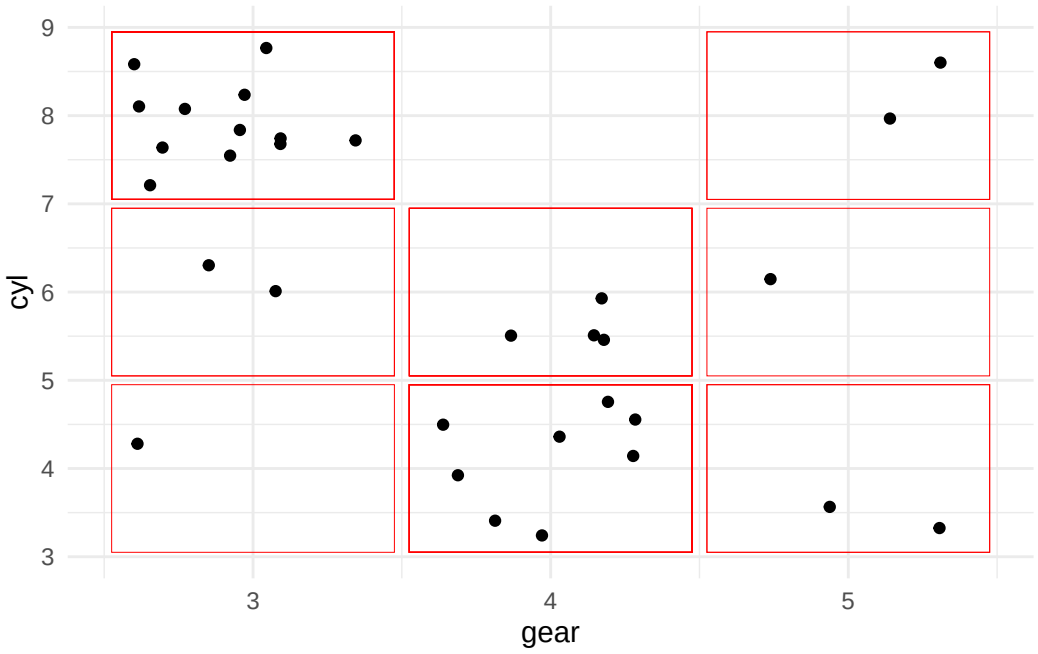


Abbildung 19.10.: Beispiel eines Jitter-Diagramms mit zwei diskreten Variablen. Jitter-Bereiche sind Rot hervorgehoben.

der entsprechenden Reihenfolge abgetragen. Die Punkte werden dann durch eine Linie verbunden.

Liniendiagramme werden häufig verwendet, um die Entwicklung von Variablen über die Zeit darzustellen.

> **Achtung**
>
> Wenn die Werte einer Variablen nicht in einer festen Reihenfolge zueinander stehen, dann dürfen die Werte nicht durch eine Linie verbunden werden! In diesem Fall muss ein Streudiagramm verwendet werden.

19.3.3. Ausgleichsgeraden

Definition 19.11. Eine **Ausgleichsgerade** ist ein Funktionsdiagramm einer Funktion, welche die Ergebnisse einer linearen Funktion zu den tatsächlichen Werten *minimiert*.

Ausgleichsgeraden verwenden *lineare Funktionen* in der Form $ax + c$. Die Funktion einer Ausgleichsgeraden *minimiert* die Funktionsergebnisse mit den tatsächlichen Werten.

Eine Ausgleichsgerade darf nicht willkürlich gezogen werden, sondern muss aus den gemessenen Werten ermittelt werden. Eine Ausgleichsgerade sollte immer gemeinsam mit den tatsächlichen Werten der Variablen dargestellt werden.

Abbildung 19.11.: Beispiel eines Liniendiagramms (DAX Index 1991-1998)

Die *Funktion* einer Ausgleichsgeraden ist ein *einfaches* Modell für die gefundenen Daten. Die Beziehung zwischen den Variablen wird in diesem Modell als *lineare Abhängigkeit* abgebildet.

19.3.4. Zwei unterschiedliche Skalenniveaus kombinieren

Sollen die Werte einer Variable mit diskreten Wertebereich gegen die Werte einer Variablen mit kontinuierlichem Wertebereich abgetragen werden, dann werden die *Verteilungen* der kontinuierlichen Variablen für die einzelnen Werte eines diskreten Wertebereichs dargestellt. Hierzu werden meist *Boxplots* oder *Violindiagramme* verwendet.

