



Elmar Wings

Python Packages and Databases for Machine Learning Installation and Use

Version: 1765
March 16, 2023

Contents

Contents	3
List of Figures	13
Acronyms	19
1. To Do	21
I. Overview	23
2. Frameworks and Libraries	25
2.1. Development Environment	25
2.1.1. Jupyter-Notebook	25
2.2. Frameworks	25
2.2.1. TensorFlow	25
2.2.2. TensorFlow Lite	26
2.2.3. TensorRT	26
2.2.4. Keras	26
2.2.5. PyTorch	26
2.2.6. Caffe	27
2.2.7. Caffe2	27
2.2.8. Theano	27
2.2.9. Apache MXNet	27
2.2.10. CNTK	27
2.2.11. Deeplearning4J	27
2.2.12. Chainer	27
2.2.13. FastAI	27
2.3. General libraries	28
2.3.1. NumPy	28
2.3.2. SciPy	28
2.3.3. pandas	28
2.3.4. IPython	28
2.3.5. Matplotlib	28
2.3.6. scikit-learn	28
2.3.7. Scrapy	29
2.3.8. NLTK	29
2.3.9. Pattern	29
2.3.10. Seaborn	29
2.3.11. Bokeh	29
2.3.12. basemap	29
2.3.13. NetworkX	29
2.3.14. LightGBM	29
2.3.15. Eli5	30
2.3.16. Mlpy - Machine Learning	30
2.3.17. Statsmodels - Statistical Data Analysis	30

2.4.	Specialised Libraries	30
2.4.1.	OpenCV	30
2.4.2.	OpenVINO	30
2.4.3.	Compute Unified Device Architecture (CUDA)	30
2.4.4.	OpenNN	31
2.5.	Special Libraries	31
2.5.1.	glob	31
2.5.2.	os	31
2.5.3.	PIL	31
2.5.4.	LCC15	31
2.5.5.	math	31
2.5.6.	cv2	31
2.5.7.	random	31
2.5.8.	pickle	32
2.5.9.	PyPI	32
3.	Libraries, Modules, and Frameworks	33
3.1.	TensorFlow Framework	33
3.1.1.	TensorFlow Use Case	33
3.2.	OpenCV	34
3.2.1.	Application of OpenCV	34
3.2.2.	Image Processing	34
3.2.3.	How does a computer read an image	34
3.3.	MediaPipe	35
3.4.	MediaPipe Applications	35
3.4.1.	Hand Landmarks Detection	35
3.4.2.	Face Detection	36
3.4.3.	MediaPipe Holistic	36
3.5.	Pyfirmata	37
3.6.	Supporting Libraries	37
3.6.1.	Pandas	38
3.6.2.	Numpy	38
3.6.3.	Matplotlib	38
3.6.4.	Scikit-learn	38
3.7.	Important Libraries for Arduino Nano 33 BLE Sense	38
3.7.1.	Tensor flow lite	38
3.7.2.	ArduCAM	39
3.7.3.	ADPS 9960	39
4.	Datenbanken und Modelle für das Maschinelle Lernen	41
4.1.	Datenbanken	42
4.1.1.	Datensatz MNIST	42
4.1.2.	CIFAR-10 und CIFAR-100	44
4.1.3.	Datensatz Fisher's Iris Data Set	45
4.1.4.	Common Objects in Context - COCO	49
4.1.5.	ImageNet	54
4.1.6.	Visual Wake Words	54
4.2.	Modelle	54
II.	Python Packages	55
5.	Machine learning: Neural networks generate content	57
5.1.	Create something new	59
5.2.	Mutual performance monitoring	59

Contents	5
----------	---

5.3. Training in rotation	60
5.4. Training of the discriminator	60
5.5. Training the generator	62
5.6. From theory to practice	62
5.7. The game structure	64
5.8. Improve step by step	64
5.9. GAN-Zoo	65
5.10. Real or computer generated?	66
6. Package Example	67
6.1. Introduction	67
6.2. Description	67
6.3. Installation	67
6.4. Example - Manual	67
6.5. Example	67
6.6. Example - Version	67
6.7. Example - Files	67
6.8. Further Reading	67
7. Data Analysis with Python: First Step with Pandas - Evaluating Football Data	69
7.1. Introduction	69
7.2. Installation	69
7.3. Understand <code>Series</code>	69
7.4. Create <code>DataFrame</code>	70
7.5. Select data	71
7.6. Integer	72
7.7. Get soccer data	72
7.8. Show info	73
7.9. Goal statistics	74
7.10. Format data	74
7.11. Date	75
8. Package Example	77
8.1. Introduction	77
8.2. Description	77
8.3. Installation	77
8.4. Example - Manual	77
8.5. Example	77
8.6. Example - Version	77
8.7. Example - Files	77
8.8. Further Reading	77
9. Barcode	79
9.1. Introduction to Barcode	79
9.1.1. 1-D Linear Bar Code	79
9.2. Types of Barcodes	80
9.2.1. UPC Code	80
9.2.2. EAN Code	80
9.2.3. Code 128	80
9.2.4. Code 39	82
9.2.5. Extended Code 39	82
9.2.6. Code 93	82
9.2.7. Coda bar	83

9.3.	2-D Matrix Code	83
9.3.1.	QR-Code	84
9.3.2.	How QR-Code works Technically.	84
9.3.3.	Basic steps How QR (Quick Response) Works.	84
9.3.4.	QR-Code Versions.	85
9.3.5.	Patterns In QR-Code.	85
9.3.6.	Versions and Data Density:	86
9.3.7.	Types of 2D-Code and their Features	86
9.4.	Software	89
9.5.	Hardware	89
9.6.	function	89
9.7.	Limitations	89
9.8.	output	89
9.9.	How to Run	89
10. EasyOCR		93
10.1.	Introduction	93
10.2.	Description	93
10.3.	Installation	94
10.4.	Example	94
10.5.	Further Reading	96
10.5.1.	Advanced Tuning	97
11. YOLO v5		99
11.1.	Introduction	99
11.2.	Description	99
11.3.	Installation	100
11.4.	Example	100
12. Description of the Python Package: Keras		103
13. Description of the Python Package: NumPy		107
13.1.	Functions:	107
13.2.	Features:	107
13.3.	Applications:	107
13.4.	Installation	107
13.5.	Further Reading:	108
14. Description of the Python Package: Matplotlib		111
14.1.	Functions:	111
14.2.	Features:	111
14.3.	Applications:	111
14.4.	Installation	112
14.5.	Further Reading:	112
III. Add Ons		115
15. Programmieren mit Python: Einfache grafische Oberfläche mit Tkinter erstellen		117
15.1.	Tk gegen Qt	117
15.2.	Erstes Fenster erstellen	117
15.3.	Hauptfenster erstellen	119
15.4.	Gitter erstellen	120
15.5.	Frame platzieren	120

15.6. Widgets erstellen	121
15.7. Beschreibungstext	123
15.8. Labels	123
15.9. Eingabefelder	124
15.10Button	124
15.11Kosmetik	124
15.12Button klickbar machen	125
15.13Fake-Browser und Login-Daten	126
15.14Fehlermeldung	126
15.15Korrekter Login	127
15.16Menüleiste erstellen	128
15.17Fazit	129
16. Hello World “Iris”	131
16.1. Your First Machine Learning Project in Python Step-By-Step	131
16.2. How Do You Start Machine Learning in Python?	131
16.2.1. Python Can Be Intimidating When Getting Started	131
16.2.2. Beginners Need A Small End-to-End Project	131
16.2.3. Hello World of Machine Learning	132
16.3. Machine Learning in Python: Step-By-Step Tutorial (start here)	132
16.4. Step 1: Downloading, Installing and Starting Python SciPy	133
16.4.1. Install SciPy Libraries	133
16.4.2. Start Python and Check Versions	133
16.5. Step 2: Load The Data	134
16.5.1. Import libraries	135
16.5.2. Load Dataset	135
16.6. Step 3: Summarize the Dataset	135
16.6.1. Dimensions of Dataset	136
16.6.2. Peek at the Data	136
16.6.3. Statistical Summary	137
16.6.4. Class Distribution	137
16.6.5. Complete Example	137
16.7. Step 4: Data Visualization	137
16.7.1. Univariate Plots	139
16.7.2. Multivariate Plots	139
16.7.3. Complete Example	141
16.8. Step 5: Evaluate Some Algorithms	141
16.8.1. Create a Validation Dataset	142
16.8.2. Test Harness	143
16.8.3. Build Models	143
16.8.4. Select Best Model	143
16.8.5. Complete Example	145
16.9. Step 6: Make Predictions	145
16.9.1. Make Predictions	145
16.9.2. Evaluate Predictions	147
16.9.3. Complete Example	147
16.10. You Can Do Machine Learning in Python	147
16.11. Summary	149
16.12. Your Next Step	149
17. Datenanalyse mit Python: Erste Schritte mit Pandas – Fussballdaten auswerten	151
17.1. Einleitung	151
17.2. Installation	151
17.3. <code>Series</code> verstehen	151

17.4. <code>DataFrame</code> erstellen	152
17.5. Daten auswählen	153
17.6. Integer	154
17.7. Fußball-Daten holen	154
17.8. Infos anzeigen	155
17.9. Tor-Statistiken	156
17.10Daten formatieren	157
17.11Datum	158
18. Pandas Gene	159
18.1. Datamining in sequenzierten Gendaten mit Pandas	159
18.2. Einführung	159
18.3. Experimentierumgebung	159
18.4. CSVs einlesen	160
18.5. Interessante Stellen	161
18.6. Prüfvereinigung	161
18.7. Daten gebändigt	163
18.8. Literatur	163
19. Datenvisualisierung mit Python	165
19.1. Plotexpress	165
19.2. Einführung	165
19.3. Installation: Anaconda	165
19.4. Installation: Pip	166
19.5. Wie gebe ich Geld aus?	166
19.6. Folgt das Gewicht dem Essen?	168
19.7. Sind Kalorien und Gewicht überhaupt korreliert?	171
19.8. Wöchentliche Durchschnitte korreliert?	172
19.9. Schnelle Statistiken	174
19.10Wofür gebe ich Geld aus?	174
19.11Gewicht folgt Konsum?	174
19.12Tasks	175
20. Welches Diagramm passt zu meinen Daten?	177
20.1. Diagramm-Typen	177
20.2. Kunst-Graph	179
20.3. Datenformate	180
20.4. JSON und GeoJSON	180
21. Daten verarbeiten mit Python: Pandas-Bibliothek für SQL-Umsteiger	183
21.1. Einleitung	183
21.2. Datenquellen	183
21.3. Spalten auswählen	184
21.4. Index anlegen	186
21.5. Datum parsen	186
21.6. Einschränkungen mit <code>where</code> -Klausel	188
21.7. Distinct Values	189
21.8. Spaltennamen ändern	189
21.9. Daten speichern	190
21.10Daten gruppieren	190
21.11Mit gruppierten Daten rechnen	191
21.12Gruppierte Daten einschränken	192
21.13Fazit	192

22. Analyse von Open Data mit Pandas	195
22.1. Einleitung	195
22.2. Immer mehr freie Daten	195
22.3. Blick auf Pandas	195
22.4. Beliebige Datenquellen	196
22.5. Wetterdaten für Deutschland	196
22.6. Visualisierung der Daten	198
22.7. Neue Erkenntnisse	199
23. Datenqualität mit der Python-Bibliothek Great Expectations sichern	201
23.1. Einleitung	201
23.2. Wenn der Data Lake versumpft	201
23.3. Schnellstart mit CVS und pandas	202
23.4. Datenerkundung im Notebook	203
23.5. Wie warm wird es in Brasilien?	203
23.6. Was tun bei unerfüllten Erwartungen?	205
23.7. Fazit	207
23.8. Quellen	208
24. Eingefangen – Data Wrangling mit pandas	209
24.1. Jupyter-Notebook als Arbeitsumgebung	209
24.2. Daten mit pandas filtern	215
24.3. Werte in DataFrames ersetzen	218
24.4. Mit DataFrames rechnen	222
24.5. Fazit	224
25. Programmieren mit Python: Bedienoberfläche via PyQt erstellen	225
25.1. PyQt gegen Pyside und Tkinter	225
25.2. Hello World mit PyQt	225
25.3. QApplication	226
25.4. Hauptfenster erstellen	226
25.5. Fenster aufrufen, Schleife starten	226
25.6. Fenstertitel ändern	228
25.7. Größe ändern und Fenster verschieben	228
25.8. Login-Fenster erstellen	229
25.9. Widgets hinzufügen	232
25.10 Layout verwalten	232
25.11 Widgets erstellen	233
25.12 Elemente ausrichten	234
25.13 Button klickbar machen	236
25.14 Einloggen	236
25.15 Menü hinzufügen	238
25.16 Fazit	240
26. GUI für Python: Bedienoberfläche per Drag-and-Drop mit dem Qt Designer erstellen	241
26.1. Installation	241
26.2. Erster Start	241
26.3. Bedienoberfläche	242
26.4. Hello World!	243
26.5. Bedienoberfläche in Python integrieren	243
26.6. pyuic5	245
26.7. <code>loadUi()</code>	247
26.8. Passendes Widget	247
26.9. Login-Fenster erstellen	248

26.10Layout	248
26.11Widgets	248
26.12Button klickbar machen	249
26.13User-Agent	250
26.14Login-Daten	251
26.15Beschreibungstext ändern	251
26.16Menü hinzufügen	251
26.17Ausblick	252
27. Programmieren mit Python: Schnittstellen entwickeln mit Pycharm und FastAPI	253
27.1. Warum Pycharm?	253
27.2. Warum FastAPI?	254
27.3. Projekt in Pycharm einrichten	254
27.4. Pakete installieren	255
27.5. Tipps und Tricks mit Pycharm	256
27.6. Daten pflegen mit FastAPI	257
27.7. Aufbau des Projekts	257
27.8. Hello World mit FastAPI	257
27.9. Kein Coden ohne Testen	259
27.10Gegenstände mit FastAPI verwalten	261
27.11Listeneinträge zurückgeben	262
27.12Daten asynchron verarbeiten	263
27.13Testen, testen testen	264
27.14OpenAPI für den Item-Lebenszyklus	266
27.15Ausblick	267
28. Große Pandas	269
28.1. Programmieren mit Python: Große Datenmengen verwalten mit vaex	269
28.2. vaex im Einsatz	269
28.3. Fazit	270
28.4. Tasks	270
29. Durchdacht in die Cloud	271
29.1. Einführung	271
29.2. Lift and Shift und andere Arten der Migration	271
29.3. Vorteile der Datenbankmigration in die Cloud	272
29.4. Auswahlkriterien für die richtige Datenbankplattform	273
29.5. Tipps für die Auswahl der Cloud-Datenbank	274
29.6. Nie ohne Strategie beginnen	275
29.7. Die Deadline im Blick haben	275
29.8. Was kommt zuerst?	275
29.9. Fazit: Die Datenbankmigration beginnt im Kopf	276
29.10Tasks	277
30. Cloud-Datenbanken der Hyperscaler	279
30.1. Einleitung	279
30.2. Auch für Anbieter praktisch	280
30.3. AWS: der Platzhirsch	280
30.4. AWS Redshift als Data Warehouse	282
30.5. Key-Value Stores übernehmen	282
30.6. Weitere Optionen	282
30.7. Azure: Der ewige Zweite?	283
30.8. Auch Open Source ist möglich	284
30.9. Cosmos für In-Memory und Cache für Redis	285

30.10Auch Azure hilft bei der Migration	285
30.11Google: im Schatten von AWS und Azure	286
30.12PostgreSQL als Ersatz	287
30.13Noch weitere DB-Arten im Angebot	287
30.14Andere Datenbankanbieter in der Cloud	288
30.15Fazit	288
30.16Quellen	290
30.17Tasks	290
31. Fünf Cloud-native SQL-Datenbanken	291
31.1. Einführung	291
31.2. Cloud-native Datenbanken existieren	292
31.3. YugabyteDB: Sharding und viel Abstraktion	292
31.4. Raft für den Konsens	293
31.5. Gute Cloud-Integration	294
31.6. Citus: auch PostgreSQL, aber anders	294
31.7. Eigentlich angelegt wie ein Plug-in	294
31.8. Für den Betrieb ist viel Handarbeit nötig	296
31.9. Vitess: eigener Unterbau und MySQL zur Abwechslung	296
31.10Zentrale Gateway-Instanzen steuern den Zugriff	297
31.11TiDB: fit für OLTP, OLAP, HTAP und ACID	298
31.12Ordentlicher Bums in der Datenbank	299
31.13CockroachDB: komplexer Veteran	299
31.14Im Layer-Fahrstuhl nach unten	300
31.15Reif für den Cloud-Einsatz	301
31.16Fazit: heiter bis wolkig	301
31.17Quellen	302
31.18Tasks	302
32. Datenbank: TimescaleDB 2.0 erfasst und analysiert große Zeitreihen	303
32.1. Einführung	303
32.2. Kontinuierliches Aggregieren über aktualisierte APIs	303
32.3. Wieso eine eigene Datenbank für Zeitreihen?	303
32.4. Weiterführende Hinweise	304
33. How to Choose a Forecasting Model – Decision tree to select a time-series methodology	305
33.1. Key takeaways	305
33.2. Introduction	305
33.3. Step 1: Analytical problem framing	306
33.4. Step 2: Data exploration and requirements	306
33.5. Step 3: Choosing a model	307
33.6. Step 4: Model performance and operations	308
33.7. Conclusion	308
33.8. References	309
34. Image Filters with Python – A concise computer vision project for building image filters using Python	311
34.1. Starter code for Brightness and Contrast modifier with Python:	311
34.2. Further development of the project with controllers:	313
34.3. Conclusion:	314
35. Reinforcement Learning mit Python: Wie eine KI lernt, ein Spiel zu gewinnen	317
35.1. Gefrorener See	317

35.2. Librarys installieren	317
35.3. Policy	319
35.4. Q-Learning	321
35.5. Exploration vs. Exploitation	321
35.6. Berechnung der Q-Werte	322
35.7. Die Trainingsschleife	323
35.8. Evaluierung	324
35.9. Ausblick	326
36. Machine learning: Neural networks generate content	327
36.1. Create something new	329
36.2. Mutual performance monitoring	329
36.3. Training in rotation	330
36.4. Training of the discriminator	330
36.5. Training the generator	332
36.6. From theory to practice	332
36.7. The game structure	334
36.8. Improve step by step	334
36.9. GAN-Zoo	335
36.10Real or computer generated?	336
37. Daten visualisieren mit Grafana	337
37.1. Container-Beobachter	337
37.2. Schnellstartanleitung	339
37.3. Diagramm anzeigen und anpassen	339
37.4. Schwellwerte definieren	341
37.5. Anzeige im Vollbild	342
37.6. Alarme aktivieren	343
37.7. Datenvisualisierung: Grafana 9.4 überarbeitet Panels und Navigation .	343
37.8. Neuerungen für Canvas- und Dashboard-Panels	343
37.9. Überarbeitete Navigation und Authentifizierung	344
38. Further Readings	345
39. Matplotlib für Einsteiger: So erzeugen Sie ganz einfach Diagramme mit Python	347
39.1. Matplotlib einrichten	347
39.2. Liniendiagrammen erzeugen	348
39.3. Titel und Legenden rendern	350
39.4. Tortendiagramme erzeugen	351
39.5. Balkendiagramme rendern	352
39.6. Funktionen und Histogramme	353
39.7. Histogramme	356
39.8. Literatur zu Matplotlib	357
39.9. Fazit	357
Literaturverzeichnis	359
Index	363
Index	363

List of Figures

2.1.	Tensorflow Datenflussdiagramm	26
3.1.	Hand Landmarks	35
3.2.	Face Detection Using Landmarks	36
3.3.	Face, Pose, and Hand Landmarks	36
3.4.	Serial Communication protocol using Pyfirmata	37
4.1.	Beispiele aus dem Trainingssatz des Datensatzes MNIST [SSS19]	43
4.2.	Auszug von zehn Abbildungen des Datensatz Canadian Institute For Advanced Research (CIFAR)-10	44
4.3.	Oberkategorien und Kategorien in Canadian Institute For Advanced Research (CIFAR)-100 mit den Originalbezeichnungen	45
4.4.	Kopfzeilen des Datensatzes Fisher's Iris Data Set	48
4.5.	Ausgabe der Kategorien des Datensatzes Fisher's Iris Data Set	49
4.6.	Namen der Kategorien im Datensatz Fisher's Iris Data Set	49
5.1.	Where can this landscape be found? In Germany? Or in France?	58
5.2.	NVIDIA's StyleGAN project generate images of non-existent people.	58
5.3.	University of Berkeley's CycleGAN transfers image style to new images	59
5.4.	The loss functions control the learning of the neural networks in GANs	60
5.5.	The mathematical representation of a loss function consists of two sums.	60
5.6.	GANs involve two networks competing against each other, unlike simpler neural networks	61
5.7.	Python GAN learns to generate handwritten digits	62
9.1.	1-D Linear Bar Code	80
9.2.	UPC-Code	81
9.3.	EAN-Code	81
9.4.	CODE-128	82
9.5.	2D Matrix	83
9.6.	QR Code	84
9.7.	Version 1-3	85
9.8.	Version 38-40	86
9.9.	Pattern in QR Code	87
9.10.	Version 1	87
9.11.	Version 10	87
9.12.	Version 40	88
9.13.	Type of 2D Code	88
9.14.	1-D Linear Bar Code	90
9.15.	1-D Linear Bar Code	90
9.16.	1-D Linear Bar Code	91
10.1.	EasyOCR Installation using Command Prompt	94
10.2.	Code Snippet	95
10.3.	Multi-Line recognition using EasyOCR	96
11.1.	YOLOv5 Result	101

14.1. Output of the example code	113
14.2. Output of the 2 nd example code	113
15.1. Das erste Fenster ist grau und unspektakulär. Aber ein Fenster.	118
15.2. Hallo Welt! Das war nicht schwer.	119
15.3. Das Gewicht der Spalten beeinflusst ihre Größe.	120
15.4. Der Titel des Fensters ist schon anders, ansonsten ist das Fenster wüst und leer.	121
15.5. Sieht aus wie ein Login-Fenster, tut aber noch nichts.	124
15.6. Da ist was schiefgelaufen.	127
15.7. Erfolg: Der Login hat geklappt.	128
15.8. Ein gewohntes Menü mit einem gewohnten Befehl.	129
16.1. Box and Whisker Plots for Each Input Variable for the Iris Flowers Dataset	139
16.2. Histogram Plots for Each Input Variable for the Iris Flowers Dataset . .	140
16.3. Scatter Matrix Plot for Each Input Variable for the Iris Flowers Dataset	141
16.4. Box and Whisker Plot Comparing Machine Learning Algorithms on the Iris Flowers Dataset	145
18.1. Pandas Dataframes in Jupyter	160
18.2. Messung von SNPs bei My Heritage und Ancestry	162
19.1. Kalorien und Gewichtsveränderung	170
19.2. Streudiagramm mit wochenweise gemittelten Werte	173
19.3. Summe der Ausgaben	174
19.4. Kurven für Kalorien und Gewicht	175
20.1. Dataviz Catalogue	178
20.2. Analysekarte	179
20.3. Long Format	181
20.4. GeoDa	182
21.1. Die ersten fünf Datensätze werden angezeigt.	184
21.2. <code>.info()</code> zeigt die Spaltennamen, die Anzahl der non-null-Werte und den Datentyp.	185
21.3. Die Spalte OBJECTID ist nun der Index.	187
21.4. Das Datum wird nun in einem ordentlichen Format angezeigt.	187
21.5. Die Auswahl haben Sie auf Niedersachsen beschränkt.	188
21.6. Mehrere Bundesländer zeigen Sie über eine Liste an, die als Filter verwendet wird.	188
21.7. Die größten Städte kommen am Anfang.	189
21.8. Pandas erstellt eine Liste aller Landkreise.	189
21.9. Die Ausgabe zeigt die neuen Spaltennamen samt neuen Indexnamen an.	190
21.10Für eine neue SQL-Tabelle müssen Sie vorher keine leere Tabelle anlegen.	190
21.11Eine einfache Ausgabe in Excel.	191
21.12Die Landkreise Schleswig-Holsteins stehen nun ganz oben.	191
21.13Der Durchschnitt der Todesfälle über alle Landkreise in den jeweiligen Bundesländern.	192
21.14Die gruppierten Daten können Sie mit Filter weiter einschränken. . . .	193
22.1. Darstellung der Seitenaufrufe des Artikels über Klimaanlagen bei Wikipedia und der Temperaturmaxima in Deutschland; Datenquelle: Meteostat, pageviews.toolforge.org	198

23.1. Der Kaggle-Datensatz im Beispiel zeigt Temperaturwerte aus verschiedenen Orten im Süden Brasiliens.	203
23.2. Great Expectations lässt sich als Validierungsinstanz in Datenpipelines einbinden. In diesem Beispiel sind die Verantwortlichkeiten von Data Engineers und Domänenexperten sauber getrennt.	207
24.1. Die Funktion <code>display()</code> aus dem Paket IPython formatiert die Ausgaben im Unterschied zu <code>print()</code> wie in pandas vorgesehen.	210
24.2. Vor dem Verändern der Spaltennamen sollten die ursprünglichen Spalten gesichert werden.	211
24.3. Die Methode <code>df.info()</code> gibt unter anderem den Speicherbedarf des Datensatzes aus.	212
24.4. Eine Anpassung der Datentypen spart Arbeitsspeicher und erleichtert die Weiterverarbeitung.	213
24.5. Ein Vorher-nachher-Vergleich zeigt, dass die Anpassung der Datentypen keinen Fehler produziert hat.	214
24.6. Die Zahl der IDs entspricht der Zeilenzahl, also sind alle IDs einzigartig.	215
24.7. Frequenz und Anteil der Familienstände. Die größte Personengruppe ist die der Verheirateten; ihr Anteil beträgt 38 Prozent.	216
24.8. Bei der grafischen Darstellung des Datensatzes fallen drei ungewöhnliche Familienstände auf.	217
24.9. Die drei Zeilen mit dem Familienstand Alone lassen sich mit einer passenden Maske ermitteln.	218
24.10 Mit einer Oder-Verknüpfung werden drei Angaben aus dem DataFrame gefiltert, die einer gesonderten Weiterbehandlung bedürfen.	219
24.11 Die Methoden <code>.loc</code> und <code>.iloc</code> bergen entscheidende Unterschiede hinsichtlich ihrer Argumente.	219
24.12 Damit fehlerhafte Kategorien im weiteren Verlauf der Datenanalyse keinen Schaden anrichten, lassen sie sich durch NAN-Werte ersetzen (not a number), hier aus dem Paket NumPy.	220
24.13 Sofern es nur einen kleinen Bruchteil aller Daten betrifft, lassen sich fehlerhafte oder unvollständige Angaben ohne spürbare Nebenwirkungen aus einem DataFrame entfernen oder durch Mittelwerte ersetzen.	221
24.14 Beim Suchen und Ersetzen fehlerhafter Daten vervielfacht sich das Tempo, wenn pandas-interne Methoden zum Einsatz kommen und kein Python-Code.	223
24.15 Auch das Errechnen neuer Spalten auf der Basis bestehender Werte (Feature Engineering) ist mit pandas schnell erledigt.	224
25.1. Das erste Fenster ist noch grau und leer.	227
25.2. Damit steht "Hello World" auf dem Bildschirm.	228
25.3. Etwas gestaucht, aber "Hello World" taucht nun im Fenster auf.	228
25.4. Nun steht "Hello World" zentriert im Fenster.	229
25.5. Auch das Login-Fenster beginnt klein, leer und grau.	231
25.6. Das sieht schon nach einem Login-Fenster aus.	234
25.7. Jetzt hat das Fenster die passende Größe.	235
25.8. Oh, da ist etwas schiefgelaufen.	237
25.9. Mit einem Menü ist das Fenster komplett.	238
25.10 PyQt (links) wirkt moderner als das Login-Programm mit Tkinter (rechts).	240
26.1. Der Qt Designer wirkt so, als hätte man das Fensterlayout mit dem Qt Designer erstellt.	242
26.2. Im Titel des Fensters steht nun "Hello World!" – das zählt.	243
26.3. Ja, das ist Comic Sans.	244

26.4. Keine Sorge, auch die gewählte Schriftart aus dem Designer wird vom Python-Skript passend angezeigt.	246
26.5. So sollen die Widgets später im Gitter platziert werden.	248
27.1. Location.	254
27.2. Project.	255
27.3. Import.	255
27.4. Packages.	255
27.5. Unicorn.	256
27.6. requirements.txt	256
27.7. Aufbau des Projekts	257
27.8. Aufbau des Projekts	258
27.9. ‘Hello World’ in JSON-Manier	258
27.10 Datei <code>test_ix_fastapi_hello.py</code>	259
27.11 ‘Run ,Unitests in test_ix...’	260
27.12 Run	260
27.13 Klicken in einem Browser	261
27.14 Datei <code>ix_fastapi.py</code>	261
27.15 ie <code>Item</code> und <code>Message</code>	262
27.16 Listeneinträge zurückgeben	262
27.17 Das Antwortmodell besteht in diesem Fall aus einer Liste der Items.	262
27.18 zur Bestätigung zurückgeschickt	263
27.19 Name neu gesetzt und danach zurückgegeben	263
27.20 Schnittstellen unabhängig von den anderen pflegen	263
27.21 Router	264
27.22 einzelnen <code>count()</code> -Aufrufe	264
27.23 Testclients	264
27.24 Liste nun drei Einträge enthält.	265
27.25 ‘Run, ‘Unitests in test_ix...’’	265
27.26 Python tests	265
27.27 Jetzt starten Sie die Tests wie zuvor und sehen sehen folgendes Ergebnis	266
27.28 OpenAPI für den Item-Lebenszyklus	266
27.29 Nachdem Sie den Request ausgelöst haben, sehen Sie folgendes in Pycharm	267
29.1. Lift and Shift	272
29.2. Bereitstellungsmodelle für Cloud-DBMS	273
29.3. Open-Source-DBMS	274
29.4. Umzug mit Big Bang	275
29.5. Schrittweise Migration zur Risikosenkung – Frontend	276
29.6. Schrittweise Migration zur Risikosenkung – Datenbankspeicher on Premises	276
29.7. Schrittweise Migration zur Risikosenkung – Vollständig	277
30.1. AWS Aurora	281
30.2. AWS RDS	281
30.3. Azure	284
30.4. Cosmos DB	285
30.5. Google GCP	286
30.6. SQL-Datenbanken aus der Cloud – Angebote der Hyperscaler	289
30.7. Key-Value Stores aus der Cloud – Angebote der Hyperscaler	290
31.1. YugabyteDB	293
31.2. Citus	295
31.3. Vitess	297
31.4. TiDB	298

31.5. CockroachDB	300
33.1. Step 1: Analytical problem framing	305
33.2. Step 2: Data exploration and requirements	306
33.3. Step 3: Choosing a model	307
33.4. Example of achieving stationarity in stock prices dataset by calculating difference between observations. Figure inspired by: https://otexts.com/fpp2/stationarity.html	307
33.5. Example of seasonal decomposition of the observed data. Figure inspired by: https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/	
34.1. Photo by Jacopo Maia on Unsplash	312
34.2. Screenshot of modified image by Author	313
35.1. In der Umgebung von “Gefrorener See” muss der Agent den See überqueren, ohne in ein Loch zu fallen.	318
35.2. Eine grafische Repräsentation der Q-Tabelle beim gefrorenen See. F1-F16 stehen für die 16 Felder, auf denen sich der Agent befinden kann. Laut dieser Q-Tabelle ist es im Zustand F1 vielversprechender nach unten zu gehen, als nach links.	320
35.3. Nach dem Training ist der Agent in der Lage, die Umgebung innerhalb von sechs Zügen zu lösen.	325
36.1. Where can this landscape be found? In Germany? Or in France?	328
36.2. NVIDIA’s StyleGAN project generate images of non-existent people.	328
36.3. University of Berkeley’s CycleGAN transfers image style to new images	329
36.4. The loss functions control the learning of the neural networks in GANs.	330
36.5. The mathematical representation of a loss function consists of two sums.	330
36.6. GANs involve two networks competing against each other, unlike simpler neural networks	331
36.7. Python GAN learns to generate handwritten digits	332
37.1. Beispiel einer Grafana-Anwendung	338
37.2. Bevor es ans Einrichten von Diagrammen geht, muss eine Datenquelle eingerichtet sein, zum Beispiel eine InfluxDB-Zeitreihendatenbank.	340
37.3. Beim Zusammenbau der Datenabfrage unterstützt Grafana mit einem Baukasten.	342
37.4. Mehrere Grafana-Dashboards lassen sich zu Playlisten zusammenstellen. Diese kann Grafana zeitgesteuert abspielen.	342
37.5. Canvas Panel: In Grafana 9.4 lassen sich Pfeile verwenden, um Elemente zu verbinden. (Bild: Grafana Labs)	344
37.6. Das Grafana-Dashboard erscheint als Preview-Feature in einem Redesign. (Bild: Grafana Labs)	344
39.1. Die Bibliothek Matplotlib unterstützt Programmierer bei der Visualisierung von Daten.	348
39.2. So präsentiert sich das erste Experiment mit Matplotlib auf dem Bildschirm.	349
39.3. Diagramme lassen sich auf Wunsch auch mit Titel versehen.	350
39.4. Eine Legende macht das Diagramm leichter verständlich.	351
39.5. Ein mit Matplotlib automatisch generiertes Tortendiagramm.	352
39.6. Das gerenderte Balkendiagramm ist einsatzbereit.	353
39.7. Matplotlib dreht Balkendiagramme auf Wunsch.	354
39.8. Ein Gitter im Hintergrund des Diagramms verbessert die Übersichtlichkeit deutlich.	354

39.9. Mit dem Aufruf von <code>plt.grid(True)</code> kann der Benutzer die Werte einfacher begreifen.	355
39.10Wer die xticks-Methode benutzen möchte, muss mitdenken.	356

Acronyms

ALPDR Automatic License Plate Detection and Recognition

API Application Programming Interface

CIFAR Canadian Institute For Advanced Research

COCO Common Objects in Context

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

FP16 Floating Point 16-bit

FP32 Floating Point 32-bit

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

INT8 Integer 8-bit

IoT Internet of Things

ResNet Residual Neural Network

RLE Run-Length Encoding

SD-1 NIST Special Database 1

SD-3 NIST Special Database 3

VPU Video Processing Unit

1. To Do

Part I.

Overview

2. Frameworks and Libraries

As a programming language for data science, Python represents a compromise between the R language, which focuses on data analysis and visualisation, and Java, which forms the backbone of many large-scale applications. This flexibility means that Python can act as a single tool that brings your entire workflow together.

Python is often the first choice for developers who need to apply statistical techniques or data analysis to their work, or for data scientists whose tasks need to be integrated into web applications or production environments. Python particularly shines in the area of machine learning. The combination of machine learning libraries and flexibility makes Python unique. Python is best suited for developing sophisticated models and predictive modules that can be directly integrated into production systems.

One of Python's greatest strengths is its extensive library. Libraries are sets of routines and functions written in a specific language. A robust set of libraries can make the job of developers immensely easier to perform complex tasks without having to rewrite many lines of code.

2.1. Development Environment

2.1.1. Jupyter-Notebook

The Jupyter Notebook application can be used to create a file in a web browser. The advantage here is that programmes can be divided into smaller programme sections and executed section by section. This makes working very interactive. The notebook also offers the possibility of annotating Python programme codes with additional text annotations. [BS19]

2.2. Frameworks

2.2.1. TensorFlow

The TensorFlow framework, which is supported by Google, is the undisputed top dog. It has the most GitHub activity, Google searches, Medium articles, books on Amazon and ArXiv articles. It also has the most developers using it, and is listed in the most online job descriptions. [Goo19]

Variant also exist for TensorFlow that is specific to a hardware. For example, NVIDIA graphics cards are addressed by a variant with Compute Unified Device Architecture (CUDA) and Intel also has an optimised variant. However, you may have to do without the latest version.

TensorFlow Lite, the small device version, brings model execution to a variety of devices, including mobile devices and IoT, and provides more than a 3x increase in inference speed compared to the original TensorFlow.

TensorFlow uses data flow graphs to represent computation, shared state, and the operations that mutate that state. By fully representing the dependencies of each step of the computation, parts of the computation can be reliably parallelised [Aba+16]. Originally, TensorFlow was to be used for Google's internal use, but was released in November 2015 under the Apache 2.0 open source licence. Since then, TensorFlow has brought together a variety of tools, libraries and communities. TensorFlow provides low-level interfaces for programming languages such as Python, Java, C++ and Go.

The mid-level and high-level APIs provide functions for creating, training, saving and loading models, training, saving and loading models. [Cho18]

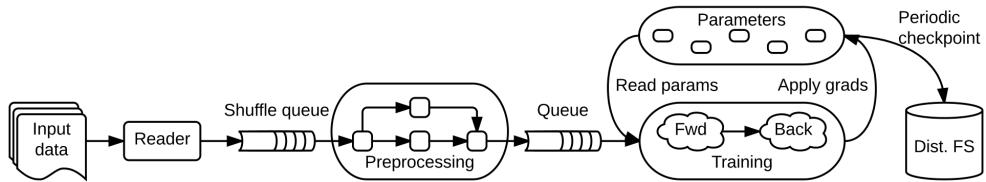


Figure 2.1.: Tensorflow Datenflussdiagramm [Aba+16]

2.2.2. TensorFlow Lite

Tensorflow Lite is an optimised environment for TensorFlow models for mobile devices and the Internet of Things (IoT). Essentially, it consists of two components: the converter, which converts pre-trained TensorFlow models into optimised TensorFlow Lite models, and the interpreter, which enables optimised execution on the various end devices. The goal here is to reduce the size of the existing models and to reduce latency on less powerful devices [Goo20].

2.2.3. TensorRT

TensorRT is a neural network runtime environment based on CUDA, which specialises in optimising the performance of neural networks [NVI15]. This is achieved by reducing the computational accuracy from Floating Point 32-bit (FP32) to Floating Point 16-bit (FP16) resp. Integer 8-bit (INT8). Since the remaining interference accuracy is sufficient for most use cases, the speed can be significantly increased with this method [GMG16]. When optimising the network, the operators used are replaced by TensorRT operators, which can only be executed within a TensorRT environment.

2.2.4. Keras

Keras is a high-level Application Programming Interface (API) application programming interface (API) for neural networks built in Python and running on TensorFlow, Theano or CNTK. It enables the definition and training of different deep learning models using optimised Tensor libraries, which serve as a backend engine. The TensorFlow backend, Theano backend and CNTK backend implementations are currently supported. Any code written in Keras can be run on the backends without customisation. Keras offers a choice of predefined layers, optimisation functions or other important neural network components. The functions can also be extended with custom modules. The two most important model types provided by Keras are sequential and functional api. With sequential, straight-line models can be created whose layers are lined up one after the other. For more complex network structures with feedback, there is the functional api. [Cho18; SIG20]

2.2.5. PyTorch

PyTorch, which is supported by Facebook, is the third most popular framework. It is younger than TensorFlow and has rapidly gained popularity. It allows customisation that TensorFlow does not. [Fac20]

2.2.6. Caffe

Caffe has been around for almost five years. It is in relatively high demand by employers and frequently mentioned in academic articles, but has had little recent coverage of its use. [JES20]

2.2.7. Caffe2

Caffe2 is another open source product from Facebook. It builds on Caffe and is now in the PyTorch GitHub repository. [Sou20]

2.2.8. Theano

Theano was developed at the University of Montreal in 2007 and is the oldest major Python framework. It has lost much of its popularity and its leader stated that major versions are no longer on the roadmap. However, updates continue to be made. [AR+16]

Theano uses a NumPy-like syntax to optimise and evaluate mathematical expressions. What makes Theano different is that it uses the Graphics Processing Unit (GPU) of the computer's graphics card. The speed thus makes Theano interesting.

2.2.9. Apache MXNet

MXNET is supported by Apache and used by Amazon. [MXN20]

2.2.10. CNTK

CNTK is the Microsoft Cognitive Toolkit. It reminds me of many other Microsoft products in the sense that it tries to compete with Google and Facebook offerings and fails to gain significant acceptance. [Mic20]

2.2.11. DeepLearning4J

DeepLearning4J, also called DL4J, is used with the Java language. It is the only semi-popular framework that is not available in Python. However, you can import models written with Keras into DL4J. The framework has a connection to Apache Spark and Hadoop. [Ecl20]

2.2.12. Chainer

Chainer is a framework developed by the Japanese company Preferred Networks. It has a small following. [PN20]

2.2.13. FastAI

FastAI is built on PyTorch. Its API was inspired by Keras and requires even less code for strong results. Jeremy Howard, the driving force behind Fast.AI, was a top Kaggle and president of Kaggle. [fas20]

Fast.AI is not yet in demand for careers, nor is it widely used. However, it has a large built-in pipeline of users through its popular free online courses. It is also both powerful and easy to use. Its uptake could grow significantly.

2.3. General libraries

These are the basic libraries that transform Python from a general-purpose programming language into a powerful and robust tool for data analysis and visualisation. They are sometimes referred to as the SciPy stack and form the basis for the special tools.

2.3.1. NumPy

NumPy is the fundamental library for scientific computing in Python, and many of the libraries in this list use NumPy arrays as their basic inputs and outputs. In short, NumPy introduces objects for multidimensional arrays and matrices, and routines that allow developers to perform advanced mathematical and statistical functions on these arrays with as little code as possible. [Fou20b]

2.3.2. SciPy

SciPy builds on NumPy by adding a collection of algorithms and high-level commands for manipulating and visualising data. The package also includes functions for numerically calculating integrals, solving differential equations, optimisation and more.

2.3.3. pandas

Pandas adds data structures and tools designed for practical data analysis in finance, statistics, social sciences and engineering. Pandas is well suited for incomplete, messy and unlabelled data (i.e. For the kind of data you are likely to face in the real world) and provides tools for shaping, merging, reshaping and splitting datasets.

2.3.4. IPython

IPython extends the functionality of Python's interactive interpreter with a souped-up interactive shell that adds introspection, rich media, shell syntax, tab completion and command archive retrieval. It also acts as an integrated interpreter for your programs, which can be particularly useful for debugging. If you have ever used Mathematica or MATLAB, you will feel at home with IPython.

2.3.5. Matplotlib

Matplotlib is the standard Python library for creating 2D diagrams and plots. The API is quite low-level, i.e. it requires several commands to produce good-looking graphs and figures compared to some more advanced libraries. However, the advantage is greater flexibility. With enough commands, you can create almost any graph with matplotlib.

2.3.6. scikit-learn

scikit-learn builds on NumPy and SciPy by adding a set of algorithms for general machine learning and data mining tasks, including clustering, regression and classification. As a library, scikit-learn has much to offer. Its tools are well documented and among the contributing developers are many machine learning experts. Moreover, it is a very helpful library for developers who do not want to choose between different versions of the same algorithm. Its power and ease of use make the library very popular.

2.3.7. Scrapy

Scrapy is a library for creating spiderbots to systematically crawl the web and extract structured data such as prices, contact information and URLs. Originally developed for web scraping, Scrapy can also extract data from APIs.

2.3.8. NLTK

NLTK stands for Natural Language Toolkit and provides an effective introduction to Natural Language Processing (NLP) or text mining with Python. The basic functions of NLTK allow you to mark up text, identify named entities and display parse trees that reveal parts of speech and dependencies like sentence diagrams. This gives you the ability to do more complicated things like sentiment analysis or generate automatic text summaries.

2.3.9. Pattern

Pattern combines the functionality of Scrapy and NLTK into a comprehensive library that aims to serve as an out-of-the-box solution for web mining, NLP, machine learning and network analysis. Its tools include a web crawler; APIs for Google, Twitter and Wikipedia; and text analytics algorithms such as parse trees and sentiment analysis that can be run with just a few lines of code.

2.3.10. Seaborn

Seaborn is a popular visualisation library built on top of matplotlib. With Seaborn, graphically very high quality plots such as heat maps, time series and violin plots can be generated.

2.3.11. Bokeh

Bokeh allows the creation of interactive, zoomable plots in modern web browsers using JavaScript widgets. Another nice feature of Bokeh is that it comes with three levels of user interface, from high-level abstractions that let you quickly create complex plots to a low-level view that provides maximum flexibility for app developers.

2.3.12. basemap

Basemap supports adding simple maps to matplotlib by taking coordinates from matplotlib and applying them to more than 25 different projections. The library Folium builds on Basemap and allows the creation of interactive web maps, similar to the JavaScript widgets of Bokeh.

2.3.13. NetworkX

This library allows you to create and analyse graphs and networks. It is designed to work with both standard and non-standard data formats, making it particularly efficient and scalable. With these features, NetworkX is particularly well suited for analysing complex social networks.

2.3.14. LightGBM

Gradient Boosting is one of the best and most popular libraries for Machine Learning and helps developers to create new algorithms by using newly defined elementary models and especially decision trees.

Accordingly, there are special libraries designed for a fast and efficient implementation of this method. These are LightGBM, XGBoost and CatBoost. All these libraries are competitors but help solve a common problem and can be used in almost similar ways.

2.3.15. Eli5

Most of the time, model prediction results in machine learning are not really accurate. Eli5, a library built into Python, helps overcome this very challenge. It is a combination of visualising and debugging all machine learning models and tracking all steps of the algorithm.

2.3.16. Mlpy - Machine Learning

As an alternative to scikit-learn, Mlpy also offers a powerful library of functions for machine learning. Mlpy is also based on NumPy and SciPy, but extends the functionality with supervised and unsupervised machine learning methods.

2.3.17. Statsmodels - Statistical Data Analysis

Statsmodels is a Python module that allows users to explore data, estimate statistical models and run statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions and result statistics is available for various data types and each estimator. The module makes predictive analytics possible. Statsmodels is often combined with NumPy, matplotlib and Pandas.

2.4. Specialised Libraries

2.4.1. OpenCV

OpenCV, derived from Open Computer Vision, is a free program library with algorithms for image processing and computer vision. The development of the library was initiated by Intel and was maintained by Willow Garage until 2013. After their dissolution, it was continued by Itseez, which has since been acquired by Intel. [Tea20a]

2.4.2. OpenVINO

The OpenVINO toolkit helps accelerate the development of powerful computer vision and deep learning in vision applications. It enables Deep Learning via hardware accelerators as well as simple, heterogeneous execution on Intel platforms, including accpu, GPU, Field-Programmable Gate Array (FPGA) and Video Processing Unit (VPU). Key components include optimised features for OpenCV. [Int19; Tea20b]

2.4.3. Compute Unified Device Architecture (CUDA)

The abbreviation CUDA stands for Compute Unified Device Architecture. It is an interface technology and computing platform that allows graphics processors to be addressed and used for non-graphics-specific computations. CUDA was developed by NVIDIA and accelerates programmes by parallelising certain parts of the programme with one or more GPUs in addition to the Central Processing Unit (CPU). [TS19; Cor20; NVI20]

2.4.4. OpenNN

OpenNN is a software library written in C++ for advanced analysis. It implements neural networks. This library is characterised by its execution speed and memory allocation. It is constantly optimised and parallelised to maximise its efficiency. [AIT20]

2.5. Special Libraries

When programming the neural network, some predefined libraries are accessed. are used. In the following, some interesting libraries are presented.

2.5.1. glob

The glob module finds all pathnames in a given directory which match a given pattern and returns them in an arbitrary way. [Fou20a]

2.5.2. os

The os module provides general operating system functionalities. The functions of this library allow programmes to be designed in a platform-independent way. platform-independent. The abbreviation os stands for Operating System. It enables access to and reference to certain paths and the files stored there. [Bal19]

2.5.3. PIL

The abbreviation PIL stands for Python Image Library and provides many functions for image processing. image processing. It provides fast data access to the basic image formats. image formats is guaranteed. In addition to image archiving and batch function applications file formats, create thumbnails, print images and perform many other functions. many other functions can be performed.

2.5.4. LCC15

2.5.5. math

As can be easily inferred from the name, this library provides access to all mathematical functions defined in the C standard. These include among others trigonometric functions, power and logarithm functions and angle functions. tions. Complex numbers are not included in this library. [Fou20b]

2.5.6. cv2

With the cv2 module, input images can be rewritten into three-dimensional arrays. OpenCV contains algorithms for image processing and machine vision. The algorithms are based on the latest research results and are continuously being further developed. The modules from image processing include, for example, face recognition, as well as many fast filters and functions for camera calibration. The machine vision modules include boosting (automatic classification), decision tree learning and artificial neural networks. [Wik20]

2.5.7. random

The Random module implements pseudo-random number generators for different distributions. distributions. For example, random numbers can be generated or a mixture of elements can be created. of elements can be generated. [Fou20c]

2.5.8. pickle

The pickle module implements binary protocols for serialising and deserialising a Python object structure. of a Python object structure. Object hierarchies can be converted in a binary stream (pickle) and then (pickle) and then be returned from a binary architecture back to the object hierarchical structure (unpickle). structure (unpickle). [Fou20d]

2.5.9. PyPI

The Python Software Foundation organisation organises and manages various packages for Python programming. [Fou21a] Here you can find packages for various tasks. An example is the official API for accessing the dataset Common Objects in Context (COCO). [Fou21b]

3. Libraries, Modules, and Frameworks

Libraries, Module, and Framework have an inevitable role in Machine learning and computer vision field. They are written on a specific topic, which have build-in functions and classes who help the developer and data scientist to run usefull application. Some usefull libraries, modules, and framework use in Machine learning (ML) and Computer Vision(CS) are:

- Tensor Flow
- MediaPipe
- OpenCV
- Pyfirmata

3.1. TensorFlow Framework

TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments. It uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). TensorFlow enables developers to experiment with novel optimizations and training algorithms. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks. Several Google services use TensorFlow in production, we have released it as an open-source project, and it has become widely used for machine learning research. In this paper, we describe the TensorFlow dataflow model and demonstrate the compelling performance that TensorFlow achieves for several real-world applications.

3.1.1. TensorFlow Use Case

Tensor flow is the framework made by google for making the Machine learning application and training the model. Google search engine also based on Tensor flow AI application, e.g; when a user type "a" in the google search bar, the search engine predict the most suitable complete word to select, or it may be predicts depends upon the previous search on the same system by the user too. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensor flow has a various collection of workflows to develop and train the model using Python and Javascripts programming languages, after training the model we can deploy the model on the Cloud or also on any device for making edge computing application no matter which language use in the device. It is obvious that, the most important part in every Machine learning application is to train the ML model, so the model will apply in real time condition and take the decision without any human intervention as the human normally make.

3.2. OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing. It plays a major role in real-time operation which is very important in today's systems. Computer Vision is the science of programming a computer to process and ultimately understand images and video, or simply saying making a computer see. Solving even small parts of certain Computer Vision challenges, creates exciting new possibilities in technology, engineering and even entertainment. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When we create applications for computer vision that we don't want to build from scratch we can use this library to start focusing on real world problems. There are many companies using this library today such as Google, Amazon, Microsoft and Toyota. OpenCV.

3.2.1. Application of OpenCV

There are lots of applications which are solved using OpenCV, some of them are listed below.

- Face recognition
- Automated inspection and surveillance
- Number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anomaly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- Object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

3.2.2. Image Processing

Image Processing is one of the basic functionality of OpenCV, it is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it.

3.2.3. How does a computer read an image

We are humans we can easily make it out that is the image of a person who is me. But if we ask computer "is it my photo?". The computer can't say anything because the computer is not figuring out it all on its own. [Image Processing] The computer reads any image as a range of values between 0 and 255. For any color image, there are 3 primary channels -red, green and blue.

3.3. MediaPipe

MediaPipe is one of the most widely shared and re-usable libraries for media processing within Google. Google open-source MediaPipe was first introduced in June, 2019. It aims to provide some integrated computer vision and machine learning features. It has some build in module who can detect the Human motion by tracking the different part of the body. This Github link [MediaPipe Github] shows all the available application the MediaPipe Module able to perform. It is best suited for gesture detection for edge computing application.

3.4. MediaPipe Applications

MediaPipe has numerous application in Machine learning and computer vision. It is a python supportive library and perform well for edge devices too. The module is train on 30,000 images, so its accuracy and precision is very high. The reason for using MediaPipe is that, it work well for independent of background, because it detects just the Landmarks of different part of body. The remaining things in the frame is invisible for the MediaPipe Module. The Following most important use cases of MediaPipe are:

- Hand Landmarks
- Face Detection
- Object Detection
- Holistic

3.4.1. Hand Landmarks Detection

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. Every hand there are 21 landmarks, each finger has 4 and there is landmark in the middle of hand. For gesture detection we can use this technique to make the different types of gesture. We are able to make different types of gestures by changing the position of landmarks. MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. Fig3.1 shows all the Hand Landmarks of a hand, which can help us to make different gestures. [Hand Landmarks]

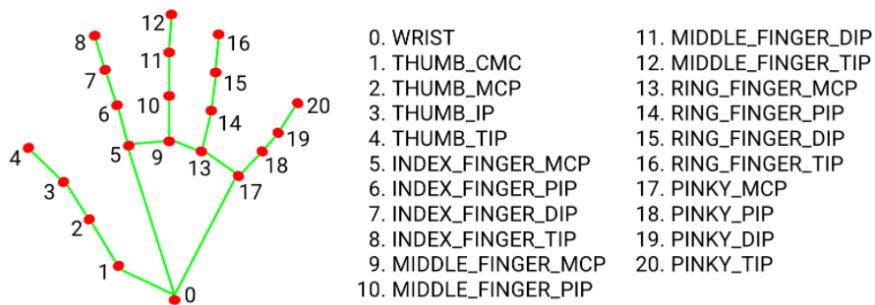


Figure 3.1.: Hand Landmarks

3.4.2. Face Detection

MediaPipe Face Detection is an ultrafast face detection solution that can detect 6 landmarks on the face and also support multiple faces. Fig3.2 shows the detection of multiple faces using landmarks. The detector's super-realtime performance enables it to be applied to any live viewfinder experience that requires an accurate facial region of interest as an input for other task-specific models, such as 3D facial keypoint or geometry estimation (e.g., MediaPipe Face Mesh), facial features or expression classification, and face region segmentation. Face Detection

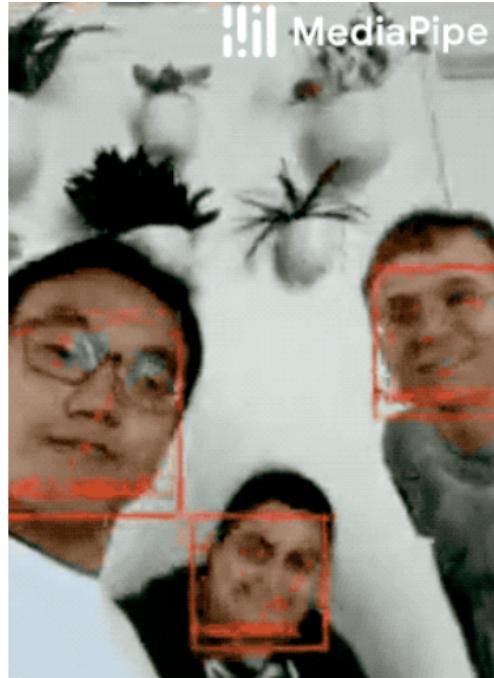


Figure 3.2.: Face Detection Using Landmarks

3.4.3. MediaPipe Holistic

The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. Fig3.3 shows the detection of face, hand, and pose:



Figure 3.3.: Face, Pose, and Hand Landmarks

MediaPipe holistic offers fast and accurate, yet separate, solutions for these tasks. Combining them all in real-time into a semantically consistent end-to-end solution is a uniquely difficult problem requiring simultaneous inference of multiple, dependent neural networks.[Holistic]

3.5. Pyfirmata

The Firmata library implements the Firmata protocol for communicating with software on the host computer. This allows you to write custom firmware without having to create your own protocol and objects for the programming environment that you are using. PyFirmata is basically a prebuilt library package of python program which can be installed in Arduino to allow serial communication between a python script on any computer and with Arduino.

- Firmata Serial communication Protocol work as a mediator between two systems as follow.
 - Firmata is used as an intermediate protocol that connects an embedded system to a host computer.
 - Using Pyfirmata, the program will run on host computer and get the certain output on embedded Devices.

Some of the packages we need to implement this project are only supported by python environment, the edge computer we are working with Arduino Nano 33 BLE Sense have Arduino (IDE), it is the c++ integrated development environment. For running the host computer python program, we need to install Firmata library. It helps serial communication protocol and work as a bridge between two environments c++ and python. Fig3.4 shows the two embedded devices, the one raspberry pi support python program and arduino support c++. For running the python program on host computer, we can see the result on c++ embedded device arduino.

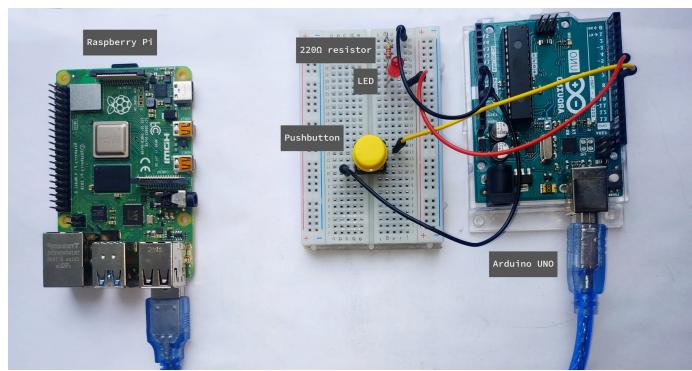


Figure 3.4.: Serial Communication protocol using Pyfirmata

3.6. Supporting Libraries

Libraries play a vital role in machine learning and computer vision for training a system. The most important libraries which help us to make mathematical calculation, matrices multiplication and data visualization and csv files are:

- Pandas
- Numpy

- Matplotlib
- Scikit-learn

3.6.1. Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

3.6.2. Numpy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

3.6.3. Matplotlib

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib’s APIs (Application Programming Interfaces) to embed plots in GUI applications.

3.6.4. Scikit-learn

Scikit-learn is arguably the most important library in Python for machine learning. After cleaning and manipulating your data with Pandas or NumPy, scikit-learn is used to build machine learning models as it has tons of tools used for predictive modelling and analysis. There are many reasons to use scikit-learn.

3.7. Important Libraries for Arduino Nano 33 BLE Sense

The Arduino integrated development environment (IDE) comes with a set of standard libraries for commonly used functionality. By installing these libraries it make us in our code more funcnality. These are the following set of libraries which are most commonly suited for our edge computing application.

3.7.1. Tensor flow lite

For training the ML Algorithm we will install the Tenson flow lite library on Arduino Nano 33 BLE sense with the help of Integrated Development Environment (IDE) software. It is flexile integrated environment who helps both the software and hardware related to arduino. TensorFlow is an open source library for numerical computation and large-scale machine learning. It make ease in the process of acquiring data, training models, serving predictions, and refining future results.By having this tensor flow lite library, we can deploy or train the ML model.

3.7.2. ArduCAM

This is a opensource library for taking high resolution images and short video clip on Arduino based platforms using ArduCAM OV2640 Mini 2MP camera module. For deploying this library on arduino and arducam is available at <https://github.com/ArduCAM/Arduino>. We need to select the desire hardware setting about our edge computer and available camera module in the (.h) file.

3.7.3. ADPS 9960

It is the build in Arduino Library for making use of Gesture, Proximity and RGB color detection sensor.

4. Datenbanken und Modelle für das Maschinelle Lernen

Daten sind die Grundlage für das Maschinelle Lernen. Der Erfolg der Modelle sind abhängig von ihrer Qualität. Denn Maschinelles Lernen beruht auf genauen und zuverlässigen Informationen im Training seiner Algorithmen. Diese Selbstverständlichkeit ist bekannt, wird leider nicht genügend berücksichtigt. Schlechte Daten führen zu ungenügenden oder zu falschen Ergebnissen.

Das ist eigentlich selbstverständlich, wird aber oft übersehen. Realistisch sind die Trainingsdaten dann, wenn sie die Daten widerspiegeln die das KI-System im echten Einsatz aufnimmt. Unrealistische Datensätze behindern das Maschinelle Lernen und führen zu teuren Fehlinterpretationen. Falls man eine Software für Drohnenkameras entwickeln möchte, so müssen auch realistische Bilder verwendet. Greift man in einem solchen Fall auf entsprechende Bilder aus dem Netz zurück, weisen sie in der Regel folgende Eigenschaften auf:

- Die Perspektive ist eher die Kopfhöhe.
- Das anvisierte Objekt befindet sich im Zentrum.

Falls man Datensätze für den eigenen Bedarf verwenden möchte, so muss darauf geachtet werden, dass nur Daten verwendet werden, die auch realistisch sind. Die Datensätze dürfen auch keine Ausreißer oder Redundanzen. Bei der Überprüfung der Qualität der Daten können folgende Fragen hilfreich sein:

- Mit welchen Mitteln und welcher Technik wurden die Daten generiert?
- Ist die Datenquelle glaubwürdig?
- Mit welcher Absicht wurden die Daten erhoben?
- Woher kommen die Daten? Sind sie für die anvisierte Anwendung geeignet?
- Wie alt sind die Daten?
- In welcher Umgebung/unter welchen Bedingungen wurden die Daten erstellt?

Gegebenenfalls sind eigene Daten zu erheben oder erheben zu lassen.

Jeder, der Data Science betreibt, kann seine entwickelten Algorithmen mit den Ergebnissen andere messen, in dem man standardisierte Datensätze verwendet. Dazu stehen stehen sehr viele Datenbanken und vortrainierte Modelle im Internet zur Verfügung. In diesem Kapitel werden einige beschrieben. Es ist zu beachten, dass viele Datensätze in verschiedenen Varianten zur Verfügung stehen. Je nach Anbieter wurden die Daten schon bearbeitet und für ein Training vorbereitet. Hier ist auf eine geeignete Variante zu achten. Zugriff auf verschiedene Datensätze erhält auf folgende Seiten des Internets:

Kaggle.com: Hier werden über 20.000 Datensätze angeboten. Dazu ist nur ein kostenloses Benutzerkonto notwendig.

lionbridge.ai: Die Website bietet eine gute Übersicht zu Datensätzen aus dem öffentlichen und kommerziellen Bereich.

```

1000 # Construct a tf.data.Dataset
1001 ds = tfds.load('mnist', split='train', shuffle_files=True)
1002
1003 # Build your input pipeline
1004 ds = ds.shuffle(1024).batch(32).prefetch(tf.data.experimental.AUTOTUNE)
1005 for example in ds.take(1):
1006     image, label = example["image"], example["label"]

```

Listing 4.1.: Laden eines datensatzes mit TensorFlow

govdata.de: Das Datenportal für Deutschland bietet frei verfügbare Daten aus allen Bereichen der öffentlichen Verwaltung in Deutschland an.

govdata.de: Das Datenportal für Deutschland bietet frei verfügbare Daten aus allen Bereichen der öffentlichen Verwaltung in Deutschland an.

Datenbank der amerikanischen Regierung: Auch die amerikanische Regierung betreibt ein Portal, wo Datensätze der Verwaltung zur Verfügung stehen.

scikit-Datensätze: Mit der Python-Bibliothek werden auch Datensätze installiert. Es sind zwar nur wenige Datensätze, diese sind aber schon vorarbeitet, so dass sie einfach zu laden und zu verwenden sind.

UCI - Center for Machine Learning and Intelligent System: Die Universität von Irvine in Kalifornien bietet rund 600 Datensätze zur eigenen Untersuchung an.

TensorFlow: TensorFlow-Datensätze: Eine Sammlung liefert gebrauchsfertiger Datensätze mit. Alle Datasätze werden über die Struktur `alstf.data.Datasets` beziehungsweise `wastf.data.Datasets` zur Verfügung gestellt. Die Datensätze können auch über GitHub einzeln abgerufen werden.

Open Science Data Cloud: Die Plattform möchte allen eine Möglichkeit schaffen, auf qualitativ hochwertige Daten zuzugreifen. Forscher können ihre eigenen wissenschaftlichen Daten unterbringen und gemeinsam nutzen, auf ergänzende öffentliche Datensätze zugreifen, angepasste virtuelle Maschinen mit den für die Analyse ihrer Daten erforderlichen Tools erstellen und gemeinsam nutzen.

Amazon: Auch Amazon stellt Datensätze zur Verfügung. Hierzu muss man sich kostenlos registrieren.

KDnuggets.com: Ähnlich wie Kaggle aufgebaut; allerdings wird hier andere Webseiten verwiesen.

paperswithcode.com: Die Plattform stellt eine Möglichkeit zur Verfügung, Datensätze auszutauschen. Hier finden sich auch viele bekannte Datensätze mit ihren Links.

Google Dataset Search: Die Website bietet nicht direkt Datensätze an, sondern eine Unterstützung bei der Suche. Google schränkt seine Suchmaschine hier auf Datensätze ein.

4.1. Datenbanken

4.1.1. Datensatz MNIST

Der Datensatz MNIST ist zum freien Gebrauch verfügbar und enthält 70.000 Bilder von handgeschriebenen Ziffern mit den entsprechenden korrekten Klassifikation. [Den+09;

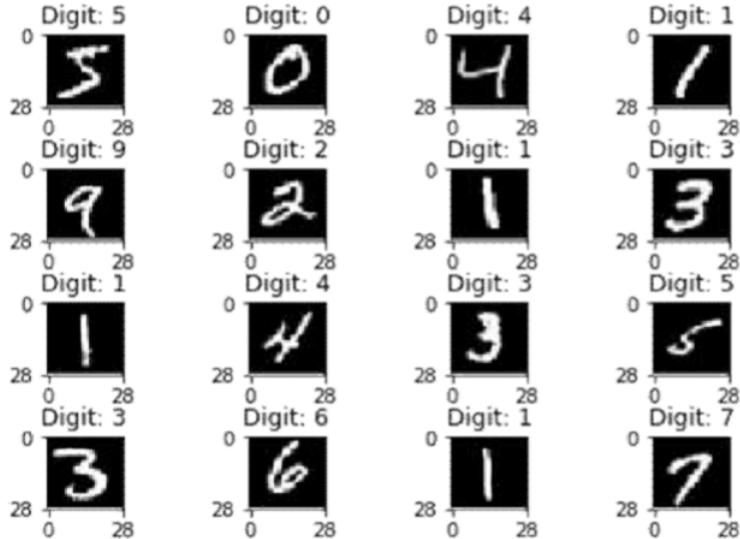


Figure 4.1.: Beispiele aus dem Trainingssatz des Datensatzes MNIST [SSS19]

[Den12] Der Name des Datensatzes begründet aus der Herkunft, da es sich um ein modifizierte Datensatz aus zwei Datensätzen des US National Institute of Standards and Technology handelt. Diese enthalten handgeschriebene Ziffern von 250 verschiedenen Personen, bestehend aus Angestellte des US Census Bureau und Schülern einer High School. Die so gesammelten Datensätze NIST Special Database 3 (SD-3) und NIST Special Database 1 (SD-1) wurden zusammengelegt, da der erstere Datensatz der Büroangestellten sauberer und leichter zu erkennende Daten enthält. Zunächst wurde SD-3 als Trainingsset und SD-1 als Testset verwendet, doch auf Grund der Unterschiede ist eine Vermengung beider sinnvoller. Der Datensatz ist bereits in 60.000 Trainingsbilder und 10.000 Testbildern aufgeteilt. [LCB13; Nie15]

Die Daten werden in vier Dateien zur Verfügung gestellt, zwei für den Trainingsdatensatz und zwei für den Testdatensatz, wovon eine Datei die Bilddaten und die andere die zugehörigen Labels enthält:

- [train-images-idx3-ubyte.gz](#): Bilder zum Training (9912422 bytes)
- [train-labels-idx1-ubyte.gz](#): Klassifikation der Trainingsdaten(28881 bytes)
- [t10k-images-idx3-ubyte.gz](#): Testbilder (1648877 bytes)
- [t10k-labels-idx1-ubyte.gz](#): Klassifikation der Testbilder (4542 bytes)

Bei den Bildern im Datensatz MNIST handelt es sich um Graustufenbilder der Größe 28×28 Pixel.[LCB13] Die Daten liegen im Dateiformat IDX vor und können so nicht standardmäßig geöffnet und visualisiert werden. Man kann aber ein Programm schreiben, um die Daten in das Format CSV zu überführen oder direkt von anderen Webseiten eine Variante im Format CSV laden. [Red20]

Die Bibliothek TensorFlow stellt den Datensatz MNIST unter [tensorflow_datasets](#) zur Verfügung. Dies ist nicht der einzige Datensatz der von TensorFlow zur Verfügung gestellt wird. Eine Liste aller Datensätze, die über so geladen werden können, findet sich im Katalog der Datensätze.

```

1000     (TRAIN_IMAGES, TRAIN_LABELS), (TEST_IMAGES, TEST_LABELS) = datasets.cifar10.
1001     ↳ load_data()
1002     TRAIN_IMAGES = TRAIN_IMAGES / 255.0
1003     TEST_IMAGES = TEST_IMAGES / 255.0

```

Listing 4.2.: Laden und Normalisieren des Datensatzes CIFAR-10

4.1.2. CIFAR-10 und CIFAR-100

Datensatz Canadian Institute For Advanced Research (CIFAR)-10

Die Datensätze CIFAR-10 und CIFAR-100 sind von Alex Krizhevsky und seinem Team entwickelt und deshalb nach dem **Canadian Institute for Advanced Research** benannt worden.

Der Datensatz CIFAR-10 ist ein Teildatensatz des Datensatzes *80 Million Tiny Images*. Dieser Teil besteht aus 60.000 Bildern, unterteilt in 50.000 Trainingsbilder und 10.000 Testbilder, welche in 10 Klassen unterteilt und entsprechend gelabelt sind. Die vorhandenen Klassen repräsentieren Flugzeuge, Autos, Vögel, Katzen, Rehe, Hunde, Frösche, Pferde, Schiffe und Lastwagen. Pro Klasse existieren somit 6.000 Bilder, wobei jedes Bild eine Größe von 32×32 Pixeln mit drei Farbkanälen hat [KH09; KSH17]. Ein Auszug des Datensatzes ist in Abb. 4.2 zu sehen.



Figure 4.2.: Auszug von zehn Abbildungen des Datensatz CIFAR-10

Um den Datensatz in die Python-Umgebung zu importieren, wird hier auf das Modul `keras.datasets` zurückgegriffen. Dies beinhaltet eine direkte Schnittstelle zum Datensatz CIFAR-10, welcher durch den Aufruf der Funktion `load_data()` auf das lokale System heruntergeladen wird. Im Hintergrund wird der Datensatz als gepackte Datei mit der Endung `.tar.gz` heruntergeladen und im Ordner `C:\Users\<Nutzername>\.keras\datasets` abgelegt. Nach dem Entpacken ist der Datensatz in mehreren serialisierten Objekten enthalten, welche über `pickle` in die Python-Umgebung geladen werden. Daraufhin kann auf die Trainingsdaten, Trainingslabel, Testdaten, und Testlabel über entsprechende Listen zugegriffen werden (Abschnitt 4.1.2). Um bei dem späteren Training das Vanishing-Gradient-Problem zu reduzieren [IK17], werden die im Datensatz abgelegten RGB-Farbwerke mit dem Wertebereich von 9 bis 255 in Werte von 0 bis 1 umgerechnet. Das Listing ?? zeigt das Laden und Normalisieren der Daten.

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figure 4.3.: Oberkategorien und Kategorien in CIFAR-100 mit den Originalbezeichnungen

```

1000 import tensorflow as tf
1001 from tensorflow.keras import datasets
1002
1003 (TRAIN_IMAGES100, TRAIN_LABELS100), (TEST_IMAGES100, TEST_LABELS100) = tf.keras.
1004     datasets.cifar100.load_data()
1005 TRAIN_IMAGES100 = TRAIN_IMAGES100 / 255.0
1006 TEST_IMAGES100 = TEST_IMAGES100 / 255.0

```

Listing 4.3.: Laden und Normalisieren des Datensatzes CIFAR-100

Datensatz CIFAR-100

Der Datensatz CIFAR-100 hingegen enthält zwar ebenfalls 60.000 Bilder, jedoch unterteilt in 100 Kategorien mit je 600 Bildern. Dabei gibt es zusätzlich 20 Oberkategorien, denen jeweils fünf der 100 Kategorien zugeordnet sind; siehe Tabelle 4.3. Der Datensatz kann ebenfalls heruntergeladen und direkt über Keras in TensorFlow importiert werden [Raj20]:

4.1.3. Datensatz Fisher's Iris Data Set

Die Erkennung der Iris ist ein weiterer Klassiker zur Bildklassifizierung. Auch zu diesem Klassiker gibt es mehrere Tutorials. Der Datensatz Fisher's Iris Data Set wurde in R.A. Fishers klassischer Arbeit von 1936 verwendet und kann im UCI Machine Learning Repository gefunden werden. [Fis; Fis36a; SW16] Es wurde vier Merkmale der Blumen Iris Setosa, Iris Versicolour und Iris Virginica vermessen. Für jede der drei Klassen stehen 50 Datensätze mit vier Attributen zur Verfügung. Gemessen wurden dabei jeweils die Breite und die Länge des Kelchblatts (Sepalum) sowie des Kronblatts (Petalum) in Zentimeter. [And35; SP06]. Eine Blumenart ist linear von den anderen beiden trennbar, aber die anderen beiden sind nicht linear voneinander trennbar.

```
1000 from sklearn.datasets import load_iris
1001 iris = load_iris()
1002
```

Listing 4.4.: Laden des Datensatzes Fisher's Iris Data Set

Auf den Datensatz kann an mehreren Stellen zugriffen werden:

- Archiv von Datensätzen für das Maschinelle Lernen der Universität von Kalifornien in Irvine [Fis36b]
- Python Paket `skikitlearn` [Ped+11]
- Kaggle - Website für Maschinelles Lernen [Kag]

Den Datensatz steht auf verschiedenen Webseiten zur Verfügung, allerdings muss auf den Aufbau des Datensatzes geachtet werden. Da es sich um eine Datei im Format CSV handelt, ist der Aufbau spaltenorientiert. In der Regel ist in der ersten Zeile der Titel der einzelnen Spalte angegeben: `sepal_length`, `sepal_width`, `petal_length`, `petal_width` und `species`. Die Werte sind in Zentimeter angegeben, die Spezies ist mit `setosa` für Iris Setosa, `versicolor` für Iris Versicolor und `virginica` für die Spezies Iris Virginica angegeben. Die Spalten in diesem Datensatz sind:

- Laufende Nummer
- Länge des Kelchblatts (Sepal) in Zentimeter
- Breite des Kelchblatts (Sepal) in Zentimeter
- Länge des Blütenblatts (Petal) in Zentimeter
- Breite des Blütenblatts (Petal) in Zentimeter
- Art

Das Ziel ist die Klassifizierung der drei verschiedenen Iris-Arten anhand der Länge und Breite von Kelchblatt und Blütenblatt. Da der Datensatz mit der Bibliothek `sklearn` ausgeliefert wird, wird dieser Zugang gewählt.

Der Datensatz ist ein `dictionary`. Seine Schlüssel kann man sich leicht anzeigen lassen:

```
1000 >>> iris.keys()
```

Die zugehörige Ausgabe ist wie folgt:

```
dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename'])
```

Die einzelnen Elemente kann man sich nun ansehen. Nach Eingabe des Befehls

```
1000 iris['DESCR']
```

wird eine ausführliche Beschreibung ausgegeben:
Mit der Eingabe

```

1000 '.. _iris_dataset:
1002 Iris plants dataset
1004 -----
1006 **Data Set Characteristics:**
1008 :Number of Instances: 150 (50 in each of three classes)
1009 :Number of Attributes: 4 numeric, predictive attributes and the class
1010 :Attribute Information:
1011   - sepal length in cm
1012   - sepal width in cm
1013   - petal length in cm
1014   - petal width in cm
1015   - class:
1016     - Iris-Setosa
1017     - Iris-Versicolour
1018     - Iris-Virginica
1019 :Summary Statistics:
1020 =====
1021 Min Max Mean SD Class Correlation
1022 =====
1023 sepal length:  4.3  7.9  5.84  0.83  0.7826
1024 sepal width:  2.0  4.4  3.05  0.43  -0.4194
1025 petal length: 1.0  6.9  3.76  1.76  0.9490 (high!)
1026 petal width:  0.1  2.5  1.20  0.76  0.9565 (high!)
1027 =====
1028
1029 :Missing Attribute Values: None
1030 :Class Distribution: 33.3% for each of 3 classes.
1031 :Creator: R.A. Fisher
1032 :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
1033 :Date: July, 1988
1034
1035 The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
1036 from Fisher's paper. Note that it's the same as in R, but not as in the UCI\ 
1037   ↪ nMachine Learning Repository, which has two wrong data points.
1038
1039 This is perhaps the best known database to be found in the
1040 pattern recognition literature. Fisher's paper is a classic in the field and
1041 is referenced frequently to this day. (See Duda & Hart, for example.) The
1042 data set contains 3 classes of 50 instances each, where each class refers to a\
1043   ↪ ntype of iris plant. One class is linearly separable from the other 2; the\
1044   ↪ n latter are NOT linearly separable from each other.
1045
1046 .. topic:: References
1047
1048 - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
1049   Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
1050   Mathematical Statistics" (John Wiley, NY, 1950).
1051 - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
1052   (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
1053 - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
1054   Structure and Classification Rule for Recognition in Partially Exposed
1055   Environments". IEEE Transactions on Pattern Analysis and Machine
1056   Intelligence, Vol. PAMI-2, No. 1, 67-71.
1057 - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions
1058   on Information Theory, May 1972, 431-433.
1059 - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II
1060   conceptual clustering system finds 3 classes in the data.
1061 - Many, many more ...'
```

Listing 4.5.: Beschreibung des Datensatzes Fisher's Iris Data Set

```
1000    iris['feature_names']
```

erhält man die Namen der Attribute:

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
--

Die Namen der Blumen, die nach der Eingabe

```
1000    iris['target_names']
```

angezeigt werden, sind

array(['setosa', 'versicolor', 'virginica'], dtype=' <U10')

Zur weiteren Untersuchung werden die Daten mit den Überschriften in einen Datenrahmen aufgenommen.

```
1000    X = pd.DataFrame(data = iris.data, columns = iris.feature_names)
        print(X.head())
```

Der Befehl (`X.head()`) zeigt – wie in Abbildung 4.4 – den Kopf des Datenrahmens an. Ersichtlich ist, dass jeder Datensatz aus vier Werten besteht.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Figure 4.4.: Kopfzeilen des Datensatzes Fisher's Iris Data Set

Jeder Datensatz enthält auch schon in dem Schlüssel `target` seine Klassifikation. In der Abbildung 4.5 ist dies für die ersten Datensätze aufgeführt.

```
1000    y = pd.DataFrame(data=iris.target, columns = ['irisType'])
        y.head()
```

irisType	
0	0
1	0
2	0
3	0
4	0

Figure 4.5.: Ausgabe der Kategorien des Datensatzes Fisher's Iris Data Set

Mittels des Befehls

```
1000    y.irisType.value_counts()
```

kann ermittelt werden, wie viele Klassen vorliegen. Die Ausgabe des Befehls wird in der Abbildung 4.6 gezeigt; es ergeben sich 3 Klassen mit den Nummern 0, 1 und 2. Jeweils 50 Datensätze sind ihnen zugeordnet.

```
2      50
1      50
0      50
Name: irisType, dtype: int64
```

Figure 4.6.: Namen der Kategorien im Datensatz Fisher's Iris Data Set

4.1.4. Common Objects in Context - COCO

Der Datensatz COCO ist beschriftet und liefert Daten zum Trainieren von überwachten Computer-Vision-Modellen, die in der Lage sind, gemeinsamen Objekte im Datensatz zu identifizieren. Natürlich sind diese Modelle noch weit davon entfernt, perfekt zu sein. Daher bietet der Datensatz COCO einen Maßstab für die Bewertung der periodischen Verbesserung dieser Modelle durch die Computer-Vision-Forschung.[Lin+14; Lin+21]

- Objekterkennung
 - Der Datensatz COCO enthält ≈ 330.000 Bilder.
 - Der Datensatz COCO enthält $\approx 1.500.000$ Annotationen für Objekte.
 - Der Datensatz COCO enthält 80 Kategorien.
 - Die Bilder haben jeweils fünf Überschriften.
 - Die Bilder haben eine mittlere Auflösung 640×480 Pixel.
- Semantische Segmentierung
 - Panoptische Segmentierung erfordert Modelle zum Ziehen von Grenzen zwischen Objekten bei der semantischen Segmentierung.
- Erkennung von Schlüsseln

```

1000 {
1002   "info": {...},
1003   "licenses": {...},
1004   "images": {...},
1005   "categories": {...},
1006   "annotations": {...}
}

```

Listing 4.6.: Informationen des Datensatzes COCO

```

1000 {
1001   "description": "COCO 2017 Dataset",
1002   "url": "http://cocodataset.org",
1003   "version": "1.0",
1004   "year": 2017,
1005   "contributor": "COCO Consortium",
1006   "date_created": "2017/09/01"
}

```

Listing 4.7.: Metainformationen des Datensatzes COCO

- Die Bilder enthalten 250.000 Personen, die mit den entsprechenden Schlüssel beschriftet sind.

Aufgrund der Größe und der häufigen Verwendung des Datensatzes gibt es zahlreiche Werkzeuge, zum Beispiel COCO-annotator und COCOapi, um auf die Daten zuzugreifen.

Eigentlich besteht der Datensatz COCO aus mehreren, die jeweils für eine bestimmte Aufgabe des Maschinellen Lernens gemacht wurden. Die erste Aufgabe ist die Bestimmung von umgebenden Rechtecke für Objekte. Das heißt, es werden Objekte identifiziert und die Koordinaten des umgebenden Rechtecke ermittelt. Die erweiterte Aufgabe ist die Objektsegmentierung. Hierbei werden auch Objekte identifiziert, aber zusätzlich anstelle der umgebenden Rechtecke Polygonzüge, die die Objekte eingrenzen. Die dritte klassische Aufgabe ist die Stoffsegmentierung. Das Modell soll eine Objektsegmentierung durchführen, aber nicht auf einzelnen Objekten, sondern auf kontinuierlichen Hintergrundmustern wie Gras oder Himmel.

Die Annotationen sind im Format JSON hinterlegt. Das Format JSON ist ein Wörterbuch mit Schlüssel-Wert-Paare in geschweiften Klammern. Es kann auch Listen, geordnete Sammlungen von Elementen innerhalb von geschweiften Klammern, oder Wörterbücher enthalten, die darin verschachtelt sind.

“info” section

Das Wörterbuch zum Abschnitt **info** enthält Metadaten über den Datensatz. Für den offiziellen Datensatz COCO sind es folgende Informatinen:

Wie man sieht, sind nur grundlegende Informationen enthalten, wobei der Wert url auf die offizielle Website des Datensatzes verweist. Dies ist bei Datensätzen für Maschinelles Lernen üblich, um auf ihre Websites für zusätzliche Informationen zu verweisen. Dort kann man zum Beispiel Informationen wie und wann die Daten erfasst wurden.

Im Abschnitt licenses finden Sie Links zu Lizzenzen für Bilder im Datensatz mit der folgenden Struktur:

```

1000 [
1001   {
1002     "url": "http://creativecommons.org/licenses/by-nc-sa/2.0/",
1003     "id": 1,
1004     "name": "Attribution-NonCommercial-ShareAlike License"
1005   },
1006   {
1007     "url": "http://creativecommons.org/licenses/by-nc/2.0/",
1008     "id": 2,
1009     "name": "Attribution-NonCommercial License"
1010   },
1011   ...
1012 ]

```

Listing 4.8.: Lizenzinformationen des Datensatzes COCO

```

1000 {
1001   "license": 3,
1002   "file_name": "000000391895.jpg",
1003   "coco_url": "http://images.cocodataset.org/train2017/000000391895.
1004   jpg",
1005   "height": 360,
1006   "width": 640,
1007   "date_captured": "2013-11-14 11:18:45",
1008   "flickr_url": "http://farm9.staticflickr.com/8186/8119368305
1009   _4e622c8349_z.jpg",
1010   "id": 391895
1011 }

```

Listing 4.9.: Bildinformationen des Datensatzes COCO

Dieses Wörterbuch images ist wohl das zweitwichtigste und enthält Metadaten über die Bilder.

Das Wörterbuch images enthält das Feld id. In diesem Feld wird die Lizenz des Bildes angegeben. Der vollständige Text ist in der URL angegeben. Bei der Verwendung der Bilder ist sicherzustellen, dass keine Lizenzverletzung erfolgt. Im Zweifel sollte auf die Verwendung verzichtet werden. Das heißt aber auch, dass man bei der Erstellung eines eigenen Datensatzes jedem Bild eine entsprechende Lizenz zuweist.

Das wichtigste Feld ist das Feld id. Dies ist die Nummer, die in im Abschnitt annotations verwendet wird, um das Bild zu identifizieren. Wenn man also zum Beispiel die Anmerkungen für die gegebene Bilddatei identifizieren möchte, muss man den Wert des Felds id für das entsprechende Bilddokument in Abschnitt images überprüfen und dann in im Abschnitt annotations einen Querverweis darauf machen. Im offiziellen Datensatz COCO ist der Wert des Feldes id der gleiche wie der Name **file_name** nach Entfernen der führenden Nullen. Falls man einer benutzerdefinierte Datensatz COCO verwendet, muss dies nicht unbedingt der Fall sein.

"Abschnitt "Kategorien

Der Abschnitt categories unterscheidet sich ein wenig von den anderen Abschnitten. Er ist für die Aufgabe der Objekterkennung und Segmentierung und für die Aufgabe der Stoffsegmentierung konzipiert.

Für die Objekterkennung und die Objektsegmentierung erhält man die Informationen gemäß des Listings 4.10

In dem Abschnitt enthalten die Listen die Kategorien von Objekten, die auf Bildern

```

1000 [
1001 { "supercategory": "person", "id": 1, "name": "person" },
1002 { "supercategory": "vehicle", "id": 2, "name": "bicycle" },
1003 { "supercategory": "vehicle", "id": 3, "name": "car" },
1004 ...
1005 { "supercategory": "indoor", "id": 90, "name": "toothbrush" }
1006 ]

```

Listing 4.10.: Klasseninformationen des Datensatzes COCO

```

1000 [
1001 { "supercategory": "textile", "id": 92, "name": "banner" },
1002 { "supercategory": "textile", "id": 93, "name": "blanket" },
1003 ...
1004 { "supercategory": "other", "id": 183, "name": "other" }
1005 ]

```

Listing 4.11.: Stoffinformationen des Datensatzes COCO

erkannt werden können. Jede Kategorie hat eine eindeutige Nummer id und sie sollte im Bereich [1, Anzahl der Kategorien] liegen. Kategorien werden auch in Oberkategorien gruppiert, die man in Programmen verwenden kann, um beispielsweise Fahrzeuge im Allgemeinen zu erkennen, wenn es Ihnen egal ist, ob es sich um ein Fahrrad, ein Auto oder einen LKW handelt.

Für die Stoffsegmentierung existieren eigene Listen, siehe Listing 4.11. Die Nummer der Kategorien in diesem Abschnitt beginnen hoch, um Konflikte mit der Objektsegmentierung zu vermeiden, da diese Aufgaben bei der panoptischen Segmentierungsaufgabe zusammen durchgeführt werden können. Die Werte von 92 bis 182 repräsentieren das wohldefinierte Hintergrundmaterial, während der Wert 183 alle anderen Hintergrundtexturen repräsentiert, die keine eigenen Klassen haben.

Der Abschnitt annotations ist der wichtigste Abschnitt des Datensatzes, der für jede Aufgabe wichtige Informationen für den spezifischen Datensatz enthält. Die Felder gemäß Listing 4.12 haben folgende Bedeutung.

"segmentation": Dies ist eine Liste von Segmentierungsmasken für Pixel; dies ist eine abgeflachte Liste von Paaren, so dass man den ersten und den zweiten Wert (x und y im Bild) nehmen sollten, dann den dritten und den vierten usw., um Koordinaten zu erhalten; dabei ist zu beachten, dass dies keine Bildindizes sind, da es sich um Fließkommazahlen handelt — sie werden von Tools wie COCO-annotator aus den Pixelkoordinaten erstellt und komprimiert.

"area": Dies entspricht die Anzahl der Pixel innerhalb einer Segmentierungsmaske.

"iscrowd": Dies ist eine Fahne, die anzeigt, ob die Beschriftung für ein einzelnes Objekt (Wert 0) oder für mehrere nahe beieinander liegende Objekte (Wert 1) gilt; bei Füllungssegmentierung ist dieses Feld immer 0 und wird ignoriert.

"image_id": Das Feld id entspricht der Nummer des Feldes id vom Wörterbuch images; Warnung: dieser Wert sollte zum Querverweis des Bildes mit anderen Wörterbüchern verwendet werden, also nicht das Feld id.

```

1000 {
1002   "segmentation": [
1004     [
1005       239.97,
1006       260.24,
1007       222.04,
1008       ...
1009     ],
1010     "area": 2765.1486500000005,
1011     "iscrowd": 0,
1012     "image_id": 558840,
1013     "bbox": [
1014       [
1015         199.84,
1016         200.46,
1017         77.71,
1018         70.88
1019       ],
1020       "category_id": 58,
1021       "id": 156
1022     }
1023 }
```

Listing 4.12.: Annotationen des Datensatzes COCO

"bbox": In der Rubrik befinden sich die umgebenden Rechtecke beziehungsweise Bounding Box, das heißt, die Koordinaten in Form der x- und y-Koordinate der oberen, linken Ecke, sowie die Breite und die Höhe des Rechtecks um das Objekt; es ist sehr nützlich, um einzelne Objekte aus Bildern zu extrahieren, da dies in vielen Sprachen wie Python durch den Zugriff auf das Bild-Array erfolgen kann wie;

```
cropped_object = image[bbox[0]:bbox[0] + bbox[2], bbox[1]:bbox[1] + bbox[3]]
```

"category_id": Das Feld enthält die Klasse des Objekts, entsprechend dem Feld id aus dem Abschnitt "categories"

"id": Die Nummer ist der eindeutige Bezeichner für die Annotation; dabei ist beachten, dass dies nur die ID der Annotation ist, sie verweist also nicht auf das jeweilige Bild in anderen Wörterbüchern.

Bei der Arbeit mit Crowd-Bildern ("iscrowd": 1) kann der Teil "Segmentierung" ein wenig anders sein:

```

1000   "segmentation": [
1001   {
1002     "counts": [179, 27, 392, 41, ..., 55, 20],
1003     "size": [426, 640]
1004   }
1005 }
```

Dies liegt daran, dass bei einer großen Anzahl von Pixeln die explizite Auflistung aller Pixel, die eine Segmentierungsmaske erstellen, sehr viel Platz benötigen würde. Stattdessen verwendet der Datensatz COCO eine benutzerdefinierte Komprimierung Run-Length Encoding (RLE), die sehr effizient ist, da Segmentierungsmasken binär sind und RLE für ausschließlich Nullen und Einsen die Größe um ein Vielfaches verringern kann.

Network	CLI argument	NetworkType enum
AlexNet	alexnet	ALEXNET
GoogleNet	googlenet	GOOGLENET
GoogleNet-12	googlenet-12	GOOGLENET_12
ResNet-18	resnet-18	RESNET_18
ResNet-50	resnet-50	RESNET_50
ResNet-101	resnet-101	RESNET_101
ResNet-152	resnet-152	RESNET_152
VGG-16	vgg-16	VGG-16
VGG-19	vgg-19	VGG-19
Inception-v4	inception-v4	INCEPTION_V4

Table 4.1.: Vortrainierte Modelle des Projekts jetson-inference

4.1.5. ImageNet

ImageNet ist eine Bilddatenbank mit mehr als 14 Millionen Bildern, die permanent wächst. Jedes Bild wird einem Substantiv zugeordnet. Zu jeder Kategorie gibt es im Schnitt mehr als 500 Bilder. ImageNet enthält mehr als 20.000 Kategorien in englischer Sprache mit einer typischen Kategorie wie beispielsweise „Ballon“ oder „Erdbeere“. Die Datenbank mit Anmerkungen zu Bild-URLs von Drittanbietern ist direkt über ImageNet frei zugänglich, obwohl die eigentlichen Bilder nicht im Besitz von ImageNet sind. [Den+09; Sha+20; KSH12]

4.1.6. Visual Wake Words

Der Datensatz „Visual Wake Words“ besteht aus 115.000 Bildern, von denen jedes mit der Angabe versehen ist, ob es eine Person enthält oder nicht. [Cho+19] Aus dem Datensatz COCO werden Bilder extrahiert. Da der Datensatz COCO Informationen über die Bilder zur Verfügung stellt, kann bei der Extraktion abgefragt werden, ob innerhalb eines umgebenden Rechtecks sich eine Person befindet oder nicht. Die Daten werden mit der Information gekennzeichnet.

4.2. Modelle

Bei der Installation des Projekts jetson-inference von NVidia bietet das Hilsprogramm eine große Auswahl an vortrainierten Deep-Learning-Modellen zum Herunterladen an. Darunter befinden sich bekannte wie das AlexNet von 2012 sowie verschiedene sogenannte Residual Neural Network (ResNet). Ebenfalls dabei sind SSD-Mobilenet-V2, das 90 Objekte vom Apfel bis zur Zahnbürste erkennt, und DeepScene der Universität Freiburg zur Erkennung von Objekten im Straßenverkehr.

Um weitere Modelle herunterzuladen, kann der Model Downloader aufgerufen werden:

```
$ cd jetson-inference/tools
$ ./download-models.sh
```

Die Modelle GoogleNet und ResNet-18, die auf der Bilddatenbank ImageNet beruhen, werden im Build-Schritt automatisch heruntergeladen. In der Tabelle 4.1 sind alle Modelle aufgelistet, die im Projekt jetson-inference zur Verfügung stehen. Die erste Spalte der Tabelle ist der Name der Struktur des Modells. [Alo+18]. In der zweiten Spalte ist der Übergabeparameter für das Argument `-network` des Programms `imagenet-camera.py` enthalten.

Part II.

Python Packages

5. Machine learning: Neural networks generate content

Generative Adversarial Networks are neural networks that generate texts, videos or music. They are good for restoring old films, but also for falsifying them.

If you take a quick look at the landscape picture in the first image, nothing in particular strikes you at first. The photo could have been taken anywhere in Germany. A look at the details, such as the tree trunks or the incidence of light, creates the feeling that something is not right here.

The solution: this landscape does not exist. The picture was generated by a programme (GauGAN). This was made possible by an idea Ian Goodfellow had in 2014. He went to a pub with a colleague and both discussed one of the biggest challenges in Deep Learning: the amount of examples a neural network needs to learn. Image recognition requires thousands of images. A human being has to look at these beforehand and describe what can be seen on them (labelling) so that the learning process later knows whether the result of the neural network is correct or incorrect.

Ian Goodfellow's colleague didn't want to "merely" analyse image content, but to generate completely new images. After a long discussion, Ian Goodfellow came up with a question: If it doesn't work with one neural network and a lot of prepared data, why not try it with two neural networks? These then compete against each other like in a game. One network generates the images and the second judges whether they are real or generated. Through the learning process, each of the networks becomes better and better at its respective activity without needing any data with descriptions (labels).

- In Generative Adversarial Networks (GAN), neural network learns from each other as counter-players and improve their skills through constant competition. The goal of their game is a generator network that can artificially create content.
- With the help of GANs, you can do a lot of useful things such as restoring old films, but you can also distort content just as well, keyword deepfake.
- Ian Goodfellow's classic GAN architecture has laid the foundation for a number of further developments, including DCGAN Deep Convolutional GAN, PGGAN Progressive Growing GAN or CGAN Conditional GAN.

Ian Goodfellow tried this approach on a simple example and achieved the hoped-for results. The later publication "[Generative Adversarial Nets](#)" ([PDF](#)) founded a new branch of machine learning. The following two figures demonstrate the possibilities.

"Generative" here describes the possibility of generating new things with these nets. The second term, "adversarial", can best be translated in German as "kontroversial" or "gegnerisch". Two neural networks compete against each other and learn from it - learning through rivalry.

The examples in this article refer to the generation of images, as this is the easiest place to publish them. However, GANs can just as well generate texts, sounds, videos or other digital data. This applies to anything that can be represented in digital data.



Figure 5.1.: Where can this landscape be found? In Germany? Or in France?



Figure 5.2.: NVIDIA's StyleGAN project generate images of non-existent people.



Figure 5.3.: University of Berkeley's CycleGAN transfers image style to new images

5.1. Create something new

When it comes to creating new content, not just anything should be generated, but there is always a specific requirement, such as creating landscape images. For this, the "first ingredient" is as many landscape images as possible. From these, the chosen machine learning method can acquire the pattern that is hidden in the images. This hidden pattern is what makes up this type of image. From a mathematical point of view, this is the common, unknown probability distribution regarding the authenticity of the images. A newly generated image should preferably have this distribution in order to pass again as a landscape image. To ensure that it is not a copy of an already existing image, the "second ingredient" is a set of random numbers as input.

In GANs, there are two neural networks. One is the generator, which generates new images, and the other is the discriminator, which judges how good the new image is. You can compare this to an art forger who tries to paint the best possible pictures in the style of a Van Gogh. The police art expert is his counterpart, who has the task of distinguishing the forged paintings from genuine works that may not yet be known. If the expert recognises a forged painting, it means that the forger was not good enough. The forger still has to learn. If the forger (generator) gets better and better, the expert (discriminator) may no longer recognise a forgery. Then the expert (discriminator) has to learn again.

During the learning process, the two networks continue to grow. They virtually build each other up, just as rivals in sport do, for example. At the end of the learning process, there is a generator network that can be used to create very good new, non-real images.

5.2. Mutual performance monitoring

Like all neural networks that are supposed to get better and better through learning processes, both the generator and the discriminator need a function that tells them how good or bad they currently are. In GANs, there is not only one loss function, but each of the networks has its own. Basically, the learning process goes like this: The generator (G) gets a set of random numbers (vector z) as input and generates new images (x_g) from them. The discriminator (D) in turn receives these (x_g) as input and also real images (x_e). Its task is to recognise the difference.

Distinguishing between two things is a classic task for neural networks. Is the picture a car or a ship? The technical term for this is binary classifier (Binary Classifier).

The result of the discriminator $D(x)$ is 1 if it is a genuine image and 0 if it detects a generated one. A value of 0.7 would express that it is more likely to be a genuine work. The discriminator would be 70 per cent certain. The result of the discriminator $D(x)$ thus corresponds to the assessment of whether it could be a genuine image.

The loss function for the discriminator is composed of the sum $D(x_e)$ for the real images and the sum $(1 - D(x_g))$ for the generated images or the sum $(1 - D(G(x)))$ if x_g is replaced by $D(G(x))$. In somewhat more mathematical terms, this looks like Ian Goodfellow's figure below.

$J(D)$ is the loss function of the discriminator and $J(G)$ with a minus in front of it is that of the generator. This represents the opposite intentions of the generator and discriminator.

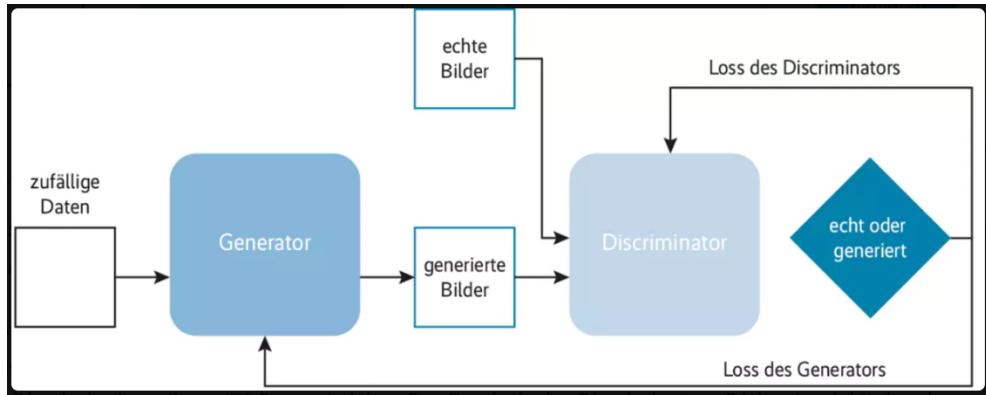


Figure 5.4.: The loss functions control the learning of the neural networks in GANs.

$$\begin{aligned} J^{(D)} &= -\frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2} \mathbb{E}_{\mathbf{z}} \log (1 - D(G(\mathbf{z}))) \\ J^{(G)} &= -J^{(D)} \end{aligned}$$

Figure 5.5.: The mathematical representation of a loss function consists of two sums.

A simple neural network corresponds to an optimisation problem in which the learning process changes its parameters until the best values, i.e. the smallest possible value for the loss function, emerge as the result. This is different with GANs.

It is a game between two participants. If one participant gets better scores, the other automatically gets worse. If one player improves by a certain amount, the other gets worse by the same amount. The sum of the values achieved always remains the same. In game theory, this is known as a zero-sum game.

The aim of the learning process is to reach the situation where the images generated by the generator correspond to the internal pattern of the real images. Then the discriminator can no longer distinguish between "generated" and "real", so the result it gives is always 50 per cent ($D(x) = 0.5$).

At the end of the learning process, the parameters of the generator are set so that its loss function $J(G)$ has reached a local minimum, and for the discriminator this is also true for its loss function $J(D)$.

5.3. Training in rotation

The training of a GAN is somewhat different from that of a simple neural network. In a learning loop, the discriminator is trained first and then the generator. In the next run, it starts all over again.

In the same way, the training procedure differs in detail for each neural network.

5.4. Training of the discriminator

- Select random images from the real sample data.
- Give the generator random data from which it generates images.

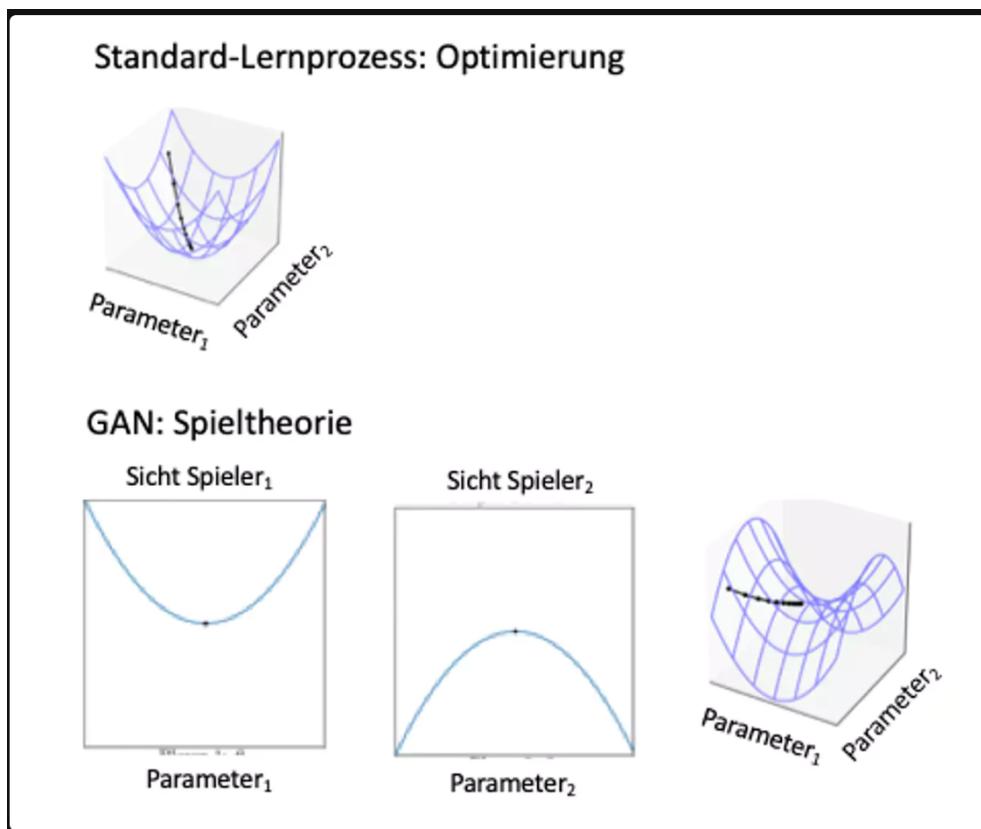


Figure 5.6.: GANs involve two networks competing against each other, unlike simpler neural networks

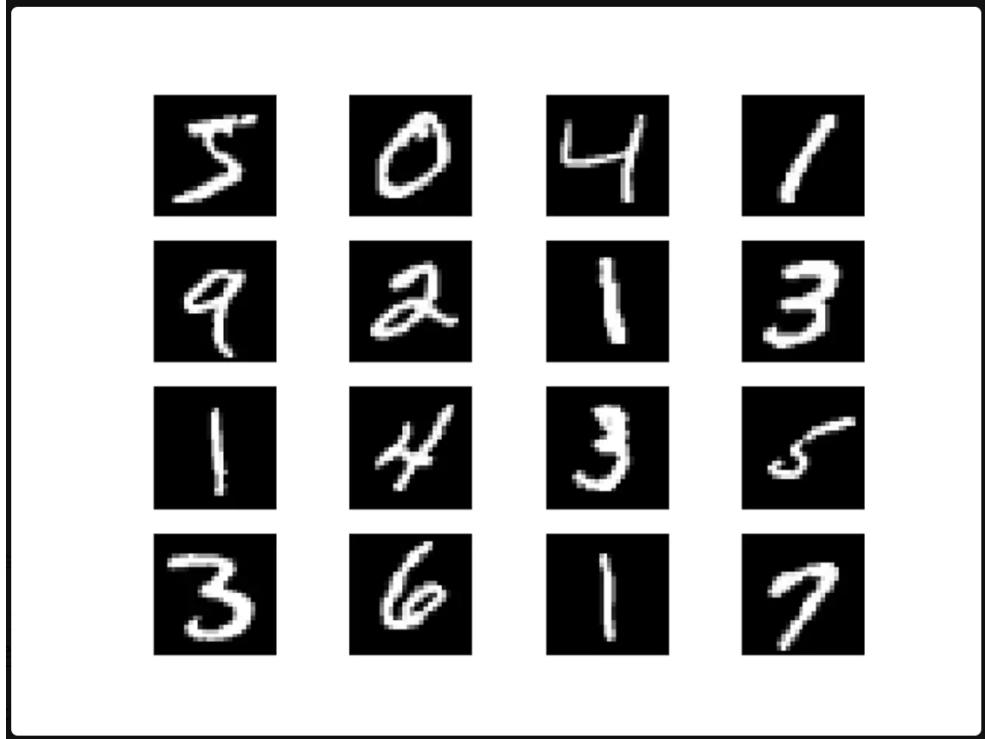


Figure 5.7.: Python GAN learns to generate handwritten digits

- Classify all images whether they are real or generated, regardless of where they come from.
- Adjust your parameters via the loss function using the backpropagation method. The aim is to minimise the error rate when classifying.

5.5. Training the generator

- Take random data and generate new images from it.
- Let the discriminator classify them.
- Adjust your parameters via the loss function using the backpropagation method. The goal is to maximise the error rate when classifying.

The important thing here is that each neural network only adjusts its own parameters during a learning process.

5.6. From theory to practice

The GAN in the example programme `gan.py` from the listing tries to create handwritten digits that look as realistic as possible.

The real images come from [Professor Yann LeCun and his colleagues in the Artificial Intelligence Department \(PDF\)](#) at New York University.