Diese Form von kombinierten Diagrammen tritt sehr häufig in Verbindung mit Sekundärindizes auf. Dabei werden Verteilungen von gruppierten Werten gegenübergestellt.

19.4. Plots mit mehr als zwei Variablen

Plots sind auf zwei Dimensionen beschränkt. Werden mehr als zwei Variablen dargestellt, dann werden die zusätzlichen Variablen durch verschiedene Darstellungsformen repräsentiert. 3D-Koordinatensysteme sollten vermieden werden, weil die Position der Punkte bei der 2D-Darstellung eines 3D-Koordinatensystems mehrdeutig ist und sich schwer interpretieren lässt.

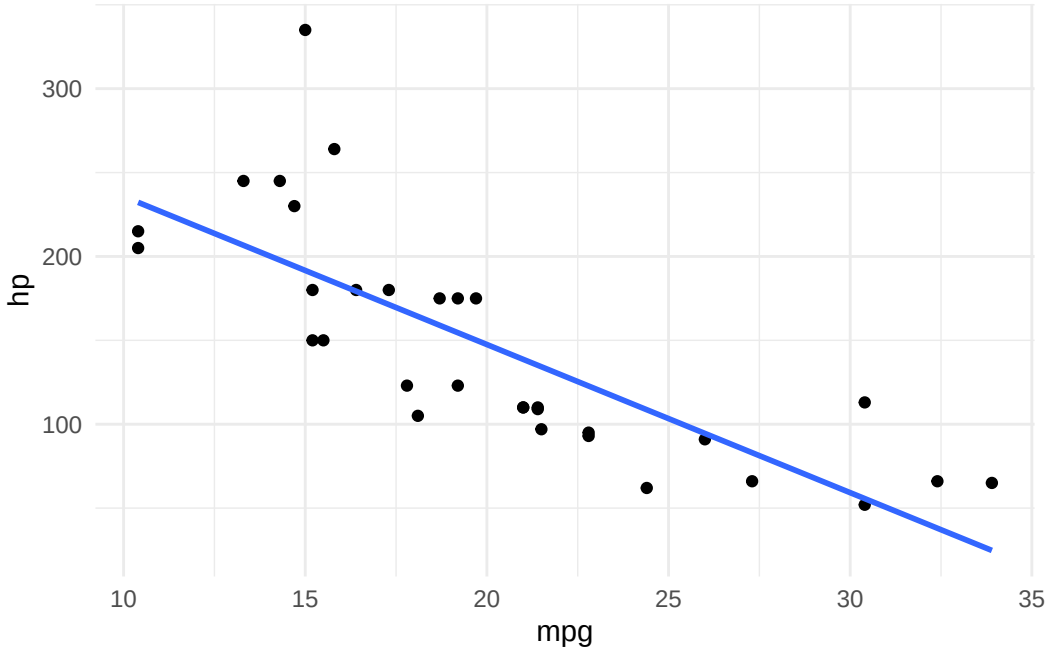


Abbildung 19.12.: Beispiel einer Ausgleichsgeraden in einem Streudiagramm

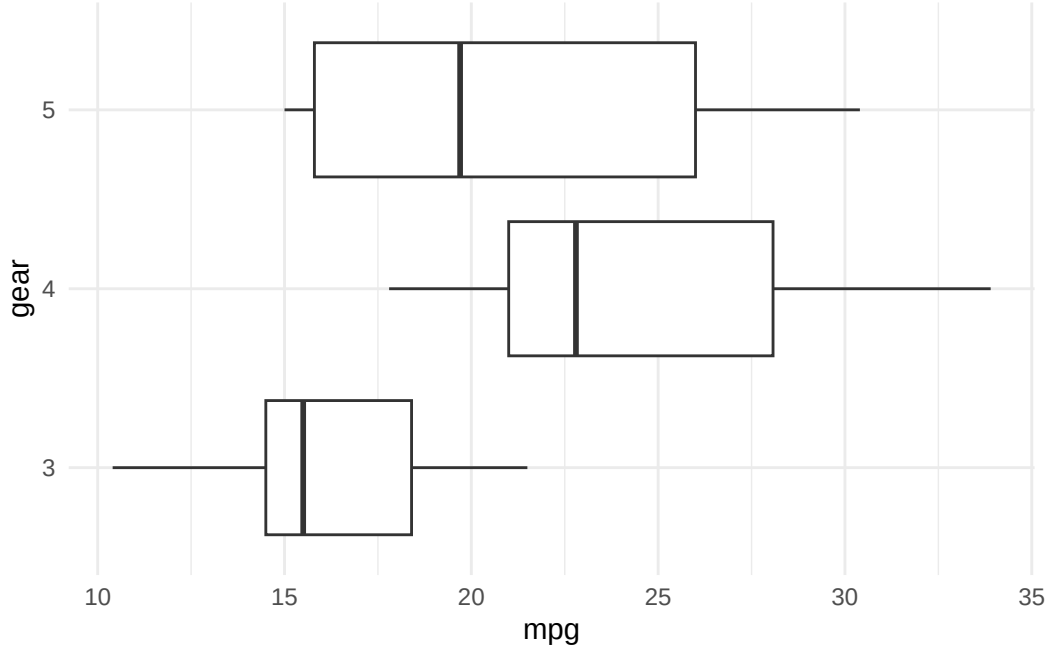


Abbildung 19.13.: Beispiel eines Diagramms mit diskreten und kontinuierlichen Variablen

Für Beziehungen zwischen mehr als zwei Variablen muss für die zusätzlichen Variablen eine andere Darstellungsform gewählt werden. Die wichtigsten Darstellungsformen sind:

- Farbliche Kodierung
- Grössenkodierung
- Form-Kodierung

Diese Varianten können kombiniert werden, um zusätzliche Daten in eine Visualisierungzu integrieren.

> **Wichtig**
>