Known as MNIST (Modified National Institute of Standards and Technology) in the field of machine learning, this image database was created from 60000 digits, written by employees of the American Census Bureau and another 10,000 examples of American

```

1000 import numpy as np
1001 from tensorflow.keras.datasets import mnist
1002 from tensorflow.keras.layers import BatchNormalization, LeakyReLU
1003 from tensorflow.keras.layers import Dense, Reshape, Flatten
1004 from tensorflow.keras.models import Sequential
1005 from tensorflow.keras.optimizers import Adam
1006
1007 # Modul mit Statistikfunktionen
1008 from statistics import Statistics
1009
1010 IMAGE_ROWS = 28
1011 IMAGE_COLS = 28
1012 CHANNELS = 1
1013 IMAGE_SHAPE = (IMAGE_ROWS, IMAGE_COLS, CHANNELS)
1014 Z_DIM = 100
1015
1016 SAMPLE_INTERVAL = 10
1017 stat = Statistics(SAMPLE_INTERVAL)
1018
1019
1020 def build_generator():
1021     model = Sequential()
1022
1023     model.add(Dense(256, input_dim=Z_DIM))
1024     model.add(LeakyReLU(alpha=0.2))
1025     model.add(BatchNormalization(momentum=0.8))
1026     model.add(Dense(512))
1027     model.add(LeakyReLU(alpha=0.2))
1028     model.add(BatchNormalization(momentum=0.8))
1029     model.add(Dense(1024))
1030     model.add(LeakyReLU(alpha=0.2))
1031     model.add(BatchNormalization(momentum=0.8))
1032     model.add(Dense(np.prod(IMAGE_SHAPE), activation='tanh'))
1033     model.add(Reshape(IMAGE_SHAPE))
1034
1035     return model
1036
1037
1038 def build_discriminator():
1039     model = Sequential()
1040
1041     model.add(Flatten(input_shape=IMAGE_SHAPE))
1042     model.add(Dense(512))
1043     model.add(LeakyReLU(alpha=0.2))
1044     model.add(Dense(256))
1045     model.add(LeakyReLU(alpha=0.2))
1046     model.add(Dense(1, activation='sigmoid'))
1047
1048     return model
1049
1050
1051 def train(iterations, batch_size=128):
1052     (X_train, _), (_, _) = mnist.load_data()
1053
1054     X_train = X_train / 127.5 - 1. # Werte zwischen -1 und 1
1055     X_train = np.expand_dims(X_train, axis=3)
1056
1057     stat.orig_images(X_train)
1058
1059     valid = np.ones((batch_size, 1)) # labels
1060     gen = np.zeros((batch_size, 1))
1061
1062     for iteration in range(iterations):
1063
1064         # Train Discriminator
1065
1066         idx = np.random.randint(0, X_train.shape[0], batch_size)
1067         imgs = X_train[idx]
1068
1069         noise = np.random.normal(0, 1, (batch_size, Z_DIM))
1070         gen_imgs = generator.predict(noise)
1071
1072         d_loss_real = discriminator.train_on_batch(imgs, valid)
1073         d_loss_gen = discriminator.train_on_batch(gen_imgs, gen)
1074         d_loss = 0.5 * np.add(d_loss_real, d_loss_gen)
1075
1076         # Train Generator
1077
1078         noise = np.random.normal(0, 1, (batch_size, Z_DIM))
1079         g_loss = combined.train_on_batch(noise, valid)
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
```

high school students. Normally, ML uses these images to generate networks that can distinguish digits in GANs to generate realistic ones.

The Python example programme works with Keras and the TensorFlow-2-API. [The free programme collection by Erik Linder-Norén serves as a model..](#) There you can find the implementations of various further developments of the classic GAN in Python. All Keras modules and objects can be found at tensorflow.keras. The MNIST images needed for learning can also be imported from there. The output of the programme is prepared in the `statistics` module.

Each image in MNIST consists of 28 by 28 pixels and different grey levels. The number of random numbers (vector z) that the generator receives as input is 100 in this example: `Z_DIM = 100`.

5.7. The game structure

The generator (see function `build_generator`) consists of several layers of the type `Dense`. All neurons are connected to each other. `LeakyReLU` is used as the activation function between the layers.

The normal function `ReLU` sets all values that are smaller than 0 to the value 0. The activation function `LeakyReLU`, on the other hand, returns a negative value that is slightly smaller than 0, such as $0.01 * x$, for such values (x). This often allows the optimiser to reach the goal more quickly.

The last layer has `tanh` as an activation function so that the values for the output of the generator are between -1 and 1.

Like the generator, the discriminator (see function `build_discriminator`) consists of layers of the type `Dense` and also works with the activation function `LeakyReLU`. The last layer of the network has `sigmoid` as its activation function so that the output of the discriminator is between 0 and 1. The output in this case corresponds to the probability that the discriminator generates the image for (0) or genuine (1).

The model for training the discriminator is composed of the neural network of the discriminator, an Adam optimiser and the loss function binary cross entropy. This describes the difference between probability distributions, in this case between the value estimated by the discriminator and the actual value (0 = generated, 1 = real). The article "A Gentle Introduction to Generative Adversarial Network Loss Functions" by Jason Brownlee describes in detail why the cross entropy can be used for GANs and ["Classical formula"\(PDF\)](#) by Ian Goodfellow cannot be used directly. For the learning process of the generator, one needs a combination of the neural network of the generator and the discriminator. Since only the generator is supposed to learn in this model, the `trainable` setting of the discriminator is set to `False`.

5.8. Improve step by step

The learning process runs in the `train` function. With each loop pass, the discriminator learns first and then the generator. For the discriminator, the `train` function first selects random images from the real ones. After that, it needs a corresponding number of generated images from the generator.

The Keras function `train_on_batches` gets the images as the first parameter and the corresponding labels as the second. When training with the real images `imgs` the vector `valid` for the labels consists of ones. The one here means "real", the zero "generated". The return value `d_loss_real` is a vector that contains the loss value of the discriminator's estimation for each image.

When training with the generated images `gen_imgs` and the vector `gen` with the labels (contains only 0), the vector `d_loss_fake` provides the loss values for the generated

images. The total loss value can be determined from the values `d_loss_real` and `d_loss_gen` with a weighting of 0.5.

In the learning process of the generator, the model combined, consisting of the networks of the generator and discriminator, is used. In the `train_on_batch` function, the generator first receives the input `noise` (random values) and thus generates an image. This is immediately passed on to the discriminator, which evaluates it. However, this function only trains the generator, as the training for the discriminator is switched off in the model `combined`.

Both neural networks have gone through their learning process and the loop starts again from the beginning. When the entire learning process is complete, the generator takes over creating the new images on its own.

Ian Goodfellow's classic GAN described here triggered an avalanche of further developments.

5.9. GAN-Zoo

The idea of the GAN architecture by Ian Goodfellow in 2014 was the starting signal for many further developments.

- **DCGAN Deep Convolutional GAN:** The DCGAN replaces the classic fully linked layers (dense) with the convolutional layers known from Deep Learning. These are more powerful for image recognition and similar topics. Fewer learning processes are necessary. In addition, data is normalised between the individual layers (batch normalisation), which further improves performance.
- **PGGAN Progressive Growing GAN:** The essential new feature of PGGAN is the learning process. This becomes increasingly difficult the more the neural networks swell. In order to achieve better resolutions for images or other data, however, the networks have to keep growing. The idea behind PGGAN is to start with the smallest possible networks in the generator and discriminator. First you take images with low resolution, together with a few layers, and increase this more and more. This shortens the time for the learning process. This allows PGGANs to achieve higher resolutions in images with the same resources.
- **Cycle GAN:** CycleGAN is about automatically transferring images from one domain, such as summer images, to another domain, winter images. Until now, pairs of images were always necessary for the learning process, a winter image and a corresponding summer image. Of course, it is very time-consuming to first obtain as many of these image pairs as possible. CycleGAN manages this transfer from one image to another without image pairs. This makes it possible, for example, to convert photos into pictures of famous painters. There can be no image pairs for this.
- **CGAN – Conditional GAN:** CGAN do not generate just any images, the user controls how they should look. If there is additional data (features) about the gender, age or appearance of the person depicted, he or she can wish the generated image to show, for example, a red-haired, 20-year-old man.

If the user can determine the appearance of the image via specifications, the perspective for the future is that such GANs could one day be able to replace the classic graphics engines, for example in computer games.

A list of many more GANs can be found on Avinash Hindupur's GAN Zoo page.

5.10. Real or computer generated?

Since 2017, the term deepfake has been doing the rounds. This stands for images or other computer-generated media that look deceptively real.

For example, if a person is to have a particularly positive effect on a social platform, he or she needs as many followers as possible - why not create many fake accounts that follow you? This in turn requires pictures of people. Anyone who has read this article about GANs will know where to get them. According to a report by heise online, researchers have already identified countless fake accounts on Facebook.

Of course, much more is possible with neural networks. For example, there are generated videos in which Barack Obama says what the "scriptwriters" have come up with. In the same way, programmes can imitate a person's biometric data or put faces of celebrities on the bodies of porn stars.

In order to be able to do something about this, it is currently up to the lawyers and the technicians. Legislation must be updated to make such offences punishable. For example, the US House of Representatives is preparing an anti-deepfake bill.

On the technology side, one of the goals is the automatic detection of deepfakes. Facebook has offered rewards totalling 10 million US dollars in its Deepfakes Detection Challenge. As is often the case in the field of IT security, there will probably be a constant neck-and-neck race between the producers of deepfakes and those who expose them.

But GANs and similar technology offer just as many positive applications, from simpler workflows for post-processing images and films to the automatic restoration of photos. Neural network techniques even have the potential to replace existing graphics engines and generate ever better images and sounds in films and computer games.

6. Package Example

6.1. Introduction

6.2. Description

6.3. Installation

`pip packageExample`

6.4. Example - Manual

6.5. Example

6.6. Example - Version

6.7. Example - Files

`PackageExample.py`

6.8. Further Reading

References

- [Aba+16] M. Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.

7. Data Analysis with Python: First Step with Pandas - Evaluating Football Data

7.1. Introduction

Pandas is the standard library for anyone who wants to process data with Python. This makes it easy to analyze even the first Bundesliga season. Anyone who wants to process or edit data with Python can hardly get around the Pandas library. In a way, it is the gold standard for Python when it comes to structured information.

The size of the original data is irrelevant to Pandas: the library can handle CSV files, Microsoft's Excel format, and large amounts of data in a JSON file. It doesn't matter whether you want to tackle a small or large data project, Pandas usually always simplifies the handling of data of all kinds.

In this article, we explain the basic concepts that Pandas works with: the series and the data frame. We also show how to get free soccer data from the internet and how to analyze the first Bundesliga season 1963/64. Basic Python knowledge is helpful for this, so you should already have installed libraries with pip

The term pandas come from the word panel data, which usually consists of rows and columns. If you want to explain pandas to a non-techie: It's like working with an Excel spreadsheet, only in Python code.

Developer Wes McKinney has been working on Pandas since 2008. At the time, he was still employed by an investment management firm and was looking for a fast and flexible tool to perform quantitative analyses. This eventually led to the open-source library Pandas. There are now 28 maintainers of the software, and McKinney is still involved

7.2. Installation

The Pandas library is installed using the Pip package manager as follows:

```
pip install pandas
```

Pandas are also part of the Sci-Py stack, an open-source software environment for mathematicians, analysts, and other scientists. Pandas developers prefer to install via the [Python-Distribution Anaconda](#),which includes the Sci-Py stack including pandas

7.3. Understand [Series](#)

The two main data structures that Pandas provides are the [Series](#) and the [DataFrame](#). The [Series](#) can be thought of as a simple indexed list, them consists of two arrays. With this code you can create a small [Series](#) with three elements:

```
series = pd.Series([1, 2, 3])
```

If you output the variable series, the index, which starts at zero, is next to the individual rows:

```
0 1  
1 2
```

2 3

The data type of the values is also given in the output, `int64` means 64-bit integers. Using the index, the user can access individual values: `print(series[1])` returns about `2`.

The `Series` may look like a normal Python list, but it works a little differently. For example, you can change the index and then access values from the `series` via the new index:

```
series = pd.Series([1, 2, 3], index=["First digit", "Second digit", "Third digit"])
print(series["Second digit"])
```

This code makes a `2` appear in the output again. In addition, mathematical operations are applied to all `series` values. A print operation like `series*2` results in the following output:

```
0 2
1 4
2 6
```

This is not possible with a Python list, this operation would only double the list, not the values. The principle is continued if you want to add two `Series objects`, for example:

```
series1 = pd.Series([1, 2, 3])
series2 = pd.Series([4, 5, 6])
print(series1+series2)
```

The result is a new `Series`, with the added values from the two previous ones:

```
0 5
1 7
2 9
```

If these were two normal list items, the output would look like this instead:

```
[1, 2, 3, 4, 5, 6]
```

7.4. Create DataFrame

A `DataFrame` can also be easily created by hand. To do this, you use the `dictionary` built into Python

```
data = "numbers": [1, 2, 3], "letters": ["A", "B", "C"]
```

The `dictionary` is enclosed in curly brackets and contains two related data separated by colons. Several data pairs are distinguished by commas in the `dictionary`. The first data point is the key, the second is the associated value.

In this example, each key-value pair represents a column in the dataframe. The key consists of a string, here numbers and letters. These will later be the column headers. The corresponding value consists in both cases of a list with three elements, which are then the individual values in the respective column.

The `dictionary` is now passed to Pandas so that it becomes a `DataFrame`: `dataframe = pd.DataFrame(data)`

`pd.DataFrame` takes the `dictionary` and stores the `DataFrame` in the variable `dataframe`. In many Pandas examples you will also find the `abbreviation` as the variable name for the `DataFrame`. If you then `print(dataframe)` this variable, you will see the data in a table representation:

```
Numbers Letters
```

```
0   1   A
1   2   B
2   3   C
```

A first overview of the data can be obtained with `shape`. So `print (dataframe.shape)` prints the row (3, 2). So the `DataFrame` consists of 3 rows and 2 columns. `print(len (data frame))` shows you only the number of rows, which is 3. This is for the sample data is not very surprising, since you know what you have typed. However, as soon as foreign data comes into play, `shape0` and `len()` become relevant again. A summary of the dataframe shows `info()`. For the small example data frame, the input `print (dataframe.info())` gives the following output:

```
1000 <class 'pandas.core.frame.DataFrame'>
1001 RangeIndex: 3 entries, 0 to 2
1002 Data columns (total 2 columns):
1003 #   Column    Non-Null Count  Dtype  
1004   0   Nmber     3 non-null    int64 
1005   1   letter    3 non-null    object 
1006   dtypes: int64(1), object(1)
1007   memory usage: 176.0+ bytes
```

At a glance you can see the index, the number of columns, how many rows the data frame has, and which data types the values in the columns consist of. The first column with the numbers consists of 64-bit integers, this is shown by the hint `int64` under `Dtype` - as already with the `Series`. Finally, `info()` shows how much memory the data frame needs.

7.5. Select data

In order to select a special column from these sample data now, it is sufficient to use the put the name of the column in square brackets behind the dataframe. `print(dataframe["letters"])` thus has this issue:

```
1000 0 A
1001 1 B
1002 2 C
```

In addition to the defined rows and values, the `DataFrame` also consists of an index that starts at zero. Via this index you can address a special row, for example, with loc: `print(dataframe.loc[1])`

If you enter this code, you will only get the values of the second row as output - without index:

```
1000 Number      2
1001 Letter     B
```

The index can also be customized by passing a list of new index values to the `DataFrame()` function. This is how the following code becomes

`dataframe = pd.DataFrame(data, index=["First row", "Second row", "Third row"])` this issue:

	Number	Letter
1000	First Row	1 A
1002	Second Row	2 B
	Third Row	3 C

The second row can now be `printed` using a `print(dataframe.loc["Second row"])`

7.6. Integer

Besides `loc` there is also `iloc` - the `i` stands for integer. About this you can call the index interdependently of its self-selected name, but again about its index number. This index also starts at zero. `print(dataframe.iloc[1])` thus gives the same output as `print(dataframe.loc["Second row"])`.

While `loc` and `iloc` select whole rows and columns, you can determine a single value with `at`. To do this, you pass at the index value and the column name so that the appropriate value is extracted from the `DataFrame`. In the example without a custom index, `print(dataframe.at[0, "letters"])` will output a single `A`. If you use your own index from before, you get the same result with `print(dataframe.at["First row", "Letters"])`.

Similar to `loc` and `iloc`, there is also `iat` in addition to `at`. Those who don't like letters will be happy with this, because the `i` again stands for integer. For the `A` to appear in the output, a `print(dataframe.iat[0, 1])` is sufficient.

7.7. Get soccer data

With the current dataframe, you can only play around and learn about pandas to a limited extent. We need real data. There are plenty of free datasets on the web that you can use for your own projects. On Github you can find all

[Spielergebnisse der Fußball-Bundesliga im CSV-Format](#).

CSV files contain data separated by commas. For humans, the files are rather difficult to read, machines love them. As an example, let's take the first matchday of the very first

[Dokumentation](#)

Pandas can of course also handle other formats, such as Excel files, JSON data or SQL files. An overview of different possibilities shows [Dokumentation](#)

Now, however, the soccer data must first enter the program. With the function `read_csv()` Pandas can read CSV files and immediately creates a dataframe from them. If you are in a hurry, just put the link to the data in quotes and pass it to `read_csv()`:

```
dataframe = pd.read_csv("https://raw.githubusercontent.com/footballcsv/deutschland/master/1960s/1963-64/de.1.csv")
```

This is not optimal. The data could eventually disappear from the network and then the whole program would no longer work. After all, nothing on the Internet is for eternity.

It is more elegant to download and save the CSV file. Pandas can then access the saved file. The easiest way to do this is to use the Python library `requests`. You have to install it first, preferably via Pip

`pip install requests`

You then get the file onto the disk like this:

```
import requests
```

```
bundesliga_csv_url = "https://raw.githubusercontent.com/footballcsv/deutschland/master/1964/de.1.csv"
download = requests.get(bundesliga_csv_url) with open("bundesliga_ester_spieltag.csv", "wb") as file: file.write(download.content)
```

The variable `bundesliga_csv_url` contains the URL to the CSV file, `requests.get` (`bundesliga_csv_url`) then gets the data via the Get method. The two further lines

save the content, that is `download.content`, as a CSV file. `bundesliga_ester_spieltag.csv` is then the name of the CSV file created from it. The file is saved here in the folder where the Python file is located. The script executes the download every time. However, you can also put the few lines in an extra script and run it only once. Pandas should now work with the downloaded data. Since the file is in the same folder as the script, it is sufficient to enter `dataframe = pd.read_csv("bundesliga_ester_spieltag.csv")`. If you save the file somewhere else, you have to adjust the path accordingly.

7.8. Show info

Using the methods from before, you can now get some information about the Show DataFrame:

```
print(len(dataframe)) print(dataframe.shape) print(dataframe.info())
len(dataframe) again outputs the number of rows, here 240 - 16 teams play 30
games each, which results in 240 games. The shape is (240, 5), so 240 rows and 5
columns. Info() finally shows the names of the columns: Matchday, Date, Team 1,
FT and Team 2. Only the data points in Matchday are 64-bit integers each, the rest
consists of indeterminate types object.
```

So you already know a lot about the data, but what exactly is in it? Is it really soccer information or has someone filled the columns with cooking recipes? Especially when you work with other people's data, you can't avoid taking a look. A first look is provided by `head()`. With this you quickly get the first 5 entries in the output. `print(dataframe.head())` thus gives in our example

	Matchday	Date	Team 1	FT
	Team 2			
1000	0	1 Sat Aug 24 1963	Werder Bremen	3–2
1002	Borussia Dortmund	1 Sat Aug 24 1963	1. FC Saarbrücken	0–2
	1. FC Köln	1 Sat Aug 24 1963	TSV 1860 München	1–1 Eintracht
1004	Braunschweig	3 Sat Aug 24 1963	Eintracht Frankfurt	1–1 1. FC
1006	Kaiserslautern	4 Sat Aug 24 1963	FC Schalke 04	2–0 VfB Stuttgart

Lucky, the first rows show soccer data. By the way, `FT` stands for Full Time, so it gives the result after the full playing time. `print(dataframe.tail())` shows the last 5 entries:

	Matchday	Date	Team 1	FT
	Team 2			
1000	235	30 Sat May 9 1964	Karlsruher SC	1–2 Eintracht
1002	Frankfurt	236 30 Sat May 9 1964	TSV 1860 München	3–2 Werder Bremen

1004	237	30	Sat May 9 1964	1. FC Saarbrücken	1–1	FC
	Schalke 04					
	238	30	Sat May 9 1964	Preußen Münster	4–2	
	Hertha BSC					
	239	30	Sat May 9 1964	MSV Duisburg	3–0	1. FC
	Kaiserslautern					

5 entries is the default value, but you can change that: `dataframe.head(3)` shows only the first 3 entries, for `tail()` it works the same way. If you still want to see data from the middle of the dataframe, you can show it with a simple `slice`, like `print(dataframe[112:116])`.

7.9. Goal statistics

With these few commands, the user now knows what data they are dealing with. Now let's get some interesting statistics from the data. For example, the `value_counts()` function shows how often a particular value occurs in the data. If you apply it to the `FT` column with the results, you immediately know which final result was most often on the scoreboard. An appended `.head(10)` would, by the way, only output the 10 most frequent results.

The first Bundesliga season must therefore have been quite boring. `print(dataframe["FT"].value_counts())` shows that a 1:1 occurred most often, namely 25 times. Directly behind it doesn't get any better, 2:2 was the final result 18 times. A 9:3, 6:1 or 7:0 was rather the exception, these results occurred only once each. By the way, soccer has not exactly become more interesting. In the 2019/20 season, 1:1 was also the most frequent result, occurring 32 times - after all, there were more 2:1 and 1:2 results than in 1963/64, but fewer goal festivals.

All results of a specific matchday can be displayed using the Matchday column. For the 5th matchday the value in the column must be 5. `fuenfter_spieltag = dataframe["Matchday"] == 5` results in a `Series` with True and False values - if the 5th matchday is the `Series` shows True, all other matchdays are False. This `Series` is then applied to the `DataFrame`:

```
print(dataframe[fifth_game_day])
```

So all rows are output, which belong to the 5th matchday - so all 8 matches. Now we don't want only a certain matchday, but the matches of a best-timed team. This works basically like before: The team columns are filtered by the club and the resulting true-false series is then applied to the data frame. Here, however, you have to search through two columns, Team 1 and Team 2. The user wants to have all matches listed and not just the home or away matches. `dataframe["Team 1"] == "Borussia Dortmund"` browses the home matches of Borussia Dortmund, `dataframe["Team 2"] == "Borussia Dortmund"` the away matches. The or operator | connects both commands, which have to be in brackets for this:

```
games_bvb = (dataframe["Team 1"] == "Borussia Dortmund") | (dataframe["Team 2"] == "Borussia Dortmund")
```

`spiele_bvb` is a series that is true whenever Borussia Dortmund plays. If you then apply this series to the entire data frame, you get a list with all the games of the club: `print(dataframe[games_bvb])`

7.10. Format data

Data from public sources seldom have the format or the designations that one would like. In the case of Bundesliga data, for example, the English column names are annoying. German names would be more appropriate. For this Pandas has the handy

function `rename()`. You simply put the changes into a `dictionary` and pass it to the function:

```
1000  dataframe = dataframe.rename(columns={"Matchday": "Spieltag",
1001      "Date": "Date",
1002      "Team 1": "Home Team",
1003      "FT": "Endergebnis",
1004      "Team 2": "Guest Team"})
```

Matchday becomes matchday, `team 2 becomes` visiting team and so on. It is somewhat clearer to define the `dictionary` outside the function and then simply call it:

```
1000  replace_column_name = {"Matchday": "Spieltag",
1001      "Date": "Date",
1002      "Team 1": "Home Team",
1003      "FT": "Final Result",
1004      "Team 2": "Guest Team"
1005  }
1006  dataframe = dataframe.rename(columns=replace_column_name)
```

7.11. Date

So the column names are correctly translated, but the dates are not yet formatted. The first matchday is `Sat Aug 24 1963`, which is better. Pandas provides the function `to_datetime()` for this:

```
dataframe["date"] = pd.to_datetime(dataframe["date"])
```

The function `to_datetime()` is fed with the date column. The old date column is then replaced with the new, formatted date column. If you output the new column, the first game day is now listed as `1963-08-24`.

`print(dataframe.info())` shows that the date column is now no longer a type `object`, but a type `datetime64[ns]`. These are internal 64-bit integers stored in nanoseconds (ns). `datetime64[ns]` can store dates between 1678 and 2262, plus-minus 292 years from January 01, 1970, the start of Unix time.

`1963-08-24` is already quite nice, but if we unambiguate everything, then also the date. The function `dt.strftime()` is responsible for that:

```
dataframe["date"] = pd.to_datetime(dataframe["date"]).dt.strftime("%A, %d. %B %Y")
```

The function converts a date element into a string. With special formating codes `Datetime` the content of the string is determined. `%A` is the written out weekday, `%d` the day, `%B` the written out month and `%Y` the year with four digits.

In order for the formatting codes to display German weekdays and months, the user may need to include the `locale` library beforehand. It is integrated in Python by default, so it does not have to be installed via pip and provides country-specific data formats:

```
import locale
locale.setlocale(locale.LC_ALL, "en_DE")
```

`LC_ALL` means that all country-specific formats are addressed. `de_DE` is a Language Code Identifier and means quasi: German in Germany. `de_AT` would be German in Austria for example. If you now output the date, this line, among others, is in the output:

`Saturday, August 24, 1963`

But this makes the date column a type `object` again and no longer a type `datetime64` - all string elements are a type `object` for Pandas. However, the focus here is on human readability, not machine readability. To change the data in the column back to a Type `datetime64`, apply the function `to_datetime()` again, but give it the previously used formatting codes of `dt.strptime()` as parameters:

```
dataframe["date"] = pd.to_datetime(dataframe["date"], format="%A, %d. %B %Y")
```

By means of the parameter `format` the function recognizes the content of the string and can format as the correct date.

8. Package Example

8.1. Introduction

8.2. Description

8.3. Installation

`pip packageExample`

8.4. Example - Manual

8.5. Example

8.6. Example - Version

8.7. Example - Files

`PackageExample.py`

8.8. Further Reading

References

- [Aba+16] M. Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.

9. Barcode

9.1. Introduction to Barcode

The barcode was invented by the Norman Joseph Woodland and Bernard Silver and was patented in the US in 1951, the invention was based on the morse code which was extended to the thin and thick bars [SW52].

“Barcodes are the machine-readable symbols that store data about part or product to which they are associated”

These are the symbols when read by the scanners or mobile phones are decoded, recoded, and processed to extract the data for variety of purposes e.g., for pricings, sortation process, order fulfillments, shipping, storing data (dimensions, colors, shelf life etc. of the parts of equipment on which they are embossed). There are four main categories of Barcode which have different purposes, different ability and they also differ in shapes.

1. 1-D Linear Bar code
2. 2-D Matrix Bar code
3. Postal codes
4. Stacked Liner Code

9.1.1. 1-D Linear Bar Code

As from the name it is well defined that it has only one dimension and was invented by the two school students in 1948 in USA based on morse code. It is a conventional and mostly used barcode today. One-dimensional barcode data is stored in linear, parallel lines that vary in width and spacing, scanner read the data from left to right. Different versions of linear barcode store different kind of data. In a simplest form barcode contains 30 black lines and 29 white lines. The lines represent numbers, which were printed underneath. Each pattern contains 95 bits of binary code. [YK17]

Identifying barcodes and their structure

To understand the structure of barcode first we have to look at its picture in figure 9.1.

Left Hand Guard Bars: These bars serve as a starting reference point for the scanning devices.

Number System Character: This digit identifies the type of manufacturer.

Manufacturer ID Number: Each company is assigned the unique MIN with the number system by the uniform code council.

Tall Center Bar: These bars serve as middle reference bar for scanning device.

Item Number: Companies are responsible for assigning unique 5-digit number to their product

Module Check Character: It is derived from a mathematical formula To check the accuracy.

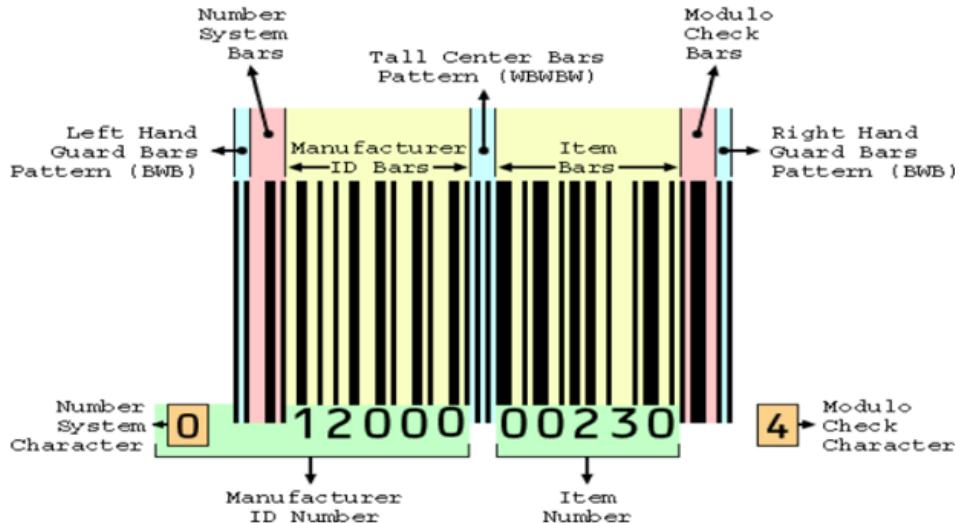


Figure 9.1.: 1-D Linear Bar Code

Right Hand Guard Bars: It helps scanner to identify ending reference point.

Furthermore,

Modulus Check Bar correspond to Modulus Check Character,

Manufacture ID bar correspond to Manufacture ID number,

Number system bar corresponds to Number system Character, And

Item Bars corresponds to the type of item for eg. Toys , Food, Automotive and etc.

9.2. Types of Barcodes

9.2.1. UPC Code

Abbreviate known as universal product code and famously known as UPC bar code are used to read the end consumer products mainly in USA, Canada and New Zealand and many others. There are two versions of the UPC code UPC-A and UPC-E. It has the ability to encode 12 numerical digits of data in UPC-A and 6 numerical digits of data in UPC-E. UPC-A is used for the products with large volume size and UPC-E which is short in size as compared to the UPC-A Variant, there industry of use is retail sector mostly, an example of UPC-Code can be visualized in fig 9.2

9.2.2. EAN Code

Also known as European Article Number mostly used in Europe and many other countries of world including Japan. It comprises of thirteen numeric digits one extra than UPC. It also has two variants one is EAN-13 and the other is EAN-8. This code is called JAN-13 in Japan and abbreviated as Japanese Article Number. ISSN and ISBN is also made up on the same pattern but ISSN is utilized for magazines and ISBN is used for books. Because of the high density of this code it can hold more data at less space, normally used in retail stores. 9.3

9.2.3. Code 128

It is the most advanced recently introduced and robust symbol in the barcode family. The number 128 refers to the ability to hold any character of the ASCII 128 character

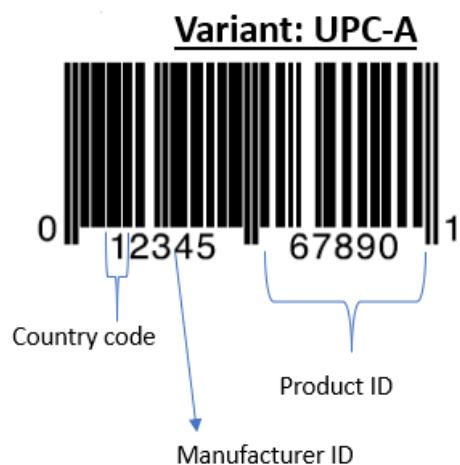


Figure 9.2.: UPC-Code

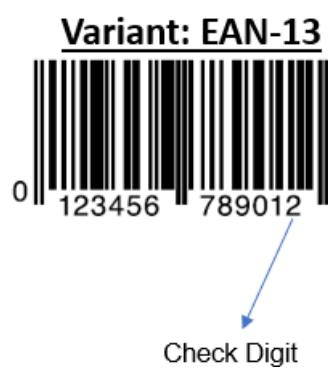
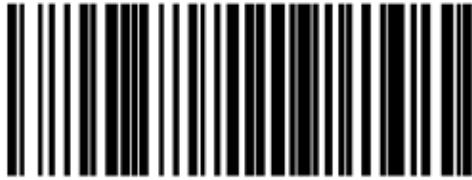


Figure 9.3.: EAN-Code



Variant: EAN 128



Figure 9.4.: CODE-128

set which includes all digits, characters and punctuation marks. Compact and Alpha Numeric data storage. It also comprises of the two variants: Code 128, EAN 128(two alpha characters in starting). It's main area of application is delivery, chemical, and electrical industries etc.

There are many other kind of barcode used for different kind of industries but as our concern is dependent on the food labels and retail we just mention the short description on the other kind of 1-D linear barcode.9.4

9.2.4. Code 39

first code to use number and letters and can encode 43 characters. widely used in Military and Automotive sectors.

9.2.5. Extended Code 39

It allows the use of special characters in code 39. The 128 characters according to ISO 646 are represented by a combination of two symbol characters, the first of which consists of one of the four characters (- + /) and is followed by one of the 26 letters. used in military and Auto.

9.2.6. Code 93

Code 93 was designed to encode data more compactly and with higher data redundancy than with older multi-length barcode types such as Code 39.used in Military, Health and automotive sector.

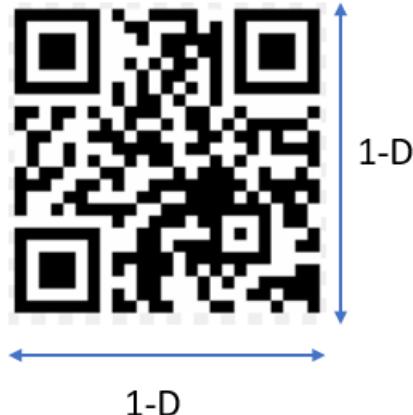


Figure 9.5.: 2D Matrix

9.2.7. Coda bar

It is a discrete, self-checking barcode that allows encoding of up to 16 different characters, plus an additional four special start and stop characters, which include A, B, C and D. used in Health and Photo industry.
And other types of Bar-Codes are listed below.

1. Interleaved 2 of 5
2. 2 of 5 data logic
3. 2 of 5 Industrial
4. 2 of 5 IATA
5. Post Net
6. Intelligent Bar code
7. MSI/Plessy
8. 4 State Bar code
9. GS1 data bar omni directional
10. GS1 data bar expanded

9.3. 2-D Matrix Code

Data is stored in 2-D Matrix (Longitudinal and Transverse Direction)
2-D Matrix Code is further divided into five branches named as

1. QR-Code
2. Data Matrix
3. Aztec
4. Maxi Code
5. PDF 417

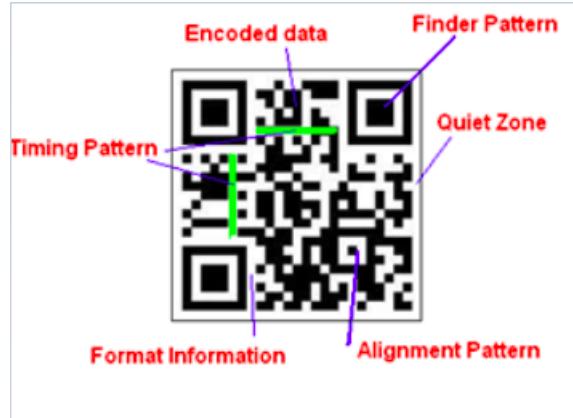


Figure 9.6.: QR Code

From the analysis we found that food labels only contain QR-Code and not the other types of 2-D Matrix code, so we try to explain the QR-Code in detail and its other types briefly to have a know-how.9.5

9.3.1. QR-Code

Finder Pattern: QR (Quick Read) codes contain square blocks of black cells on a white background with finder patterns in the top left, top right, and bottom left corners. 9.6

Alignment Pattern: Does not contain the data but tells the scanner the address for correcting the distortion of the QR code, the black isolated cell is placed in it.

Encoded Data: These are the pattern that contains the data to be stored in the code.

Quite Zone: This is the white line which helps in separating the QR code from other labels or data on site.

Timing pattern: It is placed between the two-finder pattern, use to identify the central coordinates

Formate Informateion: The scanner comes in the contact with the QR- Code in any orientation in the retail shop and it becomes more handy for the operator and increase the speed of the process if the QR -Code can be re in any orientation

9.3.2. How QR-Code works Technically.

working of the QR-code totally depends upon understanding the function of scanners use for decoding the code. • QR scanner works bottom right and goes upwards until it hits position marker. • It then goes right to left and zig-zag until all the modules are covered.

9.3.3. Basic steps How QR (Quick Response) Works.

1. Decoder first Recognizes the three position markers in the QR code. With a sufficient quiet area
2. The scanner begins at the bottom right, where it encounters the mode indicator. These four data modules indicate what data type (numeric, alphanumeric, byte, or kanji)

Version	Modules	ECC Level	Data bits (mixed)	Numeric	Alphanumeric	Binary
1	21x21	L	152	41	25	1
		M	128	34	20	1
		Q	104	27	16	1
		H	72	17	10	1
2	25x25	L	272	77	47	3
		M	224	63	38	2
		Q	176	48	29	2
		H	128	34	20	1
3	29x29	L	440	127	77	5
		M	352	101	61	4
		Q	272	77	47	3
		H	208	58	35	2

Figure 9.7.: Version 1-3

3. Next Scanner encounters the character count indicator and what are the next 8 data modules from the mode indicator this tells characters in encoded data.
4. Scanner then goes in zig-zag along data module until it completes its task.
5. Scanner then proceeds to error correction modules. In these encoded modules from one of four levels of error correction. [Han+17]

9.3.4. QR-Code Versions.

There are 40 different versions of QR-Code present until now. Each version has different data modules and (black and white spaces). Versions are directly proportional to data Modules and also directly proportional to data hold capacity. Below are shown the different versions of QR-Code, with error correction system as larger the error correction is present in the QR-Code less the data stores but its advantage is that the greater the error correction is greater is the chance to restore the data from the damaged QR-Code.9.7

Where L,M,Q and H is the error correction level. In these levels 7,15,25 and 30 percent of the data can be restored, respectively.9.8

9.3.5. Patterns In QR-Code.

Mode Indicator is used for finding the type of data saved in the QR-Code data could be Numeric Mode, Alpha Numeric, Byte Mode, Kanji Mode etc.9.9

Character Count Indicator is used for the scanner to find how many characters it should scan until it stops.

Standard Data Modules is used in the QR code for holding the useful data.

Stop Indicator tells the scanner that it has scanned all the characters in the QR-Code and it matches all the data with the data collected from the character count indicator.

		L	21,616	6,479	3,927	2,6
38	169x169	M	16,816	5,039	3,054	2,0
		Q	12,016	3,599	2,181	1,4
		H	9,136	2,735	1,658	1,1
		L	22,496	6,743	4,087	2,8
39	173x173	M	17,728	5,313	3,220	2,2
		Q	12,656	3,791	2,298	1,5
		H	9,776	2,927	1,774	1,2
		L	23,648	7,089	4,296	2,9
40	177x177	M	18,672	5,596	3,391	2,3
		Q	13,328	3,993	2,420	1,6
		H	10,208	3,057	1,852	1,2

Figure 9.8.: Version 38-40

Error Correction Modules helps the scanner to match the data if the data collected data is same or is it damaged by any means.

9.3.6. Versions and Data Density:

Load time of the data(decoding of the data) is directly proportional to the data modules present in the QR-Code. Given below are the glimpse of sizes and version of the data QR-Code and their data density.

As we can see in the fig9.10 in version 1 the data is less so the load time is less and gradually in the figure9.11 and fig9.12 the load time increases but at the same time more data can be stored.

9.3.7. Types of 2D-Code and their Features

The table above shows the brief over-view of the other types of 2D-Codes as compared to the features of the QR-Code. All the other codes (Data Matrix, Maxi Code, PDF 417 and Aztec) have different applications and different structures, with different features.9.13

ASCII : American Standard code for Information Interchange

ISO : International Organization for standards

ISBN : International Standard Book Number

ISSN : International Standard Serial Number

AIM : Association for Automatic Identification and Mobility

JIS : Japanese Industrial Standards

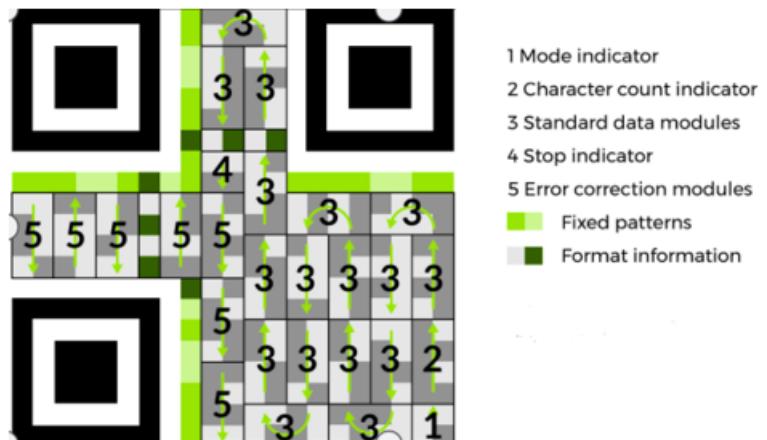


Figure 9.9.: Pattern in QR Code



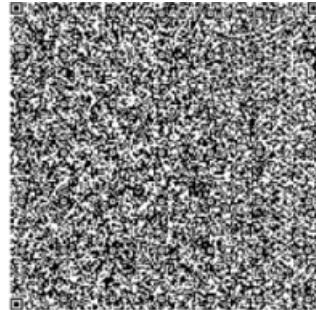
Version 1
21x21 Modules

Figure 9.10.: Version 1



Version 10
57x57 Modules

Figure 9.11.: Version 10



Version 40
177x177 Modules

Figure 9.12.: Version 40

	QR Code®	DataMatrix	MaxiCode	PDF417	Aztec
Developer(country)	Denso Wave (Japan)	CI Matrix (U.S.)	UPS (U.S.)	Symbol (U.S.)	Welch Allyn (U.S.)
Data size	Numerical	7,089	3,116	138	2,710
	Alpha-numerical	4,296	2,355	93	1,850
	Binary	2,953	1,556	—	1,108
	Kanji	1,817	778	—	554
Main features	Large capacity Space saving High-speed reading	Space saving	High-speed reading	Large capacity	High-speed reading
Main uses	All fields	FA	Logistics	OA	Transportation
Transportation	ISO JIS AIM International	ISO AIM International	ISO AIM International	ISO AIM International	ISO AIM International

Figure 9.13.: Type of 2D Code

9.4. Software

The software used in this Home-work are:

- python 3.10
- pycharm 2021.2.3

9.5. Hardware

The hardware used is

- Processor Intel Core i5
- Camera integrated and external cam
- Computer screen

9.6. function

- Decodes Qr code
- Decode Barcode

9.7. Limitations

The program is capable of decoding Barcode and QrCodes but other type of 2-D barcodes can also be decoded if the improvment should be made in the progarm code. Light on the code should also be good so the code is visble to the camera

9.8. output

The output we get through this program is the information encoded in the Barcodes and Qrcodes for eg. equipment or part specification in industry or website url, as shown in fig 9.14

9.9. How to Run

We need Pycharm and Python installed on the system for runing the code correctly we have to install Opencv, Numpy and Pyzbar libraries then the code is executed. Camera can detect all the barcodes and Qr code decode them present in the visible range as shown in figures.9.159.16

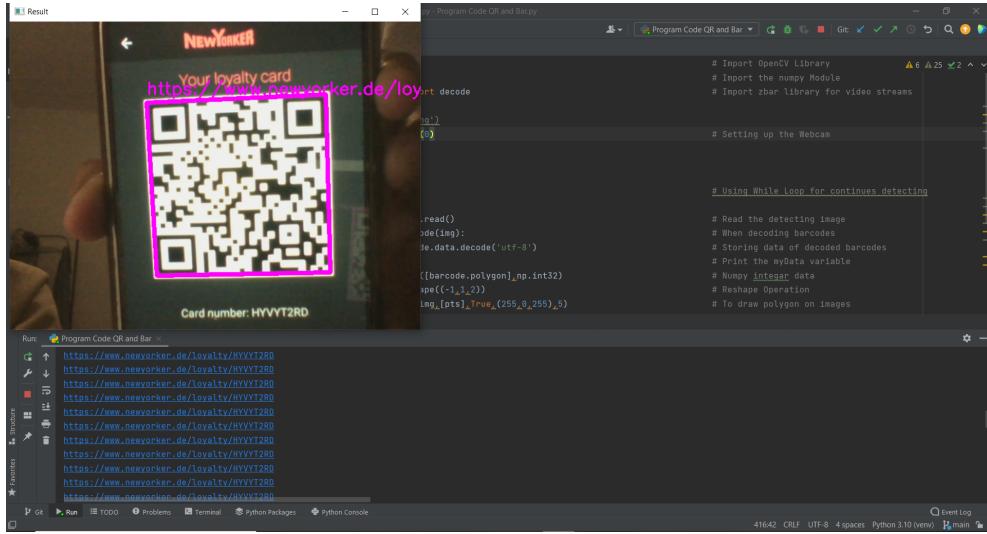


Figure 9.14.: 1-D Linear Bar Code

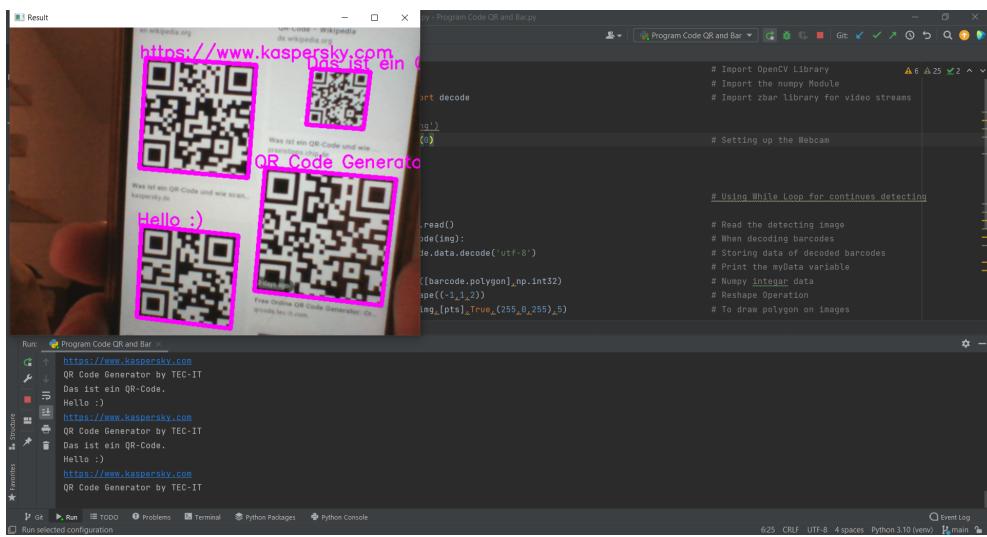


Figure 9.15.: 1-D Linear Bar Code

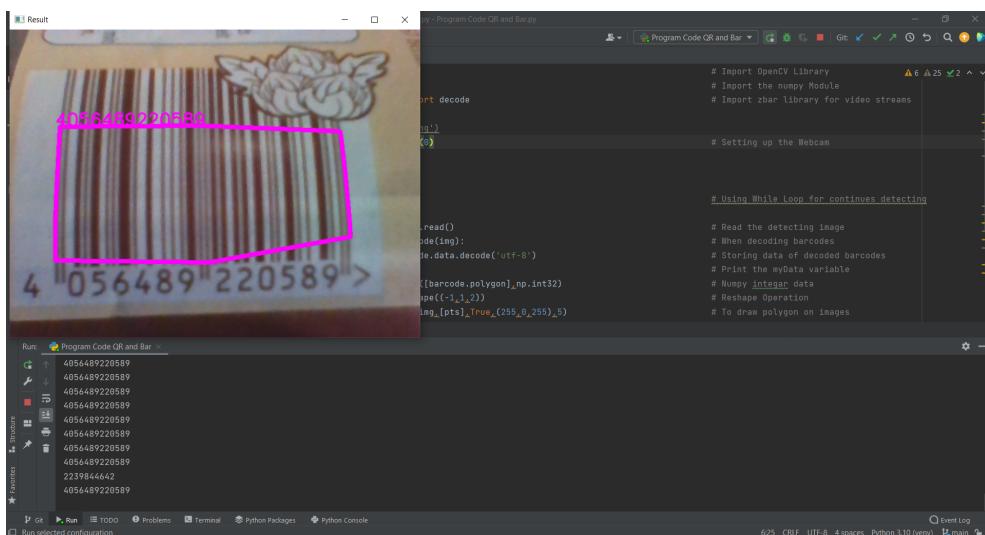


Figure 9.16.: 1-D Linear Bar Code

10. EasyOCR

10.1. Introduction

Automatic License Plate Detection and Recognition (ALPDR) uses EasyOCR for transforming the raw images of LP. After detecting the area of interest from car images OCR is used in the segmentation of characters. It isolates the character and number from the LP and recognizes the whole sequence of characters.

EasyOCR breakdowns the blobs to locate the character and their bounding box, using one of two segmentation modes:

1. Keep object mode: One blob corresponds to one character
2. Repaste objects mode: The blob is grouped into characters of nominal size. When a blob is big it can be split automatically.

Easy OCR processes the character images to normalize the size into a bounding box, extract relevant feature and stores them in a font file. The patterns in a form are stored as arrays of pixels defined by pattern width and pattern height

10.2. Description

Hyperparameter for EasyOCR Segmentation: [Ope18]

1. **Threshold:** A too-high value thickens black characters on a white background and may cause merging, a too-small value makes parts disappear.
2. **NoiseArea:** Blob areas smaller than this value are discarded. Make sure small character features are preserved.
3. MaxCharWidth,MaxCharHeight: Maximum character size. If a blob does not fit in a rectangle with these dimensions, it is discarded or split into several parts using vertical cutting lines.
4. **CharSpacing:** The width of the smallest gap between adjacent letters. If it is larger than MaxCharWidth it has no effect. If the gap between two characters is wider than this, they are treated as different characters. This stops thin characters being incorrectly grouped together.
5. **RemoveBorder:** Blobs near image/ROI edges cannot normally be exploited for character recognition.

Hyperparameters for EasyOCR Recognition: [Ope18]

1. Load: reads a pre-recorded font from a disk file.
2. BuildObjects: The image is segmented into objects or blobs (connected components) which help find the characters. This step can be bypassed if the exact position of the characters is known. If the character isolation process is bypassed, you must specify the known locations of the characters: AddChars and EmptyChars.

3. `FindAllChars`: selects the objects considered as characters and sorts them from top to bottom then left to right.
 4. `ReadText`: performs the matching and filters characters if the marking structure is fixed or a character set filter was provided.
 5. Character recognition: The characters are compared to a set of patterns, called a font.
 6. The best match is stretched to fit in a predefined rectangle and compared to each normalized template in the font database.
 7. A Character set filter can improve recognition reliability and run time by restricting the range of characters to be compared. For instance, if a marking always consists of two uppercase letters followed by five digits, the last of which is always even, it is possible to assign each character a class (maximum 32 classes) then set the character filter to allow the following classes at recognition time: two uppercase, four even or odd digits, one even digit.

10.3. Installation

Prerequisite and installation of EasyOCR:

PyTorch is a core dependency for installing the EasyOCR.

PyTorch 1.9.0 stable version for Windows is installed. A local python interpreter pip package is used. Conda package is used for python virtual interpreter. For GPU enabled system CUDA 10.2 compute platform is used.

After having above configuration following command can be used for installation of PyTorch(1.9.0)

```
conda install pytorch torchvision torchaudio cudatoolkit-10.2 -c pytorch
```

After installing the PyTorch library easyocr is installed with following steps

```
pip3 install easyocr
```

```
D:\Computer Vision Project\EasyOCR\EasyOCR-main>pip install easycv
Collecting easycv
  Downloading easycv-1.3.2-py3-none-any.whl (63.2 kB)
    63.2 kB / 60 kB
Collecting python-bidi
  Downloading python-bidi-0.4.2-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from easycv) (1.5.2)
Requirement already satisfied: torchvision<0.5 in c:\programdata\anaconda3\lib\site-packages (from easycv) (0.10.0)
Requirement already satisfied: opencv-python<=4.5.1 in c:\programdata\anaconda3\lib\site-packages (from easycv) (4.5.1.48)
Requirement already satisfied: numba in c:\programdata\anaconda3\lib\site-packages (from easycv) (1.19.2)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from easycv) (1.19.2)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site-packages (from easycv) (8.0.1)
Requirement already satisfied: PyYAML in c:\programdata\anaconda3\lib\site-packages (from easycv) (5.3.1)
Requirement already satisfied: torch in c:\programdata\anaconda3\lib\site-packages (from easycv) (1.9.0)
Requirement already satisfied: Pillow in c:\programdata\anaconda3\lib\site-packages (from easycv) (8.0.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from python-bidi>easycv) (1.15.0)
Requirement already satisfied: scikit-image in c:\programdata\anaconda3\lib\site-packages (from scikit-image>easycv) (3.3.2)
Requirement already satisfied: imageio in c:\programdata\anaconda3\lib\site-packages (from imageio>easycv) (2.5.0)
Requirement already satisfied: imageio-ffmpeg in c:\programdata\anaconda3\lib\site-packages (from imageio>easycv) (2.5.0)
Requirement already satisfied: imageio-v2 in c:\programdata\anaconda3\lib\site-packages (from scikit-image>easycv) (2.9.0)
Requirement already satisfied: tifffile<2020.10.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-image>easycv) (2020.10.1)
Requirement already satisfied: PyWavelets<1.1.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-image>easycv) (1.1.1)
Requirement already satisfied: typing-extinction in c:\programdata\anaconda3\lib\site-packages (from torch>easycv) (3.7.4..3)
Requirement already satisfied: kivy in c:\programdata\anaconda3\lib\site-packages (from easycv>easycv) (2.0.0)
Requirement already satisfied: scikit-image<0.4.1,>=0.4 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>0.3.0,>=0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>easycv) (2.4.7)
Requirement already satisfied: cyclers<0.18 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>0.3.0,>=0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>easycv) (0.18.0)
Requirement already satisfied: certifi<2020.06.20 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>0.3.0,>=0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>easycv) (2020.6.20)
Requirement already satisfied: python-dateutil<2.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>0.3.0,>=0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>easycv) (2.8.1)
Requirement already satisfied: decorator<4.3.0 in c:\programdata\anaconda3\lib\site-packages (from networkx>0.3.0 in c:\programdata\anaconda3\lib\site-packages (from easycv) (4.4.2))
Successfully installed easycv-1.3.2 python-bidi b64cv
Successfully installed easycv-1.3.2 python-bidi b64cv
```

Figure 10.1.: EasyOCR Installation using Command Prompt

10.4. Example

Sample Code ALDPR

Sample code for EasyOCR:

```
1000 # function to recognize license plate numbers using Tesseract OCR
1002 def recognize_plate_eeasyocr(img, coords, reader, region_threshold):
```

```

1004 # separate coordinates from box
1006 xmin, ymin, xmax, ymax = coords
1008 # get the subimage that makes up the bounded region
# take an additional 5 pixels on each side
1010 # nplate = img[int(ymin)-5:int(ymax)+5, int(xmin)-5:int(xmax)+5]
1012 nplate = img[int(ymin):int(ymax), int(xmin):int(xmax)]
1014 # cropping the number plate from the whole image
1016
1018 ocr_result = reader.readtext(nplate)
1020
1022 text = filter_text(region=nplate, ocr_result=ocr_result,
region_threshold=region_threshold)
1024
1026 if len(text) ==1:
1028 text = text[0].upper()
1030 return text

```

Example EasyOCR

```

img = cv2.imread(IMAGE_PATH)
spacer = 100
for detection in result:
    top_left = tuple(detection[0][0])
    bottom_right = tuple(detection[0][2])
    text = detection[1]
    img = cv2.rectangle(img,top_left,bottom_right,(0,255,0),3)
    img = cv2.putText(img,text,(20,spacer), font, 0.5,(0,255,0),2,cv2.LINE_AA)
    spacer+=15

plt.figure(figsize=(10,10))
plt.imshow(img)
plt.show()

```

Figure 10.2.: Code Snippet

In the above code snippet, we just need to focus on few points:

1. Instead of detecting one-line text, here we are looping through all the detection as we want to plot multiple lines of text.
2. While giving the coordinates on cv2.putText we are using an extra variable which is “spacer” this spacer later in the code is being incremented to +15 which is helping to restrict the text to collide over each other.
3. This spacer variable will help the text to remain sorted and equally spaced



Figure 10.3.: Multi-Line recognition using EasyOCR

10.5. Further Reading

Challenging and Invalid recognition settings:

1. MaxCharWidth, MaxCharHeight: if a blob does not fit within a rectangle with these dimensions, it is not considered a possible character (too large) and is discarded. Furthermore, if several blobs fit in a rectangle with these dimensions, they are grouped, forming a single character. The outer rectangle size should be chosen such that it can contain the largest character from the font, enlarged by a small safety margin.
2. MinCharWidth, MinCharHeight: if a blob or a group of blobs does fit in a rectangle with these dimensions, it is not considered a possible character (too small) and is discarded. The inner rectangle size should be chosen such that it is contained in the smallest character from the font, shrunk by a small safety margin.
3. RemoveNarrowOrFlat: Small characters are discarded if they are narrow or flat. By default, they are discarded when they are both narrow and flat.
4. CharSpacing: if two blobs are separated by a vertical gap wider than this value, they are considered to belong to different characters. This feature is useful to avoid the grouping of thin characters that would fit in the outer rectangle. Its value should be set to the width of the smallest gap between adjacent letters. If it is set to a large value (larger than MaxCharWidth), it has no effect.

5. CutLargeChars: when a blob or grouping of blobs is larger than MaxCharWidth, it is discarded. When enabled, the blob is split into as many parts as necessary to fit and the amount of white space to be inserted between the split blobs is set by RelativeSpacing. This is an attempt to separate touching characters.
6. RelativeSpacing: when the CutLargeChars mode is enabled, setting this value allows specifying the amount of white space that should be inserted between the split parts of the blobs.

10.5.1. Advanced Tuning

These recognition parameters can be tuned to optimize recognition:

- CompareAspectRatio: when this setting is on, EasyOCR is less tolerant of size and takes into account the measured aspect ratio. Using this mode improves the recognition when characters look similar after size normalization as it enforces the difference between narrow and wide characters.
- Filtering the characters can be used if the marking structure is fixed.
- When objects are larger than the MaxCharWidth property, they can be split into as many parts as needed, using vertical cutting lines.
- ESegmentationMode, character isolation mode defines how characters are isolated:
Keep objects mode: a character is a blob

11. YOLO v5

11.1. Introduction

YOLOv5 is a family of compound-scaled object detection models trained on the COCO dataset, and includes simple functionality for Test Time Augmentation (TTA), model ensembling, hyperparameter evolution, and export to ONNX, CoreML and TFLite. Yolov5 and CNN use extracted features from the input image by all the convolutional layers in the detection module. As the number of layers increases, the number of channels increases and the size of the feature map decreases progressively. [RF16] The later feature map has higher-level features extracted and thus is more beneficial for recognizing the LP and predicting its area of interest box. [HLT09]

11.2. Description

Hyperparameter for YOLOv5:[Ope18]

1. **Learning-rate start (lr0):** At each iteration, the learning rate begins to determine the step size. For example, if you set the learning rate (0.1) before training, your training progress will rise by 0.1 at each repetition.
2. **Learning-rate end (lr1):** Learning rate end, which is used to test the following condition,
3. **Momentum:** Momentum is the gradient descent algorithm's tuning parameter; its job is to replace the gradient with an aggregation of gradients.
4. **Mosaic:** Mosaic is used to improve model accuracy by combining many photos to generate a single image, which is then utilized for training. It is well-known for data augmentation and the discovery of optimum feature approaches.
5. **Degree:** Degree is used to improve model accuracy by randomly rotating pictures in the training data up to 360 degrees. This option is especially useful for detecting objects in numerous positions/angles.
6. **Scaling:** Scaling is used to shrink a picture to fit the grid size or to optimize the results.
7. **flipud:** flipud is used to randomly flip photos up and down from the whole dataset in order to achieve better results. This may be taken into account in the data augmentation approach.
8. **fliplr:** fliplr is used to randomly flip photos left and right from the whole dataset in order to achieve better results. This may be taken into account in the data augmentation approach.
9. **Weight-decay:** A regularization approach that works by adding a penalty term to a network's cost function, causing the weights to shrink/compress throughout the backpropagation process. There are several more parameters, such as warmup epochs, warmup momentum, and so on, whose values may be modified according on the use case.

11.3. Installation

Prerequisite and installation of YOLOv5:

PyTorch is a core dependency for installing the YOLOv5.

PyTorch 1.9.0 stable version for Windows is installed. A local python interpreter pip package is used. Conda package is used for python virtual interpreter. For GPU enabled system CUDA 10.2 compute platform is used.

After having above configuration following command can be used for installation of PyTorch (1.9.0).

```
conda install pytorch torchvision torchaudio cudatoolkit-10.2 -c pytorch
```

After installing the PyTorch library YOLOv5 is installed with following steps:

YOLO V5 is cloned from <https://github.com/ultralytics/yolo5> by running the command

```
1000 ! git clone https://github.com/ultralytics/yolo5.  
##This will create the folder structure for YOLO V5.##
```

Following is the folder structure for YOLOv5:

- `/input/data/prepro/`
- `/images/*.png`
- `/train`
- `/validation`
- `/label/*.xml`
- `/Dataset/`
- `/train/images`
- `/train/images`

11.4. Example

Sample training for YOLOv5

Sample code for YOLOv5:

```
1000 ! python train.py --img 416 --batch 16 --epochs 300 --data data/alpr.yaml  
--cfg models/yolov5s.yaml
```

Dataset is divided into training and validation datasets. 80% of the data for training and 20% data for validation. YOLOv5 first exploits the Box Regression layer to predict the bounding box. Then, referring to the relative position of the bounding box in each feature map, EasyOcr extracts the region of interest from several already rendered feature maps, merge them after pooling and feeds the combined features maps to the subsequent Classifiers.

Images that are used for modelling are first converted from BGR to RGB. These processed images are passed to the detector. The detector will plot the boxes around the LP. The region of boxes will be called the area of interest. These functions After getting results, frames and classes, filter out the boundary boxes based on the probability threshold. All the boundary boxes which have a threshold greater than 0.55 are considered for further processing.



Figure 11.1.: YOLOv5 Result

12. Description of the Python Package: Keras

For this project and in general for Machine Learning projects one of the most common packages required to understand for the development in python is Keras. In general, this is used to develop high-level neural networks, it is written in Python and is capable of running on top of TensorFlow. In the specific case of this project, Keras is been used in two stages, with the following tools:

- **ImageDataGenerator**, this tool provides a convenient and powerful way to load, pre-process, and augment image data for use in deep learning models. It allows to easily load image data from a directory structure, the application of pre-processing techniques, and the use of data augmentation to improve model performance.
- **Layers**, this tool is an essential component of a neural network model generation with Keras Package in python. A directed acyclic graph of the neural network is built using layers, where the output of one layer serves as the input for the following layer. Convolutional layers for image processing, recurrent layers for sequence data, and fully connected layers for dense neural networks are just a few of the pre-built layers available in the Keras framework. In the particular instance of this project, the following layers are mostly used for the implementation:
 - **Dense**, is also known as fully connected layer. It links all the neurons from the layer below to the neurons in the layer above. A hyper-parameter that has to be determined is how many neurons are present in the dense layer. In this layer, the activation function can also be specified.
 - **Conv2D**, is used for image processing. The input data is subjected to a convolution procedure, enabling the model to learn spatial hierarchies of features. The hyper-parameters of this layer may be categorized as the number of filters, kernel size, and strides.
 - **MaxPool2D**, is used for down-sampling the input data. It decreases the spatial dimensions of the data by applying a max pooling operation to the input data. The hyper-parameters of this layer may be categorized as the pooling size and the strides.
 - **Flatten**, is used for flattening the multi-dimensional input data into a 1D array. It is frequently applied before the last dense layer in a model so that it can process the incoming data.

Building a variety of neural network models for image classification and other image processing tasks, such object recognition and semantic segmentation, may be done by combining the Dense, Conv2D, MaxPool2D, and Flatten layers. To test out several designs and identify the one that best matches the data, the number and mix of these layers may be altered.

Keras' simplicity and ease of use are two of its key benefits. It abstracts away a significant portion of the complexity involved in creating and refining deep learning models, allowing developers to concentrate on the model's architecture and design rather than the specifics of its implementation.

Support for several back-ends, such as TensorFlow, Theano, and Microsoft Cognitive Toolkit, is another benefit of Keras (CNTK). As a result, programmers may create

models and train them using any back end of their choosing, switching back-ends without having to alter the model's architecture or training code.

Also it is a simple package to install and use in Python, in order to do this, the following steps must be followed:

1. Verify that Python is installed. This is done by running in the command prompt:
`python -V`.
2. Install the required dependencies for Keras; numpy, scipy, and six. Run the following command in the command prompt: `pip install numpy scipy six`.
3. Install TensorFlow or one of the other supported backends. Keras is a high-level library that runs on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Run the following command: `pip install tensorflow`.
4. Install Keras by running the following command: `pip install keras`.

In order to test that Keras is properly installed, use this code:

```
1000 import keras;
      print(keras.__version__)
```

As of now, the version of Keras being used is 2.9.0. Now lets present an example of how to use Keras in a simple python example:

Lastly it is important to reference the developer source for Keras releases (past, current and next). In this GitHub repository one can access more detailed information regarding this package:

Keras: Deep Learning for humans

```
1000 # Keras simple code example
1002
1003 # 1.1 Imports
1004 import keras
1005 from keras.datasets import mnist
1006 from keras.utils import np_utils
1007 from keras.models import Sequential
1008 from keras.layers import Dense
1009
1010 print(keras.__version__)
1011
1012 # 1.2 Load the MNIST dataset
1013 (X_train, y_train), (X_test, y_test) = mnist.load_data()
1014
1015 # 1.3 Normalize the input data
1016 X_train = X_train.astype('float32') / 255
1017 X_test = X_test.astype('float32') / 255
1018
1019 # 1.4 Convert the labels to categorical
1020 num_classes = 10
1021 y_train = np_utils.to_categorical(y_train, num_classes)
1022 y_test = np_utils.to_categorical(y_test, num_classes)
1023
1024 # 1.5 Create the model
1025 model = Sequential()
1026 model.add(Dense(512, activation='relu', input_shape=(784,)))
1027 model.add(Dense(num_classes, activation='softmax'))
1028
1029 # 1.6 Compile the model
1030 model.compile(loss='categorical_crossentropy',
1031                 optimizer='adam',
1032                 metrics=['accuracy'])
1033
1034 # 1.7 Train the model
1035 history = model.fit(X_train.reshape(60000, 784), y_train, batch_size=128,
1036                       epochs=5, verbose=1,
1037                       validation_data=(X_test.reshape(10000, 784), y_test))
```

..../Code/General/Keras/KerasSample.py

Listing 12.1.: Simple example for Keras

13. Description of the Python Package: NumPy

Numerical Python (NumPy) is one of the most fundamental packages available in python for numerical computation. It is a general-purpose array-processing package which provides high-performance multidimensional arrays and appropriate tools to work with them. NumPy is capable of storing generic multi-dimensional data. In NumPy, dimensions are known as axes and the rank denotes the number of axes present. NumPy's array class is termed as ndarray.

13.1. Functions:

- Computes basic array operations such as addition, multiplication, slicing, flatten, reshape and array indexing.
- Computes advanced array operations such as stacking arrays, splitting into sections and broadcasting arrays
- NumPy works with either Date Time or Linear Algebra

13.2. Features:

- Provides pre-compiled functions for numerical routines
- Array-oriented computing for better efficiency
- NumPy supports an object-oriented approach
- It is compact and performs faster computations with vectorization

13.3. Applications:

- Predominantly used in data-analysis applications.
- Used for creating powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Used as a replacement of MATLAB when used with SciPy and matplotlib

13.4. Installation

To install Numpy, you can use the pip package manager by running the command `pip install numpy` in your terminal.

13.5. Further Reading:

- The Numpy documentation (<https://numpy.org/doc/>) provides a comprehensive overview of the library's features and usage.
- The Scipy website (<https://scipy.org/>) also provides additional resources for scientific computing with Python, including tutorials and a user guide.

```
1000 # Numpy Code Example
1002 #
1003 # Here is a simple example of how to use Numpy to create
1004 # an array and perform basic operations on it
1005 #
1006 import numpy as np
1008
1009 # Version Check:
1010 print(numpy.__version__)
1011
1012 # Example 1
1013
1014 # Create a 1-dimensional array
1015 arr = np.array([1, 2, 3, 4])
1016
1017 # Perform operations on the array
1018 print(arr + 1) # [2, 3, 4, 5]
1019 print(arr * 2) # [2, 4, 6, 8]
1020
1021 # Example 2:
1022
1023 #
1024 # Here is another example of how to use Numpy to create
1025 # a 2-dimensional array and perform more
1026 # advanced operations on it
1027 #
1028 import numpy as np
1029
1030 # Create a 2-dimensional array
1031 arr = np.array([[1, 2, 3], [4, 5, 6]])
1032
1033 # Perform operations on the array
1034 print(arr.shape) # (2, 3)
1035 print(arr.mean()) # 3.5
1036
1037 # Example - Files
1038
1039 #
1040 # You can save and load numpy array into a file with numpy functions
1041 # such as follows: "numpy.save" and "numpy.load"
1042 #
1043
1044 arr = np.array([1, 2, 3, 4])
1045 np.save('my_array', arr)
1046 loaded_array = np.load('my_array.npy')
```

..../Code/General/NumPy/NumPySample.py

Listing 13.1.: Simple example for NumPy

14. Description of the Python Package: Matplotlib

Matplotlib is an open-source drawing library which has powerful and wide range of visualizations. It extensively provides an object-oriented API which is used for embedding plots into applications. With the visualization capabilities of Matplotlib one can visualise everything that can be drawn from legends and grids to spectrogram.

14.1. Functions:

With few lines of code, Matplotlib can depict a wide range of visualizations such as:

- Line plots
- Scatter plots
- Area plots
- Bar charts and Histograms
- Pie charts
- Stem plots
- Contour plots
- Quiver plots
- Spectrograms

14.2. Features:

- With the advantage of being open source, it can be used as a replacement for MATLAB.
- Supports dozens of backends and output types, which enables one to use it regardless of type of the operating system used or the output format used.
- Good runtime behavior and low memory consumption.

14.3. Applications:

- Used for correlation analysis of variables.
- Used for visualizing 95 percent confidence intervals of the models.
- Outliers can be detected using a scatter plot.
- Can be used for visualizing the distribution of data to obtain instant insights.

```

1000 # Matplotlib Code Example
1002 import matplotlib.pyplot as plt
1003 import numpy as np
1004
1005 # Example 1
1006
1007 #
1008 # Here is a simple example of how to use Matplotlib to create
1009 # an array and perform basic operations on it
1010 #
1011
1012 plt.plot([1, 2, 3, 4])
1013 plt.ylabel('some numbers')
1014 plt.show()
1015
1016 # Example 2:
1017
1018 #
1019 # Here's a more complex example of how to plot
1020 # multiple lines on the same plot:
1021 #
1022 t = np.arange(0., 5., 0.2)
1023 plt.plot(t, t, 'r—', t, t**2, 'bs', t, t**3, 'g^')
1024 plt.show()
1025
1026 # Matplotlib — Version Check:
1027 # You can check the version of Matplotlib by using the following command
1028 :
1029 #
1030 print(matplotlib.__version__)
1031
1032 # Example — Files:
1033
1034 # Matplotlib can also save plots to various file formats,
1035 # such as PNG, PDF, SVG, and more. Here's
1036 # an example of how to save a plot as a PNG file:
1037 #
1038 plt.plot([1, 2, 3, 4])
1039 plt.ylabel('some numbers')
1040 plt.savefig('output.png')

```

..../Code/General/MatPlotLib/MatPlotLibSample.py

Listing 14.1.: Simple example for MatPlotLib

14.4. Installation

Matplotlib can be installed using pip, conda, or by downloading the source code and running `setup.py`. It is recommended to install matplotlib through Anaconda distribution or using the command `pip install matplotlib`.

14.5. Further Reading:

- For more information on matplotlib, check out the official matplotlib documentation at <https://matplotlib.org/>.

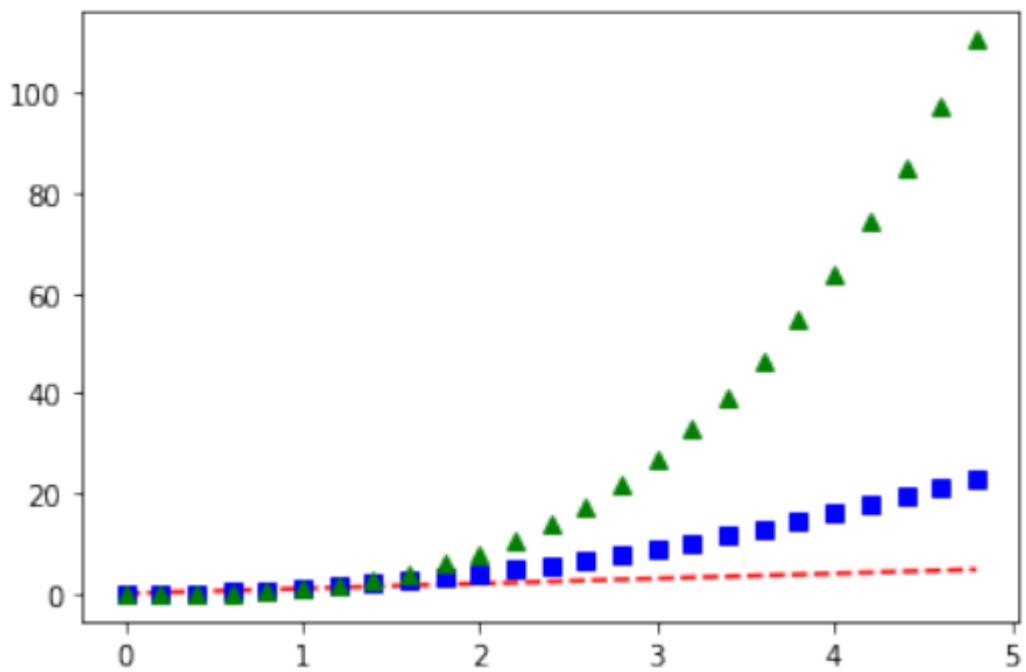
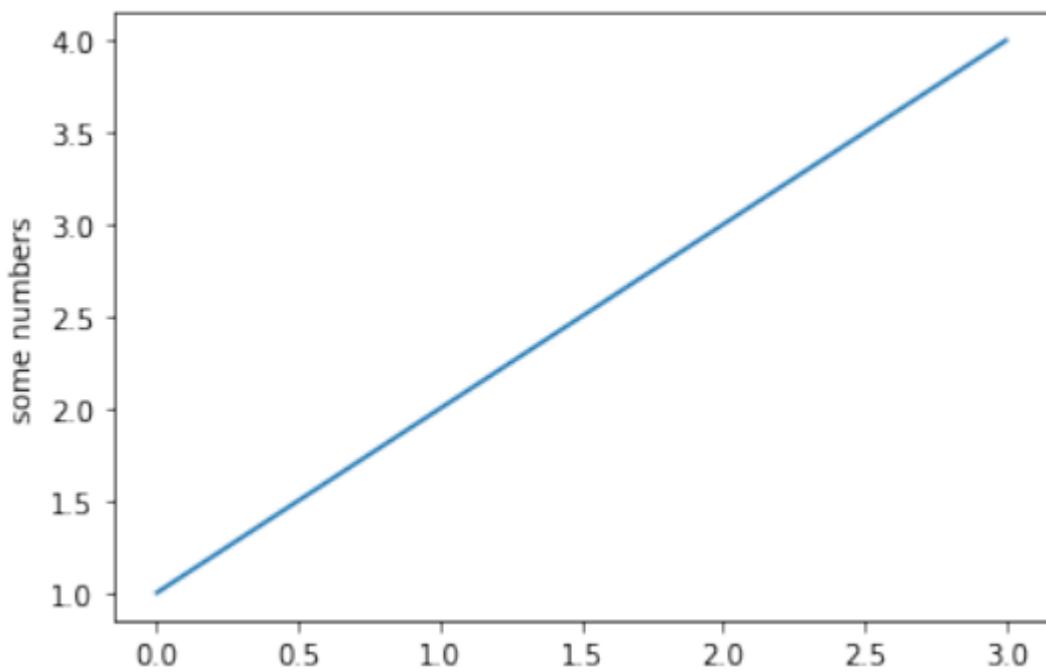


Figure 14.1.: Output of the example code

Figure 14.2.: Output of the 2nd example code

Part III.

Add Ons

15. Programmieren mit Python: Einfache grafische Oberfläche mit Tkinter erstellen

Fenster, Buttons und Menüs machen ein Python-Tool für Nutzer viel attraktiver. Mit der Bibliothek [Tkinter](#) geben Sie etwa einem Login-Programm ein Gesicht.

Für den Ersteller eines Python-Skripts ist es oft intuitiv, mit der Kommandozeile zu arbeiten. Aber sobald man das Skript weitergibt und andere Leute damit arbeiten, kommt schnell der Wunsch nach einer grafischen Bedienoberfläche auf. Normale Nutzer sind es gewohnt, Texte in Felder einzugeben, auf Buttons zu klicken und ein Menü am oberen Rand des Programms zu öffnen. Weißer Text auf einem schwarzen Hintergrund wirkt da wie ein Fremdkörper.

In diesem Beispiel versehen Sie ein simples Login-Tool mit einer Bedienoberfläche. Der Nutzer soll seinen Benutzernamen und sein Passwort eingeben und auf einen Login-Button klicken können. Das Programm sagt ihm dann, ob die Daten korrekt waren oder nicht. Eine Menüleiste rundet das Programm ab. So lernen Sie alle wichtigen Grundlagen und Konzepte von Tkinter kennen.

Tkinter ist in Python bereits integriert und bildet die Schnittstelle der Programmiersprache zu Tk, einem kostenlosen Toolkit, um grafische Bedienoberflächen zu gestalten. Tk wurde Ende der 1980er Jahre für die Skriptsprache Tcl entwickelt.

Tkinter in Ihr Python-Programm zu holen ist daher recht einfach:

```
import tkinter as tk
```

Sie müssen vorab keine Pakete mit dem Verwaltungsprogramm pip installieren. Achten sie darauf, [tkinter](#) klein zu schreiben, wenn Sie mit der neusten Python-Version arbeiten: Bei Python 2 hieß das Paket noch [Tkinter](#), ab Python 3 heißt es [tkinter](#). Wir importieren die Bibliothek in diesem Beispiel mit dem Zusatz [as tk](#), um mögliche Verwirrungen zu vermeiden und Tkinter einen eigenen Namensraum zu geben. Die Tkinter-Funktionen werden daher mit einem [tk.](#) eingeleitet.

15.1. Tk gegen Qt

Neben Tk könnte man bei Python unter anderem auf das beliebte Framework Qt für eine Bedienoberfläche setzen. Für die Python-Integration sind etwa externe Bibliotheken wie PyQt oder PySide verantwortlich. Qt ist beliebt und es gibt es Dutzende Tutorials im Netz, mit denen komplexe Fensterlayouts möglich sind.

Wir haben uns für dieses Projekt trotzdem für Tk und damit für Tkinter entschieden, weil ein Login-Fenster ein überschaubares Programm ist. Tkinter steht ohne große Vorbereitungen für jedes Python-Projekt bereit. Wer ein Kommandozeilentool schnell mit einer Oberfläche versehen will, für den reicht Tkinter aus – etwa weil die Anwender meckern und Fenster mit Buttons fordern.

15.2. Erstes Fenster erstellen

Ein erstes Fenster können Sie mit Tkinter in zwei zusätzlichen Zeilen erstellen:

```
programm = tk.Tk()
```

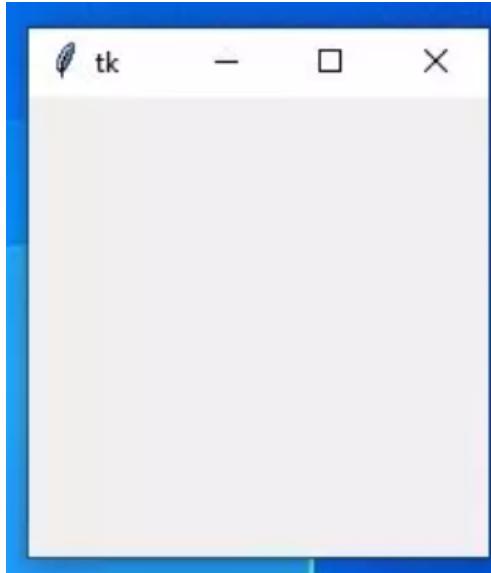


Figure 15.1.: Das erste Fenster ist grau und unspektakulär. Aber ein Fenster.

`programm.mainloop()`

`tk.Tk()` erstellt quasi das Hauptfenster des Programms und initialisiert das gesamte Tk-Framework. Mit `mainloop()` schaffen Sie eine Schleife, in der alle Ereignisse verarbeitet werden, die im Fenster passieren. Das Programm hält diese Schleife solange aufrecht, bis der Anwender das Hauptfenster schließt. Die Funktion `mainloop()` sollte immer am Ende stehen, da sie das aufruft, was Sie vorher gecodet haben.

Das erste Fenster, siehe 15.1, ist mit diesen zwei Zeilen natürlich noch etwas unspektakulär. Graue Fläche, eine Feder als Icon, die an das Tk-Symbol erinnert, tk als Fenstertitel und Schaltflächen fürs Minimieren, Maximieren und Schließen. Aber es ist schon jetzt für viele Anwender der vertraute Look eines Programmfensters.

Nun können Sie ein einfaches Hello-World-Widget hinzufügen. Widgets sind bei Tkinter die Steuerungselemente wie Labels, Buttons, Skalen, Checkboxen, Optionen und viele mehr. Sie wollen, dass im Fenster der Text Hello World! steht. Das funktioniert mit dem Label-Widget, es zeigt Texte und Bilder an. Erst definieren Sie das Label mit `Label()`:

```
label = tk.Label(programm, text="Hello World!")
```

Der erste Parameter ist das Elternwidget, in diesem Fall das Hauptfenster. Hinter dem Parameter `text` steht der Text in Anführungszeichen, den das Label anzeigen soll. Damit haben Sie das Label erstellt, nun müssen Sie es noch platzieren:

```
label.pack()
```

Tkinter kennt drei Möglichkeiten, Widgets zu platzieren: `pack()`, `grid()` und `place()`:

- `pack()` positioniert alle Elemente automatisch relativ zueinander.
- `grid()` ordnet die Widgets in einem zweidimensionalen Gitter mit Reihen und Spalten an.
- `place()` positioniert alle Elemente absolut oder relativ.

Alle drei Methoden sollten nie vermischt werden, Sie sollten sich also vorher für eine Methode entscheiden. `pack()` ist die einfachste Möglichkeit, daher verwenden wir

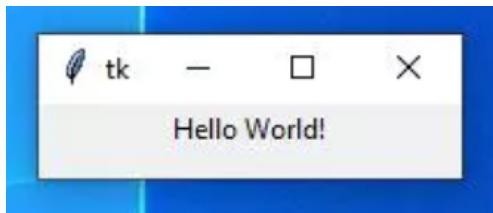


Figure 15.2.: Hallo Welt! Das war nicht schwer.

```

1000 # Tkinter Hello World
1002 import tkinter as tk
1004 programm = tk.Tk()
1006 label = tk.Label(programm, text="Hello World!")
1007 label.pack()
1008 programm.mainloop()

..../code/General/TkinterHelloWorld.py

```

Listing 15.1.: Erstes Fenster "Hello World"

sie hier für das Hello-World-Fenster, siehe 15.2. Für das echte Programm nutzen wir später `grid()`, da sich damit die Widgets genauer positionieren lassen.

Und damit haben Sie Ihr erstes Hello-World-Fenster erstellt und das in nur fünf Zeilen Code

[refTkinterHelloWorldCode](#) [TkinterHelloWorldCode](#).

15.3. Hauptfenster erstellen

Nach diesen Grundlagen gehts nun weiter mit dem echten Programm, einem simplen Login-Fenster für das Einloggen bei heise online. Damit Sie den Überblick behalten, hat es sich bei Tkinter eingebürgert, einzelne Elemente in Klassen zu packen, so auch das Hauptfenster, siehe 15.2.

`self` bezieht sich immer auf die Instanz einer Klasse, nicht auf die generelle Klasse. Andere Programmiersprachen geben sowas als versteckten Parameter mit, Python nicht. Sie müssen es nicht unbedingt `self` nennen, das Keyword ist aber eine sehr gebräuchliche Konvention. Um Ihren Code für andere lesbar zu gestalten, sollten Sie sich daran halten. `parent` bezeichnet das übergeordnete Element. Da es in diesem Fall

```

1000 programm=Programm(None)
1001 programm.mainloop()

..../code/General/Tkinter/TkinterClass.py

```

```

1000 class Programm(tk.Tk):
1001     def __init__(self, parent):
1002         tk.Tk.__init__(self, parent)
1003         self.parent = parent

..../code/General/Tkinter/TkinterClass.py

```

Listing 15.2.: Erstellung eines Hauptfensters

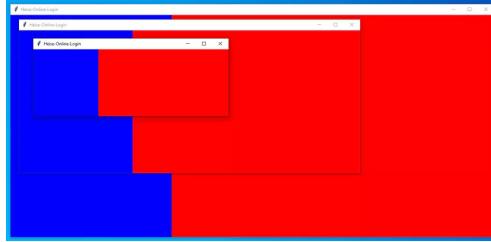


Figure 15.3.: Das Gewicht der Spalten beeinflusst ihre Größe.

das Hauptfenster ist, initialisieren Sie es mit None, es gibt schlicht kein übergeordnetes Element.

`__init__` ist in Python eine reservierte Methode. Sie wird dafür genutzt, ein Objekt zu initialisieren, daher der Name. `__init__` ist ähnlich wie ein Konstruktor in C++ oder Java.

In der Methode `__init__` können Sie einige Eigenschaften des Hauptfensters festlegen. Den Titel des Fensters ändern Sie etwa mit einem `self.title("Heise-Online-Login")`. Für die Mindestgröße eines Fensters in Pixeln ist `minsize()` verantwortlich:

```
self.minsize(300, 130)
```

Der erste Wert legt die Länge fest, der zweite Wert die Höhe. Möchten Sie nicht, dass der Nutzer die Größe des Fensters verändert, schreiben Sie `self.resizable(width=False, height=False)` in die Methode. `width` ist die Länge, `height` die Höhe, `False` beantwortet die Frage, ob der Nutzer Höhe und Länge ändern kann mit Falsch. Weitere Tk-Methoden, die das Hauptfenster beeinflussen, finden Sie online.

15.4. Gitter erstellen

Nun konfigurieren Sie das Gitter des Hauptfensters. In diesem Gitter platzieren Sie alle Unterelemente. Das Hauptfenster soll in diesem Beispiel nur als Container dienen, um etwa Frames anzuzeigen, die komplexere Layouts enthalten. Daher konfigurieren Sie nur ein Gitter mit einer Reihe und einer Spalte:

```
self.grid_columnconfigure(0, weight=1)
self.grid_rowconfigure(0, weight=1)
```

`grid_columnconfigure()` ist für die Spalte verantwortlich, `grid_rowconfigure()` für die Reihe. Der erste Parameter ist der Index, beginnend bei 0, Sie adressieren also die erste Spalte und die erste Reihe. `weight` legt ein relatives Gewicht der einzelnen Reihen und Spalten zum Rest fest.

Ein Beispiel: Ein Programm hat zwei Spalten. Sie konfigurieren die Spalten so:

```
self.grid_columnconfigure(0, weight=1)
self.grid_columnconfigure(1, weight=2)
```

Die Spalte mit dem Gewicht 2 nimmt sich doppelt so viel Platz wie die Spalte mit dem Gewicht 1. Das wird deutlich, wenn Sie einen leeren Container mit blauem Hintergrund in Spalte 0 platzieren und einen leeren Container mit rotem Hintergrund in Spalte 1. Egal wie groß sie das Fenster ziehen, der rote Container wird dank seines Gewichts immer doppelt so groß sein wie der blaue Container, siehe 15.3.

15.5. Frame platzieren

Zurück zum Login-Programm, wo Sie nur eine Spalte und eine Reihe mit demselben Gewicht konfiguriert haben. Der Login soll in einem anderen Frame stattfinden. Ein

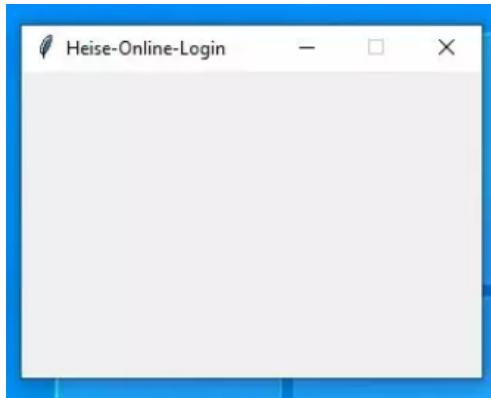


Figure 15.4.: Der Titel des Fensters ist schon anders, ansonsten ist das Fenster wüst und leer.

Frame ist ein Container für Widgets wie Labels, Buttons und so weiter. Für diesen Frame erstellen Sie eine neue Klasse, ähnlich wie beim Hauptfenster:

```
class Loginframe(tk.Frame):
    def __init__(self, parent):

        tk.Frame.__init__(self, parent)
        self.parent = parent
```

Sie haben nun kein `tk.Tk` mehr, sondern `tk.Frame`, da Sie einen Frame benötigen. Im Hauptprogramm definieren Sie nun diesen Frame:

```
self.loginframe = Loginframe(self)
```

Dadurch wird deutlich, dass die Instanz des Hauptprogramms (`self`), zum übergeordneten Element für den Loginframe wird (`parent`). Nun müssen Sie diesen Frame noch im Gitter platzieren, sonst wird er nicht angezeigt. Das geht mit `grid()`, indem Sie dem Frame Reihe 0 und Spalte 0 zuweisen:

```
self.loginframe.grid(row=0, column=0)
```

Damit sieht das Programm bisher so aus:

15.6. Widgets erstellen

Jetzt wird es Zeit, sich Gedanken über die Widgets zu machen, die im Login-Frame erscheinen sollen. Also: Was macht ein Login-Fenster aus? Am wichtigsten sind natürlich die Eingabefelder für den Benutzernamen und das Passwort. Damit der Nutzer auch weiß, welches Eingabefenster für den Benutzernamen und welches für das Passwort gedacht ist, benötigen Sie zwei Labels dafür. Diese Labels können links neben den Eingabefeldern stehen. Ein Beschreibungstext über Labels und Eingabefeldern wäre noch nett, damit der Nutzer weiß, was er da überhaupt tut. Ein Button für den Login darf abschließend unter allen Elementen natürlich nicht fehlen.

Das Schema sieht aus so aus:

1000	Beschreibung-Text
1001	Benutzername-Label
1002	Passwort-Label
	Login-Button

```

1000 import tkinter as tk
..../code/General/Tkinter/TkinterClass.py

1000 class Loginframe(tk.Frame):
1001     def __init__(self, parent):
1002         tk.Frame.__init__(self, parent)
1003         self.parent = parent
..../code/General/Tkinter/TkinterClass.py

1000 class Programm(tk.Tk):
1001     def __init__(self, parent):
1002         tk.Tk.__init__(self, parent)
1003         self.parent = parent
1004
1005         self.title("Heise-Online-Login")
1006         self.minsize(300, 130)
1007         self.resizable(width=False, height=False)
1008
1009         self.grid_columnconfigure(0, weight=1)
1010         self.grid_rowconfigure(0, weight=1)
1011
1012         self.loginframe = Loginframe(self)
..../code/General/Tkinter/TkinterClass.py

1000     self.loginframe.grid(row=0, column=0)
..../code/General/Tkinter/TkinterClass.py

1000 programm=Programm(None)
1001 programm.mainloop()
..../code/General/Tkinter/TkinterClass.py

```

Listing 15.3.: Tkinter Class mit Frame

Damit wissen Sie nun, welche Widgets Sie benötigen und wie sie später platziert werden sollen. Machen Sie sich am Anfang ruhig intensiv Gedanken über das Layout. Später etwas zu ändern ist immer nervig, da Sie dann oft die Position von mehreren Widgets anpassen müssen.

Um die Übersicht zu behalten, verweisen Sie in der Methode `__init__(self, parent)` der Klasse `Loginframe` auf die Methode `Widgets`:

```
self.Widgets()
```

Die Methode erstellen Sie anschließend und packen später die Widgets dort rein:

```
def Widgets(self):
```

15.7. Beschreibungstext

Beginnen Sie mit dem Beschreibungstext. Für einen Text mit möglichen Umbrüchen hat Tkinter das Widget `Message()` vorgesehen. Sie können es ähnlich definieren wie das Label im Hello-World-Beispiel:

```
self.message_beschreibung = tk.Message(self, text="Gib deine Logindaten  
für Heise Online ein und drücke dann den Button.", width=240, justify="center")
```

Der erste Parameter ist wieder das Elternelement, in diesem Fall `self`, also die Instanz des Frames. Hinter `text` verbirgt sich der hardgedecode Text, den Sie später im Fenster sehen möchten. Sie können den Text aber auch in eine Variable auslagern, wenn Sie möchten. `width` gibt die Länge des Widgets an, das müssen Sie nicht unbedingt definieren, ist aber gerade bei statischen Texten sinnvoll, um das Layout zu gestalten. Mit `justify="center"` zentrieren Sie den Text, ansonsten wird er standardmäßig linksbündig angezeigt.

Nun platzieren Sie das Widget in Ihrem Gitter:

```
self.message_beschreibung.grid(row=0, column=0, columnspan=2)
```

Das Widget soll ganz oben stehen, daher startet es in Reihe 0 und Spalte 0. `columnspan=2` gibt an, dass sich dieses Widget über zwei Spalten streckt. Ein Blick auf das vorher visualisierte Layout in Tabellenform ist hier hilfreich.

15.8. Labels

Zwei Labels sollen dem Nutzer klarmachen, was er später eintippen soll. Sie können die beiden schnell definieren:

```
self.label_benutzername = tk.Label(self, text="Benutzername:")  
self.label_passwort = tk.Label(self, text="Passwort:")
```

Genauso schnell platzieren Sie die Widgets im Gitter:

```
self.label_benutzername.grid(row=1, column=0, sticky="e")  
self.label_passwort.grid(row=2, column=0, sticky="e")
```

Das Label für den Benutzernamen steht unter dem vorher angelegten Beschreibungstext, also in Reihe 1. Da es links stehen soll, erhält es die Spalte 0. Das Label für das Passwort steht eine Reihe darunter.

Mit `sticky` lassen Sie Widgets an Rändern kleben. Das `e` steht für east, also für Osten. Ziel ist es, dass die Labels am rechten Rand kleben, während die Eingabefelder eine Spalte weiter am linken Rand kleben. So stehen Sie direkt beieinander. Alle Himmelsrichtungen sind möglich: `n` für Norden, `e` für Osten, `s` für Süden und `w` für Westen. Auch Kombinationen davon: `ew` lässt das Widget am linken und rechten Rand kleben, streckt es also. `nsew` würde versuchen, den gesamten Platz auszufüllen. Die Reihenfolge der Himmelsrichtungen ist dabei egal.



Figure 15.5.: Sieht aus wie ein Login-Fenster, tut aber noch nichts.

15.9. Eingabefelder

Eingabefelder kennt Tkinter unter dem Namen `Entry()`. Hier müssen Sie nicht viel festlegen:

```
self.entry_benutzername = tk.Entry(self, width=15)
self.entry_passwort = tk.Entry(self, show="*", width=15)
```

`width` ist wieder die festgelegte Breite. Spielen Sie ruhig ein wenig mit den Werten herum, um ein Gefühl fürs Layout zu erhalten. Das Passwort-Feld kommt noch mit einem `show="*"` als Parameter, damit werden alle eingetippten Buchstaben für den Nutzer durch Sternchen ersetzt – falls der neugierige Kollege über die Schulter schaut. Platzieren Sie die Eingabefelder ähnlich wie die zugehörigen Labels:

```
self.entry_benutzername.grid(row=1, column=1, sticky="w")
self.entry_passwort.grid(row=2, column=1, sticky="w")
```

Auch sie erscheinen unter dem Beschreibungstext, aber wegen dem Parameter `column=1` stehen Sie nun rechts neben den zugehörigen Labels. `sticky="w"` packt die Eingabefelder in den Westen, packt sie also am linken Rand fest.

15.10. Button

Der Button ist das wichtigste Element in diesem Ensemble, schließlich sorgt er für weitere Aktionen, wenn der Nutzer darauf klickt:

```
self.button_login = tk.Button(self, text="Einloggen", command = self.Button_login_geklickt)
```

Der Parameter `text` bestimmt, was auf dem Button stehen soll: Einloggen. `command` gibt an, was bei einem Klick passieren soll. Hier verweist der Klick auf die Methode `Button_login_geklickt`, die noch nicht existiert.

Bevor Sie der Aktion einen Sinn geben, packen Sie den Button ins Gitter:

```
self.button_login.grid(row=3, column=0, columnspan=2)
```

Er steht unter den Labels und Eingabefeldern (`row=3`), beginnt in der ersten Spalte (`column=0`) und belegt zwei Spalten (`columnspan=2`).

Und damit haben Sie eine Fläche erstellt, das wie ein echtes Login-Fenster aussieht. Der Code für die Widgets ist überschaubar, siehe 15.4.

15.11. Kosmetik

Die grundlegenden Widgets sind da, aber etwas Kosmetik fehlt noch. So muss der Nutzer immer erst in die Eingabefelder klicken, damit er loslegen kann. Ein

```

1000 def Widgets(self):
1001     self.message_beschreibung = tk.Message(self, text="Gib Deine
1002         Logindaten fuer Heise Online ein und druecke dann den Button.", width=240, justify="center")
1003
1004     self.label_benutzername = tk.Label(self, text="Benutzername:")
1005     self.label_passwort = tk.Label(self, text="Passwort:")
1006
1007     self.entry_benutzername = tk.Entry(self, width=15)
1008     self.entry_passwort = tk.Entry(self, show="*", width=15)
1009
1010     self.button_login = tk.Button(self, text="Einloggen", command = self.
1011         .Button_login_geklickt)
1012     self.parent.bind('<Return>', self.Button_login_geklickt)
1013
1014     self.message_beschreibung.grid(row=0, column=0, columnspan=2)
1015     self.label_benutzername.grid(row=1, column=0, sticky="e")
1016     self.label_passwort.grid(row=2, column=0, sticky="e")
1017     self.entry_benutzername.grid(row=1, column=1, sticky="w")
1018     self.entry_passwort.grid(row=2, column=1, sticky="w")
1019     self.button_login.grid(row=3, column=0, columnspan=2)

```

..//code/General/Tkinter/TkinterClass.py

Listing 15.4.: Widget für die Tkinter Class

`self.entry_benutzername.focus()` setzt den Fokus sofort auf das Feld für den Benutzernamen. `1337H4xxx00R97` kann also sofort anfangen, seinen Namen einzutippen. Dann wäre es noch nett, wenn der Nutzer nicht nur auf den Button klicken kann, sondern einfach die Entertaste drückt, wenn er fertig ist. Das geht mit `bind()`:

```
self.master.bind("<Return>", self.Button_login_geklickt)
```

Es bindet die Tastenkombination im ersten Parameter (`<Return>`, also Enter) an einen Befehl. In diesem Fall ist es derselbe Befehl, der auch beim Klick auf den Button ausgeführt wird.

Aber: Sie können so viel eingeben und rumdrücken wie Sie möchten, noch tut sich da nix. Als nächstes fügen Sie dem Programm echte Funktionen hinzu.

15.12. Button klickbar machen

Der Button verweist beim Klick auf die Methode `Button_login_geklickt`: `def Button_login_geklickt(self, event=None):`. Dort findet der Login mit den Daten aus den Eingabefeldern statt. Den Parameter `event=None` benötigen Sie, weil Sie vorher mit `bind()` ein Event an diese Methode gebunden haben, nämlich das Drücken der Entertaste. Dieses Event wird immer versteckt weitergereicht. In diesem Beispiel würde nur `event` als Parameter ausreichen, mit `event=None` definieren Sie es aber vorher. Mit `print(event)` in der Methode können Sie sich etwa weitere Infos zu dem Tastendruck anzeigen lassen.

Mit `get()` holen Sie die Eingaben des Nutzers aus den Feldern:

```
self.benutzername = self.entry_benutzername.get()
self.passwort = self.entry_passwort.get()
```

Damit können Sie nun weiterarbeiten und sich bei heise online einloggen. Wir nutzen dafür den Login über die externe Bibliothek Requests. Sie müssen die Bibliothek vorher installieren, etwa mit dem Paketverwalter pip:

```
pip install requests
```

Der Befehl `import` holt die Bibliothek ins Programm:

```
import requests
```

15.13. Fake-Browser und Login-Daten

Anschließend definieren Sie einen Fake-Browser in einem Dictionary, um auf die Daten zugreifen zu können:

```
self.fake_browser = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0"}
```

Der User-Agent enthält Daten über den Browser und dem verwendeten System und wird an Webseiten geschickt, wenn Sie eine aufrufen. Ihren eigenen User-Agent sehen Sie etwa auf <http://wieistmeinuseragent.de>. Ihr Python-Programm tut dann so, als wäre es dieser User-Agent.

Die Daten für den Login speichern Sie in einem weiteren Dictionary:

```
self.login_daten = {"username": self.benutzername, "password": self.passwort, "action": "/sso/login/login"}
```

Um sich an einer Website einzuloggen, müssen Sie sie vorher analysieren. Bei heise online passiert das über die Seite <https://www.heise.de/sso/login/login> und einer POST-Methode mit der Aktion `/sso/login/login`. Wie genau Sie solche Feinheiten entdecken und sich richtig anmelden, haben wir in einem anderen Artikel ausführlich aufgeschrieben.

Zusammengenommen funktioniert der Login dann so:

```
self.login = requests.Session().post(url="https://www.heise.de/sso/login/login", data=self.login_daten, headers=self.fake_browser)
```

Mit `requests.Session()` erstellen Sie eine Session, das bedeutet, dass etwa Cookies gespeichert werden und Sie später mit aktivem Login andere Seiten aufrufen können. `post()` steht für die HTML-Methode POST, die `url` ist die heise-Login-Seite, `data` enthält die vorher festgelegten Login-Daten und in `headers` geben Sie den Fake-Browser als User-Agent mit.

15.14. Fehlermeldung

Das war der Login. Aber er ist noch recht unsichtbar, schließlich tun Sie damit noch nichts. Eine Fehlermeldung ist hilfreich, falls der Nutzer falsche Daten eingibt. Sie erkennen einen fehlgeschlagenen Login daran, dass im Text von `self.login` der Satz "Der Benutzername oder das Passwort ist falsch." auftaucht. Das fragen Sie ab:

```
if "Der Benutzername oder das Passwort ist falsch." in self.login.text:
```

Wenn dieser Text vorkommt, soll sich die Beschreibung ändern und den Nutzer auf den Fehler aufmerksam machen. Anschließend kann er es nochmal probieren. Das geht etwa so:

```
self.message_beschreibung.config(text="Fehler: Der Benutzername oder das Passwort ist falsch.", width=160, fg="red")
```

Das Programm leitet die Daten nur durch und speichert sie nicht. Die Fehlermeldung wird anhand der Reaktion der Website erstellt, siehe 15.6. Sollte sich also die Fehlernachricht auf heise online ändern, würde das Skript nicht mehr funktionieren. Mit `config()` ändern Sie die Eigenschaften des Widgets und passen den Text an. Auch die Breite ändern Sie und die Textfarbe setzen Sie mit `fg` für foreground, also



Figure 15.6.: Da ist was schiefgelaufen.

```

1000 class Loginerfolg(tk.Frame):
1001     def __init__(self, parent):
1002         tk.Frame.__init__(self, parent)
1003         self.parent = parent
1004         self.Widgets()
1005
1006     def Widgets(self):
1007         self.message_erfolg = tk.Message(self, text="Sie haben sich"
1008                                         "erfolgreich eingeloggt.", width=160, justify="center")
1009         self.message_erfolg.grid(row=0, column=0)
..../code/General/Tkinter/TkinterClass.py

```

Listing 15.5.: Klasse `Loginerfolg`

Vordergrund, auf red, als rot. Ist ja ein Fehler. Parameter, die Sie nicht ändern bleiben erhalten – wie etwa `justify="center"`.

Nun müssen Sie noch entscheiden, was passiert, wenn der Nutzer sich korrekt einloggt. Sie können etwa ein neues Fenster aufrufen und den Login-Frame zerstören. Sie benötigen ihn ja nicht mehr.

15.15. Korrekter Login

Für das neue Fenster erstellen Sie eine neue Klasse namens `Loginerfolg` mit einer simplen Nachricht, siehe 15.5:

Alle Elemente daraus dürften Ihnen bekannt vorkommen.

Das neue Fenster definieren Sie zudem im Hauptprogramm, in der Klasse Programm:

```
self.loginerfolg = Loginerfolg(self)
```

Die Idee: Sie zerstören das Loginfenster, verweisen dann über das Hauptprogramm auf den Frame Loginerfolg. Gehen Sie dafür zurück in die Klasse Loginframe und in die Methode `Button_login_geklickt`. Dort legen Sie eine Alternative zum gescheiterten Login an:

```
else:
    self.destroy()
    self.parent.loginerfolg.grid(row=0, column=0)
```

`self.destroy()` schließt einen Frame, in diesem Fall die aktuelle Instanz des Frames. `self.parent.loginerfolg.grid()` verweist auf das übergeordnete Element, also auf die Klasse `Programm`. Genau für so einen Fall haben Sie in der Methode `__init__` der Klasse `Loginframe` `self.parent = parent` definiert. Die Klasse `Programm` kennt auch schon `loginerfolg`, denn das haben Sie vorher dort definiert: `self.loginerfolg = Loginerfolg(self)`. Mit `grid()` setzen Sie schließlich das

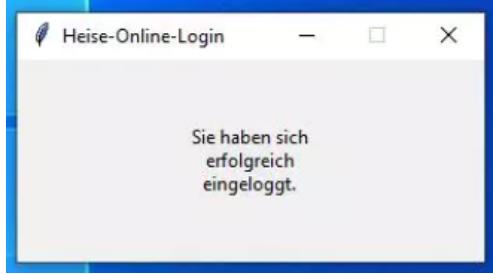


Figure 15.7.: Erfolg: Der Login hat geklappt.

neue Fenster **Loginerfolg** im Hauptfenster ein und machen es für den Nutzer sichtbar, siehe 15.7.

Es kann am Anfang ein wenig verwirrend sein, da Sie hier zwischen den Klassen hin und her springen. Vereinfacht gesagt geht der Übergang so: Das Loginfenster zerstört sich selbst und ruft über das Hauptprogramm die Erfolgsnachricht in einem anderen Fenster auf.

15.16. Menüleiste erstellen

Den Hauptteil haben Sie damit abgeschlossen, der Login funktioniert. Aber ein Programm kann ja immer schöner werden. Sie können etwa eine Menüleiste erstellen, über die der Nutzer das Programm beenden kann.

Erstellen Sie eine neue Klasse:

```
class Menu_Leiste(tk.Menu):
    def __init__(self, parent):
        tk.Menu.__init__(self, parent)
```

Sieht bekannt aus, unterscheidet sich von den anderen Klassen eigentlich nur über das **tk.Menu**, es ist ja kein Frame (**tk.Frame**) und auch kein Hauptfenster (**tk.Tk**). Es soll die Menüleiste für alle Fenster werden, daher fügen Sie in der Klasse Programm diesen Eintrag hinzu:

```
self.config(menu=Menu_Leiste(self))
```

Zurück zur Klasse **Menu_Leiste**. Nun spendieren Sie der Leiste ein Datei-Menü:

```
Datei_Menu = tk.Menu(self, tearoff=False)
```

tearoff ist auf **False** gesetzt. Wäre der Parameter **True**, würde das Menü eine gestrichelte Leiste oben erhalten, mit der der Nutzer das Menü loslösen und verschieben kann. Diese Funktion kann nützlich sein, hier verzichten wir darauf.