> Alle zusätzlichen Kodierungen **müssen** in einer *Legende* erklärt werden.

19.4.1. Farbliche Kodierung

Bei der farblichen Kodierung werden die dargestellten Werte in Farben oder Farbtöne übersetzt.

Bei diskreten Wertebereichen sollten für die farbliche Kodierung Farben bzw. Farbtöne verwendet werden, die sich voneinander unterscheiden lassen.

> **Hinweis**
>
> Die farblichen Kodierung sollte sparsam eingesetzt werden, weil die menschliche Farbwahrnehmung stark variiert. Eine vermeintlich gut gewählter Farbton lässt möglicherweise von den Adressaten einer Visualisierung nicht von anderen verwendeten Farben unterscheinde. Dadurch kann die Darstellung nicht wie vorgesehen dekodiert werden.

19.4.1.1. Farbliche Kodierung von diskreten Wertebereichen

Bei diskreten Wertebereichen werden Farben verwendet, die einen möglichst grossen Kontrast zueinander haben.

> **Hinweis**
>
> Diskrete Farben können Menschen nicht beliebig voneinander unterscheiden. Deshalb sollten der Wertebereich einer so kodierter Daten klein sein.

19.4.1.2. Farbliche Kodierung von kontinuierlichen Wertebereichen

Kontinuierliche Wertebereiche werden als sog. Farbgradienten (bzw. Farbverlauf) zwischen zwei Farben kodiert.