Das Datei-Menü erhält nun mit der Funktion **add_cascade()** ein neues Untermenü:

```
self.add_cascade(label = "Datei", menu = Datei_Menu)
```

label legt den Namen fest, **Datei**, das Datei-Menü verknüpfen Sie über den Parameter **menu**.

Nun fügen Sie dem Menü den ersten Befehl hinzu mit der Funktion **add_command()**:

```
Datei_Menu.add_command(label = "Beenden", command = self.quit, accelerator="Alt+F4")
```

label ist der Name des Befehls, **accelerator** fügt einen Text rechts neben dem Label hinzu. Sie können es etwa nutzen, um Tastenkombinationen für den Befehl anzuzeigen, siehe 15.8. **command** enthält den Befehl, hier **self.quit**, was das gesamte Programm beendet.

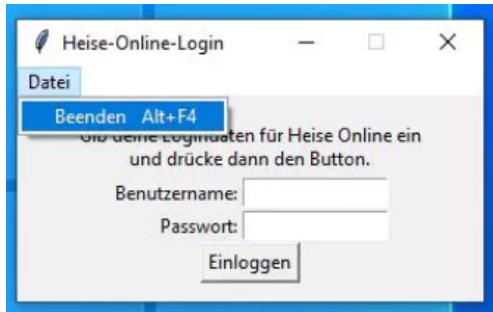


Figure 15.8.: Ein gewohntes Menü mit einem gewohnten Befehl.

Auf diese Weise können Sie noch weitere Einträge und Befehle hinzufügen, um etwa andere Fenster zu öffnen oder Einstellungen zu ändern. Der gesamte Code für das kleine Beispielprogramm ist in 15.6 dargestellt.

15.17. Fazit

Viele Möglichkeiten sind bei Tkinter selbsterklären, wenn man es mindestens einmal gemacht hat. Fenster erstellen Sie immer wieder gleich, Widgets lassen sich auf ähnliche Arten bearbeiten und Menüpunkte kann man schnell hinzufügen. Wer nicht weiterkommt, findet über Google viele beantwortete Fragen aus der Community. Bei komplexeren Layouts und Programmen kann man aber den Überblick verlieren. Daher ist es wichtig, seinen Klassen und Methoden eindeutige Namen zu geben. Für eine schnelle Umsetzung eines einfachen Konsolentools ist Tkinter aber gut geeignet und ist eine Alternativen zu Frameworks wie Qt.

```

1000 # Tkinter class
1002 import requests
1003 import tkinter as tk
1004
1005 class Menu_Leiste(tk.Menu):
1006     def __init__(self, parent):
1007         tk.Menu.__init__(self, parent)
1008
1009         Datei_Menu = tk.Menu(self, tearoff=False)
1010         self.add_cascade(label = "Datei", menu = Datei_Menu)
1011         Datei_Menu.add_command(label = "Beenden", command = self.quit,
1012                                accelerator="Alt+F4")
1013
1014 class Loginerfolg(tk.Frame):
1015     def __init__(self, parent):
1016         tk.Frame.__init__(self, parent)
1017         self.parent = parent
1018         self.Widgets()
1019
1020     def Widgets(self):
1021         self.message_erfolg = tk.Message(self, text="Sie haben sich
1022             erfolgreich eingeloggt.", width=160, justify="center")
1023         self.message_erfolg.grid(row=0, column=0)
1024
1025 class Loginframe(tk.Frame):
1026     def __init__(self, parent):
1027         tk.Frame.__init__(self, parent)
1028         self.parent = parent
1029         self.Widgets()
1030
1031     def Widgets(self):
1032         self.message_beschreibung = tk.Message(self, text="Gib Deine
1033             Logindaten fuer Heise Online ein und druecke dann den Button.",
1034             width=240, justify="center")
1035
1036         self.label_benutzername = tk.Label(self, text="Benutzername:")
1037         self.label_passwort = tk.Label(self, text="Passwort:")
1038
1039         self.entry_benutzername = tk.Entry(self, width=15)
1040         self.entry_passwort = tk.Entry(self, show="*", width=15)
1041
1042         self.button_login = tk.Button(self, text="Einloggen", command = self
1043             .Button_login_geklickt)
1044         self.parent.bind('<Return>', self.Button_login_geklickt)
1045
1046         self.message_beschreibung.grid(row=0, column=0, columnspan=2)
1047         self.label_benutzername.grid(row=1, column=0, sticky="e")
1048         self.label_passwort.grid(row=2, column=0, sticky="e")
1049         self.entry_benutzername.grid(row=1, column=1, sticky="w")
1050         self.entry_passwort.grid(row=2, column=1, sticky="w")
1051         self.button_login.grid(row=3, column=0, columnspan=2)
1052
1053         self.entry_benutzername.focus()
1054
1055     def Button_login_geklickt(self, event=None):
1056         self.benutzername = self.entry_benutzername.get()
1057         self.passwort = self.entry_passwort.get()
1058
1059         self.fake_browser = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0;
1060             Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0"}
1061         self.login_daten = {"username": self.benutzername, "password": self.
1062             passwort, "action": "/sso/login/login"}
1063
1064         self.login = requests.Session().post(url="https://www.heise.de/sso/
1065             login/login", data=self.login_daten, headers=self.fake_browser)
1066
1067         if "Der Benutzername oder das Passwort ist falsch." in self.login.
1068             text:
1069             self.message_beschreibung.config(text="Fehler: Der Benutzername
1070                 oder das Passwort ist falsch.", width=160, fg="red")
1071         else:
1072             self.destroy()
1073             self.parent.loginerfolg.grid(row=0, column=0)
1074
1075 class Programm(tk.Tk):
1076     def __init__(self, parent):
1077         tk.Tk.__init__(self, parent)
1078         self.parent = parent
1079

```

16. Hello World “Iris”

16.1. Your First Machine Learning Project in Python Step-By-Step

Do you want to do machine learning using Python, but you’re having trouble getting started?

You will complete your first machine learning project using Python.

In this step-by-step tutorial you will:

- Download and install Python SciPy and get the most useful package for machine learning in Python.
- Load a dataset and understand it’s structure using statistical summaries and data visualization.
- Create 6 machine learning models, pick the best and build confidence that the accuracy is reliable.

If you are a machine learning beginner and looking to finally get started using Python, this tutorial was designed for you.

Kick-start your project with my new book Machine Learning Mastery With Python, including step-by-step tutorials and the Python source code files for all examples.

16.2. How Do You Start Machine Learning in Python?

The best way to learn machine learning is by designing and completing small projects.

16.2.1. Python Can Be Intimidating When Getting Started

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems.

There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

- It will force you to install and start the Python interpreter (at the very least).
- It will give you a bird’s eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

16.2.2. Beginners Need A Small End-to-End Project

Books and courses are frustrating. They give you lots of recipes and snippets, but you never get to see how they all fit together.

When you are applying machine learning to your own datasets, you are working on a project.

A machine learning project may not be linear, but it has a number of well known steps:

1. Define Problem.
2. Prepare Data.
3. Evaluate Algorithms.
4. Improve Results.
5. Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

If you can do that, you have a template that you can use on dataset after dataset. You can fill in the gaps such as further data preparation and improving result tasks later, once you have more confidence.

16.2.3. Hello World of Machine Learning

The best small project to start with on a new tool is the classification of iris flowers (e.g. the iris dataset).

This is a good project because it is so well understood.

- Attributes are numeric so you have to figure out how to load and handle data.
- It is a classification problem, allowing you to practice with perhaps an easier type of supervised learning algorithm.
- It is a multi-class classification problem (multi-nominal) that may require some specialized handling.
- It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or A4 page).
- All of the numeric attributes are in the same units and the same scale, not requiring any special scaling or transforms to get started.

Let's get started with your hello world machine learning project in Python.

16.3. Machine Learning in Python: Step-By-Step Tutorial ([start here](#))

In this section, we are going to work through a small machine learning project end-to-end.

Here is an overview of what we are going to cover:

- Installing the Python and SciPy platform.
- Loading the dataset.
- Summarizing the dataset.
- Visualizing the dataset.
- Evaluating some algorithms.
- Making some predictions.

Take your time. Work through each step.

Try to type in the commands yourself or copy-and-paste the commands to speed things up.

If you have any questions at all, please leave a comment at the bottom of the post.

16.4. Step 1: Downloading, Installing and Starting Python SciPy

Get the Python and SciPy platform installed on your system if it is not already. I do not want to cover this in great detail, because others already have. This is already pretty straightforward, especially if you are a developer. If you do need help, ask a question in the comments.

16.4.1. Install SciPy Libraries

This tutorial assumes Python version 2.7 or 3.6+.

There are 5 key libraries that you will need to install. Below is a list of the Python SciPy libraries required for this tutorial:

- scipy
- numpy
- matplotlib
- pandas
- sklearn

There are many ways to install these libraries. My best advice is to pick one method then be consistent in installing each library.

The [scipy installation page](#) provides excellent instructions for installing the above libraries on multiple different platforms, such as Linux, mac OS X and Windows. If you have any doubts or questions, refer to this guide, it has been followed by thousands of people.

- On Mac OS X, you can use macports to install Python 3.6 and these libraries. For more information on macports, [see the homepage](#).
- On Linux you can use your package manager, such as yum on Fedora to install RPMs.

If you are on Windows or you are not confident, I would recommend installing the free version of [Anaconda](#) that includes everything you need.

Note: This tutorial assumes you have scikit-learn version 0.20 or higher installed.

Need more help? See one of these tutorials:

- [How to Setup a Python Environment for Machine Learning with Anaconda](#)
- [How to Create a Linux Virtual Machine For Machine Learning With Python 3](#)

16.4.2. Start Python and Check Versions

It is a good idea to make sure your Python environment was installed successfully and is working as expected.

The script below will help you test out your environment. It imports each library required in this tutorial and prints the version.

Open a command line and start the python interpreter:

```
python
```

I recommend working directly in the interpreter or writing your scripts and running them on the command line rather than big editors and IDEs. Keep things simple and focus on the machine learning not the toolchain.

```

1000 # Check the versions of libraries
1002 # Python version
1003 import sys
1004 print('Python: {}'.format(sys.version))
1005 # scipy
1006 import scipy
1007 print('scipy: {}'.format(scipy.__version__))
1008 # numpy
1009 import numpy
1010 print('numpy: {}'.format(numpy.__version__))
1011 # matplotlib
1012 import matplotlib
1013 print('matplotlib: {}'.format(matplotlib.__version__))
1014 # pandas
1015 import pandas
1016 print('pandas: {}'.format(pandas.__version__))
1017 # scikit-learn
1018 import sklearn
1019 print('sklearn: {}'.format(sklearn.__version__))

```

..../Code/General/HelloWorldIris/HelloWorldIrisCheck.py

Listing 16.1.: Example “Hello World Iris” - Versionsprüfung der Packages

Type or copy and paste the script 16.1:

Here is the output I get on my OS X workstation:

```

1000 Python: 3.6.11 (default, Jun 29 2020, 13:22:26)
1001 [GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)]
1002 scipy: 1.5.2
1003 numpy: 1.19.1
1004 matplotlib: 3.3.0
1005 pandas: 1.1.0
1006 sklearn: 0.23.2

```

Compare the above output to your versions.

Ideally, your versions should match or be more recent. The APIs do not change quickly, so do not be too concerned if you are a few versions behind. Everything in this tutorial will very likely still work for you.

If you get an error, stop. Now is the time to fix it.

If you cannot run the above script cleanly you will not be able to complete this tutorial. My best advice is to Google search for your error message or post a question on [Stack Exchange](#).

16.5. Step 2: Load The Data

We are going to use the iris flowers dataset. This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone.

The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

You can learn more about [this dataset on Wikipedia](#).

In this step we are going to load the iris data from CSV file URL.

```

1000 # Load libraries
1001 from pandas import read_csv
1002 from pandas.plotting import scatter_matrix
1003 from matplotlib import pyplot
1004 from sklearn.model_selection import train_test_split
1005 from sklearn.model_selection import cross_val_score
1006 from sklearn.model_selection import StratifiedKFold
1007 from sklearn.metrics import classification_report
1008 from sklearn.metrics import confusion_matrix
1009 from sklearn.metrics import accuracy_score
1010 from sklearn.linear_model import LogisticRegression
1011 from sklearn.tree import DecisionTreeClassifier
1012 from sklearn.neighbors import KNeighborsClassifier
1013 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
1014 from sklearn.naive_bayes import GaussianNB
1015 from sklearn.svm import SVC

```

.. /Code/General/HelloWorldIris/HelloWorldIrisLoad.py

Listing 16.2.: Example “Hello World Iris” - Load the Packages

```

1000 # Load dataset
1001 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
1002 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
1003 dataset = read_csv(url, names=names)

```

.. /Code/General/HelloWorldIris/HelloWorldIrisLoad.py

Listing 16.3.: Example “Hello World Iris” - Load the Dataset

16.5.1. Import libraries

First, let’s import all of the modules, functions and objects we are going to use in this tutorial, siehe 16.2.

Everything should load without error. If you have an error, stop. You need a working SciPy environment before continuing. See the advice above about setting up your environment.

16.5.2. Load Dataset

We can load the data directly from the UCI Machine Learning repository.

We are using pandas to load the data. We will also use pandas next to explore the data both with descriptive statistics and data visualization.

Note that we are specifying the names of each column when loading the data. This will help later when we explore the data, siehe 16.2.

The dataset should load without incident.

If you do have network problems, you can download the `iris.csv` file into your working directory and load it using the same method, changing URL to the local file name.

16.6. Step 3: Summarize the Dataset

Now it is time to take a look at the data.

In this step we are going to take a look at the data a few different ways:

1. Dimensions of the dataset.

```
1000 # shape
      print(dataset.shape)
```

..../Code/General>HelloWorldIris>HelloWorldIrisLoad.py

Listing 16.4.: Example “Hello World Iris” - Shape of the Dataset

```
1000 # head
      print(dataset.head(20))
```

..../Code/General>HelloWorldIris>HelloWorldIrisLoad.py

Listing 16.5.: Example “Hello World Iris” - Head of the Dataset

2. Peek at the data itself.
3. Statistical summary of all attributes.
4. Breakdown of the data by the class variable.

Don’t worry, each look at the data is one command. These are useful commands that you can use again and again on future projects.

16.6.1. Dimensions of Dataset

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the shape property, see 16.4.

You should see 150 instances and 5 attributes:

```
1000 (150, 5)
```

16.6.2. Peek at the Data

It is also always a good idea to actually eyeball your data, see 16.6.

You should see the first 20 rows of the data:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

```
1000 # descriptions
      print(dataset.describe())
..../Code/General>HelloWorldIris>HelloWorldIrisLoad.py
```

Listing 16.6.: Example “Hello World Iris” - Description of the Dataset

```
1000 # descriptions
      print(dataset.describe())
..../Code/General>HelloWorldIris>HelloWorldIrisLoad.py
```

Listing 16.7.: Example “Hello World Iris” - Distribution of the Dataset

16.6.3. Statistical Summary

Now we can take a look at a summary of each attribute.

This includes the count, mean, the min and max values as well as some percentiles, . We can see that all of the numerical values have the same scale (centimeters) and similar ranges between 0 and 8 centimeters.

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

16.6.4. Class Distribution

Let’s now take a look at the number of instances (rows) that belong to each class. We can view this as an absolute count.

We can see in 16.7 that each class has the same number of instances (50 or 33% of the dataset).

```
1000 class
      Iris-setosa      50
1002 Iris-versicolor   50
      Iris-virginica   50
```

16.6.5. Complete Example

For reference, we can tie all of the previous elements together into a single script. The complete example is listed in 16.8.

16.7. Step 4: Data Visualization

We now have a basic idea about the data. We need to extend that with some visualizations.

We are going to look at two types of plots:

```

1000 # Load libraries
1001 from pandas import read_csv
1002 from pandas.plotting import scatter_matrix
1003 from matplotlib import pyplot
1004 from sklearn.model_selection import train_test_split
1005 from sklearn.model_selection import cross_val_score
1006 from sklearn.model_selection import StratifiedKFold
1007 from sklearn.metrics import classification_report
1008 from sklearn.metrics import confusion_matrix
1009 from sklearn.metrics import accuracy_score
1010 from sklearn.linear_model import LogisticRegression
1011 from sklearn.tree import DecisionTreeClassifier
1012 from sklearn.neighbors import KNeighborsClassifier
1013 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
1014 from sklearn.naive_bayes import GaussianNB
1015 from sklearn.svm import SVC
1016
1017 # Load dataset
1018 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
1019 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
1020 dataset = read_csv(url, names=names)
1021
1022 # shape
1023 print(dataset.shape)
1024
1025 # head
1026 print(dataset.head(20))
1027
1028 # descriptions
1029 print(dataset.describe())
1030
1031 # class distribution
1032 print(dataset.groupby('class').size())

```

..../Code/General/HelloWorldIris/HelloWorldIrisLoad.py

Listing 16.8.: Example “Hello World Iris” - Complete Loading and Description of the Dataset

```

1000 # box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False,
             sharey=False)
1002 pyplot.show()
..../Code/General/HelloWorldIris/HelloWorldIrisVisualize.py

```

Listing 16.9.: Example “Hello World Iris” - Box and Whisker Plots

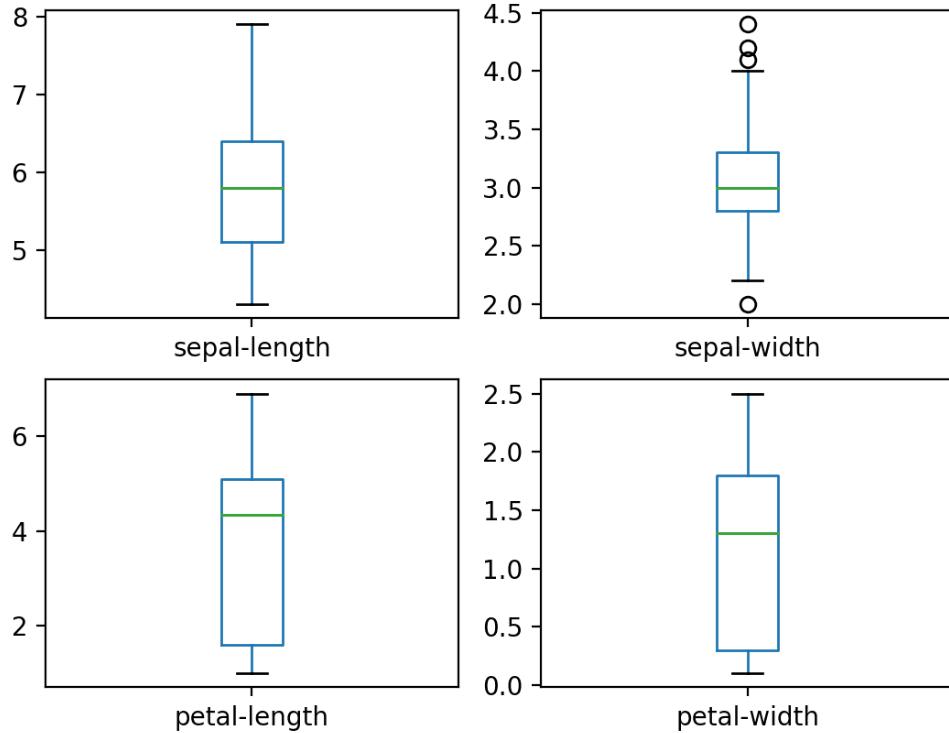


Figure 16.1.: Box and Whisker Plots for Each Input Variable for the Iris Flowers Dataset

1. Univariate plots to better understand each attribute.
2. Multivariate plots to better understand the relationships between attributes.

16.7.1. Univariate Plots

We start with some univariate plots, that is, plots of each individual variable. Given that the input variables are numeric, we can create box and whisker plots of each, see 16.9.

This gives us a much clearer idea of the distribution of the input attributes. see 16.1. We can also create a histogram of each input variable to get an idea of the distribution, see 16.10.

It looks like perhaps two of the input variables have a Gaussian distribution. This is useful to note as we can use algorithms that can exploit this assumption, see 16.2.

16.7.2. Multivariate Plots

Now we can look at the interactions between the variables.

```
1000 # histograms  
1001 dataset.hist()  
1002 pyplot.show()  
..../Code/General/HelloWorldIris/HelloWorldIrisVisualize.py
```

Listing 16.10.: Example “Hello World Iris” - Histograms

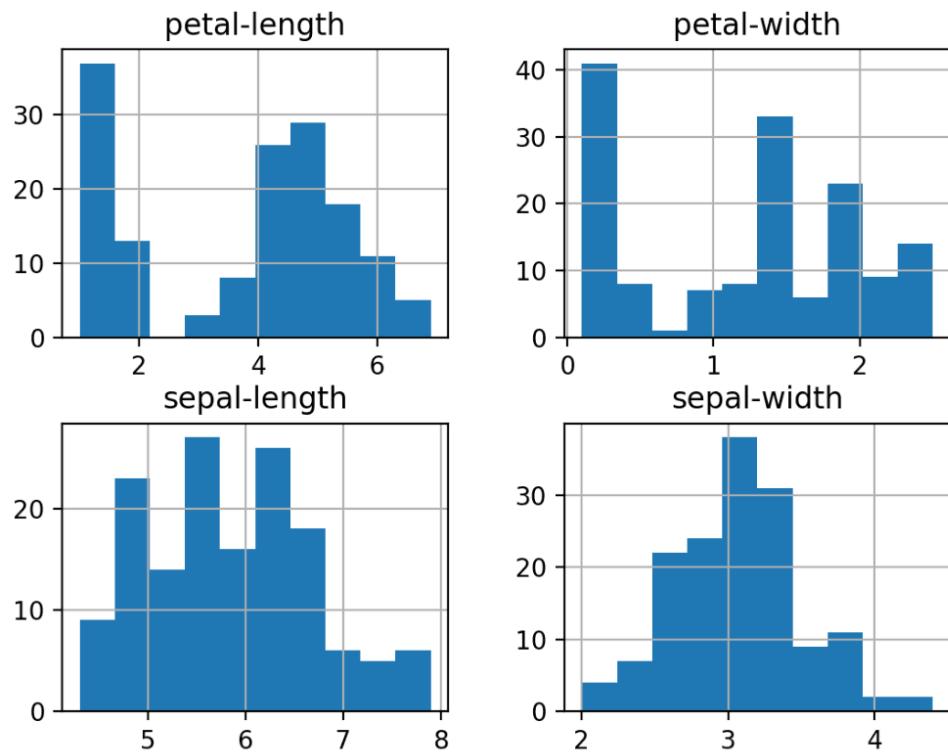


Figure 16.2.: Histogram Plots for Each Input Variable for the Iris Flowers Dataset

```
1000 # scatter plot matrix
1001 scatter_matrix(dataset)
1002 pyplot.show()
```

..../Code/General/HelloWorldIris/HelloWorldIrisVisualize.py

Listing 16.11.: Example “Hello World Iris” - Scatter Plot Matrix

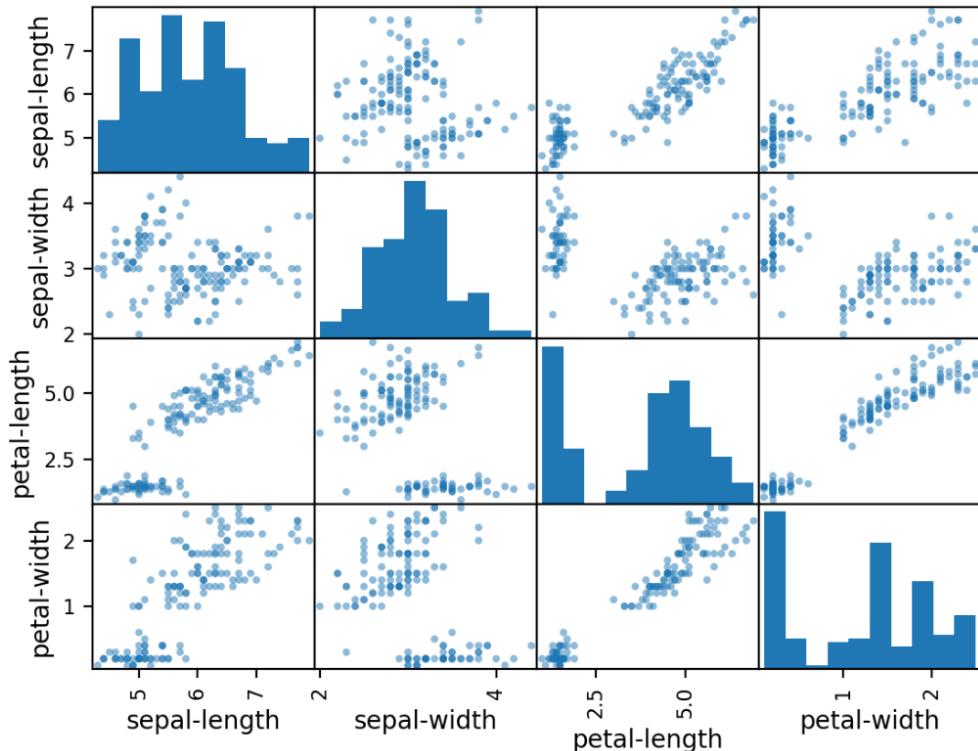


Figure 16.3.: Scatter Matrix Plot for Each Input Variable for the Iris Flowers Dataset

First, let’s look at scatterplots of all pairs of attributes. This can be helpful to spot structured relationships between input variables, see 16.11.

Note the diagonal grouping of some pairs of attributes in image 16.3. This suggests a high correlation and a predictable relationship.

16.7.3. Complete Example

For reference, we can tie all of the previous elements together into a single script. The complete example is listed in 16.12.

16.8. Step 5: Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data.

Here is what we are going to cover in this step:

1. Separate out a validation dataset.
2. Set-up the test harness to use 10-fold cross validation.

```

1000 # visualize the data
1001 from pandas import read_csv
1002 from pandas.plotting import scatter_matrix
1003 from matplotlib import pyplot
1004
1005 # Load dataset
1006 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
1007 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
1008 dataset = read_csv(url, names=names)
1009
1010 # box and whisker plots
1011 dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False,
1012               sharey=False)
1013 pyplot.show()
1014
1015 # histograms
1016 dataset.hist()
1017 pyplot.show()
1018
1019 # scatter plot matrix
1020 scatter_matrix(dataset)
1021 pyplot.show()

```

..../Code/General>HelloWorldIris>HelloWorldIrisVisualize.py

Listing 16.12.: Example “Hello World Iris” - Scatter Plot Matrix

```

1000 # Split-out validation dataset
1001 array = dataset.values
1002 X = array[:, :4]
1003 y = array[:, 4]
1004 X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
1005                     test_size=0.20, random_state=1, shuffle=True)

```

..../Code/General>HelloWorldIris>HelloWorldIrisCompare.py

Listing 16.13.: Example “Hello World Iris” - Plot Algorithm Comparison

3. Build multiple different models to predict species from flower measurements
4. Select the best model.

16.8.1. Create a Validation Dataset

We need to know that the model we created is good.

Later, we will use statistical methods to estimate the accuracy of the models that we create on unseen data. We also want a more concrete estimate of the accuracy of the best model on unseen data by evaluating it on actual unseen data.

That is, we are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be.

We will split the loaded dataset into two, 80% of which we will use to train, evaluate and select among our models, and 20% that we will hold back as a validation dataset, see 16.13.

You now have training data in the `X_train` and `Y_train` for preparing models and a `X_validation` and `Y_validation` sets that we can use later.

Notice that we used a python slice to select the columns in the NumPy array. If this is new to you, you might want to check-out this post:

- [How to Index, Slice and Reshape NumPy Arrays for Machine Learning in Python](#)

16.8.2. Test Harness

We will use stratified 10-fold cross validation to estimate model accuracy.

This will split our dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

Stratified means that each fold or split of the dataset will aim to have the same distribution of example by class as exist in the whole training dataset.

For more on the k-fold cross-validation technique, see the tutorial:

- [A Gentle Introduction to k-fold Cross-Validation](#)

We set the random seed via the argument `random_state` to a fixed number to ensure that each algorithm is evaluated on the same splits of the training dataset.

The specific random seed does not matter, learn more about pseudorandom number generators here:

- [Introduction to Random Number Generators for Machine Learning in Python](#)

We are using the metric of “accuracy” to evaluate models.

This is a ratio of the number of correctly predicted instances divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the scoring variable when we run build and evaluate each model next.

16.8.3. Build Models

We don't know which algorithms would be good on this problem or what configurations to use.

We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results.

Let's test 6 different algorithms:

- Logistic Regression (LR)
- Linear Discriminant Analysis (LDA)
- k-Nearest Neighbors (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).

This is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms.

Let's build and evaluate our models, see 16.14.

16.8.4. Select Best Model

We now have 6 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate.

Running the example above, we get the following raw results:

```

1000 # Spot Check Algorithms
models = []
1002 models.append(( 'LR', LogisticRegression(solver='liblinear', multi_class=
1003     'ovr')))
models.append(( 'LDA', LinearDiscriminantAnalysis()))
1004 models.append(( 'KNN', KNeighborsClassifier()))
models.append(( 'CART', DecisionTreeClassifier()))
1006 models.append(( 'NB', GaussianNB()))
models.append(( 'SVM', SVC(gamma='auto')))

1008 # Evaluate each model in turn
1009 results = []
1010 names = []
1011 for name, model in models:
1012     kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
1013     cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
1014         scoring='accuracy')
1015     results.append(cv_results)
1016     names.append(name)
1017     print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

```

..../Code/General>HelloWorldIris>HelloWorldIrisCompare.py

Listing 16.14.: Example “Hello World Iris” - Using the Algorithms

```

1000 # Compare Algorithms
pyplot.boxplot(results, labels=names)
1002 pyplot.title('Algorithm Comparison')
pyplot.show()

```

..../Code/General>HelloWorldIris>HelloWorldIrisCompare.py

Listing 16.15.: Example “Hello World Iris” - Plot Algorithm Comparison

```

1000 LR: 0.960897 (0.052113)
1001 LDA: 0.973974 (0.040110)
1002 KNN: 0.957191 (0.043263)
1003 CART: 0.957191 (0.043263)
1004 NB: 0.948858 (0.056322)
1005 SVM: 0.983974 (0.032083)

```

Note: Your `results may vary` given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that it looks like Support Vector Machines (SVM) has the largest estimated accuracy score at about 0.98 or 98%.

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (via 10 fold-cross validation).

A useful way to compare the samples of results for each algorithm is to create a box and whisker plot for each distribution and compare the distributions, see 16.15.

We can see in image 16.4 that the box and whisker plots are squashed at the top of the range, with many evaluations achieving 100% accuracy, and some pushing down into the high 80% accuracies.

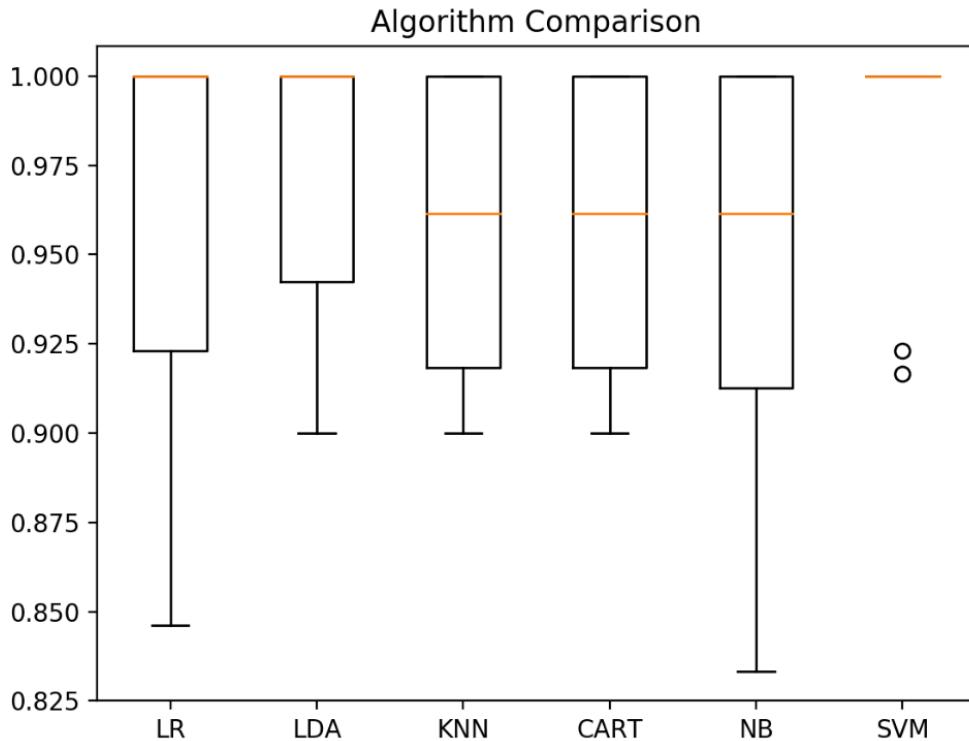


Figure 16.4.: Box and Whisker Plot Comparing Machine Learning Algorithms on the Iris Flowers Dataset

16.8.5. Complete Example

For reference, we can tie all of the previous elements together into a single script. The complete example is listed in 16.16.

16.9. Step 6: Make Predictions

We must choose an algorithm to use to make predictions.

The results in the previous section suggest that the SVM was perhaps the most accurate model. We will use this model as our final model.

Now we want to get an idea of the accuracy of the model on our validation set.

This will give us an independent final check on the accuracy of the best model. It is valuable to keep a validation set just in case you made a slip during training, such as overfitting to the training set or a data leak. Both of these issues will result in an overly optimistic result.

16.9.1. Make Predictions

We can fit the model on the entire training dataset and make predictions on the validation dataset, see 16.17.

You might also like to make predictions for single rows of data. For examples on how to do that, see the tutorial:

[How to Make Predictions with scikit-learn](#)

You might also like to save the model to file and load it later to make predictions on new data. For examples on how to do this, see the tutorial:

```

1000 # compare algorithms
1001 from pandas import read_csv
1002 from matplotlib import pyplot
1003 from sklearn.model_selection import train_test_split
1004 from sklearn.model_selection import cross_val_score
1005 from sklearn.model_selection import StratifiedKFold
1006 from sklearn.linear_model import LogisticRegression
1007 from sklearn.tree import DecisionTreeClassifier
1008 from sklearn.neighbors import KNeighborsClassifier
1009 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
1010 from sklearn.naive_bayes import GaussianNB
1011 from sklearn.svm import SVC
1012
1013 # Load dataset
1014 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
1015 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
1016 dataset = read_csv(url, names=names)
1017
1018 # Split-out validation dataset
1019 array = dataset.values
1020 X = array[:,0:4]
1021 y = array[:,4]
1022 X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
1023     test_size=0.20, random_state=1, shuffle=True)
1024
1025 # Spot Check Algorithms
1026 models = []
1027 models.append(( 'LR', LogisticRegression(solver='liblinear', multi_class=
1028     'ovr')))
1029 models.append(( 'LDA', LinearDiscriminantAnalysis()))
1030 models.append(( 'KNN', KNeighborsClassifier()))
1031 models.append(( 'CART', DecisionTreeClassifier()))
1032 models.append(( 'NB', GaussianNB()))
1033 models.append(( 'SVM', SVC(gamma='auto')))
1034
1035 # Evaluate each model in turn
1036 results = []
1037 names = []
1038 for name, model in models:
1039     kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
1040     cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
1041         scoring='accuracy')
1042     results.append(cv_results)
1043     names.append(name)
1044     print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
1045
1046 # Compare Algorithms
1047 pyplot.boxplot(results, labels=names)
1048 pyplot.title('Algorithm Comparison')
1049 pyplot.show()

```

..../Code/General/HelloWorldIris/HelloWorldIrisCompare.py

Listing 16.16.: Example “Hello World Iris” - Scatter Plot Matrix

```

1000 # Make predictions on validation dataset
1001 model = SVC(gamma='auto')
1002 model.fit(X_train, Y_train)
1003 predictions = model.predict(X_validation)

```

..../Code/General/HelloWorldIris/HelloWorldIrisPrediction.py

Listing 16.17.: Example “Hello World Iris” - Make Predictions

```

1000 # Evaluate predictions
1001 print(accuracy_score(Y_validation, predictions))
1002 print(confusion_matrix(Y_validation, predictions))
1003 print(classification_report(Y_validation, predictions))

..//Code/General/HelloWorldIris/HelloWorldIrisPrediction.py

```

Listing 16.18.: Example “Hello World Iris” - Evaluation

[Save and Load Machine Learning Models in Python with scikit-learn](#)**16.9.2. Evaluate Predictions**

We can evaluate the predictions by comparing them to the expected results in the validation set, then calculate classification accuracy, as well as a confusion matrix and a classification report.

We can see that the accuracy is 0.966 or about 96% on the hold out dataset, see 16.18. The confusion matrix provides an indication of the errors made.

Finally, the classification report provides a breakdown of each class by precision, recall, f1-score and support showing excellent results (granted the validation dataset was small).

1000	0.9666666666666667
1001	[[11 0 0]
1002	[0 12 1]
1003	[0 0 6]]
1004	precision recall f1-score support
1005	
1006	Iris-setosa 1.00 1.00 1.00 11
1007	Iris-versicolor 1.00 0.92 0.96 13
1008	Iris-virginica 0.86 1.00 0.92 6
1009	
1010	accuracy 0.97 30
1011	macro avg 0.95 0.97 0.96 30
1012	weighted avg 0.97 0.97 0.97 30

16.9.3. Complete Example

For reference, we can tie all of the previous elements together into a single script. The complete example is listed 16.19.

16.10. You Can Do Machine Learning in Python

Work through the tutorial above. It will take you 5-to-10 minutes, max!

You do not need to understand everything. (at least not right now) Your goal is to run through the tutorial end-to-end and get a result. You do not need to understand everything on the first pass. List down your questions as you go. Make heavy use of the `help("FunctionName")` help syntax in Python to learn about all of the functions that you’re using.

You do not need to know how the algorithms work. It is important to know about the limitations and how to configure machine learning algorithms. But learning about algorithms can come later. You need to build up this algorithm knowledge slowly over a long period of time. Today, start off by getting comfortable with the platform.

```

1000 # make predictions
1001 from pandas import read_csv
1002 from sklearn.model_selection import train_test_split
1003 from sklearn.metrics import classification_report
1004 from sklearn.metrics import confusion_matrix
1005 from sklearn.metrics import accuracy_score
1006 from sklearn.svm import SVC

1008 # Load dataset
1009 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
1010 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
1011 dataset = read_csv(url, names=names)

1012 # Split-out validation dataset
1013 array = dataset.values
1014 X = array[:,0:4]
1015 y = array[:,4]
1016 X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
1017     test_size=0.20, random_state=1)

1018 # Make predictions on validation dataset
1019 model = SVC(gamma='auto')
1020 model.fit(X_train, Y_train)
1021 predictions = model.predict(X_validation)

1024 # Evaluate predictions
1025 print(accuracy_score(Y_validation, predictions))
1026 print(confusion_matrix(Y_validation, predictions))
1027 print(classification_report(Y_validation, predictions))

```

.. /Code/General/HelloWorldIris/HelloWorldIrisPrediction.py

Listing 16.19.: Example “Hello World Iris” – Complete

You do not need to be a Python programmer. The syntax of the Python language can be intuitive if you are new to it. Just like other languages, focus on function calls (e.g. `function()`) and assignments (e.g. `a = "b"`). This will get you most of the way. You are a developer, you know how to pick up the basics of a language real fast. Just get started and dive into the details later.

You do not need to be a machine learning expert. You can learn about the benefits and limitations of various algorithms later, and there are plenty of posts that you can read later to brush up on the steps of a machine learning project and the importance of evaluating accuracy using cross validation.

What about other steps in a machine learning project. We did not cover all of the steps in a machine learning project because this is your first project and we need to focus on the key steps. Namely, loading data, looking at the data, evaluating some algorithms and making some predictions. In later tutorials we can look at other data preparation and result improvement tasks.

16.11. Summary

In this post, you discovered step-by-step how to complete your first machine learning project in Python.

You discovered that completing a small end-to-end project from loading the data to making predictions is the best way to get familiar with a new platform.

16.12. Your Next Step

Do you work through the tutorial?

1. Work through the above tutorial.
2. List any questions you have.
3. Search-for or research the answers.
4. Remember, you can use the `help("FunctionName")` in Python to get help on any function.

Do you have a question?

17. Datenanalyse mit Python: Erste Schritte mit Pandas – Fussballdaten auswerten

17.1. Einleitung

Pandas ist die Standard-Bibliothek für alle, die mit Python Daten verarbeiten möchten. So lässt sich auch die erste Bundesligasaison einfach analysieren.

Wer Daten mit Python ver- oder bearbeiten möchte, kommt kaum um die Bibliothek Pandas herum. Sie ist gewissermaßen der Goldstandard für Python, wenn es um strukturierte Informationen geht.

Wie groß die Ursprungsdaten sind, spielt für Pandas keine Rolle: Die Bibliothek kommt mit CSV-Dateien zurecht, dem Excel-Format von Microsoft und auch mit großen Datenmengen in einer JSON-Datei. Dabei ist es egal, ob Sie ein kleines oder großes Datenprojekt angehen möchten, Pandas vereinfacht in der Regel immer den Umgang mit Daten jeder Art.

Wir erklären in diesem Artikel die grundlegenden Konzepte, mit denen Pandas arbeitet: die Series und den Dataframe. Außerdem zeigen wir, wie man sich freie Fußballdaten aus dem Netz holt und so etwa die erste Bundesligasaison 1963/64 analysiert. Grundlegende Python-Kenntnisse sind dafür hilfreich, Sie sollten also schon einmal Bibliotheken mit Pip installiert haben.

Der Begriff Pandas kommt vom Wort Panel Data, also Paneldaten, die in der Regel aus Reihen und Spalten bestehen. Falls man Pandas einem Nicht-Techie erklären möchte: Das ist so, als würde man mit einem Excel-Datenblatt arbeiten. Nur halt im Python-Code.

Der Entwickler Wes McKinney arbeitet seit 2008 an Pandas. Damals war er noch bei einer Management-Firma für Investments angestellt und suchte nach einem schnellen und flexiblen Tool, um quantitative Analysen durchzuführen. Daraus wurde schließlich die Open-Source-Bibliothek Pandas. 28 Maintainer kümmern sich mittlerweile um die Software, McKinney ist noch immer dabei.

17.2. Installation

Über die Paketverwaltung Pip installiert man die Pandas-Bibliothek so:

```
pip install pandas
```

Pandas ist zudem Teil des Sci-Py-Stacks, eine Open-Source-Softwareumgebung für Mathematiker, Analysten und andere Wissenschaftler. Pandas-Entwickler bevorzugen die Installation über die [Python-Distribution Anaconda](#), die unter anderem den Sci-Py-Stack samt Pandas mitbringt.

17.3. [Series](#) verstehen

Die beiden wichtigsten Datenstrukturen, die Pandas bereitstellt, sind die [Series](#) und der [DataFrame](#). Die [Series](#) kann man sich als einfache Liste mit Index vorstellen, sie

besteht also aus zwei Arrays. Mit diesem Code lässt sich eine kleine **Series** mit drei Elementen anlegen:

```
series = pd.Series([1, 2, 3])
```

Gibt man die Variable `series` aus, steht neben den einzelnen Reihen noch der Index, der bei Null beginnt:

```
0 1  
1 2  
2 3
```

Außerdem steht der Datentyp der Werte in der Ausgabe, `int64` bedeutet 64-Bit-Ganzzahlen. Anhand des Indexes kann der Nutzer auf einzelne Werte zugreifen: `print(series[1])` ergibt etwa `2`.

Die **Series** mag wie eine normale Python-Liste wirken, allerdings funktioniert sie ein wenig anders. So kann man etwa den Index verändern und Werte aus der **Series** dann über den neuen Index ansprechen:

```
series = pd.Series([1, 2, 3], index=["Erste Ziffer", "Zweite Ziffer", "Dritte Ziffer"])  
print(series["Zweite Ziffer"])
```

Dieser Code lässt wieder eine `2` in der Ausgabe erscheinen.

Außerdem werden mathematische Operationen auf alle **Series**-Werte angewandt. Eine Mal-Operation wie `series*2` ergibt folgende Ausgabe:

```
0 2  
1 4  
2 6
```

Das ist mit einer Python-Liste so nicht möglich, diese Operation würde nur die Liste verdoppeln, nicht die Werte. Das Prinzip wird weitergeführt, wenn man etwa zwei **Series**-Objekte addieren möchte:

```
series1 = pd.Series([1, 2, 3])  
series2 = pd.Series([4, 5, 6])  
print(series1+series2)
```

Das Ergebnis ist eine neue **Series**, mit den addierten Werten aus den zwei vorherigen:

```
0 5  
1 7  
2 9
```

Wären das zwei normale Listen-Elemente, würde die Ausgabe stattdessen so aussehen:

```
[1, 2, 3, 4, 5, 6]
```

17.4. DataFrame erstellen

Einen **DataFrame** lässt sich ebenfalls einfach per Hand erstellen. Dafür nutzt man das in Python eingebaute **Dictionary**:

```
daten = {  
    "Zahlen": [1, 2, 3],  
    "Buchstaben": ["A", "B", "C"]  
}
```

Das **Dictionary** steht in geschwungenen Klammern und enthält zwei zusammengehörige Daten, die durch Doppelpunkte getrennt sind. Mehrere Datenpaare werden

durch Kommas im **Dictionary** unterschieden. Der erste Datenpunkt ist der Schlüssel, der zweite ist der zugehörige Wert.

In diesem Beispiel steht jedes Schlüssel-Wert-Paar für eine Spalte im Dataframe. Der Schlüssel besteht jeweils aus einem String, hier Zahlen und Buchstaben. Das sind später die Spaltenüberschriften. Der zugehörige Wert besteht in beiden Fällen aus je einer Liste mit drei Elementen, das sind dann die einzelnen Werte in der jeweiligen Spalte.

Das **Dictionary** übergibt man nun Pandas, damit daraus ein **DataFrame** wird:

```
dataframe = pd.DataFrame(daten)
```

pd.DataFrame nimmt das **Dictionary** auf und speichert den **DataFrame** in der Variable **dataframe**. In vielen Pandas-Beispielen findet man auch die Abkürzung **df** als Variablennamen für den **DataFrame**. Gibt man diese Variable anschließend mit einem **print(dataframe)** aus, sieht man die Daten in einer Tabellendarstellung:

	Zahlen	Buchstaben
0	1	A
1	2	B
2	3	C

Einen ersten Überblick über die Daten erhält man mit **shape**. So gibt **print(dataframe.shape)** die Zeile **(3, 2)** aus. Der **DataFrame** besteht also aus 3 Reihen und 2 Spalten. **print(len(dataframe))** zeigt Ihnen nur die Anzahl der Reihen, also **3**. Das ist für die Beispiel-Daten nicht sehr überraschend, man weiß ja, was man da eingetippt hat. Sobald allerdings fremde Daten ins Spiel kommen, werden **shape** und **len()** wieder relevant.

Eine Zusammenfassung des Dataframes zeigt **info()** an. Für den kleinen Beispieldataframe ergibt die Eingabe **print(dataframe.info())** folgende Ausgabe:

```
1000 <class 'pandas.core.frame.DataFrame'>
1001 RangeIndex: 3 entries, 0 to 2
1002 Data columns (total 2 columns):
1003 #   Column      Non-Null Count  Dtype  
1004   --  --          --            --    
1005   0   Zahlen      3 non-null    int64 
1006   1   Buchstaben  3 non-null    object 
1007   dtypes: int64(1), object(1)
1008   memory usage: 176.0+ bytes
```

Auf einen Blick sieht man den Index, die Anzahl der Spalten, wie viele Reihen der Dataframe hat und aus welchen Datentypen die Werte in den Spalten bestehen. Die erste Spalte mit den Zahlen besteht etwa aus 64-Bit-Ganzzahlen, das zeigt der Hinweis **int64** unter **Dtype** – wie schon bei der **Series**. Schließlich zeigt **info()** noch, wie viel Speicherplatz der Dataframe benötigt.

17.5. Daten auswählen

Um aus diesen Beispieldaten nun eine spezielle Spalte auszuwählen, reicht es aus, den Namen der Spalte in eckigen Klammern hinter den Dataframe zu packen. **print(dataframe["Buchstaben"])** hat damit diese Ausgabe:

```
1000 0 A
1001 1 B
1002 2 C
```

Neben den definierten Reihen und Werten und besteht der **DataFrame** noch aus einem Index, der bei Null beginnt. Über diesen Index können Sie etwa eine spezielle Reihe ansprechen, und zwar mit loc:

```
print(dataframe.loc[1])
```

Wer diesen Code eingibt, erhält als Ausgabe nur die Werte der zweiten Reihe – ohne Index:

1000	Zahlen	2
	Buchstaben	B

Den Index kann man auch anpassen, indem man der Funktion **DataFrame()** eine Liste mit neuen Index-Werten übergibt. So wird aus folgendem Code

```
dataframe = pd.DataFrame(daten, index=["Erste Reihe", "Zweite Reihe", "Dritte Reihe"])
```

diese Ausgabe:

		Zahlen	Buchstaben
1000	Erste Reihe	1	A
1002	Zweite Reihe	2	B
	Dritte Reihe	3	C

Die zweite Reihe lässt sich nun über ein `print(dataframe.loc["Zweite Reihe"])` ausgeben.

17.6. Integer

Neben `loc` gibt es noch `iloc` – das `i` steht für Integer. Darüber können Sie den Index unabhängig vom selbst gewählten Namen aufrufen, sondern wieder über seine Index-Zahl. Auch dieser Index beginnt bei Null. `print(dataframe.iloc[1])` ergibt so dieselbe Ausgabe wie `print(dataframe.loc["Zweite Reihe"])`.

Während `loc` und `iloc` ganze Reihen und Spalten auswählen, kann man mit `at` einen einzelnen Wert ermitteln. Dafür über gibt man `at` den Indexwert und den Spaltennamen, damit der passende Wert aus dem **DataFrame** extrahiert wird. In dem Beispiel ohne eigens festgelegten Index gibt etwa `print(dataframe.at[0, "Buchstaben"])` ein einzelnes `A` aus. Verwendet man den eigenen Index von eben, kommt man mit `print(dataframe.at["Erste Reihe", "Buchstaben"])` zum selben Ergebnis.

Analog zu `loc` und `iloc` gibt es neben `at` auch `iat`. Wer keine Buchstaben mag, wird damit glücklich, denn das `i` steht wieder für Integer. Damit das `A` in der Ausgabe erscheint, reicht ein `print(dataframe.iat[0, 1])`.

17.7. Fußball-Daten holen

Mit dem bisherigen Dataframe kann man nur begrenzt herumspielen und was über Pandas lernen. Wir brauchen echte Daten. Im Netz gibts massenhaft freie Datensätze, die man für eigene Projekte verwenden kann. Auf Github findet man etwa alle [Spielergebnisse der Fußball-Bundesliga im CSV-Format](#).

CSV-Dateien enthalten Daten, die durch Kommas getrennt sind. Für Menschen sind die Dateien eher schwierig zu lesen, Maschinen lieben sie. Als Beispiel nehmen wir den ersten Spieltag der allerersten Bundesliga von 1963/64 als CSV-Datei.

Pandas kann natürlich auch mit anderen Formaten umgehen, etwa Excel-Dateien, JSON-Daten oder SQL-Dateien. Eine Übersicht über verschiedene Möglichkeiten zeigt die [Dokumentation](#).

Nun müssen aber erst die Fußball-Daten ins Programm. Mit der Funktion `read_csv()` kann Pandas CSV-Dateien lesen und erstellt daraus sofort einen Dataframe. Wer es eilig hat, packt einfach den Link zu den Daten in Anführungszeichen und übergibt ihn `read_csv()`:

```
dataframe = pd.read_csv("https://raw.githubusercontent.com/footballcsv/deutschland/master/1964/de.1.csv")
```

Optimal ist das nicht. Die Daten könnten schließlich aus dem Netz verschwinden und dann würde das ganze Programm nicht mehr funktionieren. Im Internet ist schließlich nichts für die Ewigkeit.

Eleganter ist es, die CSV-Datei herunterzuladen und zu speichern. Pandas kann dann auf die gesicherte Datei zugreifen. Das geht am einfachsten über die Python-Bibliothek `requests`. Die müssen Sie erst installieren, am besten über Pip:

```
pip install requests
```

Die Datei holen Sie dann so auf die Platte:

```
import requests
```

```
bundesliga_csv_url = "https://raw.githubusercontent.com/footballcsv/deutschland/master/1964/de.1.csv"
```

```
download = requests.get(bundesliga_csv_url)
with open("bundesliga_erster_spieltag.csv", "wb") as datei:
    datei.write(download.content)
```

In der Variable `bundesliga_csv_url` steht die URL zur CSV-Datei, `requests.get(bundesliga_csv_url)` holt sich dann über die Get-Methode die Daten. Die zwei weiteren Zeilen speichern den Inhalt, also `download.content`, als CSV-Datei ab. `bundesliga_erster_spieltag.csv` ist anschließend der Name der daraus erstellten CSV-Datei.

Die Datei wird hier in dem Ordner gespeichert, in dem auch die Python-Datei liegt. Das Skript führt den Download jedes Mal neu aus. Sie können die paar Zeilen allerdings auch in ein Extra-Skript packen und nur einmal ausführen.

Pandas soll nun mit den heruntergeladenen Daten arbeiten. Da die Datei im gleichen Ordner wie das Skript liegt, reicht es aus, `dataframe = pd.read_csv("bundesliga_erster_spieltag.csv")` einzugeben. Wenn Sie die Datei woanders speichern, müssen Sie den Pfad entsprechend anpassen.

17.8. Infos anzeigen

Mit den Methoden von vorhin lässt man sich nun einige Informationen zu dem `DataFrame` anzeigen:

```
print(len(dataframe))
print(dataframe.shape)
print(dataframe.info())
```

`len(dataframe)` gibt wieder die Anzahl der Reihen aus, hier **240 – 16** Mannschaften absolvieren je **30** Spiele, das ergibt **240** Partien. Der `Shape` ist **(240, 5)**, also **240** Reihen und **5** Spalten. `Info()` zeigt schließlich die Namen der Spalten: `Matchday`, `Date`, `Team 1`, `FT` und `Team 2`. Nur die Datenpunkte in `Matchday` sind jeweils 64-Bit-Integer, der Rest besteht aus unbestimmten Typen `object`.

Damit weiß man schon jede Menge über die Daten, aber was genau steht drin? Sind es wirklich Fußball-Informationen oder hat jemand die Spalten mit Kochrezepten gefüllt? Gerade wenn man mit fremden Daten arbeitet, kommt man nicht drum herum, mal reinzuschauen.

Einen ersten Blick ermöglicht `head()`. Damit holt man sich schnell die ersten 5 Einträge in die Ausgabe. `print(dataframe.head())` ergibt in unserem Beispiel also

	Matchday	Date	Team 1	FT	
	Team 2				
0	1 Sat Aug 24 1963	Werder Bremen	3–2	Borussia	
1	Dortmund	1. FC Saarbrücken	0–2	1. FC	
2	Köln	TSV 1860 München	1–1	Eintracht	
3	Braunschweig	Eintracht Frankfurt	1–1	1. FC	
4	Kaiserslautern	FC Schalke 04	2–0	VfB	
	Stuttgart				

Glück gehabt, die ersten Reihen zeigen Fußball-Daten. `FT` steht übrigens für Full Time, gibt also das Ergebnis nach der vollen Spielzeit an. `print(dataframe.tail())` zeigt die letzten 5 Einträge an:

	Matchday	Date	Team 1	FT	
	Team 2				
235	30 Sat May 9 1964	Karlsruher SC	1–2	Eintracht	
236	Frankfurt	TSV 1860 München	3–2	Werder	
237	Bremen	1. FC Saarbrücken	1–1	FC Schalke	
238	04	Preussen Münster	4–2	Hertha	
239	BSC	MSV Duisburg	3–0	1. FC	
	Kaiserslautern				

5 Einträge ist der Standardwert, das kann man allerdings ändern: `dataframe.head(3)` zeigt nur die ersten 3 Einträge, für `tail()` funktioniert das genauso. Wer noch Daten aus der Mitte des Dataframes sehen möchte, kann sie mit einem einfachen `Slice` anzeigen, etwa `print(dataframe[112:116])`.

17.9. Tor-Statistiken

Mit diesen wenigen Befehlen weiß der Nutzer nun, mit welchen Daten er es zu tun hat. Jetzt holen wir einige interessante Statistiken aus den Daten. Die Funktion `value_counts()` zeigt etwa, wie oft ein bestimmter Wert in den Daten vorkommt. Wenn man ihn auf die Spalte `FT` mit den Ergebnissen anwendet, weiß man sofort, welches Endergebnis am häufigsten auf der Anzeigetafel stand. Ein angehängtes `.head(10)` würde übrigens nur die 10 häufigsten Ergebnisse ausgeben.

Die erste Bundesligasaison muss demnach recht langweilig gewesen sein. `print(dataframe["FT"].value_counts())` zeigt, dass ein 1:1 am häufigsten vorkam, nämlich 25 Mal. Direkt dahinter wird es nicht besser, 2:2 war 18 Mal das Endergebnis. Ein 9:3, 6:1 oder 7:0 war eher die Ausnahme, diese Ergebnisse kamen jeweils nur ein Mal vor. Übrigens ist Fußball nicht gerade interessanter geworden. In der Saison 2019/20 war 1:1 ebenfalls das

häufigste Ergebnis, es kam 32 Mal vor – immerhin gab es mehr 2:1- und 1:2-Ergebnisse als 1963/64, dafür weniger Torfestival-Partien.

Alle Ergebnisse eines bestimmten Spieltags kann man sich über die Spalte Matchday anzeigen lassen. Für den 5. Spieltag muss der Wert in der Spalte 5 betragen. `fuenfter_spieltag = dataframe["Matchday"] == 5` ergibt eine `Series` mit `True`- und `False`-Werten – ist der 5. Spieltag, zeigt die `Series True` an, alle anderen Spieltage sind `False`. Diese `Series` wendet man anschließend auf den `DataFrame` an:

```
print(dataframe[fuenfter_spieltag])
```

So werden alle Reihen ausgegeben, die zum 5. Spieltag gehören – also alle 8 Spiele. Jetzt wollen wir nicht nur einen bestimmten Spieltag, sondern die Spiele einer bestimmten Mannschaft. Das funktioniert im Grunde wie eben: Die Team-Spalten filtert man nach dem Verein und die daraus entstehende True-False-Series wendet man anschließend auf den Dataframe an. Hier muss man allerdings zwei Spalten durchforsten, Team 1 und Team 2. Der Nutzer möchte ja alle Spiele aufgelistet haben und nicht nur die Heim- oder Auswärts-Spiele.

`dataframe["Team 1"] == "Borussia Dortmund"` durchforstet die Heimspiele von Borussia Dortmund, `dataframe["Team 2"] == "Borussia Dortmund"` die Auswärtsspiele. Der Oder-Operator `|` verbindet beide Befehle, die dafür in Klammern stehen müssen:

```
spiele_bvb = (dataframe["Team 1"] == "Borussia Dortmund") | (dataframe["Team 2"] == "Borussia Dortmund")
```

`spiele_bvb` ist anschließend eine Series, die immer dann `True` ist, wenn Borussia Dortmund spielt. Wendet man diese Series anschließend auf den gesamten Dataframe an, erhält man eine Liste mit allen Spielen des Vereins:

```
print(dataframe[spiele_bvb])
```

17.10. Daten formatieren

Daten aus öffentlichen Quellen haben selten das Format oder die Bezeichnungen, die man gerne hätte. Bei den Bundesliga-Daten nerven etwa die englischen Spaltennamen. Deutsche Bezeichnungen wären da passender.

Dafür hat Pandas die praktische Funktion `rename()`. Die Änderungen packt man einfach in ein `Dictionary` und übergibt es der Funktion:

```
1000 dataframe = dataframe.rename(columns={"Matchday": "Spieltag",
1001     "Date": "Datum",
1002     "Team 1": "Heimmannschaft",
1003     "FT": "Endergebnis",
1004     "Team 2": "Gastmannschaft"})
```

Aus Matchday wird Spieltag, aus `Team 2` Gastmannschaft und so weiter. Etwas übersichtlicher ist es, das `Dictionary` außerhalb der Funktion zu definieren und dann einfach aufzurufen:

```
1000 spaltennamen_ersetzen = {"Matchday": "Spieltag",
1001     "Date": "Datum",
1002     "Team 1": "Heimmannschaft",
1003     "FT": "Endergebnis",
1004     "Team 2": "Gastmannschaft"
1005 }
1006 dataframe = dataframe.rename(columns=spaltennamen_ersetzen)
```

17.11. Datum

Damit sind die Spaltennamen korrekt eingedeutscht, aber die Datumswerte sind noch nicht formatiert. Beim ersten Spieltag steht etwa `Sat Aug 24 1963`. Das geht besser. Pandas bringt dafür die Funktion `to_datetime()` mit:

```
dataframe["Datum"] = pd.to_datetime(dataframe["Datum"])
```

Die Funktion `to_datetime()` füttert man mit der Datumsspalte. Die alte Datumsspalte wird dann mit der neuen, formatierten Datumsspalte ersetzt. Gibt man die neue Spalte aus, wird der erste Spieltag nun als `1963-08-24` gelistet.

`print(dataframe.info())` zeigt, dass die Datum-Spalte nun kein Type `object` mehr ist, sondern ein Type `datetime64[ns]`. Das sind interne 64-Bit-Integer, die in Nanosekunden (ns) gespeichert werden. `datetime64[ns]` kann Daten zwischen 1678 und 2262 speichern, plus-minus 292 Jahre vom 01. Januar 1970 aus gerechnet, dem Start der Unixzeit.

`1963-08-24` ist ja schon ganz nett, aber wenn wir alles eindeutschen, dann auch das Datum. Dafür ist die Funktion `dt.strftime()` verantwortlich:

```
dataframe["Datum"] = pd.to_datetime(dataframe["Datum"]).dt.strftime("%A, %d. %B %Y")
```

Die Funktion verwandelt ein Datum-Element in einen String. Mit speziellen Formatierungscodes `Datetime` bestimmt man den Inhalt des Strings. `%A` ist der ausgeschriebene Wochentag, `%d` der Tag, `%B` der ausgeschriebene Monat und `%Y` das Jahr mit vier Ziffern.

Damit die Formatierungscodes deutsche Wochentage und Monate anzeigen, muss der Nutzer vorher eventuell noch die Bibliothek `locale` einbinden. Sie ist standardmäßig in Python integriert, muss also nicht per Pip installiert werden und stellt länderspezifische Datenformate zur Verfügung:

```
import locale
locale.setlocale(locale.LC_ALL, "de_DE")
```

`LC_ALL` bedeutet, dass alle länderspezifischen Formate angesprochen werden. `de_DE` ist ein Language Code Identifier und bedeutet quasi: Deutsch in Deutschland. `de_AT` wäre etwa Deutsch in Österreich.

Gibt man nun das Datum aus, steht unter anderem diese Zeile in der Ausgabe:

`Samstag, 24. August 1963`

Damit ist die Datumsspalte aber wieder ein Type `object` und kein Type `datetime64` mehr – alle String-Elemente sind für Pandas ein Type `object`. Hier steht allerdings die Lesbarkeit durch den Menschen im Vordergrund, nicht die Maschinenlesbarkeit. Um die Daten in der Spalte wieder in ein Type `datetime64` zu verwandeln, wendet man die Funktion `to_datetime()` erneut an, gibt ihr aber die vorher genutzten Formatierungscodes von `dt.strftime()` als Parameter mit:

```
dataframe["Datum"] = pd.to_datetime(dataframe["Datum"], format="%A, %d. %B %Y")
```

Anhand des Parameters `format` erkennt die Funktion den Inhalt des Strings und kann als das korrekte Datum formatieren.

18. Pandas Gene

18.1. Datamining in sequenzierten Gendaten mit Pandas

Wer sein Genom sequenzieren lässt, bekommt die Rohdaten als CSV-Datei mit hunderttausenden Zeilen. Das Python-Framework Pandas beweist sich gerade dann als Schweizer Messer der Datenanalyse, wenn die Tabellen wie bei Gendaten zu groß für grafische Tabellenkalkulationen wie Excel werden.

18.2. Einführung

Sequenzierungsdienste fürs eigene Genom helfen bei der Ahnenforschung und das Risiko für manche erbliche Krankheiten zu schätzen. Neben hübsch aufbereiteten Zusammenfassungen liefern die Anbieter auch Rohdaten, die sie als CSV-Dateien von circa 20 Megabyte Größe verschicken. So große Dateien verarbeiten Excel, LibreOffice und Konsorten nicht mehr in erträglicher Geschwindigkeit. Das Python-Framework Pandas dagegen setzt unter der Haube auf die effizienten Datenmodelle von Numpy und analysiert Tabellen dieser Größe daher in Sekundenbruchteilen.

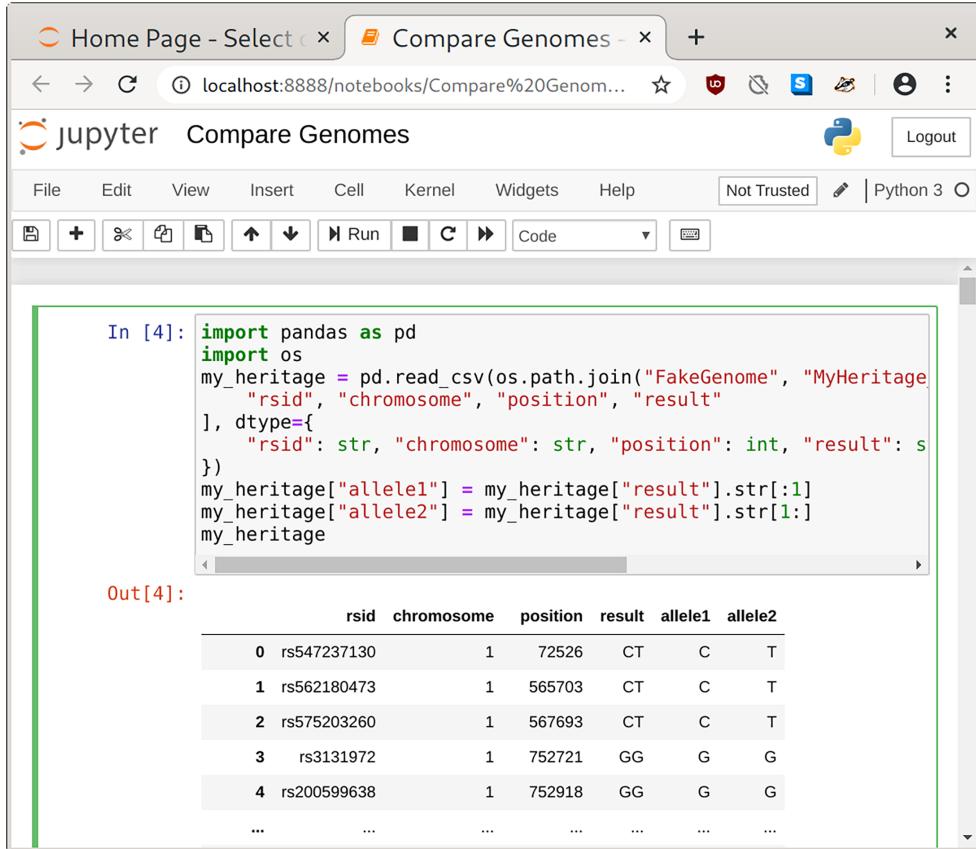
Zwecks Analyse ihrer DNA haben c't-Redakteure Proben an mehrere Ahnenforschungs-Plattformen geschickt [1] und mir von jedem Anbieter einen Satz mit Rohdaten zur Auswertung ausgehändigt. Die Dateien nutzen leicht unterschiedliche Formate, mal kommasepariert, mal mit Tabs, mal mit getrennt gelisteten Allelen (eine Base von den Genen der Mutter, eine von denen des Vaters), mal mit beiden Basen als String aus zwei Großbuchstaben. Die Daten enthalten kein vollständiges Genom, sondern Zeilen mit SNPs, also den Basen interessanter Mutationen. Die Anbieter ordnen jedem SNP eine Bedeutung beispielsweise für ein Krankheitsrisiko zu. Zu einem Identifier für das SNP wie „rs4475691“ steht in den Daten jeweils die Position im Genom als Zahl (846808) und das Chromosom, in dem das Basenpaar vorkommt (Nummer 1). Damit Sie den in diesem Artikel erklärten Pandas-Code nachvollziehen können, ohne gleich Ihr Genom analysieren zu lassen, finden Sie auf GitHub CSV-Dateien im gleichen Format, aber mit künstlich erzeugten, zufälligen Angaben zu den Basen (siehe ct.de/ybs2).

18.3. Experimentierumgebung

Mit einem Jupyter-Notebook, ein Python-Interpreter im Browser, geht die Genanalyse leicht von der Hand, da es Pandas-Dataframes, die Tabellenobjekte des Frameworks, grafisch als Tabellen aufbereitet. Eine virtuelle Umgebung mit Jupyter und Pandas ist schnell eingerichtet, zum Beispiel unter Linux mit folgenden Kommandozeilenbefehlen:

```
mkdir PandasGenom  
cd PandasGenom/  
python3 -m venv env  
source env/bin/activate  
pip install pandas, jupyter  
jupyter notebook
```

Der letzte Befehl öffnet ein Browserfenster mit einem Knopf zum Anlegen neuer Notebooks. Jupyter-Notebooks sind Textdokumente, in denen man neben Text auch in "Zellen" Code schreiben und mit Umschalt+Enter ausführen kann. Gibt der letzte



The screenshot shows a Jupyter Notebook interface in a browser window. The title bar says "Compare Genomes". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a "Not Trusted" status. Below the menu is a toolbar with various icons. The main area has two sections: "In [4]" and "Out[4]".

```
In [4]: import pandas as pd
import os
my_heritage = pd.read_csv(os.path.join("FakeGenome", "MyHeritage",
    "rsid", "chromosome", "position", "result"
], dtype={
    "rsid": str, "chromosome": str, "position": int, "result": s
})
my_heritage["allele1"] = my_heritage["result"].str[:1]
my_heritage["allele2"] = my_heritage["result"].str[1:]
my_heritage
```

Out[4]:

	rsid	chromosome	position	result	allele1	allele2
0	rs547237130	1	72526	CT	C	T
1	rs562180473	1	565703	CT	C	T
2	rs575203260	1	567693	CT	C	T
3	rs3131972	1	752721	GG	G	G
4	rs200599638	1	752918	GG	G	G
...

Figure 18.1.: In Jupyter-Notebooks programmiert man Python im Browser. Pandas Dataframes zeigt Jupyter praktischerweise als gekürzte Tabelle an.

Befehl einer Zelle etwas Darstellbares zurück, zeigt Jupyter diesen Wert unter der Zelle an. Steht beispielsweise in der letzten Zeile einer Zelle ein Pandas-Dataframe als Variable, zeigt Jupyter die Tabelle gekürzt an (die fünf ersten und letzten Zeilen).

18.4. CSVs einlesen

Die ersten Zeilen der ersten Zelle laden wie in Python-Skripten üblich per import die Bibliothek:

```
import pandas as pd
import os
```

Das Modul `os` dient lediglich dazu, den Pfad der Dateien zusammenzubauen:

```
ingo_my_heritage = pd.read_csv(os.path.join("FakeGenome", "MyHeritage_-
raw_dna_data_fake.csv"),
    header=12, names=["rsid", "chromosome", "position", "result"],
    dtype="rsid": str, "chromosome": str, "position": int, "result": str)
```

Ums Einlesen kümmert sich `pd.read_csv()`. Der Parameter `header` teilt der Funktion mit, in welcher Zeile der Datei die Spaltenbeschriftungen stehen. Ein Nebeneffekt dieser Angabe ist, dass Pandas alle Zeilen davor ignoriert, was effektiv den Header abschneidet. Die Spaltennamen kann man nämlich mit einer Liste an `names` auch

per Hand festlegen, falls die Datei keine zufriedenstellenden Spaltennamen liefert. Pandas bestimmt die Datentypen der Spalten normalerweise automatisch, arbeitet aber schneller, wenn man die Typen als Dictionary im Parameter `dtype` definiert. Die Keys im Dictionary entsprechen den bei `names` festgelegten Spaltennamen.

Die Daten von My Heritage fassen die Allele 1 und 2 im Schlüssel `"result"` zusammen. Um sie in einzelne Spalten aufzuteilen, schneidet der folgende Code die Allele als Slice heraus:

```
my_heritage["allele1"] = my_heritage["result"].str[:1]
my_heritage["allele2"] = my_heritage["result"].str[1:]
```

My Heritage hat praktischerweise die Werte mit Komma getrennt, was `read_csv()` als Standard annimmt. Für die tabulatorseparierten Daten von Ancestry muss man den Tab als Separator angeben: `sep="\t"`. Dafür enthält diese Datei eine schöne header-Zeile, was die Angabe der `names` spart:

```
ancestry = pd.read_csv(os.path.join("FakeGenome", "AncestryDNA_fake.txt"),
                      sep="\t", header=18, dtype={
                          "rsid": str, "chromosome": str, "position": int, "allele1": str,
                          "allele2": str})
```

Um auf die gleichen Spalten wie bei My Heritage zu kommen, braucht die Ancestry-Tabelle noch ein `result` kombiniert aus beiden Allelen:

```
ancestry["result"] = ancestry["allele1"] + ancestry["allele2"]
```

Danach liegen beide Tabellen als Dataframe-Objekte mit den gleichen Spalten vor. Die Auswertung kann beginnen.

18.5. Interessante Stellen

Jeder Anbieter scheint eine eigene Vorstellung davon zu haben, welche SNPs interessant sind (Spalte „rsid“). Ich habe mich gefragt, wie viel Überschneidung es dabei gibt. Pandas beantwortet diese Frage mit der Funktion `isin()`. Sie berechnet eine ganze Spalte an Wahrheitswerten. Ein `True` steht nur in den Zeilen, bei denen der Wert auch in der angegebenen Spalte vorkommt. `value_counts()` zählt, wie häufig `True` und `False` in der Spalte vorkommen und listet die Zahlen auf:

```
my_heritage["rsid"].isin(ancestry["rsid"]).value_counts()
```

Ergebnis: Die Anbieter sind sich nur bei einem Drittel der SNPs einig, dass diese interessant sind. 405.910 der von My Heritage ausgewählten SNPs kommen bei Ancestry gar nicht vor.

Eine Liste aller Zeilen, deren SNPs nur My Heritage misst, liefert die folgende Zeile:

```
my_heritage[ my_heritage["rsid"].isin(ancestry["rsid"])]
```

Sie verwendet die zuvor berechnete Spalte als Auswahlkriterium, allerdings kehrt die vorangestellte Tilde den Wahrheitswert um. Übrig bleiben so nur die einzigartigen Zeilen.

18.6. Prüfvereinigung

Ein Kollege hat neben den Daten von Ancestry und My Heritage auch einen Datensatz von 23 And Me beigesteuert. Für die 171.012 SNPs, die alle drei Anbieter gemessen haben, wollte ich wissen, wie viele Unterschiede es gibt, da das auf Messfehler oder Mutationen der gemessenen Zelle hinweisen würde.

```
In [7]: my_heritage["rsid"].isin(ancestry["rsid"]).value_counts()
Out[7]: False    405910
         True    203436
Name: rsid, dtype: int64
```

Figure 18.2.: Die Funktion `value_counts()` berechnet ein Histogramme der Werte in einer Spalte. Hier zeigt sie, welche SNPs sowohl My Heritage als auch Ancestry messen.

Für den Vergleich müssen die Daten aus den drei Dataframes in einen einzelnen zusammenfließen. Mit einem Index kann Pandas zuordnen, welche Zeilen zusammengehören und wo es fehlende Werte mit None belegen muss. `set_index()` markiert eine Spalte als Index, wodurch ein `join()` möglich wird:

```
rsid_indexed_my_heritage = my_heritage.set_index("rsid")
rsid_indexed_ancestry = ancestry.set_index("rsid")
rsid_indexed_23andme = fake23andme.set_index("rsid")
```

Danach fügt Pandas die Tabellen mit `.join()` klaglos zusammen, wobei `lsuffix="_-my_heritage"` und `rsuffix="_23andme"` dafür sorgen, dass Pandas Spalten mit gleichen Namen automatisch umbenennat:

```
joined_df = rsid_indexed_my_heritage.join(rsid_indexed_23andme, how="inner",
                                         lsuffix="_my_heritage",
                                         rsuffix="_23andme").join(rsid_indexed_ancestry, how="inner")
```

Die Option `how` wählt zwischen den vier von SQL bekannten Join-Typen `"inner"`, `"left"`, `"right"` und `"outer"`. Für den Vergleich der Messwerte sind nur Zeilen interessant, die es bei beiden Anbietern gibt, weshalb ein Inner-Join gern alle Zeilen verwerfen darf, die nur einer der Anbieter kennt.

Den so erweiterten Dataframe ergänzt ein weiterer Join um die Daten von Ancestry. Dabei ist kein Suffix nötig, da sich keine Spaltennamen doppeln. Damit erkennbar bleibt, dass die ergänzten Daten von Ancestry stammen, benennt ein `.rename()` die Spaltennamen um:

```
joined_df = joined_df.rename(columns={
    "chromosome": "chromosome_ancestry",
    "position": "position_ancestry",
    "result": "result_ancestry",
    "allele1": "allele1_ancestry",
    "allele2": "allele2_ancestry"
})
```

Ein Problem beim Vergleich ergibt sich daraus, dass die Anbieter die beiden Allele unterschiedlich sortieren. Mal stehen sie alphabetisch sortiert mit A links, mal umgekehrt. Eine for-Schleife über die drei Anbieter sortiert die Basen ruckzuck per lambda-Funktion in der `result`-Spalte und ergänzt beim Y-Chromosom, das es nur einmal geben kann, kurzerhand den zweiten Wert durch Kopieren des ersten:

```
for tester in ["_my_heritage", "_ancestry", "_23andme"]:
    joined_df["result" + tester] = (joined_df["allele1" + tester] + joined_df["allele2" + tester]).apply(lambda x: x if len(x) < 2 or x < x[1] + x[0] else x[1] + x[0])
    joined_df["result" + tester] = joined_df["result" + tester].apply(lambda x: x + x if len(x) < 2 else x)
```

Mit dieser Vorarbeit besteht der Test nur noch aus drei einzelnen Vergleichen mit `==`, die Pandas aber mit `&` statt wie bei Python sonst üblich mit `and` verknüpft haben möchte:

```
joined_df["allele_match"] = (
    (joined_df["result_ancestry"] == joined_df["result_23andme"]) &
    (joined_df["result_ancestry"] == joined_df["result_my_heritage"])
&
    (joined_df["result_23andme"] == joined_df["result_my_heritage"]))
```

Das Histogramm (`value_counts()`) für die Spalte `allele_match`) bescheinigt anschließend bei den zufälligen Daten überwiegend Abweichungen. Im echten Datensatz erlaubten sich die Anbieter jedoch keinen einzigen Fehler.

18.7. Daten gebändigt

Das Beispiel zeigt anschaulich, wie die Arbeit mit Pandas aussehen kann. Weitere Inspirationen liefert das Jupyter-Notebook im Git-Repository zum Artikel. Dort finden Sie auch den Code zum Erzeugen der Zufallsdatensätze.

Pandas erweist sich gerade dann als nützlich, wenn die Tabellen wie bei den analysierten Gendaten zu groß für grafische Tabellenkalkulationen werden. Dazu kommt, dass mit Pandas-Dataframes noch einiges mehr möglich ist, beispielsweise schicke Diagramme mit dem Framework Altair [2]. (pmk@ct.de)

18.8. Literatur

- Arne Grävemeyer, Jan-Keno Janssen, Nicht nur für Ahnenforscher: DNA gibt persönliche Details preis, Das steckt in mir, c't 5/2020, S. 24
- Pina Merkert, Plotexpress, Datenvisualisierung mit Python, c't 18/2019, S. 80

19. Datenvisualisierung mit Python

19.1. Plotexpress

Mit etwas Übung schreibt man Code schneller, als man klickt. Mit den Frameworks Pandas und Altair bereitet man in einer Handvoll Python-Zeilen Datensätze auf und erzeugt schicke Diagramme. Altair basiert auf dem JavaScript-Framework Vega, sodass sich die Plots mühelos im Web einbinden lassen – auch als interaktive Grafiken.

19.2. Einführung

Ein Klick auf den Download-Link befördert eine 25 Megabyte große CSV-Datei ins Download-Verzeichnis. Ein Doppelklick darauf startet LibreOffice und danach heißt es warten. Der Import dauert nämlich auf einem nicht ganz taufrischen i5 über 30 Sekunden. Als die Riesen-Tabelle endlich erscheint, ungeduldig die Formel für den Durchschnitt in die freie Spalte ganz rechts tippen und anschließend eine Minute lang scrollen, um die Formel auf alle Zeilen zu übertragen. Das fühlt sich alles ziemlich zäh an. Führt man sich vor Augen, dass Office die Daten alle in sein Interface rendern muss, fällt auf: Das Programm ist eigentlich nicht langsam. Es ist nur nicht für so große Tabellen gemacht.

Damit das alles flotter geht, muss das grafische Interface weichen. Stattdessen kommt Python zum Einsatz, genauer die Bibliothek Pandas. Pandas nutzt unter der Haube Numpy und damit schnellen C-Code, um Arrays effizient zu speichern. Statt in `numpy.ndarray` (ohne Spaltennamen) landet alles in Objekten vom Typ `pandas.DataFrame`. So ein `DataFrame` ist gewissermaßen ein Tabellenblatt für Programmierer, das die gleichen Funktionen wie Excel bereitstellt. Für Numpy-Veteranen bekannt, für Python eher ungewöhnlich: Die Daten in `DataFrames` sind hart typisiert. Wenn Sie ein `DataFrame` anlegen, müssen Sie daher gleich festlegen, ob dort Gleitkomazahlen, Ganzzahlen, Strings oder Zeitangaben in den Spalten stehen. Außerdem sind Spalten generell benannt, was in großen Tabelle für Übersicht sorgt.

Das Framework Altair zeichnet aus einem DataFrame mit einer Handvoll Zeilen ein hübsches Diagramm. Es bietet dafür eine kürzere und logischere Syntax als andere Python-Plotting-Bibliotheken wie Matplotlib. Altair erfindet das Rad nicht neu, sondern setzt intern auf der JavaScript-Plotting-Bibliothek Vega (oder Vega-Lite) auf, weshalb das Framework ohne Mehraufwand neben PNGs und SVGs auch Webseiten exportiert. Altairs integrierte Web-Affinität nutzt man am bequemsten, indem man den gesamten Python-Code gleich in einem Jupyter-Notebook schreibt. Aktiviert man dort nämlich mit einer Zeile den passenden Renderer, erscheinen die Diagramme direkt in der Weboberfläche des Notebook.

Damit die Datenanalyse mit Python Spaß macht, brauchen Sie also Pandas, Altair, Vega und Jupyter. Wie Sie diese Pakete am leichtesten beschaffen, hängt vom Betriebssystem ab.

19.3. Installation: Anaconda

Unter Windows haben wir gute Erfahrungen mit der Python-Distribution Anaconda gemacht, deren Installer Sie unter www.anaconda.com/distribution/ herunterladen. Anaconda arbeitet mit “Umgebungen”, die die Abhängigkeiten verschiedener

Python-Projekte voneinander abschotten. Lassen Sie den Installer ruhig eine Default-Umgebung einrichten. Die trägt alle nötigen Pfade in Umgebungsvariablen ein. Anaconda aktiviert die Umgebung auch gleich automatisch in allen neu gestarteten Konsolenfenstern. Wenn Sie das stört, schalten Sie es einfach nach der Installation mit folgendem Befehl ab:

```
conda config -set auto_activate_base false
```

Wenn Sie nun eine getrennte Umgebung namens "datavis" für Ihr Projekt einrichten wollen, geht das mit folgendem Befehl:

```
conda create -n datavis python=3.7
```

Nutzen Sie am besten Python 3.7 und nicht die veraltete Version 2.7. Falls Ihre Shell den Befehl conda nicht findet, geben Sie den ganzen Pfad an, beispielsweise:

```
/anaconda3/bin/conda create -n datavis python=3.7
```

Unter Windows erzeugen und aktivieren Sie die Umgebungen über ein Menü. Linux-Nutzer von Anaconda aktivieren die Umgebung mit folgendem Befehl:

```
conda activate datavis
```

Die Bibliotheken installieren dann folgende Befehle auf der Konsole (unter Windows starten Sie dafür innerhalb der Umgebung eine Anaconda-Konsole):

```
conda install numpy pandas
conda install vega altair
conda install jupyter
```

19.4. Installation: Pip

Linuxer sparen ein paar Befehle mit der Python-Umgebung des Systems. Virtualenv erzeugt eine virtuelle Umgebung:

```
mkdir datavis
cd datavis
python3 -m venv env
source env/bin/activate
```

Pip installiert die Bibliotheken danach mit folgenden Befehlen:

```
pip install wheel numpy pandas
pip install vega altair
pip install jupyter
```

19.5. Wie gebe ich Geld aus?

Mit der perfekt vorbereiteten Arbeitsumgebung kann die Analyse losgehen. Starten Sie dafür den lokalen Webserver des Jupyter-Notebooks:

```
jupyter notebook
```

Der Befehl öffnet auch gleich ein Browser-Fenster mit der URL 127.0.0.1:8888/tree, das eine Übersicht anzeigt. Mit dem Menü unter "New" (oben rechts) starten Sie ein neues Notebook für Python3.

In dessen erste Zelle importieren Sie Pandas, Numpy und Altair und aktivieren Altairs Renderer, damit das Notebook die Plots ohne Umschweife zeichnet:

```
import pandas as pd
```

```
import numpy as np
import altair as alt
alt.renderers.enable('notebook')
```

Den Code einer Zelle im Jupyter-Notebook führen Sie mit Shift+Enter aus.

Nun gilt es, einen Datensatz zu besorgen. Für einen schnellen Start habe ich eine CSV-Datei aus meinem Online-Banking exportiert und jeder meiner Ausgaben über ein Jahr eine Kategorie zugeordnet. `bankdaten.csv` finden Sie im Git-Repository zu diesem Artikel über ct.de/ybcz. Die Datei enthält die Spalten “Buchungstag”, “Betrag” und “Kategorie” und alle 627 Ausgaben vom 1. Juni 2018 bis 30. Juni 2019. Falls Sie lieber eigene Bankdaten analysieren möchten, passen Sie `categorize_bank-data.py` aus dem Repository an Ihre Bedürfnisse an.

Altair erwartet als Datenquelle immer ein Pandas-DataFrame. Pandas bringt eine mächtige Import-Funktion für CSV-Dateien mit, der Sie lediglich den Dateinamen mitteilen müssen:

```
df = pd.read_csv("bankdaten.csv")
```

Wenn Sie die Tabelle nun mit `df` anzeigen (Jupyter rendert sie automatisch als in der Mitte gekürzte HTML-Tabelle) sehen Sie ganz links eine Spalte mit dem von Pandas automatisch erzeugten Index der Zeilen und daneben einen identischen Index, der in der ersten Spalte der CSV-Datei steht. Um den wegzulassen, müssen Sie nur die gewünschten Spalten beim Import angeben:

```
df = pd.read_csv("bankdaten.csv", usecols=[1, 2, 3])
```

`df`, das `DataFrame`, ist nun bereit, um damit ein Diagramm zu zeichnen. Dafür frisst zuerst `alt.Chart()` den Datensatz. Das Objekt bringt die Methode `mark_bar()` mit, um intern auf ein Balkendiagramm umzustellen. Welche Spalten zu welchen Achsen des Diagramms gehören, legt anschließend die Methode `encode()` fest. Ihr gibt man die Spalten als Parameter mit. In Kombination sieht das folgendermaßen aus:

```
alt.Chart(df).mark_bar().encode(x="Buchungstag", y="Betrag")
```

Dabei entsteht ein enorm breites Balkendiagramm mit einem Balken pro Zeile im `DataFrame`. Darin sieht man leicht Ausreißer, die Übersicht geht aber verloren. Das Diagramm spart viel Platz, wenn es die Ausgaben pro Monat nur als Summe darstellt. Für solche Nöte beim Datenauswerten bringt Altair Funktionen mit, sodass man das `DataFrame` nicht anpassen muss:

```
alt.Chart(df).mark_bar().encode(
    x="yearmonth(Buchungstag):0",
    y="sum(Betrag)")
```

Die Funktion `yearmonth()` fasst auf der x-Achse die Datumsangaben in der Spalte “Buchungstag” zu Monaten zusammen. Obwohl Altair die Datumsangaben dadurch auf Monate rundet, geht es trotzdem von Tagen als kontinuierlicher Basisgröße aus. Das produziert sehr schmale Balken mit großen Abständen. Das Angehängte `:0` teilt Altair mit, dass es die Monate stattdessen als aufzählbare Größe behandeln soll, was zu sinnvoll breiten Balken führt.

Die Funktion `sum()` bei der y-Achse addiert alle Ausgaben, die auf der x-Achse denselben Wert haben – was hier zur Summe der Ausgaben in einem Monat führt. Das entstehende Diagramm zeigt mit seinen sehr unterschiedlich hohen blauen Balken, dass ich je nach Monat sehr unterschiedlich viel Geld ausgebe.

Die Achsenbeschriftung lässt etwas zu wünschen übrig, da sie die Funktionen berücksichtigt und den deutschen Spaltennamen automatisch in Englisch ergänzt. Um die Achsenbeschriftung per Hand zu setzen, bringt Altair die Klassen X und Y mit:

```
alt.Chart(df).mark_bar().encode(
    x=alt.X("yearmonth(Buchungstag):0", title="Monat"),
```

```
y=alt.Y("sum(Betrag)", title="Monatliche Ausgaben"))
```

Leider zeigt das blaue Diagramm nun aber nicht, aus welcher Kategorie die hohen Ausgaben stammen. Das könnte man gut sehen, wenn jede Kategorie einen eigenen Balken in einer eigenen Farbe pro Monat hätte und sich diese Balken so stapeln, dass die Höhe des Turms wieder die Gesamtsumme zeigt. Das klingt kompliziert? Mit Altair genügt dafür ein zusätzlicher Parameter:

```
alt.Chart(df).mark_bar().encode(
    x=alt.X("yearmonth(Buchungstag):0",
            title="Monat"),
    y=alt.Y("sum(Betrag)",
            title="Monatliche Ausgaben"))
    color="Kategorie").properties(width=700, height=400)
```

Die zuletzt angehängte Funktion `properties()` legt zusätzlich nur noch die Größe des Diagramms fest.

19.6. Folgt das Gewicht dem Essen?

Balkendiagramme sind mit Altair also einfach. Wie sieht es aber mit Diagrammen aus, an denen man erkennt, dass zwei Größen korrelieren?

Ein persönliches Beispiel: Ich habe über viele Monate mit mehreren Apps protokolliert, wie viele Kalorien ich pro Tag gegessen und getrunken habe ([PinaKaloriedaten.csv](#)). Außerdem habe ich eine smarte Waage, auf der ich mich jeden Morgen unter vergleichbaren Bedingungen gewogen habe ([PinaGewichtsdaten.csv](#)). Man vermutet: Wenn man an einem Tag viel isst, bringt man am nächsten Tag entsprechend mehr Gewicht auf die Waage. Übt man sich dagegen im Verzicht, erwartet man am nächsten Morgen ein geringeres Gewicht. Die Gewichtskurve und die Kalorienkurve sollten dabei also etwa die gleiche Form zeigen.

Die beiden CSV-Dateien sind nicht für den Import in Pandas vorbereitet. Sie entsprechen damit eher dem, was Ihnen im Alltag als Datenquellen begegnen wird. Meist gilt es nach dem Import noch Kleinigkeiten anzupassen. Beispielsweise enthalten die CSVs Datumsangaben im deutschen Format mit Punkt als Trennzeichen. Das erkennt Pandas nicht automatisch als Datentyp `Datetime`. Das Framework bringt aber die Funktion `to_datetime()` mit, der man einen Format-String mitgeben kann:

```
df_weight = read_csv(
    "PinaGewichtsdaten.csv",
    usecols=[1, 2])
df_weight["date"] = to_datetime(
    df_weight["date"],
    format="%Y-%m-%d %H:%M:%S.%f")
df_nutri = read_csv(
    "PinaKaloriedaten.csv",
    usecols=[0, 1, 2, 3],
    dtype={"Tag": str,
           "Noom": float,
           "Lose It!": float,
           "Yazio": float})
df_nutri.rename(index=str,
                columns={"Tag": "date"},
                inplace=True)
df_nutri["date"] = to_datetime(
    df_nutri["date"],
    format="%d.%m.%y")
```

Beim Import von Zahlen ist es außerdem sinnvoll, `read_csv()` mit dem Parameter `dtype=` mitzuteilen, wenn die Funktion auch dann Gleitkommazahlen laden soll, wenn sie in der CSV-Datei ohne Komma stehen. Die Funktion `.rename()` benennt Spalten um, indem man ihr ein Dictionary mit den alten und neuen Namen übergibt.

Ich habe meine Kalorien mit verschiedenen Apps protokolliert, da diese jeweils unterschiedliche Datenbanken mit Nahrungsmitteln benutzen. 170 Gramm Apfel haben dann mal 89 kcal in der einen App und 111 kcal in der anderen App. Eine neue Spalte mit dem Durchschnitt berechnet Pandas mit `.mean()`:

```
df_nutri["Kalorien"] = df_nutri[
    ["Noom", "Loose It!", "Yazio"]
].mean(axis=1, skipna=True)
```

Der Befehl filtert dafür zuerst die richtigen Spalten mit der Angabe in eckigen Klammern heraus. Dabei darf man eine Liste an Spalten angeben, weshalb in der eckigen Klammer eine eckige Klammer steht.

Der Parameter `skipna=True` in der `.mean()`-Funktion sorgt dafür, dass Pandas Zeilen mit undefinierten Werten verwirft. Undefinierte NaN-Werte entstehen beispielsweise, wenn in der CSV-Datei nichts zwischen den Trennzeichen steht.

Statt des Durchschnitts berechnet `.var()` die Varianz der Werte in den drei Spalten. Für die übliche Darstellung als Standardabweichung müssen Sie dabei noch die Wurzel ziehen, wofür Sie problemlos Numpy einspannen können, da Pandas intern ohnehin auf Numpy aufbaut:

```
df_nutri["var"] = np.sqrt(df_nutri[
    ["Noom", "Loose It!", "Yazio"]
].var(axis=1, skipna=True))
```

Die Korrelation zwischen Kalorien und Gewicht erwartet man ja nicht für den gleichen Tag. Nach einem umfangreichen Gelage geht man davon aus, dass die Waage erst am nächsten Tag nach oben ausschlägt. Um das im Diagramm zu sehen, müssen Sie die Kaloriendaten um einen Tag nach vorne verschieben:

```
from datetime import timedelta
df_nutri["date"] = (df_nutri["date"] -
timedelta(days=-1))
```

Außerdem erwartet man, dass die Kalorienkurve nur die Veränderung des Gewichts, nicht den Verlauf von dessen Absolutwerten vorzeichnet. Um die Berechnung der diskreten „Ableitung“ kümmert sich `.diff()`:

```
df_weight["diff"] = df_weight["weight"].diff()
```

An diesem Punkt stehen die Kaloriendaten und die Gewichtsdaten noch getrennt in den beiden DataFrames `df_nutri` und `df_weight`. Die Waage liefert schon länger Daten als die Kalorien-Apps. Deswegen gibt es einige Gewichtsmessungen, zu denen keine Kaloriendaten existieren. Der folgende Befehl verwirft alle Zeilen in der Gewichtstabelle, die älter sind als der älteste Kalorienwert:

```
df_weight = df_weight[
    df_weight["date"] >
    df_nutri["date"][-1]]
```

Die Daten für zwei Liniendiagramme stehen danach in zwei DataFrames. Ein Linien- und ein Balkendiagramm unterscheiden sich in Altair hauptsächlich durch den Aufruf von `.mark_line()` statt `.mark_bar()`. In dieser Funktion bestimmt der Parameter `color=`, in welcher Farbe das Framework die Linie zeichnet:

```
chart_weight = alt.Chart(df_weight
).mark_line(color="blue")
```

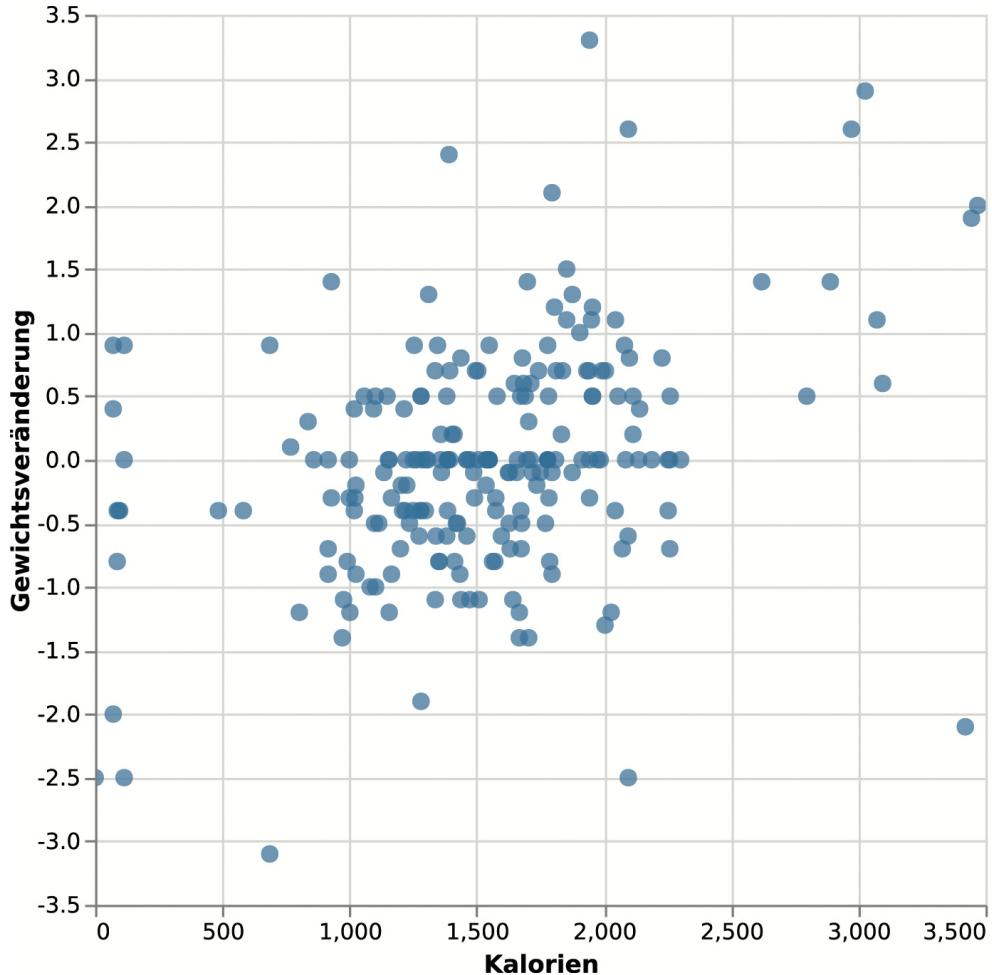


Figure 19.1.: Die Punkte im Streudiagramm ordnen sich nicht in einer Diagonalen an. Das heißt, dass es keine nennenswerte Korrelation zwischen Kalorien und Gewichtsveränderung gibt.

```

    ).encode(x='date',
              y=alt.Y('diff',
title="Gewichtsdifferenz zum Vortag",
scale=alt.Scale(zero=False,
domain=(-3, 6), type='linear'))
)

```

Der Parameter `scale=` legt in der Funktion `.encode()` fest, wie Altair die y-Achse skaliert. Das Scale-Objekt setzt dafür in `domain=` den Wertebereich und legt mit `zero=False` fest, dass die Skala nicht bei 0 beginnt. Mit dem Parameter `type=` könnte man hier beispielsweise auch eine logarithmische Skalierung einstellen.

Die Kalorienkurve sieht ähnlich aus, nutzt aber noch eine etwas aufwendigere Formatierung. Die Dicke der Linie soll nämlich einen grafischen Eindruck vermitteln, wie groß die Standardabweichung der Kalorienwerte ist. Die Dicke der Linie orientiert sich dafür an der über alle Zeilen gemittelten Varianz (`df_nutri["var"].mean()`):

```

chart_nutri = alt.Chart(df_nutri
    .mark_line(color="green",
              interpolate='linear',
              shape='stroke',

```

```

        strokeCap='round',
        strokeJoin='round',
        strokeOpacity=0.5,
        strokeWidth=(
            df_nutri["var"].mean() /
            df_nutri["Kalorien"].max() * 400
        ).encode(x='date',
                  y='Kalorien'
    )
)

```

Um diese beiden Kurven in einem Diagramm zu sehen, reicht es, die Chart-Objekte zu addieren:

```
dbl_chart = chart_nutri + chart_weight
```

Altair versucht dann aber, die y-Achsen beider Diagramme im selben Wertebereich anzuzeigen. Da Kalorien und Gewicht ganz andere Einheiten verwenden, würden die Kurven dann nicht übereinander liegen. Der folgende Befehl schaltet das ab, sodass das Diagramm anschließend zwei y-Achsen enthält: eine links und eine rechts:

```
dbl_chart = dbl_chart.resolve_scale(
    y='independent')
```

Die Angabe von `.properties(width=830, height=400)` sorgt für ein recht breites Diagramm. Da die CSV-Dateien aber enorm viele Daten enthalten, ist es trotzdem schwierig alle Zacken und Spitzen gut zu erkennen. Abhilfe schafft es da, ein interaktives Diagramm zu erstellen, in dem man mit dem Mausrad zoomen und scrollen kann. In Altair hängt man dafür einfach die Funktion `.interactive(bind_x=True, bind_y=False)` an. Die beiden Parameter legen lediglich fest, dass sich die Skalierung der y-Achse beim Scrollen nicht verändert, während die x-Achse einen Zoom an bestimmte Daten erlaubt.

Im Diagramm liegen die Kurven nicht sauber übereinander, was sie bei einer starken Korrelation tun müssten. Den Code und ein fertiges Diagramm zum Scrollen finden Sie über [ct.de/ybcz](#) im Repository auf GitHub in der Datei “Pinas Körperdaten.ipynb”.

19.7. Sind Kalorien und Gewicht überhaupt korreliert?

Ob zwei Größen korrelieren, sehen Sie leicht in einem Streudiagramm. Dafür tragen Sie eine Größe auf der x- und die zweite Größe auf der y-Achse auf und zeichnen Punkte in dieses Koordinatensystem. Bei einer starken Korrelation ordnen sich die Punkte entlang einer Diagonalen an, unkorrelierte Größen produzieren dagegen eine Wolke aus wahllos verteilten Punkten.

Für die Korrelation zwischen Kalorien und Gewichtsdifferenz vereinigen Sie die Werte dafür erst mal in einem gemeinsamen `DataFrame`. Die Zeitangaben passen dafür leider noch nicht zusammen: Während die Kaloriendaten jeweils nur einen Tag benennen, enthalten die Gewichtsmessungen auch die Uhrzeit der Messung. Pandas runden die Zeitangaben mithilfe eines `DatetimeIndex`, der die passende `.round()`-Funktion mitbringt:

```
from pandas import DatetimeIndex
dw = df_weight[['date', 'diff']]
dw['date'] = DatetimeIndex(dw['date']).round(freq='D')
```

Danach stehen in beiden DataFrames auf Tage genaue Zeitangaben in der Spalte `"date"`. Die Funktion `.merge()` fügt diese nun zu einem DataFrame mit einer Tabelle zusammen:

```
dn = df_nutri[['date', 'Kalorien']]
```

```
dm = dn.merge(dw)
```

Merge nutzt hier standardmäßig den Parameter `how="inner"` sodass `.merge()` alle Zeilen aussortiert, die nicht in beiden Tabellen stehen. Setzt man stattdessen `how="outer"`, verwirft Pandas keine Zeilen und es entsteht eine Tabelle mit vielen undefinierten Einträgen (`Nan`). Andere von SQL bekannte Joins wie `"left"` und `"right"` funktionieren ebenfalls.

`.merge()` fügt die Tabellen außerdem standardmäßig so zusammen, dass die Werte in allen gleich benannten Spalten übereinstimmen müssen. Mit dem Parameter `on=` lässt sich das auf bestimmte Spalten eingrenzen. Die ausführliche Zeile `dm = dn.merge(dw, on="date", how="inner")` produziert daher das gleiche Ergebnis wie `.merge(dw)`. Mit diesen Daten zeichnet Altair mit `.mark_circle()` ein Streudiagramm:

```
alt.Chart(dm).mark_circle(size=60
    .encode(x='Kalorien',
            y=alt.Y('diff',
                    title="Gewichtsveränderung"),
            tooltip=['date'])
    .properties(width=400, height=400
    .interactive()
```

Das leere `.interactive()` am Ende sorgt für ein Diagramm, das sich gleichzeitig in x- und y-Richtung zoomen lässt. Der Trick versteckt sich im Parameter `tooltip=['date']` in der Funktion `.encode()`. Durch ihn zeigt das Diagramm in einem kleinen Fenster das Datum zu jedem Punkt, sobald man mit der Maus darüber fährt. Das ist praktisch, um Ausreißer zu identifizieren.

Da sich im Diagramm keine Diagonale ergibt, ist die These widerlegt, dass auf ein Festmahl am nächsten Tag ein höheres Gewicht folgt. Scheinbar fallen andere Effekte wie der Füllstand der Verdauung oder Wassereinlagerungen im Körper stärker ins Gewicht als die Größe der Mahlzeiten.

19.8. Wöchentliche Durchschnitte korreliert?

Sollten Verdauung und Wassereinlagerung das Gewicht im Tagesrhythmus schwanken lassen, wäre das ein hochfrequentes Rauschen, das sich ausgleicht, wenn man Durchschnitte über einen längeren Zeitraum bildet. Wöchentliche Durchschnitte berechnet Pandas glücklicherweise auch im Handumdrehen:

```
from pandas import DateOffset
dm['dg'] = dm['date'] - DateOffset(
    weekday=0, weeks=1)
dmm = dm.groupby(by='dg',
                 as_index=False)[
    ['date', 'Kalorien', 'diff']]
    ].mean()
```

Zuerst erstellt der Code dafür eine neue Spalte mit auf Wochen gerundeten Zeitangaben. Dabei hilft `DateOffset(weekday=0, weeks=1)`, das für jede Zeile die Differenz zur nächsten vollen Woche berechnet. zieht man diese Spalte von der tagesgenauen Zeit ab, bleiben auf Wochen gerundete Zeiten.