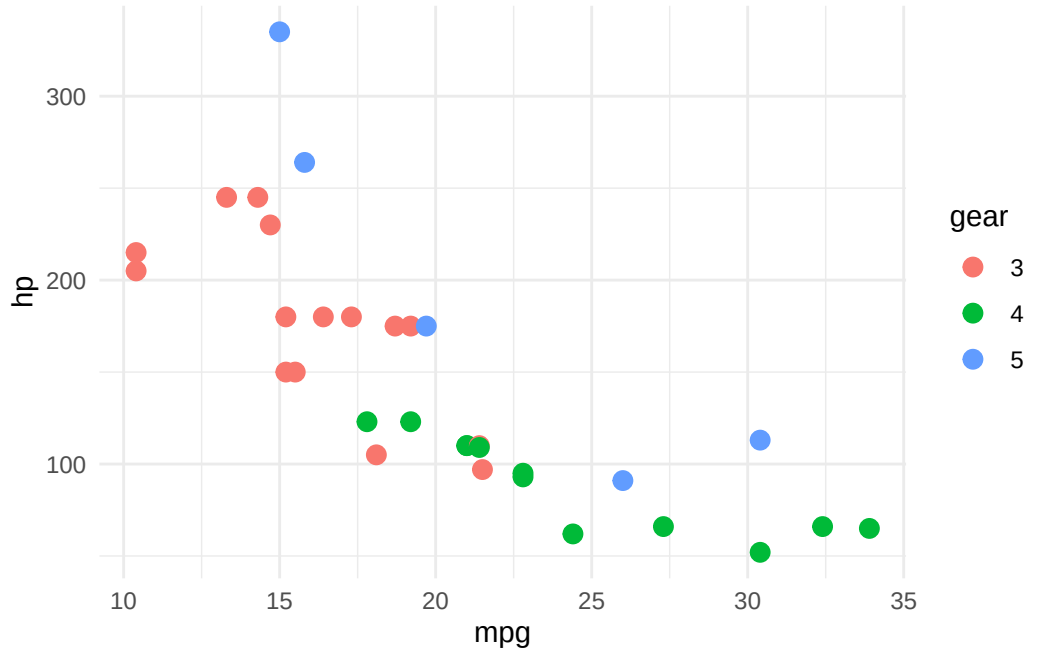


Abbildung 19.14.: Beispiel für die Farbkodierung einer dritten diskreten Dimension

Bei Wertebereichen mit einem definierten Mittelpunkt, wird dieser oft mit einem Übergang über weiss dargestellt. In diesem Fall hat der Gradient 3 Farben. Daraus ergibt sich, dass je heller ein Farbton ist, desto näher ist der Wert am Mittelpunkt. Je nach Färbung ist dann ersichtlich, ob ein Wert ober- oder unterhalb des Mittelpunkts liegt.

Farbgradienten haben den Nachteil, dass die meisten Menschen farbliche Unterschiede nicht stufenlos unterscheiden können. Deshalb sollten Farbgradienten nur verwendet werden, wenn die exakten Werte für die Visualisierung von untergeordneter Rolle sind.

19.4.2. Grössenkodierung

Bei der Grössenkodierung werden die Punkte von Streudiagrammen entsprechend eines dritten Werts vergrössert oder verkleinert.

> **Hinweis**
>
> Eine Grössenkodierung eigent sich gut für kontinuierliche Wertebereiche. Obwohl sich ordinalskalierte Wertebereiche ebenfalls so darstellen lassen, sollte das möglichst vermieden werden, weil bei dieser Darstellungsform verborgen wird, dass es sich um diskrete Daten handelt.

Definition 19.12. Ein **Bubble-Chart** oder **Ballondiagramm** ist ein *Streudiagramm* mit zusätzlichen grössenkodierten Werten.

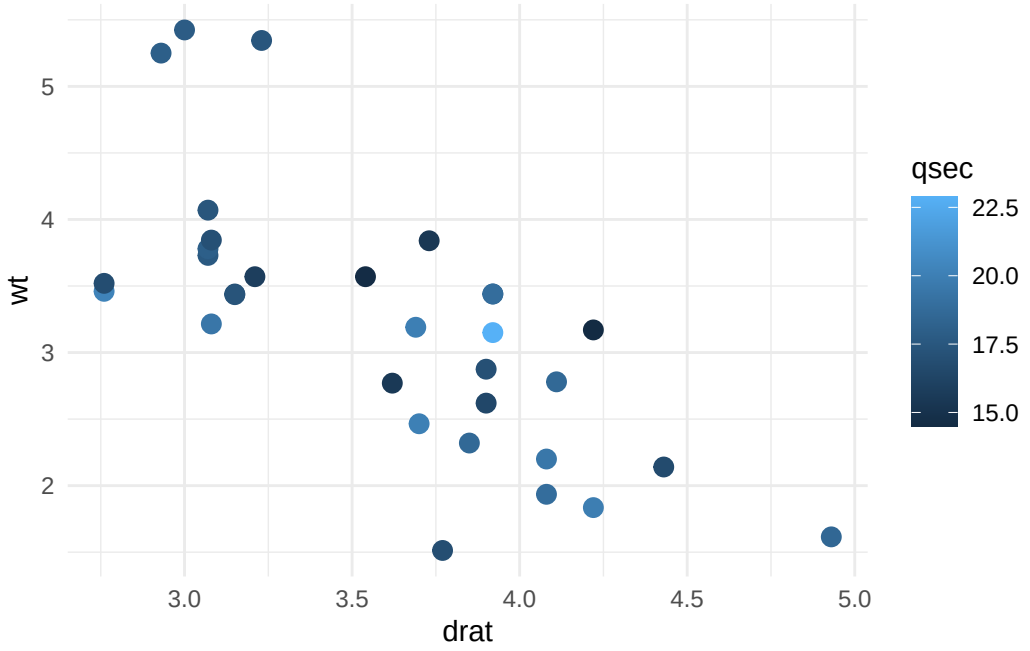


Abbildung 19.15.: Beispiel für die Farbkodierung mit einem Farbgradienten

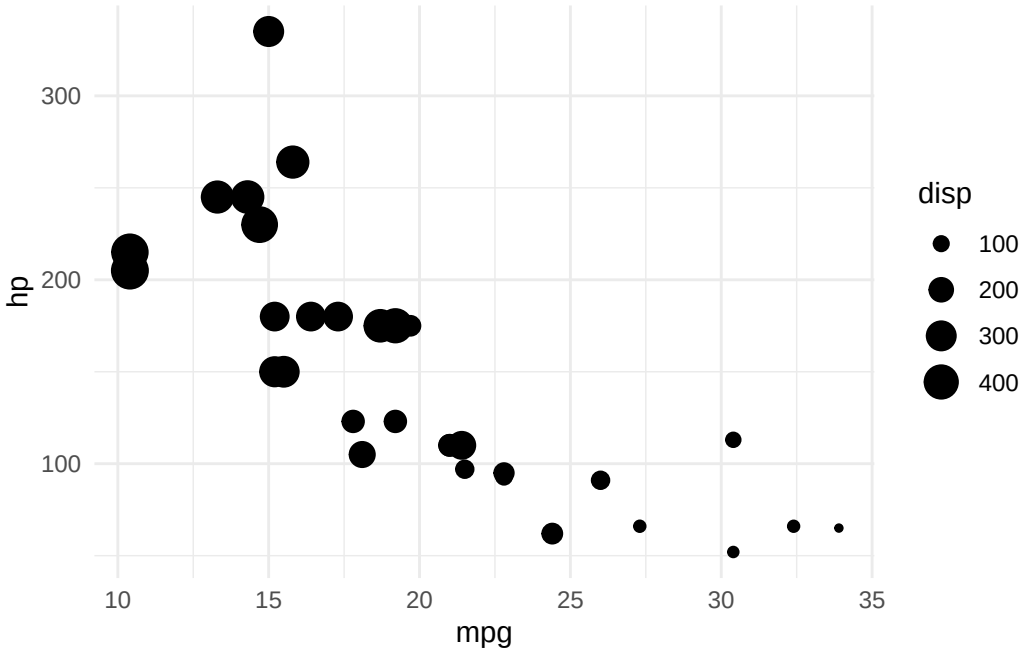


Abbildung 19.16.: Beispiel für die Grössenkodierung einer dritten diskreten Dimension

19.4.3. Form-Kodierung

Über die Form von Punkten oder Linien können zuätzliche Werte in Diagrammen abgebildet werden. Formen eignen sich nur für diskrete Wertebereiche.

Normalerweise wird die Kodierung für Punkte oder für Linien verwendet.

> **Hinweis**
>
> Bei der Formkodierung muss auf die Mindestgrösse geachtet werden, weil sehr kleine Formen nur schwer voneinander unterscheidbar sind.