Im nächsten Schritt fasst `.groupby()` alle Zeilen zusammen, die zur selben Woche gehören. Für die so gruppierte Tabelle berechnet `.mean()` den Durchschnitt der Kalorien und Gewichtsdifferenz. Im DataFrame dmm steht damit am Ende eine wesentlich kürzere Tabelle mit einer Zeile pro Woche und den berechneten Durchschnitten. Das Diagramm dazu unterscheidet sich lediglich darin, dass es die Punkte im `tooltip=` mit der Wochenangabe beschriftet:

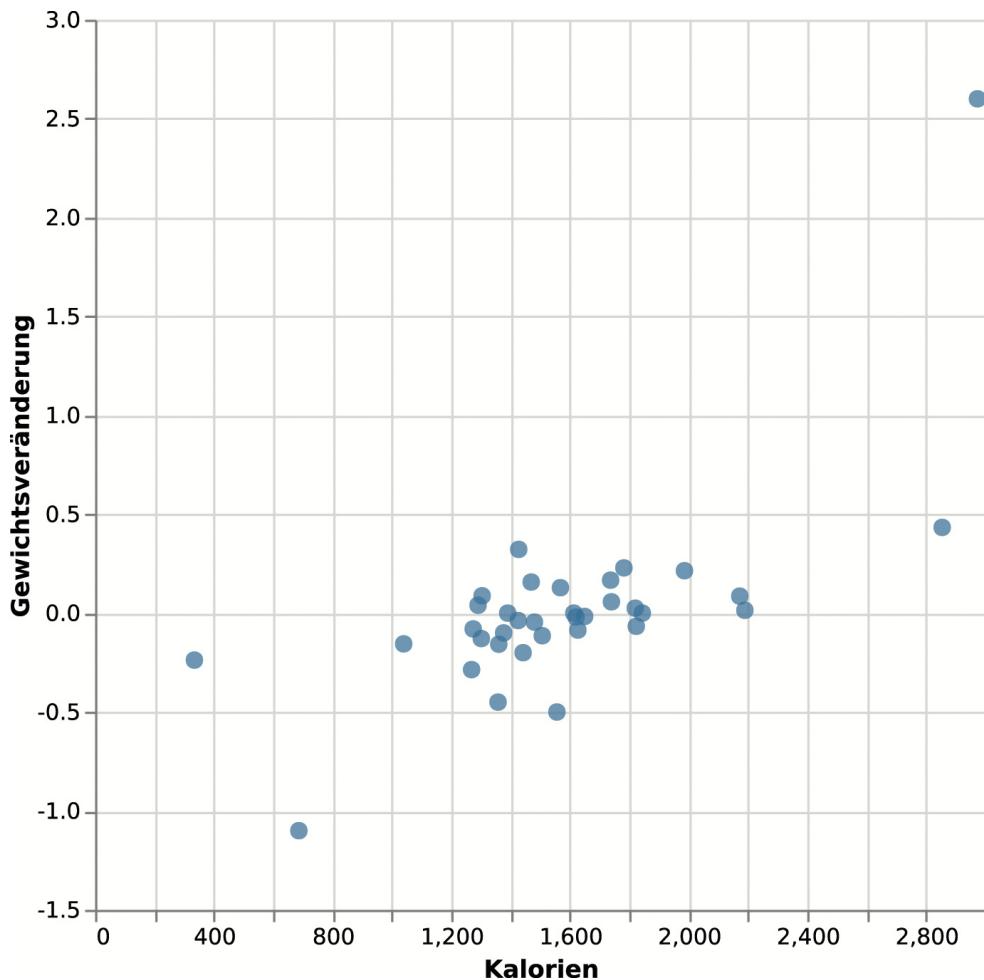


Figure 19.2.: Das Streudiagramm mit wochenweise gemittelten Werten zeigt immer noch keine starke Korrelation zwischen Kalorien und Gewicht. Die Ausreißer sind aber interpretierbar.

```
alt.Chart(dmm).mark_circle(size=60
    ).encode(x='Kalorien',
              y=alt.Y('diff',
                      title="Gewichtsveränderung"),
    tooltip=['dg']
    ).properties(width=400, height=400
    ).interactive()
```

Das entstehende Diagramm zeigt weiter keine Diagonale. Die Punkte streuen aber deutlich weniger, was auf eine geringe Korrelation hindeutet.

Interessant sind vor allem die Ausreißer: Bei dem Punkt links in der Mitte habe ich eine Woche gefastet. Beim Fasten schaltet sich die Verdauung weitgehend ab. Da die Verdauung aber selbst circa 800 Kalorien verbraucht, habe ich durch eine Woche Fasten kaum Gewicht verloren. Der Punkt links unten zeigt eine Woche, in der ich durchschnittlich nur knapp 700 kcal gegessen habe und trotz höherer Kalorienmenge mehr abgenommen habe als in der Fastenwoche.

Der Punkt rechts oben entstand im Skiurlaub. Dort habe ich mich viel bewegt, viel gegessen und dadurch viel Muskelmasse zugelegt. Bei dem Punkt rechts in der Mitte war ich dagegen auf einer Konferenz, bei der ich zwar viel geschlemmt, mich aber nur

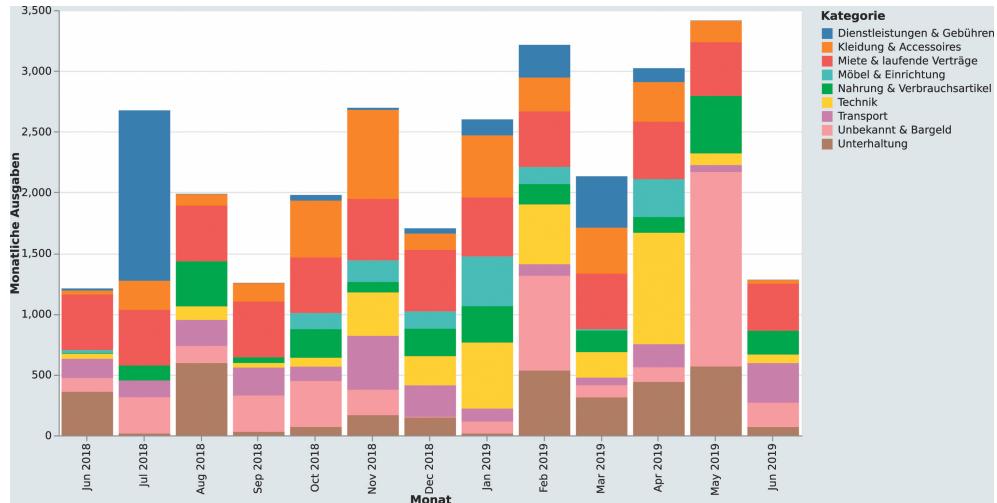


Figure 19.3.: Die Höhe der farbigen Balken zeigt jeweils die Summe der Ausgaben dieser Kategorie.

wenig bewegt habe. Statt Muskeln habe ich dort nur ein halbes Kilo Fett angesetzt.

19.9. Schnelle Statistiken

Pandas und Altair filtern und plotten auch große Datenberge in Windeseile. Da beides kostenlos ist, steht Ihnen die ganze Macht statistischer Auswertung mit nur wenigen Zeilen Code zur Verfügung. Welche Statistiken Sie weiter bringen, müssen Sie allerdings vorher wissen. Die Hauptarbeit besteht daher meist in der Auswahl der richtigen statistischen Methode und vor allem im Sammeln der Daten.

Für die Beispiele in diesem Artikel musste ich beispielsweise über viele Monate 5 bis 10 Minuten pro Tag ins Loggen von Kaloriendaten investieren. Für die gesamte Auswertung mit Pandas und Altair habe ich dagegen nicht länger als zwei Stunden gebraucht und in dieser Zeit musste ich des Öfteren erst in der Doku nachlesen, wie die Funktionen und Plots genau funktionieren. Im Idealfall sammelt ein Gerät die Daten daher automatisch, beispielsweise eine Solaranlage oder eine smarte Heizung. Es lohnt sich, nach solchen Datenquellen zu suchen und sie mal schnell in einem Jupyter-Notebook zu aufschlussreichen Diagrammen aufzuarbeiten.

- Notebooks bei GitHub, Testdaten:ct.de/ybcz

19.10. Wofür gebe ich Geld aus?

Die Höhe des Turms dieser Balken gibt die Gesamtsumme für den Monat wieder. Im Diagramm gut zu erkennen sind die circa 1200 Euro im Juli, die ich für meine Personenstandsänderung bezahlen musste ("Gebühr"). Die "Unterhaltung" und das "Bargeld" im Februar gehören zu einem Skiurlaub. Mit der "unbekannten" Ausgabe im März habe ich meine Debitkarte für eine Dienstreise in die USA befüllt.

19.11. Gewicht folgt Konsum?

Die Kurven für Kalorien und Gewicht liegen nicht übereinander. Das deutet darauf hin, dass die beiden Werte nicht stark korrelieren.

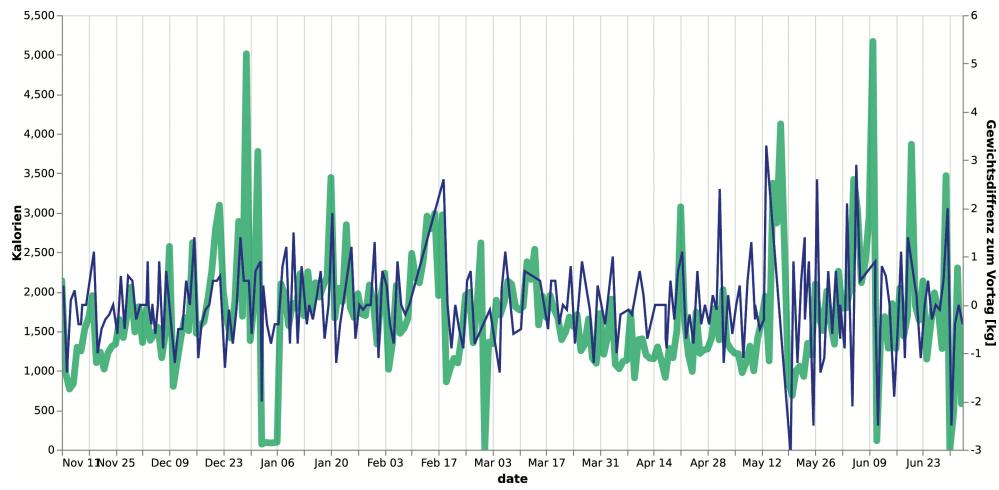


Figure 19.4.: Die Kurven für Kalorien und Gewicht liegen nicht übereinander. Das deutet darauf hin, dass die beiden Werte nicht stark korrelieren.

“Übt man sich im Verzicht, erwartet man am nächsten Morgen ein geringeres Gewicht.”
– Diese Aussage kann das Diagramm mit meinen Daten nicht bestätigen.

19.12. Tasks

- Translation
- Improvements
- Further readings
- Example code with Python
- Presentation

20. Welches Diagramm passt zu meinen Daten?

Ein Diagramm ist schnell gebaut – wenn man erst mal die richtige Darstellungsform gefunden und die Daten passend umgewandelt hat. Unser Leitfaden hilft bei der Orientierung.

Ein Diagramm, in dem man von oben auf Balken schaut, lässt Werte klein erscheinen, von unten wirken sie dagegen groß: Wie man Daten aufbereitet, welchen Diagrammtyp man wählt, sogar die Farben und Formen beeinflussen, wie ein Betrachter die zugrunde liegenden Daten interpretiert. Bevor man eine Grafik baut, muss man sich eingehend mit dem Rohmaterial beschäftigen – und gegebenenfalls auch die Datenquellen kritisch hinterfragen: Welche Erkenntnisse kann man aus den Daten gewinnen? Welche davon sind wichtig und mit welcher Art der Darstellung kann man diese eindrucksvoll vermitteln?

20.1. Diagramm-Typen

Die wohl universellste Darstellungsform sind Säulen und Balken, die sich für die unterschiedlichsten Datentypen eignen. **Säulendiagramme** wirken am besten, wenn man nur eine Handvoll Kategorien oder überschaubare Zeitreihen visualisieren möchte, etwa die jährlichen Verkaufszahlen von Tablets, Smartphones, Notebooks und PCs während der letzten fünf Jahre. Kommen zu viele Kategorien ins Spiel, rücken die Säulen immer enger zusammen. Dann fehlt Platz für die Beschriftung – und eine vertikale Ausrichtung erschwert das Lesen.

Hier schlägt die Stunde des **Balkendiagramms**, das leichter zu beschriften ist, da selbst längerer Text links oder rechts neben den Balken genügend Platz findet. Diese Anordnung unterstützt zudem die natürliche Leserichtung. Setzt man gestalterische Mittel gekonnt ein, wirken auch Balken und Säulen alles andere als langweilig: Man denke etwa an die einprägsame Form der Bevölkerungspyramide.

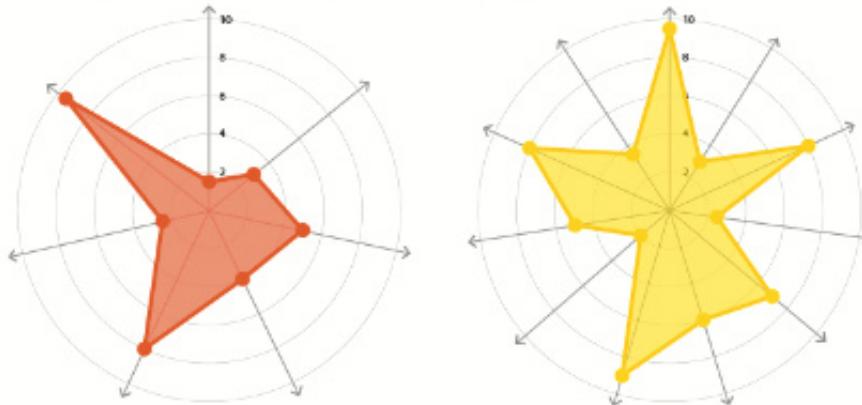
Säulen und Balken gibt es auch in **gestapelte Form**, die häufig dann sinnvoll ist, wenn noch eine weitere Dimension hinzukommt. Jedes Jahr im oben genannten Beispiel lässt sich auch als Stapel anstatt als Säulengruppe darstellen – wodurch die schrumpfenden PC-Anteile und der Trend zu Smartphones und Tablets deutlicher werden.

Kreisdiagramme haben sich etabliert, um den Anteil jeder Kategorie am Ganzen zu zeigen. Je mehr Kategorien ins Spiel kommen und je kleiner die Segmente werden, umso schwieriger wird die Interpretation. Bei mehr als fünf Kategorien ist das Balkendiagramm die bessere Wahl. Donuts wiederum sind leichter zu erfassen als Torten, weil ein Vergleich der außen liegenden Bögen leichter fällt als zwischen den teils schmalen kompletten Kuchenstücken. Zudem lässt sich im Inneren platzsparend die Überschrift oder eine markante Aussage unterbringen.

Für zeitliche Verläufe mit vielen Messwerten gibt es Linien- und Flächendiagramme, wobei die x-Achse üblicherweise als Zeitleiste fungiert.

Ob es einen Zusammenhang zwischen zwei numerischen Variablen gibt, sie also korrelieren, illustriert das **Streudiagramm**, etwa zwischen Alter und Gewicht oder Lebenserwartung und durchschnittlichem Einkommen in den Ländern der Welt. Mit **Blasen** lässt sich eine weitere numerische Größe – beispielsweise die Einwohnerzahl des

Radar Chart



Description

As known as: *Spider Chart*, *Web Chart*, *Polar Chart*, *Star Plots*.

Radar Charts are a way of comparing multiple quantitative variables. This makes them useful for seeing which variables have similar values or if there are any outliers amongst each variable. Radar Charts are also useful for seeing which variables are scoring high or low within a dataset, making them ideal for displaying performance.

Each variable is provided with an axis that starts from the centre. All axes are arranged radially, with equal distances between each other, while maintaining the same scale between all axes. Grid lines that connect from axis-to-axis are often used as a guide. Each variable value is plotted along its individual axis and all the variables in a dataset are connected together to form a polygon.

However, there are some major flaws with Radar Charts:

Having multiple polygons in one Radar Chart makes it hard to read, confusing and too cluttered. Especially if the polygons are filled in, as the top polygon covers all the other polygons underneath it.

Anatomy

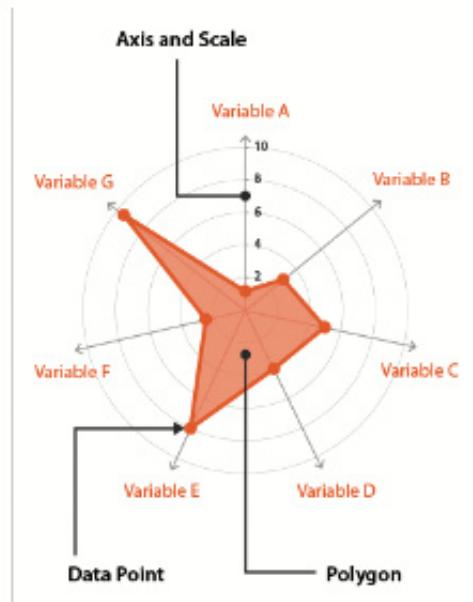


Figure 20.1.: Der Dataviz Catalogue erklärt jeden erdenklichen Diagrammtyp detailliert – auch sortiert nach Anwendungsbereich. (Bild: datavizcatalogue.com)

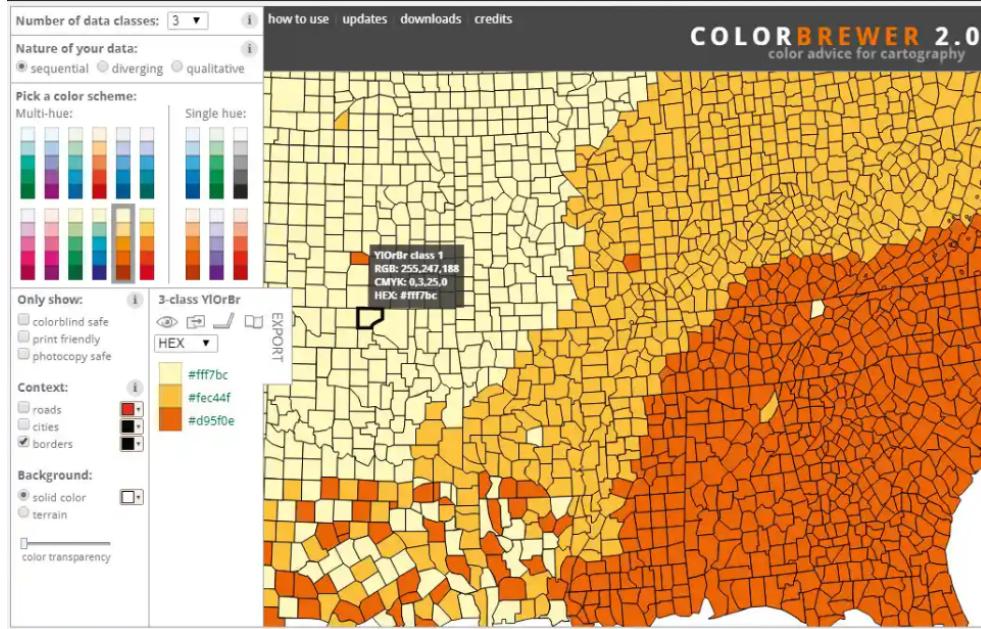


Figure 20.2.: Mit der Analysekarte von ColorBrewer kann man verschiedene Farbschemata durchtesten.

jeweiligen Landes – über die Kreisfläche visualisieren, eine kategorische Unterscheidung (etwa nach Kontinent) kann man über deren Färbung anzeigen.

20.2. Kunst-Graph

Der **Streamgraph** ist der ästhetisch ansprechendere Bruder von gestapelten Säulen- und Flächendiagrammen. Er vermittelt über einen längeren Zeitraum auftretende Trends und Muster eindrücklich. Fürs exakte Ablesen von Werten und zu viele Kategorien eignet sich die Flussdarstellung hingegen nicht.

Hierarchisch geordnete Daten lassen sich als **Baum**, **Sunburst** oder **Baumkarte** visualisieren. Die beiden letzten reflektieren durch ihre Flächen auch Mengenverhältnisse, etwa in Tierpopulationen. Die kompakten, rechteckigen Baumkarten vermitteln die Proportionen besser, während der raumgreifende Sunburst die Hierarchien besser wiederspiegelt.

Alluvial- oder **Sankey-Diagramme** kommen eigentlich aus der technisch-naturwissenschaftlichen Ecke und werden genutzt, um Bewegungen innerhalb eines abgeschlossenen Systems darzustellen, etwa Energie- und Geldflüsse. Je breiter der Fluss, umso größer die Menge, die sich von A nach B bewegt. In der politischen Berichterstattung verdeutlichen sie Wählerwanderungen.

Mit einem sehr eindrucksvollen Sankey-Diagramm hat der Bauingenieur Charles Joseph Minard 1869 die Verluste der französischen Armee während Napoleons Russlandfeldzug verdeutlicht.

Choroplethen- und **Symbolkarten** vermitteln rasch einen Eindruck von den Zuständen auf der Welt, etwa wenn es um die Wahlgewinner in den einzelnen Bundesländern oder die Verfügbarkeit von Breitbandanschlüssen geht. Wichtig ist, die Daten zuvor zu normalisieren und nicht immer wieder gefärbte Karten als Darstellungsform für Daten mit geografischem Bezug zu benutzen: Exakte Größenverhältnisse lassen sich mit Säulen und Balken besser vermitteln – oder einer Kombination aus Karten und Balken.

Einen Überblick sämtlicher Diagrammtypen und ihres Verwendungszwecks gibt der Data Visualization Catalogue. Das Chart-Kompendium lässt sich nach Namen und gewünschter Funktion sortieren.

20.3. Datenformate

Wie man Daten professionell aggregiert, aufbereitet und analysiert, füllt ganze Bücher – das lässt sich in diesem Rahmen nicht erschöpfend erklären. Für erste Experimente und zum Vergleichen der Apps präpariert man am besten eine Tabelle mit wenigen Spalten und Messwerten, die sich zu einem Balken- oder Tortendiagramm verarbeiten lässt. Oder man verwendet einfach die Beispieldatensätze, die nahezu jeder App-Anbieter zur Verfügung stellt.

Der kleinste gemeinsame Nenner beim Datenimport sind kommaseparierte Textdateien (.csv), die meisten Anwendungen akzeptieren auch Excel-Tabellen im Format .xls oder .xlsx oder Google-Tabellen. Einige zapfen Datenbanken an oder analysieren Online-Aktivitäten über das API diverser Dienste wie Twitter und Facebook in Echtzeit. Als Konvention hat sich durchgesetzt, die erste Zeile als Benennung der Variablen zu interpretieren und die Zeilen darunter als Messwerte.

Beim Einlesen kommaseparierter Files sind manche Apps arg wählerisch. Manche erwarten tatsächlich ein Komma und akzeptieren kein Semikolon als Trennzeichen – oder umgekehrt. Das lässt sich per Suchen-Ersetzen leicht beheben (sofern sich diese „reservierten“ Zeichen nicht auch in den Messwerten verstecken). Dann muss man eventuell zu schlau gewählten Suchstrings greifen.

Eine weitere kleine Hürde ist, dass zwei grundsätzlich unterschiedliche Darstellungsformen für Datensätze existieren: das Wide Format und das Long Format. Ersteres ist von Menschen leichter zu verstehen, wird aber von manchen statistischen Verfahren nicht unterstützt. Beim Long Format besitzt jeder Messwert eine eigene Zeile, das Wide Format verteilt Messwerte auf mehrere Spalten.

Beispiel: Bei drei Probanden wird zu drei Zeitpunkten das Gewicht gemessen. Das Wide Format besteht aus den Spalten Proband-Gewicht1-Gewicht2-Gewicht3, das Long Format baut sich nach dem Schema Proband-Zeitpunkt-Gewicht auf. Das erste kommt pro Proband mit einer Zeile aus, bei Letzterem belegt jeder Proband drei Zeilen.

Charticulator, RawGraphs und Tableau erwarten das Long Format, die anderen im Artikel auf Seite 72 gezeigten Apps das Wide Format; die beiden Letztgenannten können das Wide Format praktischerweise selbst konvertieren. Auf dem Desktop gelingt die Transformation mit Statistikprogrammen oder Pivot-Tabellen in einer Tabellenkalkulation.

20.4. JSON und GeoJSON

JSON (Javascript+1, inkjug5eObject Notation) ist ein standardisiertes Format, das vor allem hierarchische Daten besser speichert als CSV – und auch komplette Visualisierungsprojekte wie etwa die von RawGraphs und Highcharts.

Auf geografische Daten ist GeoJSON spezialisiert, mit dem sich Regionen als Polygone beschreiben lassen. Die Open-Source-Software GeoDa hat keine eigenen Karten hinterlegt, weshalb sie GeoJSON als Input benötigt. Datawrapper und Tableau bringen eigenes Kartenmaterial mit, weshalb hier die Angabe von Ländernamen oder Geocodes – DE, AT und so weiter – genügt, um die Regionen anhand der Messwerte einzufärben oder mit Symbolen zu versehen.

Eine sehr reichhaltige Quelle für hiesige Geodaten bis hinunter auf Kreisebene ist das Open Data Lab. Hier lassen sich benötigte Flächen interaktiv auswählen und

Wide Format

Proband	Gewicht1	Gewicht2	Gewicht3
Martina	60.4	58.3	62.0
Michael	80.5	75.9	77.2
Thomas	95.3	92.0	94.1

Long Format

Proband	Zeitpunkt	Gewicht
Martina	1	60.4
Martina	2	58.3
Martina	3	62.0
Michael	1	80.5
Michael	2	75.9
Michael	3	77.2
Thomas	1	95.3
Thomas	2	92.0
Thomas	3	94.1

Figure 20.3.: Im Long Format entspricht jeder Datenpunkt einer Zeile, das Wide Format fällt kompakter aus.

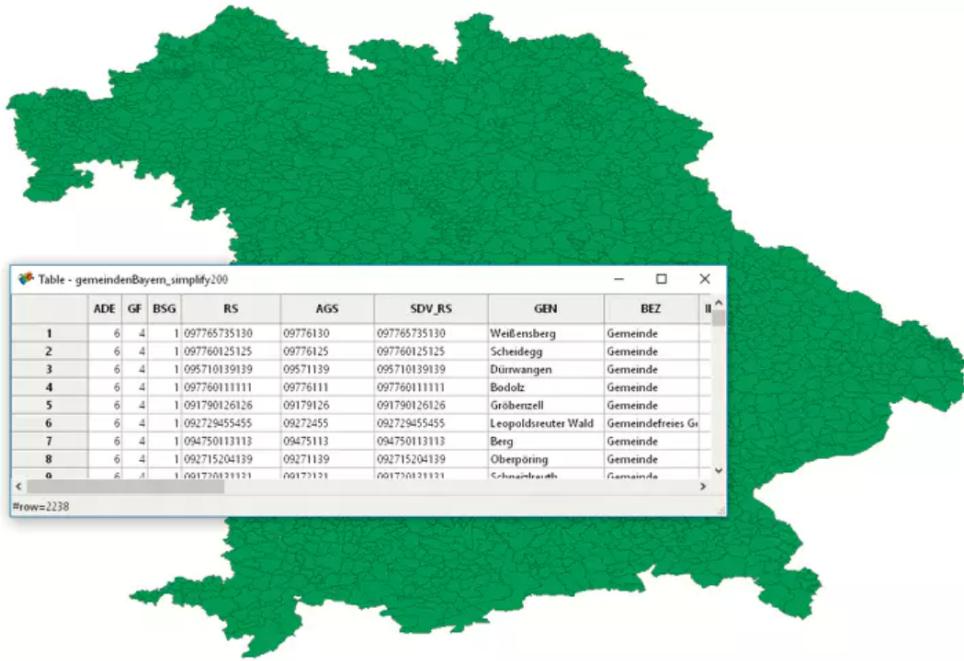


Figure 20.4.: Bei GeoDa muss man das Kartenmaterial importieren. Das kann man sich bei Open Streetmap und Open Data Lab besorgen.

als GeoJSON-Datei herunterladen. Weltweites Kartenmaterial steht auf den OpenStreetMap-Seiten zum Download.

21. Daten verarbeiten mit Python: Pandas-Bibliothek für SQL-Umsteiger

21.1. Einleitung

Mit der Bibliothek Pandas verarbeiten Sie kleine und große Datenmengen mit Python. Wer SQL-Befehle gewöhnt ist, wird sich sehr schnell mit Pandas anfreunden.

Wenn Sie mit der Skriptsprache Python Daten ver- und bearbeiten wollen, bietet sich die Programmierbibliothek Pandas an. Sie ist ein Allesköninger für Python, wenn es um Daten geht. Dabei ist es egal, ob es sich um eine kleine CSV-Datei oder einen sehr großen Datenbestand handelt. Für jede Aufgabe gibt es eine Lösung.

Dieser Artikel soll Ihnen den Einstieg in Pandas vereinfachen. Sie sollten Vorkenntnisse in SQL und Python mitbringen und es gewohnt sein, strukturiert mit Daten zu arbeiten. Am Ende zählt das Ergebnis, nicht die Programmiersprache und deren Tools. Pandas vereinfacht den Umgang mit Daten und kann mit vielen Ein- und Ausgabeketten umgehen.

In diesem Artikel werden Sie mit Daten einer SQL-Tabelle arbeiten – nur halt nicht mit SQL-, sondern mit Python-Befehlen. Sie werden Daten laden, Spalten einschränken, Zeilen auf Werte hin filtern, deren Ausgaben ordnen und sie anschließend als neue Tabelle speichern. Danach werden Sie noch Daten gruppieren und mit diesen Daten rechnen.

Ich möchte an dieser Stelle darauf hinweisen, dass es bei Pandas viele Möglichkeiten gibt, zu einem Ziel zu kommen. Ich werde Ihnen meine Wege vorstellen, wobei die Beispiele einfach gehalten sind und so nicht in einer Produktionsumgebung genutzt werden sollten. Dieser Artikel basiert auf Python 3.7.

21.2. Datenquellen

Sie können viele Quellformate für Pandas nutzen. Dazu zählen statische wie Excel- oder CSV-Dateien, aber auch dynamische wie Webseiten oder Datenbanktabellen.

Für das Beispiel in diesem Artikel nutzen wir die [COVID-19-Fallzahlen der Landkreise vom 28.06.2020](#), bereitgestellt durch das Robert-Koch-Institut. Zu Beginn habe ich die Daten per CSV-Datei ([RKI_Corona_Landkreise.csv](#)) mittels Pandas in eine MariaDB-Tabelle geladen. Den Python-Code dafür finden Sie in der Datei [transform_rki_dataset.py](#), die Tabelle in `create_sql_table.sql`. Diese Dateien und alle weiteren Projektdateien können Sie als [ZIP-Datei](#) von unserem Server herunterladen. Im eigentlichen Programm müssen Sie zuerst eine Verbindung zur MariaDB herstellen. Das geht zum Beispiel mit dem Python-Modul `sqlalchemy`. Zusätzlich müssen Sie das Modul `mysqlclient` installieren. Daher müssen Sie es am Anfang importieren, zusammen mit dem Modul `pandas`:

```
from sqlalchemy import create_engine
import pandas as pd
```

Anschließend setzen Sie die für die Datenbank notwendigen Parameter und erzeugen eine SQL-Engine, die als Verbindung zur Datenbank dient:

OBJECTID		GEN	BEZ	EWZ	KFL	death_rate	\
0	1	Flensburg	Kreisfreie Stadt	89504	56.73	6.250000	
1	2	Kiel	Kreisfreie Stadt	247548	118.65	3.521130	
2	3	Lübeck	Kreisfreie Stadt	217198	214.19	0.591716	
3	4	Neumünster	Kreisfreie Stadt	79487	71.66	2.564100	
4	5	Dithmarschen	Kreis	133210	1428.18	5.555560	
	cases	deaths	cases_per_100k	cases_per_population		BL	\
0	48	3	53.6289	0.053629	Schleswig-Holstein		
1	284	10	114.7250	0.114725	Schleswig-Holstein		
2	169	1	77.8092	0.077809	Schleswig-Holstein		
3	78	2	98.1293	0.098129	Schleswig-Holstein		

Figure 21.1.: Die ersten fünf Datensätze werden angezeigt.

```
sqlEngine = create_engine(f'mysql+mysqldb://{{user}}:{{passwd}}@{{host}}/{{db}}')
```

Beim Arbeiten mit Variablen ist es sehr zeit- und codesparend, **F-Strings** zu nutzen. Setzen Sie etwa vor einen String ein kleines f und schon können Sie innerhalb Ihres Ausdrucks jegliche Variable ohne zusätzliches Konvertieren nutzen; Sie müssen die Variable nur in geschweifte Klammern packen. Das ist sehr nützlich zum Erzeugen von Log-Nachrichten, wenn Zahlen im Text erscheinen müssen: Python nimmt Ihnen hier die Arbeit ab.

Danach verbinden Sie sich mit der Datenbank und lesen die Tabelle als Pandas **DataFrame** ein. Es ist sehr hilfreich, das Verbinden mit dem Konstrukt **with** zu steuern, da Sie sich so nicht mehr um das Schließen einer DB-Verbindung kümmern müssen:

```
with sqlEngine.connect() as dbConnection:
    rki_data = pd.read_sql_table('rki_daten', dbConnection)
```

Dieses Vorgehen bietet sich übrigens auch an, wenn Sie mit anderen Datenquellen oder Dateien arbeiten.

In der zweiten Zeile haben Sie den kompletten Inhalt der Tabelle **rki_daten** eingelesen – und das mit nur einem **pd.read_sql_table**. Das Einlesen anderer Datenquellen ist ähnlich selbsterklärend: **pd.read_csv**, **pd.read_excel** und so weiter. Auf der Projektseite von Pandas [finden Sie eine Übersicht aller möglichen Ein- und Ausgabeformate](#).

21.3. Spalten auswählen

Damit Sie besser einschätzen können, welche Spalten für Sie wichtig sind, können Sie sich mit

```
pd.set_option('display.max_columns', None)
print(rki_data.head())
```

die ersten fünf Zeilen der Daten ansehen. Mit der zuerst gesetzten Option sehen Sie zudem alle Spaltenüberschriften:

Der Befehl

```
print(rki_data.info())
```

wiederum gibt alle Spaltennamen, die Anzahl der non-null-Werte sowie den Typ der darin gespeicherten Informationen aus:

Beim Arbeiten mit **DataFrames** ist es wichtig, unnötigen Ballast zu vermeiden. Vor allem dann, wenn die Datenmenge groß ist. Daher sollten Sie nur die notwendigen

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412 entries, 0 to 411
Data columns (total 15 columns):
OBJECTID                  412 non-null int64
GEN                       412 non-null object
BEZ                       412 non-null object
EWZ                       412 non-null int64
KFL                       400 non-null float64
death_rate                 412 non-null float64
cases                      412 non-null int64
deaths                     412 non-null int64
cases_per_100k              412 non-null float64
cases_per_population        412 non-null float64
BL                         412 non-null object
BL_ID                      412 non-null int64
county                     412 non-null object
last_update                 412 non-null object
cases7_per_100k              412 non-null float64
dtypes: float64(5), int64(5), object(5)
memory usage: 48.4+ KB
None
```

Figure 21.2.: `.info()` zeigt die Spaltennamen, die Anzahl der non-null-Werte und den Datentyp.

Spalten importieren. Verschaffen Sie sich wie zuvor gezeigt einen Überblick über die Daten. Nun entscheiden Sie, welche Informationen Sie laden möchten und geben dem Import-Befehl den Namen der gewählten Spalten mit:

```
relevant_columns = ['OBJECTID', 'GEN', 'BEZ', 'EWZ', 'KFL', 'death_rate',
'cases', 'deaths']
with sqlEngine.connect() as dbConnection:
    rki_data_subset = pd.read_sql_table('rki_daten', dbConnection, columns=relevant_columns)
```

Mit dem Befehl `print(rki_data_subset.info())` sehen Sie, dass nur Ihre Auswahl importiert wurde:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412 entries, 0 to 411
Data columns (total 8 columns):
OBJECTID 412 non-null int64
GEN 412 non-null object
BEZ 412 non-null object
EWZ 412 non-null int64
KFL 400 non-null float64
death_rate 412 non-null float64
cases 412 non-null int64
deaths 412 non-null int64
dtypes: float64(2), int64(4), object(2)
memory usage: 25.8+ KB
None
```

21.4. Index anlegen

Wenn Sie beim Prüfen der eingelesenen Daten auf die Spalte ganz links geachtet haben, werden Sie eine Spalte ohne Namen gesehen haben. Falls nicht definiert, legt Python bei DataFrames automatisch einen numerisch fortlaufenden Index an. Die Daten selbst haben aber schon einen eigenen Index: die Spalte OBJECTID. Um sie beim Importieren als Index zu verwenden, setzen Sie einfach im Import-Befehl den Index neu:

```
with sqlEngine.connect() as dbConnection:
    rki_data_index = pd.read_sql_table('rki_daten', dbConnection, index_col='OBJECTID')
```

Das Resultat können Sie mit `print(rki_data_index.head())` begutachten:

21.5. Datum parsen

Beim Einlesen von Datumsfeldern ist Pandas hinreichend flexibel, um sie automatisch zu erkennen und direkt in ein DateTime64-Objekt umzuwandeln. Bei unserem Beispiel ist das Datumsfeld aber nicht an ein bekanntes Format angelehnt, da zum einen zwischen dem Datum- und dem Zeiteintrag ein Komma steht, zum anderen steht am Ende zusätzlich der Hinweis "Uhr". Daher müssen Sie dieses Format von Hand eintragen, dann wird es auch erkannt:

```
rki_format = '%d.%m.%Y, %H:%M Uhr'
with sqlEngine.connect() as dbConnection:
    rki_data_date = pd.read_sql_table('rki_daten', dbConnection, parse_dates={'last_update': {'format': rki_format}})
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412 entries, 0 to 411
Data columns (total 15 columns):
OBJECTID                  412 non-null int64
GEN                       412 non-null object
BEZ                       412 non-null object
EWZ                       412 non-null int64
KFL                       400 non-null float64
death_rate                412 non-null float64
cases                     412 non-null int64
deaths                    412 non-null int64
cases_per_100k             412 non-null float64
cases_per_population       412 non-null float64
BL                         412 non-null object
BL_ID                     412 non-null int64
county                    412 non-null object
last_update               412 non-null object
cases7_per_100k            412 non-null float64
dtypes: float64(5), int64(5), object(5)
memory usage: 48.4+ KB
None
```

Figure 21.3.: Die Spalte OBJECTID ist nun der Index.

OBJECTID	BL_ID	county	last_update	cases7_per_100k
1	1	SK Flensburg	28.06.2020, 00:00 Uhr	2.234540
2	1	SK Kiel	28.06.2020, 00:00 Uhr	1.211890
3	1	SK Lübeck	28.06.2020, 00:00 Uhr	1.381230
4	1	SK Neumünster	28.06.2020, 00:00 Uhr	0.000000
5	1	LK Dithmarschen	28.06.2020, 00:00 Uhr	0.750694

Figure 21.4.: Das Datum wird nun in einem ordentlichen Format angezeigt.

OBJECTID	GEN	BEZ	EWZ	KFL	death_rate	\
16	17 Braunschweig	Kreisfreie Stadt	248292	192.70	5.35714	
17	18 Salzgitter	Kreisfreie Stadt	104948	224.49	5.96026	
18	19 Wolfsburg	Kreisfreie Stadt	124151	204.61	13.78380	
19	20 Gifhorn	Landkreis	175920	1567.45	2.50000	
20	21 Goslar	Landkreis	137014	966.73	9.57854	

Figure 21.5.: Die Auswahl haben Sie auf Niedersachsen beschränkt.

	BL	BL_ID	county	last_update	\
15	Hamburg	2	SK Hamburg	28.06.2020, 00:00 Uhr	
16	Niedersachsen	3	SK Braunschweig	28.06.2020, 00:00 Uhr	I
17	Niedersachsen	3	SK Salzgitter	28.06.2020, 00:00 Uhr	
18	Niedersachsen	3	SK Wolfsburg	28.06.2020, 00:00 Uhr	
19	Niedersachsen	3	LK Gifhorn	28.06.2020, 00:00 Uhr	
20	Niedersachsen	3	LK Goslar	28.06.2020, 00:00 Uhr	

Figure 21.6.: Mehrere Bundesländer zeigen Sie über eine Liste an, die als Filter verwendet wird.

Mit den gewohnten Befehlen können Sie das Ergebnis auch sofort sehen:

Bei diesem Datensatz ist das Datum nicht so wichtig, aber wenn Sie Zeitreihendaten bearbeiten, werden Zeitstempel sehr oft als Index eingesetzt.

21.6. Einschränkungen mit `where`-Klausel

Da nun alle Daten wie gewünscht in einem `DataFrame` sind, können Sie mit Ihnen arbeiten. Wie bei SQL stehen Ihnen viele Möglichkeiten zur Verfügung.

Universell einsetzbar ist die `where`-Klausel von SQL, die sich genauso gut in Pandas abbilden lässt. Exemplarisch betrachten wir in dem Beispiel alle Datensätze, die das Bundesland Niedersachsen betreffen. Genau wie bei SQL müssen Sie dafür die betreffende Spalte einschränken, in diesem Fall den Wert mit dem String Niedersachsen vergleichen:

```
rki_data_niedersachsen = rki_data[rki_data['BL'] == 'Niedersachsen']
print(rki_data_niedersachsen.head())
```

So haben Sie die Datensätze gefiltert:

Um die Daten mehrerer Bundesländer anzuzeigen, nutzen Sie den Vergleich mit einer Liste. Aber Sie implementieren die Liste als Filter, mit dem Sie anschließend einen neuen DataFrame erzeugen:

```
laender_filter = rki_data['BL'].isin(['Niedersachsen', 'Hamburg', 'Bremen'])
rki_data_laender = rki_data[laender_filter]
print(rki_data_laender.head(100))
```

Das Ergebnis sieht wie folgt aus (Bremen folgt weiter unten):

Abschließend sortieren Sie den letzten `DataFrame` noch absteigend nach der Einwohnerzahl der Städte. Das geht mit der Methode `sort_values`: Sie geben als Parameter die Spalte Einwohnerzahl (`EWZ`) und die Art der Sortierung (absteigend, also `ascending=False`) mit, sowie den Hinweis, dass Zeilen ohne Einträge am Ende der Liste stehen sollen (`na_position='last'`):

OBJECTID		GEN	BEZ	EWZ	KFL	\
15	16	Hamburg	Kreisfreie Stadt	1841179	755.09	
26	27	Region Hannover	Landkreis	1157624	2297.13	
61	62	Bremen	Kreisfreie Stadt	569352	317.83	
57	58	Osnabrück	Landkreis	357343	2121.80	
25	26	Göttingen	Landkreis	328074	1755.39	
52	53	Emsland	Landkreis	325657	2883.64	
29	30	Hildesheim	Landkreis	276594	1208.35	
35	36	Harburg	Landkreis	252776	1248.45	

Figure 21.7.: Die größten Städte kommen am Anfang.

BL	BL_ID	county	last_update	\
15	Hamburg	2	SK Hamburg	28.06.2020, 00:00 Uhr
16	Niedersachsen	3	SK Braunschweig	28.06.2020, 00:00 Uhr
17	Niedersachsen	3	SK Salzgitter	28.06.2020, 00:00 Uhr
18	Niedersachsen	3	SK Wolfsburg	28.06.2020, 00:00 Uhr
19	Niedersachsen	3	LK Gifhorn	28.06.2020, 00:00 Uhr
20	Niedersachsen	3	LK Goslar	28.06.2020, 00:00 Uhr

Figure 21.8.: Pandas erstellt eine Liste aller Landkreise.

```
rki_data_laender_sort = rki_data_laender.sort_values(by=['EWZ'], ascending=False, na_position='last')
print(rki_data_laender_sort.head(100))
```

Und so sieht das Ergebnis aus:

21.7. Distinct Values

Pandas kann Ihnen auch eine Übersicht aller enthaltenen Einträge geben. Sie wird als Liste ausgegeben und kann in einem zweiten Schritt gezählt werden:

```
rki_data_distinct = rki_data['county'].unique()
print(rki_data_distinct)

distinct_values = rki_data['county'].nunique()
print(distinct_values)
```

Hier die Ausgabe der Liste:

21.8. Spaltennamen ändern

Sie können die Spaltennamen ändern, wenn Sie viele Umformungen vornehmen möchten. Das ist vor allem dann wichtig, wenn Sie die Ergebnisse Ihrer Arbeit weitergeben wollen. Dafür gibt es den Befehl rename(), dem Sie ein Dictionary mit alten und neuen Werten übergeben. Achten Sie darauf, am Ende das inplace=True zu setzen, wodurch der existierende DataFrame geändert wird und kein weiterer DataFrame erzeugt werden muss:

```
rki_data_index.rename(columns = 'GEN': 'Name', 'BEZ': 'Bezeichner', 'EWZ': 'Einwohnerzahl', 'KFL': 'Fläche', inplace=True)
print(rki_data_index.head(10))
```

index	Name	Bezeichner	Einwohnerzahl	Fläche	death_rate	\
1	Flensburg	Kreisfreie Stadt	89504	56.73	6.250000	
2	Kiel	Kreisfreie Stadt	247548	118.65	3.521130	
3	Lübeck	Kreisfreie Stadt	217198	214.19	0.591716	
4	Neumünster	Kreisfreie Stadt	79487	71.66	2.564100	
5	Dithmarschen	Kreis	133210	1428.18	5.555560	

Figure 21.9.: Die Ausgabe zeigt die neuen Spaltennamen samt neuen Indexnamen an.

rki_results (15x412)									
Index	Name	Bezeichner	Einwohnerzahl	Fläche	death_rate	cases	deaths	cases_per_100k	cases_per_population
1	Flensburg	Kreisfreie Stadt	89.504	56,73	6,25	48	3	53,6289	0,0536289
2	Kiel	Kreisfreie Stadt	247.548	118,65	3,52113	284	10	114,725	0,114725
3	Lübeck	Kreisfreie Stadt	217.198	214,19	0,591716	169	1	77,8092	0,0778092
4	Neumünster	Kreisfreie Stadt	79.487	71,66	2,5641	78	2	98,1293	0,0981293
5	Dithmarschen	Kreis	133.210	1.428,18	5,55556	72	4	54,05	0,05405
6	Herzogtum Lauenburg	Kreis	197.264	1.263,07	5,97015	268	16	135,859	0,135859
7	Nordfriesland	Kreis	165.507	2.083,53	1.1236	89	1	53,7742	0,0537742
8	Ostholstein	Kreis	200.581	1.393	0	68	0	33,9015	0,0339015

Figure 21.10.: Für eine neue SQL-Tabelle müssen Sie vorher keine leere Tabelle anlegen.

Bleibt noch die Indexspalte, die wie folgt umbenannt wird:

```
rki_data_index.index.names = ['Index']
print(rki_data_index.head())
```

Womit dann alles stimmt:

21.9. Daten speichern

Bis jetzt haben Sie zwar alle Daten bearbeitet, aber noch stehen sie nur im Arbeitsspeicher des Rechners. Pandas bietet eine Fülle von Ausgabeformaten an, von denen wir in diesem Beispiel SQL-Tabellen sowie Excel nutzen.

Um die Daten in einer SQL-Tabelle zu speichern, die es noch gar nicht gibt, führen Sie den folgenden Befehl aus:

```
with sqlEngine.connect() as dbConnection:
    rki_data_index.to_sql('rki_results', dbConnection, if_exists='fail')
```

Dadurch wird die Tabelle automatisch angelegt und mit Daten gefüllt. Sie müssen vorher nicht extra eine leere Tabelle anlegen. Aber erwarten Sie bei komplexen Daten keine Wunder.

Wenn Sie dieselbe Routine etwa am nächsten Tag erneut laufen lassen und neue Werte hinzufügen wollen, ändern Sie einfach das `fail` bei `if_exists` in `append` und schon werden die Daten angehängt.

Zum Speichern in Excel genügt es für den Anfang, der Methode `to_excel` den Dateinamen zu übergeben, sodass sie im selben Verzeichnis erstellt wird:

```
rki_excel = 'rki_excel_results.xlsx'
rki_data_index.to_excel(rki_excel)
```

So sieht die Excel-Datei dann aus:

21.10. Daten gruppieren

Mit Pandas ist das Gruppieren von Daten genauso einfach wie mit SQL. So gruppieren Sie nach Bundesland:

Index	Name	Bezeichner	Einwohnerzahl	Fläche	death_rate	cases	deaths	cases_per_100k
1	Flensburg	Kreisfreie Stadt	89504	56,73	6,25	48	3	53,6289
2	Kiel	Kreisfreie Stadt	247548	118,65	3,52113	284	10	114,725
3	Lübeck	Kreisfreie Stadt	217198	214,19	0,591716	169	1	77,8092
4	Neumünster	Kreisfreie Stadt	79487	71,66	2,5641	78	2	98,1293
5	Dithmarschen	Kreis	133210	1428,18	5,55556	72	4	54,05
6	Herzogtum Lauenburg	Kreis	197264	1263,07	5,97015	268	16	135,859
7	Nordfriesland	Kreis	165507	2083,53	1,1236	89	1	53,7742
8	Ostholstein	Kreis	200581	1393	0	68	0	33,9015
9	Pinneberg	Kreis	314391	664,27	7,6412	602	46	191,481
10	Plön	Kreis	128647	1083,56	6,66667	120	8	93,2785
11	Rendsburg-Eckernförde	Kreis	272775	2189,78	5,4902	255	14	93,4836
12	Schleswig-Flensburg	Kreis	200025	2071,33	2,51572	159	4	79,4901
13	Segeberg	Kreis	276032	1344,47	2,10843	332	7	120,276
14	Steinburg	Kreis	131347	1055,71	1,64835	182	3	138,564
15	Stormarn	Kreis	243196	766,22	7,83848	421	33	173,111

Figure 21.11.: Eine einfache Ausgabe in Excel.

Index	Name	Bezeichner	Einwohnerzahl	\
1	Flensburg	Kreisfreie Stadt	89504	
2	Kiel	Kreisfreie Stadt	247548	
3	Lübeck	Kreisfreie Stadt	217198	
4	Neumünster	Kreisfreie Stadt	79487	
5	Dithmarschen	Kreis	133210	
6	Herzogtum Lauenburg	Kreis	197264	
7	Nordfriesland	Kreis	165507	
8	Ostholstein	Kreis	200581	

Figure 21.12.: Die Landkreise Schleswig-Holsteins stehen nun ganz oben.

```
rki_data_group_01 = rki_data_index.groupby('BL')
print(rki_data_group_01.head(100))
```

Und sehen:

Weiter geht es mit einem Gruppieren nach Bundesländern und darin nach den Namen der Kreise. Wenn Sie die Ausgabe sortiert haben möchten, müssen Sie das Sortieren voranstellen:

```
rki_data_group_02 = rki_data_index.sort_values(['BL', 'Name'], ascending=True).groupby(['BL', 'Name'])
print(rki_data_group_02.head(100))
```

21.11. Mit gruppierten Daten rechnen

Oft werden gruppierte Daten als Grundlage für weitere Berechnungen verwendet. Einfache Berechnungen wie etwa das Aufsummieren können mit simplen Befehlen durchgeführt werden. Hier etwa die Summe der Todesfälle je Bundesland:

```
rki_data_calc_01 = rki_data_index.groupby('BL')['deaths'].sum()
print(rki_data_calc_01.head(16))
```

Sie können auch den Durchschnitt der Todesfälle über alle Landkreise in den jeweiligen Bundesländern berechnen. Das ist in diesem Fall ein wenig komplizierter, da Sie nicht einfach den Durchschnitt nehmen können. So müssen Sie zuerst nach Bundesland und Kreis gruppieren, die Summe der Todesfälle berechnen, den Index neu setzen, nochmal nach Bundesland alleine gruppieren – und dann erst den Durchschnitt berechnen:

	deaths
BL	
Baden-Württemberg	41.568182
Bayern	27.000000
Berlin	17.750000
Brandenburg	9.166667
Bremen	25.000000
Hamburg	259.000000
Hessen	19.461538

Figure 21.13.: Der Durchschnitt der Todesfälle über alle Landkreise in den jeweiligen Bundesländern.

```
rki_data_calc_02 = rki_data_index.groupby(['BL', 'county'])['deaths'].sum().reset_index().groupby(['BL']).mean()
print(rki_data_calc_02.head(100))
```

21.12. Gruppierte Daten einschränken

Nun geht es noch um das Einschränken gruppierter Daten. Dabei erstellen Sie zuerst einen Filter, in dem Sie nach den Bundesländern gruppieren, die Summe ihrer Einwohner berechnen und als Bedingung setzen, dass in den ausgegebenen Bundesländern mindestens 5 Millionen Einwohner leben. Anschließend legen Sie diesen Filter an den Ausgangs-DataFrame an und lassen so nur die Zeilen übrig, die den Filter überstehen:

```
rki_data_having = rki_data_index.groupby('BL')['Einwohnerzahl'].sum()
> 5000000
rki_data_having = rki_data_having.loc[rki_data_having.values==True]
print(rki_data_having.head(50))
```

21.13. Fazit

Pandas sind ein mächtiges Werkzeug. Bedenken Sie, dass fast jedes gezeigte Code-Beispiel vertieft werden kann. Dieser Artikel zeigt nur einen kleinen Teil der Möglichkeiten und soll Lust machen, Pandas selbst auszuprobieren. Im Netz finden Sie viele Tutorials und eine [sehr gute Dokumentation](#), am besten ist aber immer ein eigenes Projekt, bei dem diverse Fallstricke umgangen werden müssen.

Wenn Sie die ersten Schritte gemeistert haben, kommen weitere Themen hinzu, etwa Funktionen oder Performance-Tuning. Und weil Python eine umfassende und leicht zu lernende Sprache ist, können Sie weitere Aspekte schnell in das Auswerten Ihrer Daten einbinden – Sie können Ihre Daten etwa durch aktuelle Informationen aus anderen Quellen anreichern.

```
BL
Baden-Württemberg      True
Bayern                  True
Hessen                  True
Niedersachsen           True
Nordrhein-Westfalen    True
Name: Einwohnerzahl, dtype: bool
```

Figure 21.14.: Die gruppierten Daten können Sie mit Filter weiter einschränken.

22. Analyse von Open Data mit Pandas

22.1. Einleitung

Zahlreiche Organisationen stellen Daten frei zur Verfügung. Die Python-Library Pandas hilft bei der Auswertung.

Neben quelloffener Software befindet sich seit einigen Jahren Open Data auf dem Vormarsch. Insbesondere staatliche Organisationen öffnen ihren Datenschatz immer öfter für die Öffentlichkeit. Auch private Unternehmen stellen immer mehr Informationen unter freien Lizenzen zur Verfügung. Immer mehr frei verfügbare Daten haben der Data Science als Disziplin immensen Auftrieb gegeben.

Programmiersprachen wie Python und R profitieren von der Entwicklung. Bei Ersterer hat sich vor allem Pandas als Tool zur Verwaltung, Analyse und Aufbereitung von Daten durchgesetzt. Dazu gilt es, die passenden freien Daten zu finden, um daraus das Wissen zu extrahieren.

22.2. Immer mehr freie Daten

Die Arbeit an dem Open-Source-Projekt Meteostat, das historische Wetter- und Klimadaten von Tausenden Wetterstationen weltweit zur Verfügung stellt, begannen im Jahr 2015. Damals waren freie Daten insbesondere in Deutschland wenig verbreitet. Der Deutsche Wetterdienst (DWD) stellte nur die Daten einer Hand voll Wetterstationen im Rahmen seiner Grundversorgung frei zur Verfügung.

Obwohl die Arbeit des Wetterdienstes durch Steuergelder finanziert wurde, waren die Aufzeichnungen des deutschen Messnetzes nur gegen eine Gebühr erhältlich, womit sie für nicht kommerzielle Projekte uninteressant waren. Daher mussten andere Datenquellen die Lücke füllen. Der US-amerikanische Wetterdienst NOAA (National Oceanic and Atmospheric Administration) stellte die Meldungen Tausender Flugplätze weltweit kostenfrei zur Verfügung. Diese Daten waren damals die Basis für den Aufbau von Meteostat.

Seit dem Sommer 2017 steht ein Großteil der DWD-Daten frei zur Verfügung und zwar auch für kommerzielle Zwecke. Viele Projekte, Forschungsvorhaben und Unternehmen haben von der Freigabe der Daten profitiert. Unter anderem konnte Meteostat Forschungsprojekte zur Korrelation zwischen bestimmten Wetterbedingungen und COVID19-Infektionen unterstützen.

Open Data beschränkt sich freilich nicht auf Wetter- und Klimadaten. Öffentliche und gemeinnützige Organisationen aus anderen Bereichen stellen ebenfalls Daten unter freien Lizenzen zur Verfügung, darunter das [Datenportal für Deutschland GovData](#) und das offene [Datenportal der Europäischen Union](#). Beide fassen Datenquellen unterschiedlicher öffentlicher Quellen zusammen und eignen sich gut als Recherchewerkzeug.

22.3. Blick auf Pandas

Die Python-Bibliothek Pandas hilft beim Einlesen und Aufbereiten von Daten. Wer Erfahrung mit R hat, fühlt sich bei Pandas zuhause, wer aber einen SQL-Hintergrund hat, sollte sich zunächst mit den Datenstrukturen von Pandas vertraut machen.

Ein DataFrame ist ein ähnliches Konstrukt wie eine Tabelle, die mehrere Spalten und Zeilen enthält. Die Werte einer einzelnen Spalte heißen Series. Pandas stellt unterschiedliche Methoden bereit, um Daten einzulesen, zu verändern oder in einem bestimmten Datenformat zu speichern. Einzelne Spalten lassen sich unkompliziert selektieren und filtern. Unter der Haube basiert Pandas wie viele wissenschaftliche Python-Pakete auf NumPy, einem Tool zur performanten Verarbeitung von Array-Strukturen. Sofern [zusätzlich Matplotlib](#) installiert ist, lassen sich mit Pandas die Daten bequem visualisieren.

Das folgende Beispiel analysiert Zusammenhänge zwischen der monatlichen Maximaltemperatur in Deutschland und dem Interesse an Klimaanlagen in der deutschsprachigen Wikipedia. Mit wenigen Zeilen Code lässt sich herausfinden, ob die Erwartung zutrifft, dass das Interesse in den heißen Sommermonaten besonders groß ist.

Die Installation der erforderlichen Python-Bibliotheken erfolgt über das Paketrepository PyPI:

```
pip install pandas matplotlib meteostat
```

Im Anschluss daran gilt es, die Abhängigkeiten zu importieren:

```
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
from meteostat import Stations, Daily
```

Nun können Entwicklerinnen und Entwickler damit beginnen, Daten einzulesen und in einer Form aufzubereiten, die es erlaubt, daraus Erkenntnisse abzuleiten.

22.4. Beliebige Datenquellen

Die Zugriffszahlen des Wikipedia-Artikels über Klimaanlagen sind auf [der Seite Pageviews Analysis](#) zu finden. Das Einlesen der Seitenaufrufe benötigt mit Pandas lediglich einen Befehl:

```
pageviews = pd.read_csv('pageviews.csv',
index_col='Date',
parse_dates=['Date'])
```

Die Library überzeugt mit simplen Parametern, die das Einlesen lokaler und externer Datenquellen, das Setzen des Index sowie das Parsen von Spalten als datetime erlauben. Bei Bedarf lässt sich das resultierende DataFrame zunächst mit print() in der Konsole als Tabelle darstellen. Das kann beispielsweise hilfreich sein, um zu überprüfen, ob die Daten korrekt gelesen und interpretiert werden:

```
| Date | Klimaanlage | :-----|-----:| 2019-12-01 00:00:00 | 4866 ||
2020-01-01 00:00:00 | 7741 || 2020-02-01 00:00:00 | 5624 || 2020-03-01 00:00:00 | 5662
|| 2020-04-01 00:00:00 | 6998 || 2020-05-01 00:00:00 | 10036 || 2020-06-01 00:00:00
| 11320 || 2020-07-01 00:00:00 | 9507 || 2020-08-01 00:00:00 | 15702 || 2020-09-01
00:00:00 | 5238 || 2020-10-01 00:00:00 | 4772 || 2020-11-01 00:00:00 | 4502 |
```

22.5. Wetterdaten für Deutschland

Einen Überblick über die Temperaturen in Deutschland vermittelt die [Meteostat-Python-Library](#). Ein separater Download der Daten erübrigt sich, da die Library automatisch die erforderlichen Dateien herunterlädt.

Dafür ist zunächst eine Liste von Wetterstationen erforderlich. Da in Deutschland davon über 1000 existieren, bietet es sich an, die Analyse nur mit einer Teilmenge

durchzuführen. In der Statistik heißt dieses Verfahren Sampling. Folgendes Beispiel verwendet eine Stichprobengröße von 50:

```
# Mehrere Threads für schnellere Downloads
Daily.max_threads = 6

# Zeitliche Periode
start = datetime(2019, 12, 1)
end = datetime(2020, 11, 30)

# 50 Zufällige Wetterstationen in Deutschland
stations = Stations()
stations = stations.region('DE')
stations = stations.inventory('daily', (start, end))
stations = stations.fetch(limit=50, sample=True)
```

Der Code filtert die Wetterstationen zunächst nach der Region Deutschland und wählt nur Stationen aus, die im gewünschten Zeitraum Daten gemeldet haben. Die resultierende Liste lässt sich im Anschluss verwenden, um Tageswerte zu laden und nach Monaten zu gruppieren. Im Hintergrund nutzt Meteostat unterschiedliche Pandas-Methoden zur Gruppierung und Aggregation der Daten.

```
# Tageswerte laden
weather = Daily(stations, start, end)

# Daten monatlich aggregieren
weather = weather.aggregate('1MS', spatial=True).fetch()
```

Der Parameter `spatial` legt fest, dass die Meteostat-Bibliothek die Daten zusätzlich räumlich zusammenfassen soll. Die Library gibt für `fetch()` immer einen DataFrame zurück, der sich zum weiteren Verarbeiten der Daten nutzen lässt. Eine Ausgabe der räumlich gemittelten Maximaltemperaturen zeigt, dass nun genau eine Zeile pro Monat im `DataFrame` vorhanden ist:

	time	tmax
1000		
1002	2019-12-01 00:00:00	13.8271
1004	2020-01-01 00:00:00	12.7167
1006	2020-02-01 00:00:00	15.8875
1008	2020-03-01 00:00:00	16.7625
1010	2020-04-01 00:00:00	23.175
1012	2020-05-01 00:00:00	24.5771
	2020-06-01 00:00:00	29.3542
	2020-07-01 00:00:00	30.9776
	2020-08-01 00:00:00	34.3633
	2020-09-01 00:00:00	30.0796
	2020-10-01 00:00:00	19.9265
	2020-11-01 00:00:00	19.6367

Da die Analyse nur eine kleine Teilmenge aller Wetterstationen in Deutschland einbezieht, decken sich die Werte vermutlich nicht mit den offiziellen Gebietsmittelwerten des DWD. Sie sind aber ausreichend, um einen Trend zu erkennen. Dabei gilt es zu beachten, dass es sich bei den Werten um gemittelte und nicht um absolute Maximaltemperaturen handelt.

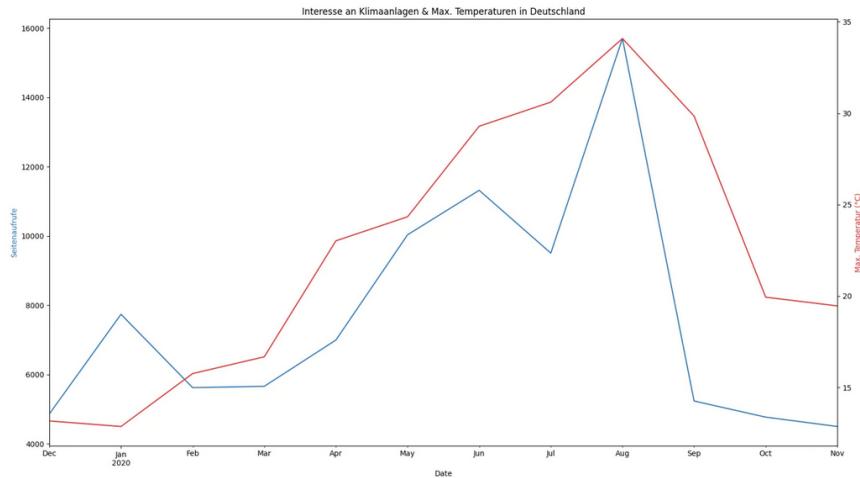


Figure 22.1.: Darstellung der Seitenaufrufe des Artikels über Klimaanlagen bei Wikipedia und der Temperaturmaxima in Deutschland; Datenquelle: Meteostat, pageviews.toolforge.org

22.6. Visualisierung der Daten

Da nun beide Datensätze als DataFrame vorliegen, lassen sich die Werte in einem gemeinsamen Diagramm visualisieren, um den Zusammenhang zu verdeutlichen. Pandas stellt dafür die Methode `plot()` bereit, die im Hintergrund Matplotlib verwendet.

```
# Titel des Diagramms
TITLE ='Interesse an Klimaanlagen & Max. Temperaturen in Deutschland'

# Darstellung der Seitenaufrufe
ax = pageviews['Klimaanlage'].plot(color='tab:blue',
                                      title=TITLE)
ax.set_ylabel('Seitenaufrufe', color='tab:blue')

# Darstellung der Temperaturspitzen
ax2 = ax.twinx()
ax2.set_ylabel('Max. Temperatur (°C)', color='tab:red')
weather['tmax'].plot(ax=ax2, color='tab:red')

# Ausgabe des Diagramms
plt.show()
```

Das Ergebnis veranschaulicht deutlich den Zusammenhang zwischen dem Aufruf der Informationen zu Klimaanlagen und den Temperaturspitzen in Deutschland:

Die Darstellung des zugegebenermaßen vorhersehbaren Ergebnisses zeigt, welche Analysen frei verfügbare Daten ermöglichen. Das Beispiel lässt sich prinzipiell mit einem beliebigen Wikipedia-Artikel reproduzieren. Das Hinzufügen weiterer Datensätze erfordert keinen großen Aufwand.

Das vollständige Skript inklusive der benötigten CSV-Datei ist auf [GitHub](https://github.com) verfügbar.

22.7. Neue Erkenntnisse

Aufbauend dem Beispiel können Entwicklerinnen und Entwickler weniger vorhersehbare Analysen erstellen, um neue Erkenntnisse zu gewinnen. Mit Pandas können sie nach einer kurzen Einarbeitungszeit relevante Ergebnisse erzielen und beispielsweise Muster in Daten visualisieren.

Die Anwendungsfälle reichen von biologischen und medizinischen Fragestellungen über die Optimierung von landwirtschaftlichen Prozessen bis hin zu Predictive Analytics. Dass Daten ein wertvolles Gut sind, steht außer Frage. Es bleibt zu hoffen, dass der Open-Data-Trend auch in Zukunft nicht abreißt und noch viele Unternehmen und Organisationen die Liberalisierung von Daten vorantreiben.

23. Datenqualität mit der Python-Bibliothek Great Expectations sichern

23.1. Einleitung

Great Expectations ist eine umfangreiche Python-Bibliothek, die Data Scientists und Data Engineers bei der Sicherung der Datenqualität unterstützt. Die Software ist komplex, doch die Einstiegshürden sind niedrig. Selbst wer nur einfache Funktionen nutzt, profitiert vom Einsatz in Projekten mit unstrukturierten Daten aus Data Lakes.

- Mit der Python-Bibliothek Great Expectations lassen sich Erwartungen an Datensätze formulieren und validieren.
- Durch ihren deklarativen Ansatz eignet sie sich auch für Domain-Experten ohne umfangreiche Programmierkenntnisse.
- Im JSON-Format gespeicherte Validierungsergebnisse sind universell wiederverwendbar.
- Dank Mechanismen zur Einbindung in Datenpipelines lassen sich Validierungen und Reaktionen auf Fehler automatisieren.

Data-Science- oder KI-Projekte sind nur dann erfolgreich, wenn die Qualität der Daten stimmt (siehe Kasten “Wenn der Data Lake versumpft”). Die kontinuierliche Überwachung der Datenqualität ähnelt ein wenig den Testing-Ansätzen zur Sicherstellung der Codequalität in der klassischen Softwareentwicklung. Das Projekt Great Expectations (GE) orientiert sich an einigen dieser Best Practices aus dem Softwareengineering und adaptiert sie auf den Datenbereich. Great Expectations ist eine quelloffene Python Library mit umfangreichen Möglichkeiten zur Sicherstellung von Datenqualität, zur Dokumentation sowie zum Profiling von Daten. Mithilfe von Great Expectations lässt sich über die gesamte Datenverarbeitungskette ein Mindestmaß von Qualitätsanforderungen systematisch überprüfen und somit der Entstehung von Data Swamps entgegenwirken.

23.2. Wenn der Data Lake versumpft

Viele der auf große Datenmengen ausgerichteten Infrastrukturen (Data Lakes), die in den letzten Jahren entstanden, verdanken ihren Erfolg zu wesentlichen Teilen der schemalosen Integration von Daten aus unterschiedlichsten Quellen. Schemalos bedeutet, dass Daten zunächst unabhängig von ihrer Struktur in Data Lakes geladen werden. Die Interpretation dieser Daten ist Aufgabe der Datenkonsumenten (Schema on Read). Dies ist ein Paradigmenwechsel im Vergleich zu dem bei relationalen Datenbanken und auch im Data Warehousing etablierten Konzept Schema on Write. Gemäß dem Motto „Viel hilft viel“ konnten Daten nun massenhaft und ohne Sicherung einer zuvor festgelegten Struktur in Data Lakes gekippt werden. Oft führte das dazu, dass in kürzester Zeit Unmengen schlecht dokumentierter Datensätze ohne klare Verantwortlichkeit anfielen. Diese Unübersichtlichkeit sowie das Fehlen qualitätssichernder

Maßnahmen ließen Data Lakes mit der Zeit zu Data Swamps verkommen, in denen Datensätze schwer aufzufinden und nicht verlässlich sind.

Mit dem Anspruch vieler Unternehmen, durch datengetriebene Softwareentwicklung Entscheidungen zu automatisieren, wächst aber (wieder) der Bedarf an hochwertigen und verlässlichen Datenquellen, zum Beispiel wenn ein vorab auf den Daten trainiertes ML-Modell zur automatisierten Entscheidungsfindung verwendet wird. Die Datenqualität bestimmt maßgeblich den Erfolg derartiger datengetriebener Entscheidungen („Garbage in, Garbage out“). Neben dem Programmcode gibt es also eine weitere Quelle für unerwartetes oder fehlerhaftes Verhalten einer Software. Das Modelltraining kann wie erwartet funktionieren und das optimale Modell ausgeben – möglicherweise aber auf der Grundlage von Daten, die real existierende Objekte oder Tatsachen nur mangelhaft abbilden. Aus diesem Grund können durch Daten verursachte Fehler nicht mit etablierten Testansätzen aus dem Softwareengineering erkannt werden.

Derartige Abweichungen von der Realität können schon bestehen, wenn das Modell produktiv geht, oder auch erst später auftreten (Concept Drift). Bleiben sie an der Quelle unerkannt, so fallen sie oft erst viel zu spät am Ende einer Reihe vieler Verarbeitungsschritte über verschiedene Teams und Produkte hinweg auf. In einem solchen Worst-Case-Szenario müssen Datenkonsumenten wie Analysten oder Produktteams zunächst die Datenproduzenten aufzufindig machen und sie auf den Fehler hinweisen. Der Entwicklungsprozess aller abhängigen Teams ist dann bis zur Behebung der Ursache stark beeinträchtigt.

Eine nur schwer zu überblickende Kaskade von Fehlern kann sich innerhalb kürzester Zeit über viele Datensätze hinweg propagieren. Sie zu beheben ist oft sehr komplex, zeitaufwendig und selbst wiederum fehleranfällig. Sind gar Kunden und Kundinnen von Fehlentscheidungen betroffen, so lässt sich dadurch entstandener Schaden in der Regel nicht mehr rückgängig machen. Mechanismen zur Sicherstellung von Datenqualität sollten also so früh wie möglich systematisch etabliert werden – möglichst vor der Bereitstellung von Daten, auf jeden Fall aber vor der Produktivierung automatisierter Entscheidungen.

23.3. Schnellstart mit CVS und pandas

Great Expectations (GE) eignet sich primär für tabellarische Datensätze, die als pandas- oder Spark-DataFrame einlesbar sind. Um auch Datenbanksysteme als Datenquelle zu verwenden, kommt der objektrelationale Mapper SQLAlchemy zum Einsatz. Die GE-Dokumentation liefert detaillierte Beispiele, wie sich nicht nur relationale Datenbanksysteme wie PostgreSQL und MySQL einbinden lassen, sondern auch Cloud-Datenbanken wie Redshift oder BigQuery. Auch lassen sich verschiedene Formate auf Cloud-Speichern als Datenquellen konfigurieren, etwa Databricks auf Azure oder pandas-DataFrames in einem S3-Speicher.

Erwartungen (Expectations) sind ein zentrales Element in GE, um Annahmen über den Inhalt oder die Struktur eines Datensatzes auszudrücken. Erwartungen lassen sich in GE entweder deklarativ in JSON-Syntax oder programmatisch per CLI oder Python-API vorgeben und später als JSON-Datei exportieren. Durch die Festlegung auf dieses generische Dateiformat ist man in der Lage, Erwartungen kontinuierlich zu erweitern, sie anzupassen und Änderungen in einem Versionskontrollsystem nachzuhalten.

Der deklarative Ansatz ermöglicht es auch Domänenexperten ohne ausgeprägte Programmierkenntnisse, Erwartungen intuitiv zu formulieren und auf diese Weise zur Sicherstellung von Datenqualität beizutragen. GE ermöglicht zudem eine explorative Herangehensweise bei der Festlegung von Erwartungen, indem man interaktiv mit pandas- oder Spark-DataFrames interagiert. So lassen sich innerhalb eines Notebooks Erwartungen ad hoc definieren, um diese anschließend zu überprüfen und unmittelbar zu verifizieren. Diese Möglichkeit wird im Folgenden vorgestellt. Voraussetzung ist

	date	hour	temp (°C)	region	state	station
0	2017-12-20	14:00	26.5	CO	DF	PARANOA (COOPA-DF)
1	2017-12-20	15:00	26.6	CO	DF	PARANOA (COOPA-DF)
2	2017-12-20	16:00	27.3	CO	DF	PARANOA (COOPA-DF)
3	2017-12-20	17:00	27.5	CO	DF	PARANOA (COOPA-DF)
4	2017-12-20	18:00	27.5	CO	DF	PARANOA (COOPA-DF)
...
995	2017-01-31	01:00	25.4	CO	MS	BELA VISTA
996	2017-01-31	02:00	24.8	CO	MS	BELA VISTA
997	2017-01-31	03:00	24.5	CO	MS	BELA VISTA
998	2017-01-31	04:00	24.9	CO	MS	BELA VISTA
999	2017-01-31	05:00	24.5	CO	MS	BELA VISTA

Figure 23.1.: Der Kaggle-Datensatz im Beispiel zeigt Temperaturwerte aus verschiedenen Orten im Süden Brasiliens.

lediglich ein Jupyter-Notebook.

23.4. Datenerkundung im Notebook

Als Beispiel dient ein Datensatz mit Wetterdaten aus Brasilien, der auf Kaggle verfügbar ist (siehe ix.de/zddp). Er enthält Uhrzeiten, Temperaturmesswerte, Standorte und viele weitere Einträge. Ein Auszug des Datensatzes ist in Abbildung 1 zu sehen. Die CSV-Datei ist sehr groß, für den Anfang genügen aber die ersten 10000 Zeilen. Nach dem Einlesen liefert die Methode `read_csv` ein Objekt vom Typ PandasDataset zurück – einen pandas-DataFrame mit zusätzlichen, GE-spezifischen Funktionen:

```
1000 import great_expectations as ge
1001 ge_df = ge.read_csv("path/to/data.csv", nrows=10000)
1002 type(ge_df)
```

Zusätzlich zu den von pandas bereitgestellten Methoden von `DataFrame` bietet das PandasDataset weitere Funktionen, um Informationen über die Tabelle und die Spalten abzufragen, um Erwartungen zu definieren sowie vorab festgelegte Erwartungen zu validieren.

Einige Statistiken über den Datensatz vermitteln einen ersten Eindruck zum Aufbau (Listing 1). Nach den ersten Untersuchungen des Datensatzes ist es Zeit, die ersten Erwartungen zu definieren. Diese sollten lauten, dass besonders relevante Spalten erstens im Datensatz enthalten sein und zweitens keine fehlenden oder ungültigen Werte enthalten sollen. In der Spalte `temp` sollen fehlende Werte aber erlaubt sein (siehe die ersten vier Zeilen im Listing 2). Die Definition der Erwartungen funktioniert auf dem PandasDataset (und allen anderen GE-Datasets) ohne erneute Zuweisung, GE fügt die Erwartungen direkt der internen Datenstruktur hinzu.

23.5. Wie warm wird es in Brasilien?

Die Spalte `temp` enthält Temperaturmesswerte einer Wetterstation in Brasilien. Zu erwarten ist, dass diese zwischen 10°C und 50°C liegen. Die letzte Zeile im Listing 2 formuliert diese Erwartung. Durch diese Vorgabe ernennt sich der GE-Anwender

Listing 23.1.: Erste Erkundungen des Datensatzes

```
1000 ge_df.info() # pandas function
1001 ge_df.get_column_nonnull_count(col)
1002 ge_df.get_column_quantiles(col, (.0, .25, .5, .75, 1.0))
1003 ge_df.describe() # pandas function
```

Listing 23.2.: Definitionen erster Erwartungen an den Datensatz

```
1000 for col in ["station", "date", "hour", "temp"]:
1001     ge_df.expect_column_to_exist(col) # (1)
1002
1003 for col in ["station", "date", "hour"]:
1004     ge_df.expect_column_values_to_not_be_null(col) # (2)
1005
1006 ge_df.expect_column_values_to_be_between("temp", min_value=10.0,
1007                                         max_value=50.0)
```

gewissermaßen selbst zum Domänenexperten, der seine Kenntnisse als Erwartung an die Daten formuliert. „Selbst in Brasilien, wo es sehr heiß werden kann, erwarte ich keine Temperatur über 50°C.“ Was jedoch, wenn ein Sensor bei einem Waldbrand höhere Werte meldet? Dieses Beispiel veranschaulicht, wie sorgfältig Erwartungen an die Daten formuliert werden sollten.

Die Spalte date enthält Datumswerte als Zeichenketten in der Form yyyy-MM-dd. Ein Wert innerhalb der Spalte hour entspricht der Stunde des Tages und ist gegeben als Zeichenkette im Format hh:mm, wobei die Minutenangabe immer „00“ ist. Diese Erwartungen lassen sich durch Vorgabe des Datumsformats oder mit einem regulären Ausdruck festlegen (Listing 3).

Testweise können die festgelegten Erwartungen auf dem aktuellen Datensatz validiert werden, um zu sehen, ob diese dort erfüllt sind:

```
1000 ge_df.validate()
```

Das Ergebnis zeigen die Listings 4 und 5. Wie erwartet sind die Werte der Spalte `hour` nicht null (`expect_column_values_to_not_be_null` in Listing 2). Die Validierung für diese Expectation auf diesem Datensatz läuft also fehlerlos durch. Die Erwartung, dass der Temperatursensor nur Werte im Bereich von 10 bis 50 Grad liefert, stellt sich allerdings als falsch heraus (Listing 5). Die Validierung findet Werte außerhalb des definierten Intervalls von `GE(-9999.0)`. Der Grund könnte hier beispielsweise ein defekter Sensor sein.

Ähnlich wie bei einem Test wird bei der Validierung ein Istzustand (Datensatz) mit einem vorab festgelegten Sollzustand (Expectation) abgeglichen. Eine Validierung

Listing 23.3.: Erwartungen an die Datumsspalte

```
1000     ge_df.expect_column_values_to_match_strftime_format("date", "%Y-%m-%d")
1001
1002     ge_df.expect_column_values_to_match_regex("hour", "\^([01]?\\d|2[0-3]):00\$")
```

Listing 23.4.: Validierungsergebnisse der Spalte hour

```

1000 {
1002   "success": true,
1003   "exception_info": {
1004     "raised_exception": false,
1005     "exception_message": null,
1006     "exception_traceback": null
1007   },
1008   "result": {
1009     "element_count": 1000,
1010     "unexpected_count": 0,
1011     "unexpected_percent": 0.0,
1012     "unexpected_percent_total": 0.0,
1013     "partial_unexpected_list": []
1014   },
1015   "meta": {},
1016   "expectation_config": {
1017     "expectation_type": "expect_column_values_to_not_be_null",
1018     "kwargs": {
1019       "column": "hour",
1020       "result_format": "BASIC"
1021     },
1022     "meta": {}
1023   }
1024 },

```

kann vor jeder Änderung eines Datensatzes durchgeführt und als Voraussetzung für die Durchführung der Änderung festgelegt werden. Die auf dem obigen DataFrame `ge_df` definierten Erwartungen fasst Great Expectations automatisch zu einer Expectation Suite zusammen. Diese lässt sich im JSON-Format exportieren, um später auf neuen Daten wiederverwendet zu werden. Sie sollte zusätzlich außerhalb des Notebooks, am besten in einem Versionskontrollsysteem gespeichert werden.

```

1000 import json
1001 with open('expectation_suite.json', 'w', encoding='utf-8') as f:
1002     json.dump(ge_df.get_expectation_suite(), f)

```

Listing 6 zeigt einen Auszug der JSON-Datei. Dort finden sich die einzelnen vorher im Code spezifizierten Erwartungen wieder, im Beispiel dargestellt sind die Datumsspalten mit den Expectations `expect_column_to_exist` und `expect_column_values_to_not_be_null`.

Falls noch keine klare Vorstellung zur Erstellung der Expectation Suite vorhanden ist, bietet GE die Möglichkeit, Vorschläge für mögliche Expectations auf Basis der Daten automatisch abzuleiten. Dies ist durch einen Profiler möglich, der statistische Merkmale des Datensatzes auswertet (siehe [ix.de/zddp](#)). Beispielsweise bewegt sich die Temperatur `temp` im obigen Datensatz immer zwischen 10°C und 50°C. Entsprechend schlägt der Profiler eine Expectation `expect_column_values_to_be_between("temp", min_value=10.0, max_value=50.0)` vor. Diese automatisch generierten Expectations sollten allerdings immer auf Sinnhaftigkeit überprüft und gegebenenfalls angepasst werden.

23.6. Was tun bei unerfüllten Erwartungen?

Nachdem eine geeignete Menge von Erwartungen definiert wurde, stellt sich die Frage, wie mit Datensätzen umzugehen ist, die diese nicht erfüllen. Hierbei sind verschiedene

Listing 23.5.: Fehlgeschlagene Validierung der Sensordaten

```

1000 {
1002     "success": false,
1003     "exception_info": {
1004         "raised_exception": false,
1005         "exception_message": null,
1006         "exception_traceback": null
1007     },
1008     "result": {
1009         "element_count": 1000,
1010         "missing_count": 0,
1011         "missing_percent": 0.0,
1012         "unexpected_count": 11,
1013         "unexpected_percent": 1.0999999999999999,
1014         "unexpected_percent_total": 1.0999999999999999,
1015         "unexpected_percent_nonmissing": 1.0999999999999999,
1016         "partial_unexpected_list": [
1017             -9999.0,
1018             -9999.0,
1019             ...
1020             -9999.0
1021         ]
1022     },
1023     "meta": {},
1024     "expectation_config": {
1025         "expectation_type": "expect_column_values_to_be_between",
1026         "kwargs": {
1027             "column": "temp",
1028             "min_value": 10.0,
1029             "max_value": 50.0,
1030             "result_format": "BASIC"
1031         },
1032         "meta": {}
1033     }
1034 }
```

Listing 23.6.: Expectations im JSON-Format

```

1000 ...
1001 "expectations": [
1002 {
1003     "expectation_type": "expect_column_to_exist",
1004     "kwargs": {
1005         "column": "date"
1006     },
1007     "meta": {}
1008 },
1009 ...
1010 ]
```

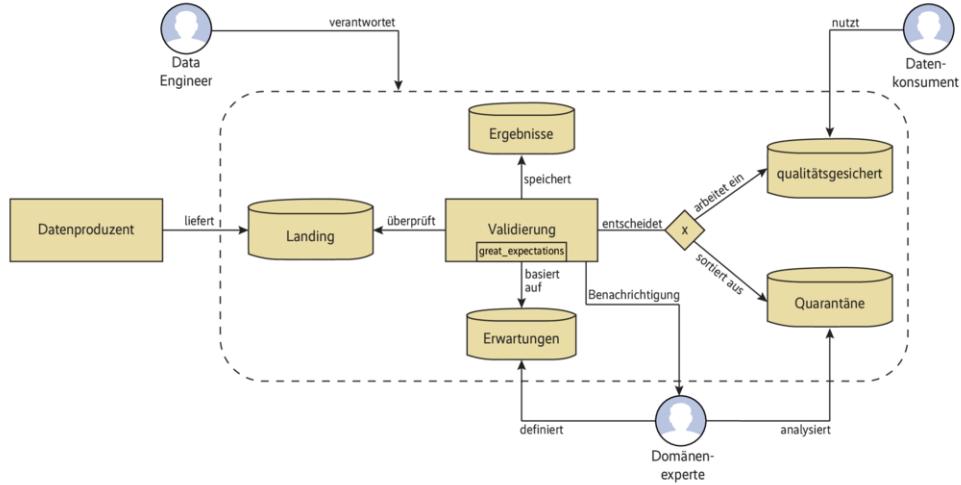


Figure 23.2.: Great Expectations lässt sich als Validierungsinstanz in Datenpipelines einbinden. In diesem Beispiel sind die Verantwortlichkeiten von Data Engineers und Domänenexperten sauber getrennt.

Szenarien denkbar: Bei weniger kritischen Verletzungen könnte es genügen, eine Gruppe von Personen über die Auffälligkeit zu benachrichtigen. Die auffälligen Datensätze werden jedoch weiterverarbeitet. Im obigen Beispiel könnte dies eine erhöhte Temperatur sein, die durch einen Waldbrand ausgelöst wurde. Bei einer schwerwiegenderen Verletzung könnte man zusätzlich zu einer Benachrichtigung die betroffenen Datensätze aussortieren und in einen Quarantänebereich umleiten.

Dies wäre etwa dann der Fall, wenn ein Temperatursensor einen Wert übermittelt, der aus technischen Gründen unmöglich zu messen ist. Great Expectations selbst bietet keine offensichtliche Möglichkeit, eine derartige Strategie für den Umgang mit auffälligen oder fehlerhaften Datensätzen umzusetzen. In einem solchen Szenario muss Great Expectations innerhalb einer Datenpipeline eingebunden sein, die Strategien auf Basis der Validierungsergebnisse von Great Expectations implementiert (Abbildung 2). GE bietet dafür das Konzept der Checkpoints an. Alternativ lassen sich die Validierungsergebnisse auch langfristig in einer Datenbank persistieren, sodass andere Tools sie von dort konsumieren können.

23.7. Fazit

Setzt man einen auf dem Python-Ökosystem basierenden Technologiestack mit pandas oder Spark voraus, bietet sich Great Expectations als einfach zu integrierende Lösung zur Sicherstellung von Datenqualität innerhalb von Datenpipelines an. Nach der Einbindung von GE lassen sich Annahmen auf Basis einer JSON-spezifizierten Suite validieren und sukzessive erweitern. Gerade während einer initialen Projektphase, die von hoher Unsicherheit hinsichtlich Tooling und Anforderungen geprägt ist, lässt sich GE – wie im obigen Beispiel beschrieben – mit minimalem Aufwand einsetzen und schrittweise gemäß den jeweiligen Ansprüchen ausbauen. Die Kernfunktionen von GE sind die Definition und die Validierung von Erwartungen auf Daten, die durch die Spark-Integration auch auf großen Datenmengen praktikabel sind.

Wünschenswert wäre eine Möglichkeit zur Umsetzung komplexerer Strategien für den Umgang mit fehlerhaften Daten, beispielsweise basierend auf vorab festgelegten Schweregraden. Für komplexere Szenarien und die Einbindung in Datenpipelines bietet GE weiterführende Abstraktionen an, die eine tiefere Einarbeitung erfordern,

aber umfangreich dokumentiert sind. Dank JSON-Export sind Validierungsergebnisse universell verwendbar. So lassen sich die Ergebnisse dauerhaft speichern und von weiteren Tools nutzen, etwa für Monitoring- oder Reportingzwecke. (ulw@ix.de)

23.8. Quellen

Dokumentation von Great Expectations und Links zu weiteren Möglichkeiten der Bibliothek: ix.de/zddp

24. Eingefangen – Data Wrangling mit pandas

Data Wrangling, das Aufbereiten von Rohdaten, macht bei Datenanalysen einen Großteil des Aufwands aus. Die Python-Bibliothek pandas bringt dafür effiziente Datenstrukturen und Funktionen mit, die die Arbeit deutlich erleichtern.

Eine besonders beliebte Bibliothek für die Datenanalyse mit Python ist pandas. Sie bietet einen einfachen und dabei sehr effizienten Weg, Daten zu analysieren und zu verwerten. pandas selbst baut auf NumPy auf, einer Bibliothek, die eine hocheffiziente Arithmetik in Python implementiert. So kann NumPy mit Vektoren und Matrizen rechnen und nicht nur mit einzelnen Zahlen. Dies erlaubt eine gewaltige Steigerung der Recheneffizienz. NumPy führt die Berechnungen nicht in Python aus, sondern in C-Code und umgeht so viele Nachteile von reinem Python.

Die Beispiele in diesem Artikel nutzen den Datensatz „Customer Personality Analysis“ von www.kaggle.com. Dank der Lizenz Creative Commons CC0 1.0 darf er beliebig genutzt werden. Es handelt sich um Personendaten einer fiktiven Marketingkampagne.

24.1. Jupyter-Notebook als Arbeitsumgebung

In diesem Beispiel kommt als IDE ein Jupyter-Notebook in Google Colab zum Einsatz. Wer lieber lokal arbeitet, sollte beachten, dass sich die pandas-Versionen teilweise stark voneinander unterscheiden. Dieser Artikel arbeitet mit Python 3.7.12 und pandas 1.1.5. Letzteres besitzt diverse Abhängigkeiten und es gilt, darauf zu achten, dass sie mit installiert werden, was in der Regel automatisch klappt.

Die Beispiele in diesem Artikel nutzen den Datensatz „Customer Personality Analysis“ von kaggle.com. Dank der Lizenz Creative Commons CC0 1.0 darf er beliebig genutzt werden. Es handelt sich um Personendaten einer fiktiven Marketingkampagne.

pandas bietet vielerlei Möglichkeiten, Daten einzulesen. Praktisch jedes Datenformat ist geeignet. Zuerst wird pandas in eine Python-Laufzeitumgebung geladen und typischerweise mit dem Kürzel `pd` versehen:

```
import pandas as pd
```

Jede Art von Datei wird in pandas mit einer Funktion eingelesen, die mit `read_` beginnt, worauf ein formatspezifisches Kürzel folgt. In dieser Analyse geht es um die Datei `marketing_campaign.csv` und somit um die Funktion `read_csv`. Wären die Daten als Excel-Tabelle hinterlegt, wäre die Funktion `read_excel` gefragt. Selbst SQL-Datenbanken kann man problemlos nutzen. Es eignen sich alle SQL-Dialekte, die das Projekt SQLAlchemy abdeckt (MySQL, PostgreSQL et cetera).

Beim Einlesen der Daten wird `read_csv` der Pfad der Daten übergeben. In diesem Fall liegen die Daten im Google Drive. Die Daten sind per Tab getrennt anstatt wie üblich mit Komma. Das berücksichtigt der Parameter `sep = '\t'` (für CSV-Dateien lautet der Standardwert `sep = ','`). Der Datensatz besteht aus 30 Spalten. Standardmäßig werden alle eingelesen. Für eine bessere Übersicht kann man das Einlesen auf bestimmte Werte oder einen Bereich beschränken: Der Parameter

```
usecols = list(range(8))
```

etwa spezifiziert die Liste `[0,1,2,3,4,5,6,7]`. Alternativ kann man direkt eine Liste übergeben: `[1,4]` führt zum Einlesen der zweiten und der fünften Spalte. Das Ergebnis der Funktion `read_csv` wird der Variablen `df` zugewiesen:

```

1 #Importiere die display Funktion.
2 #Nur ausserhalb von Jupyter Notebooks notwendig.
3 from IPython.display import display
4 #Nutze die print Funktion
5 print(df.head())
6 #Nutze die display Funktion
7 display(df.head())
8 #Nutze den Standard Jupyter Lab Output
9 df.head()

```

ID	Year_Birth	Education	...	Kidhome	Teenhome	Dt_Customer
0	5524	1957	Graduation	...	0	04-09-2012
1	2174	1954	Graduation	...	1	08-03-2014
2	4141	1965	Graduation	...	0	21-08-2013
3	6182	1984	Graduation	...	1	10-02-2014
4	5324	1981	PhD	...	1	19-01-2014

[5 rows x 8 columns]

ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014

ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014

Figure 24.1.: Die Funktion `display()` aus dem Paket IPython formatiert die Ausgaben im Unterschied zu `print()` wie in pandas vorgesehen.

```
df = pd.read_csv('/content/drive/MyDrive/marketing_campaign.csv', sep = '^',usecols = list(range(8)))
```

Bei `df` handelt es sich um ein pandas-DataFrame-Objekt, neben pandas Series die zentrale Datenstruktur in pandas. Beide besitzen ihre eigenen, pandas-internen Attribute und Methoden. Die Methode `df.head(n)` zeigt die ersten `n` Zeilen eines DataFrame an, mit der Standardeinstellung `n = 5`. In einem Jupyter-Notebook wie in Google Colab wird die letzte Zeile einer jeden Zelle automatisch ausgegeben. Außerhalb von Jupyter-Notebooks muss eine Anzeigefunktion genutzt werden.

Das Paket IPython enthält die in dieser Analyse durchgehend benutzte Funktion `display()`. Bei der Arbeit mit pandas ist sie `print()` vorzuziehen, da sie die Ausgaben so formatiert wie in pandas gedacht (siehe Abbildung 24.1). In einem Jupyter-Notebook ist IPython automatisch installiert und importiert, anderenfalls muss die Funktion erst importiert werden.

Der Datensatz besteht aus Personendaten. Jede Zeile repräsentiert eine Person. Angegeben sind die ID, das Geburtsjahr, der höchste Bildungsstand, der Familienstand, das Einkommen, ob ein Kind oder Teenager zu Hause lebt und das Datum, an dem die Person Kunde wurde. Zudem hat jede Zeile einen Index: eine ganze Zahl zwischen

```

1 #df.columns ruft die Spaltennamen auf
2 display(df.columns)
3 #Backup der Spaltennamen.
4 #Ohne die .copy() Methode würde die Variable an die gleiche Stelle im
5 #Arbeitsspeicher zeigen und Probleme verursachen.
6 old_names = df.columns.copy()
7 #Definiere die neuen Spaltennamen
8 new_names = ['ID', 'Year_Birth', 'Education', 'Marital_Status',
9              'Income in €', 'Kidhome', 'Teenhome', 'Dt_Customer']
10 #Umbenennen der Spaltennamen
11 df.columns = new_names
12 display(df.head(1))

Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income in €',
       'Kidhome', 'Teenhome', 'Dt_Customer'],
      dtype='object')
   ID  Year_Birth  Education  Marital_Status  Income in €  Kidhome  Teenhome  Dt_Customer
0   5524        1957  Graduation        Single     58138.0       0       0  04-09-2012

```

Figure 24.2.: Vor dem Verändern der Spaltennamen sollten die ursprünglichen Spalten gesichert werden.

0 und der Anzahl der Datenpunkte. Sowohl Spaltennamen als auch Index lassen sich beliebig verändern. Der Index ist im Attribut `df.index` hinterlegt. Ein pandas-Index ist ein eindimensionales Array. Man kann den Index verändern, indem man `df.index` ein neues Array zuweist. Es bietet sich beispielsweise an, die Spalte ID oder das Datum als Index zu nutzen – ein Thema für später.

Analog zum Index sind die Spaltennamen im Attribut `df.columns` hinterlegt, ebenfalls ein pandas-Index. Um die Namen zu ändern, kann `df.columns` ein neues Array mit Namen zugewiesen werden. Die Länge des Arrays muss dabei identisch bleiben. Beispielsweise kann der Name der Spalte Income zu „Income in €“ verändert werden, um die Währung zu verdeutlichen. Beim Verändern der Spaltennamen sollte immer ein Backup der ursprünglichen Namen stattfinden. Dazu wird die Methode `.copy()` genutzt und ihr Output einer Variablen zugewiesen (siehe Abbildung 24.2).

Die Methode `df.shape` gibt die Dimensionalität der Daten aus, ein Tupel aus der Anzahl der Zeilen respektive der Datenpunkte im Datensatz und der Anzahl der Spalten: `display(df.shape)` ergibt (2240, 8), also 2240 Zeilen und 8 Spalten. Wie viel Platz die Daten im Arbeitsspeicher belegen und weitere wichtige Eigenschaften zeigt die Methode `df.info()` an (siehe Abbildung 24.3).

Der Datensatz belegt rund 140 KByte Arbeitsspeicher. Zu jeder Spalte wird auch ihr Datentyp (Dtype) ausgegeben. In der mittleren Spalte steht die Anzahl an nicht fehlenden Werten (Non-Null Count). In der Spalte „Income in €“ fehlen offenbar 24 Werte. Den Umgang mit solchen Lücken behandelt ein späterer Abschnitt. Der Datentyp wird durch drei Kategorien angegeben: `int64`, also eine ganze Zahl mit 64-Bit, `float64`, eine Fließkommazahl mit 64-Bit, und `object`, ein unspezifizierter Datentyp, in der Praxis häufig ein string. Keine einzige Zahlenspalte benötigt offenbar 64 Bit.

Die ID ist beispielsweise eine aus wenigen Ziffern bestehende ganze Zahl. Es wird somit mehr Speicher belegt als nötig. Ähnliches gilt für die Spalte Education. Derzeit sind die Einträge als Strings gespeichert. Bei späteren Datenanalysen kann das zu erheblichen Nachteilen führen.

Nicht immer ist es vermeidbar, Strings zu nutzen, und es gibt diverse fortgeschrittenen Methoden, damit umzugehen. Häufig reicht es aber, sich die Daten genauer anzuschauen. Die Spalte Education kann nur einen von mehreren festgelegten Werten annehmen. Es handelt sich um kategoriale Daten, wobei die Bildungsstände die Kategorien bilden. pandas bietet für solche Daten den Datentyp `categorical` an.

```
1 #Rufe die Kurzübersicht von df auf
2 display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ID               2240 non-null    int64  
 1   Year_Birth       2240 non-null    int64  
 2   Education        2240 non-null    object  
 3   Marital_Status   2240 non-null    object  
 4   Income in €      2216 non-null    float64 
 5   Kidhome          2240 non-null    int64  
 6   Teenhome         2240 non-null    int64  
 7   Dt_Customer      2240 non-null    object  
dtypes: float64(1), int64(4), object(3)
memory usage: 140.1+ KB
None
```

Figure 24.3.: Die Methode df.info() gibt unter anderem den Speicherbedarf des Datensatzes aus.

```

1 #Speichere die originalen Datentypen
2 old_types = df.dtypes
3 #Definiere ein Dictionary mit Spaltennamen und gewollten Datentypen
4 new_types = {'ID':'int16', 'Year_Birth':'int16', 'Education':'category', 'Marital_Status':'category',
5              'Income in €':'float32', 'Kidhome':'int8', 'Teenhome':'int8', 'Dt_Customer':'datetime64'}
6 #Weise den Output einer Variablen zu
7 df2 = df.astype(new_types)
8 #Hat alles geklappt?
9 display(df2.info())

```

#	Column	Non-Null Count	Dtype
0	ID	2240 non-null	int16
1	Year_Birth	2240 non-null	int16
2	Education	2240 non-null	category
3	Marital_Status	2240 non-null	category
4	Income in €	2216 non-null	float32
5	Kidhome	2240 non-null	int8
6	Teenhome	2240 non-null	int8
7	Dt_Customer	2240 non-null	datetime64[ns]

dtypes: category(2), datetime64[ns](1), float32(1), int16(2), int8(2)
memory usage: 44.4 KB
None

Figure 24.4.: Eine Anpassung der Datentypen spart Arbeitsspeicher und erleichtert die Weiterverarbeitung.

Die Spalte `Marital_Status`, also der Familienstand, ist ebenfalls kategorisch.

Bleibt noch das Datum in der Spalte `Dt_Customer`. Für Datumsangaben stellt pandas den Datentyp `datetime64` bereit. In der Regel bietet ein passender Datentyp Vorteile, besonders bei Datumsangaben. Die Datentypen können mit der Methode `.astype()` geändert werden (siehe Abbildung 24.4). Als Argument kann ein Dictionary mit den Spaltennamen als Keys und dem gewünschten Datenformat als Values übergeben werden. Es ist sinnvoll, den Output einem neuen DataFrame zuzuweisen, um potenzielle Folgefehler besser nachvollziehen zu können. Auch die originalen Datentypen sollten in einer Variablen gespeichert werden, um Änderungen einfach rückgängig machen zu können. Die Datentypen der Spalten sind im Attribut `df.dtypes` hinterlegt.

Mit den geänderten Typen belegen die Daten nur noch knapp ein Drittel des bisherigen Arbeitsspeichers. Daneben sind die Datentypen nun auch für diverse algorithmische Nutzungen optimiert. Das `ns` in `datetime64[ns]` steht für eine nanosekundengenaue Datumsangabe. Hier sind auch andere Formatierungen möglich. Die Methode `df.describe()` liefert eine Übersicht verschiedener statistischer Kenngrößen der DataFrames `df` und `df2` (siehe Abbildung 24.5). Für jede numerische Spalte (mit Datentyp `int` oder `float`) wird die Anzahl der nicht fehlenden Werte, der Mittelwert, die Standardabweichung, der minimale Wert, das untere, mittlere und obere Quantil und der maximale Wert angegeben. Aus dem maximalen und dem minimalen Wert ergibt sich die Anzahl der für die Darstellung dieser Zahl nötigen Bits und damit der passende Datentyp für die Spalte. Die Übersichten für `df` und `df2` sind identisch, obwohl die Datentypen unterschiedlich sind. Die Datentypänderung hat also keinen Fehler verursacht. Zudem zeigt sich, dass sich `float16` nicht für die Spalte “Income in €” eignet, da die Zahl 666

666.0 nicht dargestellt werden kann.

Die bisherige Analyse behandelte die Daten als Gesamtheit. Die Änderungen erforderten keine tiefergehende Exploration und Manipulation der Daten. Für die meisten Analysen sind jedoch fortgeschrittenere Techniken nötig. Die elementarste Fähigkeit ist das Extrahieren von Untermengen aus einem DataFrame.

Einzelne Spalten lassen sich auf dreierlei Weise extrahieren. Die einfachste Möglichkeit besteht darin, das Attribut `df2.col_name` zu nutzen. `col_name` bezeichnet die zu extrahierende Spalte. Das Ergebnis ist eine pandas Series und hat somit andere

```

1 print('Das ursprüngliche DataFrame hat folgende Beschreibung:')
2 display(df.describe())
3 print('\nDas neue DataFrame hat folgende Beschreibung:')
4 display(df2.describe())

```

Das ursprüngliche DataFrame hat folgende Beschreibung:

	ID	Year_Birth	Income in €	Kidhome	Teenhome
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250
std	3246.662198	11.984069	25173.076661	0.538398	0.544538
min	0.000000	1893.000000	1730.000000	0.000000	0.000000
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000

Das neue DataFrame hat folgende Beschreibung:

	ID	Year_Birth	Income in €	Kidhome	Teenhome
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000
mean	5592.159821	1968.805804	52247.281250	0.444196	0.506250
std	3246.662198	11.984069	25173.074219	0.538398	0.544538
min	0.000000	1893.000000	1730.000000	0.000000	0.000000
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000

Figure 24.5.: Ein Vorher-nachher-Vergleich zeigt, dass die Anpassung der Datentypen keinen Fehler produziert hat.

```

1 #Zeige alle einzigartigen Werte der Spalte ID an
2 display(df2.ID.unique())
3 #Prüfe ob die IDs alle nur einmal vorkommen
4 print(len(df2.ID.unique()) == df2.shape[0])

array([5524, 2174, 4141, ..., 7270, 8235, 9405], dtype=int16)
True

```

Figure 24.6.: Die Zahl der IDs entspricht der Zeilenzahl, also sind alle IDs einzigartig.

Methoden und Attribute als ein DataFrame – eine häufige Fehlerursache wegen der Möglichkeit einer Verwechslung.

24.2. Daten mit pandas filtern

Die Ausgabe von `display(df2.ID)` besteht aus zwei Spalten: der Indexspalte, die die zugehörige Zeile des Eintrags im originalen DataFrame anzeigt, und einer Spalte mit den dazugehörigen Einträgen. Nun kann man prüfen, ob jede ID auch wirklich nur einmal vorkommt. Mit der Methode `series.unique()` lassen sich alle einzigartigen Werte als Array ausgeben. Die Länge dieses Arrays stimmt mit der Anzahl der Zeilen im DataFrame überein – also sind alle IDs einzigartig (siehe Abbildung 24.6).

Daneben bietet sich eine Übersicht über die Familienstände an. Wie viele Personen und wie viel Prozent sind zum Beispiel verheiratet? Die pandas-Series-Methode `series.value_counts()` kann beide Fragen beantworten. In ihrer Grundeinstellung zählt sie, wie oft ein Wert in einer Spalte vorkommt, also die Frequenz der einzelnen Werte. Mit dem Parameter `normalize = True` kann man anstatt der Frequenz die prozentualen Anteile der einzelnen Kategorien ausgeben (siehe Abbildung 24.6).

Eine pandas `Series` lässt sich immer in ihren Index und ihre Werte zerlegen. `series.index` extrahiert den Index und `series.values` die dazugehörigen Werte jeweils als Array. Ein typischer Anwendungsfall ist die grafische Darstellung der Ergebnisse. Für die prozentualen Anteile der jeweiligen Familienstände als Balkendiagramm werden für die x-Achse der Index und für die y-Achse die Werte benötigt (siehe Abbildung 24.8).

Aber was hat es mit den seltsamen Kategorien Alone, YOLO und Absurd auf sich? Zur Aufklärung können die entsprechenden Zeilen der Daten beitragen. Um darauf zuzugreifen, bietet sich eine boolesche Maskierung des DataFrame an. Eine Maske ist eine pandas Series, die eine Spalte mit einer Bedingung abgleicht und jeder Zeile entweder ein True oder ein False zuweist. Die erste Maske bezieht sich auf die Zeilen, in denen die Familienstände dem Wert Alone entsprechen. Man weist diese Maske der Variablen `mask_alone` zu und kann sie mithilfe der Spalte `df2.Marital_Status` formulieren. Zu erwarten sind drei True-Werte, was sich anhand ihrer Summe mithilfe der Methode `series.sum()` prüfen lässt (True entspricht 1, False entspricht 0, siehe Abbildung 24.9).