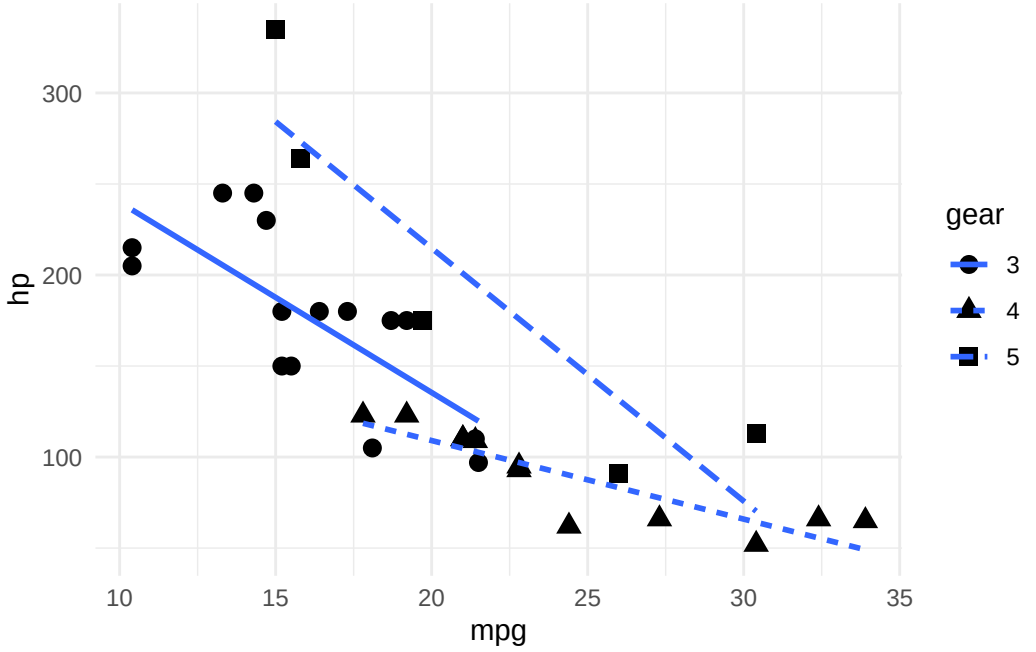


Abbildung 19.17.: Beispiel für die Formkodierung für Punkte und Linien

19.4.4. Teildiagramme

Mit Teildiagrammen werden die Werte entlang einer Variablen in Daten-Segmente getrennt und anschliessend werden die die restlichen Variablen für jedes Segment einzeln visualisiert. Diese Teildiagramme heissen *Facetten*.

Solche Facetten können nur mithilfe *diskreter Daten* gebildet werden. Jede Facette *muss eine Überschrift* haben, die den Wert der Nebendimension zeigt, der die dargestellten Datenpunkte zusammenfasst.

Teildiagramme haben den Nachteil, dass die Daten über verschiedene Diagramme verteilt dargestellt werden. Das erschwert den direkten Vergleich der dargestellten Werte. Deshalb sollten Teildiagramme gewählt werden, wenn durch die isolierte Darstellung bestimmte

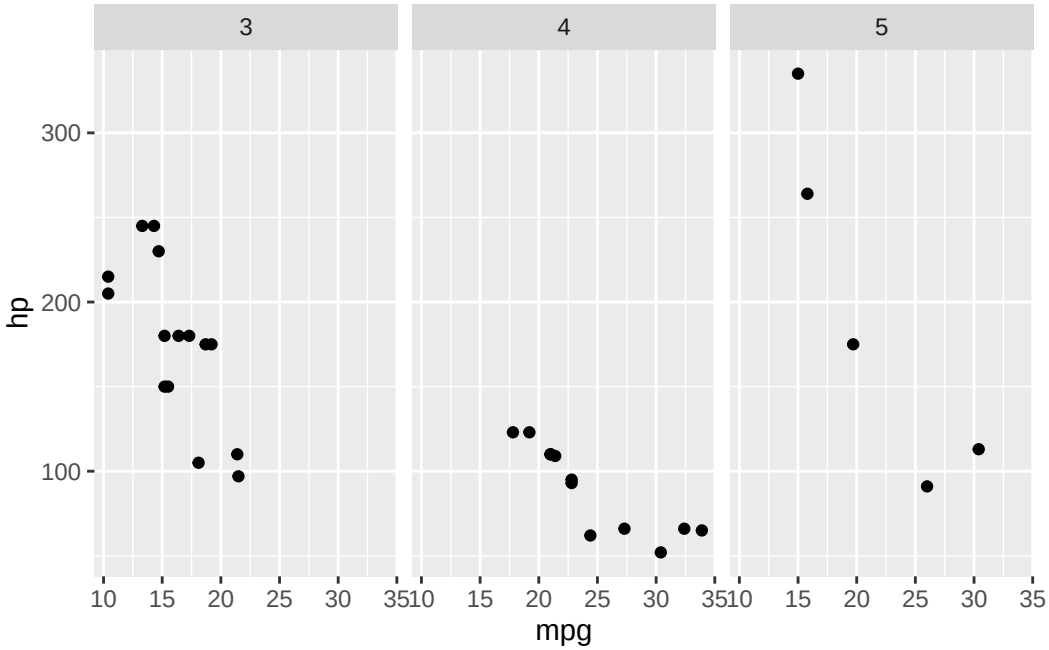


Abbildung 19.18.: Beispiel die Gliederung in Teildiagramme mithilfe einer dritten diskreten Dimension

Aspekte einer Analyse besser hervorgehoben werden. Dazu gehören beispielsweise Beziehungen von Variablen in Gruppen, die mit trotz visueller Kodierungen leicht übersehen werden können.