Auf dieselbe Weise kann man die Masken für die Familienstände YOLO und Absurd bestimmen. Zuletzt werden alle Masken durch ein logisches OR zu einer großen Maske verbunden und der Methode `df2.loc[row_labels, column_labels]` übergeben. Als Resultat ergeben sich die relevanten Zeilen (siehe Abbildung 24.10). Die Angaben Alone, Absurd und YOLO sind offenbar unsinnig und entsprechend zu markieren. Dafür ist die Methode `.loc[row_labels, column_labels]` hilfreich. Es hat diverse Nachteile, Spalten mit der Attributnotation auszuwählen. Zum einen ist sie sehr grob. Die Spalte lässt sich nicht weiter einschränken und es sind immer alle Werte einzubeziehen. Zum anderen können Spaltennamen wie "Income in €" Leerzeichen enthalten. Dann funktioniert die Attributnotation nicht. Sie sollte daher höchstens dann

```
1 print('Die Frequenzen der Kategorien sind:\n')
2 #Mit value_counts() wird die Häufigkeit der Kategorien angegeben.
3 display(df2.Marital_Status.value_counts())
4 print('\nDie prozentualen Anteile der Kategorien sind:\n')
5 #Mit dem Parameter normalize werden die prozentualen
6 #Anteile der Kategorien angegeben.
7 display(df2.Marital_Status.value_counts(normalize=True))
```

Die Frequenzen der Kategorien sind:

```
Married      864
Together     580
Single       480
Divorced     232
Widow        77
Alone         3
YOLO          2
Absurd        2
Name: Marital_Status, dtype: int64
```

Die prozentualen Anteile der Kategorien sind:

```
Married      0.385714
Together     0.258929
Single       0.214286
Divorced     0.103571
Widow        0.034375
Alone         0.001339
YOLO          0.000893
Absurd        0.000893
Name: Marital_Status, dtype: float64
```

Figure 24.7.: Frequenz und Anteil der Familienstände. Die größte Personengruppe ist die der Verheirateten; ihr Anteil beträgt 38 Prozent.

```
1 #Definiere x- und y-Werte
2 x = df2.Marital_Status.value_counts(normalize=True).index
3 y = df2.Marital_Status.value_counts(normalize=True).values
4 #Erstelle ein Balkendiagramm mit matplotlib
5 import matplotlib.pyplot as plt
6 fig, ax = plt.subplots(figsize = (8,6))
7 ax.bar(x = x, height = y)
```

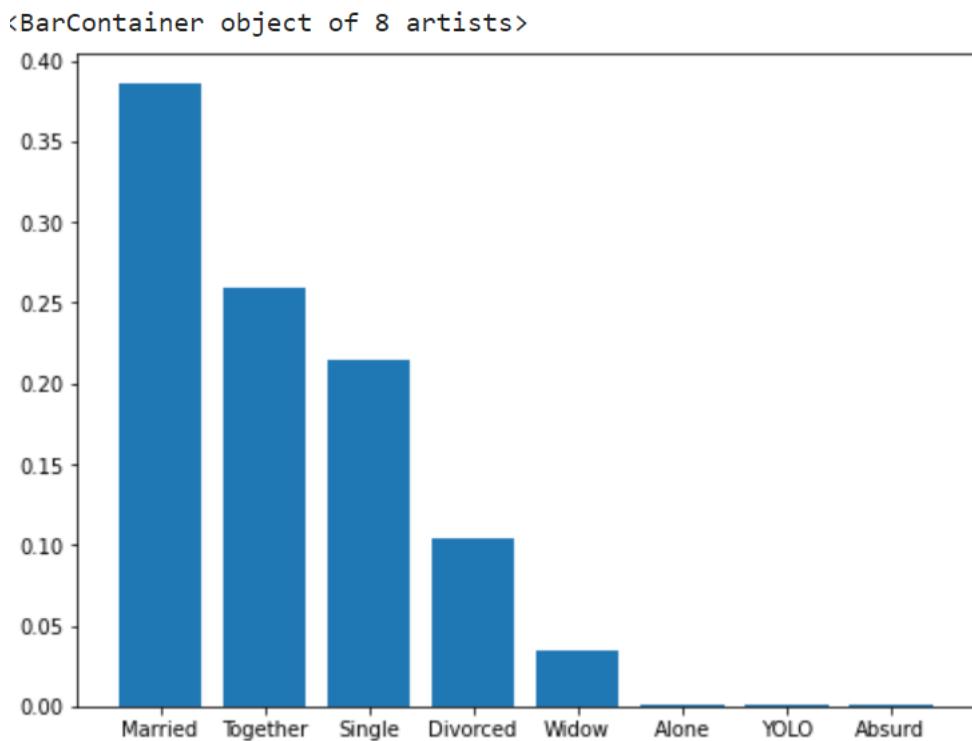


Figure 24.8.: Bei der grafischen Darstellung des Datensatzes fallen drei ungewöhnliche Familienstände auf.

```

1 #Die Maske kann durch eine boolesche Bedingung formuliert werden.
2 #Für jede Zeile wird die Bedingung geprüft und entweder True oder False ausgegeben.
3 mask_alone = df2.Marital_Status == 'Alone'
4 display(mask_alone)
5 print(f'\nDie Anzahl der Zeilen, für die die Maske wahr ist, ist: {mask_alone.sum()}')

0      False
1      False
2      False
3      False
4      False
...
2235    False
2236    False
2237    False
2238    False
2239    False
Name: Marital_Status, Length: 2240, dtype: bool

Die Anzahl der Zeilen, für die die Maske wahr ist, ist: 3

```

Figure 24.9.: Die drei Zeilen mit dem Familienstand Alone lassen sich mit einer passenden Maske ermitteln.

genutzt werden, wenn es mal schnell und einfach gehen muss. Für alle anderen Fälle stellt pandas die Methoden `.loc[row_labels, column_labels]` und `.iloc[row_number, col_number]` zur Verfügung. Beides sind wunderbare Werkzeuge zum Filtern von DataFrames. Sie funktionieren sehr ähnlich, unterscheiden sich jedoch in den Datentypen ihrer Argumente.

Die Methode `.loc` erwartet Label-basierte Argumente, sprich die Namen der benötigten Zeilen und Spalten. In `df` und `df2` ist der Name jeder Zeile durch den Index gegeben, also eine ganze Zahl. Somit kann man dem Argument `row_labels` eine beliebige Liste der Zeilenindizes übergeben, um die jeweiligen Zeilen zu extrahieren. Mit dem Argument `column_labels` wird die Ausgabe auf eine Untermenge aller Spalten beschränkt. Zum Auswählen aller Spalten kann `:` als Argument dienen. Die Methode `.iloc` funktioniert im Prinzip ebenso, nur mit numerischen Argumenten. Es gibt jedoch einen kleinen Unterschied: Die Liste `1:3` entnimmt einem DataFrame die Zeilen 1, 2, und 3 mit der Methode `.loc`, hingegen die Zeilen 1 und 2 mit der Methode `.iloc`. Im zweiten Fall bleibt das letzte Element der Liste also unberücksichtigt – eine so subtile wie häufige Fehlerquelle. Das Argument `col_number` erwartet ebenfalls eine Liste von Zahlen. Um beispielsweise die Spalten `Year_Birth` und `Teenhome` zu extrahieren, muss man die Liste `[1, 6]` nutzen (siehe Abbildung 24.10). Die Zahlen entsprechen dem Index der Spalten im Attribut `df.columns`.

Sowohl `.loc` als auch `.iloc` können ein pandas `DataFrame` oder eine pandas `Series` als Output erzeugen. Der Datentyp hängt davon ab, wie viele Spalten ausgewählt werden. Bei nur einer Spalte ist die Ausgabe eine pandas `Series`, bei mehreren Spalten ein pandas `DataFrame`.

Die Methoden `.loc` und `.iloc` gelten als Best Practice. Sie sind leserlicher, flexibler und erzeugen Ergebnisse fast immer schneller als andere Methoden. Insbesondere `.loc` ist sehr empfehlenswert, weil sich hier Daten mit einer breiten Palette an Maskierungen filtern lassen. Mit der Methode `.iloc` geht dies natürlich auch, aber es erfordert eine aufwendigere Konstruktion der Masken, da diese aus Zahlen bestehen müssen.

24.3. Werte in DataFrames ersetzen

Jetzt sind die Werte YOLO, Alone und Absurd in der Spalte `Marital_Status` durch `NAN` (not a number) zu ersetzen, das für einen fehlenden oder falschen Wert steht.

```

1 #Erzeuge alle Masken
2 mask_alone = df2.Marital_Status == 'Alone'
3 mask_YOLO = df2.Marital_Status == 'YOLO'
4 mask_Absurd = df2.Marital_Status == 'Absurd'
5 #Erzeuge finale Maske. Das logische "or" muss als "|" geschrieben werden
6 mask_full = (mask_alone | mask_YOLO | mask_Absurd)
7 #Filtere die Daten
8 display(df2.loc[mask_full, :])

```

ID	Year_Birth	Education	Marital_Status	Income in €	Kidhome	Teenhome	Dt_Customer
131	433	1958	Master	Alone	61331.0	1	1
138	7660	1973	PhD	Alone	35860.0	1	1
153	92	1988	Graduation	Alone	34176.0	1	0
2093	7734	1993	Graduation	Absurd	79244.0	0	0
2134	4369	1957	Master	Absurd	65487.0	0	0
2177	492	1973	PhD	YOLO	48432.0	0	1
2202	11133	1973	PhD	YOLO	48432.0	0	1

Figure 24.10.: Mit einer Oder-Verknüpfung werden drei Angaben aus dem DataFrame gefiltert, die einer gesonderten Weiterbehandlung bedürfen.

```

1 #Extrahiere die Zeilen 1-3 aus den Spalten Year_Birth und Teenhome.
2 #Beachte den Unterschied in den Zeilenindizes.
3 display(df2.loc[1:3, ['Year_Birth', 'Teenhome']])
4 display(df2.iloc[1:4, [1,6]])

```

	Year_Birth	Teenhome
1	1954	1
2	1965	0
3	1984	0

	Year_Birth	Teenhome
1	1954	1
2	1965	0
3	1984	0

Figure 24.11.: Die Methoden `.loc` und `.iloc` bergen entscheidende Unterschiede hinsichtlich ihrer Argumente.

```

1 #Importiere das Paket Numpy
2 import numpy as np
3 #Ersetze die Kategorien 'YOLO', 'Alone' und 'Absurd'
4 #Erzeuge ein dictionary für die zu ersetzenen Werte
5 d = {'Alone':np.nan, 'YOLO':np.nan, 'Absurd':np.nan}
6 #Überschreibe die gesamte Spalte
7 df2.loc[:, 'Marital_Status'] = df2.loc[:, 'Marital_Status'].replace(d)
8 #Ändere die Werte inplace
9 df2.loc[:, 'Marital_Status'].replace(d, inplace=True)
10 display(df2.info())

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 8 columns):
 # Column Non-Null Count Dtype
--- --
 0 ID 2240 non-null int16
 1 Year_Birth 2240 non-null int16
 2 Education 2240 non-null category
 3 Marital_Status 2233 non-null object
 4 Income in € 2216 non-null float32
 5 Kidhome 2240 non-null int8
 6 Teenhome 2240 non-null int8
 7 Dt_Customer 2240 non-null datetime64[ns]
dtypes: category(1), datetime64[ns](1), float32(1), int16(2), int8(2), object(1)
memory usage: 59.4+ KB

Figure 24.12.: Damit fehlerhafte Kategorien im weiteren Verlauf der Datenanalyse keinen Schaden anrichten, lassen sie sich durch NAN-Werte ersetzen (not a number), hier aus dem Paket NumPy.

Dafür bietet sich die Methode `.replace()` an. Als Argument kann ein Dictionary übergeben werden. Dabei sind die zu ersetzenen Kategorien die Keys und die neuen Werte die Values. Zwei mögliche Wege sind sehr populär. Entweder kann die Spalte `Marital_Status` mit dem Output der Methode `.replace()` überschrieben oder der Parameter `inplace = True` genutzt werden. Damit werden die Änderungen im DataFrame direkt gespeichert und es entsteht kein Output. Als NAN-Wert dient häufig `numpy.nan` aus dem Python-Paket NumPy (siehe Abbildung 24.12).

Nun erkennt pandas automatisch, dass in der Spalte `Marital_Status` sieben Werte fehlen. Doch der Datentyp der Spalte hat sich geändert, weil `np.nan` den Datentyp `float` hat und nun zwei unterschiedliche Datentypen in einer Spalte auftreten. Solche Spalten erkennt pandas immer als `object`. Das kann man korrigieren, indem man die NAN-Werte durch plausible Werte ersetzt oder aus dem DataFrame löscht, etwa mithilfe der Methode `.dropna()`: Standardmäßig entfernt sie alle Zeilen, in denen ein Wert fehlt. Schlimmstenfalls werden so 31 Zeilen gelöscht. Bei einer Gesamtgröße des DataFrame von 2240 Zeilen würden über 98 Prozent der Daten zurückbleiben und deren Repräsentativität dürfte nicht gefährdet sein. Alternativ kann die Methode `.fillna()` die NAN-Werte durch einen bestimmten Wert ersetzen. Bei numerischen Spalten kann dies der Median oder der Mittelwert sein. Bei kategorischen Spalten könnte man etwa den am häufigsten vorkommenden Wert nutzen. Der Output der jeweiligen Methode ist einem neuen DataFrame zuzuweisen. Als Alternative eignet sich auch hier der Parameter `inplace = True` (siehe Abbildung 24.13).

Der Datentyp kann nun ohne Weiteres mithilfe der Methode mit der Methode `.astype()` geändert werden. Zudem kann man mit `.value_counts()` prüfen, ob die unerwünschten Kategorien in der Spalte `Marital_Status` ersetzt wurden.

Außer in kategorischen Spalten sind häufig auch in kontinuierlichen Spalten Werte zu

```

1 #Lösche alle Zeilen mit fehlenden Werten aus dem DataFrame
2 df_dropped = df2.dropna()
3 #Ersetze die fehlenden Werte in der Spalte 'Marital_Status' mit 'Married'
4 df2.loc[:, 'Marital_Status'] = df2.loc[:, 'Marital_Status'].fillna('Married')
5 #Ersetze die fehlenden Werte in der Spalte 'Income in €' mit dem Mittelwert
6 mean = df2.loc[:, 'Income in €'].mean()
7 df2.loc[:, 'Income in €'].fillna(mean, inplace = True)
8 #Prüfen wir die Veränderungen
9 display(df_dropped.info())
10 display(df2.info())

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2209 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   ID                2209 non-null    int16  
 1   Year_Birth        2209 non-null    int16  
 2   Education         2209 non-null    category
 3   Marital_Status    2209 non-null    object  
 4   Income in €       2209 non-null    float32 
 5   Kidhome           2209 non-null    int8   
 6   Teenhome          2209 non-null    int8   
 7   Dt_Customer       2209 non-null    datetime64[ns]
dtypes: category(1), datetime64[ns](1), float32(1), int16(2), int8(2), object(1)
memory usage: 75.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   ID                2240 non-null    int16  
 1   Year_Birth        2240 non-null    int16  
 2   Education         2240 non-null    category
 3   Marital_Status    2240 non-null    object  
 4   Income in €       2240 non-null    float32 
 5   Kidhome           2240 non-null    int8   
 6   Teenhome          2240 non-null    int8   
 7   Dt_Customer       2240 non-null    datetime64[ns]
dtypes: category(1), datetime64[ns](1), float32(1), int16(2), int8(2), object(1)
memory usage: 59.4+ KB
None

```

Figure 24.13.: Sofern es nur einen kleinen Bruchteil aller Daten betrifft, lassen sich fehlerhafte oder unvollständige Angaben ohne spürbare Nebenwirkungen aus einem DataFrame entfernen oder durch Mittelwerte ersetzen.

ersetzen. Die Spalte “Income in €” besteht aus kontinuierlichen Zahlen. Der kleinste Wert ist 1730 €, der größte 666.666 €. Wenn über das Einkommen aber bekannt ist, dass es zwischen 25.000 € und 120.000 € liegt, ist ein Teil der Daten offenbar fehlerhaft. Die fehlerhaften Werte müssen als `NAN` markiert werden, damit im Laufe der weiteren Bearbeitung keine schwerwiegenden Fehler auftreten. Die Methode `.replace` eignet sich hier nicht: Da es sich um kontinuierliche Zahlen handelt, ist es unmöglich, fehlerhafte Kategorien zu erfassen.

24.4. Mit DataFrames rechnen

Eine mögliche Lösung – und typischer Anfängerfehler – besteht darin, mit einer for-Schleife durch die Zeilen zu iterieren und mit zwei if-Statements die fehlerhaft erscheinenden Werte zu ersetzen. Dann kommt jedoch keine pandas-interne Methode zum Einsatz, sondern Python-Code und damit entsteht ein Bottleneck. Die pandas-interne Methode `.mask()` schafft hier Abhilfe. In Analogie zur Methode `.loc` wird ihr eine boolesche Maskierung übergeben. Dann werden alle Zeilen, für die die Maskierung `True` lautet, durch `NAN` ersetzt. Wenn `.mask()` auf einen DataFrame angewandt wird, füllt sich die gesamte Zeile mit `NANs`. Deswegen sollte man immer nur eine einzelne Spalte verändern (siehe Abbildung 24.14). Der Einsatz von pandas kann die Codeausführung etwa um den Faktor 150 beschleunigen.

Es ist häufig notwendig, neue Spalten zu erzeugen (auch Feature Engineering genannt), etwa mit der Methode `.loc`. Als Spaltenname dient dann einfach ein bisher nicht existierender Name, der dem Ergebnis einer Rechnung zugewiesen wird. Es muss sich aber um eine pandas Series oder ein Array handeln, mit einer Länge, die der Zeilenanzahl des DataFrame entspricht. Mit `:` als Zeilenparameter ist das leicht kontrollierbar.

Mit DataFrames und einzelnen Spalten kann man auch Arithmetik betreiben. Um beispielsweise eine Spalte namens `Non_adults_home` zu erzeugen, lassen sich die Spalten `Kidhome` und `Teenhome` mit einem gewöhnlichen `+` addieren. Bei Datumsangaben wie in der Spalte `Dt_Customer` ist oft eine Spalte mit den Monatsangaben für jede Zeile sinnvoll. Hier ergibt sich der Vorteil des Datentyps `datetime`. Mit dem Attribut `.month` aus dem Modul `datetime` werden die Monate aus der Spalte `Dt_Customer` extrahiert. Es entsteht eine Spalte mit den Monaten als Zahl zwischen 1 und 12 entsprechend den Kalendermonaten. Daneben wird die Periodizität der Monate in einer neuen Spalte namens `Month_Sinus` codiert. Dies ist besonders wichtig für Machine-Learning-Anwendungen. Dann muss der Algorithmus die Information “1 ist sehr nah an 12” umsetzen. Dafür eignet sich die Methode `.apply()`, mit der man eine Funktion auf jedes Element anwenden kann.

Als periodische Funktion dient die Sinusfunktion `numpy.sin`. Hier findet sich auch die NumPy-Darstellung der Kreiszahl Pi in Gestalt von `numpy.pi`. Für eine schnelle Umsetzung bietet es sich an, anonyme Funktionen mit dem lambda-Mechanismus von Python zu nutzen (siehe Abbildung 24.15).

Selbstverständlich sind auch komplexere Berechnungen möglich, etwa Multiplikation, Division und Exponentiation eines gesamten DataFrame oder einer Series.

In vielen Fällen ist es sinnvoll, den Index eines DataFrame zu ändern: Die Methode `.set_index()` kann ihn durch ein beliebiges NumPy-Array aus Werten ersetzen, solange dessen Länge mit der Zeilenanzahl übereinstimmt. Der Parameter `inplace = True` unterdrückt den Output.

Den Index zu verändern kann viele Vorteile mit sich bringen. So ist es bei Zeitreihen sinnvoll, das Datum als Index zu nutzen. Dann sind Zeitintervalle einfach zu filtern – eine gängige Technik in der Zeitreihenanalyse. Die Spalte `Dt_Customer` wird als Index festgelegt. Den Index kann die Methode `.sort_index()` sortieren, der optionale Parameter `ascending` legt fest, ob aufsteigend (`ascending = True`) oder absteigend (`ascending = False`). Anhand des Datentyps `datetime` erkennt pandas automatisch

```

1 #Erzeuge eine Kopie von df2, um beide Methoden zu vergleichen.
2 df3 = df2.copy()
3 #Definiere Masken.
4 mask_max_income = df2.loc[:, 'Income in €'] > 120000
5 mask_min_income = df2.loc[:, 'Income in €'] < 25000
6 mask_final_income = mask_max_income | mask_min_income
7 #Importiere das time Modul, um Zeiten messen zu können.
8 import time
9 #Erzeuge ersten Zeitstempel.
10 t1 = time.time()
11 #Iteriere mit einer for-Schleife durch den DataFrame.
12 for i in range(df2.shape[0]):
13     if df2.loc[i, 'Income in €'] < 25000 or df2.loc[i, 'Income in €'] > 120000:
14         df2.loc[i, 'Income in €'] = np.nan
15 #Erzeuge zweiten Zeitstempel.
16 t2 = time.time()
17 #Löse dasselbe Problem mit der .mask() Methode.
18 df3.loc[:, 'Income in €'].mask(mask_final_income, inplace = True)
19 #Erzeuge dritten Zeitstempel.
20 t3 = time.time()
21 #Vergleiche Resultate
22 display(df2.info())
23 display(df3.info())
24 #Vergleiche Zeitdifferenzen.
25 print(t2 - t1)
26 print(t3 - t2)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  --  
 0   ID                2240 non-null    int16  
 1   Year_Birth        2240 non-null    int16  
 2   Education         2240 non-null    category
 3   Marital_Status    2240 non-null    object  
 4   Income in €       1990 non-null    float32 
 5   Kidhome           2240 non-null    int8   
 6   Teenhome          2240 non-null    int8   
 7   Dt_Customer       2240 non-null    datetime64[ns]
dtypes: category(1), datetime64[ns](1), float32(1), int16(2), int8(2), object(1)
memory usage: 59.4+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  --  
 0   ID                2240 non-null    int16  
 1   Year_Birth        2240 non-null    int16  
 2   Education         2240 non-null    category
 3   Marital_Status    2240 non-null    object  
 4   Income in €       1990 non-null    float32 
 5   Kidhome           2240 non-null    int8   
 6   Teenhome          2240 non-null    int8   
 7   Dt_Customer       2240 non-null    datetime64[ns]
dtypes: category(1), datetime64[ns](1), float32(1), int16(2), int8(2), object(1)
memory usage: 59.4+ KB
None
0.13439416885375977
0.0009653568267822266

```

Figure 24.14.: Beim Suchen und Ersetzen fehlerhafter Daten vervielfacht sich das Tempo, wenn pandas-interne Methoden zum Einsatz kommen und kein Python-Code.

```

1 #Importiere datetime
2 import datetime as dt
3 #Erzeuge eine Spalte als Summe zweier Spalten
4 df3.loc[:, 'Non_adults_home'] = df3.loc[:, 'Kidhome'] + df3.loc[:, 'Teenhome']
5 #Erzeuge eine Spalte unter Ausnutzung des datetime Datentyps
6 df3.loc[:, 'Month'] = df3.loc[:, 'Dt_Customer'].dt.month
7 #Erzeuge eine Spalte durch Anwendung einer Funktion
8 df3.loc[:, 'Month_Sinus'] = df3.loc[:, 'Month'].apply(lambda x: np.sin(2*np.pi*(x-1)/11))
9 #Betrachte die Ergebnisse
10 display(df3.head())

```

	ID	Year_Birth	Education	Marital_Status	Income in €	Kidhome	Teenhome	Dt_Customer	Non_adults_home	Month	Month_Sinus
0	5524	1957	Graduation	Single	58138.0	0	0	2012-04-09	0	4	0.989821
1	2174	1954	Graduation	Single	46344.0	1	1	2014-08-03	2	8	-0.755750
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-21	0	8	-0.755750
3	6182	1984	Graduation	Together	26646.0	1	0	2014-10-02	1	10	-0.909632
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-19	1	1	0.000000

Figure 24.15.: Auch das Errechnen neuer Spalten auf der Basis bestehender Werte (Feature Engineering) ist mit pandas schnell erledigt.

die zugrunde liegende Chronologie. Auch hierfür eignet sich der Parameter `inplace = True`.

24.5. Fazit

Die ersten Schritte des Data Wrangling mit pandas sind damit absolviert: das Einlesen der Daten, das Ändern von Spaltennamen und wie man sich einen Überblick über die Struktur der Daten verschafft. Es gibt mehrere Methoden, die Daten zu filtern, einzelne Werte zu ersetzen und mit fehlenden Werten umzugehen. Hinzu kommt das Feature Engineering mit pandas: das Erstellen neuer Spalten mittels Arithmetik, Extraktion und Funktionsanwendung. Es lohnt sich, dranzubleiben und sich tiefergehende Techniken anzueignen. Der nächste Schritt zur Datenanalyse könnte in der visuellen Exploration mit pandas bestehen.

25. Programmieren mit Python: Bedienoberfläche via PyQt erstellen

Wir zeigen Ihnen, wie Sie mit PyQt ein grafisches Login-Programm erstellen. Dabei lernen Sie die Grundlagen des beliebten GUI-Toolkits für Python kennen.

Buttons, Checkboxen, Dropdown-Menüs – was normale Nutzer lieben, kann zu einem Albtraum für den Entwickler werden. Schließlich muss er seinen gut funktionierenden Code mit Zeilen für grafische Elemente überfrachten. Das kann eine weitere Fehlerquelle sein. Trotzdem: Normalerweise muss eine anständige grafische Benutzeroberfläche (graphical user interface, GUI) her, damit jeder Nutzer das jeweilige Programm effizient bedienen kann.

Wir zeigen Ihnen, wie Sie ein simples Login-Fenster mit dem GUI-Toolkit PyQt erstellen. Das Fenster besteht aus einem Hinweistext, zwei Labels für Passwort und Benutzername, zwei Eingabefeldern und einem Button. Der Nutzer soll seine Daten für heise online eingeben und nach dem Buttonklick eine Rückmeldung erhalten, ob der Login funktioniert hat. Außerdem soll es ein kleines Menü geben, über das der Nutzer das Programm beendet.

25.1. PyQt gegen Pyside und Tkinter

PyQt bindet das Open-Source-Toolkit Qt in Python ein und wird von Riverbank Computing Limited entwickelt. Qt wird schon seit 1991 entwickelt und steht als Binding für viele verschiedene Sprachen zur Verfügung, etwa für Ruby, Java oder C#. Für PyQt finden Sie im Netz eine [umfangreiche Dokumentation](#) zu allen Elementen. Neben PyQt gibt es noch andere Möglichkeiten, ein GUI für ein Python-Skript zu erstellen. In Python eingebaut ist das Toolkit Tkinter, das wir bereits in einem anderen Artikel beleuchtet haben. Mit Tkinter können Entwickler mit wenig Aufwand relativ altbackene Bedienoberflächen erstellen. Dagegen ist PyQt eine moderne Alternative. Neben PyQt gibt es zudem noch PySide, das ebenfalls Qt einbindet. Dieses Binding wird von der Qt Company unterstützt. Der größte Unterschied zwischen PyQt und PySide besteht in der Lizenz. PyQt nutzt unter anderem die GNU GPL (General Public License) während PySide unter der offeneren GNU Lesser General Public License (LGPL) steht. Wer kommerzielle Projekte entwickelt, sollte sich daher mit den Feinheiten dieser Modelle auseinandersetzen.

25.2. Hello World mit PyQt

Anders als Tkinter ist PyQt nicht in Python enthalten. Sie müssen daher die Bibliothek installieren und einbinden. Laden Sie die Bibliothek mit der Paketverwaltung Pip herunter:

```
pip install PyQt5
```

PyQt5 ist die aktuelle Version des PyQt-Projekts. Im Netz finden Sie viele Anleitungen, die sich auf das veraltete PyQt4 beziehen – ignorieren Sie diese. Neue und sinnvolle PyQt5-Funktionen würden Ihnen sonst entgehen.

Bei PyQt hat es sich eingebürgert, nur die Elemente einzubinden, die Sie auch wirklich brauchen. Statt `import PyQt5` zu verwenden, holen Sie sich die benötigten Elemente mit einem Befehl `from`. Das wichtigste Element ist die `QApplication`. Dieser Befehl verfrachtet Sie ins Programm:

```
from PyQt5.QtWidgets import QApplication
```

25.3. QApplication

Mit der `QApplication` legen Sie die wichtigsten Eigenschaften eines Fensters fest. Das Programm beginnt damit, dass Sie die `QApplication` starten und es wird geschlossen, wenn Sie die `QApplication` beenden. `QApplication` hält währenddessen eine Schleife aufrecht, in der alle Ereignisse verarbeitet werden, die im Fenster passieren, den Mainloop. Kurz gesagt: Ohne `QApplication` gibts kein Programm.

Daher legen Sie als nächstes die `QApplication` fest:

```
programm = QApplication(sys.argv)
```

Unser Programm heißt kreativerweise `programm`. Der Befehl `sys.argv` ist wichtig, falls Ihr Programm Argumente über die Kommandozeile verarbeiten soll: `sys.argv` ist quasi eine Liste, die den Dateinamen des Projekts enthält sowie alle Argumente, die über die Kommandozeile kommen.

Für `sys.argv` müssen Sie aber vorher noch die Bibliothek `sys` importieren, die systemspezifische Funktionen bereithält:

```
import sys
```

Sind Ihnen die Kommandozeile und die Argumente, die darüber kommen, egal, dann können Sie `QApplication` einfach eine leere Liste mit `[]` übergeben:

```
programm = QApplication([])
```

25.4. Hauptfenster erstellen

Nun erstellen Sie das eigentliche Fenster. Dafür ist `QWidget` verantwortlich, das Sie wie `QApplication` erst importieren müssen:

```
from PyQt5.QtWidgets import QWidget
```

Anschließend erstellen Sie das Fenster ähnlich kreativ wie das Hauptprogramm:

```
fenster = QWidget()
```

`QWidget` hat kein Elternelement, also kein Argument wie `parent=`. Daher wird es als Waise sofort zum Hauptfenster befördert. Generell funktioniert die Eltern-Kind-Beziehung in PyQt so: Zerstören Sie ein Elternelement, löschen Sie gleichzeitig alle zugehörigen Kinder. Außerdem sollte jedes Element ein Elternelement haben, außer es handelt sich um ein Hauptfenster.

25.5. Fenster aufrufen, Schleife starten

Nur noch zwei zusätzliche Zeilen, und schon sehen Sie das erste Fenster. Zuerst müssen Sie das Fenster auf den Bildschirm bringen. Dafür sorgt `show()`:

```
fenster.show()
```

Nun starten Sie die Hauptschleife des Programms mit einem `programm.exec()`. In der Hauptschleife finden alle Ereignisse einer GUI statt, etwa Eingaben des Nutzers oder Klicks. Sie wird so lange ausgeführt, bis jemand das Programm beendet.

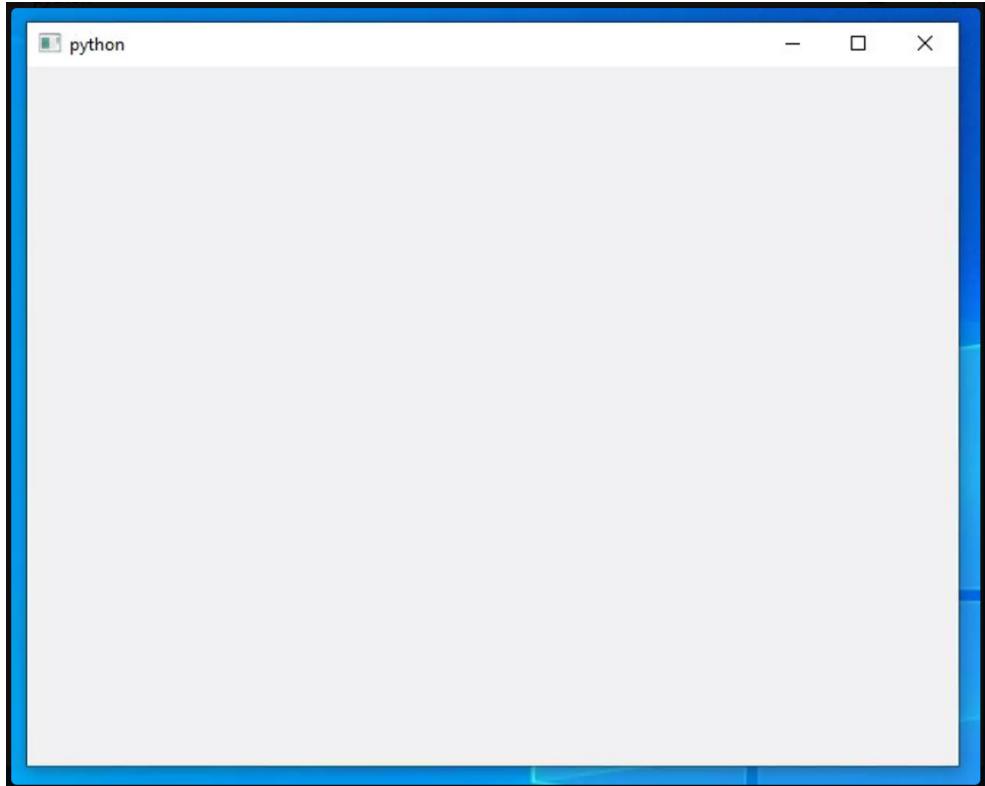


Figure 25.1.: Das erste Fenster ist noch grau und leer.

`programm.exec()` würde alleine schon reichen, um die unendliche Schleife zu starten. Allerdings hat es sich eingebürgert, die Hauptschleife in ein `sys.exit()` zu packen:

```
sys.exit(programm.exec())
```

Damit gibt `sys.exit` den Exitcode des Programms an das Elternelement weiter, wenn Sie das Programm beenden – etwa an den Explorer oder die Kommandozeile, womit Sie das Programm gestartet haben. Ein Exitcode, der nicht Null ist, weist immer auf einen Fehler hin und kann dementsprechend interpretiert werden.

In älteren Programmbeispielen werden Sie noch auf einen zusätzlichen Unterstrich stoßen: `exec_()`. Das liegt daran, dass `exec` ein von Python reserviertes Keyword war. Jedenfalls bis Python 3 kam, dort ist es rausgeflogen. Wenn Sie mit PyQt5 und Python 3 arbeiten, brauchen Sie den Unterstrich nicht mehr und können `exec()` verwenden.

Und damit haben Sie Ihr erstes Fenster mit nur sechs Zeilen Code erstellt:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget

programm = QApplication(sys.argv)

fenster = QWidget()
fenster.show()

sys.exit(programm.exec())
```



Figure 25.2.: Damit steht "Hello World" auf dem Bildschirm.

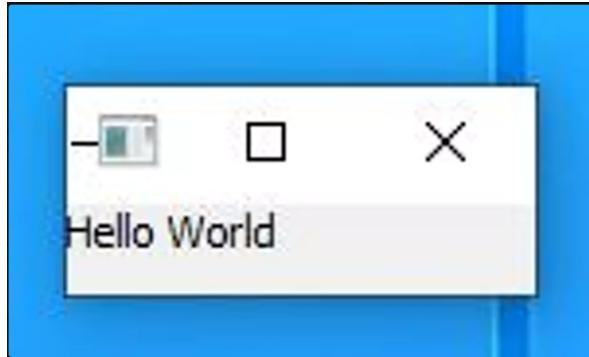


Figure 25.3.: Etwas gestaucht, aber "Hello World" taucht nun im Fenster auf.

25.6. Fenstertitel ändern

Nur eingestellt haben Sie noch nichts. Das Fenster verfügt zwar über einen Minimieren-, Maximieren- und Schließen-Button, ist an sich jedoch grau und öde. Als Fenstertitel steht auch nur ein Nichtssagendes "Python" oben links in der Ecke.

Also weiter: Am einfachsten ist es, den Titel des Fensters zu verändern. Das geht mit `setWindowTitle()`:

```
fenster.setWindowTitle("Hello World")
```

Gut, damit steht nun "Hello World" auf dem Bildschirm. Nun wollen wir den Text direkt im Fenster anzeigen. Dafür holen Sie ein neues Widget ins Programm, das `QLabel`. Es zeigt Texte und Bilder an. Sie importieren es wie `QApplication` oder `QWidget`, indem Sie `QLabel` getrennt durch ein Komma hinter den bestehenden Befehl `Import` schreiben oder eine neue Zeile aufmachen:

```
from PyQt5.QtWidgets import QLabel
```

Unser Label bekommt den Namen `helloLabel` und wird mit dem Hauptfenster `fenster` als Elternelement erstellt:

```
helloLabel = QLabel("Hello World", parent=fenster)
```

25.7. Größe ändern und Fenster verschieben

Wenn Sie das Programm nun ausführen, steht tatsächlich "Hello World" im Fenster. Allerdings ist das Fenster zusammengestaucht. Eine feste Größe können Sie dem Fenster mit `resize()` geben:

```
fenster.resize(400, 200)
```

Der erste Parameter ist die Breite, der zweite Parameter die Höhe. Beide Werte beziehen sich auf Pixel. Nun verschieben Sie das Fenster noch an die gewünschte Position mit `move()`:

```
fenster.move(500, 500)
```

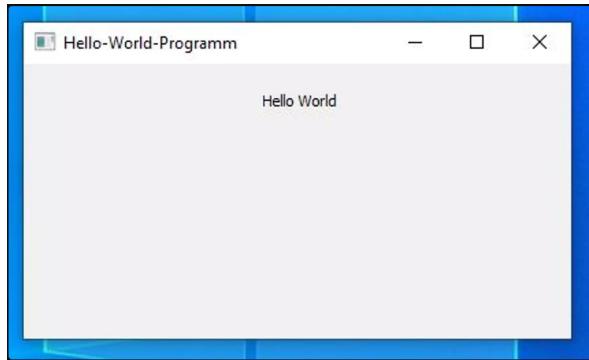


Figure 25.4.: Nun steht "Hello World" zentriert im Fenster.

```

1000 import sys
1001 from PyQt5.QtWidgets import QApplication, QWidget, QLabel
1002
1003 programm = QApplication(sys.argv)
1004 fenster = QWidget()
1005 fenster.setWindowTitle("Hello-World-Programm")
1006 fenster.setGeometry(500, 500, 400, 200)
1007
1008 helloLabel = QLabel("Hello World", parent=fenster)
1009 helloLabel.move(175,20)
1010
1011 fenster.show()
1012
1013 sys.exit(programm.exec())
1014

```

..//code/General/PyQt/PyQtMinimal.py

Listing 25.1.: Komplettes Minimalprogramm

Dabei ist der erste Wert die X-Koordinate und der zweite Wert die Y-Koordinate. Falls Sie es etwas kompakter haben möchten, können Sie resize() und move() auch in einem Attribut zusammenfassen. Nämlich `setGeometry()`:

`fenster.setGeometry(500, 500, 400, 200)`

Der erste Parameter ist die Position auf der X-Achse, der zweite die Position auf der Y-Achse, der dritte beschreibt die Breite in Pixel und der vierte die Höhe in Pixel. Die Position des Label-Widgets im Fenster können Sie ebenfalls bestimmen. Schieben Sie es doch etwas nach rechts und nach unten:

`helloLabel.move(175,20)`

Und schon haben Sie ein nettes Hello-World-Fenster, das sogar ein wenig gelayoutet aussieht.

Der Programmcode dafür ist überschaubar:

25.8. Login-Fenster erstellen

Wie schon beim Tkinter-Beispiel erstellen wir ein Login-Fenster für heise online. So lässt sich der Code der beiden GUI-Frameworks gut miteinander vergleichen und viele grundlegende Funktionen lassen sich während des Codens erklären.

Wie bei Tkinter benötigen Sie auch bei PyQt zu Beginn eine leere Hülle, nämlich das reine Fenster. Im Laufe des Codes füllen Sie es dann mit Widgets und Ereignissen.

Die Hülle besteht aus der Funktion `programm`, die sich um Formalien kümmert und der Klasse `HoLoginGui`, die das grundlegende Fenster zusammenbaut.

Beginnen Sie mit den Formalien in der Funktion `programm`:

```
def programm():
```

Wie schon beim Hello-World-Programm definieren Sie zu Beginn eine `QApplication`, diesmal mit dem Namen `hlogin`. Das `sys.argv` deutet darauf hin, dass das Programm Argumente über die Kommandozeile annehmen kann:

```
hlogin = QApplication(sys.argv)
```

Nun definieren Sie die Variable `gui` und verweisen auf die Klasse `HoLoginGui`, die noch nicht existiert. Aber da soll ja später das grundlegende Fenster entstehen. Mit `show()` bringen Sie das GUI dann auf den Bildschirm der Nutzer:

```
gui = HoLoginGui()
gui.show()
```

Die `QApplication` `hlogin` müssen Sie noch mit einem `exec()` ausführen und für ein standardmäßiges Auslesen des Exitcodes in ein `sys.exit()` packen:

```
sys.exit(hlogin.exec())
```

Die Funktion `programm` ist damit durch. Außerhalb der Funktion, im Hauptprogramm müssen Sie `programm` nur noch aufrufen:

```
programm()
```

Nun fehlt noch die Klasse `HoLoginGui`, auf die Sie in der Funktion `programm` verweisen:

```
class HoLoginGui(QMainWindow):
```

Hier nutzen Sie nicht `QWidget`, wie im Hello-World-Beispiel, sondern `QMainWindow`. Mit `QMainWindow` können Sie recht einfach Statusleisten, Menüs oder Toolbars hinzufügen, daher ist es für viele Programme optimal. Vor allem kann man es verwenden, um ein zentrales Widget festzulegen, das den Kern des Programms bilden soll. Ohne ein zentrales Widget können Sie `QMainWindow` nicht verwenden.

Die folgende Methode initialisiert das `QMainWindow`:

```
def __init__(self):
    super().__init__()
```

Mit `self` ist immer die aktuelle Instanz einer Klasse gemeint. `__init__` ist eine Methode, die in Python bereits reserviert ist und die man verwendet, um ein Objekt zu initialisieren. Sie funktioniert ähnlich wie ein Konstruktor in C++ oder Java. Statt `super().__init__()` könnten Sie auch `QMainWindow.__init__(self)` verwenden. Allerdings vermeiden Sie es mit `super()`, das übergeordnete Element hart reinzucoden und sorgen für eine flexiblere Weiterverwendung des Codes.

Anschließend legen Sie den Titel des Fensters fest und definieren ein `QWidget` als zentrales Widget des `QMainWindows` mit `setCentralWidget()`:

```
self.setWindowTitle("Login fuer heise online")
self.zentralesWidget = QWidget(self)
self.setCentralWidget(self.zentralesWidget)
```

Damit steht das grundlegende Fenster. Vergessen sie nicht die nötigen Importe über `import sys` und `from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget` einzubinden.

So sieht der Code bisher aus:

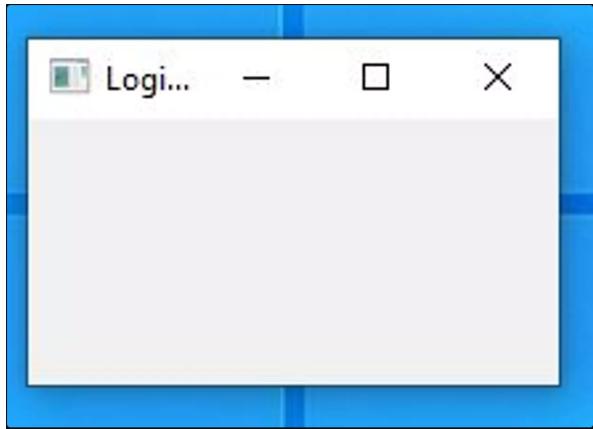


Figure 25.5.: Auch das Login-Fenster beginnt klein, leer und grau.

```
1000 import sys
1001 from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget
1002
1003 class HoLoginGui(QMainWindow):
1004     def __init__(self):
1005         super().__init__()
1006         self.setWindowTitle("Login fuer heise online")
1007
1008         self.zentralesWidget = QWidget(self)
1009         self.setCentralWidget(self.zentralesWidget)
1010
1011     def programm():
1012         hlogin = QApplication(sys.argv)
1013
1014         gui = HoLoginGui()
1015         gui.show()
1016
1017         sys.exit(hlogin.exec())
1018
1019
1020 programm()
```

.. /code/General/PyQt/PyQtClass.py

Listing 25.2.: Komplettes Programm

25.9. Widgets hinzufügen

Wenn Sie den Code ausführen, sehen Sie bereits das kleine, graue Basisfenster vor sich. Wie ein Login-Fenster sieht es allerdings noch nicht aus. Wie schon beim Tkinter-Beispiel benötigen wir einen Beschreibungstext, Eingabefelder für den Benutzernamen und das Passwort, die passenden Labels für die Eingabefelder und natürlich einen Button, auf den der Nutzer klicken kann. Das Tabellschema ist hilfreich, um sich das fertige Programm vorzustellen und die Widgets an die passende Stelle zu packen:

	Beschreibung–Text
1000	Benutzername–Label Benutzername–Eingabefeld
1002	Passwort–Label Passwort–Eingabefeld
	Login–Button

Um dieses Layout zu verwirklichen, müssen Sie noch einige Widgets importieren. `QLabel` kennen Sie ja schon, es sorgt für die Benutzername- und Passwort-LABELs, außerdem können Sie es für den Beschreibungstext verwenden – bei Tkinter war das nur über ein zusätzliches Widget namens Message möglich. Weiterhin brauchen Sie einen Button, den `QPushButton`, und die Eingabefelder, die bei PyQt `QLineEdit` heißen:

```
from PyQt5.QtWidgets import QLabel, QPushButton, QLineEdit
```

25.10. Layout verwalten

Sie können alle Widgets mit `setGeometry()` platzieren, wie im Hello-World-Beispiel. Allerdings ist es nicht sehr elegant, Widgets auf den Pixel genau im Fenster rumzuschieben. Wenn Sie etwa Widgets tauschen oder andere Änderungen vornehmen wollen, halsen Sie sich damit viel Arbeit auf. PyQt bringt vier Manager mit, mit denen Sie das Layout einfacher verwalten können:

- **QHBoxLayout**: Mit diesem Manager platzieren Sie alle Widgets in Kästen horizontal nebeneinander, daher das H im Namen. Über `addWidget()` fügen Sie so von links nach rechts jeweils ein Widget hinzu.
- **QVBoxLayout**: Das V steht diesmal für Vertikal, ansonsten arbeitet es genauso wie `QHBoxLayout`. Alle Widgets fügen Sie mit `addWidget()` von oben nach unten ein.
- **QGridLayout**: Mit diesem Manager platzieren Sie die Widgets in einem Gitter, indem Sie jeweils Reihe und Spalte für die Widgets in `addWidget()` festlegen.
- **QFormLayout**: Mit diesem Manager erstellen Sie ein einfaches Layout, das aus zwei Spalten besteht. Die erste Spalte können Sie etwa für Labels verwenden und die zweite Spalte für Eingabefelder. Von oben nach unten fügen Sie Reihen mit `addRow()` hinzu und definieren dort etwa die Label-Texte und mit `QLineEdit` die Eingabefelder.

Das `QGridLayout` lässt sich flexibel für unsere Zwecke anpassen. Importieren Sie es wie die üblichen Widgets über `from PyQt5.QtWidgets import QGridLayout`. Nun verweisen Sie in der Variable `layout` auf `QGridLayout` und legen es anschließend mit `setLayout()` als Layout für das zentrale Widget fest:

```
self.layout = QGridLayout()
self.zentralesWidget.setLayout(self.layout)
```

25.11. Widgets erstellen

Nun fügen Sie die Widgets ein. Dafür erstellen Sie eine neue Methode namens `widgets` und binden Sie über `self.widgets()` in die Methode `__init__` ein.

Jedes Widget platzieren Sie nun einzeln über `addWidget` in das vorher erstellte `layout`. Beginnen Sie mit dem Beschreibungstext. Es soll in Spalte 0 und Reihe 0 beginnen und sich über zwei Spalten ziehen. Der Text soll außerdem "Gib deine Logindaten für heise online ein und drücke dann den Button." lauten. Das erreichen sie mit diesem Code:

```
self.layout.addWidget(QLabel("Gib deine Logindaten fuer heise online  
ein und druecke dann den Button."), 0, 0, 1, 2)
```

`self.layout.addWidget()` fügt dem `QGridLayout` hinter der Variable `layout` ein Widget hinzu. Der erste Parameter ist das Widget, ein `QLabel` mit dem vorher definierten Text in Anführungszeichen. Der zweite Parameter ist die Reihe, der dritte die Spalte, in der das Widget erscheinen soll. Es soll in der ersten Spalte und der ersten Reihe erscheinen, daher werden beide Parameter mit einer Null gefüllt – die Zählung beginnt bei Null, nicht bei Eins. Der vierte Parameter gibt an, über wie viele Reihen sich das Widget strecken soll – in diesem Fall nur über eine. Im fünften und letzten Parameter legen Sie schließlich fest, über wie viele Spalten sich das `QLabel` ziehen soll, nämlich über zwei.

Nun fügen Sie die Benutzername- und Passwort-Labels hinzu:

```
self.layout.addWidget(QLabel("Benutzername:"), 1, 0)  
self.layout.addWidget(QLabel("Passwort:"), 2, 0)
```

Auch dabei handelt es sich um `QLabels`, sie werden im zweiten Parameter in Reihe eins beziehungsweise Reihe zwei platziert. Die Spalte ist bei beiden die erste, also ist der dritte Parameter jeweils eine Null. Da sie sich nicht über mehrere Reihen oder Spalten ziehen sollen, müssen Sie den vierten und fünften Parameter nicht ausfüllen, sie bleiben standardmäßig auf `1`.

Es folgen die Eingabefelder, die Sie mit einem `QLineEdit` in Reihe eins und zwei hinzufügen:

```
self.layout.addWidget(QLineEdit(), 1, 1)  
self.layout.addWidget(QLineEdit(), 2, 1)
```

Die Spalte ist diesmal auf `1` gesetzt, nicht auf Null. So erscheinen die Eingabefelder rechts neben den Labels.

Zu guter Letzt benötigen Sie noch den Button, den Sie über `QPushButton` einbinden:

```
self.layout.addWidget(QPushButton("Einloggen"), 3, 0, 1, 2)
```

Er wird mit dem Text "Einloggen" versehen, in Reihe drei (Parameter zwei) und Reihe null (Parameter drei) platziert und streckt sich über eine Reihe (Parameter vier) und zwei Spalten (Parameter fünf). Das Strecken über zwei Spalten kennen Sie schon vom Beschreibungstext.

Wenn Sie nun das Programm starten, können Sie Ihr Layout begutachten. Der Beschreibungstext und der Button strecken sich wie beabsichtigt über zwei Spalten, die Labels mit "Benutzername:" und "Passwort:" haben jeweils ein Eingabefeld rechts daneben. Das sieht doch schon Mal nach einem Login-Fenster aus. Der Button ist zwar klickbar, tut aber noch nichts.

Das Fenster ist momentan noch am Beschreibungstext ausgerichtet und daher langgezogen. Mit einem `\n` im Text geben Sie dem `QLabel` an der passenden Stelle einen Zeilenumbruch:

```
self.layout.addWidget(QLabel("Gib deine Logindaten fuer heise online  
ein \n und druecke dann den Button."), 0, 0, 1, 2)
```

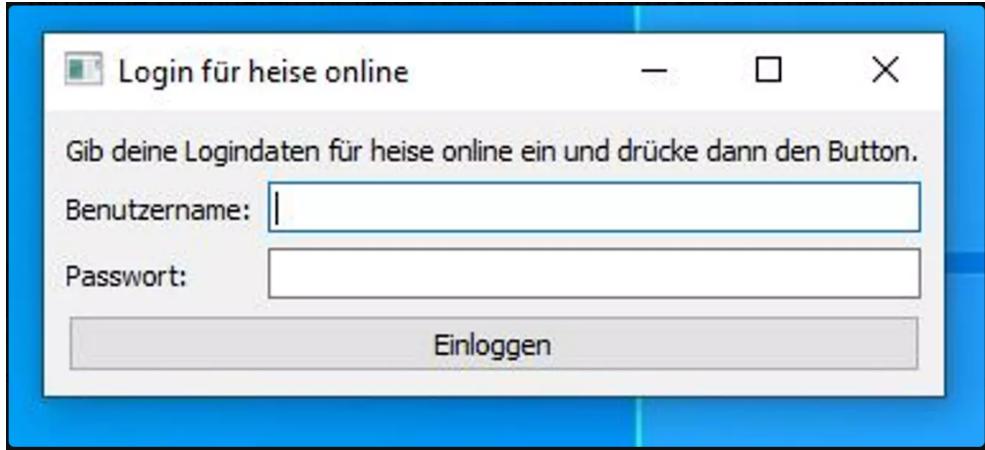


Figure 25.6.: Das sieht schon nach einem Login-Fenster aus.

25.12. Elemente ausrichten

Der Zeilenumbruch steht nun an der passenden Stelle, aber der Text im Widget ist noch linksbündig. Um es zu zentrieren, bauen wir das `QLabel` etwas um und importieren neue Methoden:

```
from PyQt5.QtCore import Qt
```

Denn um den Text zu zentrieren, benötigen Sie die Ausrichtungsmethoden aus dem `QtCore`.

Nun lagern Sie das Widget in die Variable `labelBeschreibungstext` aus:

```
self.labelBeschreibungstext = QLabel("Gib deine Logindaten fuer heise  
online ein \n und druecke dann den Button.")
```

Anschließend wenden Sie `setAlignment` darauf an und richten es mit `Qt.AlignCenter` mittig aus:

```
self.labelBeschreibungstext.setAlignment(Qt.AlignCenter)
```

Dann platzieren Sie das Widget wieder im Gitter, verweisen aber nur noch auf die Variable:

```
self.layout.addWidget(self.labelBeschreibungstext, 0, 0, 1, 2)
```

Auch die Labels für den Benutzernamen und das Passwort gehören neu ausgerichtet. Sie sollen rechtsbündig an den zugehörigen Eingabefeldern kleben. Dafür packen Sie die Labels ebenfalls in neue Variablen:

```
self.labelBenutzername = QLabel("Benutzername:")  
self.labelPasswort = QLabel("Passwort:")
```

Dieses Mal regeln Sie die Ausrichtung allerdings nicht über `setAlignment`, sondern verwenden einen neuen Parameter namens `alignment` in `addWidget`. Damit beeinflussen Sie die gesamte Ausrichtung des Widgets, nicht nur die Ausrichtung des Textes:

```
self.layout.addWidget(self.labelBenutzername, 1, 0, alignment=Qt.AlignRight)  
self.layout.addWidget(self.labelPasswort, 2, 0, alignment=Qt.AlignRight)
```

Genauso verfahren Sie mit den Eingabefeldern, verpassen ihnen aber ein `AlignLeft`, um sie an die linke Seite zu tackern. Außerdem bekommen Sie eine feste Größe von 120 Pixel mit `setFixedWidth`:

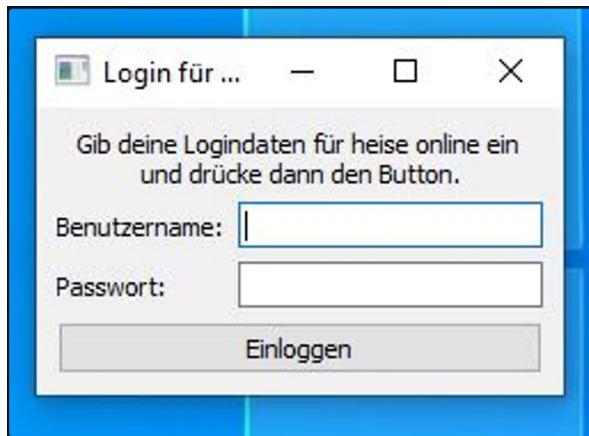


Figure 25.7.: Jetzt hat das Fenster die passende Größe.

```
self.eingabeBenutzername = QLineEdit()
self.eingabeBenutzername.setFixedWidth(120)

self.eingabePasswort = QLineEdit()
self.eingabePasswort.setFixedWidth(120)

self.layout.addWidget(self.eingabeBenutzername, 1, 1, alignment=Qt.AlignLeft)
self.layout.addWidget(self.eingabePasswort, 2, 1, alignment=Qt.AlignLeft)
```

Zudem möchten Sie, dass der Text durch Punkte ersetzt wird, wenn ein Nutzer sein Passwort eingibt – es könnte ja jemand über die Schulter schauen. Dafür nutzen Sie `setEchoMode` und `QLineEdit.Password`:

```
self.eingabePasswort.setEchoMode(QLineEdit.Password)
```

Der Button fehlt noch. Er benötigt zwei Ausrichtungen: Er soll nämlich mittig platziert sein und gleichzeitig oben an den anderen Elementen kleben. Ein | trennt verschiedene Ausrichtungen:

```
self.buttonEinloggen = QPushButton("Einloggen")
self.buttonEinloggen.setFixedWidth(120)
self.layout.addWidget(self.buttonEinloggen, 3, 0, 1, 2, alignment=Qt.AlignCenter|Qt.AlignTop)
```

Einen letzten Parameter `alignment` benötigen Sie noch, nämlich für den Beschreibungstext. Er soll unten an den anderen Elementen kleben. Dafür sorgt `Qt.AlignBottom`:

```
self.layout.addWidget(self.labelBeschreibungstext, 0, 0, 1, 2, alignment=Qt.AlignBottom)
```

Und damit steht auch schon das grundlegende Layout inklusive Maskierung des Passworts:

```
self.labelBeschreibungstext = QLabel("Gib deine Logindaten fuer heise
online ein \n und druecke dann den Button.")
self.labelBeschreibungstext.setAlignment(Qt.AlignCenter)

self.labelBenutzername = QLabel("Benutzername:")
self.labelPasswort = QLabel("Passwort:")

self.eingabeBenutzername = QLineEdit()
self.eingabeBenutzername.setFixedWidth(120)
```

```
self.eingabePasswort = QLineEdit()
self.eingabePasswort.setEchoMode(QLineEdit.Password)
self.eingabePasswort.setFixedWidth(120)

self.buttonEinloggen = QPushButton("Einloggen")
self.buttonEinloggen.setFixedWidth(120)

self.layout.addWidget(self.labelBeschreibungstext, 0, 0, 1, 2, alignment=Qt.AlignBottom)
self.layout.addWidget(self.labelBenutzername, 1, 0, alignment=Qt.AlignRight)
self.layout.addWidget(self.labelPasswort, 2, 0, alignment=Qt.AlignRight)
self.layout.addWidget(self.eingabeBenutzername, 1, 1, alignment=Qt.AlignLeft)
self.layout.addWidget(self.eingabePasswort, 2, 1, alignment=Qt.AlignLeft)
self.layout.addWidget(self.buttonEinloggen, 3, 0, 1, 2, alignment=Qt.AlignCenter | Qt.AlignTop)
```

25.13. Button klickbar machen

Das Fenster sieht gut aus und verhält sich auch schon recht ordentlich. Allerdings führt der Klick auf den Button noch ins Leere. Erstellen Sie eine neue Methode namens `buttonGeklickt`:

```
def buttonGeklickt(self):
```

Nun verbinden Sie den Button aus der Methode `widgets` mit den noch zu erstellenden Aktionen in der Methode `buttonGeklickt`. Die Verbindung entsteht bei PyQt über Signale und Slots. Ein Signal kann etwa ein Buttonklick sein, ein veränderter Text, eine markierte Checkbox und noch viel mehr. Ein Slot ist etwa eine Methode, die aufgerufen wird, wenn ein Widget das Signal aussendet. Signal und Slot verbindet man mit einem simplen `connect()`.

Für unser Beispielprojekt sieht die Verbindung in der Widget-Methode so aus:

```
self.buttonEinloggen.clicked.connect(self.buttonGeklickt)
```

`self.buttonEinloggen` ist der QPushButton, `clicked` ist das Signal, nämlich ein geklickter Button, `connect()` ist die Verbindung und `self.buttonGeklickt` ist die neu erstellte Methode, in der nach dem Klick Aktionen stattfinden.

25.14. Einloggen

Nun ist `buttonGeklickt` noch leer, das sollten Sie ändern. Als erstes benötigen Sie die Eingaben des Nutzers, also seinen Benutzernamen und das Passwort. Diese Daten holen Sie mit `text()` aus den QLineEdits:

```
self.benutzername = self.eingabeBenutzername.text()
self.passwort = self.eingabePasswort.text()
```

Beim weiteren Login orientieren wir uns an dem Code, der schon im Tkinter-Beispiel gut funktioniert hat. Sie definieren einen Fake-Browser in einem Dictionary, um sich einzuloggen:

```
self.fake_browser = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0"}
```

Der User-Agent enthält Daten über den Browser und dem verwendeten System, was anschließend an die Website gesendet wird. Ihren eigenen User-Agent sehen Sie etwa auf <http://wieistmeinuseragent.de>.

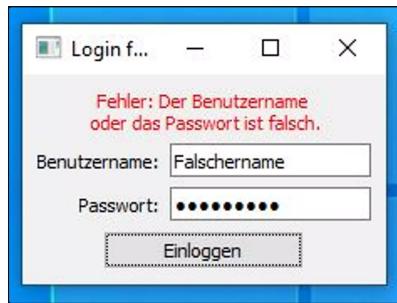


Figure 25.8.: Oh, da ist etwas schiefgelaufen.

Die Daten für den Login speichern Sie in einem weiteren Dictionary:

```
self.login_daten = "username":self.benutzername, "password":self.passwort, "action":"/sso/login/login"}
```

Bei heise online loggen Sie sich über die Website <https://www.heise.de/sso/login/login> ein. Verantwortlich für den Login ist eine Methode **POST** mit der Aktion **/sso/login/login**, daher wird auch sie im Dictionary festgelegt.

Der Login läuft dann über **requests**, das Sie vorher importieren müssen:

```
import requests

self.login = requests.Session().post(url="https://www.heise.de/sso/login/login",
data=self.login_daten, headers=self.fake_browser)
```

In einer **Session()** werden Cookies gespeichert und Sie können später mit aktivem Login andere Seiten aufrufen. **post()** steht für die HTML-Methode **POST**, die **url** ist die heise-Login-Seite, data enthält die vorher festgelegten Login-Daten und in headers geben Sie den Fake-Browser als User-Agent mit.

Der Login steht, beim Klick auf den Button passiert aber noch immer nichts. Kein Wunder, läuft der Login doch im Hintergrund ab. Es gibt noch keinen visuellen Hinweis, ob der Login geklappt hat. Als nächstes wollen wir daher den Beschreibungstext ändern, entweder in eine Erfolgsmeldung oder in einen Hinweis, dass der Benutzername oder das Passwort falsch war.

Falls der Login nicht geklappt hat, erkennen Sie dies, wenn in **self.login.text** die Zeile "Der Benutzername oder das Passwort ist falsch." auftaucht. Das nutzen Sie für eine If-Abfrage:

```
if "Der Benutzername oder das Passwort ist falsch." in self.login.text:
    self.labelBeschreibungstext.setText("Fehler: Der Benutzername \n oder
    das Passwort ist falsch.")
    self.labelBeschreibungstext.setStyleSheet("color: red;")
```

setText ändert den Inhalt des Beschreibungstextes. Mit **setStyleSheet** ändern Sie die Farbe des Textes auf Rot.

Nun müssen Sie noch ein **else** definieren, falls der Login richtig war:

```
else:
    self.labelBeschreibungstext.setText("Sie haben sich erfolgreich ein-
    geloggt.")
    self.labelBeschreibungstext.setStyleSheet("color: green;")
```

Auch hier ändern Sie wieder den Text, färben ihn aber dieses Mal grün ein.

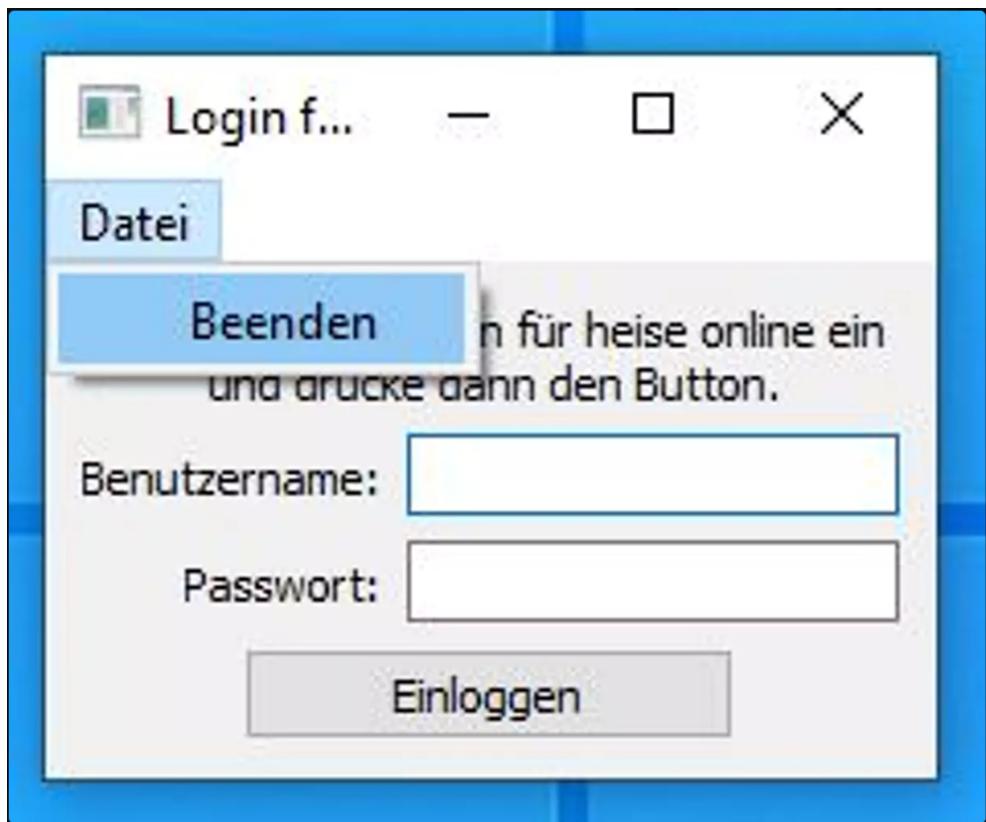


Figure 25.9.: Mit einem Menü ist das Fenster komplett.

25.15. Menü hinzufügen

Wie schon beim Tkinter-Beispiel soll auch dieses Fenster ein kleines Menü bekommen, damit der Nutzer es einfach beenden kann. Da Sie das GUI als `QMainWindow` angelegt haben, kostet das kaum Codezeilen.

Erst erstellen Sie eine neue Methode namens `menu`:

```
def menu(self):
```

Dort legen Sie nun ein Datei-Menü an über `self.menuBar().addMenu()`:

```
self.menu = self.menuBar().addMenu("&Datei")
```

Datei in den Anführungszeichen ist dabei der Name des Menüs, das **&** macht das **D** zu einem Shortcut, um das Dateimenü aufzurufen, also den ersten Buchstaben. Dieses Menü ist noch leer, es fehlt eine Aktion. Mit `addAction()` bringen Sie Leben rein:

```
self.menu.addAction("&Beenden", self.close)
```

Der Eintrag heißt **Beenden** und wenn man darauf klickt, wird `self.close` ausgeführt – der Nutzer schließt damit das Programm. Die Methode `menu` müssen Sie nun noch in der Methode `__init__` aufrufen:

```
self.menu()
```

Und damit haben Sie ein kleines Login-Programm gebaut, inklusive Reaktion und Menü:

```

1000 import sys
1001 import requests
1002 from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget
1003 from PyQt5.QtWidgets import QGridLayout
1004 from PyQt5.QtWidgets import QLabel, QPushButton, QLineEdit
1005 from PyQt5.QtCore import Qt
1006
1007
1008 class HoLoginGui(QMainWindow):
1009     def __init__(self):
1010         super().__init__()
1011         self.setWindowTitle("Login fuer heise online")
1012
1013         self.zentralesWidget = QWidget(self)
1014         self.setCentralWidget(self.zentralesWidget)
1015
1016         self.layout = QGridLayout()
1017         self.zentralesWidget.setLayout(self.layout)
1018
1019         self.menu()
1020         self.widgets()
1021
1022     def widgets(self):
1023         self.labelBeschreibungstext = QLabel("Gib deine Logindaten fuer
1024             heise online ein \n und druecke dann den Button.")
1025         self.labelBeschreibungstext.setAlignment(Qt.AlignCenter)
1026
1027         self.labelBenutzername = QLabel("Benutzername:")
1028         self.labelPasswort = QLabel("Passwort:")
1029
1030         self.eingabeBenutzername = QLineEdit()
1031         self.eingabeBenutzername.setFixedWidth(120)
1032
1033         self.eingabePasswort = QLineEdit()
1034         self.eingabePasswort.setEchoMode(QLineEdit.Password)
1035         self.eingabePasswort.setFixedWidth(120)
1036
1037         self.buttonEinloggen = QPushButton("Einloggen")
1038         self.buttonEinloggen.setFixedWidth(120)
1039         self.buttonEinloggen.clicked.connect(self.buttonGeklickt)
1040
1041         self.layout.addWidget(self.labelBeschreibungstext, 0, 0, 1, 2,
1042             alignment=Qt.AlignBottom)
1043         self.layout.addWidget(self.labelBenutzername, 1, 0, alignment=Qt.
1044             AlignRight)
1045         self.layout.addWidget(self.labelPasswort, 2, 0, alignment=Qt.
1046             AlignRight)
1047         self.layout.addWidget(self.eingabeBenutzername, 1, 1, alignment=
1048             Qt.AlignLeft)
1049         self.layout.addWidget(self.eingabePasswort, 2, 1, alignment=Qt.
1050             AlignLeft)
1051         self.layout.addWidget(self.buttonEinloggen, 3, 0, 1, 2,
1052             alignment=Qt.AlignCenter | Qt.AlignTop)
1053
1054     def menu(self):
1055         self.menu = self.menuBar().addMenu("&Datei")
1056         self.menu.addAction("&Beenden", self.close)
1057
1058     def buttonGeklickt(self):
1059         self.benutzername = self.eingabeBenutzername.text()
1060         self.passwort = self.eingabePasswort.text()
1061
1062         self.fake_browser = {"User-Agent": "Mozilla/5.0 (Windows NT
1063             10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0"}
1064         self.login_daten = {"username": self.benutzername, "password": self.
1065             passwort, "action": "/sso/login/login"}
1066
1067         self.login = requests.Session().post(url="https://www.heise.de/
1068             sso/login/login",
1069                                         data=self.login_daten,
1070                                         headers=self.fake_browser)
1071
1072         if "Der Benutzername oder das Passwort ist falsch." in self.
1073             login.text:
1074             self.labelBeschreibungstext.setText("Fehler: Der
1075                 Benutzername \n oder das Passwort ist falsch.")
1076             self.labelBeschreibungstext.setStyleSheet("color: red;")
1077         else:
1078             self.labelBeschreibungstext.setText("Sie haben sich
1079                 erfolgreich eingeloggt.")
1080             self.labelBeschreibungstext.setStyleSheet("color: green;")

```

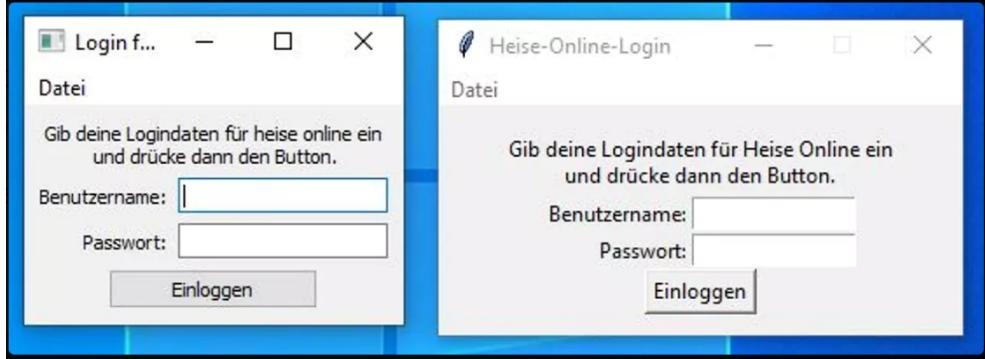


Figure 25.10.: PyQt (links) wirkt moderner als das Login-Programm mit Tkinter (rechts).

25.16. Fazit

Mit PyQt lassen sich mit nur wenigen Dutzend Codezeilen kleine Programme erstellen. Tkinter-Code ist dagegen zwar noch etwas kompakter, dafür sind die einzelnen Zeilen eines PyQt-Programms besser zu verstehen. Vor allem das System aus Signalen und Slots bei PyQt macht es sehr leicht, Aktionen mit Reaktionen zu verknüpfen.

Während die Tkinter-Widgets von Haus aus etwas altbackener wirken, sehen die Widgets eines PyQt-Programms recht modern aus. Tkinter versucht sich immerhin mit dem Modul tkk einen modernen Look zu verpassen. Tkinter hat den Vorteil, dass Sie keine externen Bibliotheken installieren müssen. Bei PyQt sollten Sie hingegen genau darauf achten, welche Module Sie importieren, um das Programm nicht zu überlasten.

Per Hand werden Sie vermutlich bei komplexeren Programmen schnell an Grenzen stoßen. Für Qt gibt es etwa den kostenlosen Qt Designer, [der Teil des Qt Creators ist](#), einer Entwicklungsumgebung für Qt-Projekte. Wer nicht alles per Hand coden will, kommt auch so auf eine schöne Bedienoberfläche für seine Nutzer.

26. GUI für Python: Bedienoberfläche per Drag-and-Drop mit dem Qt Designer erstellen

Mit dem Qt Designer erstellen Sie schnell Bedienoberflächen für Python-Programme. Die Elemente lassen sich intuitiv anordnen, coden muss man nur noch wenig.

Es ist oft recht mühsam, Bedienoberflächen für Software per Hand zu programmieren. Man muss viele Einstellungen fürs Layout festlegen und am Ende hat man ein Kuddelmuddel aus Design und Funktion im Code stehen. Gerade bei komplexen grafischen Benutzeroberflächen (Graphical User Interface/GUI) ufert der Code schnell aus und man kann die Übersicht verlieren.

Mit dem Qt Designer erstellen Sie per Drag-and-drop einfache oder komplexe Layouts. Über das GUI-Toolkit PyQt kann man das erstellte Layout dann für sein Python-Programm nutzen. Dabei werden Aussehen und Logik streng getrennt: Das Design wird in einer extra UI-Datei ausgelagert, die Funktionen dafür erstellt man wie gewohnt in der Python-Datei. PyQt-Kentnisse helfen natürlich, allerdings eignet sich der Designer auch gut, um grundlegende Konzepte von PyQt und beziehungsweise Qt zu lernen. Wir zeigen, wie man ein simples Login-Fenster baut. Das Fenster besteht aus einem Hinweistext, zwei Labels für Passwort und Benutzername, zwei Eingabefeldern und einem Button. Der Nutzer soll seine Daten für heise online eingeben und nach dem Buttonklick eine Rückmeldung erhalten, ob der Login funktioniert hat. Außerdem soll es ein kleines Menü geben, über das der Nutzer das Programm beendet. Für dieses Beispiel nutzen wir PyQt5 und Python 3.8 auf Windows 10.

26.1. Installation

Am einfachsten holt man sich den Designer über die pyqt5-tools:

```
pip install pyqt5-tools
```

Anschließend findet man etwa auf einem Windows-Rechner die [Designer.exe](#) unter [../Lib/site-packages/qt5_applications/Qt/bin](#). Der österreichische Entwickler Michael Hermann bietet zudem Standalone-Versionen des Designers [für Windows](#) oder [MacOS](#) auf seiner Website an.

Später benötigt man noch PyQt5, das sich per pip installieren lässt:

```
pip install pyqt5
```

26.2. Erster Start

Startet man den Designer zum ersten Mal, soll der Nutzer gleich ein neues Formular erstellen und kann aus verschiedenen Vorlagen wählen: Dialog mit Buttons unten, Dialog mit Buttons rechts, Hauptfenster, Widget und so weiter.

Als erstes Beispiel wollen wir nur ein Fenster auf den Bildschirm bringen, in dem der Text ‘Hello World!’ steht. Dafür reicht die Vorlage Main Window völlig aus. Die Bildschirmgröße belässt man auf “Vorgabe” und klickt anschließend auf “Neu von Vorlage”. Im Arbeitsbereich des Qt Designers erscheint ein Fenster mit grauem Hintergrund und schwarzen Punkten.

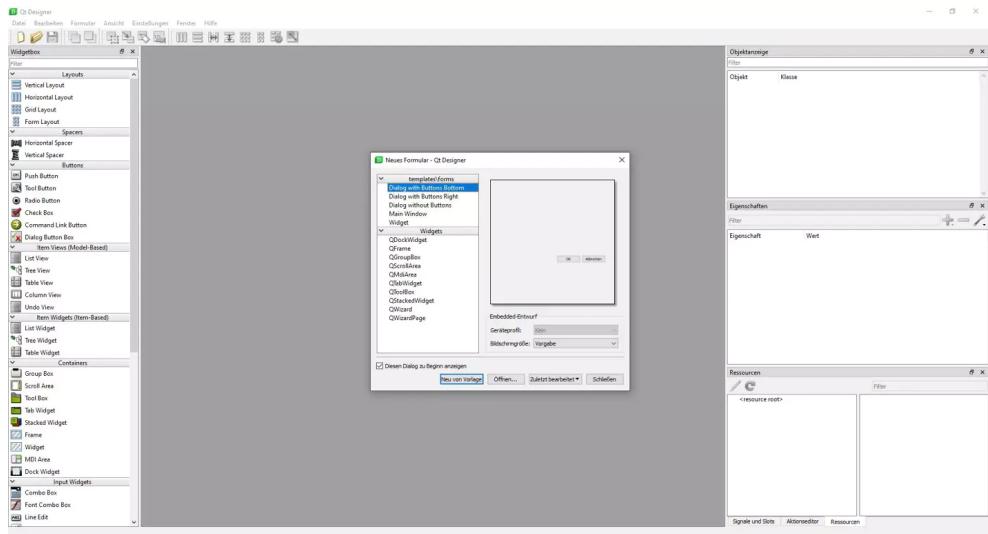


Figure 26.1.: Der Qt Designer wirkt so, als hätte man das Fensterlayout mit dem Qt Designer erstellt.

26.3. Bedienoberfläche

Der Arbeitsbereich wird von mehreren Menüs umrahmt: links findet man die Widgetbox. Nutzer können Elemente wie Listen, Checkboxen oder Dropdown-Menüs daraus per Drag-and-drop in den Arbeitsbereich ziehen. Die Widgets sind in verschiedene Gruppen unterteilt, etwa Input Widgets, Layouts oder Container. Über ein Eingabefeld am oberen Rand kann der Nutzer gezielt nach bestimmten Widgets suchen.

Rechts stehen die Objektanzeige, die Eigenschaften des Layouts sowie ein Tabmenü für die genutzten Ressourcen, Aktionen und das Signale-und-Slots-System von Qt. In der Objektanzeige findet man die derzeit genutzten Widgets – das Hauptfenster besteht in diesem Fall aus dem Objekt `MainWindow`, das ein zentrales Widget, eine Menüleiste und eine Statusleiste enthält. In einem Hauptfenster bei Qt ist das zentrale Widget häufig umgeben von Dock Widgets, die wiederum von Toolbars eingerahmt sind. Am oberen Rand des Fensters steht dann die Menüleiste, am unteren Rand die Statusleiste.

Der Qt Designer selbst ist ein schönes Beispiel für das Layout eines Hauptfensters mit zentralem Widget. Es hat Dock Widgets an den Seiten, sowie jeweils eine Menü-, Tool- und Statusleiste.:

In den Eigenschaften findet man etwa Optionen für die Größe des Fensters – hier 800×600 Pixel –, das Symbol für den Mauszeiger oder die verwendete Schriftart.

Unter den Eigenschaften steht schließlich das Tabmenü mit drei weiteren Menüs:

- Über das Ressourcen-Menü lassen sich externe Dateien wie etwa Bilder einbinden.
- Das Aktionen-Menü erleichtert es, verschiedenen Elementen eine bestimmte Aktion zuzuweisen. Ein Beispiel: In einem Programm soll der Nutzer eine neue Datei anlegen. Das geht über einen Menüeintrag, ein Tastenkürzel und über einen Button in einer Toolbar. Anstatt nun an drei verschiedenen Stellen die gleichen Befehle zu hinterlegen, erstellt man eine Aktion und verknüpft sie mit verschiedenen Objekten.
- Das Signale-und-Slots-System: Ein Signal kann etwa ein Klick auf einen Button sein, ein veränderter Text, eine markierte Checkbox und noch viel mehr. Alles,

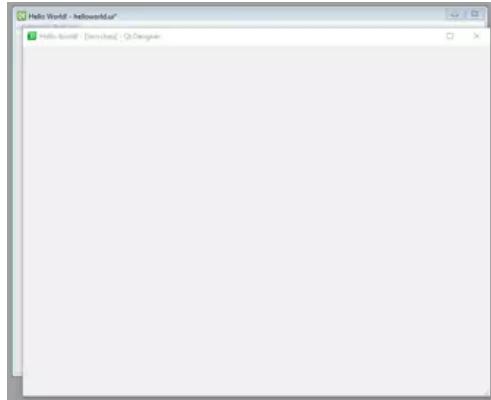


Figure 26.2.: Im Titel des Fensters steht nun "Hello World!" – das zählt.

was ein Widget sendet, kann ein Signal sein. Ein Slot ist etwa eine Methode, die aufgerufen wird, wenn ein Widget das Signal aussendet.

26.4. Hello World!

Der einfachste Weg, ein "Hello World!" auf den Bildschirm zu zaubern: den Titel des Fensters verändern. In den Eigenschaften sucht man dafür nach dem Eintrag `windowTitle` und ändert den zugehörigen Wert zu "Hello World!".

Mit der Tastenkombination STRG+R können Sie eine Live-Vorschau der bisher erstellten Bedienoberfläche aufrufen: Ein Fenster mit grauem Hintergrund erscheint, es ist 800×600 Pixel groß und im Titel steht tatsächlich "Hello World!".

Das ist für den Anfang nicht schlecht, aber der Text soll doch lieber in der Mitte des Fensters stehen, nicht nur im Titel. Schließen Sie dafür das Vorschaufenster und ziehen Sie dann ein Label-Widget in den Arbeitsbereich. Ein Label ist genau für diesen Zweck gemacht, es soll schlicht Text anzeigen.

Mit einem Doppelklick im Label lässt sich der Text anpassen. Dort tippt man einfach "Hello World!" ein. Das Standardlabel dürfte für diesen Text zu klein sein: Ziehen Sie an den blauen Quadranten, um die Größe des Labels zu ändern. Wer lieber Zahlen mag, passt die Größe in den Eigenschaften unter `geometry` an. Nach einem STRG+R sieht man nun, dass der Hello-World-Text nicht nur im Titel des Fensters steht, sondern auch in der Mitte erscheint.

Standardmäßig ist der Text im Label linksbündig ausgerichtet. Über die Option `alignment` des Bereichs `QLabel` in den Eigenschaften kann man das ändern. Neben linksbündig stehen hier zentriert, rechtsbündig oder Blocksatz zur Auswahl. Zusätzlich lässt sich der Text etwa am oberen oder unteren Rand ausrichten. Die Schriftart und -größe legt man im Abschnitt `QWidget` über die Option `font` fest. Hier lässt sich der Text auch kursiv gestalten, unterstreichen, durchstreichen und so weiter.

26.5. Bedienoberfläche in Python integrieren

Momentan existiert das Hello-World-Fenster nur im Qt Designer. Damit ein Python-Skript darauf zugreifen kann, muss man es exportieren. Im Designer speichert man die Datei wie gewohnt unter "Datei", "Speichern unter..."; ein passender Name wäre `helloworld.ui`.

Der Designer speichert die Bedienoberfläche als UI-Datei. Das ist eine Datei im XML-Format, in der alle relevanten Widgets, Einstellungen und Layout-Entscheidungen

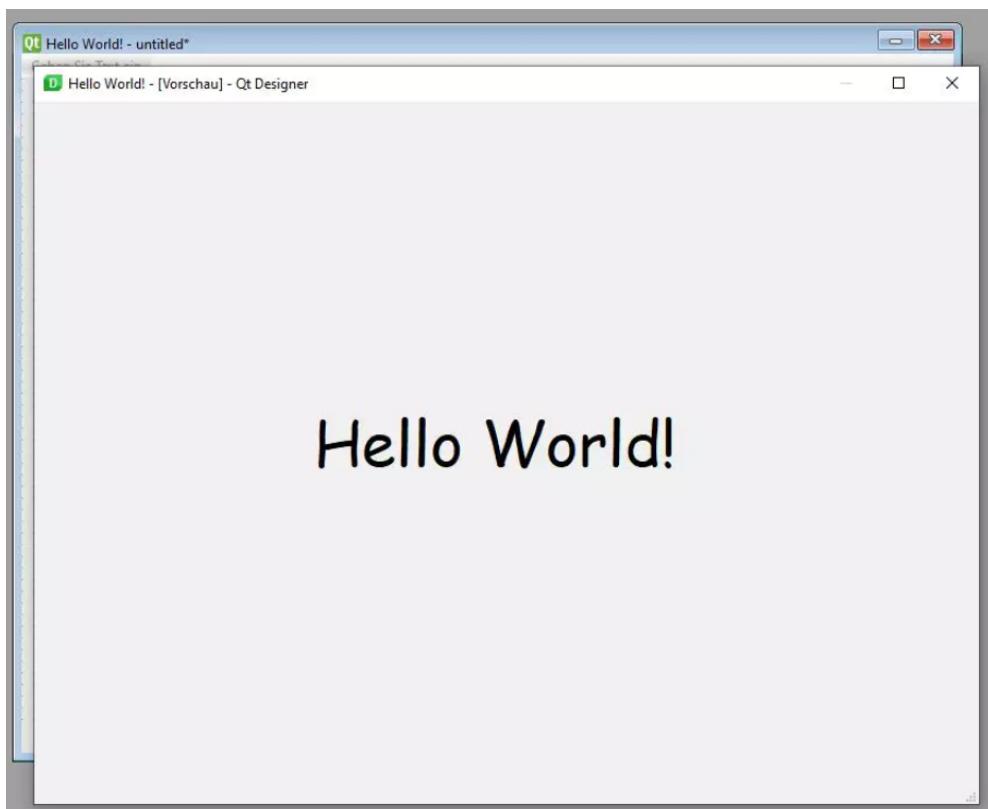


Figure 26.3.: Ja, das ist Comic Sans.

stehen. Es gibt verschiedene Möglichkeiten, wie daraus die Bedienoberfläche für ein Python-Programm wird.

26.6. pyuic5

Mit diesem Befehl in der Konsole wird aus der zuvor erstellten helloworld.ui-Datei eine Datei `helloworld.py`:

```
pyuic5 helloworld.ui -o helloworld.py
```

pyuic5 haben Sie zuvor mit dem Designer über die PyQt5-tools installiert. Das `-o` steht schlicht für Output.

In der neuen Python-Datei wird das komplette Layout nachgebildet, das vorher im Qt Designer erstellt wurde. Die ersten Zeilen von `helloworld.py` sehen in diesem Beispiel so aus:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'helloworld.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5
# is
# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
```

Die Python-Datei auszuführen bringt noch nicht das Hello-World-Fenster auf den Bildschirm. Aber Sie können nun in einer weiteren Python-Datei auf die Funktion `setupUi` in der Klasse `Ui_MainWindow` verweisen und so die Bedienoberfläche aufrufen. Dafür erstellt man etwa eine Datei namens `main.py` im selben Ordner und importiert zu Beginn die Widgets von PyQt5 sowie die Datei `helloworld.py`:

```
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys
import helloworld
```

Ein kleines Programm, dass das Fenster anzeigt, kann dann etwa so aussehen:

```
class HelloworldProgramm(QMainWindow, helloworld.Ui_MainWindow):
    def __init__(self, parent=None):
        super(HelloworldProgramm, self).__init__(parent)
        self.setupUi(self)

    programm = QApplication(sys.argv)
    fenster = HelloworldProgramm()
    fenster.show()
    sys.exit(programm.exec())
```

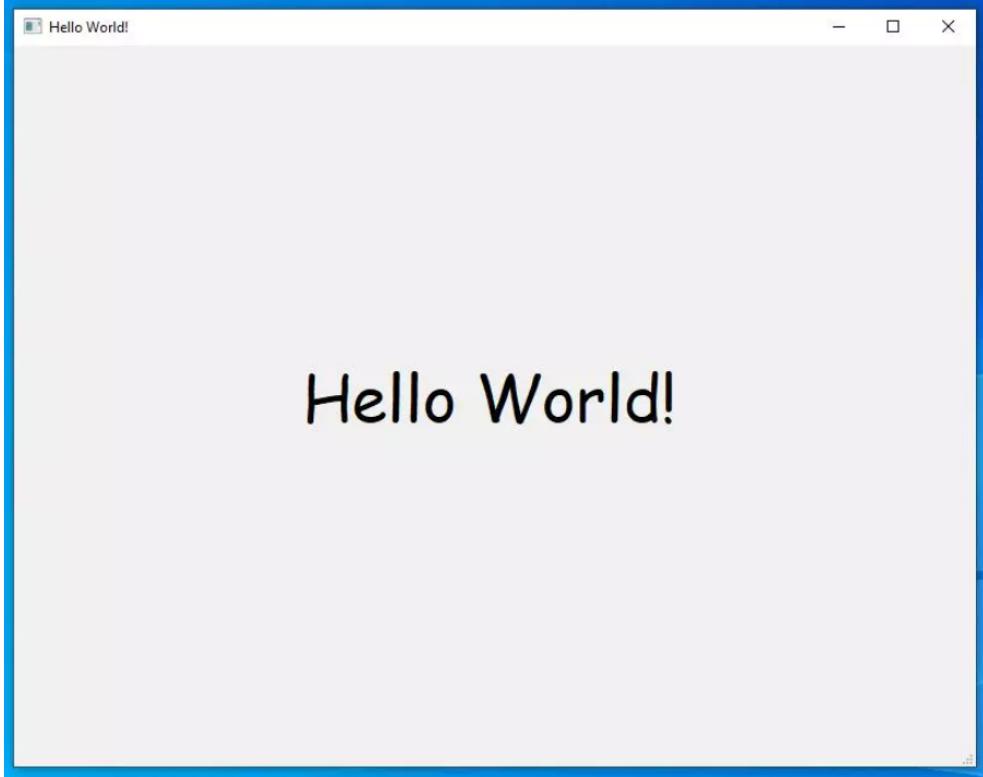


Figure 26.4.: Keine Sorge, auch die gewählte Schriftart aus dem Designer wird vom Python-Skript passend angezeigt.

Mit `QApplication` legen Sie die wichtigsten Eigenschaften eines Fensters fest. Das Programm beginnt damit, dass `QApplication` startet und es wird geschlossen, wenn `QApplication` beendet wird. `QApplication` hält währenddessen eine Schleife aufrecht, in der alle Ereignisse verarbeitet werden, die im Fenster passieren, den Mainloop. Kurz gesagt: Ohne `QApplication` gäbe es kein Programm.

`programm.exec()` würde alleine schon reichen, um die unendliche Schleife zu starten. Allerdings hat es sich eingebürgert, die Hauptschleife in ein `sys.exit()` zu packen. Damit gibt `sys.exit` den Exitcode des Programms an das Elternelement weiter, wenn Sie das Programm beenden – etwa an den Explorer oder die Kommandozeile, womit Sie das Programm gestartet haben.

Der Befehl `sys.argv` ist wichtig, falls Ihr Programm Argumente über die Kommandozeile verarbeiten soll: `sys.argv` ist quasi eine Liste, die den Dateinamen des Projekts enthält sowie alle Argumente, die über die Kommandozeile kommen. Für `argv` und `exit` wird die Bibliothek `sys` importiert, die systemspezifische Funktionen bereithält. `fenster` ist hier ein Verweis auf die Klasse `HelloworldProgramm`, die schließlich mit `fenster.show()` startet. In `HelloworldProgramm` wird dann `self.setupUi(self)` aufgerufen, das auf die Funktion `setupUi()` in der Klasse `Ui_MainWindow()` verweist, die in der Datei `helloworld.py` steht – das ist die Python-Datei, die Sie vorher aus der UI-Datei generiert haben und die alle Elemente enthält.

Mit `self` ist immer die aktuelle Instanz einer Klasse gemeint. `__init__` ist eine Methode, die in Python bereits reserviert ist und die man verwendet, um ein Objekt zu initialisieren. Sie funktioniert ähnlich wie ein Konstruktor in C++ oder Java.

Wenn man nun die `main.py` startet, dann erscheint auch das Fenster auf dem Bildschirm, das vorher nur im Qt Designer gelebt hat.

26.7. `loadUi()`

Die Umwandlung der UI-Datei in eine-Python-Datei durch pyuic5 ist nicht sehr aufwendig. Trotzdem ist es ein extra Schritt, an den der Entwickler denken muss. Verändert man zudem die UI-Datei, dann muss man die Umwandlung erneut anstoßen, sonst werden die Änderungen im Python-Programm nicht übernommen.

Es gibt noch eine andere Möglichkeit, über die PyQt direkt mit der UI-Datei arbeiten kann: `uci.loadUi()`. Damit verlässt man allerdings den puren Python-Weg und das Programm muss sich mit der externen UI-Datei herumschlagen. Python-IDEs wie Pycharm können mit einer UI-Datei zudem wenig anfangen und so etwa keine Auto vervollständigungen für Widget-Namen anbieten.

Der Code für die Lösung `loadUi()` ist ähnlich kompakt wie beim vorherigen Beispiel:

```
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5.uic import loadUi
import sys

class HelloworldProgramm(QMainWindow):
    def __init__(self, parent=None):
        super(HelloworldProgramm, self).__init__(parent)
        loadUi("helloworld.ui", self)

programm = QApplication(sys.argv)

fenster = HelloworldProgramm()
fenster.show()

sys.exit(programm.exec())
```

Zu Beginn wird die Funktion `loadUi` importiert:

```
from PyQt5.uic import loadUi
```

In der Klasse `HelloworldProgramm` wird anschließend die UI-Datei mit einem `loadUi("helloworld.ui", self)` geladen. Startet man nun dieses Programm, erscheint wieder das Hello-World-Fenster.

26.8. Passendes Widget

Sowohl bei der vorherigen Umwandlung durch pyuic5 als auch bei der Nutzung von `loadUi()` ist es wichtig, in der Klasse `HelloworldProgramm` auf das passende Widget zu verweisen. Im Qt Designer hat man vorher ein `QMainWindow` erstellt. Das `MainWindow` ist auch das Root-Element im XML-Baum der gespeicherten UI-Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>MainWindow</class>
    <widget name="MainWindow">
```

Daher wird die Klasse `HelloworldProgramm` immer als `QMainWindow`-Objekt definiert, egal welche Methode man verwendet, um die UI-Datei ins Python-Programm zu bekommen:

```
class HelloworldProgramm(QMainWindow):
```

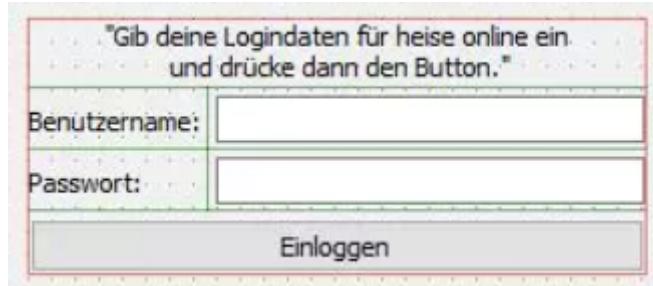


Figure 26.5.: So sollen die Widgets später im Gitter platziert werden.

26.9. Login-Fenster erstellen

Nun ist klar, wie man ein Fenster mit dem Qt Designer erstellt und wie man das Layout in einem Python-Programm nutzen kann. Aber nur ein Hello-World-Fenster ist doch etwas langweilig – selbst wenn es mit einer feschen Schriftart wie Comic Sans daherkommt.

Als Nächstes zeigen wir, wie man ein kleines Login-Fenster für heise online bastelt. Der Nutzer soll seinen Benutzernamen und sein Passwort für heise online eingeben und auf einen Login-Button klicken können. Das Programm sagt ihm dann, ob die Daten korrekt waren oder nicht. Eine Menüleiste rundet das Programm ab.

Dieses Login-Fenster haben wir schon per Hand mit Tkinter und mit PyQt gebastelt. So kann man den Code einfach vergleichen. Im PyQt-Beitrag gehen wir zudem stärker auf die Grundlagen von Qt ein.

Im Qt Designer startet man mit einem neuen Projekt und holt sich wieder ein MainWindow aus den Vorlagen – das war auch der Startpunkt beim handgecodeten PyQt-Fenster. Eine Statusleiste benötigt man für dieses Projekt nicht. Das Element können Sie mit einem Rechtsklick auf Statusbar in der Objektanzeige oben rechts löschen. Als Titel für das Fenster ist etwa "Login für heise online" passend.

26.10. Layout

Qt arbeitet mit Layouts, etwa vertikalen Layouts, horizontalen Layouts oder Gitterlayouts. Das sind quasi Konzepte, wie Widgets angeordnet sein sollen. In einem vertikalen Layout stehen sie nebeneinander, in einem Gitter werden sie nach Zeilen und Spalten in Zellen angeordnet.

Für das Login-Fenster eignet sich ein Gitterlayout. Ziehen Sie es aus dem Bereich Layout in das Fenster. Ein roter Kasten erscheint, in dem man nun Widgets platzieren kann. Ist bereits ein Widget im Layout-Kasten, ziehen Sie ein neues Widget an die Ränder des bestehenden Widgets, um es vertikal oder horizontal daneben zu platzieren. Grüne Linien zwischen den Widgets zeigen die Ränder der Gitterzellen an. Soll ein Widget etwa über zwei Spalten gehen, dann zieht man es einfach so lang, bis es in die Zelle daneben ragt. Qt Designer passt dann automatisch das Gitter an.

26.11. Widgets

Für den Login benötigt man sechs Widgets: drei Labels, zwei Eingabefelder und einen Button. Die Labels stehen im Bereich der Display Widgets; als Eingabefelder kommen Line Edits aus dem Bereich der Input Widgets hinzu; der Button ist ein Push Button im Button-Bereich. Per Drag-and-drop landen sie alle im neuen Gitterlayout, genauer gesagt im gridlayout, das im zentralen Widget sitzt, das im MainWindow sitzt.

```

Beschreibung-Text
Benutzername-Label Benutzername-Eingabefeld
Passwort-Label Passwort-Eingabefeld
Login-Button

```

Table 26.1.: Das Layout soll im Gitter so aussehen.

Das Layout soll im Gitter so aussehen:

Die Objektanzeige hat mittlerweile folgende Struktur:

```

1000 MainWindow
1001   Centralwidget
1002     gridLayout
1003       Label
1004       Label_2
1005       Label_3
1006       lineEdit
1007       lineEdit_2
1008       pushButton
1009       menubar

```

Das erste Label wird der Beschreibungstext, der über den Input-Feldern stehen soll. Hier ist das: "Gib deine Logindaten für heise online ein \n und drücke dann den Button." Diesen Text packt man in die Eigenschaft text des ersten Labels – \n fügt an dieser Stelle einen Zeilenumbruch ein. Wenn Sie den Text eingeben, erscheint ein kleiner Button mit drei Punkten neben dem Eingabefeld. Ein Klick darauf enthüllt einen kleinen visuellen Texteditor, mit dem Sie etwa Absätze einfügen, Teile fetten oder Zahlen hochstellen können – so können sich auch das \n sparen. Zurück in den Eigenschaften lässt sich der Text in den alignment-Optionen noch horizontal zentrieren.

Der Objektname ist mit "Label" noch etwas generisch und sollte geändert werden, um später Verwechslungen zu vermeiden. Dafür passt man im Bereich QObject der Eigenschaften den Wert objectName an und nennt das Label etwa "Beschreibungstext". Ähnlich geht man bei den anderen beiden Labels vor und füttet sie mit dem Text "Benutzername:" und "Passwort:". Auch hier sollte der Objektname angepasst werden, genauso wie bei den Line-Edit-Objekten, in denen der Nutzer seine Daten angibt – **Eingabe_Password** und **Eingabe_Benutzername** wären etwa passend. In der Passwort-Eingabe sollen zudem Sternchen erscheinen, wenn der Nutzer sein Passwort eingibt: Das regelt man über die Option echoMode im Bereich QLineEdit, indem man den Wert "Password" aus dem Dropdown-Menü wählt. Nun braucht der Button noch einen Text. Im Bereich **QAbstractButton** ändert man dafür den Wert von **text**, etwa zu "Einloggen".

Und damit stehen auch schon die Widgets im passenden Layout. Drückt man STRG+R dann kann man Texte in die Felder eintragen, die Labeltexte werden angezeigt und der Button ist drückbar. Nur tut er noch nichts.

26.12. Button klickbar machen

Damit der Button eine Funktion erhält, wechselt man ins Signale-und-Slots-Menü. Das geht am einfachsten über das siebte Icon von links in der Toolbar, das graue Feld mit dem schwarzen Pfeil, der auf ein hellblaues Feld zeigt.

Klicken Sie dann auf den Button im Fenster und halten Sie die linke Maustaste gedrückt. Anschließend ziehen Sie eine rote Linie in das Hauptfenster: Der Button erstellt so eine Verbindung mit dem MainWindow.

Nun erscheint ein Menü, um die Verbindung zu bearbeiten. Links stehen die möglichen Auslöser für den Button, etwa `clicked()`, `pressed()` oder `released()`. Für diesen Fall ist `clicked()` die richtige Wahl. Im Bereich `MainWindow` auf der rechten Seite des neuen Menüs muss man nun “Ändern...” auswählen und kann die Signale und Slots von `MainWindow` bearbeiten.

`MainWindow` benötigt einen neuen Slot, also klickt man auf den grünen Plus-Button, um einen hinzuzufügen. Anschließend gibt man den Namen der Funktion ein, die der Button-Klick später auslösen soll, etwa `button_geklickt()`. Nun bestätigt man den Vorgang mit einem “OK” und wählt im Verbindungs menü die neue Funktion aus.

Da der Qt Designer nicht an eine Programmiersprache gebunden ist, bietet er keinen Editor für die Funktion an. Was später genau nach einem Klick auf den Button passiert, muss man in der Python-Datei festlegen. Wie schon beim Hello-World-Programm erstellt man dafür eine Python-Datei, die das Fensterlayout aus dem Designer aufgreift: Entweder mit `pyuic5` oder `loadUi()`. Anschließend erstellt man in der Klasse des `MainWindows`s eine neue Funktion namens `button_geklickt()`:

```
def button_geklickt(self):
    self.benutzername = self.Eingabe_Benutzername.text()
    self.passwort = self.Eingabe_Passwort.text()
    self.fake_browser = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:94.0) Gecko/20100101 Firefox/94.0"}
    self.login_daten = {"username": self.benutzername, "password": self.passwort, "action": "/sso/login/login"}
    self.login = requests.Session().post(url="https://www.heise.de/sso/login/login", data=self.login_daten, headers=self.fake_browser)
    if "Der Benutzername oder das Passwort ist falsch." in self.login.text():
        self.Beschreibungstext.setText("Fehler: Der Benutzername \n oder das Passwort ist falsch.")
        self.Beschreibungstext.setStyleSheet("color: red;")
    else:
        self.Beschreibungstext.setText("Sie haben sich erfolgreich eingeloggt.")
        self.Beschreibungstext.setStyleSheet("color: green;")
```

Beim Code orientieren wir uns an der Funktion aus dem PyQt-Artikel. Als Erstes benötigen Sie die Eingaben des Nutzers, also seinen Benutzernamen und das Passwort. Diese Daten holen Sie mit `text()` aus den QLineEdits:

```
self.benutzername = self.Eingabe_Benutzername.text()
self.passwort = self.Eingabe_Passwort.text()
```

Dabei muss man darauf achten, dass man die Eingabefelder so anspricht, wie man sie auch im Qt Designer benannt hat, also `Eingabe_Benutzername` und `Eingabe_Passwort`.

26.13. User-Agent

Der User-Agent enthält Daten über den Browser und dem verwendeten System, was anschließend an die Website gesendet wird:

```
self.login_daten = {"username": self.benutzername, "password": self.passwort, "action": "/sso/login/login"}
```

Ihren eigenen User-Agent sehen Sie etwa auf <http://wieistmeinuseragent.de>.

26.14. Login-Daten

Die Daten für den Login speichern Sie in einem weiteren Dictionary:

```
import requests
...
self.login = requests.Session().post(url="https://www.heise.de/sso/login/login",
data=self.login_daten, headers=self.fake_browser)
```

Bei heise online loggt man sich über die Website <https://www.heise.de/sso/login/login> ein. Verantwortlich für den Login ist eine POST-Methode mit der Aktion `/sso/login/login`, daher wird auch sie im Dictionary festgelegt.

Der Login läuft dann über die Bibliothek `requests`, die Sie vorher im Import-Bereich importieren müssen:

```
if "Der Benutzername oder das Passwort ist falsch." in self.login.text:
    self.Beschreibungstext.setText("Fehler: Der Benutzername \n oder das
Passort ist falsch .")
    self.Beschreibungstext.setStyleSheet("color: red;")
else:
    self.Beschreibungstext.setText("Sie haben sich erfolgreich eingeloggt .")
    self.Beschreibungstext.setStyleSheet("color: green;")
```

In einer `Session()` werden Cookies gespeichert und Sie können später mit aktivem Login andere Seiten aufrufen. `post()` steht für die HTML-Methode POST, die `url` ist die heise-Login-Seite, `data` enthält die vorher festgelegten Login-Daten und in `headers` geben Sie den Fake-Browser als User-Agent mit.

Es gibt noch keinen visuellen Hinweis, ob der Login geklappt hat. Als Nächstes soll daher der Beschreibungstext geändert werden, entweder in eine Erfolgsmeldung oder in einen Hinweis, dass der Benutzername oder das Passwort falsch war.

26.15. Beschreibungstext ändern

Falls der Login nicht geklappt hat, erkennt man das, wenn in `self.login.text` die Zeile "Der Benutzername oder das Passwort ist falsch." auftaucht. Das nutzt man für eine If-Abfrage:

```
self.actionBeenden.triggered.connect(self.close)
```

`setText` ändert den Inhalt des Beschreibungstextes. Mit `setStyleSheet` ändert man die Farbe des Textes in Rot oder Grün. Auch hier muss man wieder darauf achten, dass der richtige Name des Beschreibungstext-Labels aus dem Designer angegeben wird.

26.16. Menü hinzufügen

Das Menü ist recht schnell angelegt, da eine Menüleiste schon besteht. Im Arbeitsbereich findet man oben, unter dem Fenstertitel, ein kleines graues Feld mit dem Text "Geben Sie Text ein". Dieser Platzhalter ist für das erste Menü gedacht. Nach einem Doppelklick darauf kann man einen Namen für das Menü vergeben, etwa "Datei". Anschließend möchte Qt Designer auch schon den ersten Menüpunkt haben. Wir nennen ihn in diesem Beispiel "Beenden", da ein Klick darauf das Programm beenden soll. Qt Designer erstellt eine neue Aktion namens `actionBeenden`, die auch im Aktionseditor aufgeführt ist. Dort kann man etwa die Tastenkürzel bearbeiten. Was passiert, wenn man auf den Menüeintrag klickt, muss man wieder in der Python-Datei festlegen, etwa in der Funktion `__init__`:

```
self.actionBeenden.triggered.connect(self.close)
```

`triggered.connect()` verbindet das Aktionsobjekt `actionBeenden` aus dem Designer mit einer konkreten Aktion, hier `self.close`, das einfach das Programm beendet.

26.17. Ausblick

Und das war's. Alle Widgets sind durch das passende Layout an ihrem richtigen Platz. Der Button tut genau das, was er tun soll, über kleines Menü kann der Nutzer das Programm beenden. Dank des Qt Designers ist die Gestaltung von der Logik getrennt und Sie können sich voll darauf konzentrieren, die Funktionen für Ihr Programm zu erstellen.

27. Programmieren mit Python: Schnittstellen entwickeln mit Pycharm und FastAPI

IDEs wie Pycharm machen das Leben für Programmierer einfacher. Wir zeigen, wie Sie in Pycharm mit FastAPI eine Python-REST-Schnittstelle erstellen.

Entwicklungsumgebungen (Integrated Development Environment, IDE) laufen auf den meisten Rechnern problemlos und bieten einen enormen Mehrwert durch die vielen Module, die sie mitbringen: Zum Beispiel die Integration von Debuggern, Module für das Refactoring, also das Umbenennen etwa von Variablennamen über Dateigrenzen hinweg, Test-Umgebungen oder Code-Revisions-Werkzeuge wie Git. Für viele IDEs existiert ein ganzes Universum an Erweiterungen, etwa für die Code-Analyse oder Virtualisierungs- beziehungsweise Cloud-Integration.

In diesem Artikel zeigen wir, wie man mit FastAPI eine Python-REST-Schnittstelle schnell und einfach erstellen kann. Dazu nutzen wir mit Pycharm eine Entwicklungsumgebung, die viele Funktionen für das Entwickeln mit Python zur Verfügung bereithält.

Wichtig für das Arbeiten mit Pycharm ist vor allem der Debugger, der es zur Laufzeit von Programmen ermöglicht, die Werte von Variablen auszulesen und somit die Qualität des Codes live zu begutachten. Und zwar, ohne den Code selbst mit `print()`-Statements zu verschönern. Mithilfe von Breakpoints können Sie den Ablauf des Programmes bei jedem Befehl stoppen, den Sie genauer betrachten wollen, etwa weil Sie auf der Suche nach einer bisher nicht beachteten Ausnahme sind.

Wenn Sie die Beispiele selbst nachvollziehen möchten, [laden Sie einfach den Quellcode in einer 7Z-Datei von unserem Server herunter](#), entpacken ihn und nutzen ihn in der IDE Ihrer Wahl – es muss nicht unbedingt Pycharm sein. In der IDE müssen Sie nur ein neues Projekt anlegen und den Code reinkopieren. Und danach die Datei `requirements.txt` im Ordner `data` als Grundlage für die Python-Paket-Installationen verwenden.