Referenzen

American National Standards Institute. (1977). *Code for Information Interchange* (ANSI X3.4-1977). https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub1-2-1977.pdf

American Psychological Association. (2020). *Publication manual of the American Psychological Association (7th ed.)*. American Psychological Association. https://doi.org/10.1037/0000165-000

Ben-Kiki, O., Evans, C., & Net, I. döt. (2021). *YAML Ain't Markup Language* (Version v1.2.2). https://yaml.org/spec/1.2.2/

Boole, G. (1847). *The Mathematical Analysis of Logic,Being an Essay Towards a Calculus of Deductive Reasoning*. https://www.gutenberg.org/files/36884/36884-pdf.pdf

Bray, T. (2017). *The JavaScript Object Notation (JSON) Data Interchange Format* (RFC 8259). https://datatracker.ietf.org/doc/html/rfc8259

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)* (Version 1.0). https://www.w3.org/TR/xml/

Carpenter, M. (2023). *[MS-XLSX]: Excel (.xlsx) Extensions to the Office Open XML SpreadsheetML File Format* (Version v23.4). https://learn.microsoft.com/en-us/openspecs/office_standards/ms-xlsx/2c5dee00-eff2-4b22-92b6-0738acd4475e

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, *5*(2), 56–68. https://doi.org/10.2307/2266170

GitHub Inc. (o. J.). *GitHub flow*. https://docs.github.com/en/get-started/quickstart/github-flow

Google. (2023). *Google Forms*. https://docs.google.com/forms

Hickson, I., Pieters, S., Kesteren, A. van, Jägenstedt, P., & Denicola, D. (o. J.). *HTML* (Version 5). https://html.spec.whatwg.org/multipage/

IEEE. (2018). *IEEE Reference Guide*. https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf

ISO/IEC JTC 1/SC 2. (2020). *Information technology — Universal coded character set (UCS)* (ISO/IEC 10646:2020). https://www.iso.org/standard/76835.html

ISO/IEC JTC 1/SC 2/WG 3. (1998a). *DIS 8859-1, 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No.1* (N 411). https://www.open-std.org/JTC1/sc2/wg3/docs/n411.pdf

ISO/IEC JTC 1/SC 2/WG 3. (1998b). *ISO/IEC FCD 8859-15 Information technology – 8 –bit singlebyte coded graphic character sets – Part 15 : Latin Alphabet 0 (Covering the EURO symbol and full support for the French and Finish languages* (N 404). https://www.open-std.org/JTC1/SC2/WG3/docs/n404.pdf

ISO/IEC JTC 1/SC 34. (1999). *Information processing — Text and office systems — Standard Generalized Markup Language (SGML)* (ISO 8879:1986). https://www.iso.org/standard/16387.html

ISO/TC 68/SC 8. (2020). *Financial services — International bank account number (IBAN) — Part 1: Structure of the IBAN* (ISO 13616-1:2020). https://www.iso.org/standard/81090.html

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory*, *2*(2), 127–145. https://doi.org/10.1007/BF01692511

Lindner, P. (1993). *Definition of tab-separated-values (tsv)*. https://www.iana.org/assignments/media-types/text/tab-separated-values

Microsoft. (2023). *Office Forms*. https://www.microsoft365.com/launch/forms

Python Software Foundation. (2013, Juli 6). *Bitwise Operators*. https://wiki.python.org/moin/BitwiseOperators

Rautenberg, W. (2008). *Einführung in die mathematische Logik: ein Lehrbuch* (3., überarb. Aufl). Vieweg + Teubner.

Rodkin, C. (2023). *ACM Citation Style and Reference Formats*. https://www.acm.org/publications/authors/reference-formatting

Sauer, S. (2019). *Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-21587-3

Shafranovich, Y. (2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files* (RFC 4180). https://datatracker.ietf.org/doc/html/rfc4180

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*, 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Shannon, C. E. (1949). Communication Theory of Secrecy Systems. *Bell System Technical Journal*, *28*(4), 656–715. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

The Unicode Consortium. (2022). *Unicode*. https://www.unicode.org/versions/latest/

The University Of Chicago Press Editorial Staff. (2017). *The Chicago Manual of Style, 17th Edition*. University of Chicago Press. https://doi.org/10.7208/cmos17

ZHAW. (2023). *ZHAW Meldung vom 24.8.2023*. https://twitter.com/ZHAW/status/1694731482051309775