27.1. Warum Pycharm?

Pycharm ist ein guter Kompromiss aus standardmäßigem Funktionsumfang und flüssiger Bedienung. Im Vergleich dazu legt etwa die IDE Spyder als Teil der Anaconda-Distribution den Schwerpunkt auf Data Science. Sie bringt mehr Module mit, um vor allem wissenschaftliche Daten zu bearbeiten.

[Pycharm](#) gibt es in zwei Ausführungen: der kostenlosen Community Edition sowie der kostenpflichtigen Professional Edition. Die Professional Edition bietet vielfältigere Möglichkeiten wie Code-Coverage (Testabdeckung) oder Werkzeuge für die Webentwicklung mit diversen Frameworks wie Django, Flask oder aktuellen JavaScript-Entwicklungen.

Pycharm installiert man etwa unter Windows mit einer heruntergeladenen EXE-Datei. Weitere Möglichkeiten, auch für andere Betriebssysteme, [finden Sie auf der Website der Entwicklerfirma JetBrains](#).

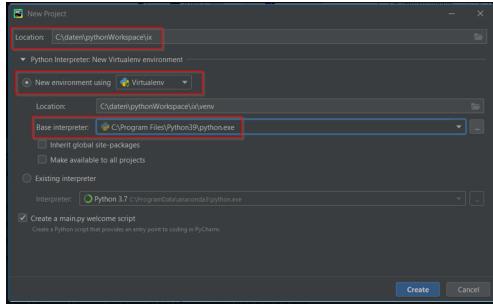


Figure 27.1.: Location.

27.2. Warum FastAPI?

FastAPI ist für den Umgang mit REST-Schnittstellen eine sehr gute Wahl. Es ähnelt als kleines Framework Flask: Man muss etwa für das Zusammenspiel mit Datenbanken weitere Module selbst installieren. Damit grenzt es sich von einem Framework wie Django ab, das von Haus aus schon viele Komponenten mitbringt.

Daher fokussiert sich FastAPI eher auf seinen Kern: nützliche Komponenten rund um REST-Schnittstellen. Dazu gehört [das automatische Erstellen von Dokumentation via OpenAPI](#) und somit auch die Möglichkeit, direkt und interaktiv testen zu können. Weiterhin kann FastAPI [eingehende Anfragen nativ asynchron verarbeiten](#), um etwa Wartezeiten auf Datenbank- oder Festplattenzugriffe zu minimieren.

Zusätzlich beherrscht FastAPI Type Hinting - es kann also beim Aufruf von Funktionen direkt den Typ einer Variablen vorgeben und so falsche Eingaben standardmäßig erkennen. Type Hinting ist übrigens mit der Python-Version 3.9 im Standard angelangt. Zudem kann FastAPI auch mit WebSockets und GraphQL umgehen. Noch mehr Details und eine sehr gute Dokumentation finden Sie [auf der Webseite des FastAPI-Projekts](#).

27.3. Projekt in Pycharm einrichten

Beim ersten Starten von Pycharm öffnet sich ein Dialogfenster, in dem Sie auf "New Project" klicken. Anschließend erscheint eine Konfigurationsmaske, in der Sie von oben nach unten folgendes festlegen: den Speicherort Ihres Projektes, also die "Location" sowie die Detaileinstellungen des Python Interpreters:

Während Sie für den Speicherort wie gewohnt über das Ordnersymbol rechts einen Ordner Ihrer Wahl festlegen können, treffen Sie bei den Einstellungen zum Interpreter wichtige Entscheidungen für Ihr Projekt. Zum einen definieren Sie, ob und welche virtuelle Python-Umgebung Sie nutzen möchten. Damit die Pakete einzelner Projekte auch nur diesen Umgebungen zur Verfügung stehen, sollten Sie in jedem Fall eine virtuelle Umgebung auswählen. Für dieses Projekt nutzen wir Virtualenv, weiterhin kann man noch zwischen Conda und Pipenv wählen.

Den Ort für die virtuelle Umgebung können Sie übernehmen und danach den Python Interpreter, also den "Base Interpreter", wählen, falls Sie mehrere Python-Versionen installiert haben. Das ist etwa dann sinnvoll, wenn Sie ältere Projekte pflegen und dort mit Python 2.7 arbeiten müssen.

Wenn Sie den untersten Haken nicht abwählen, erstellt Pycharm eine erste Python-Datei im neuen Projekt. Ihr neues Projekt sieht dann wie folgt aus:

In dem Hauptordner "ix" befindet sich am Anfang nur der Ordner für die virtuelle Umgebung. Wenn Sie auf den Hauptordner mit der rechten Maustaste klicken und "New" auswählen, können Sie nun weitere Ordner oder Dateien anlegen. Wir haben das für die Datei `ix_fastapi.py` schon gemacht und die ersten Zeilen in die Datei kopiert:

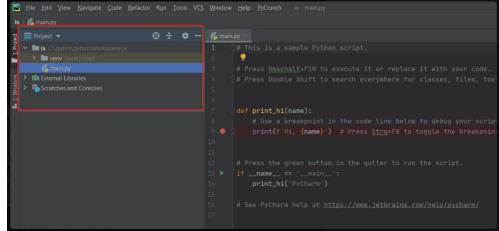


Figure 27.2.: Project.

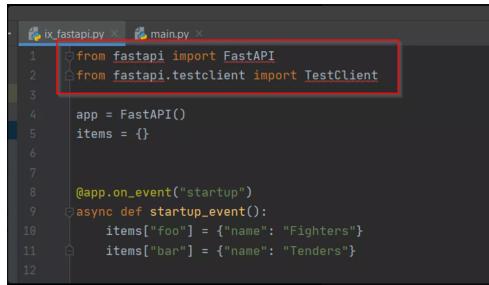


Figure 27.3.: Import.

27.4. Pakete installieren

Dabei fällt auf, dass Pycharm bei den Paketimporten beide Zeilen rot unterstreicht. Das bedeutet, dass in unserer virtuellen Umgebung das Paket fastapi noch nicht bekannt ist und wir es importieren oder installieren müssen. Gehen Sie oben links auf "File" und anschließend auf "Settings". Dort sehen Sie die Übersicht der Einstellungen: Wichtig sind die Informationen zum "Project: ix" und dort zum "Python Interpreter", die Sie nacheinander anklicken. Somit gelangen Sie zu der Übersicht installierter Python-Pakete oder "Packages". Um weitere Pakete hinzuzufügen, klicken Sie auf das Plus-Zeichen oben rechts und es öffnet sich eine Übersicht aller verfügbaren Pakete. In das anfangs leere Suchfeld oben links geben Sie "fastapi" neben der Lupe ein, wählen dann aus den angezeigten Paketen das richtige aus und klicken dann auf "Install Package". Neben dem Paket erscheint nun ein Hinweis darauf, dass es installiert wird und nach dem Abschluss des Installierens erscheint ein Hinweis unten links.

Wenn Sie den aktuellen Dialog schließen, sehen Sie in der Paketübersicht jetzt neben "fastapi" auch "pydantic" und "starlette". Das zeigt, dass fastapi andere Paket benötigt, die automatisch mitinstalliert werden.

Da FastAPI nicht wie Flask mit einem eigenen Server kommt, installieren Sie wie vorher den Server Uvicorn:

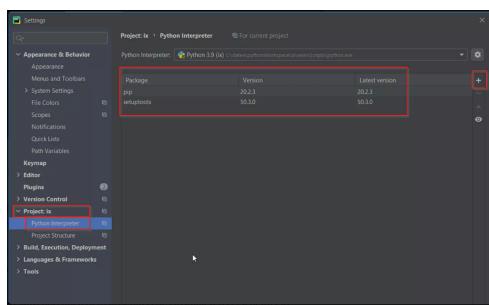


Figure 27.4.: Packages.

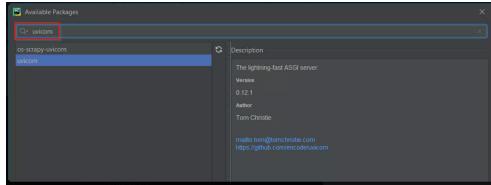


Figure 27.5.: Unicorn.

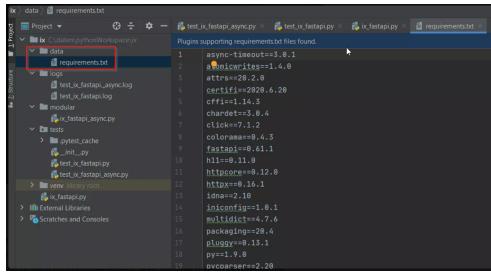


Figure 27.6.: requirements.txt

Wenn Sie später alle installierten Pakete in einer Datei speichern wollen, um das Projekt etwa in einer anderen Umgebung aufzusetzen oder nur zu sichern, können Sie im Pycharm-Terminal unten einfach den Befehl `pip freeze > data/requirements.txt` ausführen. Dieser Befehl erstellt in der Datei `data/requirements.txt` eine Übersicht der Module, die dann so aussieht:

27.5. Tipps und Tricks mit Pycharm

Fehlermeldungen: Oben rechts im Editor-Fenster zeigt Pycharm vorhandene Fehler und Warnungen an, falls vorhanden – oder ein grüner Haken, wenn alles passt. Pycharm sucht nach Programmierfehlern, aber auch nach Verstößen gegen die Python-Programmerrichtlinien wie PEP. Wenn Sie auf jetzt auf die Zahl der Fehler klicken, öffnet sich unten der "Problems Editor" und listet alle Probleme inklusive Beschreibung auf. Wenn Sie von dort auf einen Eintrag klicken, springen Sie direkt zum Fehler.

Debuggen: Im Debugmodus kann man sich unter anderem Werte von Variablen zur Laufzeit anzeigen lassen und so das Programm prüfen. Nutzen Sie Umschalt+F9 oder klicken auf das Käfer-Symbol rechts neben dem grünen Pfeil, um den Debugger zu starten.

Code Folding: Sie können komfortabel Ihre Methoden oder andere Elemente einklappen, um eine Übersicht über Ihren Text zu erhalten. Drücken Sie die Tastenkombination STRG+Umschalt+Minustaste und alle einklappbaren Textteile werden zusammengefaltet. Mit STRG+Umschalt+Plustaste können Sie alles wieder aufklappen.

Logging: Wenn Sie die Ausgaben eines Programms sichern wollen, können Sie Log-Dateien anlegen, in die Pycharm die normalen Konsoleausgaben schreibt. Dafür gehen Sie im Menü auf "Run" und dann auf "Edit Configurations". Im nächsten Dialog wählen Sie links Ihr zu loggendes Programm aus und klicken rechts auf den Reiter "Logs". Nun können Sie jetzt den Speicherort der Log-Datei festlegen

Versionsvergleich: Den aktuellen Stand einer Datei können Sie mit früheren lokalen Versionen vergleichen. Machen Sie dazu einen Rechtsklick auf den Dateinamen in der Projektübersicht, klicken auf "Local History" und wählen "Show History" aus. Im nächsten Fenster ist rechts die aktuelle Version zu sehen, links eine vorherige. Wenn Sie die Uhrzeit im linken Fenster anklicken, legen Sie fest, welche Version Sie

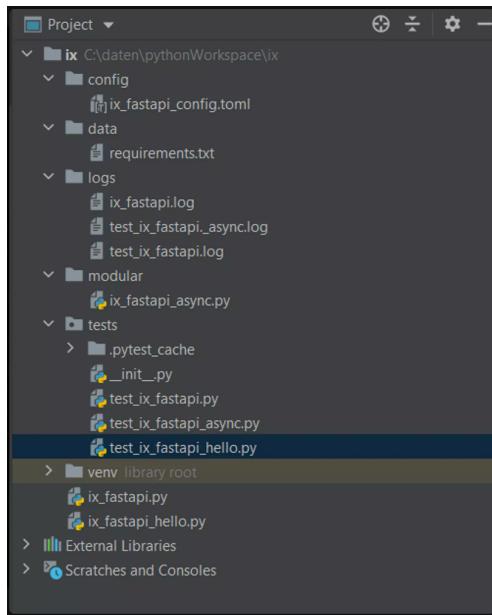


Figure 27.7.: Aufbau des Projekts

betrachten wollen. Außerdem hebt Pycharm die Unterschiede zwischen den beiden Dateien hervor.

27.6. Daten pflegen mit FastAPI

In modernen webbasierten Anwendungen trennt man oft die Frameworks, die Inhalte darstellen, von den Frameworks, die Daten bereitstellen. Außerdem tauschen Unternehmen Daten direkt über Web-Schnittstellen aus. Daher benötigt man leistungsfähige Programme, die den kompletten Lebenszyklus dieser Daten abdecken. Dieser Zyklus wird auch CRUD genannt: Create, Read, Update, Delete. FastAPI bietet diesen Umfang inklusive Sicherheit, asynchroner Verarbeitung und vielem mehr an.

In diesem Beispiel geht es darum, eine Liste von Gegenständen zu verwalten: neue Gegenstände anzulegen, Details eines bestimmten oder aller Gegenstände abzufragen, diese Details zu verändern oder Gegenstände ganz zu löschen. Mit einer Liste ist das Projekt einfacher nachzuvollziehen, eine Datenbank müssten Sie erst aufsetzen.

27.7. Aufbau des Projekts

Dieses Schnittstellenprojekt ist ein typisches Python-Projekt mit Konfiguration, Daten und Tests. Zudem gibt es noch den Ordner modular, in den wir einzelne Teile der REST-Schnittstelle ausgelagert haben, um den Code besser verwalten zu können. In der Datei `ix_fastapi.py` ist der Code der Schnittstelle gespeichert.

27.8. Hello World mit FastAPI

Fas Hello-World-Projekt in der Datei `ix_fastapi_hello.py` gibt einen Überblick über den kompletten Entwicklungszyklus in Pycharm.

Oben erscheinen die notwendigen Importe. Mit `app = FastAPI()` wird das Applikations-Objekt app als FastAPI-Instanz erzeugt. Darunter wird per `@app` auf app

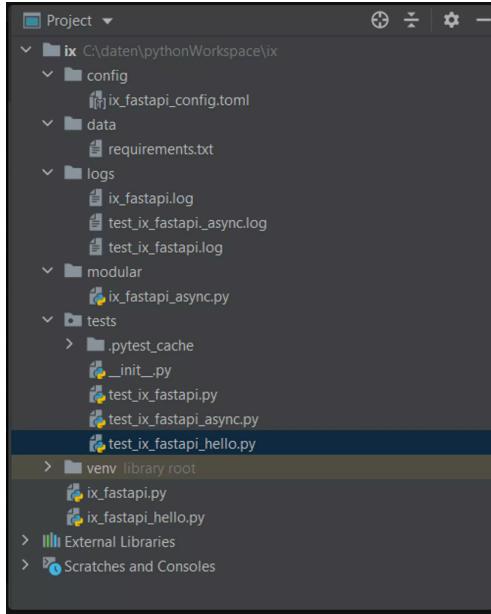


Figure 27.8.: Aufbau des Projekts

```

1  from fastapi import FastAPI
2  import toml
3  import uvicorn
4
5
6  app = FastAPI()
7
8
9  @app.get("/")
10 def hello():
11     return {"hello": "world"}
12
13
14 if __name__ == "__main__":
15     config = toml.load("./config/ix_fastapi_config.toml")
16     env = config['SETUP']['env']
17     host = config['PROJECT'][env]['host']
18     port = config['PROJECT'][env]['port']
19     protocol = config['PROJECT'][env]['protocol']
20
21     print(f"environment set to {env}, server to {protocol}://{host}:{port}")
22
23     uvicorn.run(app, host=host, port=port)
24
25

```

Figure 27.9.: ‘Hello World’ in JSON-Manier

referenziert. Außerdem stellen Sie unter dem Pfad /hello auf dem Server eine Schnittstelle zur Verfügung, die per GET-Request abgefragt werden kann. Abgesprochen wird diese Schnittstelle dann unter <http://{server-ip}:{port}/hello>.

Nun verknüpft man die Python-Methode `hello` mit dieser Schnittstelle. Da die Klammern leer sind, werden keine Parameter erwartet. Bei FastAPI werden standardmäßig JSON-Strings zurückgegeben, sodass sie nicht erst wie bei Flask in einem zusätzlichen Schritt erzeugt werden müssen. In diesem Fall gibt die Methode ein “Hello World” in JSON-Manier zurück.

In der Python-Main-Methode starten Sie den Server. Doch zuvor müssen Sie noch relevante Informationen wie Setupdaten, Ports oder Hosts aus der Konfigurationsdatei `ix_fastapi.config.toml` auslesen.

Weitere Informationen zum Konfigurieren mit TOML finden Sie im Netz.

Hinweis: Auch wenn dieses Projekt eine Konfiguration für eine Produktionsumgebung aufweist, so sollte sie nicht ohne entsprechende weitere Sicherheitseinstellungen produktiv eingesetzt werden.

```

1
2     [SETUP]
3         env = "DEV"
4         #env = "INT"
5         #env = "PROD"
6
7
8     [PROJECT]
9         [PROJECT.DEV]
10            host = '0.0.0.0'
11            port = 5000
12            protocol = 'http'
13            [PROJECT.INT]
14            host = 'int.test_server.de'
15            port = 5001
16            protocol = 'https'
17            [PROJECT.PROD]
18            host = 'rest.ix_fastapi_server.de'
19            port = 4999
20            protocol = 'https'
21

```

Figure 27.10.: Datei `test_ix_fastapi_hello.py`

27.9. Kein Coden ohne Testen

Um sicherzugehen, dass unser Code auch wie erwartet reagiert, erstellen Sie einen Unit-Test in der Datei `test_ix_fastapi_hello.py`:

Unit-Tests sind die erste Stufe im Entwicklungsprozess, weitere Test wie Integrations-, System- und Usability-Tests sollten in jedem Fall zusätzlich erfolgen. Oben finden Sie wieder die notwendigen Paketimporte. Dabei ist es wichtig, unter anderem die von der zu testenden Python-Datei `ix_fastapi_hello.py` erstellte Anwendung `app` zu importieren.

Im nächsten Schritt erzeugen Sie den Test-Client für die Anwendung und den Unit-Test. Zuerst erstellen Sie eine Klasse als Instanz eines Testcases, in der Sie dann die eigentlichen Tests als Funktionen packen. Dabei muss jeder Test, genau wie der Datei- und der Klassename, ebenfalls mit `test_` beginnen, da ansonsten die Testwerkzeuge diese Tests nicht finden. Außerdem müssen alle Tests im Ordner `tests` angelegt werden. In Zeile 9 wird über `client` ein GET-Request mit dem Pfad `/hello` abgesetzt und die Antwort von FastAPI der Variablen `response` übergeben. In den drei folgenden Zeilen erfolgt dann das Testen der Antwort: `assert` zeigt immer eine Prüfung an, darauf folgt die Bedingung, welcher ein Aspekt der Antwort genügen muss. Etwa dass der Status-Code der FastAPI-Antwort den Wert 200 haben muss, das entspricht einem OK. Zeile 11 testet, wie der JSON-Anteil aussehen muss, Zeile 12 wie er nicht aussehen darf.

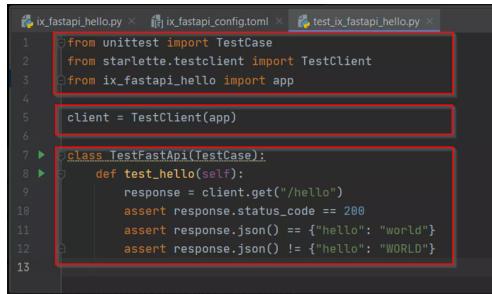
Beim Testen ist es immer hilfreich, nicht nur die jeweiligen richtigen Fälle zu testen, sondern auch die negativen. Bei numerischen Variablen ist es wichtig, immer die Randfälle zu testen. Wenn eine Variable nur einen Wert von 0 bis 5 annehmen darf, sollten Sie Tests für negative Zahlen kleiner als -1, 0, 5, 6 sowie eine Zahl größer als 6 verfassen. So sind Sie auf der sicheren Seite.

Um den Test nun zu starten, machen Sie einen Rechtsklick auf die Datei und wählen "Run ,Unitests in test_ix..." aus:

Die Tests laufen dann links unten in Pycharm. In diesem Fall ist es nur einer. Und da überall nur grüne Haken zu sehen sind, ist der Test fehlerfrei durchgelaufen. Sollten Sie keine Details sehen, klicken Sie auf den hellgrauen Pfeil im dunkelgrauen Kasten neben dem Run-Button und sie werden eingeblendet:

Ein Vorteil von FastAPI besteht darin, dass Sie Ihre Schnittstellen interaktiv in einem Browser testen können. Starten Sie dafür Ihren Server, öffnen einen Browser und geben dort die beim Start des Servers angezeigte URL samt Port sowie den Pfad `/docs` ein. In unserem Fall `127.0.0.1:5000/docs`. Bitte beachten Sie, dass die IP 0.0.0.0 bedeutet, dass auch andere Rechner in dem Netzwerk auf Ihren Rechner zugreifen können.

Im Browser erhalten Sie zuerst einen Überblick über alle Schnittstellen, hier nur



```

1 from unittest import TestCase
2 from starlette.testclient import TestClient
3 from ix_fastapi_hello import app
4
5 client = TestClient(app)
6
7 class TestFastApi(TestCase):
8     def test_hello(self):
9         response = client.get("/hello")
10        assert response.status_code == 200
11        assert response.json() == {"hello": "world"}
12        assert response.json() != {"hello": "WORLD"}
13

```

Figure 27.11.: “Run ,Unittests in test_ix...”

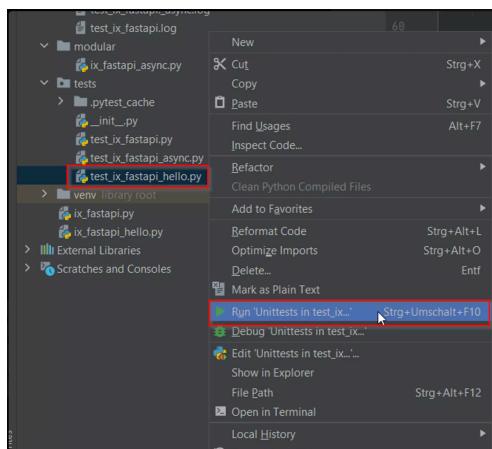


Figure 27.12.: Run

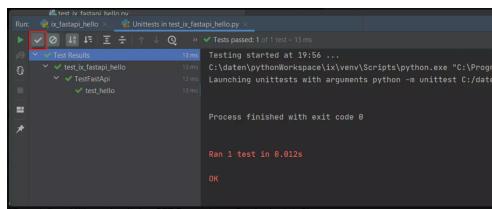


Figure 27.13.: Klicken in einem Browser

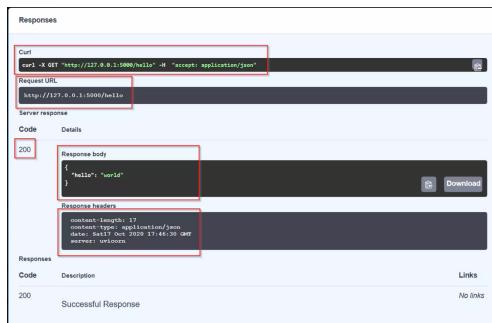


Figure 27.14.: Datei ix_fastapi.py

[/hello](#). Klicken Sie auf GET und Sie gelangen zu den Details. Klicken Sie nun auf "Try it out". Da für unser Beispiel keine Parameter verlangt werden, klicken Sie direkt auf "Execute" und führen damit den GET-Request aus. Das bedeutet, Sie senden die Abfrage - analog zum Klicken in einem Browser:

Oben steht unter "Curl" der Text, mit dem Sie dieselbe Anfrage von einer Kommandozeile absetzen können. Oder Sie können den Text von "Request URL" [mit Postman nutzen](#). Postman ist eine Anwendung, mit der Sie REST-Schnittstellen entwickeln und testen.

Darunter steht die Antwort des Servers, die "Server response": als erstes der Status-Code und daneben die Details, also die Header-Informationen sowie die eigentliche Antwort, der "Response body".

27.10. Gegenstände mit FastAPI verwalten

Nun wollen wir eine Liste von Gegenständen über das API verwalten. Die Gegenstände nennen wir Items. Sie verfügen jeweils über zwei Details, nämlich eine `item_id` und ihren Namen in `name`.

Die Datei `ix_fastapi.py` ist dabei genauso aufgebaut wie die zuvor beschriebene Hello-World-Beispieldatei:

Nach den notwendigen Paketen folgt wieder die Instanziierung der FastAPI-Anwendung in Zeile 12. Und in Zeile 14 definiert man die Liste der Gegenstände – zum Start sind es zwei.

FastAPI bietet via OpenAPI die Möglichkeit, Nutzern direkt anzeigen zu können, welche Struktur die Antwort einer Schnittstelle hat. Diese Strukturen werden als Instanzen des BaseModels erzeugt, hier heißen sie `Item` und `Message`.

In der ersten Schnittstelle rufen Sie Details zu einem bestimmten Item über seine `item_id` auf. Das ist wieder ein GET-Request, somit steht `@app.get` vor der Funktion. Der Pfad besteht aus dem Namen `item` und der zu übergebenden ID, die so dem Funktionsaufruf von `read_item` übergeben wird. Dabei wird direkt geprüft, ob es sich bei der ID um einen numerischen Wert handelt, ansonsten wird ein Werte-Fehler zurückgegeben. In der Funktion selbst wird in Zeile 42 zuerst mittels eines Tests

```

1  from fastapi import FastAPI, HTTPException
2  from fastapi.encoders import jsonable_encoder
3  from fastapi.responses import JSONResponse
4  from pydantic import BaseModel
5  import tomllib
6  from typing import List
7  import uvicorn
8
9  from modular import ix_fastapi_async
10
11 app = FastAPI()
12
13 items = [{"item_id": 1, "name": "Bar"}, {"item_id": 2, "name": "Baz"}]
14
15
16 class Item(BaseModel):
17     item_id: int
18     name: str
19
20 class Message(BaseModel):
21     message: str
22
23

```

Figure 27.15.: ie `Item` und `Message`

```

34     def get_entry_from_list(item_list, entry_id):
35         entry = [i for i, j in enumerate(item_list) if j.get('item_id') == entry_id]
36         return entry[0]
37
38
39     @app.get("/item/{item_id}", response_model=Item)
40     async def read_item(item_id: int):
41         try:
42             entry = get_entry_from_list(items, item_id)
43             json_compatible_item_data = jsonable_encoder(entry)
44             return JSONResponse(content=json_compatible_item_data)
45         except IndexError as ie:
46             raise HTTPException(status_code=404, detail=f"Item with item_id {item_id} not found")
47
48

```

Figure 27.16.: Listeneinträge zurückgeben

(`get_entry_from_list`) geprüft, ob die ID schon in der Liste enthalten ist. Falls ja, wird der Eintrag in Zeile 43 in einen validen JSON-Ausdruck überführt, der dann als JSON-Antwort zurückgegeben wird (Zeile 44).

Für die Antwort ist auch schon ein Modell vorgegeben, das `responseModel` in Zeile 39. Wenn man etwa Item 1 abfragt, gibt es `{'item_id': 1, 'name': 'Bar'}` zurück. Falls das Item nicht vorhanden ist, wird zur Laufzeit des Programms ein `IndexError` erzeugt, den die Ausnahmebehandlung in Zeile 45 auffängt. In Zeile 46 wird dann eine HTTP-Ausnahme mit dem Fehlercode 404 erzeugt sowie einer Fehlerbeschreibung, die dann als Antwort auf den GET-Request gesendet wird.

27.11. Listeneinträge zurückgeben

In der zweiten Schnittstelle werden auf nach dem Aufruf des Pfades `/items` alle Listeneinträge zurückgegeben. Das Antwortmodell besteht in diesem Fall aus einer Liste der Items.

Mit der dritten Aufrufmöglichkeit schicken Sie Daten, um ein neues Item anzulegen. Daher muss das ein POST-Request sein und in Zeile 64 heißt der Verweis auf die Anwendung `@app.post`. Diesmal werden neben der `item_id` noch der Name übergeben, weshalb die Funktion in Zeile 65 beide Parameter erfordert. Innerhalb der Funktion wird zuerst geprüft, ob unter der `item_id` schon ein Eintrag in der Liste existiert und falls ja, wieder eine Ausnahme in Zeile 75 und 76 erzeugt. Falls nicht, wird der Item an die Liste angehängt und zur Bestätigung zurückgeschickt.

Um ein bestehendes Item zu ändern, muss ein PUT-Request an etwa den Pfad `/item/1/neuer_name` geschickt werden. Wenn die ID vorhanden ist, wird in Zeile 76 der Name neu gesetzt und danach zurückgegeben.

```

49     @app.get("/items", response_model=List[Item])
50     async def read_items():
51         json_compatible_item_data = jsonable_encoder(items)
52         return JSONResponse(content=json_compatible_item_data)
53
54

```

Figure 27.17.: Das Antwortmodell besteht in diesem Fall aus einer Liste der Items.

```

53     # check entry in list((item_list, entry_id))
54     entry = [i for i, _ in enumerate(item_list) if i.get('item_id') == entry_id]
55 
56     if entry:
57         return True
58     else:
59         return False
60 
61 
62 @app.post("/item/{item_id}/{name}", response_model=Item)
63 async def post_item(item_id: int, name: str):
64     # check if item_id already exists
65     try:
66         entry_exists = check_entry_in_listitems(item_id)
67 
68         if not entry_exists:
69             items.append(item(item_id, name))
70             entry = get_entry_from_list(items, item_id)
71             json_compatible_item_data = jsonable_encoder(entry)
72             return JSONResponse(content=json_compatible_item_data)
73         else:
74             raise HTTPException(status_code=409, detail=f"item with item_id {item_id} already exists")
75 
76     except IndexError as ie:
77         raise HTTPException(status_code=404, detail=f"item with item_id {item_id} not found")
78 
```

Figure 27.18.: zur Bestätigung zurückgeschickt

```

72     @app.put("/item/{item_id}/{name}", response_model=Item)
73     async def put_item(item_id: int, name: str):
74         try:
75             entry = get_entry_from_list(items, item_id)
76             items[item_id] = name
77             json_compatible_item_data = jsonable_encoder(items[entry])
78             return JSONResponse(content=json_compatible_item_data)
79         except IndexError as ie:
80             raise HTTPException(status_code=404, detail=f"item with item_id {item_id} not found")
81 
```

Figure 27.19.: Name neu gesetzt und danach zurückgegeben

Und falls ein Eintrag gelöscht werden soll, muss ein DELETE-Request an /item/1 geschickt werden. Nach dem Löschen wird eine Bestätigung in Form einer Message (Zeile 82) gesendet.

27.12. Daten asynchron verarbeiten

FastAPI bietet es an, den Code zu entzerren und teilweise in andere Dateien auszulagern. Das macht Sinn, wenn es sich um viele Schnittstellen handelt und eine Datei nur die Schnittstellen verwaltet. Dadurch können Entwickler ihre Schnittstellen unabhängig von den anderen pflegen und es kommt zu weniger Fehlern.

In unserem Beispiel haben wir die asynchronen Schnittstellen in eine eigene Datei ausgelagert. Deshalb muss bei den Importen zuerst die Datei bekanntgegeben werden – hier wird die Datei `ix_fastapi_async.py` aus dem Unterverzeichnis modular eingetragen. Und in Zeile 94 wird dann der Anwendung ein `Router` für diese Datei hinzugefügt.

In dieser Datei wird zuerst zu Vergleichszwecken in Zeile 14 eine normale Zählfunktion implementiert.

Wenn der Pfad `/counter_normal` über einen GET-Request aufgerufen wird, wird in der entsprechenden Funktion dreimal die Unter-Funktion `count()` aufgerufen, die aus zwei Schreibbefehlen und einem Warten für eine Sekunde besteht. Von allem wird die benötigte Zeit in den Zeilen 15 und 19 erfasst, welche zurückgeschickt wird. Da die interne Verarbeitung nacheinander erfolgt, können die einzelnen `count()`-Aufrufe nur nacheinander abgearbeitet werden.

In der zweiten Schnittstelle der ausgelagerten Datei wird dasselbe Ziel erreicht – diesmal aber asynchron. Dafür muss dieser Aufruf Teil des laufenden Loops werden (Zeile 37). Denn der Server, der auf Anfragen wartet, ist selbst ein Loop, weshalb kein neuer Loop gestartet werden kann. Anschließend übergibt man an den erweiterten Loop in Zeile

```

89     @app.delete("/item/{item_id}", response_model=Message)
90     async def delete_item(item_id: int):
91         try:
92             entry = get_entry_from_list(items, item_id)
93             items.pop(entry)
94             return {"message": f"item with item_id {item_id} deleted from items"}
95         except IndexError as ie:
96             raise HTTPException(status_code=404, detail=f"item with item_id {item_id} not found")
97 
```

Figure 27.20.: Schnittstellen unabhängig von den anderen pflegen

```

91 |     from modular import ix_fastapi_async
92 |
93 |     app.include_router(ix_fastapi_async.router)
94 |
95 |

```

Figure 27.21.: Router

```

1  import asyncio
2  import time
3  from fastapi import APIRouter
4
5  router = APIRouter()
6
7  # test of synchronous behaviour
8  def count():
9      print("One")
10     time.sleep(1)
11     print("Two")
12
13 @router.get("/counter_normal")
14 async def counter_normal():
15     s = time.perf_counter()
16     for _ in range(3):
17         count()
18
19     elapsed = time.perf_counter() - s
20
21     return {"counter": "normal counter", "time": elapsed}
22

```

Figure 27.22.: einzelnen `count()`-Aufrufe

42 eine Aufgabe und wartet in Zeile 43 auf deren Ergebnis. Diese Aufgabe sammelt ein, was asynchron erzeugt wurde: dreimal der Aufruf des asynchronen Warte-Counters

27.13. Testen, testen testen

Die Tests selbst folgen immer demselben Prinzip – zuerst wird der entsprechende Request abgeschickt und die Antwort danach auf ihren Status-Code sowie ein- oder zweimal auf ihren Inhalt geprüft. Die Tests sind nummeriert, damit sie auch genau in dieser Reihenfolge ausgeführt werden. Offensichtlich schauen die Testclients nach den Namen der Tests und spulen sie alphabetisch sortiert ab.

Als erstes führen wir die `test_ix_fastapi`-Tests als Unit-Tests aus. Zuerst wird ein Item hinzugefügt und danach dasselbe mit derselben Anfrage noch einmal probiert – und schlägt fehl. Das war zu erwarten.

Beim sechsten Test hat der Fehlerteufel zugeschlagen und ID und Name vertauscht – die Rückgabe meldet den Fehler, denn die Item-ID ist kein Integer. Test sieben zeigt, dass die Liste nun drei Einträge enthält.

Im achten Test wird ein Item erfolgreich geändert, im neunten aber nicht, da die Item-ID 4 nicht existiert. Test 10 beschreibt die aktualisierte Liste. In Test Nr. 11

```

1  import asyncio
2  import time
3  from fastapi import APIRouter
4
5  router = APIRouter()
6
7  # test of synchronous behaviour
8  def count():
9      print("One")
10     time.sleep(1)
11     print("Two")
12
13 @router.get("/counter_normal")
14 async def counter_normal():
15     s = time.perf_counter()
16     for _ in range(3):
17         count()
18
19     elapsed = time.perf_counter() - s
20
21     return {"counter": "normal counter", "time": elapsed}
22

```

Figure 27.23.: Testclients

```

1 # test of asynchronous behaviour
2
3     @async def async_count():
4         print("one")
5         await asyncio.sleep(1)
6         print("two")
7
8     @async def async_count_starter():
9         await asyncio.gather(async_count(), async_count(), async_count())
10
11 @router.get("/counters-async")
12     async def counter_async():
13         s = time.perf_counter()
14
15         try:
16             loop = asyncio.get_running_loop()
17         except RuntimeError:
18             loop = None
19
20         if loop and loop.is_running():
21             tsk = loop.create_task(async_count_starter())
22             await tsk
23
24         elapsed = time.perf_counter() - s
25
26         return {"counter": "async counter", "time": elapsed}
27
28

```

Figure 27.24.: Liste nun drei Einträge enthält.

```

1 from unittest import TestCase
2
3 from starlette.testclient import TestClient
4 from ix_fastapi import app
5
6 client = TestClient(app)
7
8 class TestFastApi(TestCase):
9     def test_01_create_item(self):
10         response = client.post('/item/1')
11         assert response.status_code == 200
12         assert response.json() == {'item_id': 1, 'name': 'Bar'}
13
14     def test_02_read_item_negative(self):
15         response = client.get('/item/Bar')
16         assert response.status_code == 404
17         assert response.json() == {'detail': 'Item not found'}
18         assert response.json() == {'detail': [{"loc": ["path", "item_id"], "msg": "value is not a valid integer",
19                                         "type": "type_error.integer"}]}
20
21     def test_03_read_items(self):
22         response = client.get('/items')
23         assert response.status_code == 200
24         assert response.json() == [{"item_id": 1, "name": "Bar"}, {"item_id": 2, "name": "Bar"}]
25
26

```

Figure 27.25.: “Run, ’Unitests in test_ix...’”

wird das Item mit der ID 3 gelöscht, das dann im zwölften Test nicht noch einmal gelöscht werden kann und einen Fehler erzeugt. Im finalen Test 13 wird noch einmal die Liste geprüft, die nun wieder der ursprünglichen gleicht.

Um die zuvor beschriebenen Tests laufen zu lassen, machen Sie einen Rechtsklick auf `test_ix_fastapi.py` im Ordner `tests` und klicken danach auf “Run, ’Unitests in `test_ix...`’”:

Wieder sind alle Tests erfolgreich durchgelaufen und rechts neben den Tests sehen Sie die Durchlaufzeit. Wenn Sie wirklich zuerst die Tests und dann den Code erstellen, läuft anfangs kein Test durch und alles ist rot. Sie müssen sich also bemühen, alles schnell grün zu bekommen. Das motiviert ordentlich.

Die asynchronen Tests müssen Sie als PyTest anlegen: Gehen Sie dazu noch einmal über “Run/Debug Configurations” in die Maske und klicken dort auf das Plus-Zeichen oben links. Danach klicken Sie auf den Eintrag “pytest” in der Rubrik “Python tests”, fügen die markierten Einträge ein und speichern:

Jetzt starten Sie die Tests wie zuvor und sehen folgendes Ergebnis:

Die asynchronen Tests laufen nur etwas mehr als eine Sekunde, während die synchronen 3 oder 9 Sekunden benötigen. Beim ersten Test wird der normale Counter aufgerufen,

```

1 def test_08_put_item(self):
2     response = client.put('/item/3/Now')
3     assert response.status_code == 200
4     assert response.json() == {'item_id': 3, 'name': 'Now'}
5
6 def test_09_put_item_negative(self):
7     response = client.put('/item/4/Now')
8     assert response.status_code == 404
9     assert response.json() == {'detail': 'Item with item_id 4 not found'}
10
11 def test_10_read_items_after_put(self):
12     response = client.get('/items')
13     assert response.status_code == 200
14     assert response.json() == [{"item_id": 1, "name": "Bar"}, {"item_id": 2, "name": "Bar"}, {"item_id": 3, "name": "Now"}]
15
16

```

Figure 27.26.: Python tests

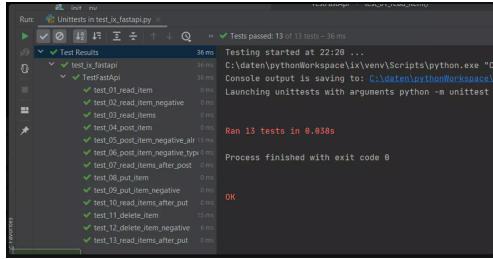


Figure 27.27.: Jetzt starten Sie die Tests wie zuvor und sehen folgendes Ergebnis

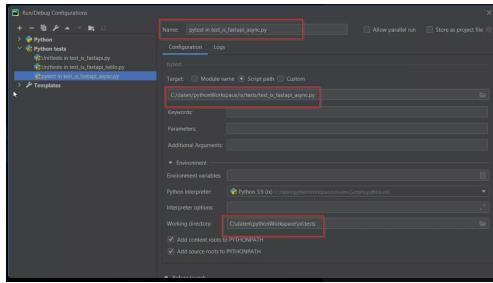


Figure 27.28.: OpenAPI für den Item-Lebenszyklus

bei dem die Unter-Funktion nacheinander aufgerufen wird. Im Test 3 hingegen wird der asynchrone Counter angesprochen, der alle drei Aufrufe quasi gleichzeitig absetzt und dann mit `gather` die Ergebnisse zusammenfügt. Während das erste Vorgehen notwendigerweise etwas mehr als 3 Sekunden benötigt, wird das im zweiten Fall in nur gut 1 Sekunde erledigt – Vorteil Asynchronität!

Aber es wird noch besser: Bei den `multi`-Tests (Nr. 2 und 4) wird über eine komplett identische Logik jeder Counter dreimal aufgerufen – und der Unterschied wird noch deutlicher:

- Synchron: 9 Sekunden.
- Asynchron: immer noch etwa 1 Sekunde.

Die synchrone Schnittstelle kann zwar asynchron angesprochen werden, muss aber immer auf die dieselbe synchrone Untermethode warten. Das ist in der asynchronen Logik nicht der Fall.

Sie sehen also, dass es sich lohnt, auch die nachgelagerte Verarbeitung von den Schnittstellen zu entkoppeln.

27.14. OpenAPI für den Item-Lebenszyklus

Wenn Sie den Schnittstellen-Server im Debugging-Modus starten und dann in der Datei `ix_fastapi.py` einen Breakpoint durch einen Klick rechts neben der Nummerierung im Editor setzen, können Sie die Belegung der Variablen sehen, wenn Sie etwa über die OpenAPI-Oberfläche ein neues Item anlegen wollen.

Nachdem Sie den Request ausgelöst haben, sehen Sie folgendes in Pycharm:

- Der blaue Balken markiert die Zeile, in der der Debugger gerade steht.
- Der Kasten unten zählt die aktuell vorhandenen Variablen inklusive Wert auf.
- Die beiden Kästchen oben geben ebenfalls die Werte der Variablen aus, allerdings im Code.

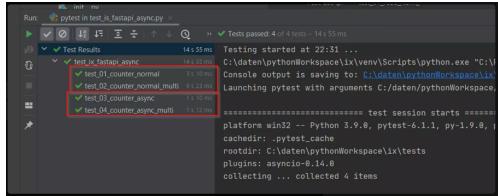


Figure 27.29.: Nachdem Sie den Request ausgelöst haben, sehen Sie folgendes in Pycharm

Gerade der letzte Aspekt machen Pycharm attraktiv. Im Code bieten die Variablen-Ausgaben einen viel intuitiveren Zugang.

27.15. Ausblick

Natürlich gibt es viele IDEs und jede hat ihre eigenen Stärken und Schwächen. Mit Pycharm machen Sie wenig falsch.

Mit Python Daten von beliebigen Websites auslesen am Beispiel Talkshows Dieser Artikel konnte nur einen Einstieg in die REST-Schnittstellenentwicklung mit Python geben. FastAPI bietet noch viel mehr Möglichkeiten, etwa weitere Parameter im Pfad oder Body von Anfragen, Dateiübertragung, Sicherheit und noch viel mehr. Ein Umstieg von Flask auf FastAPI lohnt sich auf jeden Fall. Gerade, wenn Sie noch nicht so viele Schnittstellen im Einsatz haben. Viel Erfolg!

28. Große Pandas

28.1. Programmieren mit Python: Große Datenmengen verwalten mit vaex

Wer häufig mit Python und pandas arbeitet, kennt das Problem: Irgendwann reicht der Speicher nicht mehr. vaex schafft Abhilfe und ist kompatibel zu pandas.

Die Python-Bibliothek pandas mit ihren zahlreichen Funktionen zum Umgang mit umfangreichen Datensätzen hat sich als Standard in der Data Science etabliert. Wer sich an die teilweise etwas spröde Bibliothek gewöhnt hat, kann sehr effizient damit arbeiten und viele Standardprobleme in der Datenvorbereitung und -analyse elegant lösen. pandas arbeitet dabei sehr schnell, weil es die Daten in DataFrames im Arbeitsspeicher hält.

Bei sehr großen Datensätzen passen die Daten aber irgendwann nicht mehr in den Speicher. Oft ist es einen Versuch wert, sich auf relevante Teile der Daten zu beschränken – aber das ist nicht immer möglich. Besonders bei der Datenaggregation kommt es oft zu Problemen. Perfekt wäre eine Bibliothek, die kompatibel mit pandas ist, aber bei Bedarf mit Daten im nichtflüchtigen Speicher arbeitet. Genau das leistet vaex.

Dabei ist vaex sehr schnell. Auch DataFrames, die nicht im Arbeitsspeicher liegen, können häufig mit 109 Zeilen pro Sekunde und schneller verarbeitet werden: In vielen Anwendungsfällen stehen Ergebnisse praktisch unmittelbar zur Verfügung. vaex ist dabei außerdem äußerst effizient und berechnet Spalten erst dann, wenn sie wirklich benötigt werden. Das kann erhebliche Berechnungszeit sparen, aber in einzelnen Fällen auch dazu führen, dass Wartezeit entsteht, wenn man gar nicht damit rechnet.

Den Speicher verwaltet vaex ebenso intelligent. Bei den besonders häufigen Filter- oder Auswahloperationen vermeidet es, Daten zu kopieren, sondern verweist nur auf Referenzen. Das führt bei Datenumrechnungen nicht nur zu einer schnellen Verarbeitung, sondern belastet auch den Arbeitsspeicher nur minimal. Einen großen Vorteil von pandas übernimmt vaex auch gleich: Daten lassen sich direkt im Jupyter-Notebook visualisieren. Außerdem zeigt vaex in Jupyter-Notebooks die von pandas bekannten Vervollständigungen an.

Alles in allem fühlt es sich also nicht viel anders an, mit vaex statt mit pandas zu arbeiten. Der Unterschied wird erst spürbar, wenn die Datenmengen für pandas zu groß werden und vaex einfach weitermacht.

28.2. vaex im Einsatz

vaex ist modular aufgebaut und besteht aus unterschiedlichen Paketen. Installiert wird es mit

```
pip install vaex
```

Beispiele und Datensets bringt vaex bereits mit, sodass man unmittelbar loslegen kann. Passenderweise sind die DataFrames recht groß. vaex kann sie direkt von Amazon S3 öffnen, beispielsweise Taxifahrten in New York von 2009 bis 2015:

```
import vaex
tdf = vaex.open('s3://vaex/taxi/yellow_taxi_2009_2015_f32.hdf5')
tdf.shape()
```

(1173057927, 18)

Hier sieht man bereits, wie ähnlich vaex zu pandas ist (das allerdings Schwierigkeiten hätte mit über einer Milliarde Datensätzen). Einige gewohnte Funktionen verhalten sich etwas anders, so liefert `df.columns` nicht direkt die Spaltennamen zurück, sondern ein Proxy-Objekt: Ein DataFrame kann auch sehr viele Spalten enthalten und aufgrund der Lazy-Evaluierung berechnet vaex die erst, wenn sie benötigt werden (`list(df.columns)` hilft hier weiter). Genau wie bei pandas kann der DataFrame mit `tdf.head()` im Jupyter-Notebook ausgegeben werden.

Funktionen wie `sample()` stehen auch zur Verfügung. Filteroperationen kann man gefahrlos ohne Download des Datensets durchführen, weil vaex die Operationen erst bei Bedarf durchführt:

```
many_passengers = tdf[tdf.passenger_count > 10]
```

Vorsicht allerdings bei `len(many_passengers)`, das führt zum Download und zur Berechnung (1165 Fahrten hatten mehr als zehn Passagiere). Mit einem kleineren Beispiel-Datenset (`df = vaex.example()`) lassen sich noch weitere Features ausprobieren. Besonders spannend und häufig sinnvoll ist das Binning, bei dem man mit einem kleineren DataFrame arbeitet und dafür eine Statistik berechnet, um den großen Download zu sparen. vaex teilt dazu den DataFrame in ein x-y-Grid auf und berechnet für jedes Quadrat im Grid den Mittelwert der Größe z:

```
edf=vaex.example()  
edf.mean(edf.z, binby=[edf.x, edf.y], shape=32, limits=[-10, 10])
```

Das Ergebnis kann man als eine Verallgemeinerung eines Histogramms betrachten. Besonders in sehr großen Datensätzen ist es häufig sinnvoll, mit solchen Aggregaten zu arbeiten, um sich einen Eindruck über die Struktur der Daten zu verschaffen.

vaex kann mit vielen Datenformaten umgehen, besonders gut mit solchen für große Datenmengen. Das oben genutzte Taxi-Datenset liegt im HDF5-Format vor, das ursprünglich aus der Hadoop-Welt stammt. Aber auch CSV, Apache Arrow, Apache Parquet oder FIT stellen kein Problem für vaex dar. Diese Datenformate können auch platzsparend spaltenorientiert geschrieben werden.

28.3. Fazit

vaex ist ein sehr leistungsfähiges Tool für den Umgang mit sehr großen Datenmengen, die zu umfangreich für pandas sind. In vielen Bereichen ist es dazu kompatibel, es gibt allerdings einige Unterschiede, besonders in der Lazy-Evaluierung von Werten. vaex erlaubt auch direktes Machine Learning auf sehr umfangreichen Datenmengen. Prof. Dr. Stefanie Scholz forscht an der Wilhelm-Löhe-Hochschule unter anderem zu Themen rund um KI-gestütztes Data-driven Marketing.

28.4. Tasks

- Translation
- Improvements
- Further readings
- Example code with Python
- Presentation

29. Durchdacht in die Cloud

29.1. Einführung

Sollen Unternehmensdatenbanken in die Cloud wandern, sind zahlreiche Aspekte zu bedenken. Damit diese Operation am offenen Herzen gelingt, steht am Anfang jedes Migrationsprojekts eine durchdachte Planung.

- Eine Verlagerung von Datenbanken in die Cloud ist kein Spaziergang, bietet aber Vorteile unter anderem bei Flexibilität, Skalierbarkeit und der Entlastung des IT-Personals im Unternehmen.
- Ohne durchdachte Strategie sollte man keine Datenbankmigration in Angriff nehmen.
- Für den Weg einer Datenbank in die Cloud gibt es mehrere Varianten: vom harten Big Bang Lift and Shift bis zu mehrstufigen Ansätzen.
- Die Cloud-Migration der Datenbank ist die beste Gelegenheit, auch anderen Anwendungen und Prozessen die Vorteile der Cloud zu verschaffen.

Muss über die Cloud noch diskutiert werden? Weit mehr als 80 Prozent aller deutschen Unternehmen nutzen wenigstens ein Cloud-Angebot. Ganze Softwaregattungen wie CRM- oder Office-Pakete sind bereits in die Cloud umgezogen. Ihre Anziehungskraft wirkt auch auf Datenbankmanagementsysteme (DBMS). Alle Hyperscaler bieten relationale oder NoSQL-Datenbanken als Service an und einige Datenbankspezialisten offerieren neben ihren On-Premises-Anwendungen eigene SaaS-Ansätze, wie die beiden weiteren Titelartikel ab Seite 42 und 50 in dieser Ausgabe zeigen.

Also ab in die Cloud mit der Datenbank? Ganz so einfach geht das häufig nicht. Eingriffe in die Datenbankarchitektur entsprechen schließlich Operationen am offenen Herzen. Zudem sind Datenbanken nur so nützlich wie die Anwendungen, die auf sie zugreifen – etwa ERP, Warenwirtschaft oder Buchhaltungssoftware. Gehören diese nicht auch in die Cloud? Damit ist die Diskussion eröffnet – nicht über das Ob, sondern über das Wie.

29.2. Lift and Shift und andere Arten der Migration

Grundsätzlich gibt es drei Möglichkeiten, eine Datenbank in die Cloud zu bringen. Der erste Fall ist der einfachste: Das Unternehmen erzeugt eine Kopie der gesamten Datenbank bei einem Infrastruktur-Anbieter. Dabei wird nichts am Datenbanksystem und den damit verbundenen Anwendungen geändert, sondern es werden lediglich die physischen Server durch virtuelle ersetzt. Diese Form der Datenbankmigration ist allerdings nicht besonders interessant für das Unternehmen, es sei denn als Failover-/Stand-by-Alternative in einem Disaster-Recovery-Szenario (Abbildung 1). Der Umstieg schleppt alle Einschränkungen der ehemaligen On-Premises-Systeme mit, neue Möglichkeiten und Funktionen der Cloud sind zunächst nicht verfügbar.

Erst mit dem zweiten Fall wird es ernst: der Lift and Shift eines DBMS zu einem Plattformservice, auch Database as a Service (DBaaS) genannt. Wenn sowohl die lokale Variante als auch die Plattform denselben Funktionsumfang besitzen, sind keine grundlegenden Änderungen an Anwendungen oder Daten erforderlich. Dies

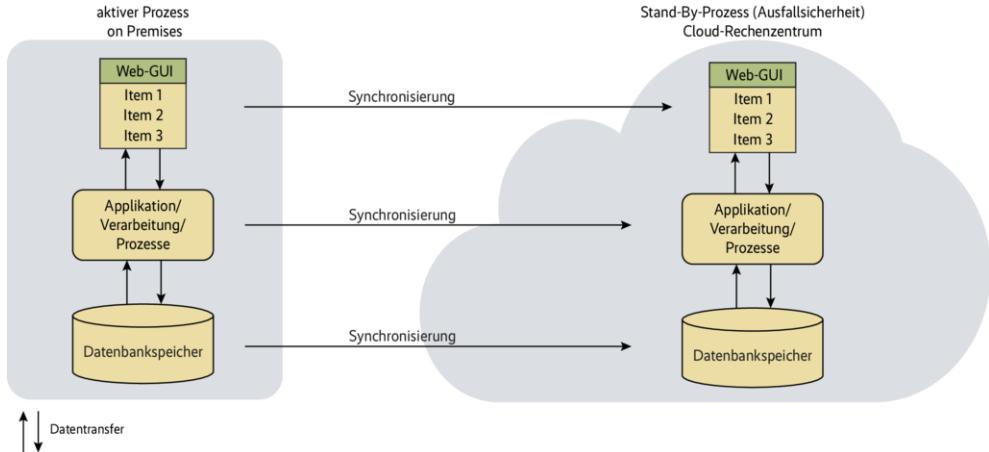


Figure 29.1.: Lift and Shift: Das On-Premises-DBMS und der Cloud-Service haben denselben Umfang, Quelle: MariaDB Corporation

setzt natürlich voraus, dass die neue Datenbank mit der alten kompatibel ist. Ein Beispiel hierfür wäre ein Lift and Shift von MySQL zu MariaDB SkySQL. Beim Lift and Shift wird die vorhandene Datenbankarchitektur einfach dupliziert. Sie ist anschließend leicht zu optimieren, da DBaaS-Versionen oft zusätzliche Funktionen zur Leistungsdiagnose und zur Optimierung enthalten.

Der dritte Fall ist eine Migration bei unterschiedlichen Datenbanksystemen, die nicht oder nur sehr eingeschränkt miteinander kompatibel sind. Diese erfordert eine detaillierte Planung, die weit über das triviale Lift-and-Shift-Verfahren hinausgeht. Denn in diesem Fall sind teils erhebliche Anpassungen an den Datenbankschemata, den genutzten Schnittstellen sowie den Anwendungen nötig. Auch Stored Procedures und Querys sind unter Umständen intensiv anzupassen.

Kurzum: Eine Datenbankmigration in die Cloud lässt sich nicht nebenbei erledigen. Eine naheliegende Frage lautet also: Warum sollte ein Unternehmen diese Anstrengung auf sich nehmen? Diese Frage stellt sich vor allem in Situationen, in denen die Systeme problemlos laufen. Doch es gibt viele Gründe für den Einsatz einer (neuen) Cloud-Datenbank.

29.3. Vorteile der Datenbankmigration in die Cloud

Zunächst bringt eine DBaaS eine erhebliche Arbeitserleichterung: Die Unternehmen erhalten in der Cloud eine vollständige Infrastruktur, die ohne eigenen Personaleinsatz überwacht wird. Alle Instanzen werden bedarfsgerecht bereitgestellt und für Transaktions-, Analyse- und HTAP-Workloads optimiert. Automatische Sicherungen der Datenbank über Nacht decken zumindest basale Backup-Funktionen ohne zusätzlichen Aufwand ab. Hinzu kommt die rasche Skalierbarkeit: Um zusätzliche Datenbankleistung kurzfristig hinzuzufügen braucht es kaum mehr als das Starten eines Skripts. Um die dahinterliegenden Virtualisierungsfunktionen und Hardwaregegebenheiten kümmert sich der Provider, der dafür bezahlt wird.

Die einfache Skalierung betrifft die gesamte Infrastruktur. Wenn beispielsweise Entwickler Testsysteme für Änderungen an der Datenbankstruktur benötigen, sind sie – inklusive Kopie des DBMS und einiger Testdaten – meist noch am selben Tag verfügbar. Auch Replikate der aktuellen Datenbank für den Aufbau eines Clusters mit Loadbalancer sind schnell bereitgestellt und ebenso schnell wieder deaktiviert, sobald der Workload auf normale Werte zurückgeht.

Vor allem weltweit agierende Unternehmen profitieren von regionaler Verfügbarkeit,

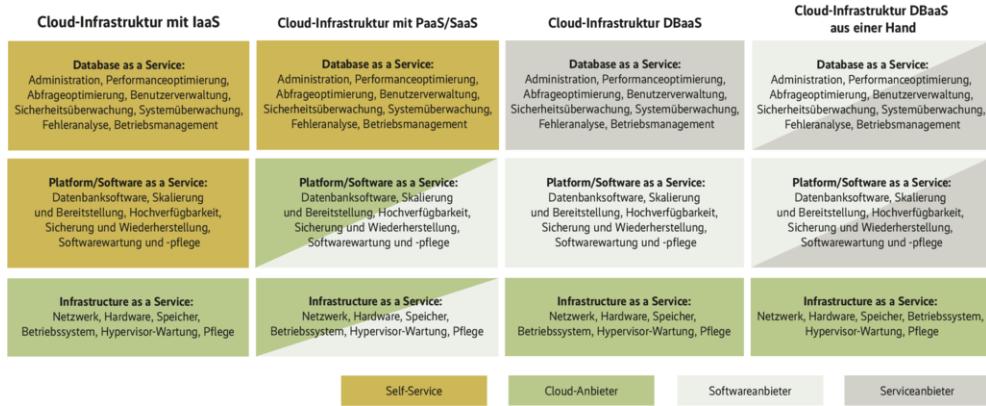


Figure 29.2.: Vielfalt: Bereitstellungsmodelle für Cloud-DBMS vom virtuellen Server bis zu DBaaS aus einer Hand, Quelle: MariaDB Corporation

da nicht alle globalen Zugriffe auf dieselben Instanzen geleitet werden. Dies erlaubt die genaue Verteilung der Arbeitslast, beispielsweise im globalen E-Commerce. Denn Spalten wie der Black Friday wandern mit den Zeitzonen rund um den Globus. Für diese Zwecke stellen die meisten Provider Instanzen in definierten Größen und mit unterschiedlichen Storage-Parametern bereit.

Für kleinere und mittelgroße Unternehmen ist ein solcher Service besonders wichtig, da sie oft ohnehin auf externe Dienstleister zurückgreifen. Je nach Servicelevel ist die technische Unterstützung in der Cloud ohne Zusatzgebühren verfügbar.

Apropos Kosten: Eine DBaaS wird, wie bei Cloud-Services üblich, nach Nutzung abgerechnet, auch bei kurzfristiger Skalierung. So werden die Lastspitzen nicht übertrieben teuer, da keine Rechenleistung „auf Vorrat“ gebucht werden muss. Die Unternehmen bezahlen einen Preis, der je nach Workloads und Anforderungen an die Rechenleistung schwanken kann. Dies gilt auch für eine sinkende Leistung, etwa bei einem stark saisonabhängigen Geschäftsmodell.

29.4. Auswahlkriterien für die richtige Datenbankplattform

Es gibt also viele Gründe, dem Datenbanksystem den Weg in die Cloud zu bahnen. Doch ein Spaziergang wird das nicht. Deshalb müssen Unternehmen vieles bedenken und einige Vorbereitungen treffen. Besonders wichtig: Die genutzte Cloud-Datenbank sollte für die im Unternehmen üblichen produktiven Workloads optimiert sein und nicht etwa eine allgemeine Open-Source-Basisversion.

Der Markt bei Cloud-Datenbanken ist inzwischen recht groß. Auf der einen Seite werben klassische Datenbankanbieter für ihre penibel an das jeweilige Datenbanksystem angepassten Clouds. Am anderen Ende des Spektrums bieten die großen Hyperscaler alles für Datenbanken – allerdings leider oft nur proprietär. In beiden Fällen ist der gefürchtete Vendor Lock-in nicht auszuschließen. Denn beide Angebotstypen erschweren die Nutzung anderer DBMS aus geschäftlichen Gründen oder schließen sie gleich ganz aus.

Deshalb ist eine wichtige Eigenschaft von Datenbanksystemen für die Cloud mit dem Begriff Cloud-agnostisch am besten beschrieben. DBMS sollten für unterschiedliche Cloud-Ökosysteme geeignet sein, damit Unternehmen nicht an einen Anbieter gebunden sind. Das ist die Voraussetzung für Multi-Cloud-Strategien, die viele Unternehmen gerne nutzen, um technologisch flexibel zu bleiben.

Allerdings gibt es hier den Stolperstein der Datenbanktransfers zwischen unter-

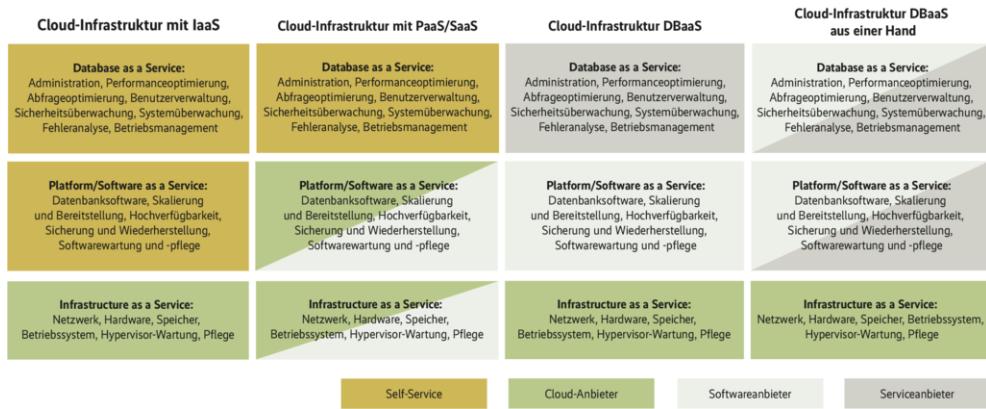


Figure 29.3.: Open-Source-DBMS sind überall verfügbar und unterstützen so Lift and Shift, Quelle: MariaDB Corporation

schiedlichen Clouds. Dafür berechnen die meisten Anbieter zusätzliches Geld. Deshalb muss das DBMS so gestaltet werden, dass keine überflüssigen Transfers nötig sind. Ganz generell sollte das IT-Management immer die Gebührenstruktur der Cloud im Blick haben. So kostet beispielsweise eine zweite Verfügbarkeitszone (Region) zusätzliches Geld, ist aber nicht für jedes Einsatzszenario notwendig.

29.5. Tipps für die Auswahl der Cloud-Datenbank

Bei der Auswahl des Providers sind viele weitere Parameter zu beachten.

Verfügbarkeit: Einige Datenbanksysteme sind in der Cloud nur bedingt verfügbar (Abbildung 3). Open-Source-DBMS haben die Nase vorn und sind (fast) überall zu haben.

Latenz: Die Paketumlaufzeit zwischen den Datenbankservern und den Anwendungen sollte so gering wie möglich sein. Eine weit abgelegene Cloud-Region ist nicht sinnvoll, was die Auswahl auf in Deutschland aktive Cloud-Provider beschränkt.

Vertragsgestaltung: Rollen, Verantwortlichkeiten und das Eigentum an den Daten müssen geregelt sein, ebenso der Ablauf beim Providerwechsel.

Servicelevel: Bei vielen Providern sind das 99,9 oder 99,95 Prozent Verfügbarkeit. Bei einigen Anbietern gibt es eine höhere Verfügbarkeit gegen zusätzliche Gebühren.

Aktualisierungen: Der DBaaS-Anbieter stellt die Software-Upgrades bereitg. Dabei ist es wichtig, dass alle neuen Versionen und Sicherheitsaktualisierungen so schnell wie möglich ausgerollt werden.

Verschlüsselung: Die Datenbank sollte sowohl eine Transportverschlüsselung als auch eine Verschlüsselung der einzelnen Tabellen anbieten. So ist sichergestellt, dass wirklich ausschließlich das nutzende Unternehmen Einblick in die Daten erhält.

Technologie und Software: Viele Firmen besitzen hier eine große Vielfalt, die auch in der Cloud vorhanden sein muss. Darüber hinaus unterstützen Cloud-Provider normalerweise nur die jeweils neuesten stabilen Versionen von Anwendungen und Bibliotheken. Wer hier im Rückstand ist, muss schleunigst nacharbeiten.

Zusätzlicher Support durch Admins: Cloud-Datenbanken automatisieren vieles, aber nicht alles. Einige Cloud-Datenbanken bieten daher zusätzliche Supportdienstleistungen für die Datenbankadministration durch qualifizierte Mitarbeiter an.

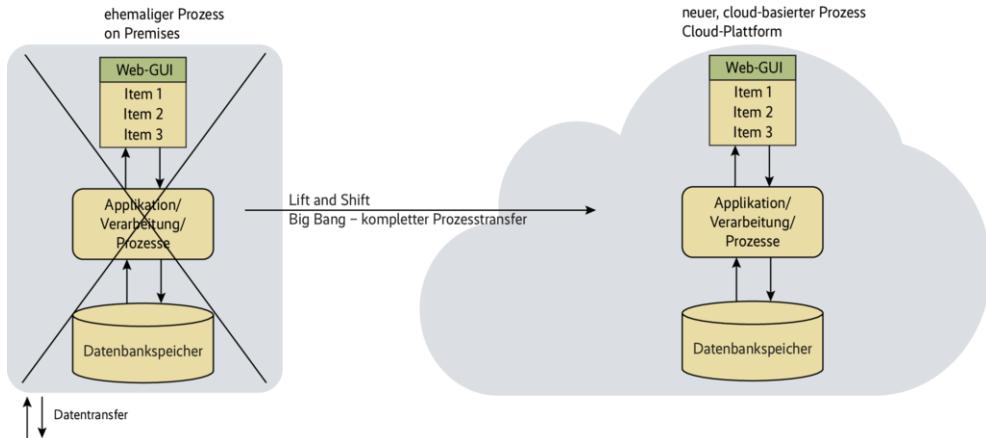


Figure 29.4.: Umzug mit Big Bang – alles auf einen Schlag in die Cloud schießen,
Quelle: MariaDB Corporation

29.6. Nie ohne Strategie beginnen

Grundsätzlich benötigen Migrationsprojekte eine klare Strategie, da häufig auch andere Systeme in die Cloud migriert werden. Oft steht sogar die Abschaffung einzelner Altanwendungen auf der Tagesordnung, wenn es diese in der Cloud nicht gibt. Je nach Größe des Unternehmens und des IT-Budgets kann die Migration entweder langsam und schrittweise oder auf einen Schlag (Big Bang) erfolgen. Die zweite Strategie ist allerdings aufwendig und risikoreich, da sie einen Parallelbetrieb erfordert, um die neuen Infrastrukturen zu optimieren. Die erste Strategie dagegen ist risikobewusster und im Einzelfall weniger komplex.

29.7. Die Deadline im Blick haben

Wichtig ist auch der Zeithorizont. Eine Migration nach dem Motto „Lift and Shift“ verschiebt alles auf einmal in die Cloud, aber ohne jede Änderung an den zugrunde liegenden Systemen. Das geht schnell, erhält der IT-Abteilung aber die altbekannten Ineffizienzen und Störfaktoren. Zumindest mittelfristig ist ein Reengineering meistens unvermeidbar, schließlich soll der Gang in die Cloud nicht nur ein werbewirksames neues Label bringen.

Sinnvoll ist es also, lieber direkt umfassend zu planen. Dabei gehören sowohl die Prozesse als auch die Datenbankarchitektur auf den Prüfstand. Der Gang in die Cloud ist nämlich eine sehr gute Gelegenheit, alte Zöpfe abzuschneiden und die gesamte Umgebung zu optimieren. Dies bedeutet allerdings auch erhebliche Komplexität. Selbst mittelgroße Unternehmen sollten mit mindestens drei Jahren Projektlaufzeit rechnen, legt die Migrationspraxis bei MariaDB-Kunden nahe.

29.8. Was kommt zuerst?

Im Vorfeld müssen Unternehmen klären, was zuerst migriert wird: Frontend oder Backend, also das eigentliche DBMS? Die Antwort hängt in erster Linie vom Anwendungsszenario ab. Wer sein Datenbanksystem in die Cloud bringen will, muss zunächst die datenintensiven Prozesse bestimmen und in die Cloud bringen. Anschließend folgt der Rest. Ein anderer Ansatz stellt hingegen die Kosten in den Vordergrund. Dabei wird die Reihenfolge der Migration anhand der Schwierigkeit der Umsetzung bestimmt.

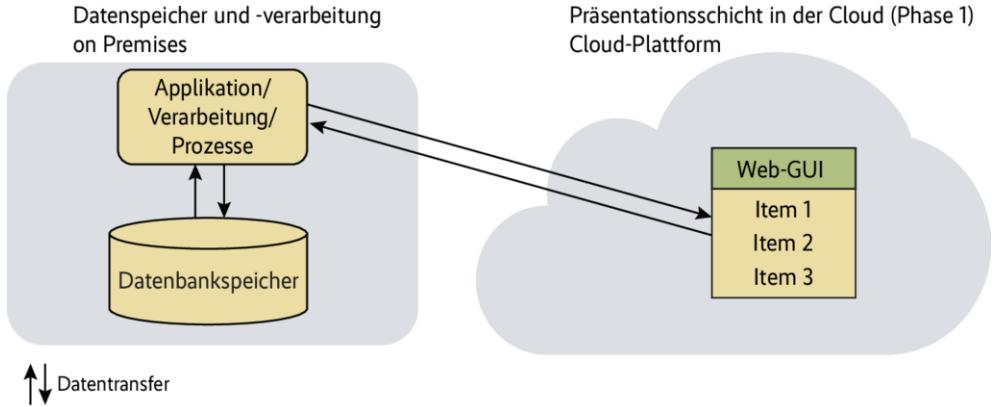


Figure 29.5.: Schrittweise Migration zur Risikosenkung – erst wandert das Frontend in die Cloud ..., Quelle: MariaDB Corporation

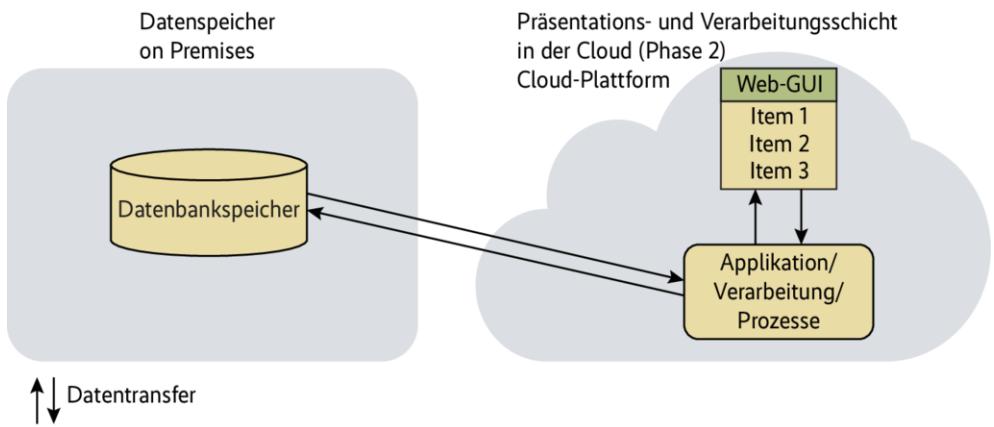


Figure 29.6.: ... und im zweiten Schritt bleibt nur der Datenbankspeicher on Premises ..., Quelle: MariaDB Corporation

Die Konsequenz ist meist, zunächst das leichter zu portierende Frontend in die Cloud zu verschieben.

Ein weiteres Szenario für die Cloud-Migration ist das Verbessern der Skalierung. In diesem Fall identifiziert die IT-Organisation zuerst alle langsamen Systeme. Das wird in vielen Fällen auch das Datenbanksystem sein. Anschließend folgt dann die schrittweise Übertragung in die Cloud.

Dabei ist es sinnvoll, zur Risikosenkung die eigentliche Migration in drei Schritten umzusetzen: Zuerst wird das Frontend in die Cloud verschoben, dann die Präsentations- und Anwendungsschicht und zuletzt die eigentliche Datenbank (siehe Abbildungen). Auf jeden Schritt folgen dann spezifische Anpassungen und eine Testphase, sodass die Komplexität der Migration beherrschbar bleibt.

29.9. Fazit: Die Datenbankmigration beginnt im Kopf

Bis hierhin sollte klar geworden sein, dass Cloud-Migration zunächst im Kopf stattfindet. Ohne strategische Überlegungen, ein ausreichendes Budget für Zeit und Geld sowie eine gute Vorbereitung ist das Projekt zum Scheitern verurteilt.

Bei Erfolg macht die Cloud alles deutlich agiler, vieles einfacher und einiges kosteneffizienter. Doch Migrationsprojekte sind aufwendig, oft teuer und die Datenbankmigra-

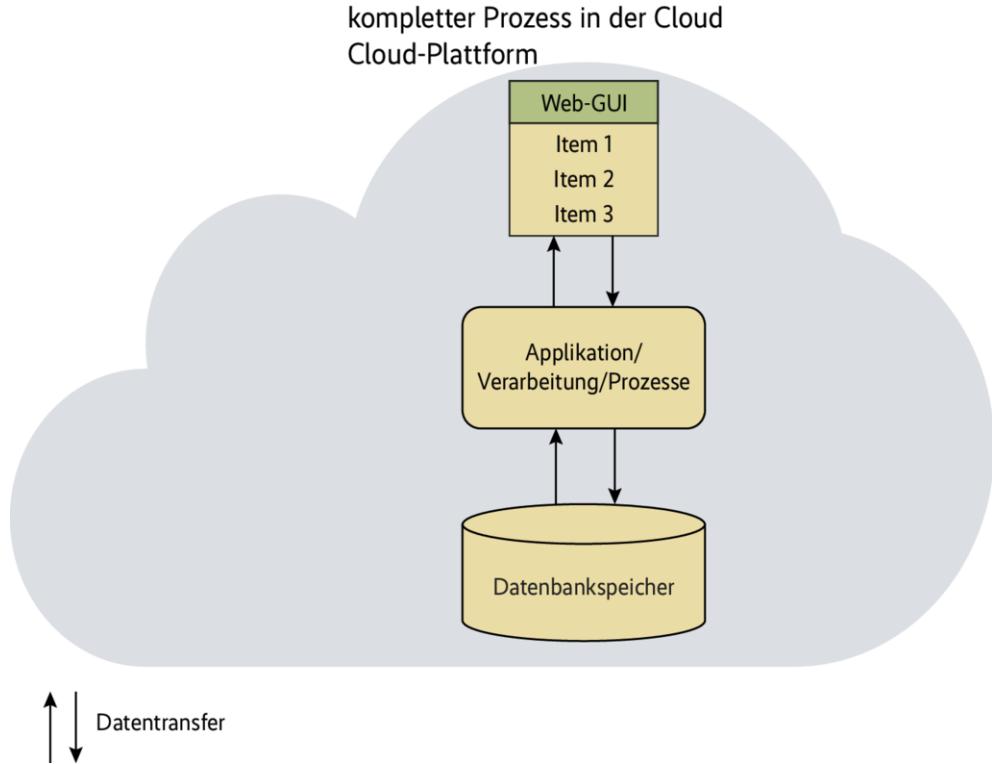


Figure 29.7.: Schritt drei vollendet schließlich die Migration. Jetzt arbeitet das gesamte DBMS in der Cloud, Quelle: MariaDB Corporation

tion sogar recht risikoreich. Zeit und Geld für intensive Tests und eine kurze Phase des Parallelbetriebs sollten direkt eingeplant werden.

Deshalb ist es nicht besonders sinnvoll, nur die Datenbank und vereinzelte Anwendungen in die Cloud zu bringen. Im Gegenteil: Die Cloud-Migration der Datenbank ist die beste Gelegenheit, auch anderen Anwendungen und Prozessen die Vorteile der Cloud zu verschaffen. (avr@ix.de)

29.10. Tasks

- Translation
- Improvements
- Further readings
- Presentation

30. Cloud-Datenbanken der Hyperscaler

30.1. Einleitung

Alle Hyperscaler kodern Kunden heute mit mehreren Datenbanken, für die sie Betrieb und Wartung übernehmen. Was haben AWS, Azure und GCP im Angebot, welche Funktionen zeichnen die Cloud-Datenbanken aus und womit können und müssen Anwender rechnen? Ein Überblick.

- Mit ihren Database-as-a-Service-Angeboten wollen Provider den Admins den Aufwand für Wartung und Betrieb abnehmen.
- Meist provisionieren Kunden per API oder GUI eine Datenbank, legen das Feature-Set fest und erhalten dann die Zugangsdaten für den Zugriff aus der lokalen Umgebung.
- Hochverfügbarkeit, Backup und Recovery sowie andere klassische Wartungsaufgaben übernimmt der Cloud-Provider vollständig.
- Die Angebote der Hyperscaler beinhalten einen unangenehmen Lock-in-Effekt: Wer einmal einen DBaaS-Dienst eines Anbieters nutzt, kann ohne großen Migrationsaufwand kaum auf eine andere Plattform umsteigen.

Set-up, Pflege und Betrieb von Datenbanken gehören üblicherweise nicht zu den Lieblingsaufgaben von Administratoren. Dennoch benötigt jede Applikation heute in irgendeiner Form eine Datenbank, meist eine relationale Variante mit SQL – auch wenn NoSQL- und andere Datenbanken hier aufgeholt haben. Die für den Betrieb und das Set-up von Datenbanken anfallenden Tasks sind über alle Typen hinweg ähnlich bis identisch: Dem initialen DB-Set-up folgt die Frage nach Hochverfügbarkeit, damit die Datenbank nicht zum Single Point of Failure wird. Ist das geklärt, stehen Backup und Restore auf der Agenda. Die sind im Datenbankkontext noch wichtiger als sonst, da die Datenbank heute in vielen Umgebungen die einzige Instanz ist, die noch persistente Daten hält.

Schon früh haben die großen Cloud-Anbieter – allen voran Amazon mit AWS – den Reiz von Database as a Service (DBaaS) erkannt. Die Idee ist so simpel wie genial: Statt sich eine Datenbank mühsam selbst zu bauen, setzt ein Nutzer oder Administrator per API bloß noch den passenden Befehl ab oder klickt im Webinterface auf den passenden Button. Nach ein paar Minuten steht ihm dann eine Instanz von MySQL, PostgreSQL oder einer beliebigen anderen Datenbank zur Verfügung, für die er nur die IP und die Log-in-Daten kennen muss. Um den Betrieb kümmert sich in so einer Managed Database der Anbieter komplett autark und meistens per Automatisierung: Die Plattform legt Backups von den jeweiligen Datenbanken ebenso automatisch an, wie sie eine gegebenenfalls gewünschte Replikation erstellt. Geht mal etwas schief und ein Restore ist nötig, funktioniert das per GUI oder API ebenfalls in Sekundenschnelle. Kein Vergleich also mit den Klimmzügen, die Administratoren beim händischen Betrieb zu vollführen haben.

30.2. Auch für Anbieter praktisch

Weit über die Annehmlichkeiten für Nutzer hinaus reichen die Vorteile für ihre jeweiligen Anbieter. Zwar verursacht es Aufwand und Mühe, ein DBaaS-Angebot auf die Beine zu stellen. Doch läuft es erst einmal, lässt es sich ohne großen Zusatzaufwand beliebig oft verkaufen. Obendrein eignet sich ein solcher Dienst aus Sicht des Anbieters hervorragend für einen klassischen Lock-in-Effekt: Wer etwa einmal an die DBaaS-Angebote von AWS gewöhnt ist und seine Applikationen und virtuellen Set-ups so gebaut hat, dass sie mit diesen perfekt funktionieren, steigt nicht „mal eben“ auf einen anderen Anbieter um.

Wie so oft gilt deshalb: Es prüfe, wer sich ewig bindet. Genau das ist vor dem Hintergrund des großen Angebots der Hyperscaler eine Herausforderung. Diese Marktübersicht schafft Abhilfe und stellt die DBaaS-Angebote von AWS, Azure und GCP gegenüber. Sie enthüllt deren Stärken, Schwächen und Eigenheiten und gibt eine ungefähre Vorstellung davon, welches Portfolio sich für welchen Einsatzzweck eignet.

30.3. AWS: der Platzhirsch

Amazons AWS rühmt sich, das Konzept DBaaS Ende der 2000er-Jahre mit erfunden zu haben. Tatsächlich gehört DBaaS nach EC2 und S3 zu den ersten auf AWS verfügbaren Diensten. Damals war die Implementierung noch durchaus rustikal: Der Start einer Instanz etwa führte dazu, dass per CloudFormation automatisch eine neue VM gestartet und mit der passenden Software betankt wurde. Gut möglich, dass heute immer noch vergleichbare Prozesse am Werk sind. Doch hat Amazon seine DBaaS-Funktionen immer stärker abstrahiert, sodass Nutzer von den Vorgängen hinter den Kulissen praktisch nichts mehr mitbekommen. Vermutlich wäre das für manchen ohnehin zu viel, denn AWS prahlt damit, DBaaS-Dienste mit elf Datenbanken im Serviceangebot zu haben. Drei davon sind relational, andere basieren auf In-Memory-Prinzipien oder sind NoSQL-Datenbanken.

Wer eine klassische SQL-Datenbank etwa für den Betrieb von WordPress braucht, landet wahlweise bei AWS Aurora (Abbildung 1) oder AWS RDS (Abbildung 2). Wobei auf den ersten Blick im Grunde gar nicht ersichtlich ist, wodurch die Dienste sich unterscheiden – denn AWS bewirbt beide als gemanagte relationale Datenbanken. Ein genauerer Blick zeigt fundamentale Differenzen zwischen den Architekturen der Dienste. RDS entspricht eher dem klassischen Ansatz, wirkt also so, als startete ein Admin in einer eigenen EC2-Instanz eine eigene Datenbank. Die komplette Wartung übernimmt aber AWS. Ihre Daten legen RDS-Instanzen auf elastischen Volumes (Elastic Block Store, EBS) ab, Funktionen wie automatisches Failover und Backups lassen sich auf Wunsch aktivieren.

Aurora hingegen ist deutlich stärker abstrahiert. Hier nutzt der Dienst keine EBS-Volumes, sondern einen proprietären Mechanismus, der Replikate der Daten in verschiedenen Verfügbarkeitszonen anlegt – insgesamt sechsmal. Läuft etwas schief, eröffnet das ganz andere Möglichkeiten als bei RDS: Zu den erklärten Stärken von Aurora gehört etwa die extrem schnelle Wiederanlaufzeit nach Ausfällen, die im Regelfall deutlich unter einer Minute liegen soll. Auch in Sachen Performance profitiert Aurora vom proprietären Speicher im Hintergrund. Sie leistet mehr IOPS bei gleicher Request-Größe und bietet auch eine höhere Bandbreite als das etwas in die Jahre gekommene RDS.

Kaum erwähnenswerte Unterschiede ergeben sich hingegen bei den zur Verfügung stehenden SQL-Frontends. Aurora unterstützt nur PostgreSQL und MySQL, was für 95 Prozent der Anwendungen ausreichen dürfte. RDS erweitert den Protokollsupport um MariaDB, MS SQL und Oracle und gibt sich dadurch deutlich vielseitiger.

Ebenfalls fortgeschritten präsentiert sich Aurora in Sachen Skalierbarkeit. Weil er sich im geclusterten Multi-Master-Modus betreiben lässt, bietet der Datenbankdienst

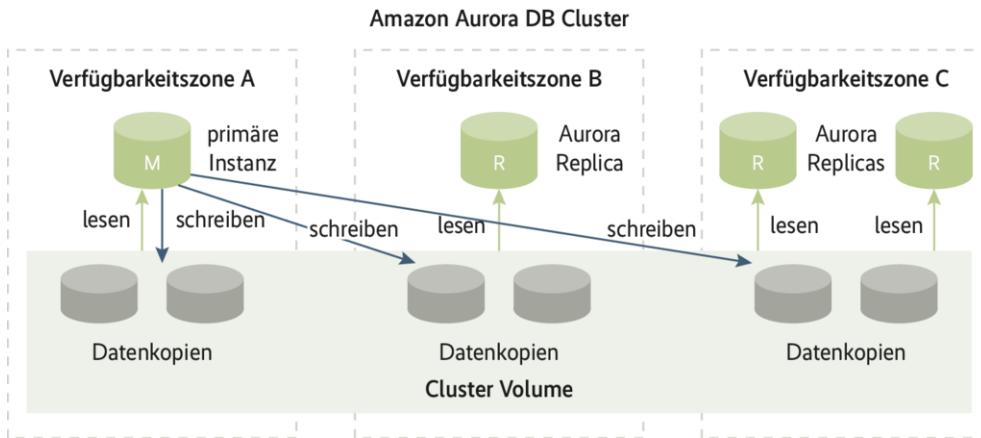


Figure 30.1.: AWS Aurora ist eine vom Backend komplett abstrahierte Datenbank, die aber trotzdem relational funktioniert. Ihre Daten sichert sie diffus “in der Cloud”, aber dort mindestens sechsmal. Quelle: AWS

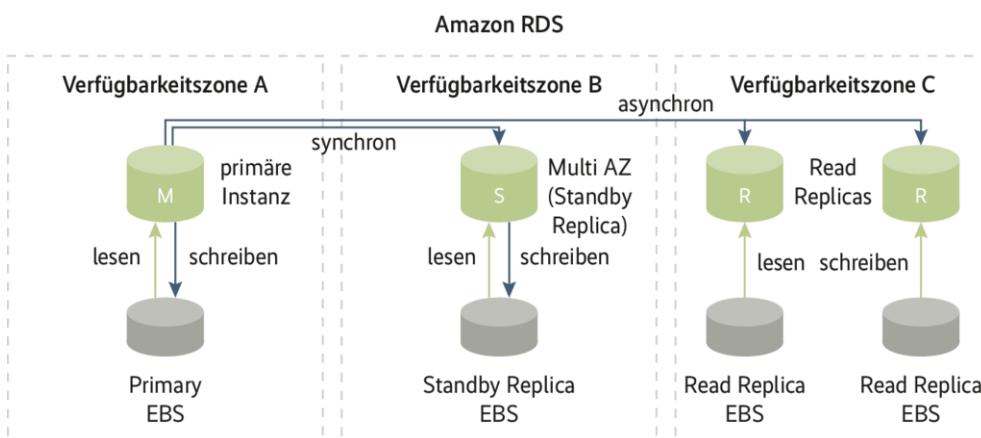


Figure 30.2.: AWS RDS ist in gewisser Weise der Gegenentwurf zu Aurora: eine in einer eigenen Instanz laufende Datenbank mit lokalem Storage, die zudem etwas günstiger als Aurora ist. Quelle: Awesome Cloud

nahtlose Skalierbarkeit in die Breite. Das funktioniert auf Wunsch auch automatisch anhand der Auslastung, sodass Aurora unnötige Kosten verhindert. Vergleichbares fehlt bei RDS.

Im Gegenzug ist Aurora bei Berücksichtigung aller Faktoren aber auch merklich teurer als RDS. Hier gilt es also, im Vorfeld die Anforderungen zu klären und das passende Produkt zu wählen. Für eine einfache WordPress-Instanz ist Aurora vermutlich zu viel des Guten. Wer global verteilte, auf hohe Performance angewiesene Anwendungen nutzt, liegt mit Aurora aber ganz richtig.

30.4. AWS Redshift als Data Warehouse

Ebenfalls eine relationale Datenbank ist AWS Redshift, allerdings taugt das Produkt nicht als General-Purpose-Datenbank. Denn es ist auf den Betrieb eines Data Warehouse optimiert, dessen Inhalte sich anschließend per KI oder Machine Learning automatisch analysieren lassen. Intern funktioniert es aber wie eine relationale Datenbank, zudem nutzt es für die Kommunikation mit der Außenwelt SQL. Redshift ist allerdings eher ein Nischenprodukt.

30.5. Key-Value Stores übernehmen

Derzeit sind Key-Value Stores ein Hauptkonkurrent relationaler Datenbanken. Hier ist die interne Struktur der Datenbank viel trivialer als bei SQL, wo jede Datenbank in Tabellen, Spalten und Reihen unterteilt ist. Klassische Key-Value Stores verfolgen das Prinzip von Nodes: Jeder Node hat einen eindeutigen Namen oder ein anderes eindeutiges Merkmal, anhand dessen er identifizierbar ist. Das soll viel schneller sein als relationale Datenbanken.

Tatsächlich hat sich am Markt etwa RocksDB von Facebook, ein klassischer Key-Value Store, mittlerweile gut etabliert und ist heute Bestandteil etwa vieler Cloud-native-Datenbankdienste wie YugabyteDB [1]. Da darf Amazon natürlich nicht fehlen: Mit DynamoDB bietet man einen eigenen komplett gemanagten Key-Value Store an.

30.6. Weitere Optionen

Darüber hinaus finden sich in AWS etliche Datenbanken für spezielle Aufgaben. In-Memory-Datenbanken dienen meist als Cache und halten strukturierte Daten im RAM vor. Sie kommen auch für das Session-Management oder bei Onlinespielen zum Einsatz, bei denen viele Spielstände parallel zu speichern und aktuell zu halten sind, und das über die Grenzen einzelner Knoten hinweg. AWS greift Administratoren mit zwei Datenbanken dieser Art unter die Arme. ElasticCache lässt sich mit dem Key-Value-Speicher Redis kombinieren sowie mit Memcached, einem ebenso für das Caching im RAM entworfenen Werkzeug. Alternativ dazu steht in Form von MemoryDB for Redis eine weitere AWS-Datenbank zur Verfügung, die wie ElasticCache als Unterbau für Redis dienen kann.

Dokumentdatenbanken sind durch ihre spezielle Struktur auf das Hosting textbasierter Inhalte wie JSON-Blobs optimiert. Amazon verspricht für seine DocumentDB Kompatibilität mit MongoDB. MongoDB-Anwendungen lassen sich darüber also in AWS sinnvoll betreiben.

Die verbliebenen vier Datenbanken im AWS-Portfolio gehören in die Kategorie Sonderfall, obgleich sie nicht weniger Leistung bieten als die bislang vorgestellten Vertreter. Keyspaces ist kompatibel zu Apache Cassandra und richtet sich an Nutzer, die einen Wide-Column-Speicher brauchen. Darunter versteht man eine NoSQL-Datenbank, die im Grunde wie ein zweidimensionaler Key-Value Store funktioniert. Zwar gibt es hier auch Tabellen, Reihen und Spalten wie in relationalen Datenbanken; jedes einzelne

Feld darin kann jedoch unterschiedliche Bezeichnungen und andere Formate nutzen als die direkt angrenzenden Felder.

Die Graphdatenbank Diagramm ist darauf spezialisiert, anhand der Beziehungen eines großen Datensatzes mit spezifischen Parametern schnell Verbindungen zu finden und daraus Bezugsbäume zu generieren. Ein Beispiel dafür wäre die schnelle Suche nach einer Person in einer Gruppe von Menschen.

Deutlich praxisnäher für die meisten Administratoren ist AWS Timestream, eine Zeitreihendatenbank, deren zentrales Element ein Zeitstrahl ist. Damit eignet sie sich besonders gut für das Aufzeichnen von Metrikdaten, die im Kontext von Monitoring, Alerting und Trending bei skalierbaren Aufgaben wichtig sind. Nicht fehlen darf schließlich Ledger, eine Datenbank mit verifizierbaren Transaktionen auf Basis kryptografischer Operationen für sicherheitstechnisch hochsensible Bereiche.

AWS ist mit seinem Portfolio gemanagter Datenbanken nicht nur als Erstes am Markt gewesen, sondern bietet auch eine große Vielfalt. Für praktisch alle Datenbankbedürfnisse des Alltags gibt es einen Dienst, der sich in wenigen Sekunden ausrollen und nutzen lässt. Um Themen wie Backups, Replikation oder Restore brauchen sich Administratoren keine Gedanken zu machen – es sei denn, sie wollen aus spezifischen Gründen explizit auf die Konfiguration Einfluss nehmen. Nach den Erfahrungen während unseres Tests scheint das aber nur dann sinnvoll zu sein, wenn berechtigter Grund zur Annahme besteht, ein DBaaS-Dienst von AWS könne hinsichtlich der eigenen Konfiguration eine Fehlannahme treffen. Für die meisten Set-ups hingegen gilt, dass die AWS-DBaaS-Dienste gut und zuverlässig funktionieren und dabei noch zu den günstigeren Komponenten im Portfolio gehören. Der zu zahlende Preis hängt letztlich von den genutzten Ressourcen ab. Übrigens: AWS bietet für etliche der eigenen Dienste auch Migrationswerkzeuge an.

30.7. Azure: Der ewige Zweite?

Das in Azure verfügbare DBaaS-Portfolio ist nicht ganz so breit aufgefächert wie bei AWS (Abbildung 3). Das Fundament des Produktes bildet naturgemäß Microsofts SQL Server, der jetzt auch in der gerade vorgestellten Version 2022 vorliegt. Gleich drei Produkte hat Azure im Angebot, die den SQL Server in verschiedenen Spielarten bereitstellen.

Den Einstieg bildet dabei Azure SQL Database. Hierbei handelt es nicht mehr um klassisches DBaaS, sondern ähnlich wie bei Azure Aurora eher um eine Art Platform as a Service (PaaS). Wer sich bei Azure SQL einmietet, bekommt keine eigene Instanz mit einer eigenen Datenbank, sondern nutzt einen vom Hersteller betriebenen Datenbankdienst. Dessen Kern ist zwar eine SQL-Datenbank, doch diese hat Microsoft um diverse Cloud-typische Funktionen erweitert. Azure SQL Database kann beispielsweise den eigenen Speicher in die Breite skalieren. Um Redundanz und Hochverfügbarkeit kümmert sich ebenso der Anbieter, den Ausfall einzelner Instanzen werden Anwender in der Regel gar nicht bemerken. Obendrein hat Microsoft seinen gehosteten Dienst um HTAP-Fähigkeiten erweitert (Hybrid Transactional and Analytics Processing); die Datenbank eignet sich laut Hersteller also besonders für Analysefunktionen. Dabei sollte man aber stets im Hinterkopf behalten, dass Microsoft SQL Server eine normale relationale Datenbank ist.

Wer es etwas persönlicher will, greift stattdessen zu Microsofts Azure SQL Managed Instance, die in etwa AWS RDS entspricht. Man erhält also eine eigene, von Microsoft vollständig verwaltete Instanz mit darin installiertem MS SQL Server. Der direkte Zugriff auf die Instanz ist Anwendern allerdings verwehrt, sie erhalten wie üblich nur die Zugangsdaten zur Datenbank, nicht jedoch zum System. Auch die Azure SQL Managed Instance lässt sich hybrid betreiben, sodass beispielsweise Inhalte einer lokalen Datenbank „beinahe in Echtzeit“ in die Cloud repliziert werden. Darüber hinaus hält sich das Feature-Feuerwerk allerdings in Grenzen: Viel mehr als der

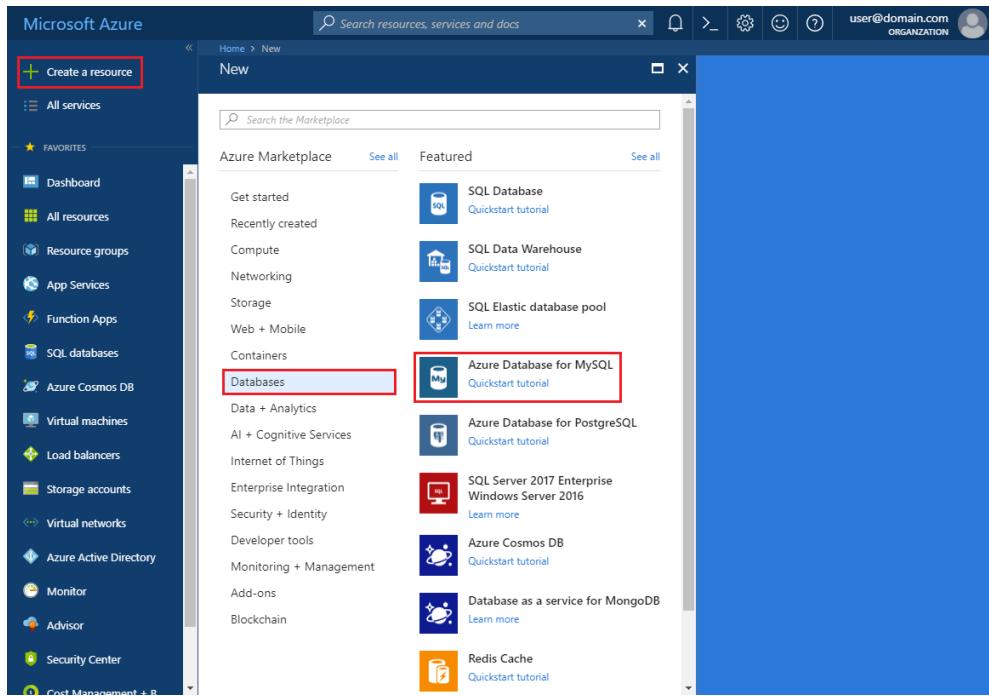


Figure 30.3.: Azure hat in seinem Portfolio einige Datenbanken, reicht aber an das AWS-Angebot noch nicht heran. Jedoch finden Admins hier alles, was für den alltäglichen Bedarf benötigt wird. Quelle: Microsoft

klassische SQL Server beherrscht der Dienst nicht.

Ähnliches gilt für das dritte Produkt der Azure-SQL-Familie, den SQL Server on Virtual Machines. Das funktioniert im Kern wie die SQL Managed Instance, hier erhält der Administrator aber vollen Zugriff auf die darunterliegende VM. Es simuliert also ein IaaS mit aufgepfropfter Datenbank. Das Management ist allerdings beschränkt: Nur wenn der IaaS Azure Agent installiert wird, übernimmt der Hersteller im Namen und Auftrag des Admins noch grundlegende Dienste für die Pflege der MS-SQL-Server-Instanz. Die verfügbaren SLAs für diese Art des Betriebs unterscheiden sich von denen der anderen beiden Produkte der Azure-SQL-Familie erheblich.

30.8. Auch Open Source ist möglich

Bekanntlich richtet sich Azure längst nicht mehr nur an Windows-Anwender und die, die Windows-Server in der Cloud betreiben wollen. Schon vor Jahren hat Microsoft seine Liebe zu Linux entdeckt und bewirbt sie offensiv. Für Azure pflegt der Konzern mittlerweile gar eine eigene Linux-Distribution. Da dürfen beim Thema Managed Database die Vertreter der wichtigsten Open-Source-Datenbanken selbstverständlich nicht fehlen. Obendrein ist nachvollziehbar, dass Azure auch als Deployment-Ziel für diejenigen attraktiv sein soll, die einen Webshop, eine WordPress-Instanz oder ein Forum betreiben wollen. Praktisch sämtliche freie Software für diese Aufgaben setzt jedoch nicht auf SQL Server, sondern auf MariaDB, MySQL und manchmal PostgreSQL.

Entsprechend geht Azure mit DBaaS-Angeboten für diese drei OSS-Vertreter an den Start. Wenig kreativ heißen diese „Azure Database for“ MySQL, PostgreSQL und MariaDB. Beim Funktionsumfang liefern sie genau das Erwartete: eine gemanagte Instanz des jeweiligen Dienstes, die Redundanz und das Einrichten der Hochverfügbarkeit

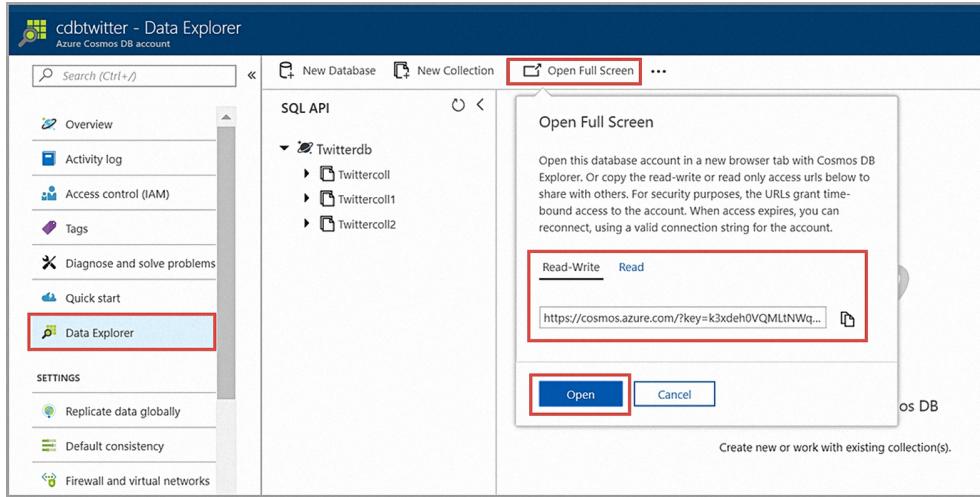


Figure 30.4.: Cosmos DB ist Microsofts Universal Datenbank für spezielle Fälle, die sich für Dokumente ebenso eignet wie für Analysedaten und zudem ein eigenes Anzeigewerkzeug mitbringt. Quelle: Microsoft

übernimmt und ansonsten die bekannte SQL-Schnittstelle exponiert. PostgreSQL hat Microsoft in der eigenen Cloud sogar noch etwas aufgebohrt: Auf der Dienstebene Hyperscale lässt es sich – anders als die MySQL- und MariaDB-Pendants – hybrid betreiben und in die Breite skalieren.

30.9. Cosmos für In-Memory und Cache für Redis

Die letzten beiden DBaaS-Angebote bedienen spezielle Anforderungen, doch erzwingt der Cloud-ready-Ansatz ja gerade deren Entstehung regelmäßig. Azure Cosmos DB ist ein etwas eigenartig anmutender Hybrid, eigentlich eine nicht relationale Datenbank, aber mit PostgreSQL-Schnittstelle und gedacht als Datenbank für Dokumente und Wide-Column-Anwendungen sowie als Key-Value-Speicher oder als Datenbank für Graphing. Es handelt sich also um eine Art Eier legende Wollmilchsau, die der Hersteller in höchsten Tönen lobt. Das Produkt biete beispiellose Performance aufgrund seiner Architektur und mit seinen vielfältigen Schnittstellen PostgreSQL-, MongoDB- und Apache-Cassandra-Kompatibilität.

In der Tat ist das Feature-Set erstaunlich. Wie bei den anderen Hosted-Angeboten übernimmt auch hier Microsoft das Management und dem Anwender bleibt nur, die nun gegebene Datenbank sinnvoll mit Daten zu betanken. Weshalb der Ansatz einer Lösung für viele Schnittstellen allerdings besser sein soll als etwa der AWS-Ansatz, für die einzelnen Schnittstellen eigene Produkte zu definieren, darauf bleibt Microsoft die Antwort schuldig. Immerhin liefert der Hersteller für Cosmos DB ein eigenes Werkzeug mit, das die Analyse der Daten in der Datenbank erleichtert (Abbildung 4). Azure Cache for Redis schließlich ist ein Key-Value Store, der als In-Memory-Datenbank für Redis arbeitet und so dessen Performance steigern soll. Praktisch handelt es sich also um ein direktes Gegenstück zu Amazon MemoryDB.

30.10. Auch Azure hilft bei der Migration

Wie Amazon will auch Microsoft in Sachen Datenbankmigration in die Cloud nichts dem Zufall überlassen. Hierfür stellt der Anbieter ein eigenes Tool bereit, das sich

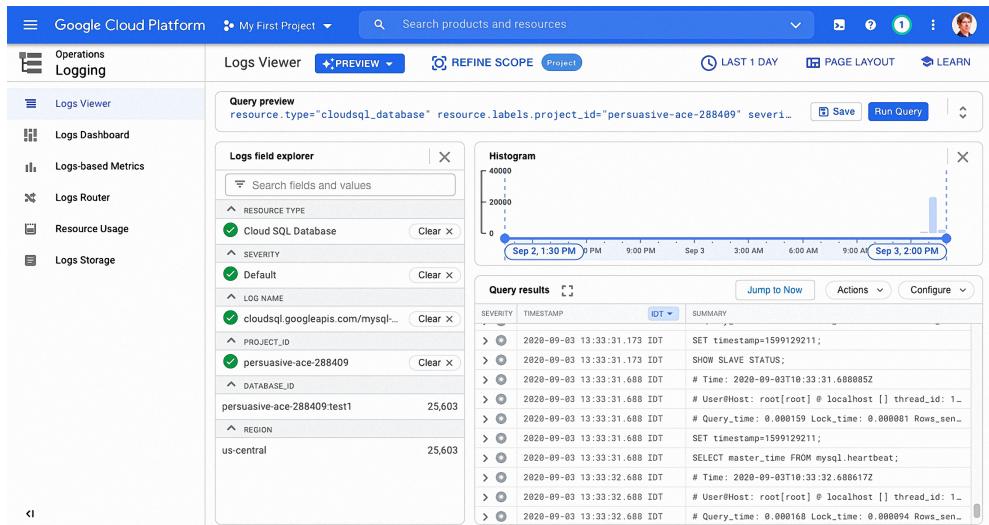


Figure 30.5.: Google GCP erfindet einerseits einen eigenen SQL-Dialekt, unterstützt bei Cloud SQL aber auch klassische Schnittstellen wie MySQL oder PostgreSQL. Quelle: Google

allerdings nur auf Azure Database for MariaDB bezieht. Logisch: MS SQL Server lässt sich unmittelbar per Spiegelung in die Cloud kopieren. Nutzer von PostgreSQL oder MySQL schauen in die Röhre.

Insgesamt präsentiert sich Azures Datenbankangebot auf stabilem Niveau und dürfte für die meisten Einsatzzwecke eine passende Lösung parat haben. Ganz so umfangreich wie bei AWS ist das Angebot aber nicht. Wer also für eine Nischenfunktion eine Datenbank wie eine Zeitreihendatenbank braucht, betreibt diese entweder selbst oder braucht eine andere Plattform.

30.11. Google: im Schatten von AWS und Azure

Wer das Cloud-Geschäft schon eine Weile verfolgt, weiß, dass sich Google mit der eigenen Position darin lange schwergetan hat. Heute reiht GCP sich neben Azure und AWS in die Riege der Hyperscaler ein. Beim Thema Cloud-Datenbanken liegt Google zumindest bei der Breite des Angebots rein zahlenmäßig mit Amazon auf Augenhöhe, denn das DBaaS-Portfolio aus Mountain View umfasst zehn verschiedene Dienste.

Bei den relationalen Datenbanken sticht Google AWS dabei sogar aus. Zunächst steht Kunden in Form von Cloud SQL eine von Google betriebene Datenbank zur Verfügung. Die arbeitet intern relational und bietet Interfaces für MySQL, SQL Server und PostgreSQL (Abbildung 5). Zu Cloud SQL gehört auch ein eigener Dienst für Migrationen. Über die Datastream-Funktion lassen sich Cloud-SQL-Instanzen untereinander mit niedriger Latenz replizieren.

Ein Unikum im Google-Portfolio ist Cloud Spanner, eine Cloud-native Datenbank mit SQL-Schnittstelle. Sie verspricht georeplizierte Set-ups bei einer Verfügbarkeit von 99,999 Prozent, soll dabei hochperformant sein und auch die Migration von Datenbanken wie Oracle oder DynamoDB erlauben. Allerdings nutzt Cloud Spanner mit Google Standard SQL einen Dialekt, den nur wenige Anwendungen ab Werk beherrschen. Hier zeigt sich, dass Cloud Spanner sich nicht an Allerweltskunden richtet, sondern an Konzerne, die ihre Applikationen selbst schreiben und eine Cloud-basierte Enterprise-Datenbank benötigen. An dieser Stelle schlägt der schon mehrmals erwähnte Lock-in-Effekt stärker zu: Aus einer bei Azure betriebenen MySQL-Instanz kann man

immerhin noch in eine MySQL-Instanz bei AWS migrieren. Wer sich jedoch einmal auf Cloud Spanner eingeschossen hat, müsste bei einer Migration die Datenbank konvertieren. Das dürfte gerade bei großen, historisch gewachsenen Datenbanken praktisch unmöglich sein.

Immerhin bietet Google für Cloud Spanner das PostgreSQL-Interface an. Das arbeitet als eine Art Proxy und verhindert, dass Endkunden ihre PostgreSQL-fähigen Anwendungen umstricken müssen. Informationen über den Grad der Kompatibilität bietet Google allerdings nicht.

30.12. PostgreSQL als Ersatz

Für diejenigen Administratoren oder Anwender, die mit Cloud Spanner wegen seiner proprietären Schnittstelle nicht zureckkommen, hat Google AlloyDB for PostgreSQL im Angebot. Auch AlloyDB ist ein Google-eigenes Produkt und steht unter einer nicht freien Lizenz. Es schneidet laut Hersteller etwas schlechter als Cloud Spanner in den Kategorien Skalierbarkeit und Performance ab, bietet im Gegenzug aber eine PostgreSQL-Schnittstelle. Genau lässt Google sich aber auch hier nicht in die Karten gucken, sodass unklar bleibt, ob ein unter dem Namen AlloyDB auf GitHub firmierendes Projekt möglicherweise ein historischer Vorgänger des Google-Produktes ist. In jenem Git-Verzeichnis stammt der letzte Commit aus dem Jahr 2019. Fest dürfte aber stehen, dass auch AlloyDB im Kern kein PostgreSQL nutzt, sondern vermutlich einen Key-Value-Speicher mit einer PostgreSQL-Kompatibilitätschnittstelle. Wo genau hier die Unterschiede zu Cloud Spanner mit seinem PostgreSQL-Frontend liegen, verrät der Hersteller ebenfalls nicht. In Summe riecht AlloyDB mithin etwas „fishy“ und könnte zu jenen Produkten gehören, die Google – wie der Hersteller es gern und häufig tut – irgendwann zugunsten von Cloud Spanner auslaufen lässt.

Als relationale Alternative zu AWS Redshift bringt Google zudem das Produkt BigQuery an den Start. Das dient ebenfalls dem Aufbau von Data Warehouses zu Analysezwecken und dem Verwalten betrieblicher Echtzeitdaten. Wer aus der Oracle-Ecke kommt, kann beim Anbieter zudem eine „Bare-Metal-Lösung für Oracle“ bekommen, wobei es sich dabei um ein Built-to-order-Produkt handeln dürfte, das nicht von der Stange kommt.

30.13. Noch weitere DB-Arten im Angebot

Auch bei den anderen Datenbanktypen hat Google etwas zu bieten. So vermarktet das Unternehmen unter dem Namen Cloud Bigtable einen Key-Value-Speicher als gemanagten Dienst. Dies bewirbt der Hersteller wie Azure mit einer Verfügbarkeit von 99,999 Prozent, hoher Skalierbarkeit und der Fähigkeit, bis zu fünf Milliarden Anfragen pro Sekunde global verteilt zu verarbeiten. Das soll all jene Kunden ansprechen, die von HBase oder Cassandra in die Cloud migrieren wollen.

Firestore richtet sich an dieselben Kunden wie Microsofts Cosmos DB oder Amazons DocumentDB. Es geht also um die Verarbeitung von viel vorformatiertem Text. In der Praxis dürfte das vor allem IoT-Anwendungen betreffen, die mit ihrem Mutterschiff telefonieren und mit diesem zu Informationszwecken große Datenmengen austauschen. Hier trifft man die Art von Inhalt am häufigsten an, für die Dokumentendatenbanken gemacht sind.

Wer Daten zwischen mehreren Systemen in Echtzeit synchronisieren und speichern muss, findet im Google-Fundus mit Firebase Realtime eine weitere typische Dokumentendatenbank.

Natürlich darf auch eine In-Memory-Datenbank nicht fehlen. Im Kern steckt in Googles Memorystore Redis mit Memcached für den schnellen Zugriff auf gecachte

Daten, sodass jede mit Redis kompatible App auch mit Memorystore funktionieren sollte.

Wer stattdessen auf MongoDB festgelegt ist, findet bei Google mit MongoDB Atlas die einzige herstellereigene Datenbank, die der Plattformbetreiber mitvermarktet. Weil es sich um MongoDB handelt, sind zur Funktionalität des gemanagten Dienstes an dieser Stelle weitere Details entbehrlich – es sei auf die diversen MongoDB-Artikel in früheren iX-Ausgaben verwiesen.

30.14. Andere Datenbankanbieter in der Cloud

MongoDB in der Google Cloud bietet eine schöne Überleitung zu einem letzten, plattformübergreifenden Thema. Gemanagte Datenbanken findet man in Azure, AWS und GCP nämlich nicht nur von den Plattformbetreibern selbst. Es gehört zu deren Geschäftsmodell, dass sie Partner ins Boot holen. Denn bei aller Größe wird es weder Google noch Amazon noch Microsoft gelingen, genügend Experten für jede auf dem Markt existierende kommerzielle Datenbank zu finden. Der in diesem Heft auf Seite 42 erschienene Artikel zu Cloud-nativen Datenbanken macht das schnell deutlich [1]. Denn von den dort vorgestellten fünf Kandidaten – Citus, CockroachDB, TiDB, Vitess und YugabyteDB – findet sich bei keinem der drei Hyperscaler auch nur ein Wort.

Trotzdem lässt sich beispielsweise TiDB als direkt vom Hersteller PingCAP angebotener Cloud-Dienst in AWS nutzen. Beispiele für ganz ähnlich gelagerte Fälle dürfte es Tausende geben. Wer also mit dem Gedanken spielt, die eigene Workload in die Cloud zu migrieren, sollte das im Hinterkopf behalten: Auch wenn die aktuell genutzte DB in der Liste der unterstützten DBaaS-Angebote fehlt, bedeutet das nicht automatisch, dass der gemanagte Betrieb der Datenbank bei AWS und Co. unmöglich ist. Allerdings sind dann die Dritthersteller zuständig und auch verantwortlich.

30.15. Fazit

Im Database-as-a-Service-Duell der Hyperscaler kann sich keine Plattform als klarer Sieger etablieren. Wer halbwegs normale Datenbankanforderungen hat, wird in der Regel an einer relationalen Datenbank interessiert sein, die automatisch durch die Plattform gesichert wird und nach Möglichkeit das Skalieren in die Breite und Hochverfügbarkeit bietet. Sowohl AWS als auch Azure und GCP decken diese Anforderungen bereits mit den Grunddiensten ihrer DBaaS-Angebote ab.

Bei den speziellen Diensten ergibt sich jedoch ein differenzierteres Bild. Googles Cloud Spanner etwa dürfte am Markt einzigartig sein, lohnt sich aber nur für Unternehmen, die wirklich große verteilte Datenbanken mit hohen Performanceansprüchen benötigen. Allerdings birgt die Nutzung des Google-eigenen SQL-Dialekts auch die Gefahr eines Lock-in.

Darüber hinaus gleichen sich die Portfolios weitgehend. Eine NoSQL-Datenbank, eine Dokumentendatenbank und ein Data Warehouse findet sich bei allen Anbietern ebenso wie In-Memory-Datenbanken für die Verwendung mit Redis. Es gilt insofern das bei Cloud-Diensten so häufig Gesagte: Technisch können Nutzer bei keinem der Anbieter etwas falsch machen. Am Ende entscheidet entweder die bereits zugunsten eines Anbieters getroffene Plattformauswahl auch über die DBaaS-Dienste oder aber der Administrator lässt das Thema DBaaS bei der Wahl einer Plattform in seine Überlegungen einfließen. Wie die Tabelle zeigt, unterscheiden sich die verschiedenen Angebote zumindest finanziell deutlich, sodass sich hier womöglich ein Anhaltspunkt ergibt. (avr@ix.de)

SQL-Datenbanken aus der Cloud – Angebote der Hyperscaler									
Anbieter	AWS				Azure				
Produktname	Aurora	RDS	Redshift	SQL	Managed SQL Instance	SQL Server on virtual Instance	PostgreSQL	MariaDB	Cloud SQL
Serverless	✓	–	–	✓	–	–	–	–	–
SQL-Dialekt	MySQL, PostgreSQL	MySQL, MariaDB, PostgreSQL, MS SQL	Redshift SQL (PostgreSQL-Derivat, teilkompatibel)	MS SQL	MS SQL	MS SQL	PostgreSQL	MariaDB	Cloud SQL
als Managed Service	✓	✓	✓	✓	✓	(✓, teilweise)	✓	✓	✓
Preis ²	2,506 US-\$	4,153 US-\$	8,208 US-\$	8,832 €	6,18 €	6,75 €	1,631 €	1,604 €	2,53 US-\$

¹ mit Konnektor; ² für Instanz mit folgender Konfiguration: 6 vCPUs, 64 oder 128 GByte RAM, Standort Europa oder Deutschland zzgl. Storage

Angebote der Hyperscaler									
	Azure					Google Cloud			
Redshift	SQL	Managed SQL Instance	SQL Server on virtual Instance	PostgreSQL	MariaDB	Cloud SQL	Cloud Spanner	AlloyDB	
Redshift	✓	–	–	–	–	✓	–	–	–
Redshift SQL (PostgreSQL-Derivat, teilkompatibel)	MS SQL	MS SQL	MS SQL	PostgreSQL	MariaDB	MS SQL, MySQL, PostgreSQL	Cloud SQL (PostgreSQL ¹)	PostgreSQL	Cloud SQL
	✓	✓	(✓, teilweise)	✓	✓	✓	✓	✓	✓
208 US-\$	8,832 €	6,18 €	6,75 €	1,631 €	1,604 €	2,53 US-\$	1,08 US-\$	2,49 US-\$	Cloud SQL

duration: 6 vCPUs, 64 oder 128 GByte RAM, Standort Europa oder Deutschland, pro Stunde oder vergleichbar, Preise zum Teil

Figure 30.6.: SQL-Datenbanken aus der Cloud – Angebote der Hyperscaler

Key-Value Stores aus der Cloud – Angebote der Hyperscaler			
Anbieter	AWS	Azure	Google Cloud
Produktnname	DynamoDB	Cosmos DB	Bigtable
Serverless	✓	✓	✓
Kompatibilität	Cassandra	Cassandra	Cassandra, HBase
als Managed Service	✓	✓	✓
Abrechnungsfaktoren ¹	on demand oder managed, danach gestaffelt nach Reads/Writes, Streams, ingress/outgoing Traffic, benutztem Storage, Menge der globalen Tabellen; Gebühren für den Export zu anderen Diensten wie S3 abhängig davon, in welcher Zone die Instanz läuft	Zone, Autoscale oder Standard Scale in Request Units (RU), zuzüglich fixer Gebühren für Traffic und genutzten Speicher; 100 RU kosten 0,0008 Euro pro Stunde	Region, dann Nodes eines Clusters (Shards) mit 0,65 US-Cent pro Stunde, zuzüglich dynamisch genutztem RAM und Traffic bei auf mehrere Regionen verteilten Nodes

¹je nach Instanztyp

Figure 30.7.: Key-Value Stores aus der Cloud – Angebote der Hyperscaler

30.16. Quellen

- Martin G. Loschwitz; Fünf Cloud-native SQL-Datenbanken; iX 1/2023, S. 42
- Weiter Informationen zu den DBaaS-Angeboten der Hyperscaler: ix.de/zkze

30.17. Tasks

- Translation
- Improvements
- Further readings
- Presentation

31. Fünf Cloud-native SQL-Datenbanken

31.1. Einführung

Auch in modernen Cloud-ready-Anwendungen geht ohne Datenbanken fast nichts. Hier mischen Cloud-native Datenbanken gerade den Markt auf. iX stellt fünf Vertreter mit SQL-Schnittstelle vor: YugabyteDB, Citus, TiDB, Vitess und CockroachDB.

- Datenbanken spielen als zentraler persistenter Speicher für das Gros der Nutzdaten auch in modernen Cloud-Set-ups eine wichtige Rolle.
- In der Cloud sind Datenbanken jedoch mit anderen Anforderungen konfrontiert und müssen mehr Funktionen bieten: Skalierbarkeit etwa ist ebenso wichtig wie die Fähigkeit, auf viele Knoten einer Datenbank parallel schreibend zuzugreifen.
- Am Markt etabliert sich derzeit mit den Cloud-nativen Datenbanken ein neuer Datenbanktyp: Die behaupten, bereits ab Werk an die Voraussetzungen der Cloud angepasst zu sein und daher in der Cloud besonders gut zu funktionieren.

Datenbanken sind ein fester Bestandteil moderner Cloud-Anwendungen. Derzeit schicken sich die Cloud-nativen Datenbanken an, den Platzhirschen MySQL und Co. Konkurrenz zu machen. Doch was ist eine solche Cloud-native Datenbank überhaupt, und welche Werkzeuge stehen zur Verfügung?

State ist ein Zauberwort, mit dem sich Sysadmins in eine lebhafte Diskussion verwickeln lassen. Stateful und Stateless sind dabei zwei Begriffe, die sich früher üblicherweise auf die Art und Weise bezogen, wie Anwendungen Netzwerkverbindungen handhaben. Heute kommt der Begriff State aber regelmäßig auch im Kontext von Daten vor, mit denen Anwendungen hantieren. Und wie üblich sind sich Administratoren und Entwickler keinesfalls einig, ob State bei Anwendungen grundsätzlich eine gute Sache oder eine Ausgeburt der Hölle ist.

Zumindest im Kontext von Clouds scheint das Thema aber durch: Hier gilt heute das Mantra, dass Anwendungen so wenig Daten wie möglich selbst halten sollen, weil jede Instanz jedes Dienstes grundsätzlich als Wegwerfartikel gilt, der sich jederzeit woanders wieder aus dem Boden stampfen lassen muss. In Clouds und Cloud-nativen Set-ups müssen die Daten daher an zentraler Stelle gelagert sein, sodass alle berechtigten Dienste und Komponenten darauf zugreifen können. Üblicherweise realisieren das die meisten Installationen über Datenbanken. Was vielen Entwicklern und Administratoren allerdings im Verlauf der vergangenen Jahre erst langsam klar geworden ist: Das neue Einsatzszenario in der Cloud verändert natürlich auch die Voraussetzungen, die Datenbanken erfüllen müssen.

Früher war es Usus, eine zentrale Datenbank in einem Set-up zu haben, die mit den Mitteln klassischer Hochverfügbarkeit auf Redundanz getrimmt wurde. Wer gerade zitternd an Pacemaker und Co. denkt, weiß zumindest für Linux, wovon die Rede ist. Und der nicht besonders beliebte Cluster-Manager für Linux kam nicht alleine daher, sondern hatte üblicherweise noch Werkzeuge wie DRBD dabei, die die Replikation von Daten im Hintergrund sicherstellen. Skalierbarkeit war nur in die Höhe ein Thema: Weil sich viele Unternehmen beim Design ihrer Datenbanken so wenig Mühe gaben, dass diese absurde Rechenkapazitäten verschlangen, wurden oft Rufe nach neuer,

potenterer Hardware laut. Die Last stattdessen mit mehreren Datenbank-Backends abzufedern, hat sich als Prinzip in konventionellen Umgebungen nie etabliert.

In Clouds ist das naturgemäß anders: Hier herrscht die Economy of Scale, ein wirtschaftlicher Betrieb erfordert, dass zu jedem Zeitpunkt jede Komponente eines Set-ups in die Breite erweiterbar ist. Das schafft mehr Bandbreite und nutzt der Performance. Es bewirkt aber auch, dass sich Set-ups sinnvoll skalieren lassen, etwa wenn später eine größere Datenbank als ursprünglich geplant benötigt wird.

31.2. Cloud-native Datenbanken existieren

Am Markt der Datenbanken stiftet das einige Unruhe. PostgreSQL, MySQL, das mit MySQL noch immer weitgehend kompatible MariaDB und andere etablierte Datenbanken würden gern besser in die Cloud passen, scheitern meist aber schon an den technischen Grundvoraussetzungen. Gleichzeitig zieht am Horizont so etwas wie eine neue Generation von Datenbanken auf, die zwar von außen (oft) wie MySQL oder PostgreSQL aussehen, hinter den Kulissen aber völlig anders arbeiten. Hier sind die Voraussetzungen der Cloud in der Architektur der Datenbanken berücksichtigt, sodass diese sich perfekt beispielsweise in Mikroarchitekturanwendungen in Containerumgebungen einfügen. Ihnen allen ist gemein, dass sie nahtlos in die Breite skalieren können, ab Werk keinen Single Point of Failure mehr haben und dass sie die etablierten Datenbanken in Sachen Performance regelmäßig alt aussehen lassen. Grund genug, sich ein paar Vertreter dieser Gattung näher anzusehen: iX stellt hier die fünf Cloud-nativen Datenbanken YugabyteDB, Citus, TiDB, Vitess sowie CockroachDB vor und vergleicht sie miteinander.

Sämtliche Probanden haben oder emulieren SQL-Schnittstellen. Sie folgen dem klassischen Schema der relationalen Datenbanksysteme. Datenbanken mit diesem Protokoll sind nicht die einzigen Cloud-nativen Datenbanken. Parallel dazu entwickelt sich ein ganzer Zweig von Datenbanken, die andere Formate und Protokolle nutzen, etwa Zeitreihendatenbanken oder Key-Value-Speicher. Üblicherweise spricht man zur groben Unterscheidung von SQL- sowie NoSQL-Datenbanken. Dieser Text geht ausschließlich auf Datenbanken ein, die nach außen SQL emulieren. Denn auch wenn sich die Architektur von Datenbanken unter der Haube ändert, sind die Anforderungen der meisten Programme an eine Datenbank heute SQL-Fähigkeit und ein relationales Modell auf Basis von Tabellen, Zeilen und Reihen.

31.3. YugabyteDB: Sharding und viel Abstraktion

YugabyteDB haben wir bereits in einem Artikel in iX 5/2022 vorgestellt [1], trotzdem sollen an dieser Stelle die wichtigsten Details des Designs nicht fehlen. Was bei einem Blick auf sein Architekturdiagramm (Abbildung 1) sofort ins Auge fällt: YugabyteDB exponiert zur Außenwelt hin zwar einen SQL-Dialekt, doch findet sich von einer klassischen relationalen Datenbank unter der Haube keine Spur. Stattdessen setzt das Produkt zum Speichern von Daten auf eine eigene Datenbank namens DocDB. Deren Kern indes basiert auf RocksDB, dem von Facebook konzipierten, extrem schnellen Key-Value-Speicher. Anders formuliert: Auch wenn eine Applikation Daten in Yugabyte über eine SQL-Schnittstelle ablegt, landen diese letztlich als Key-Value-Paar in RocksDB. Dass die Datenbank zur Außenwelt hin nicht nur SQL spricht, sondern auch Schnittstellen für die Protokolle der Key-Value-Speicher Redis und Cassandra bereitstellt, liegt da auf der Hand.

Wie fast alle verteilten Lösungen arbeitet auch YugabyteDB im Hintergrund auf der Ebene der DocDB mit Sharding. Sämtliche Probanden in diesem Test kümmern sich um das Verteilen ihrer Shards komplett autark. Der Administrator greift nur ein, um

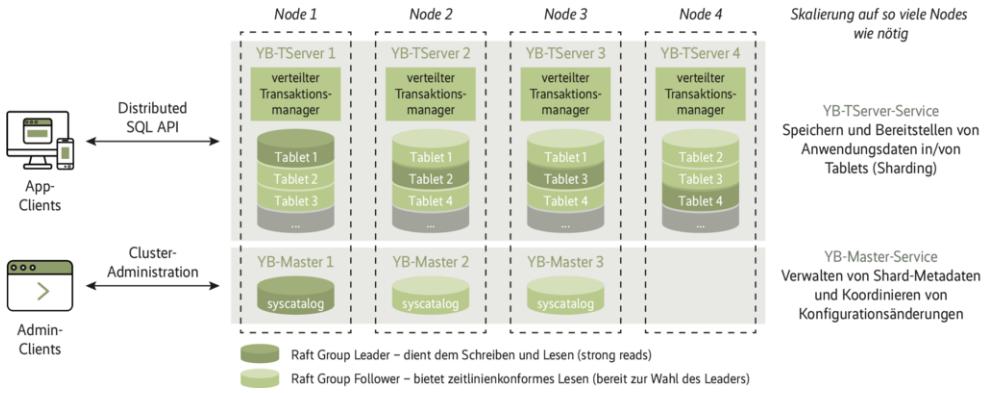


Figure 31.1.: YugabyteDB ist ein Key-Value Store, der mehrere Standardschnittstellen zur Außenwelt exponiert, unter der Haube massiv auf Sharding setzt und somit typisch für die Klasse der skalierbaren Cloud-nativen Datenbanken ist. Quelle: Yugabyte

neuen Speicherplatz zur Verfügung zu stellen, falls der alte ausgeht. Bei YugabyteDB heißen Shards Tablets.

Eingehende Daten verteilt die Software gleichmäßig über alle im Set-up vorhandenen DocDB-Instanzen; dabei ist jede Zeile im klassischen SQL-Denken ein eigener Key-Value-Eintrag in einer DocDB. Für die Verteilung der Shards sind die sogenannten TServer zuständig. Auf jedem Knoten läuft zudem eine Master-Instanz. Sie verwaltet die anfallenden Metadaten und versetzt YugabyteDB in die Lage, aus den Key-Value-Einträgen in den Tablets über sämtliche Datenbanken hinweg wieder zusammenhängende Ergebnisse für die Antwort auf eine SQL-Anfrage zu generieren. Die gesamte Magie bleibt den Clients allerdings verborgen. Die Entwickler geben an, dass ihre Software weitgehend mit dem PostgreSQL-Dialekt kompatibel ist. Die wenigen nicht implementierten Funktionen sind in der Dokumentation aufgelistet. Sie betreffen aber Features, die nur in wenigen Set-ups tatsächlich in Verwendung sein dürften, sodass sich YugabyteDB in den meisten Fällen als Drop-in-Replacement für PostgreSQL nutzen lässt.

31.4. Raft für den Konsens

Wie alle verteilten Systeme steht auch YugabyteDB vor der Herausforderung, einen konsistenten Zustand aller Daten über alle Knoten hinweg sicherzustellen. Darin enthalten sind freilich auch Situationen, in denen ein Cluster wegen Netzwerkproblemen in mehrere Partitionen zerbricht. Die meisten Produkte am Markt setzen hier auf den Konsensalgorithmus Paxos, YugabyteDB nutzt das alternative Raft-Protokoll. Die Datenbank ist implizit redundant, sodass der komplette Ausfall einer Instanz keinen Datenverlust nach sich zieht. Raft repliziert dafür auf der Ebene der TServer seine Shards beziehungsweise Tablets, wobei für jedes Tablet im gesamten Cluster stets ein primärer TServer und beliebig viele sekundäre TServer aktiv sind. Nur die aktive Instanz lässt sich für den Zugriff nutzen.

Weil es in einem Cluster beliebig viele Shards geben kann, bündelt YugabyteDB beim Zugriff auf Daten durch diese Art der Verteilung die Bandbreite aller verfügbaren Backends. In Sachen Performance hat das Produkt daher ausreichend Reserven.

31.5. Gute Cloud-Integration

Technisch präsentiert sich Yugabytes Datenbank auf der Höhe der Zeit, doch stellt sich auch die Frage nach der Integration in Cloud-Umgebungen und verteilte Systeme. Grundsätzlich gilt, dass gerade in eher konventionellen Set-ups nachempfundenen Umgebungen Entwickler mit YugabyteDB alleine kaum glücklich werden. Denn die meisten Clients, die SQL für den Zugriff auf eine Datenbank nutzen, beherrschen schlicht die Funktion nicht, mehrere IP-Adressen für den Server zu verwalten. Ein Loadbalancer ist also nötig, was sich gerade in Cloud-ready-Umgebungen auf Containerbasis etwa mittels Kubernetes aber leicht realisieren lässt. Wer es etwas umfangreicher will, kann auf externe Produkte wie Istio zugreifen, das Clients automatisch auf eines der Datenbank-Frontends leitet.

Da passt es gut ins Konzept, dass YugabyteDB sich gerade in Kubernetes besonders leicht in Betrieb nehmen lässt. Fertige Container mitsamt einer fertigen Integration für Kubernetes existieren, sodass sich YugabyteDB aus Kubernetes heraus wie ein echter First-Class Citizen steuern lässt.

In Summe präsentiert sich YugabyteDB aufgeräumt und komplett: Skalierte und auch skalierbare Datenbanken sind der Kern der Lösung und funktionieren gut und performant. Die Protokolle zur Anbindung an die Außenwelt sind vielfältig und decken SQL- wie Nicht-SQL-Bedürfnisse gut ab. Hinsichtlich der Integration in moderne verteilte Umgebungen und gerade im Hinblick auf Kubernetes gibt es nur wenig zu bemängeln.

31.6. Citus: auch PostgreSQL, aber anders

Der zweite Kandidat im Vergleich ist Citus (Abbildung 2), das sich von YugabyteDB in etlichen Punkten zentral unterscheidet. Um die Motivation hinter und den Werdegang von Citus zu verstehen, ist ein bisschen PostgreSQL-Geschichte unumgänglich. Anders als andere Datenbanksysteme konnte PostgreSQL nämlich über viele Jahre hinweg nicht mit einer integrierten Replikation aufwarten, die den gleichzeitigen Zugriff auf mehrere synchronisierte PostgreSQL-Instanzen ermöglicht hätte. Für MySQL und später MariaDB gibt es eine solche Umsetzung in Form von Galera seit fast 15 Jahren. Replikationsansätze gibt es für PostgreSQL mehrere, aber keiner davon hat sich als Standard durchgesetzt. Und einige der am Markt verfügbaren Varianten sind Bestandteil gebündelter Produkte oder kostenpflichtig. Das erklärt auch, wieso die Yugabyte-Macher es für einfacher hielten, gleich eine komplett neue Datenbank zu konstruieren und diese „nur“ mit einer PostgreSQL-kompatiblen Schnittstelle zu versehen, statt noch ein weiteres Plug-in für PostgreSQL zu bauen.

31.7. Eigentlich angelegt wie ein Plug-in

Die Citus-Entwickler indes wollten offensichtlich nicht ganz so radikal bei der Wahl ihrer Mittel vorgehen. Die Datenbank ist im Hinblick auf ihre Architektur ein Mittelding zwischen YugabyteDB und klassischen Ansätzen wie Galera für MySQL. Der zentrale Unterschied ist dabei, dass in Citus echtes PostgreSQL zum Einsatz kommt, sodass die Entwickler keine eigene Kompatibilitätsschicht bauen mussten.

Streng genommen handelt es sich bei Citus also gar nicht um eine eigenständige Cloud-native Datenbank. Vielmehr ist Citus eine Erweiterung für PostgreSQL, die Features wie Sharding und die zentrale Zugriffskontrolle für PostgreSQL bereitstellt und daraus eine skalierbare Datenbank konstruiert. Das kann man sich ähnlich wie bei Hadoop vorstellen: Die Worker in Citus speichern eingehende Inhalte ganz regulär in den ihnen zur Seite gestellten lokalen PostgreSQL-Instanzen. Das Sharding erfolgt dabei anhand der in PostgreSQL genutzten Tabellen. Für eine Tabelle beispiel legt

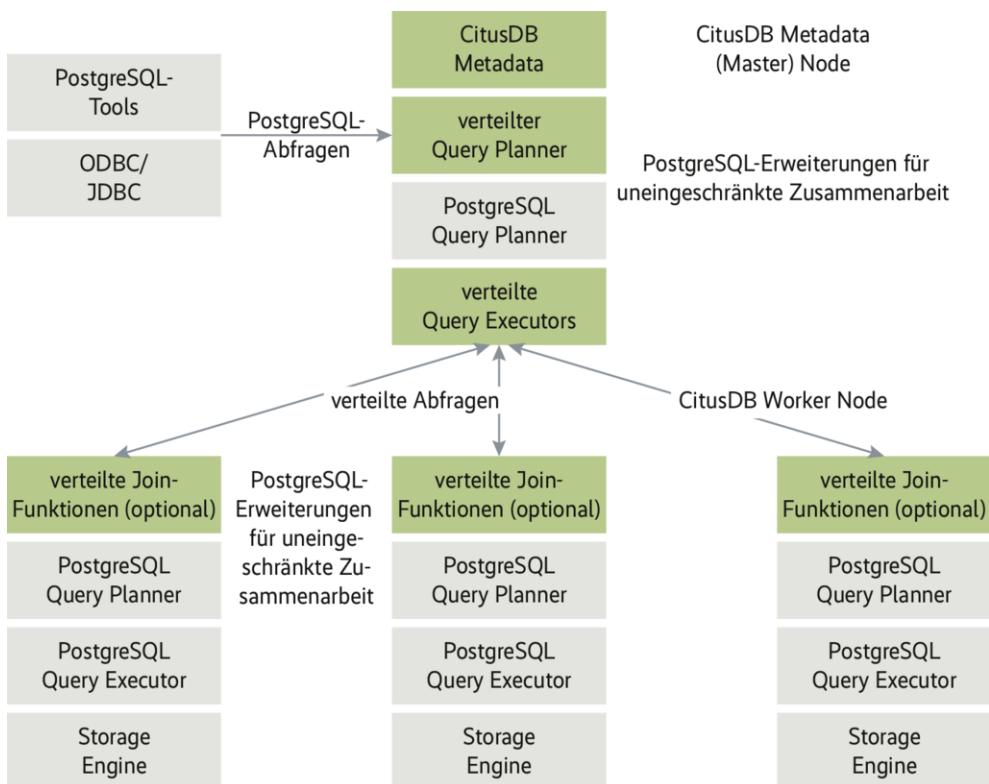


Figure 31.2.: Citus baut als einziges Produkt im Test keinen Key-Value Store mit Interface für verschiedene Protokolle, sondern ist ein Plug-in für PostgreSQL. Quelle: Citus

Citus dabei einmal fest, welcher Knoten administrativ verantwortlich ist und auf welchen Knoten die Replicas einer Tabelle landen sollen. Denn auch für Citus gilt, dass es die Daten intern repliziert und damit vor Ausfällen schützt.

Das Prinzip funktioniert aber nur, wenn es – und hier ähnelt Citus dem Ansatz von YugabyteDB – eine zentrale Instanz gibt, in deren Metadaten verzeichnet ist, welcher Worker für eine bestimmte Tabelle zuständig ist. Und nicht nur das: PostgreSQL kennt für Clients schlicht keine Funktion, sich Daten von unterschiedlichen Storage-Backends zusammenzusuchen und daraus eine einheitliche Antwort auf eine Datenbankabfrage zu basteln. Hier kommt der Dienst ins Spiel, den Citus Coordinator nennt: Er verwaltet die Metadaten und weiß, welche Daten wo liegen. Ein Client stellt seine Anfrage im SQL-Dialekt von PostgreSQL stets an den Coordinator, der die Daten für den Client transparent zusammensucht und als einheitliche Antwort ausgibt. Gleichzeitig regelt der Coordinator auch, dass von jedem angelegten Shard im Hintergrund Kopien für die Redundanz bereitgestellt werden. Lesezugriffe erfolgen dann parallel von allen verfügbaren Replicas.

31.8. Für den Betrieb ist viel Handarbeit nötig

Citus präsentiert sich nicht als so universell und vollständig wie YugabyteDB. Die Dokumentation beispielsweise verweist etliche Male auf die PostgreSQL-Doku. So zum Beispiel, wenn es um die Redundanz des Coordinator-Knotens geht: Den muss sich ein Citus-Administrator nämlich komplett selbst bauen, idealerweise unter Einsatz der in PostgreSQL vorhandenen Streaming-Replikation. Damit lässt sich eine Hot-Stand-by-Instanz des Coordinators anlegen, die im Falle eines Falles einen ausgefallenen Knoten ersetzen kann. Allerdings – und das ist der Haken – nicht automatisch: Falls das gewünscht ist, müsste der Entwickler doch wieder in Istio mit automatisch wechselnden IP-Adressen und ähnlichem Komplikationen umgehen.

Und das ist nicht das einzige Komfortproblem bei Citus. So gibt es für Citus keine vernünftige Integration in typische skalierte Umgebungen wie Kubernetes. Fertige Container-Images fehlen ebenso wie die Integration in Kubernetes selbst. Das ist für eine Software, die sich explizit an skalierte Umgebungen richtet, eigentlich eine Ungeheuerlichkeit. Der Selbermach-Faktor ist hier also einigermaßen hoch, sodass Citus nur eine bedingte Empfehlung bekommt. Empfehlung vor allem deshalb, weil sich die Software im Test bewiesen hat und gut funktionierte.

31.9. Vitess: eigener Unterbau und MySQL zur Abwechslung

PostgreSQL und MySQL rangieren heute auf Augenhöhe, was Funktionsumfang, Zuverlässigkeit und Bedienbarkeit angeht. Trotzdem entscheidet sich das Gros der Softwareanbieter bei der Wahl der Datenbank für MySQL. Kein Wunder also, dass Datenbankentwickler am Markt auch Potenzial für eine skalierbare Datenbank mit MySQL-Schnittstelle sahen, und Vitess dürfte zweifelsohne die bekannteste Implementierung einer solchen Datenbank sein. Die Software hat sich im produktiven Einsatz hinlänglich bewiesen und trug bei YouTube eine ganze Weile die gesamte Datenbanklast.

Architektonisch gleicht Vitess YugabyteDB in vielerlei Hinsicht, wie ein Blick auf die Grafik (Abbildung 3) verdeutlicht. Dreh- und Angelpunkt in Vitess sind die VTTablets. Das sind Dienste, die einem lokal laufenden MySQL zur Seite gestellt werden, dieses aber lediglich als lokalen Datenspeicher nutzen. Anders als bei Citus wird bei Vitess die lokale Datenbank also nicht zum Anwender durchgeschleift. Stattdessen sind die VTTablet-Instanzen dafür zuständig, die einzelnen Shards, die bei Vitess am ehesten mit Tabellen in einer Datenbank vergleichbar sind, über die Grenzen aller Knoten

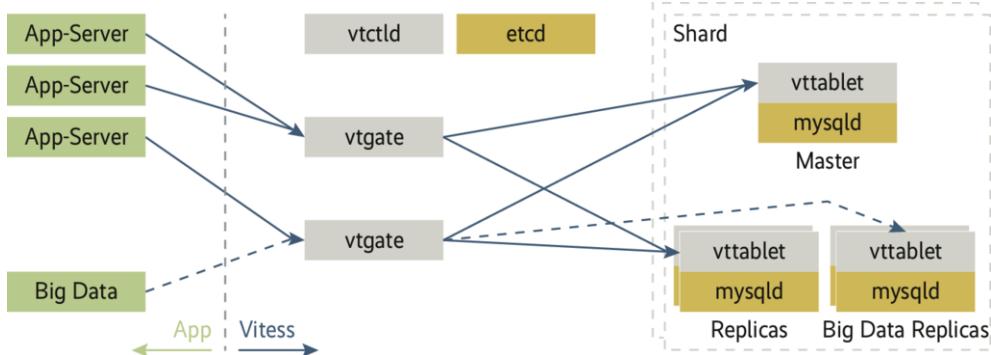


Figure 31.3.: Bei Vitess kommen im Backend zwar echte MySQL-Datenbanken zum Einsatz, doch installiert die Software ein Gateway zu den Clients; die echten MySQL-Datenbanken sehen Clients also nicht. Quelle: Vitess

hinweg zu verteilen und zu verwalten. Was die Arbeit etwas verwirrend macht, ist die eigene Nomenklatur, die die Vitess-Entwickler sich ausgedacht haben: Auf oberster Ebene heißt eine Datenbank (im Normalfall) in Vitess Keyspace. Die Benennung soll aufzeigen, dass es bei Vitess so etwas wie logische Datenbanken gibt – also solche, die der Client von außen zwar als normale MySQL-Datenbank sieht, die im Cluster aber so auf keiner der VTTablet-Instanzen liegen. Ähnlich wie Yugabyte arbeitet Vitess mit einem hohen Maß an Abstraktion.

31.10. Zentrale Gateway-Instanzen steuern den Zugriff

Damit das funktioniert, sind weitere Dienste nötig. Von zentraler Bedeutung ist vor allem das Vitess-Gateway VTGate für Anfragen von Clients. Ihm zur Seite steht ein Topology-Dienst, der sämtliche Vitess-eigenen Metadaten verwaltet. Clients reden mit keiner echten MySQL-Datenbank, sondern mit einer Art Nachbau des MySQL-Serverprotokolls, das im VTGate implementiert ist. Dieses nimmt eingehende Anfragen entgegen, sucht die Information aus den Shards heraus, die über die beliebig vielen VTTablet-Instanzen verteilt sind, und liefert dem Client schließlich eine valide, zusammenhängende Antwort im MySQL-Format.

Mittels CLI oder GUI lassen sich dabei ähnlich wie mit dem MySQL-Client Änderungen an den Vitess-Metadaten vornehmen. Wer beispielsweise eine neue Datenbank, also einen neuen Keyspace, anlegen möchte, tut das mit eben diesen Werkzeugen. Die Anzahl der laufenden Instanzen des Gateways sowie auch des Topology-Dienstes ist dabei nicht begrenzt, sie lassen sich clustern, um Hochverfügbarkeit zu erreichen. Apropos HA: Auch Vitess bietet ab Werk eine interne Replikation, die Datenverlust etwa beim Ausfall einzelner Instanzen verhindert.

Positiv fällt auf, dass das Vitess-Gateway Antworten auf Querys in cleverer Art und Weise kompiliert. Parallel eintreffende identische Querys erledigt das Gateway nicht etliche Male, sondern nimmt stattdessen ein Ergebnis und gibt dieses für alle Abfragen heraus. Droht eine große Datenbank-Query den Dienst in die Knie zu zwingen, gibt es in MySQL derzeit keine zuverlässige Möglichkeit, diesen Vorgang während der Ausführung abzubrechen. Vitess erlaubt das jedoch im VTGateway per Kommandozeilenwerkzeug. Und schließlich führt das VTGateway auch ein sinnvolles Connection Pooling ein: Statt die VTTablets im Hintergrund mit Hunderten und Tausenden gleichzeitig offener Verbindungen zu bombardieren, die alle nur sporadischen Traffic erzeugen, führt Vitess Abfragen im Hintergrund durch wenige offene Verbindungen parallel aus. Das spart Speicher auf den Servern mit laufenden Datenbanken und schont die vorhandenen

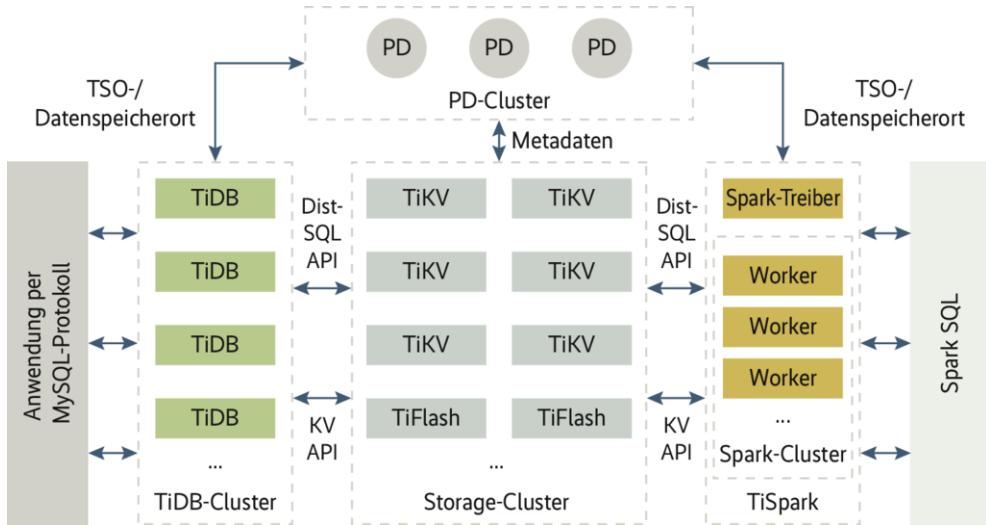


Figure 31.4.: TiDB ist letztlich auch nur ein Key-Value-Speicher, der verschiedene Protokolle exponiert. Bemerkenswert ist, dass es unterschiedliche Storage-Backends gibt, die sich für verschiedene Datentypen besonders gut eignen sollen. Quelle: TiDB

Ressourcen.

Alles in allem macht Vitess im Test einen guten Eindruck. Vorbildlich ist anders als bei Citus und ähnlich wie bei YugabyteDB die gute Integration in skalierbare Umgebungen wie Kubernetes. Zwar lässt Vitess sich in Paketform auch als Add-on zu MySQL auspacken. Die fertigen Container-Images, die der Anbieter per Docker Hub zur Verfügung stellt, machen es aber leicht, einen Cluster schnell und ohne großes Vorwissen in Betrieb zu nehmen. Dabei agiert Vitess in Kubernetes wie Yugabyte als First-Class Citizen und lässt sich vorbildlich aus Kubernetes heraus steuern. So mögen es Admins im Zeitalter von Cloud-ready.

31.11. TiDB: fit für OLTP, OLAP, HTAP und ACID

Wer die Dokumentation der TiDB liest, dem fliegen die Abkürzungen nur so um die Ohren. Was vorrangig etwas damit zu tun hat, dass die Entwickler die Werbetrommel für ihr Produkt heftig röhren. So röhmt man sich, dass TiDB – das übrigens für Titanium DB steht – Transaktionen unter Einhaltung der ACID-Regeln beherrscht und sich mithin besonders für Workloads eignet, die OLTP-Bedingungen erfüllen müssen. OLTP steht dabei für Online Transactional Processing und beschreibt Workloads, die Transaktionen durchführen. OLAP wiederum steht für Online Analytical Processing und meint die Analyse von Big-Data-Mengen, für die TiDB sich ebenfalls bestens eignen soll. Dass Hochverfügbarkeit und implizite Redundanz der Daten auf der Featureliste nicht fehlen, versteht sich da fast schon von allein.

Tatsächlich ist TiDB gar nicht so aufregend oder weltbewegend (Abbildung 4), wenn einem die Funktion von Cloud-nativen Datenbanken im Ansatz klar ist. Denn zwischen Vitess, YugabyteDB und TiDB bestehen deutliche Parallelen. So arbeitet auch TiDB mit Sharding, wofür allerdings ein eigener Dienst namens TiKV zum Einsatz kommt, ein Key-Value Store. Wie bei Yugabyte repräsentieren einzelne Querys in TiDB einzelne Nodes im Key-Value Store, und folglich gibt es auch bei TiDB ein Frontend, das die Daten im Hintergrund zusammensucht und sie dann in einem definierten Protokoll ausgibt. Wie die Vitess-Entwickler haben sich die Entwickler von TiDB für

MySQL als Backend entschieden. Die Frontend-Komponente heißt TiDB und tritt stets, wie TiKV auch, geclustert auf und ist implizit redundant.

Der Dienst, der in TiDB die Metadaten verwaltet, heißt Placement Driver Server (PD) und fungiert als eine Art Hirn des gesamten Set-ups. Auch die PDs sind zu einem Cluster zusammengeschaltet und erzwingen ein Quorum, sodass eine Netzwerkpartition keine Probleme verursacht. Weil zu TiDB relativ viele Dienste gehören, ist das Set-up allerdings aufwendiger und im Hinblick auf die Wartung etwas komplexer als bei den anderen Kandidaten. Denn je nach Einsatzzweck gesellt sich an die Seite der TiKV-Server ein weiterer Dienst namens TiFlash: Der nutzt intern eine andere Datenstruktur als TiKV und soll besonders dafür geeignet sein, Analysedaten zu speichern und schnell auszugeben.

Die einzelnen TiDB-Komponenten kommunizieren untereinander über eigene APIs. Die TiDB-Instanzen etwa sprechen über ein eigenes Protokoll namens DistSQL miteinander. Viele Unternehmen migrieren beim Umstieg in die Cloud Daten aus bestehenden lokalen Diensten wie MySQL in eine Cloud-native Datenbank, um die Performance insgesamt zu verbessern. Ein Set-up mit entsprechend vielen Daten stand im Rahmen dieses Tests jedoch nicht zur Verfügung. TiDB veröffentlicht auf der eigenen Website als einziger Anbieter gemessene Zahlen mit nachvollziehbarem Messprotokoll, die einen Eindruck geben von dem, was möglich ist. Angetreten ist dabei eine lokale, einzelne MySQL-Instanz gegen TiDB.

31.12. Ordentlicher Bums in der Datenbank

Die Daten stammen vom asiatischen Logistikkonzern Ninjavan und lesen sich beeindruckend: Bis zu 60-mal schneller wickelt TiDB demnach verschachtelte JOIN-Befehle ab, als es zuvor bei MySQL der Fall war. Im Mittel liegt die Performanceverbesserung „nur“ beim Zehnfachen, aber auch diese Zahl dürfte vielen leidgeplagten Admins lokaler MySQL-Instanzen bereits die Freudentränen in die Augen treiben. TiDB weist hier darauf hin, dass sich eine Migration in die Cloud auch im Hinblick auf die Performance mehr als auszahlen kann.

Darüber hinaus hebt sich TiDB von der Konkurrenz dadurch ab, dass die Firma eigene Werkzeuge zur Verfügung stellt, um bestehende Datenbanken hin zu TiDB zu migrieren. Wie eine solche Migration funktionieren kann, dokumentieren zumindest Vitess und Yugabyte zwar auch, doch greifen beide Firmen dabei auf externe Werkzeuge zurück, mit denen sich möglicherweise nicht jeder Fall abdecken lässt.

Wenn Admins schließlich vor der Wahl stehen, ob sie lieber Vitess oder TiDB nutzen wollen, dürfte ein entscheidender Faktor die Kompatibilität mit MySQL sein. Je exotischer die Kommandos werden, die beim Betrieb der eigenen Datenbank zum Einsatz kommen, desto höher ist die Wahrscheinlichkeit einer Inkompatibilität. Ein Blick in die Kompatibilitätsanmerkungen von Vitess und TiDB offenbart aber schnell, dass bei TiDB die Liste der nicht implementierten Kommandos viel kürzer ist als bei Vitess. Generell hat TiDB also die bessere MySQL-Emulation und verschafft sich gegenüber der Konkurrenz so einen kleinen Vorsprung.

31.13. CockroachDB: komplexer Veteran

Der fünfte und letzte Proband im Test erweist sich als der mit der komplexesten Grundarchitektur: CockroachDB. Die Datenbank schaufelt Daten durch nicht weniger als fünf Layer, um sie letztlich irgendwann auf die Platte zu schreiben. Doch der Funktionsumfang von CockroachDB gleicht dem der anderen Produkte im Test, und von der Komplexität hinter den Kulissen bekommen Administratoren wie Nutzer kaum etwas mit.

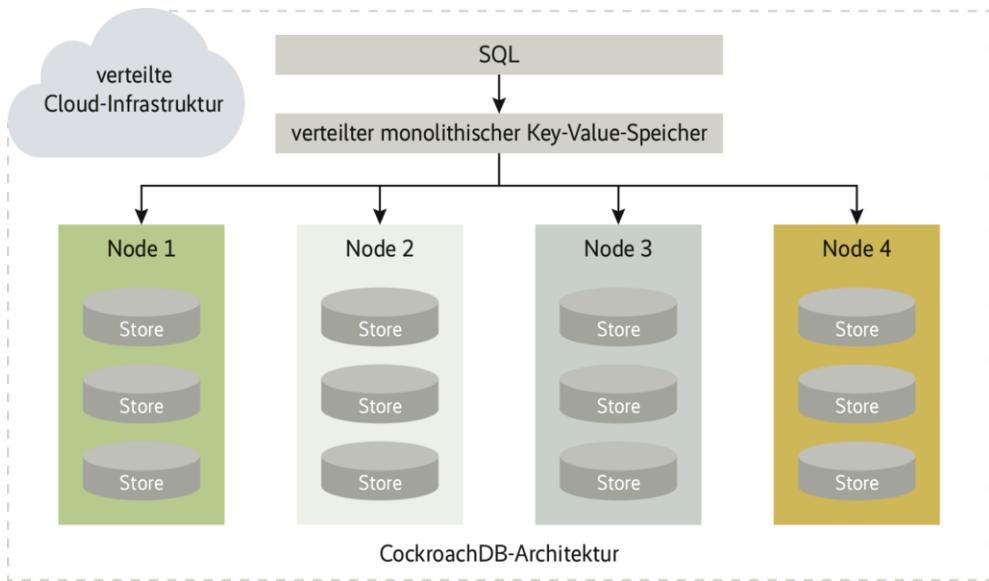


Figure 31.5.: CockroachDB ist so etwas wie der Veteran unter den verteilten Datenbanken mit Cloud-native-Funktionalität, wie Yugabyte und TiDB im Kern aber ebenfalls ein monolithischer Key-Value-Speicher. Quelle: CockroachDB

CockroachDB ist ein Key-Value-Speicher, nutzt also keine bestehende Datenbank. Damit ähnelt es hinsichtlich seiner Struktur YugabyteDB und TiDB. Seine Entwickler haben dafür ein SQL-Frontend entworfen, das das PostgreSQL-Protokoll implementiert – wie üblich mit manchen Einschränkungen, sodass keine volle Kompatibilität zu aktuellen PostgreSQL-Datenbanken besteht. Wie bei den anderen Kandidaten gilt aber, dass die fehlenden Features nur selten zum Einsatz kommen. Obendrein sind die Inkompatibilitäten gut dokumentiert. Administratoren wissen also, worauf sie sich einlassen.

Das Rad erfinden die CockroachDB-Entwickler aber auch nicht neu, was das Thema des Speicherns von Daten auf Datenträgern angeht. Hier kommt stattdessen RocksDB zum Einsatz, das sich zu einer Art Standard im DB-Umfeld entwickelt.

Jede Cockroach-Instanz hat wenigstens einen Store-Prozess, der für das lokale persistente Ablegen von Daten verantwortlich ist. Allerdings treffen die Store-Prozesse keine Entscheidung darüber, welche Daten auf welcher der verfügbaren Store-Instanzen landen. Das tut eine eigene Komponente, der monolithische KV-Speicher, der im Cockroach-Universum ein eigener Dienst ist.

Die Entwickler des Produktes nehmen ihren Mund allerdings ziemlich voll, was Transaktionsicherheit und Konsistenz angeht: CockroachDB soll verteilte Transaktionen ebenso problemlos beherrschen wie die Probleme beseitigen, die der Ansatz der Eventual Consistency mit sich bringt. Dieser führt unter anderem dazu, dass im täglichen Betrieb von Datenbanken Lese- und Schreibblockaden auftreten, wenn auf ein bestimmtes Objekt mehrere Instanzen eines Dienstes gleichzeitig zugreifen wollen. Die CockroachDB-Entwickler begegnen dem Problem mit dem bereits genannten Layer-Kuchen.

31.14. Im Layer-Fahrstuhl nach unten

Den darf man sich wie eine Art Etagenmodell vorstellen, in dem Daten von oben nach unten durchgereicht werden (Abbildung 5). Anfragen und Eingaben machen Clients

per SQL-Protokoll. Cockroach übersetzt den SQL-Befehl dann in eine logische Abfrage für den eigenen Key-Value Store, der die Daten sammelt und eine SQL-konforme Antwort an den Client ausgibt. Dabei beachtet er bereits laufende Querys und sich gerade zutragende Veränderungen.

Viele Begrifflichkeiten kommen einem bekannt vor, hat man sich zuvor mit Vitess beschäftigt: Auch hier ist die Rede von Keyspaces, die in etwa das logische Äquivalent zu einer Datenbank in einer klassischen nicht skalierten SQL-Datenbank sind. Zusätzlich nutzt auch Cockroach Sharding, wickelt dieses aber völlig transparent ab und kümmert sich dabei auch gleich um die Replikation der Daten. Dabei ist es möglich, Failure Domains einzurichten, also etwa verschiedene Racks als Ziele zu definieren. Ähnlich wie bei Yugabyte kommt bei CockroachDB zudem der Raft-Algorithmus anstelle des sonst üblichen Paxos zum Einsatz.

31.15. Reif für den Cloud-Einsatz

Grundsätzlich präsentiert sich CockroachDB als durchaus fit für die Cloud, gerade im Kubernetes-Kontext. Hierfür stellt der Anbieter Nutzern einen eigenen K8s-Operator zur Verfügung, der alle zum Betrieb benötigten Komponenten aufs System holt. Ausführliche Anleitungen zum Deployment von CockroachDB etwa in AWS oder GCE finden sich in der Dokumentation des Anbieters, die gut gelungen ist und zu den besseren im Test gehört.

Cockroach weist darauf hin, dass der Operator aktuell keine CockroachDB-Set-ups erstellen kann, die mehrere geografische Regionen überspannen. Das sei zwar technisch machbar, erfordere in K8s jedoch das händische Ausführen der entsprechenden Schritte. Das schmälert die Freude am Produkt aber nicht. Mit CockroachDB kommt man flott zu einer verteilten Datenbank in der Cloud, die hohen Ansprüchen genügt und gut zu administrieren ist.

31.16. Fazit: heiter bis wolzig

Der Vergleich der fünf Probanden zeigt: Wer skalierbare verteilte Datenbanken mit SQL-Schnittstelle in der Cloud betreiben möchte, hat reichlich Optionen. Drei der vorgestellten Produkte emulieren eine PostgreSQL-Schnittstelle. Yugabyte und CockroachDB nutzen dabei ähnliche Architekturen, Citus fällt etwas aus der Reihe. Zugleich schneidet Citus aus mehreren Gründen im Vergleich am schwächsten ab, was nicht zuletzt auf die fast vollständig fehlende Integration in quasi alle modernen Cloud-Technologien zurückzuführen ist. Ein Produkt, das sich in Kubernetes nicht in wenigen Sekunden in Betrieb nehmen lässt, verdient kaum den Stempel Cloud-ready, auch wenn die Software per se die nötigen Funktionen bietet.

Vitess und TiDB exponieren MySQL und ähneln im Hinblick auf ihre Architektur nicht nur einander, sondern auch YugabyteDB und CockroachDB. Allen vier ist gemein, dass sie für den Einsatz in der Cloud hervorragend vorbereitet sind und Admins mit allen vier Produkten schnell zu brauchbaren Ergebnissen kommen.

Wie üblich beanspruchen die getesteten Produkte ganz unterschiedliche Eigenschaften für sich: TiDB etwa bewirbt das eigene Produkt besonders mit der hohen Performance, die man im realen Set-up und nicht nur in der Teststellung im Labor erreichen soll. Signifikante Unterschiede bei der Performance waren im Test aber nicht zu erkennen – es sei jedoch auf die beschränkte Aussagefähigkeit hingewiesen, die von unserem kleinen Test-Set-up ausgeht. Dass TiDB und Vitess vergleichbare Performancewerte erreichen, ist nachvollziehbar: Letztlich lösen sie die Probleme verteilter Systeme auf vergleichbare Art und Weise – was für die PostgreSQL-basierten Kandidaten genauso gilt.

Table 31.1.: Cloud-native Datenbanken mit SQL-Schnittstelle

	Citus	CockroachDB	TiDB	Vitess	YugabyteDB
Anbieter	Citus Data	Cockroach	PingCAP	CNCF	Yugabyte
URL	www.citusdata.com	www.cockroachlabs.com	github.com/pingcap/tidb	vitess.io	www.yugabyte.com
Lizenz	AGPL 3.0	Business Source License 1.1	Apache 2.0	Apache 2.0	Apache 2.0, POLYFORM
Storage-Engine	lokales PostgreSQL	KV-Store, Pebble	KV-Store, RocksDB (TiKV, TiFlash)	lokales MySQL	KV-Store, DocsDB (RocksDB-Ableger)
Konsensalgorithmus	-	Raft	Raft	Paxos-Variante GCS	Raft
SQL-Dialekt	PostgreSQL	PostgreSQL	MySQL 5.7	MySQL	PostgreSQL
weitere Protokolle	-	-	-	-	Cassandra, Redis
als Managed Service verfügbar	ok	ok	ok	-	ok
Kubernetes-Integration	nur händisch	ok	TiDB Operator	Vitess Operator for Kubernetes, fertige Container	fertige Container

Am Ende dürften also Details wie die implementierten SQL-Befehle das Zünglein an der Waage sein. Wie üblich sollten Admins die für sie infrage kommenden Systeme einem direkten Vergleich unterziehen, um das ideale für den eigenen Use Case zu finden. Zum mindest die Aufteilung in die beiden SQL-Varianten PostgreSQL und MySQL hilft dabei, den Kandidatenkreis etwas kleiner zu halten.

Der Bedarf für Cloud-native Datenbanken ist jedenfalls da: Applikationen müssen heute fast selbstverständlich skalierbar und mithin auch implizit redundant sein, und anders als früher können sie sich dabei kaum noch etwa auf die Replizierfähigkeiten zwischengelagerter Speicherschichten verlassen. Das klassische MySQL, das früher auf einem replizierten Volume lag und von Knoten A zu Knoten B wanderte, ist in Clouds so nicht mehr sinnvoll umzusetzen. Gut zu wissen, dass valide Alternativen existieren.

31.17. Quellen

- Martin G. Loschwitz; Die verteilte Cloud-native-Datenbank YugabyteDB; iX 5/2022, S. 62
- Links zu den Produktseiten der Anbieter s. ix.de/zj1q

31.18. Tasks

- Translation
- Improvements
- Further readings
- Presentation

32. Datenbank: TimescaleDB 2.0 erfasst und analysiert große Zeitreihen

32.1. Einführung

Die frei skalierbare Multi-Node-Datenbank kann Datenmengen in Petabyte-Größenordnung aufnehmen und ist speziell auf das Erfassen von Zeitreihen zugeschnitten.

Von TimescaleDB, einer auf Zeitreihen spezialisierten skalierbaren Datenbank, liegt nach zwei Jahren – mit mehreren Betaversionen und einem ersten Release Candidate – nun der zweite Release Candidate (RC) vor. Die Zeitreihen-Datenbank unterstützt SQL, setzt als Erweiterung auf PostgreSQL auf und ist im Gegensatz zu relationalen Datenbanken frei skalierbar.

Sie bietet eine verteilte Multi-Node-Architektur, die laut Herausgebern Daten von Zeitreihen im Größenbereich von Petabyte speichern und besonders schnell verarbeiten kann. Für selbstverwaltete Installationen der Software steht der RC in Produktionsreife vor, das finale Release und Ausrollen in den von Timescale verwalteten Diensten ist zum Jahresende geplant. Die Datenbank ist als Open Source kostenlos verfügbar.

32.2. Kontinuierliches Aggregieren über aktualisierte APIs

Der Release Candidate stellt laut Ankündigung im Timescale-Blog auch alle Enterprise-Features kostenlos zur Verfügung und gewährt Nutzern mehr Rechte als bislang. Kontinuierliches Aggregieren von Daten soll über aktualisierte APIs möglich sein und Nutzern mehr Kontrolle über den Aggregiervorgang einräumen. Neuerdings lassen sich Aufgaben offenbar benutzerdefiniert anpassen und innerhalb der Datenbank soll es möglich sein, mit einem Zeitplan die einzelnen Aufgaben und Verhaltensweisen bei der Ausführung genauer zu steuern. In puncto Geschwindigkeit der Datenverarbeitung verweisen die Herausgeber auf [Rankings der am Markt verfügbaren relationalen Datenbanken](#), bei denen zurzeit PostgreSQL in den Top 4 zu finden ist und offenbar mit MongoDB gleichauf liegt oder diese leicht überholt.

32.3. Wieso eine eigene Datenbank für Zeitreihen?

Bei der Datenbank geht es nicht um Peanuts, sondern um besonders große Datensätze, die oft verteilt über mehrere Server und Knoten aus der Erhebung telemetrischer Daten laufend eingehen und zum Beispiel im Falle von Finanzdienstleistern oder wissenschaftlichen Projekten über eine Milliarde Datenreihen täglich umfassen können. Produktionslinien in Fabriken, Smart-Home-Geräte, Fahrzeuge, der Aktienmarkt, Software-Stacks, aber auch private Geräte zum Beispiel im Gesundheits-Bereich produzieren über Apps kontinuierlich telemetrische Daten, deren Ordnungskriterium die Zeitreihe ist.

Da das Volumen solcher Datenreihen wächst und relationale Datenbanken bei der Sammlung und Verarbeitung offenbar an ihre Grenzen stoßen, haben die Timescale-

Macher vor dreieinhalb Jahren das Projekt einer spezialisierten Datenbank aus der Taufe gehoben. So unterschiedliche Unternehmen wie Bosch, Siemens, Credit Suisse, IBM, Samsung, Walmart, Uber, Microsoft und Warner Music nutzen und unterstützen die Entwicklung laut Anbieter. Nach Angaben im Timescale-Blog steht neben der PostgreSQL-Community und deren Ökosystem auch eine speziell an Zeitreihen interessierte Entwicklergemeinschaft hinter dem Projekt.

32.4. Weiterführende Hinweise

Ein Rückblick zum ersten Release Candidate zeigt, dass damals eine Reihe von Leistungen noch kostenpflichtig war und dass die Reichweite offenbar gestiegen ist: Die Nutzerzahlen haben sich laut Anbieter von damals (2018) einer Million auf heute zehn Millionen Downloads gesteigert. Weiterführende Informationen zum zweiten Release Candidate lassen sich im Timescale-Blog finden. Der Blog listet eine Reihe von Demo-Videos und bietet mehrere Downloadoptionen an, die Software liegt als Docker-Image sowie in weiteren Varianten vor. Für mit TimescaleDB vertraute Nutzer dürfte das Changelog von Belang sein, für das Aktualisieren hat das Timescale-Team einen Update-Leitfaden zusammengestellt.

33. How to Choose a Forecasting Model – Decision tree to select a time-series methodology

33.1. Key takeaways

- When dealing with time series, classical statistical methods are not directly applicable, so make sure to use a time series-specific methodology
- Understanding the business problem and the data are crucial to select the most suitable model
- Choosing the model is just the start of an exciting journey of model training, evaluation, improvement and operations

33.2. Introduction

Forecasting is a vital function for any business operating in volatile environments. Capturing the variability of the economy and future demand is a crucial skill to react fast to changes and adjust the operations within the supply, procurement, logistics and others. The benefits of forecasting include cutting costs of stock keeping, reducing waste, decreasing out-of-stock and achieving higher service levels.

Time series analysis is a specific methodology that can help businesses improve their forecasting capabilities. It analyzes a sequence of equally spaced data points collected over an interval of time to extract meaningful statistics of the data and predict its future values.

However, with the increasing popularity of data science and the growing complexity of the data, the number of possible time-series models can be overwhelming. Do you need a simple solution to best capture historical sales or is your business strongly dependent on the number of external factors that need to be considered? These are example questions that need to be asked during the data exploration, to select the best-fitting model for the analytical problem. In this article, we will present our approach to perform time-series modeling, including analytical problem framing, data exploration and our novel framework for time-series model selection.

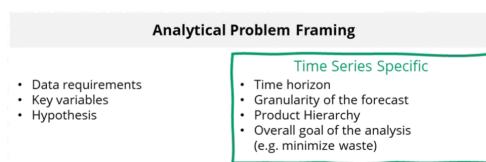


Figure 33.1.: Step 1: Analytical problem framing



Figure 33.2.: Step 2: Data exploration and requirements

33.3. Step 1: Analytical problem framing

In the first stage of our approach, the business problem is translated into an analytical problem and consequently, the scope is defined. This step is crucial to set up the approach of the analysis. In general, analytical problem framing includes defining hypotheses to be tested, variables to be extracted and success criteria for the overall solution. However, time series analysis requires additional steps which extend the standard AI methodology.

In general, the overall goal of time series solution is to minimize the error of the forecast. This can be translated to the business as avoided costs from unnecessary inventory or last-minute freight by increasing the accuracy of the demand forecast. Defining this goal analytically is important and will determine the variables of interest, the time horizon, the forecast granularity and the data hierarchy.

Time horizon is defined as the period of time for which we are forecasting data. For example, a procurement team needs to make an order for a specific number of materials 10 weeks in advance. In that case, the forecast horizon is at least 10 weeks, with 10 weeks probably providing the best accuracy. Based on the business problem, it is important to choose the right time horizon to receive actionable insights from the solution at the right time.

Forecast granularity is defined as the time dimension in which the data is formatted for the forecast, e.g. weekly, monthly or annual. For example, a weekly forecast will require sub-week forecast granularity to include the patterns in the data. The granularity should also be decided to gain the best insights but can also be impacted by data availability.

Product hierarchy is important to understand before starting a forecasting analysis on data. The choice of an optimal hierarchy is determined by the business problem but also by the availability of data. For example, if products are changing regularly but their compositions are highly similar, it can be interesting to group similar products and forecast these together

33.4. Step 2: Data exploration and requirements

Data exploration typically consists of splitting the data into a training and testing set, outlier detection and identifying missing values. These steps are also performed with time series data but additional steps are required in order to start modeling.

When using time series data, it is important to detect any time-related pattern in the data which can help in understanding the problem at hand and choosing the model later. Seasonal patterns are present in time series when seasonal factors affect the data such as days in the week[1]. By plotting the data or by seasonal decomposition in the data exploration stage, it is possible to identify these patterns more clearly.

Feature engineering is a step that can have a large impact on the quality of the model. In time series there are different types of features that can be engineered, such as date time features, lag features and rolling window statistics. See a great article on feature engineering for time-series models[2].

In addition to historical data, there can be other external factors that can help in better predicting the future. We call these variables “external factors”. In this data

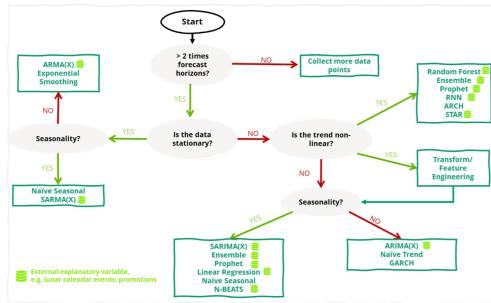
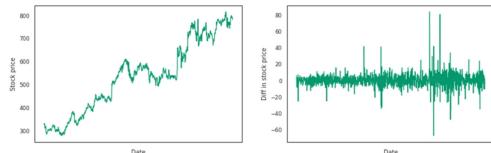


Figure 33.3.: Step 3: Choosing a model

Figure 33.4.: Example of achieving stationarity in stock prices dataset by calculating difference between observations. Figure inspired by: <https://otexts.com/fpp2/stationarity.html>

exploration phase, it is important to identify these variables, understand them and transform them to fit the model. For example, car sales can help predict the demand for car parts. By adding this external factor to the explanatory variables, the performance of the forecasting model can improve.

33.5. Step 3: Choosing a model

Time series data has many characteristics that influence future values, therefore we have created a decision tree to help choose the best fitting model. Here an example on how to use the decision tree:

First, you check how many data points are in your dataset. Our rule of thumb is that the dataset should contain at least 2 forecast periods. If that is not the case, you need to collect more data.

If you have enough data, stationarity is evaluated. Stationary data is defined as having a constant variance and mean over time. You can run an Augmented Dickey-Fuller (ADF) test to check this statistically, but you can also see it visually when plotting the data over time. To achieve stationarity, use the difference between observations at time t and $t-1$ instead of the observation at time t .

Next, you will need to identify the seasonality of the data. By decomposing your time series data, you will identify the trend, seasonal patterns and residuals of the data. The most common method is the classical seasonal decomposition method but there are also others (e.g. X11 decomposition, SEATS decomposition). If your data is not seasonal, you can choose between ARMA(X) and Exponential Smoothing, depending on if you need to add external variables to the model.

If your data is not stationary, checking if the trend is linear is necessary to choose between linear and non-linear models. In non-linear time series, the current value of the series is not a linear function of past observations. BDS test, when applied on residuals from the linear model, can detect a presence of omitted non-linear structure. However, just like with classical regression analysis, non-linearity of the data can be omitted by the transformation of the forecast or predictor variables (e.g. `log()`). Additionally, piece-wise linear analysis can be performed if non-linear data can be

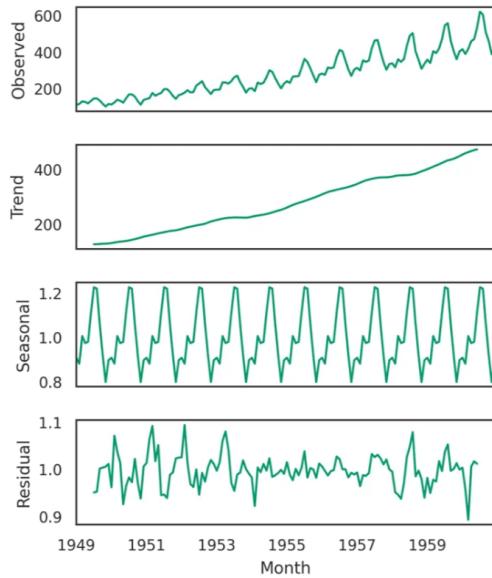


Figure 33.5.: Example of seasonal decomposition of the observed data. Figure inspired by: <https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/>

divided into linear periods. Nevertheless, non-linear models, incl. the ones based on ensemble modeling or neural networks, are known for their high performance and allow for incorporation of external variables in the forecasting, which can help with interpretation of the results.

33.6. Step 4: Model performance and operations

There is a number of great articles on time-series assessment metrics and we recommend reading those [3–5]. Additionally, custom metrics can be designed that work best in a certain organization and can be easily interpreted by business stakeholders. Your forecasting model can also be benchmarked against simpler methods, like naive forecast or moving average.

Forecasting is done using historical data, and as it often happens in machine learning using a model trained once will lead to an increasing error over time. Model lifecycle monitoring is crucial to spot the decrease in performance over time and out-of-sample forecast is a useful methodology to circumvent it, by training the model with a rolling window over time.

Finally, it's important to remember that choosing a model is just the beginning of the process. Thorough experimentation and subsequent model improvements will ensure it is as accurate as possible. Operationalization of the model needs to be considered, including how it will be integrated into existing systems and business processes.

33.7. Conclusion

Time-series modeling is an essential skill for various businesses. By understanding the business problem and the data, selecting the most suitable model, training and evaluating the model, and iteratively improving and operationalizing it, businesses can improve their forecasting capabilities and make better data-driven decisions. We hope that by following these steps and using a systematic approach, you can choose

a forecasting model that meets the needs of the business and helps you make more accurate forecasts.

All images are by the author.

33.8. References

- 1 Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. OTexts.com/fpp3. Accessed 04-03-2022.
- 2 <https://medium.com/data-science-at-microsoft/introduction-to-feature-engineering-for-time-series-forecasting-5a2a2a2a2a2a>
- 3 <https://medium.com/analytics-vidhya/error-metrics-used-in-time-series-forecasting-models-1f1a2f1a2f1a>
- 4 <https://towardsdatascience.com/time-series-forecast-error-metrics-you-should-know-cc2e2e2e2e2e>
- 5 <https://medium.com/@dave.cote.msc/rdr-score-metric-for-evaluating-time-series-forecasting-accuracy-1f1a2f1a2f1a>

34. Image Filters with Python – A concise computer vision project for building image filters using Python

Images exist in different scales, contrasts, bit depths, and qualities. We are filled with a variety of unique and beautiful images that encompass us all over our surroundings and across the internet. Manipulating these images can lead to several intriguing results, which are used for a wide array of fun and helpful applications.

In image processing and computer vision, playing around with images is a critical component to solving different tasks and acquiring desirable results for numerous projects. With the proper handling of imaging tasks, we can recreate a modified version of the image useful for several computer vision and deep learning applications, such as data augmentation.

In this article, we will focus on developing a simple image filter application to primarily modify the brightness and contrast of a particular image. Several other noteworthy modifications, including shader styles, clip art, emojis, and other similar additions, can be implemented and added to your project.

If the readers are not familiar with computer vision and OpenCV, I would suggest checking out one of my previous articles on a complete and extensive beginner guide to get started with OpenCV and computer vision. The link for the same is provided below. I would recommend checking it out first before proceeding with the remaining contents of this article.

34.1. Starter code for Brightness and Contrast modifier with Python:

In this section, we will look at a simple starter code that will help us get started with the basic image filters of modifying the brightness and contrast of the original image with the help of the Open CV computer vision library. For this task, we will make use of a random image for testing the sample code and understanding its fundamental working.

In order to comprehend this test case, I am using the above image as a test sample to analyze and understand the working process of the brightness and contrast modifiers accordingly. To follow along with this project, I highly recommend downloading the above image and storing it in the working directory as [test_image.jpg](#).

Note that you can store the image with any other functional name and use images with any other format as well. The only essential step is to mention the appropriate name and its respective format while reading the images. The first step is to import the Open CV library, as demonstrated in the code block below, and ensure that the library is completely functional.

```
# Importing the computer vision library  
import cv2
```

Once the library is imported and its working is verified, we can proceed to read the original image that we recently saved in our working directory. The cv2.imread function can read the image once the image name is mentioned.



Figure 34.1.: Photo by Jacopo Maia on Unsplash

If the image is not stored in the working directory, ensure that the specific file, along with the image name, is mentioned. Since the original dimension of the above image is 4608 x 3072, it would be optimal to shrink the image by a little bit and reduce the dimensions. I have resized the image to a (512, 512) scale to easily track the progress and perform the desired tasks with a slightly higher speed.

In the final step of the below code sample, we will define alpha and gamma parameters which will act as the contrast and brightness modifiers accordingly. Using these two parameters, we can control the values accordingly. Note that the contrast parameter varies from a range of 0 to 127, while the brightness parameter can vary from 0 to 100.

```
# read the input image
image = cv2.imread('test_image.jpg')
image = cv2.resize(image, (512, 512))

# Define the alpha and gamma parameters for brightness and contrast accordingly
alpha = 1.2
gamma = 0.5
```

Once we have incorporated all the previous steps, we will proceed to use the “add weighted” function in the Open CV library, which helps us to compute the alpha and gamma (the brightness and contrast values) accordingly. This function is primarily useful for blending images by using the alpha, beta, and gamma values. Note that for a single image, we can just assign beta as zero and gain the appropriate results.

Finally, we will display the modified image in the computer vision window, as shown in the sample code block below. Once the image is displayed, we can proceed to close the window when the user clicks the close button. For further understanding of some of the primary functions of Open CV, I highly recommend checking out my previously mentioned article to gain additional information.

```
# Used the added weighting function in OpenCV to compute the assigned values together
```

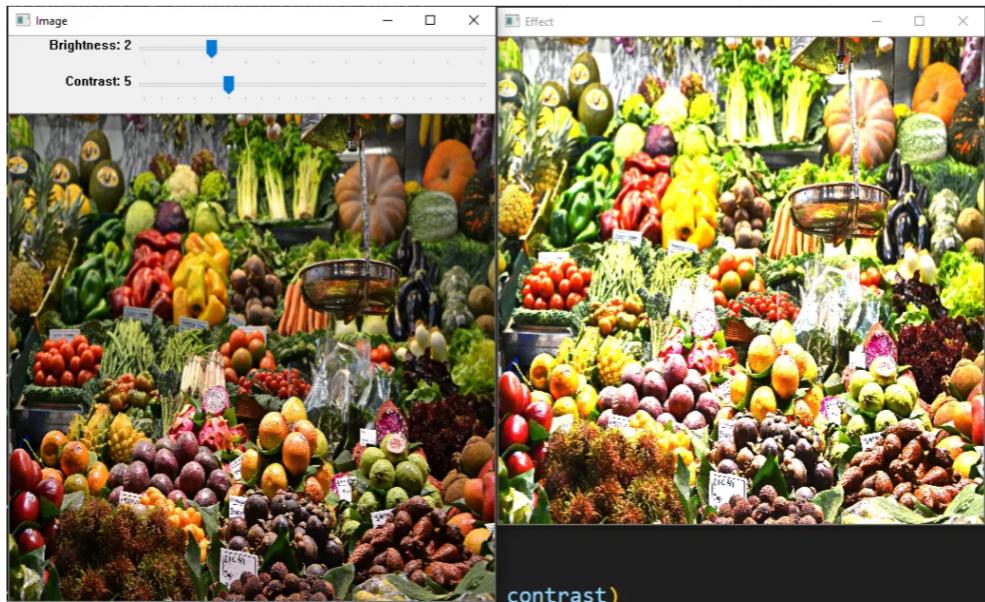


Figure 34.2.: Screenshot of modified image by Author

```
final_image = cv2.addWeighted(image, alpha, image, 0, gamma)

# Display the final image
cv2.imshow('Modified Image', final_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The parameters alpha acts as the contrast, and gamma acts as the brightness modifier in the above starter code sample. However, modifying these parameters as provided in this starter code sample for many different variations might be slightly unrealistic. Hence, the best idea would be to create a GUI interface and find the best values for each parameter of your required image filters. This topic will be covered further in the upcoming section.

34.2. Further development of the project with controllers:

We can create a custom GUI with Open CV using controllers for optimizing the brightness and contrast parameters and adjusting our results accordingly. We can have a slide bar for each of these attributes, which, when moved along its respective variables, can create a different effect on the image by applying a unique filter with each variation of brightness and contrast.

The first few steps are similar to the previous section, where we will import the Open CV library, read the respective image and resize it accordingly. The readers can choose their own appropriate sizes. The code snippet for the following is provided below.

```
# Import the Open CV library
import cv2

# read the input image
image = cv2.imread('test_image.jpg')
image = cv2.resize(image, (512, 512))
```

In the next step, we will define the brightness and contrast function through which we can define the get trackbar position function in the Open CV library to obtain the current position of both the brightness and contrast elements. The weighted function will be utilized to compute both these parameters to calculate a combined output of both these combinations together, as shown in the code block below.

```
# Creating the control function for the brightness and contrast
def BrightnessContrast(brightness=0):
    brightness = cv2.getTrackbarPos('Brightness',
        'Image')

    contrast = cv2.getTrackbarPos('Contrast',
        'Image')

    effect = cv2.addWeighted(image, brightness, image, 0, contrast)

    cv2.imshow('Effect', effect)
```

Once the trackbar function is defined, we will create a named window and the trackbar for which we are obtaining the required parameters. We will create two separate trackbars for both the brightness and contrast features, as shown in the below code block. The wait key function will activate once both the original image window and the trackbar window with the image are closed to help terminate the program.

```
# Defining the parameters for the Open CV window
cv2.namedWindow('Image')

cv2.imshow('Image', image)

cv2.createTrackbar('Brightness',
    'Image', 0, 10,
    BrightnessContrast)

cv2.createTrackbar('Contrast', 'Image',
    0, 20,
    BrightnessContrast)

BrightnessContrast(0)
cv2.waitKey(0)
```

We can adjust the track bar positions for both the brightness and contrast slide bars to change the numbers from 0 to 10 and 0 to 20, respectively. The variations can be observed accordingly by adjusting the respective positions. However, this project still has much room for improvement and a higher degree of scale for varying the parameters. We can scale brightness values on a 255 scale and contrast on a 127 scale to obtain finer and more detailed images.

To fix some of these issues and enhance this project further, I suggest the Geeks for Geeks website, which I considered for a portion of the code used in this section. I highly recommend the readers check the following [link](#) for further reading and understanding of this project.

34.3. Conclusion:

Images play a critical role in computer vision and image processing. By modifying images to create a similar or highly filtered variation of the image, we can solve a wide

array of projects by accumulating more data, as in the case of data augmentation or other similar tasks.

We can also achieve a more desirable and enhanced version of particular images through which several other useful deep learning tasks can be performed. In this article, we explored the use of adding features such as brightness and contrast to modify the original image. In the first section, we learned how to utilize the alpha and gamma parameters to act as brightness and contrast parameters.

In the following sections, we developed a GUI interface through which the original image can be manipulated to obtain copies of its modified version. All the tasks were performed with some basic computer vision and image processing knowledge and libraries.

35. Reinforcement Learning mit Python: Wie eine KI lernt, ein Spiel zu gewinnen

Mit Reinforcement Learning reagiert eine KI auf Belohnungen. Wir zeigen, wie Sie das mit Q-Learning in einem kleinen Praxisbeispiel für Python umsetzen können.

Kleinkinder, die gerade laufen lernen und künstliche Intelligenzen wie AlphaGo haben mehr gemeinsam, als man denkt. Beide lernen durch Versuch und Irrtum: Das Kleinkind lernt, wie es seine Beine nutzen kann, um aufrecht zu stehen und sich fortzubewegen. Während AlphaGo lernt, wie es beim Go-Spiel gewinnen kann. Und genau darum geht es auch beim Reinforcement Learning: Ein Agent lernt, ein Problem ohne Vorkenntnisse zu lösen, indem er verschiedene Aktionen ausprobiert und dafür Feedback in Form von Belohnungen und Bestrafungen erhält.

Zwar können solche Systeme sehr komplex werden, kleinere Probleme lassen sich allerdings bereits mithilfe von simplen Reinforcement-Learning-Agenten lösen. In diesem Artikel zeigen wir, wie Sie einen einfachen Reinforcement-Learning-Agenten mit Python implementieren können. Zudem erklären wir Ihnen die benötigten theoretischen Kernkonzepte und verdeutlichen sie praktisch anhand des Programmierbeispiels.

Das Problem, das der Agent zu lösen hat, wird durch die Umgebung (Englisch Environment) bestimmt. In dieser interaktiven Umgebung ist vorgegeben, welche Aktionen der Agent verwenden kann, um sein Ziel zu erreichen und wann der Agent eine Belohnung erhält. Eine Python-Library, die viele Reinforcement Learning Umgebungen bereitstellt, ist OpenAI Gym. Im Folgenden beschäftigen wir uns mit der Umgebung "Gefrorener See" aus der Library, um die benötigten Konzepte zu verdeutlichen.

35.1. Gefrorener See

Die Umgebung besteht aus 16 Feldern, einem Startfeld links oben, einem Zielfeld rechts unten, vier Löchern und zehn normalen, sicheren Feldern. Ziel der Aufgabe ist, dass der Agent vom Startfeld zum Zielfeld navigiert, ohne in ein Loch zu fallen. Die Löcher befinden sich dabei immer an denselben Stellen. Bei jedem Schritt kann der Agent wählen, in welche der vier Richtungen er sich bewegt: nach oben, unten, links oder rechts.

Schafft der Agent es, auf das Zielfeld zu gelangen, erhält er eine Belohnung und dieser Durchlauf ist beendet – ein Durchlauf bezeichnet auch eine Episode. Bewegt sich der Agent allerdings vorher in ein Loch, endet die Episode, ohne dass der Agent eine Belohnung erhält. In diesem Kontext sind Belohnungen natürlich keine konkreten Gegenstände wie Süßigkeiten, sie werden stattdessen durch Zahlenwerte repräsentiert – hier unterscheiden sich Agent und Kleinkind dann wieder.

35.2. Librarys installieren

Die Umgebung gefrorener See (Frozen Lake) stammt von OpenAI Gym, entsprechend müssen wir die Library gym installieren. Zudem verwenden wir noch numpy, eine Library für mathematische Operationen, und random, eine Library für Zufallszahlen.



Figure 35.1.: In der Umgebung von “Gefrorener See” muss der Agent den See überqueren, ohne in ein Loch zu fallen.

Diese Librarys lassen sich mithilfe von Pip installieren – etwa mit dem Befehl `pip install gym`. Anschließend lassen sie sich in Python wie folgt importieren:

```
import gym
import numpy as np
import random
```

Nun lässt sich die Umgebung mithilfe von `gym.make()` erstellen:

```
env = gym.make("FrozenLake-v1", is_slippery=False)
```

Damit steht die Umgebung bereits. Nun ist es an der Zeit, sich konkretere Gedanken darüber zu machen, wie der Agent lernen kann, das gegebene Problem zu lösen.

35.3. Policy

Anfangs ist unser Agent vollkommen ahnungslos; er weiß weder, wo das Ziel ist, das er erreichen soll, noch, dass es Löcher gibt, in die er fallen kann. Entsprechend ist der Agent gezwungen, diese Dynamiken der Umgebung und ihre möglichen Belohnungen durch das Ausprobieren von verschiedenen Aktionen zu lernen.

Das Ziel des Agenten ist es, eine Policy (Handlungsstrategie) zu finden, die zu einer maximalen Belohnung führt. Ein Beispiel für eine einfache Policy wäre: Immer nach rechts gehen. Beim gefrorenen See würde das allerdings dazu führen, dass der Agent am rechten Rand festhängt und entsprechend keine Belohnung erhält. Der Agent muss also eine komplexere Policy lernen, um das Problem zu lösen.

Um eine optimale Policy zu finden, lassen sich den Zuständen und Aktionen Qualitätswerte zuweisen – kurz Q-Werte. Diese Qualitätswerte sagen aus, wie vielversprechend die Situation von einem bestimmten Zustand aus ist, wenn man eine bestimmte Aktion ausführt. Beim gefrorenen See sind die 16 Zustände die Felder, auf denen sich der Agent befinden kann. Die vier Aktionen sind links, rechts, oben und unten. Die Kombination aus Zustand und Aktion nennt man Zustands-Aktions-Paar. Ein Beispiel für ein Zustands-Aktions-Paar beim gefrorenen See ist: (**Startzustand, Unten**), wobei sowohl Zustände als auch Aktionen meist durch Zahlen repräsentiert sind. Diese Q-Werte lassen sich sehr einfach in einer Tabelle speichern. Eine solche Tabelle nennt sich Q-Tabelle, wobei jede Zeile einen Zustand und jede Spalte eine Aktion repräsentiert. Der Q-Wert für ein gegebenes Zustand-Aktions-Paar lässt sich in der entsprechenden Zelle der Tabelle speichern.

Mit Python lässt sich die Q-Tabelle wie folgt erstellen und mit Nullen initialisieren:

```
n_states = env.observation_space.n # Hier 16, da es 16 Felder gibt
n_actions = env.action_space.n # Hier 4, da es 4 mögliche Aktionen gibt

# Initialize die Q-Tabelle mit Nullen
q_table = np.zeros((n_states, n_actions))

env.observation_space.n liefert uns die Anzahl an Zuständen, während env.action_space.n die Anzahl der Aktionen angibt. Diese beiden Werte nutzen wir als Dimensionen für unsere Q-Tabelle. Mithilfe von np.zeros((n_states, n_actions)) erstellen wir ein zweidimensionales Array, das lediglich mit Nullen gefüllt ist und unsere anfangs leere Q-Tabelle repräsentiert. Es sieht so aus:
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
```

	Links	Unten	Rechts	Oben
F1	0.531	0.591	0.591	0.531
F2	0.531	0	0.656	0.591
F3	0.591	0.729	0.591	0.656
F4	0.656	0	0.591	0.591
F5	0.591	0.656	0	0.531
F6	0	0	0	0
F7	0	0.81	0	0.656
F8	0	0	0	0
F9	0.656	0	0.729	0.591
F10	0.656	0.81	0.81	0
F11	0.729	0.9	0	0.729
F12	0	0	0	0
F13	0	0	0	0
F14	0	0.81	0.9	0.729
F15	0.81	0.9	1	0.81
F16	0	0	0	0

Figure 35.2.: Eine grafische Repräsentation der Q-Tabelle beim gefrorenen See. F1-F16 stehen für die 16 Felder, auf denen sich der Agent befinden kann. Laut dieser Q-Tabelle ist es im Zustand F1 vielversprechender nach unten zu gehen, als nach links.

```
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]
```

Nachdem wir die Q-Tabelle initialisiert haben, benötigen wir sinnvolle Q-Werte, um die Q-Tabelle zu befüllen. Intuitiv lässt sich festhalten, dass der Q-Wert für ein Zustands-Aktions-Paar, das in ein Loch führt, gering sein sollte; schließlich ist es unmöglich von dort aus eine Belohnung zu erreichen, da die Episode sofort endet. Die Q-Werte der Zustands-Aktions-Paare, die näher ans Ziel führen, sollten allerdings hoch sein, da diese Situationen sehr vielversprechend sind.

Haben wir die Q-Tabelle sinnvoll befüllt, kann der Agent von jedem Zustand aus die laut Q-Tabelle vielversprechendste Aktion auswählen und damit einen optimalen Weg zum Ziel finden. Natürlich ergibt es wenig Sinn, dass wir dem Agenten die Werte vorgeben, schließlich soll er sie selbst lernen. Hier kommt das Q-Learning ins Spiel.

35.4. Q-Learning

Beim tabellarischen Q-Learning initialisiert der Entwickler die Werte der Q-Tabelle zunächst zufällig oder, wie bei uns, mit Nullen. Diese Werte lassen sich im Laufe der Zeit aktualisieren, während der Agent mit der Umgebung interagiert und Feedback in Form von Belohnungen erhält. Um dieses Feedback von der Umgebung zu bekommen, muss der Agent im Laufe des Trainings verschiedene Aktionen ausprobieren.

35.5. Exploration vs. Exploitation

Dabei stoßen wir auf das Exploration-vs.-Exploitation-Problem. Es ist ein wichtiges Konzept beim Q-Learning und anderen Verfahren des maschinellen Lernens, das sich mit der Frage auseinandersetzt, wie der Agent in einer unbekannten Umgebung handeln soll. Bei der Exploration versucht der Agent, neue Informationen über die Umgebung zu sammeln, indem er Handlungen ausführt, die er bisher noch nicht ausprobiert hat. Auf diese Weise kann der Agent lernen, welche Handlungen in welchen Zuständen belohnt werden und wie sich die Umgebung verhält. Bei der Exploitation hingegen versucht der Agent, den maximalen Nutzen aus seinen bisher gesammelten Informationen zu ziehen, indem er immer die Handlung wählt, die den höchsten erwarteten Nutzen liefert – hier den bisher höchsten Q-Wert. Auf diese Weise kann der Agent sich schneller an die Umgebung anpassen und möglicherweise schneller belohnt werden.

Das Exploration-vs.-Exploitation-Problem entsteht, weil der Agent in der Regel nicht sicher weiß, welche Handlung in einem gegebenen Zustand die beste ist, solange er nicht genügend Informationen über die Umgebung gesammelt hat. Das bedeutet, dass der Agent immer ein gewisses Maß an Exploration und Exploitation ausführen muss, um eine sinnvolle Policy zu lernen.

Eine Möglichkeit, das Problem in Q-Learning zu lösen, ist die Verwendung von einer Epsilon-Greedy-Policy. Bei Epsilon-Greedy wählt der Agent mit einer Wahrscheinlichkeit von Epsilon eine zufällige Handlung (Exploration) und mit einer Wahrscheinlichkeit von $1 - \epsilon$ die Handlung mit dem höchsten Q-Wert (Exploitation). Die Wahrscheinlichkeit Epsilon kann je nach Problem beliebig gewählt werden. Eine simple Epsilon-Greedy Policy für Q-Learning lässt sich etwa so implementieren:

```

def eps_greedy(state, epsilon):
    r = random.uniform(0, 1)

    if r > epsilon:
        # Wähle die laut Q-Tabelle beste Aktion
        action = np.argmax(q_table[state])

    else:
        # Wähle eine zufällige Aktion
        action = env.action_space.sample()
    return action

```

Die Funktion nimmt den Parameter `state`, den aktuellen Zustand, und unser gewähltes `epsilon` entgegen. Zuerst generiert das Programm mit `random.uniform(0,1)` eine Zufallszahl `r` zwischen null und eins. Ist `r` größer als Epsilon, wird mit `q_table[state]` auf die zum Zustand gehörenden Q-Werte zugegriffen. Beim gefrorenen See sind das vier Q-Werte für die vier möglichen Aktionen. `np.argmax(q_table[state])` gibt uns dann die laut Q-Tabelle beste Aktion für den aktuellen Zustand zurück. Ist `r` kleiner oder gleich Epsilon, wählt das Programm eine zufällige Aktion mit `env.action_space.sample()`.

In komplexeren Szenarien wählt der Entwickler zu Beginn des Trainings meist ein hohes Epsilon, um viel Exploration zu ermöglichen, wenn die Umgebung noch unbekannt ist. Dieses Epsilon lässt sich dann im Verlauf des Trainings reduzieren, sodass mehr Exploitation genutzt wird, wenn die Umgebung hinreichend bekannt ist. Dementsprechend nimmt die Exploration im Laufe der Zeit ab.

35.6. Berechnung der Q-Werte

Nachdem wir nun wissen, wie wir eine Q-Tabelle und eine Epsilon-Greedy-Policy erstellen, geht es nun darum, wie wir die korrekten Werte für die Q-Tabelle berechnen können.

Das Schema funktioniert dabei wie folgt: Der Agent führt von einem bestimmten Zustand `s` eine Aktion `a` aus und erhält dafür eine Belohnung (`reward`). Basierend auf dieser Erfahrung aktualisiert der Agent den Q-Wert des Zustands-Aktions-Paares `(s,a)` mithilfe der folgenden Formel:

$$Q(s,a) = Q(s,a) + \alpha * (reward + \gamma * \max(Q(s',a')) - Q(s,a))$$

Q-Werte beschreiben, wie vielversprechend die Situation von einem Zustands-Aktions-Paar aus ist. Um das zu beurteilen, berücksichtigt das Programm bei der Berechnung der Q-Werte die möglichen zukünftigen Belohnungen. Dazu betrachtet der Agent alle möglichen Aktionen `a'`, die er aus dem Zustand `s'`, der auf das aktuelle Zustands-Aktions-Paar `(s,a)` folgt, unternehmen kann. Der Agent wählt dann die Aktion mit dem höchsten Q-Wert aus `max(Q(s',a'))`, da es die vielversprechendste zukünftige Belohnung verspricht.

Gamma, der Discount-Faktor, bestimmt bei der Berechnung der Q-Werte, wie sehr der Agent zukünftige Belohnungen im Vergleich zu unmittelbaren Belohnungen berücksichtigen soll. Ein niedriger Wert von Gamma führt dazu, dass der Agent hauptsächlich auf unmittelbare Belohnungen achtet, während ein hoher Wert dazu führt, dass langfristige Belohnungen stärker in die Entscheidungen des Agenten einbezogen werden. Die Wahl des richtigen Discount-Faktors hängt von den Eigenschaften der Umgebung und den Zielen des Agenten ab und lässt sich experimentell bestimmen.

Zu `gamma * max(Q(s',a'))` addiert man zudem noch die direkte Belohnung der Aktion (`reward`). Diese direkte Belohnung ist natürlich essenziell, da es zum Ziel gehört, die Gesamtbelohnung zu maximieren. Der alte Q-Wert `Q(s,a)` lässt sich

dann subtrahieren, somit gibt uns der hintere Teil der Formel die Veränderung zum bisherigen Q-Wert.

Alpha bestimmt die Lernrate des Agenten und gibt an, wie stark der Q-Wert in jedem Schritt aktualisiert werden soll. Ein höherer Wert von Alpha führt dazu, dass der Agent schneller lernt, während ein niedrigerer Wert dazu führt, dass der Agent langsamer lernt.

35.7. Die Trainingsschleife

Nachdem wir nun alle benötigten Kernkonzepte besprochen haben, können wir die Trainingsfunktion für den Q-Learning-Agenten implementieren:

```
def train(episodes):
    alpha = 0.9
    gamma = 0.9
    epsilon = 0.5

    avg_reward = 0

    # Wir trainieren den Agenten für die übergebene Anzahl an Episoden
    for episode in range(episodes):
        state = env.reset()[0]
        done = False

        # Die innere Schleife wird wiederholt, bis eine Episode endet
        while not done:

            # Wir wählen die Aktion mithilfe von unserer eps_greedy() Funktion
            action = eps_greedy(state, epsilon)

            # Der Agent führt die Aktion aus und erhält den neuen Zustand,
            # die Belohnung und ob das Spiel zu Ende ist
            new_state, reward, done, t, p = env.step(action)

            # Die Q-Tabelle wird nach der gegebenen Formel aktualisiert
            q_table[state, action] = q_table[state, action] + alpha * (reward + gamma * np.max(q_table[new_state]) - q_table[state, action])

            # Der Folgezustand wird zum aktuellen Zustand
            state = new_state
            avg_reward += reward

            # Wenn der Agent das Ziel erreicht oder in ein Loch läuft, endet
            # die Episode und wir brechen die innere Schleife ab
            if done:
                break

            if episode % 1000 == 0 and episode != 0:
                print("Episode:", episode)
                print("Average Reward:", avg_reward / episode)

    avg_reward /= episodes
    print("Average Reward during training:", avg_reward)
```

```
print("Q-Table:", q_table)
```

Die Funktion `train()` nimmt den Parameter `episodes` entgegen, der die Anzahl der Episoden angibt, die der Agent im Umfeld trainieren soll. Zuerst initialisieren wir die besprochenen Variablen: Die Lernrate Alpha, den Discount-Faktor Gamma und Epsilon, also die Wahrscheinlichkeit, dass der Agent eine zufällige Aktion auswählt. All diese Werte liegen zwischen null und eins.

Das eigentliche Training des Agenten findet in der Trainingsschleife statt. Mit `for episode in range(episodes):` iterieren wir über die gegebene Anzahl an Episoden. Zu Beginn jeder Episode lässt sich der Umgebungszustand mit `env.reset()` auf die Startposition setzen. Um diesen Startzustand für die Funktion `eps_greedy()` nutzen zu können, initialisieren wir die Variable `state` mit dem ersten Wert von `env.reset()`, also mit `env.reset()[0]`. Die Variable `done` gibt an, ob die Episode zu Ende ist – da das am Anfang nicht der Fall ist, initialisieren wir sie mit `False`. Wir führen dann eine zweite Schleife durch, bis eine Episode endet. Am Anfang dieser inneren Schleife wählen wir durch den Aufruf unserer zuvor definierten Funktion `eps_greedy()` die nächste Aktion aus. Als Parameter geben wir dabei den aktuellen `state` und unser festgelegtes `epsilon` mit. Der Agent führt mit `env.step(action)` die gewählte Aktion aus und erhält den neuen Zustand, die Belohnung und die `done`-Variable, die angibt, ob das Spiel beendet ist,

Die ebenfalls übergebenen Werte `t` und `p` interessieren uns in dieser Implementierung nicht. Anschließend berechnen wir den neuen Q-Wert mithilfe des neuen Zustandes und der Belohnung nach der oben besprochenen Formel. Schließlich setzt man den aktuellen Zustand auf den neuen Zustand und der gesamte `reward` wird aktualisiert. Anschließend lässt sich die innere Schleife so lange wiederholen, bis der Agent entweder das Ziel erreicht oder in einem Loch feststeckt.

Um den Fortschritt des Trainings beobachten zu können, lassen wir uns alle 1000 Episoden mit `print("Average Reward:", avg_reward / episode)` den bisher durchschnittlichen `reward` ausgeben.

Am Ende des Trainings gibt das Programm ebenfalls den durchschnittlichen `reward` des gesamten Trainings sowie die aktualisierte Q-Tabelle aus.

35.8. Evaluierung

Um den Fortschritt des Trainings zu überprüfen, schreiben wir noch eine Evaluierungsfunktion:

```
def eval():
    state = env.reset()[0]
    done = False

    while not done:
        env.render()
        action = np.argmax(q_table[state])
        state, reward, done, t, p = env.step(action)

        if done:
            env.render()
            print("Reward during the evaluation:", reward)
            env.close()
            break
```

In manchen Aspekten ähnelt diese Funktion der inneren Schleife der Trainingsfunktion, allerdings lassen wir uns hier die Umgebung mit `env.render()` anzeigen, um den Agenten bei seiner Arbeit beobachten zu können. Außerdem wählen wir während



Figure 35.3.: Nach dem Training ist der Agent in der Lage, die Umgebung innerhalb von sechs Zügen zu lösen.

der Evaluierung die nächste Aktion immer mithilfe der Q-Tabelle mit `np.argmax(q_table[state])` statt mit `eps_greedy()`.

Sobald das Ende einer Episode erreicht ist, gibt das Programm die endgültige Belohnung mit `print("Reward during the evaluation:", reward)` aus. So können wir beurteilen, wie der Agent in der Episode abgeschnitten hat.

Jetzt haben wir alle Komponenten zusammen und können einen Agenten mit beliebig vielen Episoden trainieren lassen:

```
if __name__ == "__main__":
    env = gym.make("FrozenLake-v1", is_slippery=False)
    n_states = env.observation_space.n # Hier 16, da es 16 Felder gibt
    n_actions = env.action_space.n # Hier 4, da es 4 mögliche Aktionen gibt
    q_table = np.zeros((n_states, n_actions)) # Initialisierung der Q-Tabelle

    train(10000)

    env = gym.make("FrozenLake-v1", is_slippery=False, render_mode="human")
    eval()
```

Wir initialisieren zuerst die Umgebung und die Q-Tabelle. Anschließend lassen wir den Agenten 10000 Episoden lang trainieren, was in dieser Umgebung absolut ausreicht. Da das Programm in der Funktion `eval()` die Umgebung rendert, spezifizieren wir nach dem Training den `render_mode = "human"`, um ein Bild der Umgebung nach jedem Schritt in einem Pop-up-Fenster zu erhalten. Anschließend führt das Programm die Funktion `eval()` aus. Das Ergebnis des Trainings ist das folgende Verhalten des Agenten:

Der Agent schafft es also in nur sechs Zügen das Ziel zu erreichen und hat somit eine optimale Lösung gefunden. Die Q-Tabelle, die der Agent gelernt hat, enthält die folgenden Werte:

```
[[0.531441 0.59049 0.59049 0.531441]
 [0.531441 0. 0.6561 0.59049]
 [0.59049 0.729 0.59049 0.6561]
 [0.6561 0. 0.59049 0.59049]]
```

```
[0.59049 0.6561 0. 0.531441]  
[0. 0. 0. 0.]  
[0. 0.81 0. 0.6561]  
[0. 0. 0. 0.]  
[0.6561 0. 0.729 0.59049]  
[0.6561 0.81 0.81 0.]  
[0.729 0.9 0. 0.729]  
[0. 0. 0. 0.]  
[0. 0. 0. 0.]  
[0. 0.81 0.9 0.729]  
[0.81 0.9 1. 0.81]  
[0. 0. 0. 0.]]
```

35.9. Ausblick

In diesem Artikel haben wir Ihnen die grundlegenden Konzepte des Reinforcement Learnings vermittelt. Damit können Sie einfache Probleme in diesem Bereich lösen. Sollten Sie sich jedoch mit komplexeren Umgebungen befassen wollen, stellen Sie fest, dass es oft nicht möglich ist, alle Werte in einer Q-Tabelle zu speichern. In diesem Fall müssen die Q-Werte approximiert werden. Das bedeutet, dass man sich ihnen nähern muss – meistens mithilfe neuronaler Netze.

Das Verfahren, bei dem die Q-Learning-Formel mit einem neuronalen Netz approximiert wird, heißt Deep Q-Learning. Es nutzt viele der Konzepte, die wir in diesem Artikel genannt haben. Wenn Sie sich tiefer mit dem Thema Reinforcement Learning auseinandersetzen möchten, empfehlen wir das englischsprachige Standardwerk “Reinforcement Learning: An Introduction” von Richard Sutton und Andrew Barto.

36. Machine learning: Neural networks generate content

Generative Adversarial Networks are neural networks that generate texts, videos or music. They are good for restoring old films, but also for falsifying them.

If you take a quick look at the landscape picture in the first image, nothing in particular strikes you at first. The photo could have been taken anywhere in Germany. A look at the details, such as the tree trunks or the incidence of light, creates the feeling that something is not right here.

The solution: this landscape does not exist. The picture was generated by a programme (GauGAN). This was made possible by an idea Ian Goodfellow had in 2014. He went to a pub with a colleague and both discussed one of the biggest challenges in Deep Learning: the amount of examples a neural network needs to learn. Image recognition requires thousands of images. A human being has to look at these beforehand and describe what can be seen on them (labelling) so that the learning process later knows whether the result of the neural network is correct or incorrect.

Ian Goodfellow's colleague didn't want to "merely" analyse image content, but to generate completely new images. After a long discussion, Ian Goodfellow came up with a question: If it doesn't work with one neural network and a lot of prepared data, why not try it with two neural networks? These then compete against each other like in a game. One network generates the images and the second judges whether they are real or generated. Through the learning process, each of the networks becomes better and better at its respective activity without needing any data with descriptions (labels).

- In Generative Adversarial Networks (GAN), neural network learns from each other as counter-players and improve their skills through constant competition. The goal of their game is a generator network that can artificially create content.
- With the help of GANs, you can do a lot of useful things such as restoring old films, but you can also distort content just as well, keyword deepfake.
- Ian Goodfellow's classic GAN architecture has laid the foundation for a number of further developments, including DCGAN Deep Convolutional GAN, PGGAN Progressive Growing GAN or CGAN Conditional GAN.

Ian Goodfellow tried this approach on a simple example and achieved the hoped-for results. The later publication "[Generative Adversarial Nets](#)" ([PDF](#)) founded a new branch of machine learning. The following two figures demonstrate the possibilities.

"Generative" here describes the possibility of generating new things with these nets. The second term, "adversarial", can best be translated in German as "kontroversial" or "gegnerisch". Two neural networks compete against each other and learn from it - learning through rivalry.

The examples in this article refer to the generation of images, as this is the easiest place to publish them. However, GANs can just as well generate texts, sounds, videos or other digital data. This applies to anything that can be represented in digital data.



Figure 36.1.: Where can this landscape be found? In Germany? Or in France?



Figure 36.2.: NVIDIA's StyleGAN project generate images of non-existent people.

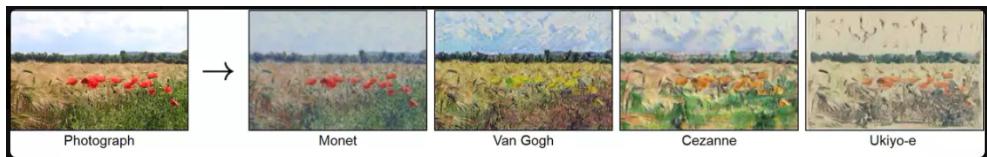


Figure 36.3.: University of Berkeley's CycleGAN transfers image style to new images

36.1. Create something new

When it comes to creating new content, not just anything should be generated, but there is always a specific requirement, such as creating landscape images. For this, the "first ingredient" is as many landscape images as possible. From these, the chosen machine learning method can acquire the pattern that is hidden in the images. This hidden pattern is what makes up this type of image. From a mathematical point of view, this is the common, unknown probability distribution regarding the authenticity of the images. A newly generated image should preferably have this distribution in order to pass again as a landscape image. To ensure that it is not a copy of an already existing image, the "second ingredient" is a set of random numbers as input.

In GANs, there are two neural networks. One is the generator, which generates new images, and the other is the discriminator, which judges how good the new image is. You can compare this to an art forger who tries to paint the best possible pictures in the style of a Van Gogh. The police art expert is his counterpart, who has the task of distinguishing the forged paintings from genuine works that may not yet be known. If the expert recognises a forged painting, it means that the forger was not good enough. The forger still has to learn. If the forger (generator) gets better and better, the expert (discriminator) may no longer recognise a forgery. Then the expert (discriminator) has to learn again.

During the learning process, the two networks continue to grow. They virtually build each other up, just as rivals in sport do, for example. At the end of the learning process, there is a generator network that can be used to create very good new, non-real images.

36.2. Mutual performance monitoring

Like all neural networks that are supposed to get better and better through learning processes, both the generator and the discriminator need a function that tells them how good or bad they currently are. In GANs, there is not only one loss function, but each of the networks has its own. Basically, the learning process goes like this: The generator (G) gets a set of random numbers (vector z) as input and generates new images (x_g) from them. The discriminator (D) in turn receives these (x_g) as input and also real images (x_e). Its task is to recognise the difference.

Distinguishing between two things is a classic task for neural networks. Is the picture a car or a ship? The technical term for this is binary classifier (Binary Classifier).

The result of the discriminator $D(x)$ is 1 if it is a genuine image and 0 if it detects a generated one. A value of 0.7 would express that it is more likely to be a genuine work. The discriminator would be 70 per cent certain. The result of the discriminator $D(x)$ thus corresponds to the assessment of whether it could be a genuine image.

The loss function for the discriminator is composed of the sum $D(x_e)$ for the real images and the sum $(1 - D(x_g))$ for the generated images or the sum $(1 - D(G(x)))$ if x_g is replaced by $D(G(x))$. In somewhat more mathematical terms, this looks like Ian Goodfellow's figure below.

$J(D)$ is the loss function of the discriminator and $J(G)$ with a minus in front of it is that of the generator. This represents the opposite intentions of the generator and discriminator.

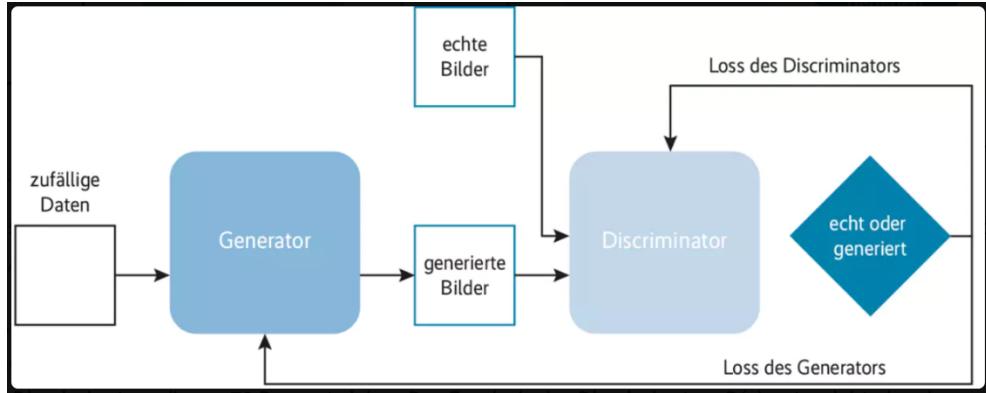


Figure 36.4.: The loss functions control the learning of the neural networks in GANs.

$$J^{(D)} = -\frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2} \mathbb{E}_{\mathbf{z}} \log (1 - D(G(\mathbf{z})))$$

$$J^{(G)} = -J^{(D)}$$

Figure 36.5.: The mathematical representation of a loss function consists of two sums.

A simple neural network corresponds to an optimisation problem in which the learning process changes its parameters until the best values, i.e. the smallest possible value for the loss function, emerge as the result. This is different with GANs.

It is a game between two participants. If one participant gets better scores, the other automatically gets worse. If one player improves by a certain amount, the other gets worse by the same amount. The sum of the values achieved always remains the same. In game theory, this is known as a zero-sum game.

The aim of the learning process is to reach the situation where the images generated by the generator correspond to the internal pattern of the real images. Then the discriminator can no longer distinguish between "generated" and "real", so the result it gives is always 50 per cent ($D(x) = 0.5$).

At the end of the learning process, the parameters of the generator are set so that its loss function $J(G)$ has reached a local minimum, and for the discriminator this is also true for its loss function $J(D)$.

36.3. Training in rotation

The training of a GAN is somewhat different from that of a simple neural network. In a learning loop, the discriminator is trained first and then the generator. In the next run, it starts all over again.

In the same way, the training procedure differs in detail for each neural network.

36.4. Training of the discriminator

- Select random images from the real sample data.
- Give the generator random data from which it generates images.

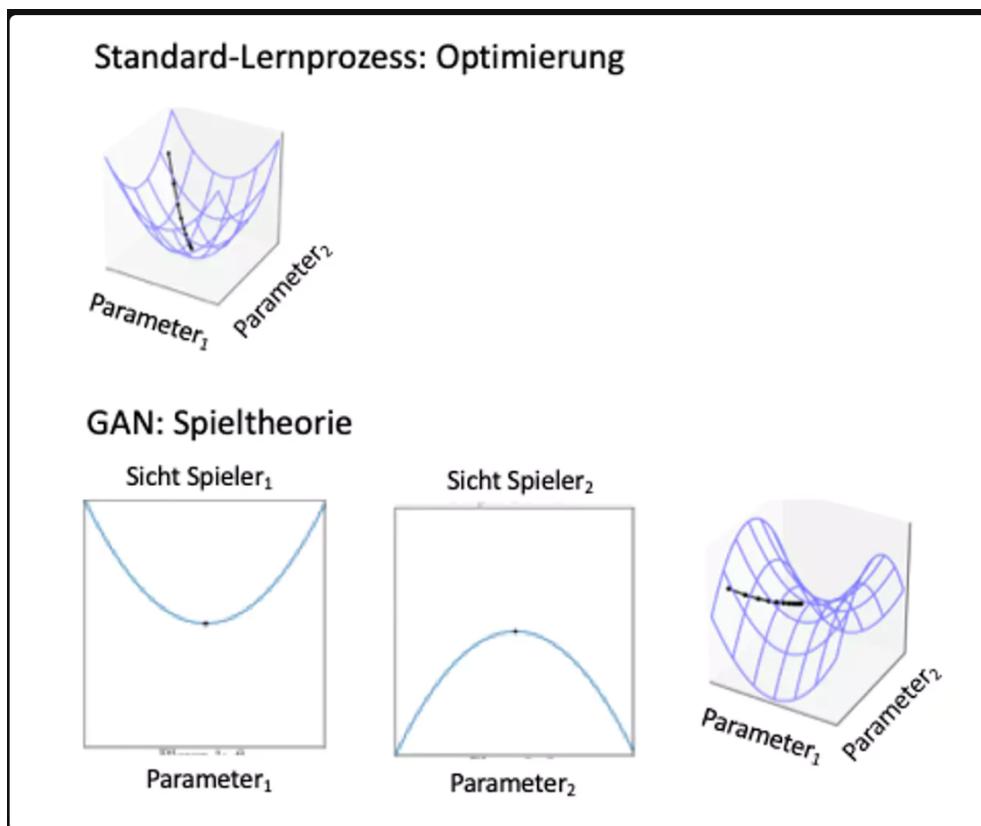


Figure 36.6.: GANs involve two networks competing against each other, unlike simpler neural networks

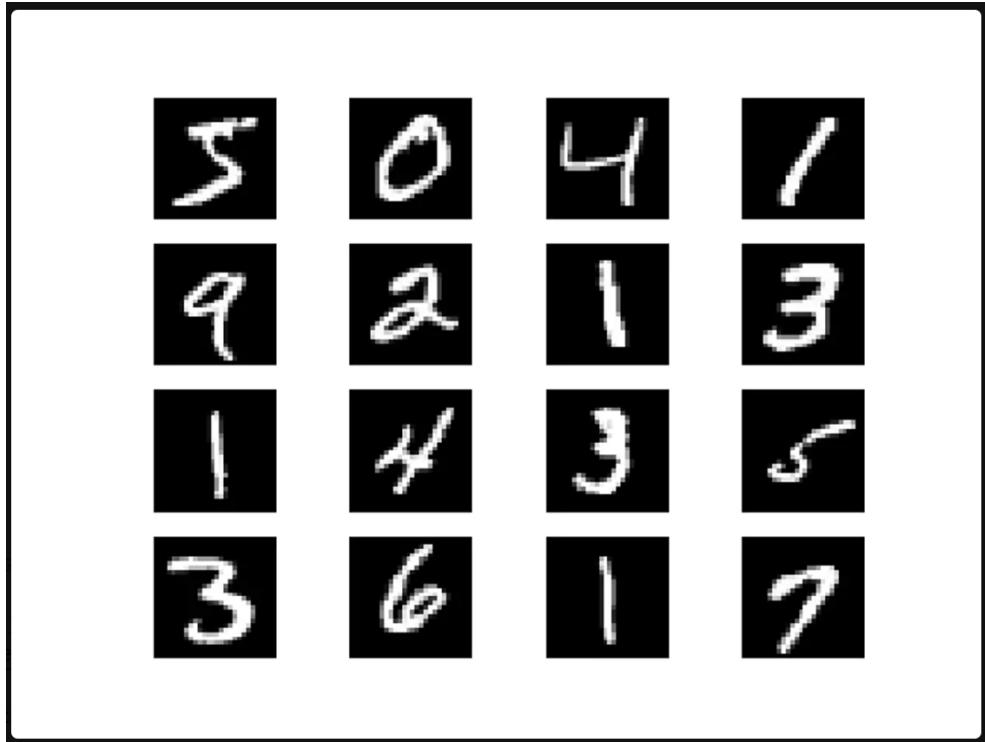


Figure 36.7.: Python GAN learns to generate handwritten digits

- Classify all images whether they are real or generated, regardless of where they come from.
- Adjust your parameters via the loss function using the backpropagation method. The aim is to minimise the error rate when classifying.

36.5. Training the generator

- Take random data and generate new images from it.
- Let the discriminator classify them.
- Adjust your parameters via the loss function using the backpropagation method. The goal is to maximise the error rate when classifying.

The important thing here is that each neural network only adjusts its own parameters during a learning process.

36.6. From theory to practice

The GAN in the example programme `gan.py` from the listing tries to create handwritten digits that look as realistic as possible.

The real images come from [Professor Yann LeCun and his colleagues in the Artificial Intelligence Department \(PDF\)](#) at New York University.

Known as MNIST (Modified National Institute of Standards and Technology) in the field of machine learning, this image database was created from 60000 digits, written by employees of the American Census Bureau and another 10,000 examples of American

```

1000 import numpy as np
1001 from tensorflow.keras.datasets import mnist
1002 from tensorflow.keras.layers import BatchNormalization, LeakyReLU
1003 from tensorflow.keras.layers import Dense, Reshape, Flatten
1004 from tensorflow.keras.models import Sequential
1005 from tensorflow.keras.optimizers import Adam
1006
1007 # Modul mit Statistikfunktionen
1008 from statistics import Statistics
1009
1010 IMAGE_ROWS = 28
1011 IMAGE_COLS = 28
1012 CHANNELS = 1
1013 IMAGE_SHAPE = (IMAGE_ROWS, IMAGE_COLS, CHANNELS)
1014 Z_DIM = 100
1015
1016 SAMPLE_INTERVAL = 10
1017 stat = Statistics(SAMPLE_INTERVAL)
1018
1019
1020 def build_generator():
1021     model = Sequential()
1022
1023     model.add(Dense(256, input_dim=Z_DIM))
1024     model.add(LeakyReLU(alpha=0.2))
1025     model.add(BatchNormalization(momentum=0.8))
1026     model.add(Dense(512))
1027     model.add(LeakyReLU(alpha=0.2))
1028     model.add(BatchNormalization(momentum=0.8))
1029     model.add(Dense(1024))
1030     model.add(LeakyReLU(alpha=0.2))
1031     model.add(BatchNormalization(momentum=0.8))
1032     model.add(Dense(np.prod(IMAGE_SHAPE), activation='tanh'))
1033     model.add(Reshape(IMAGE_SHAPE))
1034
1035     return model
1036
1037
1038 def build_discriminator():
1039     model = Sequential()
1040
1041     model.add(Flatten(input_shape=IMAGE_SHAPE))
1042     model.add(Dense(512))
1043     model.add(LeakyReLU(alpha=0.2))
1044     model.add(Dense(256))
1045     model.add(LeakyReLU(alpha=0.2))
1046     model.add(Dense(1, activation='sigmoid'))
1047
1048     return model
1049
1050
1051 def train(iterations, batch_size=128):
1052     (X_train, _), (_, _) = mnist.load_data()
1053
1054     X_train = X_train / 127.5 - 1. # Werte zwischen -1 und 1
1055     X_train = np.expand_dims(X_train, axis=3)
1056
1057     stat.orig_images(X_train)
1058
1059     valid = np.ones((batch_size, 1)) # labels
1060     gen = np.zeros((batch_size, 1))
1061
1062     for iteration in range(iterations):
1063
1064         # Train Discriminator
1065
1066         idx = np.random.randint(0, X_train.shape[0], batch_size)
1067         imgs = X_train[idx]
1068
1069         noise = np.random.normal(0, 1, (batch_size, Z_DIM))
1070         gen_imgs = generator.predict(noise)
1071
1072         d_loss_real = discriminator.train_on_batch(imgs, valid)
1073         d_loss_gen = discriminator.train_on_batch(gen_imgs, gen)
1074         d_loss = 0.5 * np.add(d_loss_real, d_loss_gen)
1075
1076         # Train Generator
1077
1078         noise = np.random.normal(0, 1, (batch_size, Z_DIM))
1079         g_loss = combined.train_on_batch(noise, valid)
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120
20121
20122
20123
20124
20125
20126
20127
20128
20129
20130
20131
20132
20133
20134
20135
20136
20137
20138
20139
20140
20141
20142
20143
20144
20145
20146
20147
20148
20149
20150
20151
20152
20153
20154
20155
20156
20157
20158
20159
20160
20161
20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173
20174
20175
20176
20177
20178
20179
20180
20181
20182
20183
20184
20185
20186
20187
20188
20189
20190
20191
20192
20193
20194
20195
20196
20197
20198
20199
20200
20201
20202
20203
20204
20205
20206
20207
20208
20209
20210
20211
20212
20213
20214
20215
20216
20217
20218
20219
20220
20221
20222
20223
20224
20225
20226
20227
20228
20229
20230
20231
20232
20233
20234
20235
20236
20237
20238
20239
20240
20241
20242
20243
20244
20245
20246
20247
20248
20249
20250
20251
20252
20253
20254
20255
20256
20257
20258
20259
20260
20261
20262
20263
20264
20265
20266
20267
20268
20269
20270
20271
20272
20273
20274
20275
20276
20277
20278
20279
20280
20281
20282
20283
20284
20285
20286
20287
20288
20289
20290
20291
20292
20293
20294
20295
20296
20297
20298
20299
20300
20301
20302
20303
20304
20305
20306
20307
20308
20309
20310
20311
20312
20313
20314
20315
20316
20317
20318
20319
20320
20321
20322
20323
20324
20325
20326
20327
20328
20329
20330
20331
20332
20333
20334
20335
20336
20337
20338
20339
20340
20341
20342
20343
20344
20345
20346
20347
20348
20349
20350
20351
20352
20353
20354
20355
20356
20357
20358
20359
20360
20361
20362
20363
20364
20365
20366
20367
20368
20369
20370
20371
20372
20373
20374
20375
20376
20377
20378
20379
20380
20381
20382
20383
20384
20385
20386
20387
20388
20389
20390
20391
20392
20393
20394
20395
20396
20397
20398
20399
20400
20401
20402
20403
20404
20405
20406
20407
20408
20409
20410
20411
20412
20413
20414
20415
20416
20417
20418
20419
20420
20421
20422
20423
20424
20425
20426
20427
20428
20429
20430
20431
20432
20433
20434
20435
20436
20437
20438
20439
20440
20441
20442
20443
20444
20445
20446
20447
20448
20449
20450
20451
20452
20453
20454
20455
20456
20457
20458
20459
20460
20461
20462
20463
20464
20465
20466
20467
20468
20469
20470
20471
20472
20473
20474
20475
20476
20477
20478
20479
20480
20481
20482
20483
20484
20485
20486
20487
20488
20489
20490
20491
20492
20493
20494
20495
20496
20497
20498
20499
20500
20501
20502
20503
20504
20505
20506
20507
20508
20509
20510
20511
20512
20513
20514
20515
20516
20517
20518
20519
20520
20521
20522
20523
20524
20525
20526
20527
20528
20529
20530
20531
20532
20533
20534
20535
20536
20537
20538
20539
20540
20541
20542
20543
20544
20545
20546
20547
20548
20549
20550
20551
20552
20553
20554
20555
20556
20557
20558
20559
20560
20561
20562
20563
20564
20565
20566
20567
20568
20569
20570
20571
20572
20573
20574
20575
20576
20577
20578
20579
20580
20581
20582
20583
20584
20585
20586
20587
20588
20589
20590
20591
20592
20593
20594
20595
20596
20597
20598
20599
20600
20601
20602
20603
20604
20605
20606
20607
20608
20609
20610
20611
20612
20613
20614
20615
20616
20617
20618
20619
20620
20621
20622
20623
20624
20625
20626
20627
20628
20629
20630
20631
20632
20633
20634
20635
20636
20637
20638
20639
20640
20641
20642
20643
20644
20645
20646
20647
20648
20649
20650
20651
20652
20653
20654
20655
20656
20657
20658
20659
20660
20661
20662
20663
20664
20665
20666
20667
20668
20669
20670
20671
20672
20673
20674
20675
20676
20677
20678
20679
20680
20681
20682
20683
20684
20685
20686
20687
20688
20689
20690
20691
20692
20693
20694
20695
20696
20697
20698
20699
20700
20701
20702
20703
20704
20705
20706
20707
20708
20709
20710
20711
20712
20713
20714
20715
20716
20717
20718
20719
20720
20721
20722
20723
20724
20725
20726
20727
20728
20729
20730
20731
20732
20733
20734
20735
20736
20737
20738
20739
20740
20741
20742
20743
20744
20745
20746
20747
20748
20749
20750
20751
20752
20753
20754
20755
20756
20757
20758
20759
20760
20761
20762
20763
20764
20765
20766
20767
20768
20769
20770
20771
20772
20773
20774
20775
20776
20777
20778
20779
20780
20781
20782
20783
20784
20785
20786
20787
20788
20789
20790
20791
20792
20793
20794
20795
20796
20797
20798
20799
20800
20801
20802
20803
20804
20805
20806
20807
20808
20809
20810
20811
20812
20813
20814
20815
20816
20817
20818
20819
20820
20821
20822
20823
20824
20825
20826
20827
20828
20829
20830
20831
20832
20833
20834
20835
20836
20837
20838
20839
20840
20841
20842
20843
20844
20845
20846
20847
20848
20849
20850
20851
20852
20853
20854
20855
20856
20857
20858
20859
20860
20861
20862
20863
20864
20865
20866
20867
20868
20869
20870
20871
20872
20873
20874
20875
20876
20877
20878
20879
20880
20881
20882
20883
20884
20885
20886
20887
20888
20889
20890
20891
20892
20893
20894
20895
20896
20897
20898
20899
20900
20901
20902
20903
20904
20905
20906
20907
20908
20909
20910
20911
20912
20913
20914
20915
20916
20917
20918
20919
20920
20921
20922
20923
20924
20925
20926
20927
20928
20929
20930
20931
20932
20933
20934
20935
20936
20937
20938
20939
20
```

high school students. Normally, ML uses these images to generate networks that can distinguish digits in GANs to generate realistic ones.

The Python example programme works with Keras and the TensorFlow-2-API. [The free programme collection by Erik Linder-Norén serves as a model..](#) There you can find the implementations of various further developments of the classic GAN in Python. All Keras modules and objects can be found at tensorflow.keras. The MNIST images needed for learning can also be imported from there. The output of the programme is prepared in the `statistics` module.

Each image in MNIST consists of 28 by 28 pixels and different grey levels. The number of random numbers (vector z) that the generator receives as input is 100 in this example: `Z_DIM = 100`.

36.7. The game structure

The generator (see function `build_generator`) consists of several layers of the type `Dense`. All neurons are connected to each other. `LeakyReLU` is used as the activation function between the layers.

The normal function `ReLU` sets all values that are smaller than 0 to the value 0. The activation function `LeakyReLU`, on the other hand, returns a negative value that is slightly smaller than 0, such as $0.01 * x$, for such values (x). This often allows the optimiser to reach the goal more quickly.

The last layer has `tanh` as an activation function so that the values for the output of the generator are between -1 and 1.

Like the generator, the discriminator (see function `build_discriminator`) consists of layers of the type `Dense` and also works with the activation function `LeakyReLU`. The last layer of the network has `sigmoid` as its activation function so that the output of the discriminator is between 0 and 1. The output in this case corresponds to the probability that the discriminator generates the image for (0) or genuine (1).

The model for training the discriminator is composed of the neural network of the discriminator, an Adam optimiser and the loss function binary cross entropy. This describes the difference between probability distributions, in this case between the value estimated by the discriminator and the actual value (0 = generated, 1 = real). The article "A Gentle Introduction to Generative Adversarial Network Loss Functions" by Jason Brownlee describes in detail why the cross entropy can be used for GANs and ["Classical formula"\(PDF\)](#) by Ian Goodfellow cannot be used directly. For the learning process of the generator, one needs a combination of the neural network of the generator and the discriminator. Since only the generator is supposed to learn in this model, the `trainable` setting of the discriminator is set to `False`.

36.8. Improve step by step

The learning process runs in the `train` function. With each loop pass, the discriminator learns first and then the generator. For the discriminator, the `train` function first selects random images from the real ones. After that, it needs a corresponding number of generated images from the generator.

The Keras function `train_on_batches` gets the images as the first parameter and the corresponding labels as the second. When training with the real images `imgs` the vector `valid` for the labels consists of ones. The one here means "real", the zero "generated". The return value `d_loss_real` is a vector that contains the loss value of the discriminator's estimation for each image.

When training with the generated images `gen_imgs` and the vector `gen` with the labels (contains only 0), the vector `d_loss_fake` provides the loss values for the generated

images. The total loss value can be determined from the values `d_loss_real` and `d_loss_gen` with a weighting of 0.5.

In the learning process of the generator, the model combined, consisting of the networks of the generator and discriminator, is used. In the `train_on_batch` function, the generator first receives the input `noise` (random values) and thus generates an image. This is immediately passed on to the discriminator, which evaluates it. However, this function only trains the generator, as the training for the discriminator is switched off in the model `combined`.

Both neural networks have gone through their learning process and the loop starts again from the beginning. When the entire learning process is complete, the generator takes over creating the new images on its own.

Ian Goodfellow's classic GAN described here triggered an avalanche of further developments.

36.9. GAN-Zoo

The idea of the GAN architecture by Ian Goodfellow in 2014 was the starting signal for many further developments.

- **DCGAN Deep Convolutional GAN:** The DCGAN replaces the classic fully linked layers (dense) with the convolutional layers known from Deep Learning. These are more powerful for image recognition and similar topics. Fewer learning processes are necessary. In addition, data is normalised between the individual layers (batch normalisation), which further improves performance.
- **PGGAN Progressive Growing GAN:** The essential new feature of PGGAN is the learning process. This becomes increasingly difficult the more the neural networks swell. In order to achieve better resolutions for images or other data, however, the networks have to keep growing. The idea behind PGGAN is to start with the smallest possible networks in the generator and discriminator. First you take images with low resolution, together with a few layers, and increase this more and more. This shortens the time for the learning process. This allows PGGANs to achieve higher resolutions in images with the same resources.
- **Cycle GAN:** CycleGAN is about automatically transferring images from one domain, such as summer images, to another domain, winter images. Until now, pairs of images were always necessary for the learning process, a winter image and a corresponding summer image. Of course, it is very time-consuming to first obtain as many of these image pairs as possible. CycleGAN manages this transfer from one image to another without image pairs. This makes it possible, for example, to convert photos into pictures of famous painters. There can be no image pairs for this.
- **CGAN – Conditional GAN:** CGAN do not generate just any images, the user controls how they should look. If there is additional data (features) about the gender, age or appearance of the person depicted, he or she can wish the generated image to show, for example, a red-haired, 20-year-old man.

If the user can determine the appearance of the image via specifications, the perspective for the future is that such GANs could one day be able to replace the classic graphics engines, for example in computer games.

A list of many more GANs can be found on Avinash Hindupur's GAN Zoo page.

36.10. Real or computer generated?

Since 2017, the term deepfake has been doing the rounds. This stands for images or other computer-generated media that look deceptively real.

For example, if a person is to have a particularly positive effect on a social platform, he or she needs as many followers as possible - why not create many fake accounts that follow you? This in turn requires pictures of people. Anyone who has read this article about GANs will know where to get them. According to a report by heise online, researchers have already identified countless fake accounts on Facebook.

Of course, much more is possible with neural networks. For example, there are generated videos in which Barack Obama says what the "scriptwriters" have come up with. In the same way, programmes can imitate a person's biometric data or put faces of celebrities on the bodies of porn stars.

In order to be able to do something about this, it is currently up to the lawyers and the technicians. Legislation must be updated to make such offences punishable. For example, the US House of Representatives is preparing an anti-deepfake bill.

On the technology side, one of the goals is the automatic detection of deepfakes. Facebook has offered rewards totalling 10 million US dollars in its Deepfakes Detection Challenge. As is often the case in the field of IT security, there will probably be a constant neck-and-neck race between the producers of deepfakes and those who expose them.

But GANs and similar technology offer just as many positive applications, from simpler workflows for post-processing images and films to the automatic restoration of photos. Neural network techniques even have the potential to replace existing graphics engines and generate ever better images and sounds in films and computer games.

37. Daten visualisieren mit Grafana

Grafana macht aus gesammelten Messwerten anschauliche Diagramme und verschickt auf Wunsch Alarne. Damit ist es für Admins wie Hausautomatisierer interessant. Wenn man eine Server-Infrastruktur, das vernetzte Zuhause oder eine Industrieanlage überwacht, kommt man nicht umhin, Messwerte einzusammeln und in einer geeigneten Datenbank abzuspeichern. Für diesen Zweck haben Entwickler sogar spezialisierte Datenbanken entwickelt – InfluxDB ist einer der bekanntesten Vertreter. Wie die ersten Schritte mit InfluxDB gelingen, stellen wir im Artikel [InfluxDB: Spezialisierte Datenbank für Messwerte und Logging](#) vor. Mit dem Datensammeln allein ist aber noch nicht viel gewonnen, erst eine grafische Auswertung macht Zahlen für Menschen begreifbar und hilft bei der Fehlersuche. Für diese Aufgabe hat die Webanwendung Grafana weite Verbreitung gefunden. Sie wird von [Grafana Labs](#) als Open-Source-Projekt entwickelt; das Unternehmen verdient mit Support, Beratung und einigen Zusatzfeatures Geld. Der natürliche Weg für Software aus diesem Umfeld ist ein Docker-Container. Ein passendes Docker-Image stellt Grafana Labs bereit. Eine Grafana-Instanz könnte man für Windows, macOS und Linux durchaus als Programm Paket herunterladen und ausführen. Die Docker-Variante macht die Einrichtung aber viel einfacher: Voraussetzung ist ein Computer mit Windows, Linux oder macOS mit installiertem Docker und Docker-Compose. Wie man Docker aufspielt, erklärt [Docker einrichten unter Linux, Windows, macOS](#).

37.1. Container-Beobachter

Zum Üben bringt Grafana einen Demo-Daten-Generator mit. Für ein anschauliches Dashboard mit Diagrammen braucht es aber realistische Daten. Als Beispielprojekt wird in diesem Artikel ein Server überwacht, auf dem Docker läuft. Messwerte wie Prozessorauslastung und Arbeitsspeicher-Füllstand landen dann in einer InfluxDB und Grafana zeigt sie an.

Zum Einsatz kommen [das kleine Werkzeug cAdvisor](#), das Messwerte zusammenträgt, eine InfluxDB-Instanz, die sie speichert, und Grafana für die Visualisierung. Das klingt wesentlich komplizierter, als es ist. Windows-Nutzer müssen jetzt aber zu einer virtuellen Maschine mit Linux oder einem angemieteten Server greifen – unter Windows hat cAdvisor keinen Zugriff auf die Messwerte, weil Systempfade wie `/sys` nicht existieren.

Damit Sie schnell die Grafana-Oberfläche zu sehen bekommen, haben wir für das Zusammenspiel der [Container](#) eine Docker-Compose-Zusammenstellung vorbereitet, die nur noch gestartet werden muss. Auf dem Listing sehen Sie das Docker-Compose-File mit den beteiligten Containern:

cAdvisor bekommt lesenden Zugriff auf `/sys`, `var/lib/docker` und `/var/run` und den Auftrag, die extrahierten Messwerte in die InfluxDB zu schreiben. Grafana bekommt einen Datenordner und soll auf Port 3000 die Weboberfläche anzeigen. Abtippen müssen Sie das Compose-File nicht, es liegt bei GitHub zum Klonen bereit. Laden Sie den Versuchsaufbau über die Kommandozeile herunter:

```
SHELLgit clone https://github.com/jamct/influx-logger
```

Wechseln Sie mit `cd influx-logger` in das geladene Verzeichnis. Mit `docker-compose up -d` fahren die Container hoch und beginnen mit ihrer Arbeit. Unter der Adresse `<IP des Servers>:3000` im Browser antwortet jetzt Grafana.

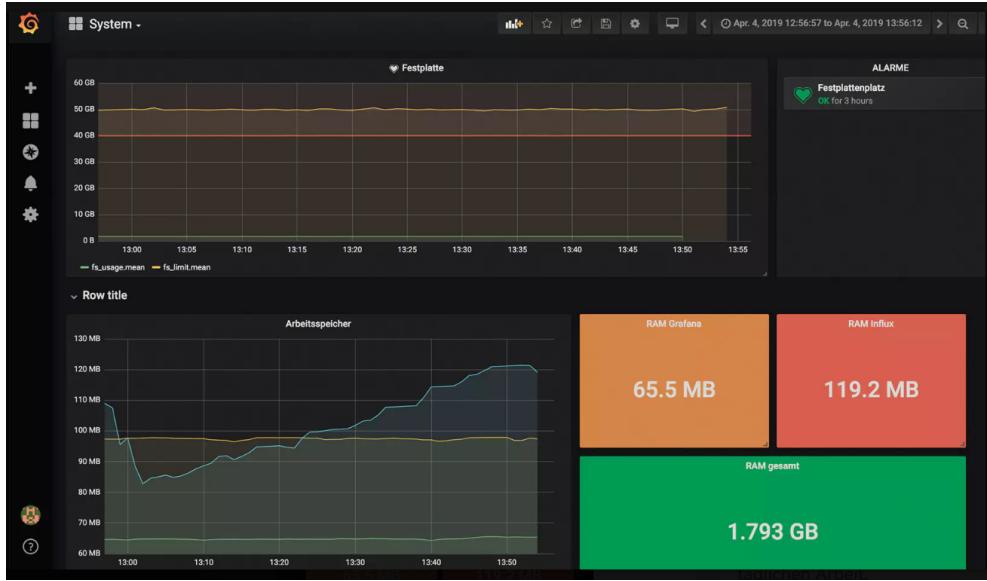


Figure 37.1.: Beispiel einer Grafana-Anwendung

```

1000 version: "3.6"
1001 services:
1002   cadvisor:
1003     image: google/cadvisor:latest
1004     restart: unless-stopped
1005     volumes:
1006       - /sys:/sys:ro
1007       - /var/lib/docker/:/var/lib/docker:ro
1008       - /var/run:/var/run:ro
1009     command: --storage_driver=influxdb \
1010       .--storage_driver_db=cadvisor \
1011       .--storage_driver_host=influx:8086
1012   influx:
1013     image: influxdb
1014     restart: unless-stopped
1015     environment:
1016       - INFLUXDB_DB=cadvisor
1017     volumes:
1018       - ./data/influx:/var/lib/influxdb
1019   grafana:
1020     image: grafana/grafana
1021     restart: unless-stopped
1022     ports:
1023       - 3000:3000
1024     volumes:
1025       - ./data/grafana:/var/lib/grafana
1026

```

Listing 37.1.: Konfiguration von influxdb und Grafana

37.2. Schnellstartanleitung

Beim ersten Start verlangt Grafana Benutzernamen und Kennwort. Mit **admin** dürfen Sie sich anmelden und werden aufgefordert, ein neues Kennwort zu vergeben.

Bevor es ans Einrichten von Diagrammen geht, muss eine Datenquelle eingerichtet sein, zum Beispiel eine InfluxDB-Zeitreihendatenbank. Bevor es losgehen kann, brauchen Sie eine Datenquelle, also einen Adapter für die verwendete Datenbank. Klicken Sie auf **Add data source** und wählen Sie **InfluxDB**. In der ersten Zeile können Sie einen Namen vergeben, zum Beispiel **Docker-Metriken**. Geben Sie dann unter **URL** die Adresse der Datenbank an. Docker sorgt dafür, dass der Grafana-Container den InfluxDB-Container unter dem Hostnamen **influx** findet (dieser Name ist in der Compose-Datei angegeben). Die Adresse lautet also: **http://influx:8086**. Weiter unten im Formular unter **Database** geben Sie **cadvisor** ein. Dieser Name wurde ebenfalls in der Compose-Datei festgelegt. Jetzt ist die Datenquelle bereit für einen Test: Klicken Sie ganz unten auf **Save & Test**, Grafana meldet dann **Data source is working**.

Das erste Dashboard bauen Sie mit einem Klick auf das + im linken Menü. Wählen Sie im Untermenü **Dashboard** und dann **Add Query**. Es öffnet sich ein auf den ersten Blick etwas verwirrender Dialog. Oben erscheint ein leeres Diagramm (Grafana nennt Diagrammkästen **Panel**), darunter hinter dem Begriff **Queries to** ein Dropdown-Menü für die zu verwendende Datenquelle. Wählen Sie hier die gerade angelegte Quelle **Docker-Metriken**. Darunter befindet sich ein Baukasten für die InfluxDB-Abfrage, die der SQL-Syntax stark ähnelt. Grafana zeigt beim Klick auf die einzelnen Bausteine alle Optionen an. Klicken Sie auf **select measurement**, um zu sehen, welche Messwertreihen in der Datenbank angelegt sind. Das erste Diagramm soll die Auslastung des Arbeitsspeichers anzeigen. Wählen Sie **memory_usage** aus. Daneben befindet sich der Block "WHERE", über den man genauer filtern kann – dazu später mehr. Oben sehen Sie jetzt bereits eine Vorschau des Diagramms.

Im Abschnitt **GROUP BY** wird festgelegt, in welchen Zeitabschnitten die Daten zu Mittelwerten zusammengefasst werden sollen. Voreingestellt ist **time(\$_interval)**. Klicken Sie auf den Inhalt der Klammern und probieren Sie die einzelnen Zeitabschnitte durch. Je größer der Wert, desto glatter wird das Diagramm - **1m**, also eine Minute, sollte für einen Überblick über die Auslastung ausreichen.

Die erste Abfrage ist fertig und Sie können sich an die optischen Feinheiten machen. Klicken Sie dazu links auf das zweite runde Icon, das ein Diagramm darstellt. Oben können Sie den Typ der Visualisierung ändern. **Graph** ist voreingestellt und steht für Balken- oder Liniendiagramme. Der nächste Abschnitt ist selbsterklärend und erlaubt die Darstellung von Linien, Balken oder Punkten. Scrollen Sie weiter zum Abschnitt **Axes**. In der ersten Spalte **Left Y** für die Y-Achse sollte die **Unit** umgestellt werden. Der Logging-Container speichert die Werte für den Arbeitsspeicher in Byte. Wählen Sie daher unter **Data (Metric)** die Option **bytes**. Grafana denkt mit und wählt ein zu den Werten passendes Präfix: Die Werte in der Vorschau oben erscheinen in **MB**. Wenn Sie noch weiter scrollen, finden Sie den Abschnitt **Legend**, der festlegt, welche Details als Legende angezeigt werden sollen.

37.3. Diagramm anzeigen und anpassen

Das erste Diagramm ist fertig und sollte gespeichert werden. Ganz oben rechts findet sich das Disketten-Symbol. Grafana fragt beim Speichern nach einer Beschreibung der Änderungen. Das ist vor allem in Mehrbenutzerumgebungen wichtig – Entwickler kennen das Konzept von Versionsverwaltungen wie Git. Mit dem Pfeil oben links gelangen Sie zurück in die Ansicht des neuen Dashboards. Das Diagramm können Sie jetzt per Drag-and-Drop verschieben und über die untere rechte Ecke in der Größe anpassen. Auch solche Änderungen müssen über die Diskette oben rechts gesichert

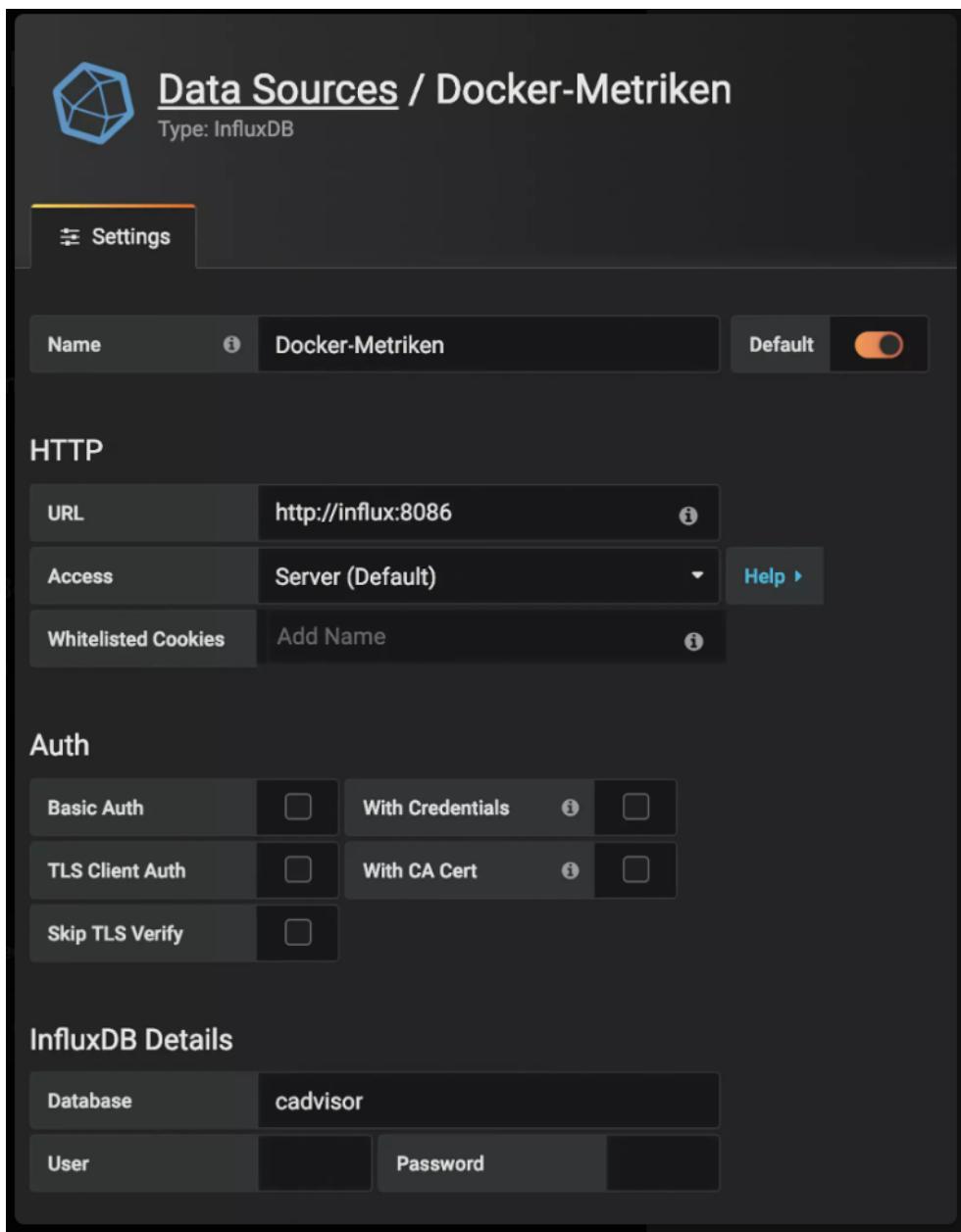


Figure 37.2.: Bevor es ans Einrichten von Diagrammen geht, muss eine Datenquelle eingerichtet sein, zum Beispiel eine InfluxDB-Zeitreihendatenbank.

werden. Daneben finden Sie den Button zum Anlegen weiterer Diagramme. Mit Ihrem ersten Diagramm können Sie aber bereits die Funktionsweise von Grafana kennenlernen. Oben rechts befindet sich der Auswahldialog für den anzuzeigenden Zeitbereich. Klicken Sie darauf und wählen zum Beispiel **Today**. Da Ihr Logging-Versuchsaufbau erst wenige Minuten läuft, ist der Großteil des Diagramms leer. Zum Hereinzoomen klicken Sie einfach im Diagramm auf den gewünschten Anfangszeitpunkt und ziehen Sie mit gedrückter Maustaste bis zum Ende. Grafana ändert den ausgewählten Zeitbereich für das gesamte Dashboard - also für alle anderen Diagramme, die Sie noch ergänzen. Das macht das Erkennen von Zusammenhängen sehr einfach. Sehen Sie in einem Diagramm zum Beispiel einen auffälligen Wert beim Arbeitsspeicher, schneiden Sie den Bereich entsprechend zu, um zu sehen, ob zur gleichen Zeit etwa auch die Netzwerkkarte viele Daten übertragen hat. Mit der Lupe oben rechts zoomen Sie wieder heraus.

Anmerkungen ins Diagramm schreiben

Wenn Sie den Grund für den Fehler gefunden haben, möchten Sie diesen möglicherweise gern im Diagramm notieren, damit Ihr Administratoren-Kollege nicht am nächsten Tag verwundert auf den gleichen Datenpunkt schaut und erneut mit der Fehlersuche beginnt. Grafana beherrscht dafür **Annotations**. Ziehen Sie mit gedrückter Strg-Taste (Mac-Benutzer nutzen die Cmd-Taste) vom Beginn zum Ende des Ereignisses, das Sie kommentieren möchten. Hier können Sie einen Text eingeben und Tags hinzufügen (für diese Tags gibt es Möglichkeiten zur Integration in Bugtracking-Software, die den Rahmen dieses Artikels sprengen würden). Der markierte Bereich wird mit blau gestrichelten Linien gekennzeichnet. Fährt man mit der Maus über den blauen Balken an der x-Achse, kann man den hinterlegten Text auslesen.

Bisher zeigt das Diagramm nur eine Messwertreihe für die gesamte Speicherauslastung an. Spannend wäre es aber, das nach Containern zu filtern. Klicken Sie zum Bearbeiten eines Diagramms auf das kleine Dreieck neben dem Titel, der noch **Panel Title** lautet. Mit **Edit** kommen Sie wieder in den bekannten Bearbeiten-Dialog. Bei der Gelegenheit können Sie dem Diagramm einen Namen geben: Klicken Sie auf das Zahnrad-Icon links und vergeben Sie einen Namen wie **Arbeitsspeicher**.

Um eine weitere Datenreihe einzufügen, klicken Sie links auf das Datenbank-Icon. Hier sehen Sie, dass die erste Datenreihe den Namen **A** bekommen hat. Rechts könnten Sie mit **Add Query** eine weitere einfügen. Um die nächste Abfrage nicht ganz von vorne einrichten zu müssen, ist es sinnvoll, Reihe A zu duplizieren. Der entsprechende Button, der zwei Dokumente darstellt, befindet sich rechts.

In der neuen Abfrage B klicken Sie neben **WHERE** auf das Plus und wählen Sie **container_name**. Das ist ein Tag, das der Logging-Container an seine Messwerte heftet. Grafana fügt ein Gleichheitszeichen ein und daneben den Block **select tag value**. Wählen Sie aus dem Menü einen Container, zum Beispiel **influx-logger_grafana_1**. Damit erfahren Sie, wie viel Arbeitsspeicher Ihre Grafana-Instanz selbst beansprucht. Im letzten Feld **ALIAS BY** können Sie einen sprechenden Namen für den Messwert vergeben, der dann in der Legende angezeigt wird. Speichern nicht vergessen, bevor Sie zur Ansicht des Dashboards zurückkehren.

37.4. Schwellwerte definieren

Neben den Balken- oder Liniendiagrammen für Zeitreihen bringt Grafana auch Elemente für die Darstellung eines aktuellen Messwerts. Praktisch wäre es zum Beispiel, den aktuellen Arbeitsspeicherfüllstand groß anzuzeigen - mit einem leuchtend roten Hintergrund, wenn der Wert über einen Schwellwert steigt. Legen Sie ein neues Panel an (Button in der oberen Leiste). Klicken Sie auf **Choose Visualization**. Eingefügt werden soll ein **Singlestat** für den Arbeitsspeicher, darunter wählen Sie aus, welcher Wert angezeigt wird. Wechseln Sie von **Average** auf **Current** für den aktuellen Messwert statt eines Durchschnittswertes. Unter **Unit** ist es wieder sinnvoll, auf **bytes**

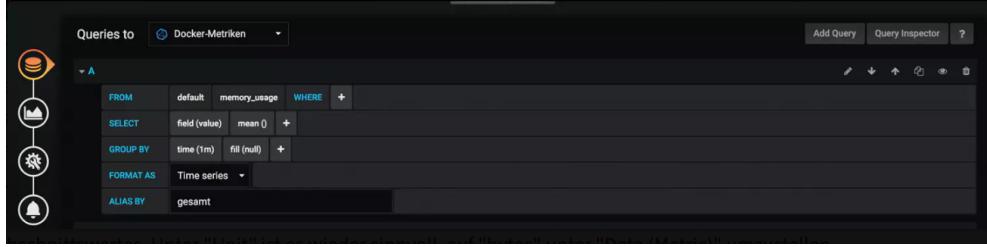


Figure 37.3.: Beim Zusammenbau der Datenabfrage unterstützt Grafana mit einem Baukasten.

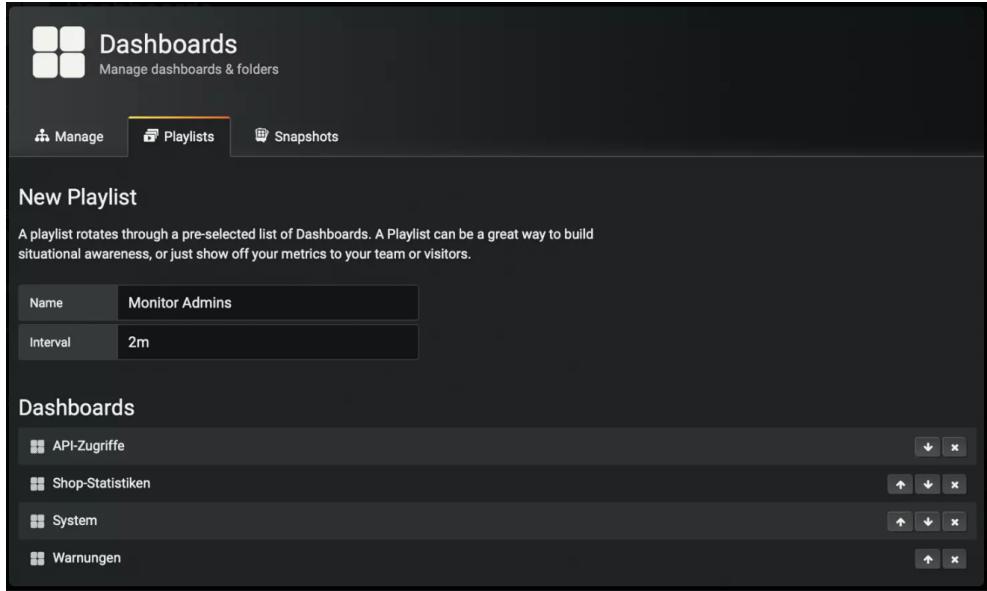


Figure 37.4.: Mehrere Grafana-Dashboards lassen sich zu Playlisten zusammenstellen. Diese kann Grafana zeitgesteuert abspielen.

unter **Data (Metric)** umzustellen.

Wechseln Sie in den Reiter für die Abfragen (Icon auf der linken Seite) und legen Sie wie oben beschrieben eine Abfrage für **memory_usage** an. Für die farbliche Hervorhebung wechseln Sie wieder auf den Reiter für die Visualisierung. In der zweiten Spalte aktivieren Sie unter **Coloring** den bunten Hintergrund. Unter **Thresholds** müssen die Schwellwerte für gelben und roten Hintergrund angegeben werden – und zwar in diesem Fall in Byte. Soll der Kasten ab 100 MByte gelb und ab 200 MByte rot werden, geben Sie **100000000,200000000** in das Feld ein. Nach dem Abspeichern können Sie Ihr Werk auf dem Dashboard bestaunen.

37.5. Anzeige im Vollbild

Je nach Anwendungsgebiet werden Sie sich die Dashboards nicht allein ansehen wollen, sondern mit anderen teilen. Grafana ist darauf ausgelegt, im Vollbild auf unterschiedlich großen Bildschirmen zu laufen und passt die Größe der Elemente an den Bildschirm an. Für den Vollbild-Modus gibt es in der oberen Button-Leiste ein TV-Symbol. Mit einem Klick verschwindet die linke Menüleiste, mit einem weiteren sämtliche Steuerelemente. Mit ESC verlassen Sie diesen Modus wieder. Soll ein PC dauerhaft Grafana-Dashboards anzeigen, können Sie den Link kopieren und in den

Autostart legen. Enthält die URL **&kiosk**, startet der Vollbild-Modus. Grafana kann auch Playlisten mit mehreren Dashboards nacheinander anzeigen - gedacht ist das für Bildschirme ohne Maus und Tastatur. Die Funktion **Playlists** erreichen Sie über den Button für Dashboards (Symbol mit vier Quadranten) im linken Menü. Hier können Sie Listen anlegen und festlegen, wie lange eine Seite angezeigt werden soll.

Außerdem ist es möglich, einzelne Diagramme (**Panels**) in einem eigenen Fenster zu öffnen oder als Iframe in eine andere HTML-Seite einzubetten. Klicken Sie dazu neben dem Titel eines Diagramms auf den kleinen Pfeil. Hinter dem Menüpunkt **Share** erfahren Sie den direkten Link und bekommen einen Iframe-Schnipsel zum Herauskopieren. Sollten Sie Grafana in einem Unternehmensnetz einsetzen, ist es jetzt an der Zeit, einen Nutzer anzulegen, der nur lesenden Zugriff auf die Diagramme bekommt. Den Dialog finden Sie hinter dem Zahnrad-Icon im linken Menü.

37.6. Alarme aktivieren

Die Möglichkeiten von Grafana sind damit noch nicht erschöpft. Einen Blick wert ist die Fähigkeit, Alarme beim Über- oder Unterschreiten von Schwellwerten zu versenden. Grafana kann sich zum Beispiel per Mail melden, über einen Messenger-Bot Alarm schlagen oder ein schon vorhandenes Bugtracking-System informieren. Einen solchen Alarmierungsweg richten Sie über das Glocken-Icon im linken Menü ein.

Alle Feinheiten von Grafana lernen Sie am besten beim Experimentieren mit den eigenen Daten – die von cAdvisor gesammelten über die eigene Docker-Umgebung geben schon viel her. Die [Online-Dokumentation von Grafana](#) beantwortet viele Fragen sehr anschaulich.

37.7. Datenvizualisierung: Grafana 9.4 überarbeitet Panels und Navigation

Neben der Query-Sprache TraceQL bringt Grafana ein erneuertes Panel-Design, eine aktualisierte Suche und Navigation sowie neue Authentifizierungsfeatures.

Grafana ist in Version 9.4 erschienen. Das Softwareunternehmen Grafana Labs hat seinem Werkzeug für Datenvizualisierung und Monitoring unter anderem ein überarbeitetes Dashboard-Panel-Design und neue Features für Suche, Navigation und Authentifizierung spendiert. Zudem bietet das neue Release allen Grafana-Cloud-Usern Zugriff auf die neue Query-Sprache TraceQL für Distributed Tracing.

TraceQL erschien Anfang Februar 2023 in [Grafana Tempo 2.0](#), einem Open-Source-Backend für Distributed Tracing. Zu dem Zeitpunkt befand sich Grafana 9.4 noch in einer Beta-Version. Die Query-Sprache basiert auf LogQL – ebenfalls aus dem Hause Grafana Labs – sowie PromQL (Prometheus Query Language) und ist auf das Aufspüren von Traces ausgelegt. In der [Dokumentation](#) finden sich Beispiele für ihren Einsatz.

37.8. Neuerungen für Canvas- und Dashboard-Panels

Das neue Minor Release baut das mit Version 9.3 eingeführte Canvas Panel weiter aus. Damit lassen sich innerhalb des Grafana User Interface benutzerdefinierte Visualisierungen gestalten und Daten-Overlays erzeugen, die mit Standard-Panels nicht möglich wären. Canvas-Visualisierungen sind erweiterbare form-built Panels und erlauben das Platzieren von Elementen innerhalb statischer und dynamischer Layouts. Neu in Version 9.4 sind der Support für Data Links und die Möglichkeit, Pfeile hinzuzufügen. Als Beta-Feature lässt sich das Canvas Panel in Grafana OSS, Grafana Cloud (Free, Pro und Advanced) und Grafana Enterprise verwenden.

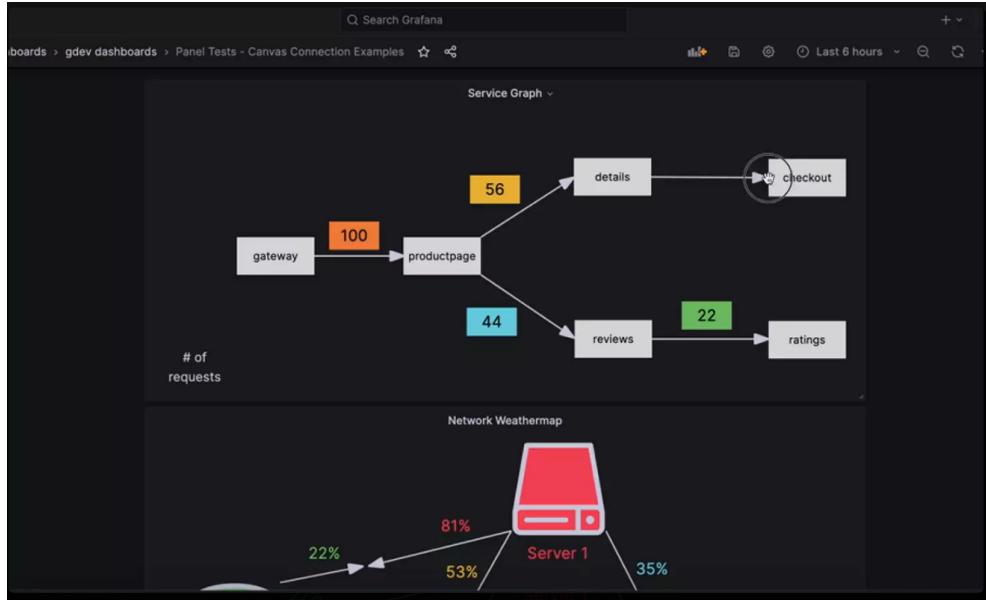


Figure 37.5.: Canvas Panel: In Grafana 9.4 lassen sich Pfeile verwenden, um Elemente zu verbinden. (Bild: Grafana Labs)

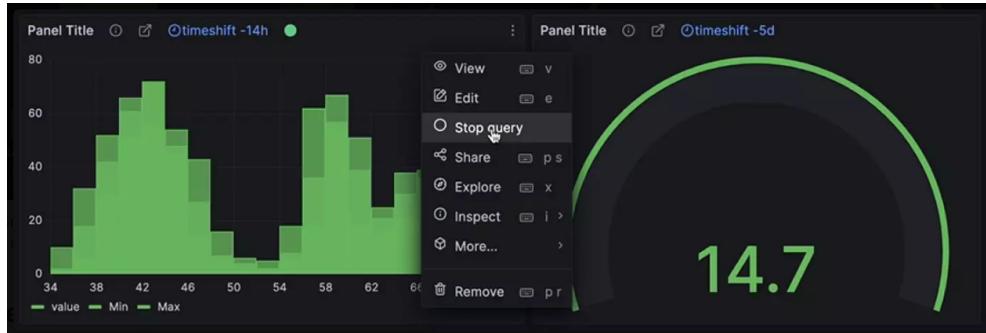


Figure 37.6.: Das Grafana-Dashboard erscheint als Preview-Feature in einem Redesign. (Bild: Grafana Labs)

Den Panels im Grafana Dashboard widmet sich das neue Release ebenfalls. Durch das Hinzufügen und Verschieben von Schlüsselementen sollen diese nun leichter zugänglich und besser verständlich sein. Zudem sind wichtige Komponenten nun im Header jedes Panels zu finden. Dieses Redesign ist als Preview-Feature in Grafana Cloud (Free, Pro, Advanced) sowie per Feature-Toggle (`newPanelChromeUI`) in Grafana OSS und Grafana Enterprise verfügbar.

37.9. Überarbeitete Navigation und Authentifizierung

Eine Keyboard-basierte Navigation und Suche hat in Grafana 9.4 Einzug gehalten. Mit dem Tastenkürzel **Strg/Cmd + K** erscheint die Befehlspalette, die Suche und Zugriff für alle Seiten und Dashboards bietet. Zu den neuen Optionen für die Authentifizierung zählen das Aktivieren eines automatischen Logins für SAML-Anbieter und das Verwenden OAuth-Anbieter-spezifischen Rollen-Mappings.

Alle Details zu diesen und weiteren neuen Features sind in der ["What's New"-Dokumentation](#), im Changelog auf [GitHub](#) und in einem [Blogbeitrag von Grafana Labs](#) zu finden.

38. Further Readings

- <https://www.heise.de/news/Datenvisualisierung-Grafana-9-4-ueberarbeitet-Panels-und-N>
- <https://www.smarthome-tricks.de/grafana/6-1-grafana-anpassung-der-konfiguration>

39. Matplotlib für Einsteiger: So erzeugen Sie ganz einfach Diagramme mit Python

Diagramme müssen nicht viel Aufwand verursachen: Die Bibliothek Matplotlib für Python spart eine Menge Programmierarbeit und erzeugt anschauliche Grafiken. Endlose Zahlenreihen in endlosen Tabellen – es gibt anschaulichere Wege, umfangreiches Datenmaterial einem Publikum näherzubringen. Selbst komplexe Sachverhalte sind in Diagrammen gut zu erfassen. Die Frage ist allerdings, wie man Zahlen möglichst anschaulich und verständlich darstellt.

Sicher: Solche Dinge kann man praktisch mit jeder Programmiersprache umsetzen. Doch dieser Weg ist eher mühsam und setzt fundierte Kenntnisse in Mathematik und Trigonometrie voraus. Ganz zu schweigen von den vielen Fehlern, die sich bei dieser kleinteiligen Vorgehensweise einschleichen können.

Eine effektivere Lösung für Python-Programmierer ist die Bibliothek Matplotlib: Sie dient Entwicklern als Werkzeug, mit dem sie mit geringem Aufwand eine umfassende Palette an Diagrammen erzeugen können. Erweiterungen, beispielsweise für den Export in häufig verwendete Grafikformate, erlauben die Integration der Diagramme in vorhandene Workflows. Die Syntax ist nachvollziehbar und der Katalog an verfügbaren Diagrammen ist umfangreich, zudem steht die Bibliothek unter einer Open-Source-Lizenz. Gute Gründe also, Matplotlib auszuprobieren.

39.1. Matplotlib einrichten

Als erste Aufgabe steht die Einrichtung der Bibliothek auf der To-do-Liste. Um Probleme mit den für die verschiedenen Skripte notwendigen Abhängigkeiten zu vermeiden, empfiehlt sich für Tests die Verwendung einer virtuellen Umgebung. Hier bietet sich der Paket-Manager Conda an. Für ihn spricht unter anderem, dass er in der weitverbreiteten Anaconda-Distribution von Python für wissenschaftliche Anwendungen von Haus aus enthalten ist.

Im ersten Schritt müssen Sie ein neues virtuelles Environment erzeugen. Die angebotenen Voreinstellungen nicken Sie durch Drücken der Y-Taste ab:

```
(base) tamhan@tamhan-thinkpad: $ conda create -name heisematplotlib
Collecting package metadata (current_repodata.json): done
...

```

Nachdem der Virtual-Environment-Generator durchgelaufen ist, aktiviert sich die Umgebung von Conda nicht automatisch. Das holen Sie nach folgendem Schema nach:

```
(base) tamhan@tamhan-thinkpad: $ conda activate heisematplotlib
(heisematplotlib) tamhan@tamhan-thinkpad: $
```

Conda informiert den Nutzer über ein vor den Prompt gestelltes Präfix darüber, welches virtuelle Environment gerade aktiviert ist. Der String (base) würde also für das Base-Environment stehen. Nach der Aktivierung ändert sich der String zu (heisematplotlib), was über die Aktivierung der Virtual Environment informiert.

Im nächsten Schritt müssen Sie den in Conda integrierten Paket-Manager nutzen, um die Virtual Environment mit der Bibliothek auszustatten:

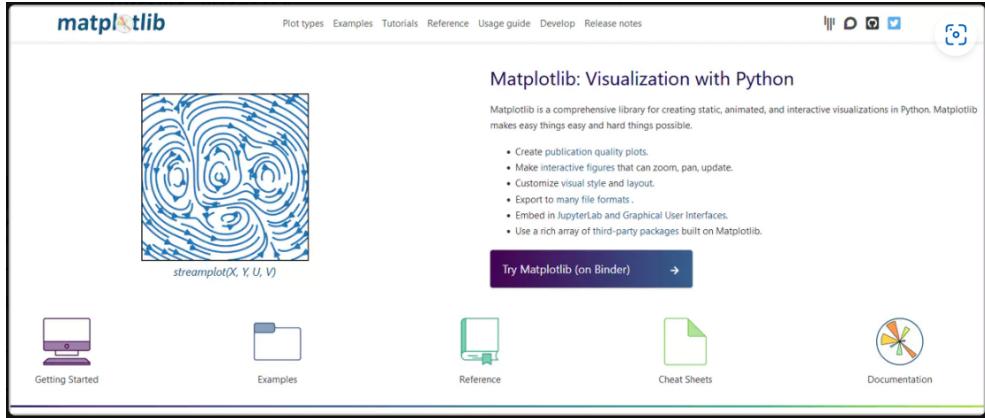


Figure 39.1.: Die Bibliothek Matplotlib unterstützt Programmierer bei der Visualisierung von Daten.

```
(heisematplotlib) tamhan@tamhan-thinkpad: $ conda install matplotlib
...
Executing transaction: done
```

Sobald Sie dieses Kommandos ausführen, startet ein Download von gut 50 MByte Daten. Insbesondere die Neusynchronisation des Virtual Environments kann etwas Zeit in Anspruch nehmen. Schließlich sehen Sie, dass Ihre Workstation um das Paket matplotlib-3.5.1 reicher geworden ist.

Im nächsten Schritt erzeugen Sie ein Arbeitsverzeichnis und öffnen die Arbeitsdatei worker.py in Visual Studio Code:

```
(heisematplotlib) tamhan@tamhan-thinkpad: $ mkdir chartspace
(heisematplotlib) tamhan@tamhan-thinkpad: $ cd chartspace/
(heisematplotlib) tamhan@tamhan-thinkpad: /chartspace$ touch worker.py
(heisematplotlib) tamhan@tamhan-thinkpad: /chartspace$ code
(heisematplotlib) tamhan@tamhan-thinkpad: /chartspace$ code worker.py
```

Der zweistufige Aufruf von code und code worker.py stellt sicher, dass das direkte Öffnen der Datei die Tabs in Visual Studie, die aus vorigen Sessions erhalten sind, nicht überschreibt.

39.2. Liniendiagrammen erzeugen

Nachdem Sie die Bibliothek auf den Rechner geladen haben, müssen Sie anschließend eine Arbeitsdatei erzeugen. Matplotlib kommt normalerweise fast immer im Zusammenspiel mit NumPy zum Einsatz. Die Bibliothek stellt eine Gruppe von Datenstrukturen zur Verfügung, in denen die von der Matplotlib später zu verarbeitenden Informationen unterkommen können:

```
import matplotlib.pyplot as plt
import numpy as np
```

Für eine erste Übung erzeugen Sie jetzt ein Liniendiagramm. Matplotlib stammt aus dem mathematischen Bereich, weshalb die Datenwolke prinzipiell aus zwei Feldern mit x und y aufzubauen ist:

```
x = np.array([4, 2, 2, 4])
y = x*2
```

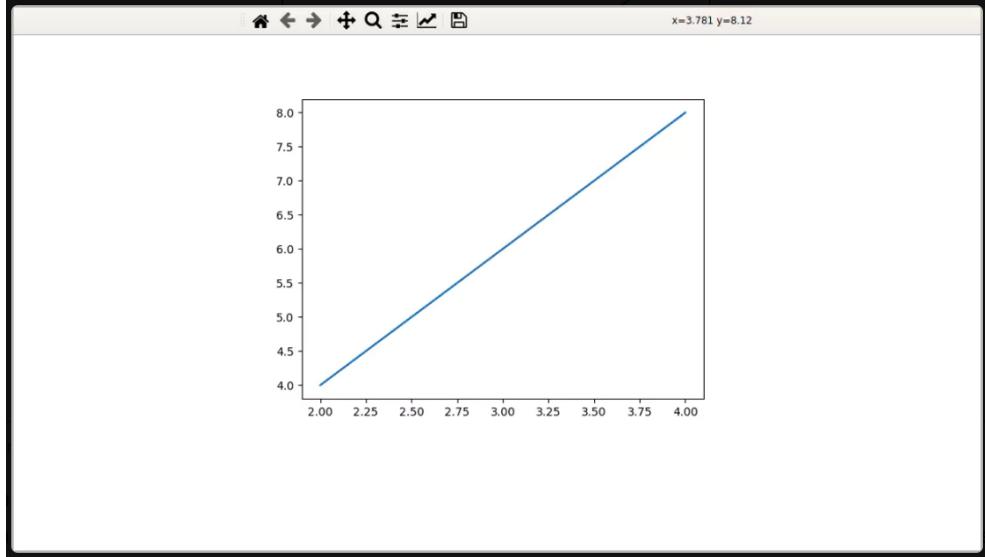


Figure 39.2.: So präsentiert sich das erste Experiment mit Matplotlib auf dem Bildschirm.

Dank der Nutzung eines NumPy-Arrays können Sie die Y-Koordinaten auf eine bequeme Art und Weise erzeugen. Die Multiplikation eines NumPy-Arrays mit einem Integer führt zur Erzeugung eines zweiten Arrays. Im nächsten Schritt sorgen Sie für die eigentliche Anzeige des Diagramms:

```
plt.plot(x, y)  
plt.show()
```

Interessant ist, dass die Erzeugung eines Matplotlib-Diagramms in einem zweistufigen Prozess erfolgt: Der Aufruf der Methode `plot` sorgt zunächst dafür, dass der Speicher das Diagramm aufnimmt; `show` sorgt dann dafür, dass das Diagramm-Anzeigefenster auf dem Bildschirm erscheint.

Beachten Sie, dass der Diagrammanzeiger von Haus aus diverse Komfortfunktionen mitbringt. Benutzer können das Chart beispielsweise skalieren, vergrößern oder über das Disketten-Symbol in verschiedenen Formaten exportieren.

Die Zweiteilung in Aufrufe von `plot` und `show` ist dabei keine willkürliche Gängelung des Bibliotheksnutzers. In der Praxis ist es möglich, eine Gruppe von Liniendiagrammen gleichzeitig zu rendern:

```
plt.plot(x, y)  
  
x1 = [2, 4, 6, 8]  
y1 = [1, 2, 3, 4]  
plt.plot(x1, y1, '.')  
  
plt.show()
```

Der erste Aufruf von `plot` sorgt dabei dafür, dass das aus den Feldern `x` und `y` bestehende Diagramm in den Speicher der Matplotlib eingepflegt wird. Der darauffolgende Aufruf nutzt stattdessen die Felder `x1` und `y1`. Die Übergabe des Werts `'.'` sorgt für die Auswahl des für die Darstellung der Linie zu verwendenden Symbols. Abschließend rendert der Aufruf von `show` abermals den gesamten Speicher.

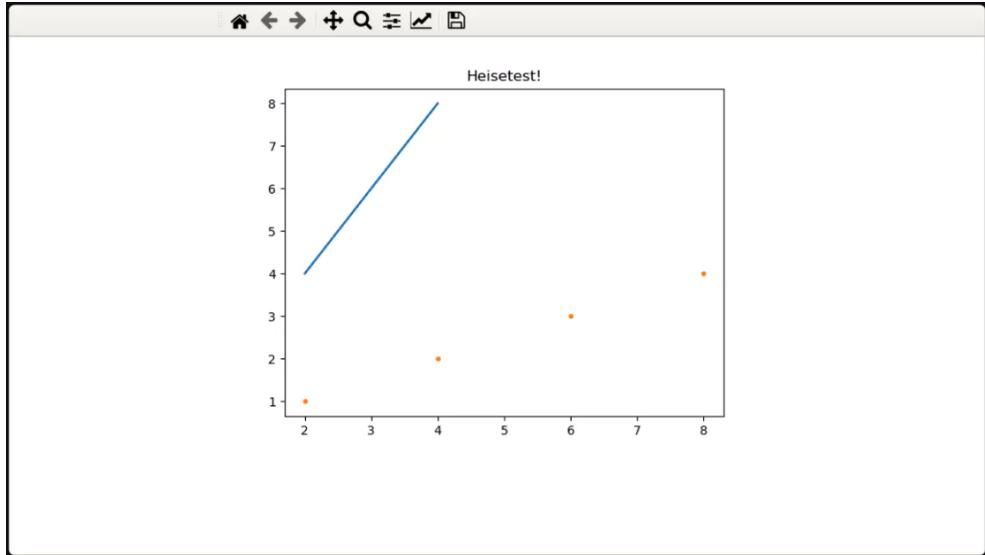


Figure 39.3.: Diagramme lassen sich auf Wunsch auch mit Titel versehen.

39.3. Titel und Legenden rendern

Diagramme profitieren von Beschriftungen, welche die angezeigten Linien und Kurvenformen in einen Kontext stellen. Die einfachste Möglichkeit, um derartige Metadaten einzubinden, ist die Verwendung der Methode `title`. Die platzieren Sie einfach irgendwo vor dem Aufruf der Methode `show`:

```
plt.plot(x1, y1, '.')
plt.title('Heisetest!')
plt.show()
```

Im Allgemeinen hält sich diese Funktion an die Regel Nomen est omen. Die folgende Abbildung zeigt, dass der übergebene String als Titelzeile des erscheinenden Diagrammfensters dient.

Um komplexere Legenden zu erzeugen, benötigt Matplotlib Informationen darüber, was die einzelnen Daten- beziehungsweise Diagramm-Elemente darstellen. Dies lässt sich durch Nutzung des Label-Parameters einpflegen. Dank der Named-Parameter-Funktion der Programmiersprache Python können Sie den Aufruf von Plot mit folgenden Zeilen anpassen:

```
plt.plot(x, y, label="erste Datenreihe")
...
plt.plot(x1, y1, '.', label="erste Datenreihe")
```

Die Anzeige der im Label-Speicherfeld befindlichen Strings erfolgt in Matplotlib nach einer expliziten Aufforderung. Hierzu rufen Sie die Funktion `legend` auf, die eine Gruppe von Parametern übernimmt:

```
plt.legend(facecolor = 'pink', title = 'Legend', loc = 'upper left')
plt.show()
```

Wir nutzen hier das Attribut `facecolor`, um die Hintergrundfarbe der Legende festzulegen. Über das `loc`-Attribut können Sie die Position des Diagramm-Elements grob bestimmen. Weitere Informationen hierzu finden Sie auf matplotlib.org.

Interessant ist in diesem Zusammenhang die Frage, wie Matplotlib auf mehrfaches Aufrufen der Legende-Funktion reagiert. Dies lässt sich durch einen zweiten Aufruf überprüfen, der den Wert `bbox_to_anchor` verändert:

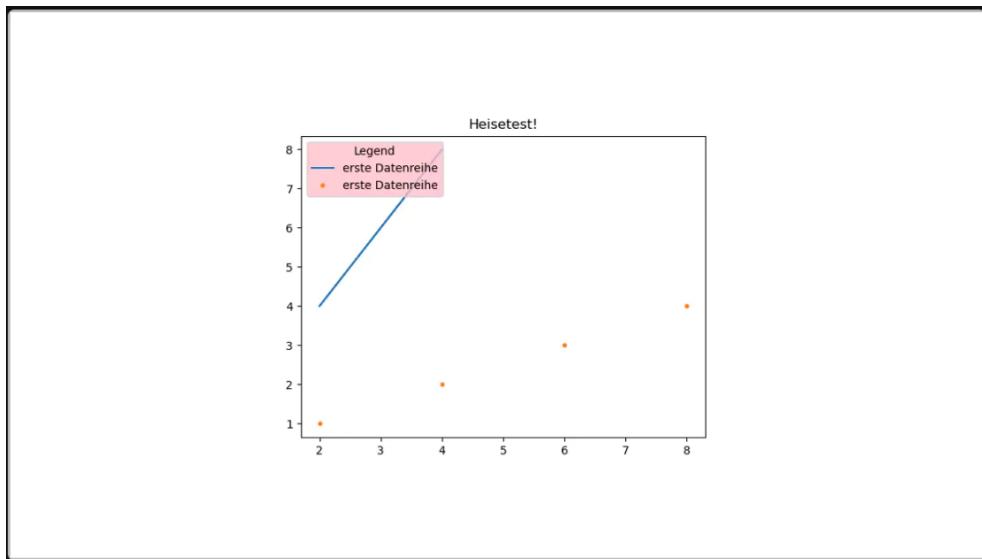


Figure 39.4.: Eine Legende macht das Diagramm leichter verständlich.

```
plt.legend(facecolor = 'pink', title = 'Legend', loc = 'upper left')
plt.legend(bbox_to_anchor= (1.02, 1));
plt.show()
```

Dabei erlaubt `bbox_to_anchor` eine abstrakte Positionierung eines Elements in seinem Container. Wir würden hier dafür sorgen, dass die Legende am rechten Rand erscheint. Wenn Sie diese Version des Programms abermals zur Ausführung freigeben, sehen Sie, dass die Standard-Legende an der vorgegebenen Position erscheint – ihr Hintergrund ist aber weiß. Aus diesem Verhalten können Sie ableiten, dass mehrere Aufrufe von `legend` dazu führen, dass nur der zuletzt übergebene Parametersatz aktiviert wird.

39.4. Tortendiagramme erzeugen

Manuell wäre die Programmierung von Tortendiagrammen eine mühsame Angelegenheit. Die Matplotlib nimmt dem Entwickler an dieser Stelle Arbeit ab. Das folgende Snippet erzeugt ein einfaches Tortendiagramm:

```
import matplotlib.pyplot as plt
sizes=[12,11,13,30]
plt.pie(sizes)
plt.show()
```

Der Aufruf der Funktion `pie` informiert die Bibliothek darüber, dass das angelieferte Daten-Array in ein Tortendiagramm umzuwandeln ist. Matplotlib kümmert sich dabei automatisch um die Summierung der Informationen und andere Housekeeping-Aktivitäten.

Interessant ist daran vor allem, dass die Nutzung von NumPy-Arrays zur Inbetriebnahme der Matplotlib nicht unbedingt erforderlich ist. Es ist genauso legitim, auf ein gewöhnliches Python-Array zurückzugreifen.

Besonders in ihrer dreidimensionalen Variante sind Tortendiagramme bei Grafikern beliebt. Doch bei Ergonomie- und Cognitive-Studies-Professoren sind sie heiß umstritten. Eine lesewerte [Zusammenfassung möglicher Probleme mit diesem Format](#) ist online verfügbar.

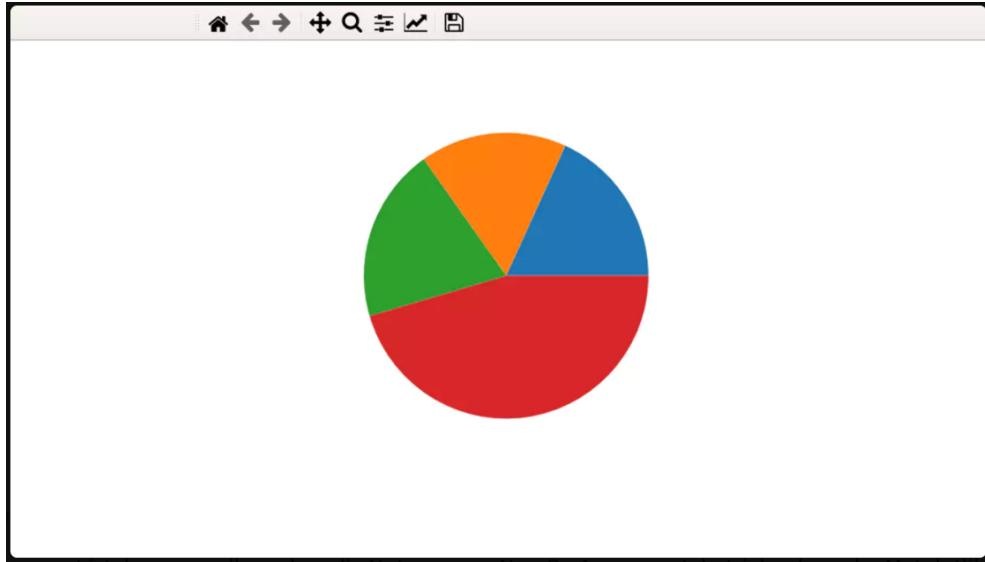


Figure 39.5.: Ein mit Matplotlib automatisch generiertes Tortendiagramm.

Matplotlib beschränkt Entwickler dabei nicht auf das Erzeugen von Tortendiagrammen, die keine Legenden-Informationen anliefern. Die Bibliothek erlaubt auch das Einschreiben verschiedener Hinweise, die dem Nutzer bei der Interpretation der in den Diagrammen befindlichen Informationen helfen.

Der folgende Test nutzt die `legend`-Methode und schreibt dem Tortendiagramm zusätzlich ein Label-Feld ein:

```
labelField = ['A', 'B', 'C', 'D']
sizes=[12,11,13,30]
plt.pie(sizes, labels = labelField)
plt.legend()
```

Zu beachten ist, dass die Anzeige der Tortennamen neben den einzelnen Tortenstücken auch dann erfolgen würde, wenn Sie im Programm die Methode `legend` nicht aufrufen.

39.5. Balkendiagramme rendern

Balkendiagramme sind oftmals der bequemere und übersichtlichere Weg zur Darstellung von größeren Informationsmengen. Matplotlib leistet auch bei diesem Diagramm-Typ wertvolle Hilfe. Für einen ersten Versuch im Bereich der Erzeugung von Balkendiagrammen reicht es aus, wenn Sie unser Codebeispiel anpassen:

```
labelField = ['A', 'B', 'C', 'D']
sizes=[12,11,13,30]
plt.bar(labelField, sizes)
plt.legend()
plt.show()
```

Balken- und Tortendiagramme haben mehr gemeinsam, als man auf den ersten Blick annehmen würde. Wichtig ist hier der Austausch der Generatorfunktion `pie` gegen `bar` und die Umkehrung der Reihenfolge zwischen den Parametern `labelField` und `sizes`. Sonst verhält sich auch diese Version des Programms erwartungsgemäß.

Möchten Sie stattdessen ein horizontal ausgerichtetes Balkendiagramm haben, müssen Sie sich nicht mit Bitmap-Transformationen und anderen Komplikationen herumärgern.

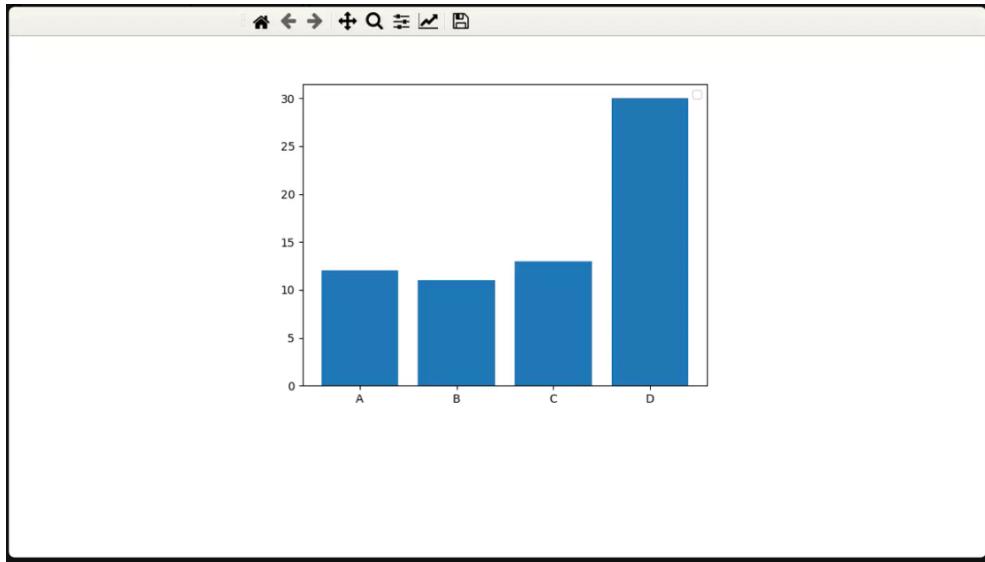


Figure 39.6.: Das gerenderte Balkendiagramm ist einsatzbereit.

Stattdessen reicht es aus, die Generatorfunktion `bar` durch ihre identisch parametrierte Kollegin `barh` zu ersetzen:

```
labelField = ['A', 'B', 'C', 'D']
sizes=[12,11,13,30]
plt.barh(labelField, sizes)
plt.legend()
plt.show()
```

Lohn der Mühen ist dann die Erzeugung des horizontal ausgerichteten Balkendiagramms.

Beachten Sie, dass die hier vorgestellten Methoden mit diversen anderen Komfortfunktionen der Matplotlib vereinbar sind: Im Fall von Balkendiagrammen kann es etwa sehr nützlich sein, im Hintergrund ein Gitter zu platzieren, das die Übersichtlichkeit erhöht. Nutzen Sie dafür einfach die Methode `plt.grid (true)`:

```
labelField = ['A', 'B', 'C', 'D']
sizes=[12,11,13,30]
plt.bar(labelField, sizes)
plt.legend()
plt.grid(True)
plt.show()
```

39.6. Funktionen und Histogramme

Eine letzte Aufgabe soll demonstrieren, wie sich die Kombination aus Matplotlib und NumPy-Berechnungsfunktionen zur Darstellung von Funktionen nutzen lässt. Ziel ist die Umsetzung einer Bildschirm-Ansicht, die an das Schirmbild erinnert, das der durchschnittliche grafische Taschenrechner präsentiert.

Im ersten Schritt benötigen wir dabei Datenfelder, die Sinus- und Kosinus-Werte aufnehmen. Problematisch ist an dieser Sache, dass die in NumPy enthaltenen Funktionen Werte in Radian erwarten. Die Radian-Werte sind allerdings schwieriger zu generieren als eine einfache Folge (0, 1, 2, 3, ... 360). Ein möglicher Lösungswert besteht

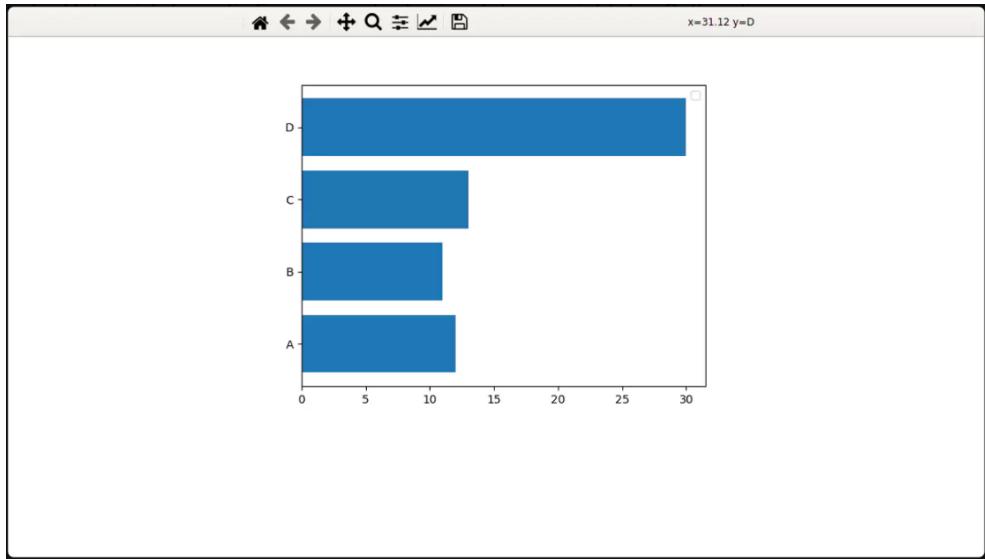


Figure 39.7.: Matplotlib dreht Balkendiagramme auf Wunsch.

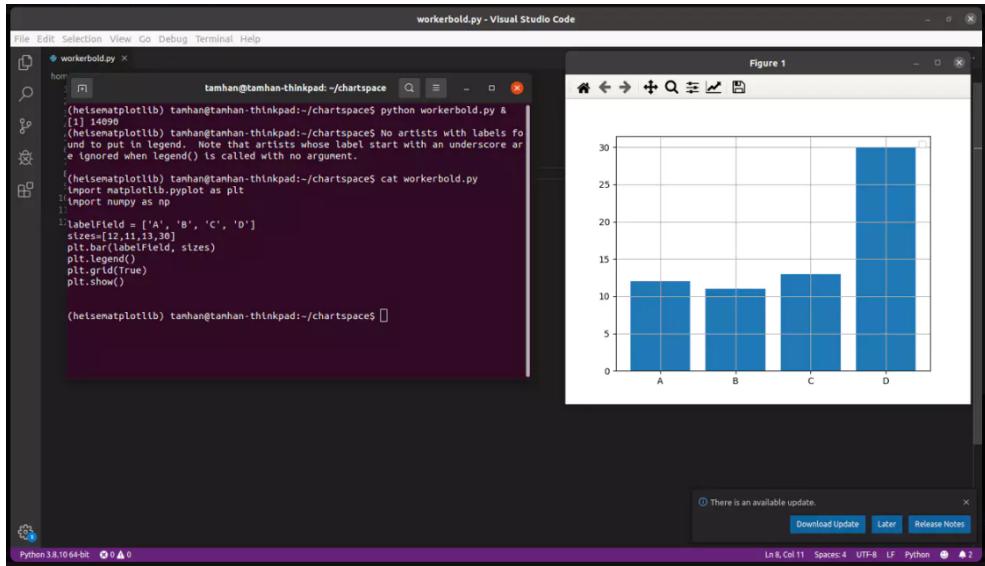


Figure 39.8.: Ein Gitter im Hintergrund des Diagramms verbessert die Übersichtlichkeit deutlich.

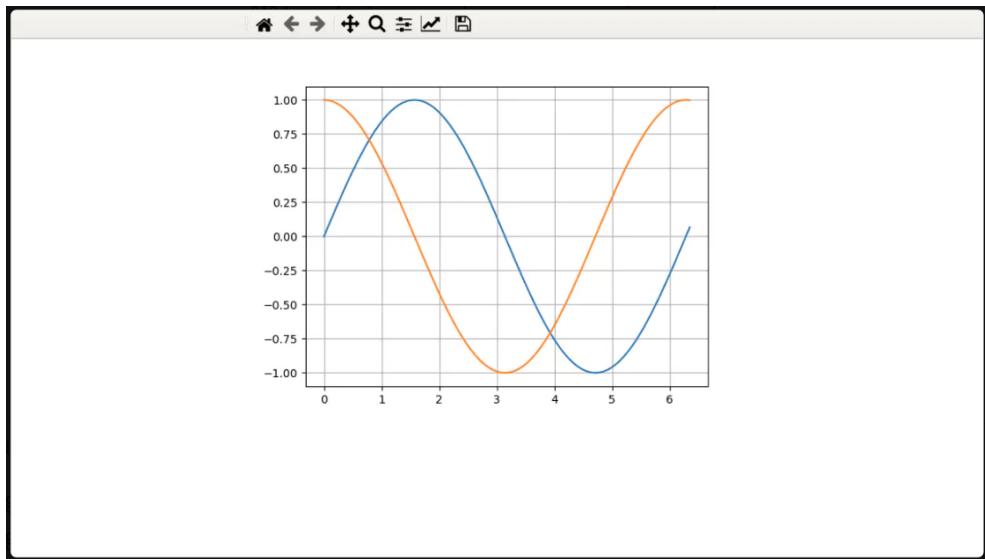


Figure 39.9.: Mit dem Aufruf von `plt.grid(True)` kann der Benutzer die Werte einfacher begreifen.

darin, die Funktionen `np.sin` und `np.cos` gegen ein NumPy-Array beziehungsweise ein NumPy-Range anzuwenden.

NumPy bietet mit Rangemarker-Funktionen nämlich die Möglichkeit an, fortlaufende Zahlenbereiche nach verschiedenen Definitionen automatisch zu generieren:

```
laeufer = np.arange(0,6.4,0.05)
sins = np.sin(laeufer)
cosins = np.cos(laeufer)
```

`arange` erzeugt dabei ein von 0 bis 6.4 laufendes Zahlenfeld, der dritte Parameter legt den zwischen den einzelnen Elementen anzulegenden Abstand fest.

Für einen ersten Versuch nutzen Sie am besten die schon erwähnte Plot-Funktion, um sowohl die Sinus-Schwingung als auch die Kosinus-Schwingung auf den Bildschirm zu bringen:

```
plt.grid(True)
plt.plot(laeufer, sins)
plt.plot(laeufer, cosins)
plt.show()
```

Interessant ist im vorliegenden Code, dass wir die Methode `plt.grid` mit `True` als Parameter aufrufen. Das Ergebnis ist, dass Matplotlib das angezeigte Diagrammfenster mit einem karierten Hintergrund ausstattet, der die Korrelation der Datenpunkte erleichtert.

Diagramme zu beschriften und Achsen zu beschreiben, ist immer wichtig. Das gilt umso mehr für Diagramme, die sehr viele Kurven aufweisen. Dies erledigen Sie durch die Methoden `plt.xlabel` und `plt.ylabel`, welche die als Achsbeschriftungen vorgesehenen Strings aufnehmen:

```
plt.grid(True)
plt.plot(laeufer, sins)
plt.plot(laeufer, cosins)
plt.xlabel("Time")
plt.ylabel("Value")
```

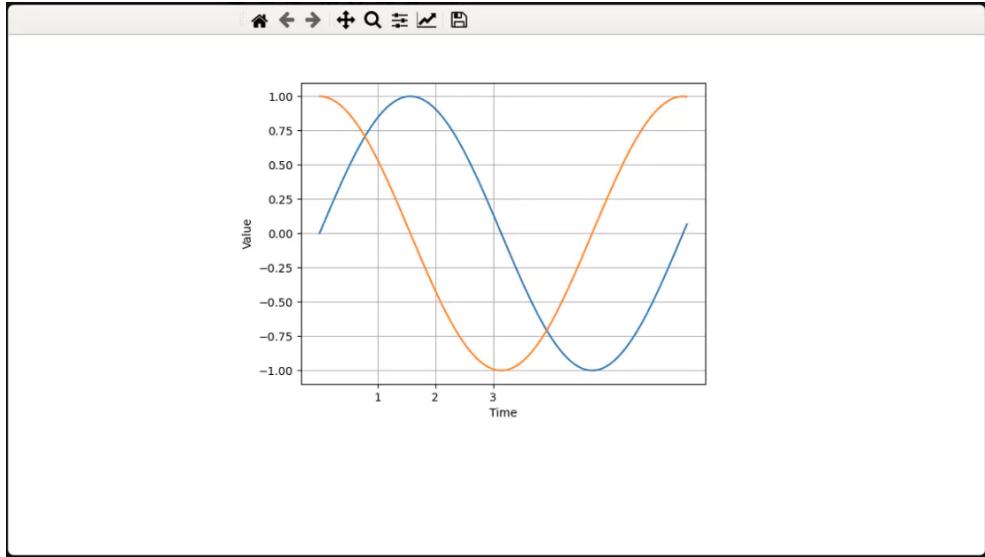


Figure 39.10.: Wer die `xticks`-Methode benutzen möchte, muss mitdenken.

```
plt.show()
```

Nach der Programmausführung zeigt sich, dass diese Beschreibungen ein Diagramm verständlicher machen.

Matplotlib ist in der Lage, die Abstände von Grids und von Achsmarkierungen bis zu einem gewissen Grad selbst zu errechnen. In der Praxis gibt es allerdings immer wieder Situationen, in denen man sich eine formalere Vorgehensweise wünscht. Die Methode `xticks` erlaubt in diesem Fall die Übergabe einer Gruppe von Werten, die dann als Basis-Stellen für die im Diagramm eingeblendeten Hilfselemente herangezogen werden:

```
plt.xlabel("Time")
plt.xticks([1,2,3])
plt.ylabel("Value")
plt.show()
```

Die Matplotlib hält sich nach dem Aufruf der Methode komplett aus der Berechnung heraus. Übergeben Sie wie hier beispielsweise nur die Werte von 1 bis 3, überstreicht der gesamte Wert allerdings 0 bis 6.4. Deshalb erhalten Sie freie Bereiche:

39.7. Histogramme

Die in Matplotlib enthaltenen Statistikfunktionen erlauben das Berechnen von Histogrammen. Dazu müssen Sie das normalerweise von Hand durchzuführende Binning nicht durchführen. Die Bibliothek kümmert sich darum, wenn sie für die Diagramm-Erzeugung die Methode `hist` verwenden:

```
plt.grid(True)
plt.hist(sins)
plt.show()
```

Wer `hist()` nur mit einem einzelnen Datenfeld aufruft, animiert die Matplotlib dazu, alle notwendigen Werte selbst festzulegen. Möchten Sie Ihr Diagramm stattdessen mit einer bestimmten Menge von Klassen (Bins) rendern, bietet sich die Nutzung eines named parameters an. Die folgende Version des Codes würde beispielsweise 30 Bins anlegen:

```
plt.grid(True)
plt.hist(sins, bins = 30)
plt.show()
```

Insbesondere im Bereich Binning bietet die Matplotlib diverse Komfortfunktionen an, die auf [matplotlib.org](#) im Detail beschrieben sind. Es ist zum Beispiel erlaubt, ein Array mit den Grenzen der Bins anzuliefern. Dies erlaubt wiederum das Verändern der Erfassungsbreite von Teilen des Histogramms.

39.8. Literatur zu Matplotlib

Wie fast alle anderen Projekte im Python-Ökosystem ist auch die Matplotlib exzellent dokumentiert: Es stehen [spezifische Informationen zu einzelnen Teilen des Frameworks](#) zur Verfügung. Dort können Sie beispielsweise mehr darüber erfahren, welche Rolle die einzelnen benannten Parameter der Funktionen übernehmen.

Wegen der immensen Verbreitung der Bibliothek hat sich bei der praktischen Erzeugung von Matplotlib-Diagrammen allerdings eine etwas andere Vorgehensweise herauskristallisiert. Sowohl auf [matplotlib.org](#) als auch auf [python-graph-gallery.com](#) finden Sie eine Gruppe von Diagramm-Beispielen, die neben einem Rendering auch mit dem für ihre Erzeugung vorgesehenen Quellcode präsentiert werden.

39.9. Fazit

Wer in Python Diagramme rendern möchte und die Matplotlib nicht nutzt, macht unnötige Fleißarbeit. Die im Allgemeinen leicht verständliche Syntax und der immense Umfang der in der Bibliothek verfügbaren Diagrammtypen tut dann sein übriges, um das Produkt zu einem wahren Schweizer Messer der Visualisierung zu machen.

Literaturverzeichnis

- [AIT20] L. Artificial Intelligence Techniques, ed. *OpenNN*. 2020. URL: <https://www.opennn.net/>.
- [AR+16] R. Al-Rfou et al. “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688 (May 2016). [pdf], [Link]. URL: <http://arxiv.org/abs/1605.02688>.
- [Aba+16] M. Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [Alo+18] M. Z. Alom et al. *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. Tech. rep. [pdf]. 2018. arXiv: 1803.01164 [cs.CV].
- [And35] E. Anderson. “The irises of the Gaspé Peninsula”. In: *Bulletin of the American Iris Society* 59 (1935), pp. 2–5.
- [BS19] P. Buxmann and H. Schmidt. *Künstliche Intelligenz*. Springer, 2019.
- [Bal19] Balakreshnan. *Inferencing Model - Jetson Nano Without Docker container*. Eingesehen am 20.04.2020 [online]. 2019. URL: <https://github.com/balakreshnan/WorkplaceSafety/blob/master/InferencinginJetsonNano.md#inferencing-model---jetson-nano-without-docker-container>.
- [Cho18] F. Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.
- [Cho+19] A. Chowdhery et al. *Visual Wake Words Dataset*. [pdf], [github]. 2019. doi: 10.48550/ARXIV.1906.05721. arXiv: 1906.05721 [cs.CV]. URL: <https://github.com/Mxbonn/visualwakewords>.
- [Cor20] N. Corporation, ed. *CUDA*. 2020. URL: <https://developer.nvidia.com/cuda-zone>.
- [Den+09] J. Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.
- [Den12] Deng, L. “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]”. In: *IEEE Signal Processing Magazine* 29.6 (2012). [pdf], Link, pp. 141–142.
- [Ecl20] Eclipse, ed. *Deep Learning for Java*. 2020. URL: <https://deeplearning4j.org/>.
- [Fac20] Facebook, ed. *PyTorch*. 2020. URL: <https://pytorch.org/>.
- [Fis] R. A. Fisher. “The use of Multiple Measurements in Taxonomic Problems”. In: () .
- [Fis36a] R. A. Fisher. *Fisher's Iris Dataset*. [pdf]. 1936. URL: <https://archive.ics.uci.edu/ml/datasets/iris>.
- [Fis36b] R. A. Fisher. *Fisher's Iris Dataset*. [pdf]. 1936. URL: <https://archive.ics.uci.edu/ml/datasets/iris>.

-
- [Fou20a] P. S. Foundation, ed. *glob — Unix style pathname pattern expansion*. Eingesehen am 15.12.2019 [online]. 2020. URL: <https://docs.python.org/3/library/glob.html>.
- [Fou20b] P. S. Foundation, ed. *numpy*. Eingesehen am 15.12.2019 [online]. 2020. URL: <https://numpy.org/>.
- [Fou20c] P. S. Foundation, ed. *python-pickle-module-save-objects-serialization*. Eingesehen am 17.12.2019 [online]. 2020. URL: <https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>.
- [Fou20d] P. S. Foundation, ed. *random*. Eingesehen am 16.12.2019 [online]. 2020. URL: <https://docs.python.org/3/library/random.html>.
- [Fou21a] P. S. Foundation, ed. *The Python Package Index*. 2021. URL: <https://pypi.org>.
- [Fou21b] P. S. Foundation, ed. *pycocotools*. 2021. URL: <https://pypi.org/project/pycocotools>.
- [GMG16] P. Gysel, M. Motamedi, and S. Ghiasi. “Hardware-oriented approximation of convolutional neural networks”. In: *arXiv preprint arXiv:1604.03168* (2016).
- [Goo19] Google, ed. *TensorFlow*. 2019. URL: <https://www.tensorflow.org/>.
- [Goo20] Google, ed. *TensorFlow Lite Guide*. 2020. URL: \url{https://www.tensorflow.org/lite/guide}.
- [HLT09] W. T. Ho, H. W. Lim, and Y. H. Tay. “Two-Stage License Plate Detection Using Gentle Adaboost and SIFT-SVM”. In: *2009 First Asian Conference on Intelligent Information and Database Systems*. IEEE, 2009, pp. 109–114. ISBN: 978-0-7695-3580-7. DOI: [10.1109/ACIIDS.2009.25](https://doi.org/10.1109/ACIIDS.2009.25).
- [Han+17] D. Hansen et al. “Real-Time Barcode Detection and Classification using Deep Learning”. In: (Jan. 2017), pp. 321–327. DOI: [10.5220/0006508203210327](https://doi.org/10.5220/0006508203210327).
- [IK17] H. Ide and T. Kurita. “Improvement of learning for CNN with ReLU activation by sparse regularization”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017. DOI: [10.1109/ijcnn.2017.7966185](https://doi.org/10.1109/ijcnn.2017.7966185).
- [Int19] Intel Corporation, ed. *OpenVino - System Requirements*. 2019. URL: <https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit/system-requirements.html>.
- [JES20] Y. Jia and E. Evan Shelhamer. *Caffe*. Ed. by B. A. I. Research. 2020. URL: <http://caffe.berkeleyvision.org/>.
- [KH09] A. Krizhevsky and G. Hinton. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.
- [KSH12] A. Krizhevsky, I. Sutskever, and G. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: [10.1145/3065386](https://doi.org/10.1145/3065386). URL: http://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf.
- [KSH17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *CIFAR-10 and CIFAR-100 datasets*. 2017. URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [Kag] *Iris Flower Dataset*. 2018. URL: <https://www.kaggle.com/arshid/iris-flower-dataset>.

- [LCB13] Y. LeCun, C. Cortes, and C. Burges. *MNIST handwritten digit database*. 2013. URL: <http://yann.lecun.com/exdb/mnist/>.
- [Lin+14] T. Lin et al. “Microsoft COCO: Common Objects in Context”. In: *The Computing Research Repository (CoRR)* abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
- [Lin+21] T.-Y. Lin et al. *COCO - Common Objects in Context*. 2021. URL: <https://cocodataset.org/>.
- [MXN20] A. S. F. A. MXNet, ed. *MXNet*. 2020. URL: <https://mxnet.apache.org/>.
- [Mic20] Microsoft, ed. *The Microsoft Cognitive Toolkit*. 2020. URL: <https://docs.microsoft.com/en-us/cognitive-toolkit/>.
- [NVI15] NVIDIA Corporation, ed. *NVIDIA TensorRT*. 2015. URL: \url{https://developer.nvidia.com/tensorrt}.
- [NVI20] NVIDIA Corporation, ed. *NVIDIA CUDA Toolkit 11.0.161*. [pdf]. 2020.
- [Nie15] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: <http://neuralnetworksanddeeplearning.com/chap1.html>.
- [Ope18] Open eVision, ed. *EasyOCR – Reading Texts*. 2018. URL: https://documentation.euresys.com/Products/Open_eVision/Open_eVision_2_5/en-us/Content/03_Using_Open_eVision/D1_EasyOCR_-_Reading_Texts/EasyOCR_-_Reading_Texts.htm.
- [PN20] I. Preferred Networks, ed. *Chainer*. 2020. URL: <https://chainer.org/>.
- [Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [RF16] J. Redmon and A. Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242.
- [Raj20] A. Raj. “AlexNet CNN Architecture on Tensorflow (beginner)”. In: (2020). Ed. by Kaggle. URL: <https://www.kaggle.com/vortexkol/alexnet-cnn-architecture-on-tensorflow-beginner>.
- [Red20] J. Redmon. *MNIST in CSV*. 2020. URL: <https://pjreddie.com/projects/mnist-in-csv/>.
- [SIG20] K. SIG, ed. *Keras*. 2020. URL: <https://keras.io/>.
- [SP06] V. Sahni and C. Patvardhan. “Iris Data Classification Using Quantum Neural Networks”. In: *AIP Conference Proceedings* 864.1 (2006), pp. 219–227. DOI: 10.1063/1.2400893. eprint: <https://aip.scitation.org/doi/pdf/10.1063/1.2400893>. URL: <https://aip.scitation.org/doi/abs/10.1063/1.2400893>.
- [SSS19] F. Siddique, S. Sakib, and M. A. B. Siddique. “Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers”. In: *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*. [pdf]. 2019, pp. 541–546.
- [SW16] M. Schutten and M. Wiering. “An Analysis on Better Testing than Training Performances on the Iris Dataset”. In: *Belgian Dutch Artificial Intelligence Conference*. Link, Link. Sept. 2016.
- [SW52] B. Silver and N. J. Woodland. *Classifying Apparatus and Method*. US 2612994. 1952. URL: <https://worldwide.espacenet.com/patent/search/family/022402610/publication/US2612994A?q=pn%3DUS2612994>.

-
- [Sha+20] V. Shankar et al. “Evaluating Machine Accuracy on ImageNet”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. [pdf]. PMLR - Proceedings of Machine Learning Research, 2020, pp. 8634–8644. URL: <http://proceedings.mlr.press/v119/shankar20c.html>.
 - [Sou20] F. O. Source, ed. *Caffe2*. 2020. URL: <https://caffe2.ai/>.
 - [TS19] S. Tan and A. Sidhu. “CUDA implementation”. In: Apr. 2019, pp. 63–67. ISBN: 978-3-030-17273-2. DOI: [10.1007/978-3-030-15585-8_5](https://doi.org/10.1007/978-3-030-15585-8_5).
 - [Tea20a] O. Team, ed. *OpenCV*. 2020. URL: <https://opencv.org/>.
 - [Tea20b] O. Team, ed. *OpenVino*. 2020. URL: <https://www.openvino.org/>.
 - [Wik20] Wikipedia, ed. *OpenCV*. Eingeschen am 16.12.2019 [online]. 2020. URL: <https://de.wikipedia.org/wiki/OpenCV>.
 - [YK17] I. Yun and J. Kim. “VIision-based 1D Barcode Localization Method for Scale and Rotation Invariant”. In: (Nov. 2017), p. 6. DOI: [10.1109/TENCON.2017.8228227](https://doi.org/10.1109/TENCON.2017.8228227).
 - [fas20] fast.ai, ed. *FastAI*. 2020. URL: <https://www.fast.ai/>.

Index

- AlexNet, 54
- ALPDR, 19, 93
- Amazon, 27
- API, 19, 26, 32
 - Application Programming Interface
 - see* API, 19, 26
- Automatic License Plate Detection and Recognition *see* ALPDR, 19, 93
- Basemap, 29
- Bibliotheiken, 25
- Bokeh, 29
- Caffee, 27
- Caffee2, 27
- Canadian Institute For Advanced Research
 - see* CIFAR
 - see* Datensatz, 13, 19, 44
- Canadian Institute for Advanced Research, 44
- Central Processing Unit
 - see* CPU, 19, 30
- Chainer, 27
- CNTK, 27
- COCO, 19, 32, 49–54
- Common Objects in Context
 - see* COCO, 19, 32
- Compute Unified Device Architecture
 - see* CUDA, 19, 25
- CPU, 19, 30
- CUDA, 19, 25, 26, 30
- cv2, 31
- Datensatz
 - CIFAR, 13, 19, 44, 45
 - Fisher's Iris Data Set, 45–49
 - ImageNet, 54
 - MNIST, 42, 43
 - NIST Special Database 3, 19, 43
 - NIST Special Database 1, 19, 43
 - TensorFlow, 43
- Deeplearning4J, 27
- DeepScene, 54
- EasyOCR, 93
- Eli5, 30
- Example, 67, 69, 77
 - Description, 67, 69, 77
- Installation, 67, 70, 77
- Introduction, 67, 69, 77
- FastAI, 27
- Frameworks, 25
- glob, 31
- GoogleNet, 54
- GPU, 19, 27, 30
- Graphics Processing Unit
 - see* GPU, 19, 27
- ImageNET, 54
- ImageNet
 - see* Datensatz 54
- Inception, 54
- IPython, 28
- jetson-inference, 54
- Jupyter-Notebook, 25
- Keras, 26
- LightGBM, 29
- math, 31
- matplotlib, 28
- Microsoft Cognitive Toolkit, 27
- MLpy, 30
- MXNet, 27
- NetworkX, 29
- NIST Special Database 1
 - see* Datensatz, 19, 43
- NIST Special Database 3
 - see* Datensatz, 19, 43
- NLTK, 29
- NumPy, 28
- OpenCV, 30
- OpenNN, 31
- OpenVINO, 30
- os, 31
- Pandas, 28
- Pattern, 29
- pickle, 32
- PIL, 31
- PyPI, 32

Python Software Foundation, 32
PyTorch, 26

random, 31
Residual Neural Network
 see ResNet, 19, 54
ResNet, 19, 54
ResNets, 54
RLE, 19, 53
Run-Length Encoding, 19, 53

scikit-learn, 28
SciPy, 28
Scrapy, 29
Seaborn, 29
SSD-Mobilenet-V2, 54
Statsmodels, 30

TensorFlow, 25
TensorFlow Lite, 26
TensorRT, 26
Theano, 27

UCI Machine Learning Repository, 45

VGG, 54
Video Processing Unit
 see VPU, 19, 30
VPU, 19, 30

YOLO v5, 99